Dorian Nedelcu

Tihomir Latinovic

Programming engineering application in Python, wxPython and Streamlit

MONOGRAPHY

Presa Universitară Clujeană

Dorian Nedelcu | Tihomir Latinovic

•

Programming Engineering Application in Python, Wxpython and Streamlit

MONOGRAPHY

About the Authors

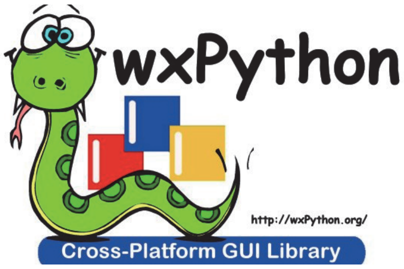
Prof. dr. DORIAN NEDELCU (https://dorian-nedelcu.streamlit.app) is an international member of the Bosnian and Herzegovinian American Academy of Arts and Sciences (BHAAAS) and a former full professor at Babes-Bolyai University Cluj-Napoca (BBCN), Faculty of Engineering, Resita, Romania, currently retired according to the regulations and teaching policy of BBCN, Romania. Is a graduate of the University "Politehnica" Timișoara, Faculty of Mechanics, class of 1983, obtained a doctorate in technical sciences in 1996, under the supervision of academician Ioan Anton. Scientific research and design in the field of hydraulic machines (1983 - 1999), Programmer Analyst (1999-2001), Lecturer (2001), Associate Professor (2002), Full Professor (2008) at "Eftimie Murgu" University of Resita (UEMR). Starting with 2020 UEMR merged with BBCN. PhD supervisor in the field of "Mechanical Engineering" at UEMR & BBCN; 10 doctoral theses supervised, of which 10 students were awarded the title of doctor. Advanced user of programming languages and libraries: PYTHON, VIZUAL BASIC, VIZUAL FoxPro, Microsoft Excel, Microsoft Access, FORTRAN, BASIC, dBASE, PASCAL, FoxPRO, HTML, Basic4Android (B4A), Matplotlib, Streamlit, Three.js. Extensive experience in using the following 3D CAD design & Reverse Engineering software: SolidWorks (Part and Assembly design, Simulation, Flow Simulation, Motion), Autodesk Inventor, AutoCAD, Microstation, Geomagic Design X, Agisoft Metashape, 3DF ZEPHYR. Visiting professor at the master's program of Industrial Engineering offered by the Postgraduate Studies and Research Section from the Professional Interdisciplinary Unit of Engineering and Management and Social Sciences (UPIICSA) from the National Polytechnic Institute (IPN), Mexico (2020-2022). Published books: 9 books, of which 6 as unique author. 148 x articles published (24 x ISI articles indexed in the Web of Science, 11 x Proceedings Paper articles indexed in the Web of Science, 113 x articles included in Conferences, Journals & Databases).

Prof. dr. TIHOMIR LATINOVIC (https://tihomirlatinovic.wordpress.com/) is a corresponding member of the Bosnian and Herzegovinian American Academy of Arts and Sciences (BHAAAS), a professor at University VITEZ in Travnik, Bosnia and Herzegovina, and a former professor at the University of Banja Luka's Faculty of Mechanical Engineering, Republic of Srpska, Bosnia and Herzegovina. Author of nine books on teaching and science. More than 154 papers have been published in domestic and foreign journals, as well as in domestic and international conferences. Romanian international expert, ARACIS member, and Bosnia and Herzegovina expert in licensing faculties and universities. Accreditation like external evaluator for Romanian agency ARACIS, member of HEA in Commission of the University, Doctoral Scholls Study Programs for five Universities from Romania. Visiting professor at over 20 universities in Albania, Austria, Germany, Serbia, Romania, Hungary, Poland, Bulgaria, Croatia, North Macedonia, and Slovakia as part of the CEEPUS and Erasmus initiatives. Coordinator for the inaugural B&H CEEPUS network. Established more than twenty Erasmus KA1+ projects. Organizer of four international conference of applied science (ICAS) conferences ICAS 2018, 2019, 2022 and ICAS 2024. Participated in two domestic and several international Tempus initiatives, including CEEPUS and HERD. Editor for four international publications. Reviewer for the Technical Gazette magazine on the SCI list. Active knowledge of English, having completed four degrees in the subject. Co-Mentor of the commission for the defense of 5 doctoral dissertations, Mentoring of 5 MSc, Member of the commission for 5 MSc, Mentor on 25 bachelor papers, Chairman of the commission for the 42 final work of the first cycle, Member of the commission for 25 candidates for the final work of the first cycle. Advanced knowledge of programming languages and libraries includes MS OFFICE, Word, Excel, PowerPoint, Access, programming languages SQL, C ++, Java, JavaScript, Html, SWISH, PHP, Visual Basic, Pyton, Basic4Android (B4A), Matplotlib, and Streamlit. He has 304 citations with h-index 9 and i10-index 9 with Research Interest Score 960.9.

Dorian Nedelcu | Tihomir Latinovic

Programming Engineering Application in Python, Wxpython and Streamlit

MONOGRAPHY



PRESA UNIVERSITARĂ CLUJEANĂ
2025

Referenți științifici:
Prof. dr. Manuela Pănoiu
Prof. dr. Branko Latinovic

ISBN 978-606-37-2500-5

Universitatea Babeș-Bolyai
Presa Universitară Clujeană
Director: Codruța Săcelean
Str. Hasdeu nr. 51
400371 Cluj-Napoca, România
Tel.: (+40)-264-597.401
E-mail: editura@ubbcluj.ro
http://www.editura.ubbcluj.ro/
https://libraria.ubbcluj.ro/

Content

Preface

The book "*Programming Engineering Application in Python, wxPython and Streamlit*" by Dorian Nedelcu and Tihomir Latinovic is designed for both beginners and those with some programming experience. It covers various applications of Python, including numerical integration, databases, image operations~~,~~ and more. The content is accessible, making it suitable for students, teachers~~,~~ and self-learners. Each application comes with the complete code as well as detailed explanations and visual results, which can be a great help in the learning process.

The context of the book authored by Dorian Nedelcu and Tihomir Latinovic, focuses on programming engineering applications using Python, wxPython~~,~~ and Streamlit. It provides comprehensive explanations and code examples for a variety of applications, including numerical integration, databases, image operations~~,~~ and other relevant scenarios that utilize different versions of Python and its libraries. Complete code examples: each topic is accompanied by complete code snippets and detailed explanations, facilitating hands-on learning.

This comprehensive approach ensures that readers gain practical and theoretical knowledge across various programming aspects, making it suitable for both beginners and those looking to enhance their skills in Python programming. In summary, while a basic understanding of Python is beneficial, the book is structured to support learners at various levels, making it accessible to a broad audience interested in programming with Python and its libraries.

Familiarity with Python programming language is essential, as the book focuses on applications built using Python~~,~~ and libraries including: wxPython, Matplotlib, Streamlit, Open 3D, VTK and OpenCV. Ability to install Python and associated libraries. A desire to explore programming applications in fields such as: numerical integration, image and PDF operations, optical character recognition, database management (SQLite, MySQL). The book is suitable for students, teachers~~,~~ and anyone interested in programming, making it accessible for educational purposes.

Readers are encouraged to engage with the material actively by trying out the complete code and detailed explanations provided for various applications. These applications are designed to cater to various educational and practical scenarios, making them useful for students, teachers, and professionals alike. Each application is supported by complete code examples and detailed explanations of the underlying functions and classes used in their development. These libraries and frameworks facilitate a wide range of engineering applications, from numerical integration and data visualization to machine learning and image processing, making Python a versatile choice for developers in various fields.

The book effectively combines theoretical concepts with practical coding examples, making it a valuable resource for anyone looking to deepen their understanding of Python in engineering contexts and artificial intelligence applications.

Based on its structure, comprehensive coverage of topics, practical examples, and accessibility, this book is highly recommended for anyone looking to learn Python effectively. It is particularly useful for beginners due to its clear explanations and focus on application-based learning, making it a valuable resource in the field of programming.

The book encompasses a wide range of programming concepts primarily focused on Python, wxPython, and Streamlit.

Being aware that any work can be improved, we are waiting for suggestions or comments on the book at *dorian.nedelcu@ubbcluj.ro* and *tihomirlatinovic@live.com*.

Authors

1. Introduction to Python and its role in Artificial Intelligence

This Python programming language is known for its simplicity. The syntax provides concise and readable code, making it easier for non-programmers to understand. This is why Python is the best choice for those new to programming. Also, its syntax enables fast testing (without implementation) of complex algorithms, which also saves development time.

Python's flexibility is perhaps its greatest strength. Developers can choose to use it as an object-oriented or scripting language. Its flexibility reduces the possibility of error, and for even better results, Python can be combined with other programming languages.

The programming language was created in the early nineties by the Dutch programmer Guido van Rossum. It is intended as a language for learning programming and a replacement for languages such as BASIC and Pascal. The programming language was named after the British comedy series Monty Python's Flying Circus. The IDLE development environment (**I**ntegrated **D**evelopment and **L**earning **E**nvironment) is named after group member Eric Idle.

The first version of the Python language appeared in 1994. Two different versions of the language are currently in use. Version 2, which first appeared in 2000, is still in use. The new version appeared in 2008 and influenced further parallel development of the previous version. In practice, the latest editions of versions 2 and 3 are used in parallel. Programs developed for a specific version are not compatible, but there are tools for automatic conversion of programs from version 2 to version 3.

Major releases of individual versions of the Python language are: Python 1 / 1994, Python 2 / 2000, Python 3 / 2008, Python 3.4 / 2014, Python 3.5 / 2015, Python 3.6 / 2016, Python 3.7 / 2018, Python 3.8 / 2019.

The Python language is relatively simple, but it contains extensive and high-quality programming libraries. According to the annual analysis of the World Association of Electrical and Electronics Engineers, the Institute of Electrical and Electronics Engineers (Institute of Electrical and Electronics Engineers, IEEE) Python is currently the most popular programming language. The most important reasons for its popularity are:

❶ simple syntax and very readable code.
❷ great software capabilities and wide use.
❸ used by large companies and organizations, such as YouTube, Google, Yahoo and NASA.
❹ it's free and has good customer support.

The program code can be executed in two ways:

➊ Interactive - where the code is executed at the time it is written.
➋ Scripted - where the code is stored within a text file that is executed when the script is run.

In engineering practice, the script execution method is most often used. Well-known applications written in Python are:

❶ Blender3D - 3D animation, gaming, and video editing software.
❷ Dropbox - web hosting site for files.
❸ Firefox - one of the most popular web browsers.
❹ Google App Engine - many components of the most famous search engine and web application development platform.
❺ Instagram - social network for sharing photos.
❻ Pinterest - social network for sharing photos.
❼ Quora - a famous site and network for asking questions.
❽ Yahoo Maps - certain software services for maps.
❾ YouTube - the most famous site for sharing and displaying videos, etc.

Artificial intelligence (AI) automates human intelligence based on the way the human brain processes information. This opened the door to a new world of possibilities for developers, but also new horizons for us, the users. AI not only enables the Spotify or Netflix platforms to recommend artists, songs, and movies, but is also a great help to companies in developing customer services, improving business and employee productivity, and especially for processing large amounts of data, where most of it is used.

In the first place is certainly many libraries specially created for machine learning (NumPy, SciPy, Pandas, scikit-learn, NLTK, TensorFlow), as well as a large selection of frameworks that greatly facilitate development and at the same time save time that programmers can devote to solving specific project problems.

Many developers choose Python for its many advantages when it comes to machine learning and artificial intelligence. Such projects are already playing a significant role in various economic sectors such as fintech, transportation, healthcare, travel, and the entertainment industry. In addition to the development of artificial intelligence, Python is used by Google, Facebook, as well as NASA and CERN, and its application is expanding day by day.

Basic reasons why Python is a good choice for AI:

➊ Python is a high-level programming language; this means you don't have to worry about memory allocation, pointers, or machine code in general.

➋ Python is in the first place as the favorite language of programmers, because it is readable and simple, and it is also great for fast application development.

➌ The main advantage is memory efficiency because Python can handle large databases.

➍ Python is a general-purpose language; it can be used to create desktop applications, database applications, mobile applications, and games. Its network programming characteristics should also be mentioned. Python is a great prototyping tool.

2. Facial recognition system

2.1 Introduction

Facial recognition technology has become so widespread that today it is hard to imagine life without it. Most of us use this very technology every day without even realizing it. Facial recognition technology has become one of the main features of all smartphones. It made it possible to unlock the phone quickly and easily without remembering and typing in codes, it is enough just to place the phone screen in front of the face and the unlocking process is complete. Also, its high level of security has allowed smartphones to become virtual wallets. In addition to everyday applications, facial recognition technology is also used in security systems in various places. It is used by the police, airports, army, banks, and other security services to check whether a certain person is in the suspect database. Also, the application of this technology is growing day by day, and in the future, it is expected that this technology will be "infiltrated" in almost all smart devices.

2.2 Face recognition system

Facial recognition is a biometric software algorithm that can identify or verify a person by analyzing the contours of a person's face. In principle, facial recognition systems differ, but their concept is the same. The software scans photos of people's face and converts them into a suitable geometric record, then compares that record with a database of other photos looking for a match, **Figure 2.1**. Face images can be obtained by taking photos or recording videos. In addition to face recognition, this category also includes recognition using a fingerprint, iris, voice, ... [2.1].

The situations in which these systems are used are usually related to security, so the users of such systems are usually airports, military bases, police, banks, and other organizations that need a high level of protection. Such systems are used around the world to determine whether a person from a photograph is in the database of convicted or suspected persons. A big obstacle for these systems is that they must work with photos that are of very poor quality, because not every photo can be taken in ideal conditions. But today's systems and technologies have managed to overcome this, and poor-quality photos can be improved in just a few steps [2.2], [2.3].

In facial recognition technology, facial features are divided into a main subset and an elaborated subset. The main subset consists of the corners of the eyebrows, eyes, tip of the nose, mouth, and chin. The elaborated subset represents only the augmented main subset, **Figure 2.2** [2.1].

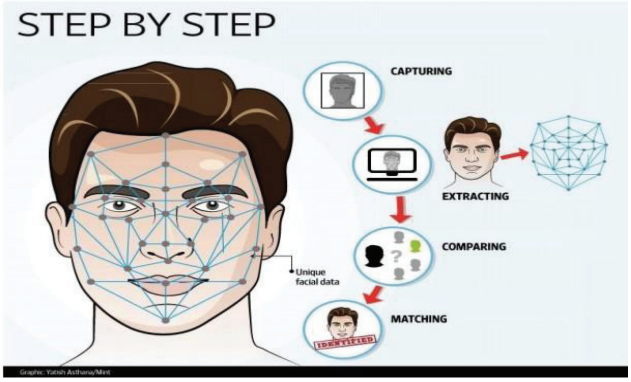


Figure 2.1 Concept of face recognition system [2.1]

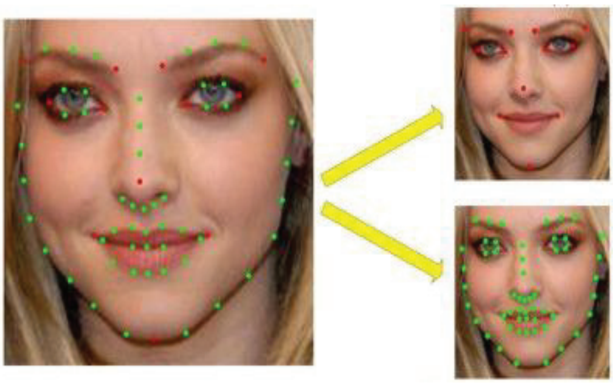


Figure 2.2 Division of facial features [2.1]

Facial recognition cameras are installed in public places in almost all major cities and special attention is paid to making these cameras as invisible to people as possible. Also, one "trick" is used to get the best possible quality photos: facial recognition cameras are made to resemble ordinary cameras as much as possible. This is why people instinctively look at the camera, not knowing that it is a facial recognition camera. In this way, good quality photographs are obtained, which greatly facilitate the process of face recognition [2.2].

There are many facial recognition technologies available today, but some of the most important are:

❶ The traditional 2D face recognition method.

❷ 3D face recognition method.
❸ Face detection using thermography.

2.2.1 Traditional 2D face recognition method

The traditional 2D face recognition method, or "Frontal recognition", is an approach in which, during image processing, certain features of the face of the frontal 2D image are found and extracted, which are then compared with the corresponding features of the face in a known database. The goal of the image processing phase is to reduce the amount of data to process and speed up the subsequent parameter search procedure.

There are two approaches to the 2D facial recognition method, namely:

❶ geometric approach,
❷ photometric approach.

The advantages of 2D face recognition are that the images can be acquired remotely (no physical contact with the scanner) and no observing/imaging subject is required. The biggest disadvantages are the variations that occur when taking pictures (pose, lighting, facial expression, aging, hairstyle, glasses, makeup, beard...). Another major drawback is that the system can be fooled very easily by planting a printed 2D photo [2.4].

2.2.1.1 *Geometric 2D face recognition*

The concept of geometric 2D face recognition is based on modeling the human face based on characteristic details (eyes, mouth, nose, eyebrows...) and the arrangement of these details on the face. Geometric relations (areas, angles, and distances) are generated between the corresponding facial details used as points of interest. Each face contains about 80 key details, **Figure 2.3** [2.4].

The first time this technology was tested was in January 2000 at the American football game, "Super Bowl XXXV", which was held in Tampa Bay, Florida. The visitors of this event, by purchasing their ticket, not only attended the match, but also tested the new facial recognition technology on them for the first time. "FaceIt", which makes it possible to use a video camera to detect the face of each visitor and compare it with the database of criminals. The "FaceIt" system uses 2D geometric recognition and 36 cameras were used to operate the system, placed in different parts of the city. The system was used by the police, and it made it possible to keep a closer eye on general activities and to recognize people with a file. When this testing was done, the visitors were not aware of it at all and did not notice anything strange. The system cost about 30,000 US dollars, **Figure 2.4** [2.5].

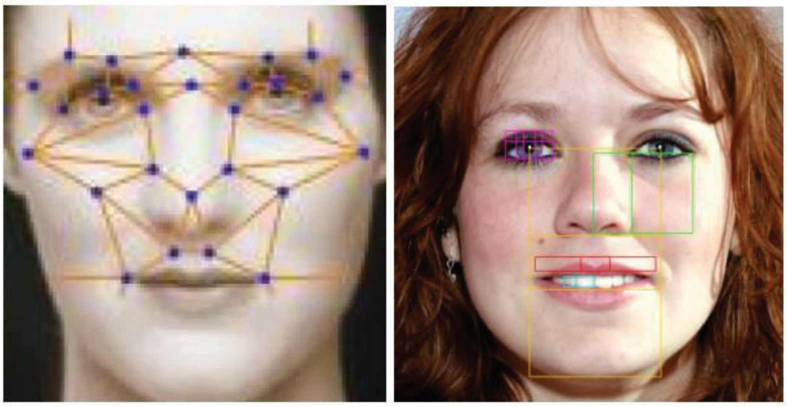


Figure 2.3 Geometric 2D face recognition [2.4]

Figure 2.4 Using the software FaceIt along with the monitoring system [2.5]

The principle of operation of such a system consists of several steps, and they are:

❶ **Discovery**- When the system is connected to a video surveillance system, the recognition software searches for faces in the field of view video cameras. If there is a face in the camera's field of view, it is detected in a fraction of a second. If you work with low resolutions, the system has an algorithm that also overcomes this, **Figure 2.5**.

❷ **Alignment**- Once a face is detected, the system determines the size and position of the head. The face should be turned at least 35 degrees towards the camera for the system to register it.

❸ **Normalization**- The head image is scaled and rotated so that it can be registered and mapped to the appropriate size and position. Normalization is performed regardless of the position of the head and the distance from the camera. Brightness does not affect the normalization process, **Figure 2.6**.

❹ **Representation**- The system translates face data into a unique code. This coding procedure allows easier comparison of newly acquired face data with stored face data, **Figure 2.7**.

❺ **Matching**- The newly acquired face data is compared with the stored data and (ideally) is linked to at least one saved face representation in the database, **Figure 2.7** [2.4], [2.5].

This biometric technology can also be used to protect files on computer. By mounting the web cameras to the computer and installing facial recognition software, the face can become the password used to enter the computer. IBM was the first to incorporate this technology into screensaver for their own portable computers, [2.5].

Such systems have the ability to recognize between 15-60 million images per minute depending on the quality of the images and the user population, **Figure 2.8** [2.5].

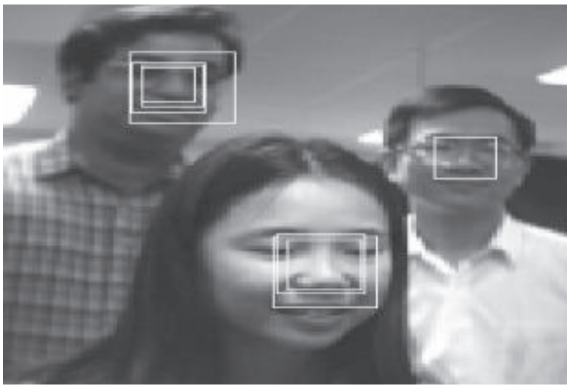


Figure 2.5 Phase of face detection [2.4]

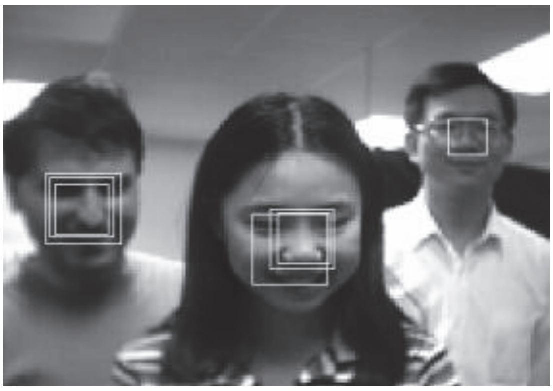


Figure 2.6 Normalization phase [2.4]

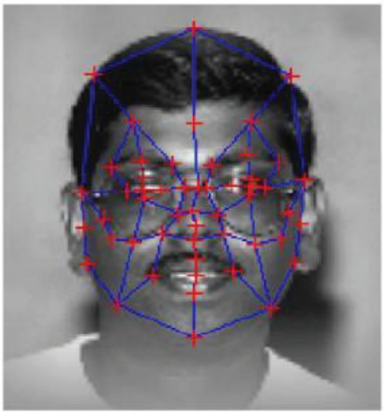


Figure 2.7 Phase of face detection [2.4]

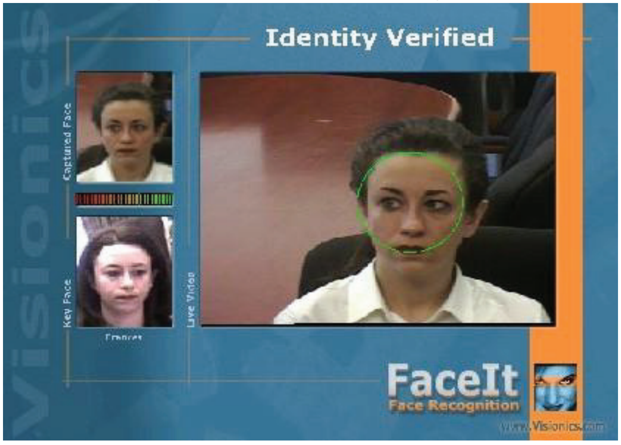


Figure 2.8 Matching phase[2.5]

2.2.1.2 *Photometric 2D face recognition*

Photometric systems are used for recognition based on the overall appearance of the face. Every human face can be represented as a combination of standard building blocks - eigenfaces. The creators of this method are Matthew Turk and Alex Pentland, and they started from the fact that for face recognition it is not enough to find characteristic points and measure their mutual distance [2.6].

What Turk and Pentland have come up with is a new way of encoding and decoding facial photographs, while highlighting facial features. They showed that the characteristics of the face do not have to correspond to the characteristic points of the face (eyes, mouth, nose, ...). Turk and Pentland did the following: they extracted information from a photo of a face and somehow captured the variations of faces in a collection of photos, while not paying attention to the characteristic points of the face (mouth, nose, eyebrows, ...). They then coded that information and later used it to compare it with face photos from the database [2.7].

Mathematically speaking, each photo is treated as a point (or vector) in a high-dimensional space. Looking at the matrices of a set of face photos, the eigenvectors of the specified matrix are searched for. A set of eigenvectors can be viewed as a set of features of a face obtained from photo variations. Inherent vectors are called eigenfaces. The eigenfaces are gray in color and are shown in Figure 2.9 [2.7].

For this system to work, the face photos from the database and the face image of the query must be the same size. It is necessary to perform normalization to match the positions of the eyes and mouth in the images. And finally, based on the numerical value carried by each characteristic face, the system compares the query image with the characteristic faces from the database, and based on the comparison, produces a matching result. Figure 2.10 shows the result of the system operation. The loaded image is made up of 55% of eigenface 1 and 10% of eigenface 2, etc. [2.6].

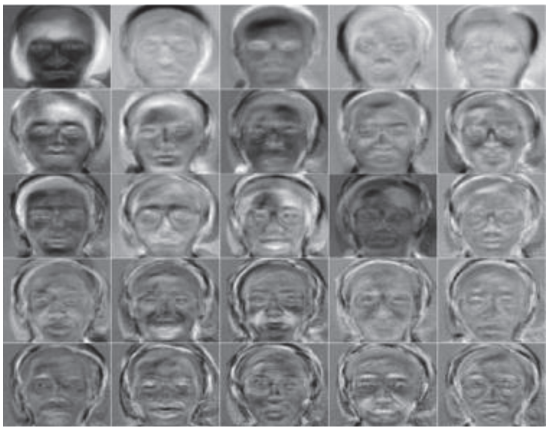


Figure 2.9 Eigenfaces [2.4]

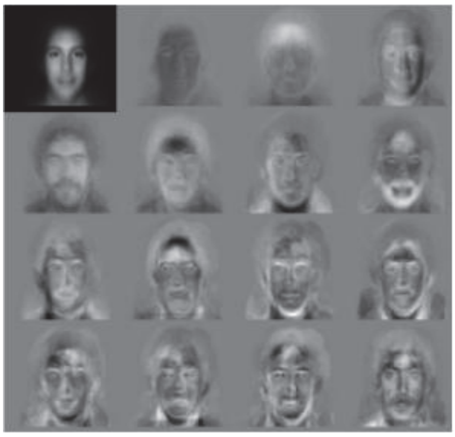


Figure 2.10 Face recognition using characteristic faces [2.6]

2.2.1.3 *3D face recognition method*

As the 3D imaging process is getting cheaper and faster, the use of 3D sensors thought to have the potential for more accurate facial recognition than 2D recognition. The advantage of using 3D technology is that depth information does not depend on pose and lighting, so the representation of the object does not change with these parameters, which makes the complete system more robust. The 3D face recognition method introduces additional information into the system compared to 2D systems (the shape/structure of the face in space entered). The system produces a three-dimensional structure of the face, that is, the image of the face gains depth (shape of the skull, eye part, jaw, ...). With a 2D system, the several tens of characteristic points are recorded on the image, while with a 3D system, several thousand characteristic points are recorded (that is, a network of face images in space is obtained). The disadvantage of 3D systems is the price and the fact that some systems require a person to remain in front of the camera for several seconds [2.8]

3D face model or better known as 3D Morphable Model is the most famous method used for 3D face recognition. The method is designed to work with 2D color photos and create 3D models from them, Figure 2.11 [2.1]. 3D Morphable Model method uses laser scans of faces as a set of photographs. The laser scans were made as if the person was wearing a cap over their hair and only their head was scanned. From the scans, the face is separated from the rest of the head and after that the processing continues, Figure 2.12 [2.1].

The models of the 3D Morphable Model method are processed in such a way that x, y ,z coordinates and the red, blue and green components of each pixel are taken for each pixel of the scanned face. With the help of this data, the face is identified using the method of characteristic faces, that is, the matching of all faces from the database with the scanned face is sought. As with the method of eigenfaces, the percentage of agreement is obtained as a result [2.1].

Aging is a big problem for these systems, but with today's algorithms it can be solved. For people aged 2-18 years, variations in the shape of the face due to aging are mainly the product of "displacement" of facial features. In older people, variations due to aging are mainly the product of changes in facial texture. On Figure 2.13 can be seen how the algorithm predicts changes due to aging [2.4].

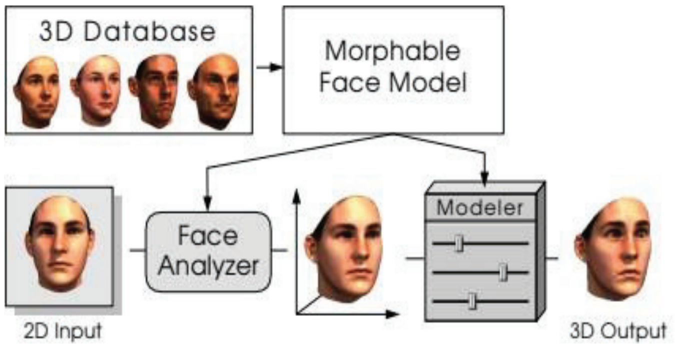


Figure 2.11 3D Morphable Model [2.9]

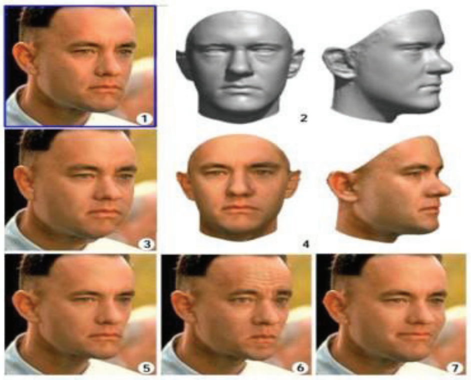


Figure 2.12 Laser skews [2.1]

2.2.2 Face detection using thermography

Thermographic cameras detect light in the infrared spectrum and produce images called thermograms. The amount of radiation emitted by the body increases with temperature and therefore thermography allows the visualization of temperature variations. Seen with a thermal imaging camera, warm bodies are clearly distinguished from the colder environment, which clearly enables the detection of a person's face [2.4].

Advantages of detection using thermography:

❶ Detection of objects in motion and in real time.
❷ Object detection in inaccessible environments.
❸ Possibility of object detection in the dark.
❹ Object detection does not depend on age, facial expressions, and aesthetic modifications.

In addition to all these advantages, the biggest disadvantage of thermography is the price of the cameras. On **Figure 2.13** you can see what a thermograph of a person's face looks like.

Another important feature of thermography is that it makes it very easy to distinguish twins. The reason is that each twin has a different set of blood vessels on the face that can be very easily seen using a thermographic camera **Figure 2.14** [2.4].

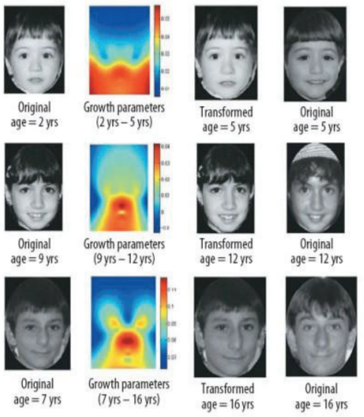


Figure 2.13 Prediction of facial changes due to aging [2.4]

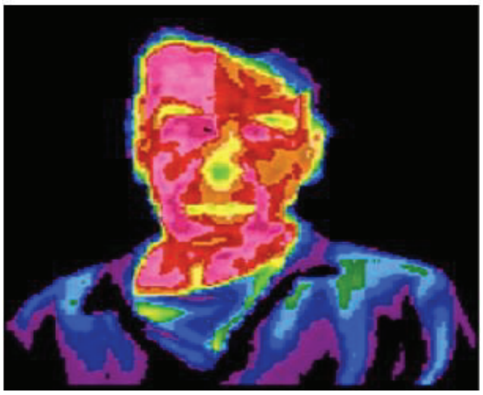


Figure 2.14 Thermograph of the face [2.4]

2.3 Conclusion

Facial recognition technology has attracted the attention of both ordinary people and many experts and researchers. Today, this technology represents the fastest biometric technology aimed at identifying the human face. This is precisely the main characteristic recognized by various experts. It enabled them to search several tens of thousands of people in just one minute, to identify suspicious persons and to prevent possible incidents. Because of its great ease of use, every day many people provide their facial biometric information. Most people are not even aware of the dangers that can arise due to the misuse of biometric information, but this is the price that must be paid to live in the modern world.

3. Python and associated library installation

This book is intended for those who want to assimilate the Python language and get in touch with its multiple libraries: wxPython, Matplotlib, Streamlit, Open CV, etc. It can also be used for educational purposes, as the presented applications are accompanied by the complete code and detailed explanations.

Most of every script was created in two versions: one based on Python & wxPython library and the other based on Python & Streamlit library. Specific elements of the two languages are presented in comparison, as well as the conceptual differences resulting from their use in the creation of the script. All applications were generated and tested on Windows 11 operating system.

Python & wxPython applications are saved with **pyw** extension and were designed using Python version 2.7.10 and libraries from **table 3.1**. **Listing 3.1** display the code for printing library versions for Python version 2.7.10.

Python & Streamlit applications are saved with **py** extension and were designed using Python version 3.12.2 and libraries from **table 3.2**. **Listing 3.2** display the code for printing library versions for Python version 3.12.2.

Table 3.1

Python version 2.7 and libraries
Python version: 2.7.10
wxPython version: 2.8.12.1
Matplotlib version: 1.4.3
sqlite version: 3.7.12.1
Pandas version: 0.16.2
Open CV version: 2.4.11
Numpy version: 1.9.2
SciPy version: 0.15.1
Pillow (PIL) version: 1.7
MySQL server: 8.0.39.0

Table 3.2

Python version 3.12 and libraries
Python version: 3.12.2
wxPython version: 4.2.1
Matplotlib version: 3.8.2
sqlite version: 3.43.1
Pandas version: 2.2.0
Open CV version: 4.10.0
Numpy version: 1.26.4
SciPy version: 1.14.0
Pillow (PIL) version: 10.2.0
Streamlit version: 1.37.0

Listing 3.1

```
import sys
import wx
import matplotlib
import pandas as pd
import numpy as np
import cv2
import scipy
from PIL import Image
from pysqlite2 import dbapi2 as sqlite
print "Python version:", sys.version
print "wxPython version:", wx.__version__
print "Matplotlib version:", matplotlib.__version__
print "sqlite version:", sqlite.sqlite_version
print "Pandas version:", pd.__version__
print "Open CV version:", cv2.__version__
print "Numpy version:", np.__version__
print "SciPy version:", scipy.version.version
print("Pillow (PIL) version:", Image.VERSION)
```

Listing 3.2

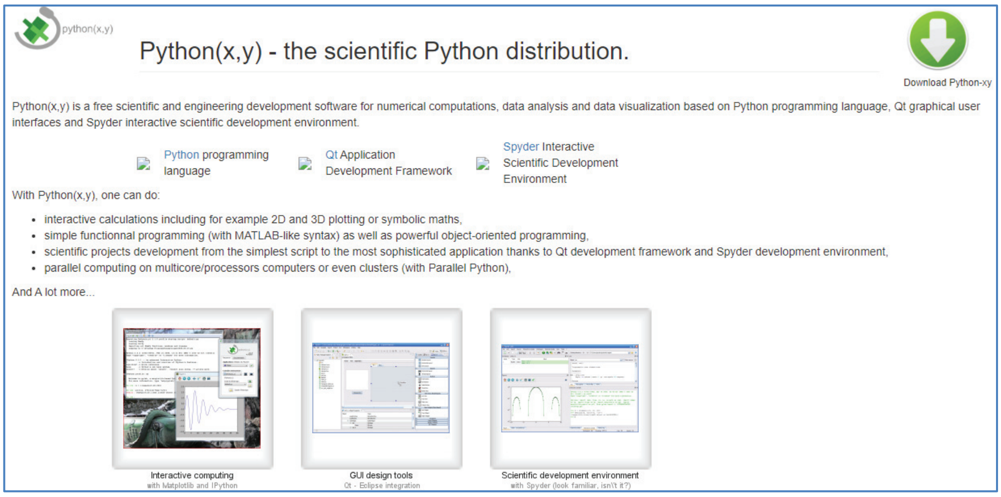
```
import sys
import wx
import matplotlib
import pandas as pd
import numpy as np
import cv2
import scipy
import PIL
import sqlite3
import streamlit
print ( "Python version:", sys.version )
print ( "wxPython version:", wx.__version__ )
print ( "Matplotlib version:", matplotlib.__version__ )
print ( "sqlite version:", sqlite3.sqlite_version )
print ( "Pandas version:", pd.__version__ )
print ( "Open CV version:", cv2.__version__ )
print ( "Numpy version:", np.__version__ )
print ( "SciPy version:", scipy.version.version )
print ( "Pillow (PIL) version:", PIL. __version__  )
print ( "Streamlit version:", streamlit.__version__ )
```

The Python version 2.7.10 environment and associated libraries do not come prepackaged with Windows. It is not a simple task to install the desired version along with the required dependencies, so we recommend the most efficient way using the **Python(x,y)** package from [**3.1**]. Python (x,y) is a free scientific and engineering development environment for numerical calculations/methods, data analysis and data visualisation based on the Python programming language. The downloaded file is **Python (x, y) - 2.7.10.0.exe** with size ~808 Mb and can be accessed via the green **Download Python-xy** button placed at the upper right of the window, **Figure 3.1**. After downloading, the file **Python (x, y) -2.7.10.0.exe** must be launched to install the Python environment. We recommend that you do not change the Python directory proposed by the installer and leave the one proposed by the installer. Installation will take a few minutes and will require minimal user intervention by following the steps on the screen. Select the **Recommended** option to install the package version. At the end of the installation, the **Python(x,y)** entry is placed in the Windows Start menu. **Figures 3.2** show the interface of this library, launched from the Python(x,y) icon after installation:

❶ **Shortcuts** – to acces Spyder (a free and open source scientific environment written in Python, for Python), Qt Designer (the Qt tool for designing and building graphical user interfaces (GUIs) with Qt Widgets), gnuplot (a graphical library);

❷ **Documentation** – Qt Help, Python language manuals, NumPy reference guide, 2D and 3D plotting, 3D visualization, etc.;

❸ **About** – show all installes libraries for Python 2.7.



Python(x,y) - the scientific Python distribution.

Python(x,y) is a free scientific and engineering development software for numerical computations, data analysis and data visualization based on Python programming language, Qt graphical user interfaces and Spyder interactive scientific development environment.

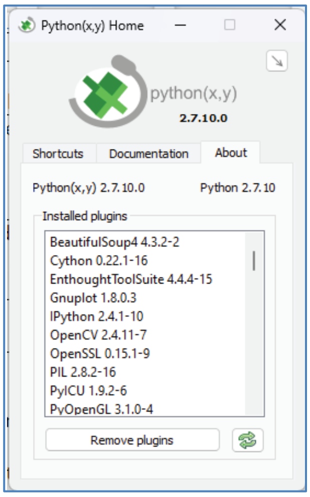
With Python(x,y), one can do:

- interactive calculations including for example 2D and 3D plotting or symbolic maths,
- simple functionnal programming (with MATLAB-like syntax) as well as powerful object-oriented programming,
- scientific projects development from the simplest script to the most sophisticated application thanks to Qt development framework and Spyder development environment,
- parallel computing on multicore/processors computers or even clusters (with Parallel Python),

And A lot more...

python(x,y)
Download Python-xy
Python programming language
Qt Application Development Framework
Spyder Interactive Scientific Development Environment
Interactive computing
with Matplotlib and IPython
GUI design tools
Qt - Eclipse integration
Scientific development environment
with Spyder (look familiar, isn\'t it?)

Figure 3.1 Download the Python(x,y) environment for installation



Python(x,y) 2.7.10.0　　Python 2.7.10

Installed plugins

BeautifulSoup4 4.3.2-2
Cython 0.22.1-16
EnthoughtToolSuite 4.4.4-15
Gnuplot 1.8.0.3
IPython 2.4.1-10
OpenCV 2.4.11-7
OpenSSL 0.15.1-9
PIL 2.8.2-16
PyICU 1.9.2-6
PyOpenGL 3.1.0-4

Figure 3.2 Python(x,y) interface

If the following libraries are missing after Python(x,y) installation, they will have to be installed manually in order to run some of the Python 2.7 scripts:

❶ **pywin32-214.win32-py2.7.exe** - Win32 is a programming interface that allows developers to create applications for the Microsoft Windows operating system, by exemple connect with Excel or Word from Python script [3.2];

❷ **pysqlite** - Interface to the SQLite 3.x embedded relational database engine

❸ **wxPython3.0-win32-docs-demos-3.0.0.0.exe** – this library can be optional installed to study wxPython demo, other samples and wxWidgets documentation; this package install a complete list of educational scripts exemples to generate wxPython controls [3.3].

To run Python 3.12 version Streamlit scripts, we recommend the use of the **Visual Studio Code** (**VSC**) editor and the installation of the following libraries:

❶ go to [3.4] and download and run the **VSCodeUserSetup-x64-1.90.2.exe** file (~ 96 Mb);

❷ select **VSC** icon from **Start** => **All aps;**

❸ install **Python Extension for VSC** - From **VSC** environment, activate **Extension** option with **Ctrl+Shift+X** keyboard, **Figure 3.3** and, from **Search extension** field, write **Python** to select the **Python** module; click **Install** blue button for installation; also, the **Pylance** and **Python Debugger** modules will be automatically installed to provide Python language support and to provide a debug experience with **debugpy**;

❹ use the **Ctrl+Shift+X** keyboard shortcut to enable the **Extension** option and, from **Search extension** field, write **Python Image Preview** to select this module; click **Install** blue button which will install the libraries: **Numpy**, **Pillow**, **OpenCV**, **Matplotlib**, **Plotly**, **ImageIO**, **Scikit Image, Tensorflow**, **Pytorch Image Preview**;

❺ use the **Ctrl+Shift+X** keyboard shortcut to enable the **Extension** option and, from **Search extension** field, write **SQLite** to select this module; click **Install** blue button to explore and query SQLite databases;

❻ if the content of **Search extension** field is erased, the installed modules will be displayed and, from the bottom **Recommended** list, new modules can be selected for installation;
❼ close **VSC** - from **VSC** menu go to : **File** => **Exit**;
❽ install **Streamlit** - launch **cmd** as administrator and write the following command: **pip install streamlit**.

To create an **Streamlit** application from scratch and run with Python 3.12 version the following steps must be performed:
➊ create a main folder **_Apps_Streamlit** that will keep all **Streamlit** applications; you can use any other name for main folder; it's recommended that each application has its own subfolder in the main folder.
➋ create subfolder **Hello** in main folder **_Apps_Streamlit**; this subfolder will contain the future **Streamlit Hello.py** application;
➌ create & save the **Hello.py** Python file in **Hello** subfolder, with the following content:

```
import streamlit as st
st.write('Hello World')
st.write('Modified Hello World ')
```

➍ this file will print ***Hello World*** & ***Modified Hello World*** textd on the default browser at run time; any text editor can be used to create the **Hello.py** file;
➎ start **VSC** and open **Hello** subfolder in **VSC** environment; from **VSC File** menu select **Add Folder to Workspace...** and select **Hello** subfolder with **Add** button; the folder files list will be displayed on **VSC** left side;
➏ in **VSC** environment open a new terminal by click on three dots points (**...**) and select **Terminal** => **New Terminal** option from the menu or taste **Ctrl + Shift + `** combination keybord;
➐ to run **Hello.py** file in **Terminal** window write the following command and hit **Enter** key:

```
streamlit run Hello.py
```

➑ in local browser at http://localhost:8501/ address will be displayed the following text, **Figure 3.4**:
➒ if the script is modified, a save operation must be performed with **File** => **Save**; however, the easiest way is to turn on the **Auto Save** setting by selecting this option from the menu.: **File** => **Auto Save**; any changes will be updated on browser window;
➓ assuming that the **Hello.py** is saved in **E:_Apps_Streamlit\Hello\** it is also possible to open a **Windows Terminal** and write the command to run the **Hello.py** application:

```
streamlit run E:\_Apps_Streamlit\Hello\Hello.py.
```

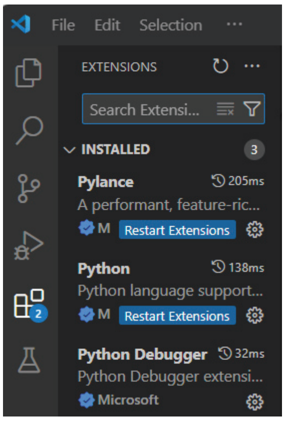
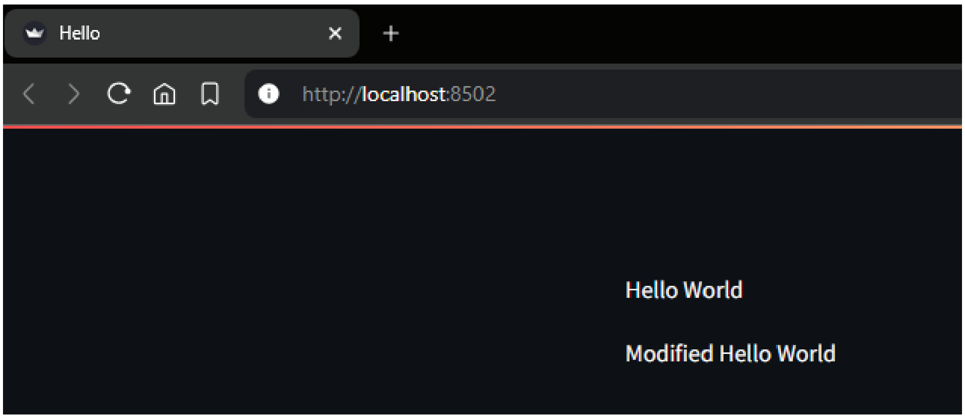


Figure 3.3 VSC extension interface



Hello World

Modified Hello World

Figure 3.4 Hello.py script running from **VSC** in browser window

To create an **Streamlit** application from scratch in **VSC** environment and run with Python 3.12 version the following steps must be performed:

➊ create a subfolder in main folder **_Apps_Streamlit**; this subfolder will contain the future application;
➋ start **VSC** and open the created subfolder in **VSC** environment; from **VSC File** menu click on **Add Folder to Workspace…** option and select the created subfolder with **Add** button; the list of folder files (if any) will be displayed on the left hand side of the **VSC**;

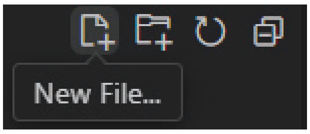


Figure 3.5 Create a new file

➌ click on **New File** icon, Figure 3.5, to create a new file *file name*.**py**;
➍ save the file *file name*.**py**;
➎ in **VSC** environment open a new terminal by click on three dots points (**…**) and select **Terminal** => **New Terminal** option from the menu or taste **Ctrl + Shift + `** combination keybord;
➏ to run the Python file in **Terminal** window write the following command and hit **Enter** key:

```
streamlit run file name.py
```

➐ in local browser at http://localhost:8501/ address will be displayed the file results.

To deploy an app on **Streamlit Community Cloud** the following steps must be taken [3.5], [3.6]:
➊ create a github repository with the code and with a requirements.txt file (where the app dependencies must be specified);
➋ sign in **https://streamlit.io/;**
➌ connect the **github** repo with your **Streamlit** account;
➍ deploy your app linking to the github repo where the code was stored;
➎ in some minutes the app will be deployed; the app can be set public from settings; the app subdomain will be something like: **myapp.streamlit.app**.

If you want to select from **VSC** one of the Python 2 or 3 version already installed on the computer, to run a non-**Streamlit** file, you must load the file from the **File** => **Open File** option menu and click on the bottom right line of **VSC** on the Python current name, Figure 3.6 and select the desired interpreter from the top list, Figure 3.7; to start file execution, click on the right arrow icon located in the top right area, Figure 3.8.

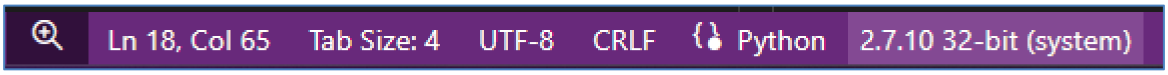
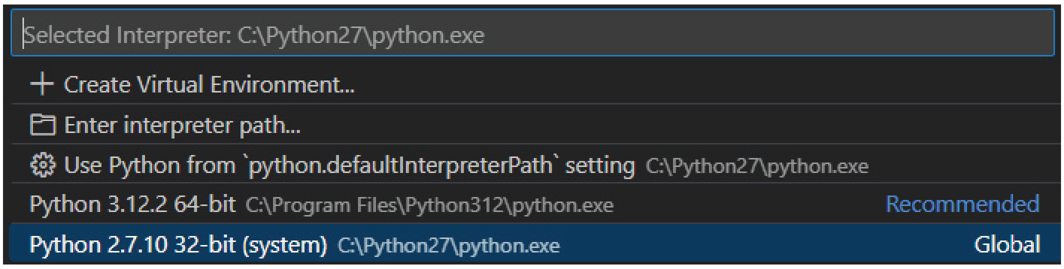


Figure 3.6 Activate the selection of the desired Python version



Selected Interpreter: C:\Python27\python.exe
+ Create Virtual Environment...
Enter interpreter path...
Use Python from `python.defaultInterpreterPath` setting C:\Python27\python.exe
Python 3.12.2 64-bit C:\Program Files\Python312\python.exe Recommended
Python 2.7.10 32-bit (system) C:\Python27\python.exe Global

Figure 3.7 Select the desired Python version

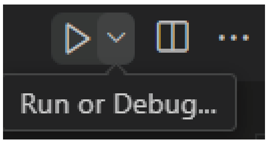


Figure 3.8 The icon to run the Python file

If you want to install a specific library for a specific version of Python, you can go to the [3.7] web page, where are saved ~ 120,000,000 files.

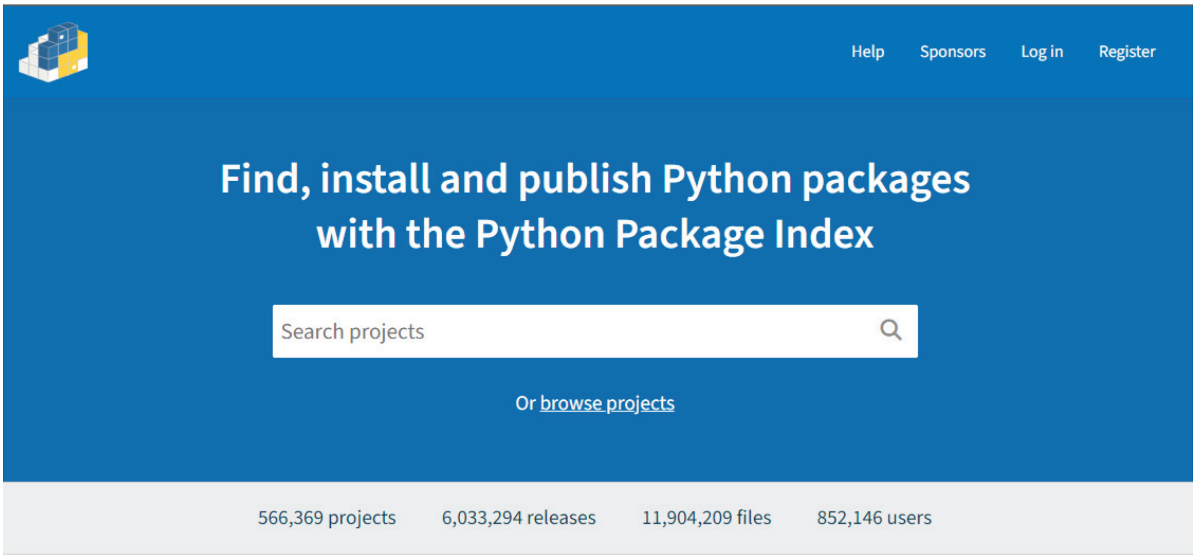


Figure 3.9 WEB page to find Python libraries [3.7]

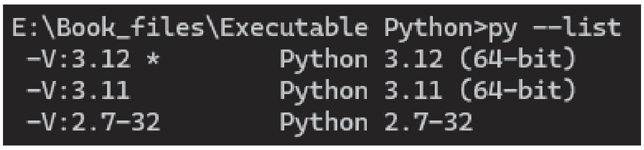
To display the Python versions installed on the computer, the command **py –list** or **py -0** can be use in **command prompt**, **Figure 3.10**.

The **py.exe** placed in **command prompt** will automatically select and launch the most recent version of Python you've installed, **Figure 3.11**.

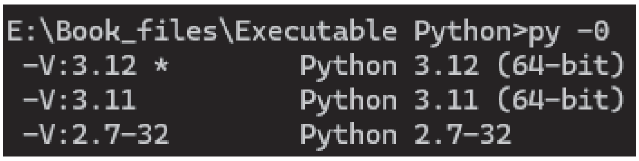
You can also use commands like **py -2.7** to select a particular version, **Figure 3.12**.

The **python --version** command display the active Python version, **Figure 3.13**.

To install a **library** for a specific Python versions, the command **pip2.7 install library** or **pip3.11 install library** can be use in **command prompt**, **Figure 3.14**.

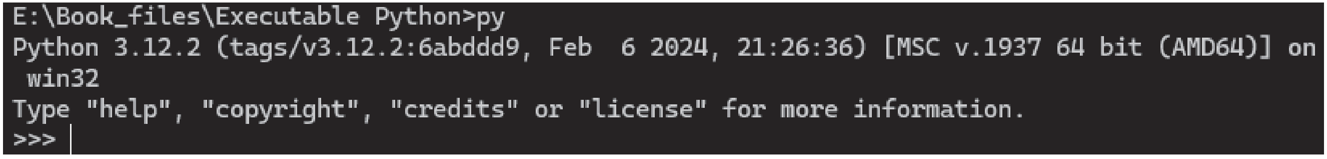


```
E:\Book_files\Executable Python>py --list
 -V:3.12 *         Python 3.12 (64-bit)
 -V:3.11           Python 3.11 (64-bit)
 -V:2.7-32         Python 2.7-32
```



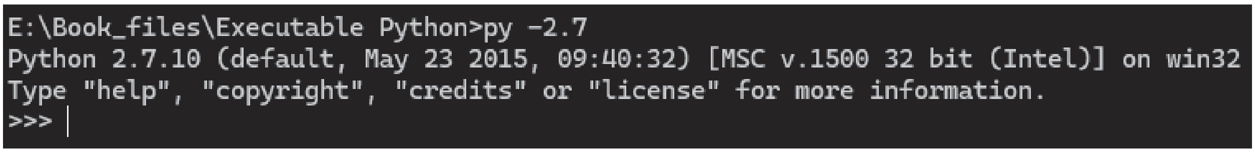
```
E:\Book_files\Executable Python>py -0
 -V:3.12 *         Python 3.12 (64-bit)
 -V:3.11           Python 3.11 (64-bit)
 -V:2.7-32         Python 2.7-32
```

Figure 3.10 Display the Python versions installed on the computer



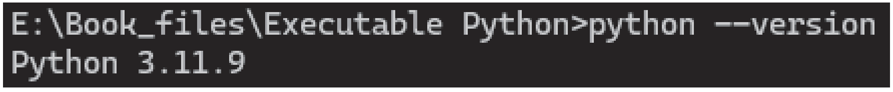
```
E:\Book_files\Executable Python>py
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on
 win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 3.11 Launch the most recent version of installed Python



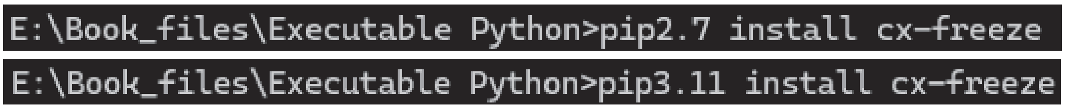
```
E:\Book_files\Executable Python>py -2.7
Python 2.7.10 (default, May 23 2015, 09:40:32) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 3.12 Launch a specific version of installed Python



```
E:\Book_files\Executable Python>python --version
Python 3.11.9
```

Figure 3.13 Display the active Python version



```
E:\Book_files\Executable Python>pip2.7 install cx-freeze
E:\Book_files\Executable Python>pip3.11 install cx-freeze
```

Figure 3.14 Install a **library** for a specific Python versions

4. Integration by rectangles method

Let consider the parabolic function: **Y=X^2**, Figure 4.1. The area under the curve **Y=X^2**, between the limits **X=a** and **X=b** respectively, is computed exactly by the integral formula:

$$Aria = \int_a^b y \cdot dx = \int_a^b X^2 \cdot dx = \frac{X^3}{3} \Big|_b^a = \frac{b^3 - a^3}{3}$$

The approximate calculation of the integral can be done by numerical integration through method of rectangles, approximating the area by the sum of the elementary rectangles **ad$_i$ = l x Y$_i$** , of width **l=(b-a)/N** and height **Y$_i$=Xi ^{2}**, where N - is the imposed number of rectangles. The abscissa **X$_i$** in the **Y$_i$** expression is calculated as follows:

$$\mathbf{X_1=a+l/2 \; ; \; X_2=a+l+l/2 \; ; \; X_3=a+2*l+l/2 \; ; \; X_4=a+3*l+l/2 \; ....,}$$

and for the general case "i" :

$$\mathbf{X_i=a+(i-1)*l+l/2.}$$

The input data are: the number of intervals (elementary rectangles) N and the limits of integration a, b; the results of this script is the area calculated by rectangles method approximation, compared with that obtained by exact integration (for the parabolic function).

The interval [a, b] is divided into N elementary rectangles of constant width 'l' and variable height 'Y$_i$', calculated in the middle of the elementary rectangles width. The area obtained by summing the elementary rectangles is an approximation of the area under the curve; it is expected that the accuracy approximation is expected to increase with increasing the number of elementary rectangles. The procedure can be applied for functions whose integration cannot be applied because their complexity.

The variable **'aria'** is the variable that collects the sum of the elementary rectangles areas **ad$_i$** in a FOR cycle, inside which the parameters of each elementary rectangle are successively summed. Finally the area computed by summation is displayed compared with the exact area computed by the formula resulting from the analytical integration.

The script exemplifies a mathematical application, reducing the integral calculation to a summation. The script results, for different values of integration limits a, b respectively numbers of intervals N, are given in table 4.1, from which the accuracy increases for higher values of N.

Of course the script is customized for the parabolic function **Y=X^2**, but, by changing the function expression, the script can also be applied to other functions.

This is the simplest numerical integration method. There are other methods with higher accuracy, such as: the trapezoidal method, the Simpson method, the Romberg method with Richardson extrapolation or the Gauss method.

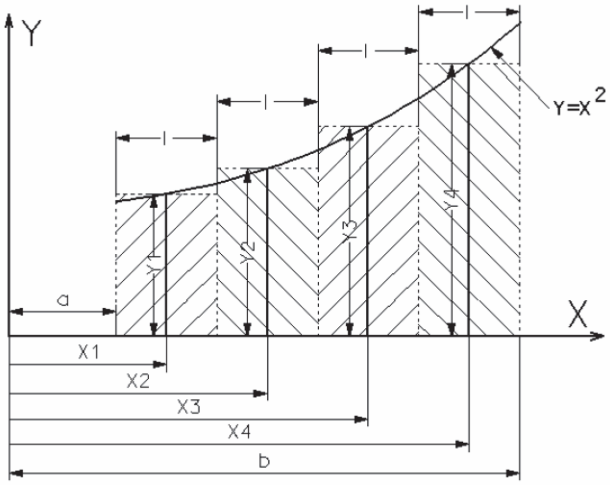


Figure 4.1

Table 4.1

a	b	N	Exact area	Approximate area
1	3	3	8.6667	8.5926
		10		8.66
		50		8.6664
		100		8.6666
1	50	3	41,666.3333	40,576.907
		10		41,568.295
		50		41,662.4117
		100		41,665.3529

Listing 4.1 displays the Python & wxPython version of script, Integrate.pyw and Listing 4.2 displays the Python & Streamlit version of the same script, Integrate.py. Figure 4.2a & 4.2b display the screens generated by Python & wxPython version of the script, while Figure 4.3a and 4.3b display the screens generated by Python & Streamlit version.

The Integrate.pyw script (Python & wxPython version) include the funct function and two classes: Integrate & Plot_ Window. The script begin with importing libraries: wx (the cross-platform GUI toolkit for the Python language), os (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.) and matplotlib (library for creating static, animated and interactive visualizations in Python). The public funct define the parabola expression, before any class of the script.

The Integrate class compute the integration based on the rectangles method and include the following functions:

- __init__ This function is is used to access variables that belongs to the class; at the end the calculation function is called. The variables self.X and self.Y are initialized as list type. Here the Lim_A, Lim_B, N values are transmitted when the class is called in line 97.
- calculation Here is applied rectangles method algorithm were the elementary rectangles areas adi are calculated and summed to self.aria variable.
- get_aria Return outside the class the value of area under the curve (lines 34, 121).
- get_X Return outside the class the values of self.X list (lines 36, 98).
- get_Y Return outside the class the values of self.Y list (lines 38, 98).

The Plot_ Window class include the following functions:

- __init__ Define the frame (a window whose size and position can be changed by the user), the panel (that will hold the contents of the frame) the StatusBar (placed at the frame bottom, were messages can be displayed); also, call the _TOOLBAR function, initializes top level container for all plot elements and center the frame in the display.
- _TOOLBAR Create the toolbar (lines 55÷57), the Image button (lines 59÷60) and connect with Bind statement (line 60) to OnImage function to display the picture from Figure 4.2a and create three text fields (wx.TextCtrl) to input numerical values of input data variables; before text fields three wx.StaticText controls are defined as labels; create the Calculate button (line 67) and connect with Bind statement to OnClicked function (line 68). Lines 70÷77 add these controls to the ToolBar and lines 78÷79 generate and show the ToolBar.
- OnImage Display the picture from Figure 4.2a (lines 81÷83).
- MouseMotion Display in the StatusBar the current values of X & Y mouse position in the chart.
- Recreate_Axis Initialize the chart (line 88), show the chart grid (line 89) and set the labels for X & Y axes of the chart (lines 90÷91).
- OnClicked The function is called by Calculate button; extract the input data values (a, b, N) from text fields (lines 93÷95), calculate the interval length (l - line 96), call Integrate class (line 97) and extract the list values to X and Y list type (line 98); create the canvas element were the chart will be placed (line 99); connect the MouseMotion function with the canvas figure (line 100); create the sizer (line 101) and place the canvas inside the sizer (line 102); line 103 call Recreate_Axis function; lines 104÷116 plot the chart elements from 4.2b; the chart is updated by the draw statement from line 120; lines 121÷124 place the exact and approximate area values in StatusBar.

Line 126 create the app which initializes the GUI toolkit for xPython; line 127 call Plot_Window class to create the window showed by line 129; finally, the line 131 enter into infinite continuous loop to receive key events from the user; the script can be interrupted with ☒ button of the frame.

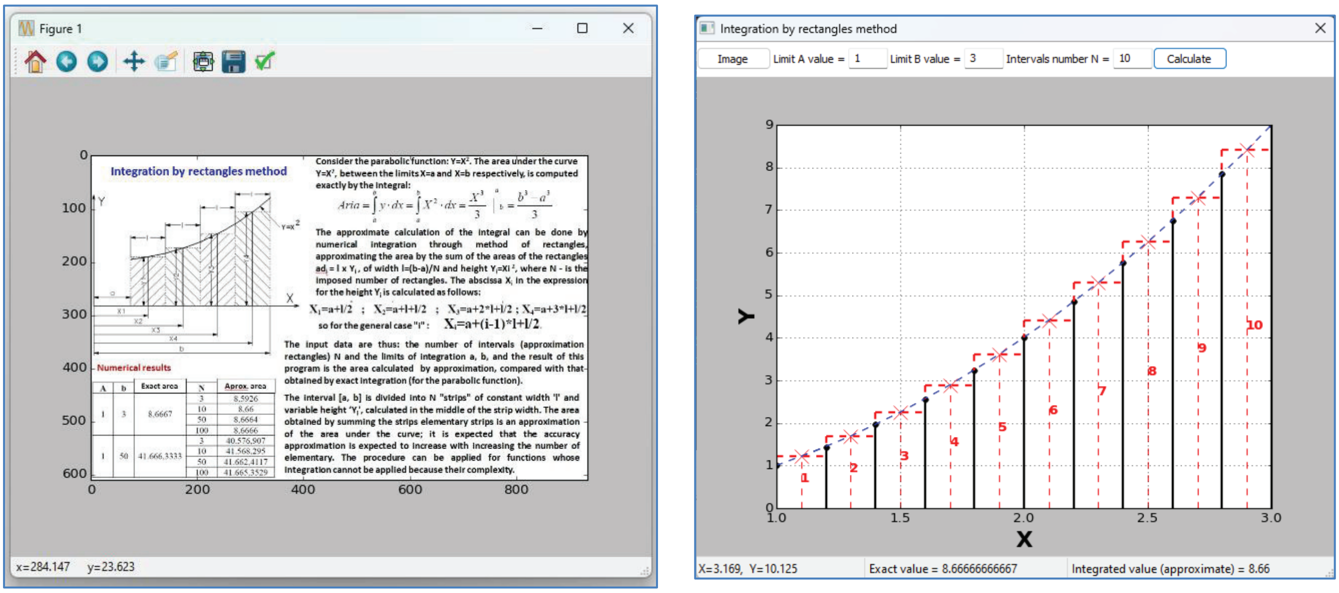
Listing 4.1 Integrate.pyw - Integration by rectangles method – Python & wxPython version

```
# The // operator will be available to request floor division unambiguously. This
# statement will change the / operator to mean true division throughout the module.
from __future__ import division

import wx    # import wx module
import os
import matplotlib as mpl  # The tools are imported from the Matplotlib library
import matplotlib.pyplot as plt
import matplotlib.image as img
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar

def funct(x):   # Define public function 'funct'
    y=x*x       # Calculate the 'y' value as power of 'x' variable (parabolic function)
    return y   # The function return 'y' value

class Integrate:
    # The self parameter is a reference to the current instance
    # of the class, and is used to access variables that belongs to the class.
    def __init__(self, Lim_A, Lim_B, N):
        # The __init__() function is called automatically every
        # time the class is being used to create a new object.
        self.A = Lim_A   ;   self.B = Lim_B   ;   self.N = N
        self.aria=0   ;   self.X=[]   ;   self.Y=[]
        self.calculation()   # Call 'calculation' function
    def calculation(self):
        l=(self.B-self.A)/self.N
        for i in range(1,self.N+1):
            xi = self.A+(i-1)*l+l/2  ;  self.X.append(xi)
            yi = funct(xi)  ;  self.Y.append(yi)
            adi = l * yi
            self.aria = self.aria+adi
    def get_aria(self):
        return self.aria
    def get_X(self):
        return self.X
    def get_Y(self):
        return self.Y

class Plot_Window(wx.Frame):
    def __init__(self, parent, title):
        wx.Frame.__init__(self, parent, title=title, size=(650,600),
            style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
            wx.MAXIMIZE_BOX )) # Initialize the frame
        self.panel = wx.Panel(self)  #  The panel will hold the contents of the frame
        self.statusbar = self.CreateStatusBar() # Create the status bar
        self.statusbar.SetFieldsCount(3)
        self.statusbar.SetStatusWidths([-25, -30, -40])
        self.statusbar.SetStatusText("This is the plot window",0)  # Add text to StatusBar
        self._TOOLBAR() # Create toolbar
        self.figure = mpl.figure.Figure()  # Initializes top level container for all plot elements
        self.axes=self.figure.add_subplot(111) # Add only one subplot area
        self.Center()  # Center the frame in display
    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour("white")
        self.toolbar.SetToolBitmapSize((24,24))
        self.btn_Img = wx.Button(self.toolbar, -1, "Image")
        self.btn_Img.SetToolTipString("Show image of integrate algorithm")
        self.btn_Img.Bind(wx.EVT_BUTTON,self.OnImage)
        st1 = wx.StaticText(self.toolbar, -1, " Limit A value = ")
        self.txt_Lim_A = wx.TextCtrl(self.toolbar, value="1", size=(40,-1))
        st2 = wx.StaticText(self.toolbar, -1, " Limit B value = ")
        self.txt_Lim_B = wx.TextCtrl(self.toolbar, value="3", size=(40,-1))
        st3 = wx.StaticText(self.toolbar, -1, " Intervals number N = ")
        self.txt_N = wx.TextCtrl(self.toolbar, value="10", size=(40,-1))
        self.btn = wx.Button(self.toolbar, -1, "Calculate")
        self.btn.Bind(wx.EVT_BUTTON,self.OnClicked)

        self.toolbar.AddControl(self.btn_Img)
        self.toolbar.AddControl(st1)
```

Listing 4.1 Integrate.pyw - Integration by rectangles method – Python & wxPython version

```
        self.toolbar.AddControl(self.txt_Lim_A)
        self.toolbar.AddControl(st2)
        self.toolbar.AddControl(self.txt_Lim_B)
        self.toolbar.AddControl(st3)
        self.toolbar.AddControl(self.txt_N)
        self.toolbar.AddControl(self.btn )
        self.toolbar.Realize()    # Toolbar generation
        self.toolbar.Show()       # Toolbar show
    def OnImage(self, event):
        testImage = img.imread(os.getcwd()+'/Integrate.jpg')
        plt.imshow(testImage)
        plt.show()
    def MouseMotion(self, event):
        a1,b1=self.axes.transData.inverted().transform([event.x, event.y])
        self.statusbar.SetStatusText("X="+'%0.3f' % a1+",  Y="+'%0.3f' % b1,0)
    def Recreate_Axis(self):
        self.axes.cla()
        self.axes.grid(True)
        self.axes.set_xlabel('X', fontsize=20, fontweight='bold')
        self.axes.set_ylabel('Y', fontsize=20, fontweight='bold')
    def OnClicked(self, event):
        a = float(self.txt_Lim_A.GetValue())
        b = float(self.txt_Lim_B.GetValue())
        N = int(self.txt_N.GetValue())
        l=(b-a)/N
        cv = Integrate(a, b, N)
        X = cv.get_X()  ;   Y = cv.get_Y()  ; lx=len(X)    ; lx1=len(X)-1
        self.canvas = FigureCanvas(self.panel,-1,self.figure)
        self.axes.figure.canvas.mpl_connect('motion_notify_event', self.MouseMotion)
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas, 1, wx.TOP | wx.LEFT | wx.EXPAND)
        self.Recreate_Axis()
        for i, ( xplot, yplot) in enumerate(zip(X, Y)):
            self.axes.plot(xplot,yplot, 'x', color="Red", markersize=12) # Mark median curve points
            self.axes.plot((xplot,xplot),(0,funct(xplot)), '--', color="Red",linewidth=1.0) # Median vertical lines
            self.axes.text(xplot,funct(xplot)/2, str(i+1), color="Red", weight='bold', ha='center',va='center')
        x1=xplot-l/2  ; y1= funct(x1)
            self.axes.plot(x1,y1, '.', color="Black", markersize=10) # Mark curve points
            self.axes.plot((x1,x1),(0,funct(x1)), '-', color="Black",linestyle="solid",linewidth=2.0) # Vertical lines
        self.axes.plot(b,funct(b), '.', color="Black", markersize=10)  # Left vertical lines
        self.axes.plot((b,b),(0,funct(b)), '-', color="Black",linestyle="solid",linewidth=2.0) # Right vertical lines
        self.axes.plot(X,Y, 'b--',linewidth=1.0) # Draw the parabola curve
        self.axes.plot((X[0]-l/2,X[0]+l/2),(funct(a),funct(X[0]+l/2)), '--', color="Blue",linewidth=1.0)  # Close the left curve
        self.axes.plot((X[lx1],X[lx1]+l/2),(funct(X[lx1]),funct(b)), '--', color="Blue",linewidth=1.0) # Close the right curve
        for i in range(lx):  # Draw rectangle horizontal lines
            self.axes.plot((X[i]-l/2,X[i]+l/2),(funct(X[i]),funct(X[i])), '--', color="Red",linewidth=2.0)
        for i in range(lx): # Close red rectangles with left vertical lines
            self.axes.plot((X[i]-l/2,X[i]-l/2),(funct(X[i]),funct(X[i]-l/2)), '--', color="Red",linewidth=2.0)
    self.axes.figure.canvas.draw()
        Exact_val = (b**3-a**3)/3
        self.statusbar.SetStatusText("Exact value = "+str(Exact_val),1)  # Add text to StatusBar
        Integrated_value=str(cv.get_aria())
        self.statusbar.SetStatusText("Integrated value (approximate) = "+Integrated_value,2)
        self.SetSizer(sizer)
        self.Fit()

app = wx.App(redirect=False)   # wx.App initializes the GUI toolkit for WxPython.
window=Plot_Window(None, "Integration by rectangles method")  # Call 'Plot_Window' class
window.Show()  # Show 'window' object
# This keeps the GUI in a continuous loop that is ready to receive key events
# from the user. It does not return until the program closes!
app.MainLoop()
```

The **Integrate.py** script (Python & Streamlit version) include the public **funct** function to define the parabola expression. The script begin with importing libraries: **streamlit** (create a great interactive web application with Python), **PIL** (Python Imaging Library), **matplotlib** (library for creating static, animated and interactive visualizations in Python), **pathlib** (provide classes to represent abstract/concrete paths), **numpy** (the fundamental package for scientific computing with Python). This script contains 62 lines.



Integration by rectangles method

Consider the parabolic function: Y=X². The area under the curve Y=X², between the limits X=a and X=b respectively, is computed exactly by the integral:

$$Aria=\int_a^b y\cdot dx=\int_a^b X^2\cdot dx=\frac{X^3}{3}\Big|_b^a=\frac{b^3-a^3}{3}$$

The approximate calculation of the integral can be done by numerical integration through method of rectangles, approximating the area by the sum of the areas of the rectangles $ad_i = l \times Y_i$, of width $l=(b-a)/N$ and height $Y_i=Xi^2$, where N - is the imposed number of rectangles. The abscissa X_i in the expression for the height Y_i is calculated as follows:

$X_1=a+l/2$; $X_2=a+l+l/2$; $X_3=a+2*l+l/2$; $X_4=a+3*l+l/2$

so for the general case "i": $X_i=a+(i-1)*l+l/2$.

The input data are thus: the number of intervals (approximation rectangles) N and the limits of integration a, b, and the result of this program is the area calculated by approximation, compared with that obtained by exact integration (for the parabolic function).

The interval [a, b] is divided into N "strips" of constant width 'l' and variable height 'Y_i', calculated in the middle of the strip width. The area obtained by summing the strips elementary strips is an approximation of the area under the curve; it is expected that the accuracy approximation is expected to increase with increasing the number of elementary. The procedure can be applied for functions whose integration cannot be applied because their complexity.

Numerical results

A	b	Exact area	N	Aprox. area
1	3	8,6667	3	8,5926
			10	8,66
			50	8,6664
			100	8,6666
1	50	41.666,3333	3	40.576,907
			10	41.568,295
			50	41.662,4117
			100	41.665,3529

x=284.147 y=23.623

Integration by rectangles method

Limit A value = 1 Limit B value = 3 Intervals number N = 10

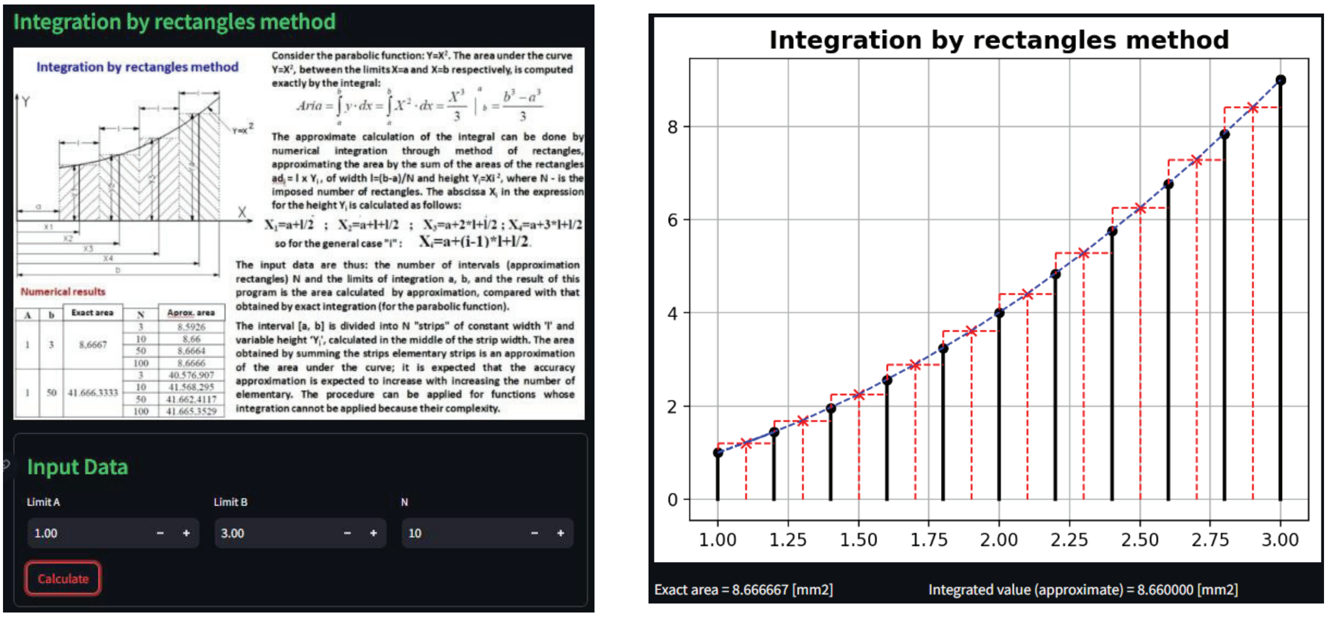
X=3.169, Y=10.125 Exact value = 8.66666666667 Integrated value (approximate) = 8.66

Figure 4.2a

Figure 4.2b

Figure 4.2 The screens generated by Python & wxPython version of the Integrate.pyw script

Lines 1 to 5 import the required modules. Lines 7 to 9 define the **funct** function. Line 12 defines the current folder. Line 13 defines the script title with **subheader** command. Line 15 inserts a container element (with **st.empty()** command) into the script that can be used to hold a single element like **Integrate.jpg** image (**Figure 4.3a**) and also define the width of the image and no caption. Line 17 creates a form where input data will be specified. Line 18 defines the form title with **subheader** command. Line 19 creates columns with 3 elements. Lines 20 to 22 define the fields of input data (**a**, **b**, **N**) into the previous columns created, using **number_input** command. The form require 3 numerical data: **a**, **b**, **N** and the **number_input** command specify the default values that can be changed by the user. Line 23 creates the button with the **form_submit_button** command and set the caption **Calculate**. When this button is clicked, all widget values inside the form will be sent to Streamlit in a batch. Every form must have a **form_submit_button**, which cannot exist outside a form. Line 24 verifies if the button was clicked and, only for **True** value, run the lines 25 to 31 where integration calculation is made. Lines 34÷56 plot the chart elements from **Figure 4.3b**. Lines 61÷62 display the exact and approximate area values.



Integration by rectangles method

Integration by rectangles method

Consider the parabolic function: Y=X². The area under the curve Y=X², between the limits X=a and X=b respectively, is computed exactly by the integral:

$$Aria=\int_a^b y\cdot dx=\int_a^b X^2\cdot dx=\frac{X^3}{3}\Big|_b^a=\frac{b^3-a^3}{3}$$

The approximate calculation of the integral can be done by numerical integration through method of rectangles, approximating the area by the sum of the areas of the rectangles $ad_i = l \times Y_i$, of width $l=(b-a)/N$ and height $Y_i=Xi^2$, where N - is the imposed number of rectangles. The abscissa X_i in the expression for the height Y_i is calculated as follows:

$X_1=a+l/2$; $X_2=a+l+l/2$; $X_3=a+2*l+l/2$; $X_4=a+3*l+l/2$

so for the general case "i": $X_i=a+(i-1)*l+l/2$.

The input data are thus: the number of intervals (approximation rectangles) N and the limits of integration a, b, and the result of this program is the area calculated by approximation, compared with that obtained by exact integration (for the parabolic function).

The interval [a, b] is divided into N "strips" of constant width 'l' and variable height 'Y_i', calculated in the middle of the strip width. The area obtained by summing the strips elementary strips is an approximation of the area under the curve; it is expected that the accuracy approximation is expected to increase with increasing the number of elementary. The procedure can be applied for functions whose integration cannot be applied because their complexity.

Numerical results

A	b	Exact area	N	Aprox. area
1	3	8,6667	3	8,5926
			10	8,66
			50	8,6664
			100	8,6666
1	50	41.666,3333	3	40.576,907
			10	41.568,295
			50	41.662,4117
			100	41.665,3529

Input Data

Limit A 1.00 Limit B 3.00 N 10

Exact area = 8.666667 [mm2] Integrated value (approximate) = 8.660000 [mm2]

Figure 4.3a

Figure 4.3b

Figure 4.3 The screens generated by Python & Streamlit version of the Integrate.py script

Listing 4.2 Integrate.py - Integration by rectangles method – Python & Streamlit version

```python
import streamlit as st # Import streamlit library
from PIL import Image  # PIL is the Python Imaging Library
import matplotlib.pyplot as plt
from pathlib import Path # This module provides classes to represent abstract/concrete paths
import numpy

def funct(x): # Define public function 'funct'
    y=x*x     # Calculate the 'y' value as power of 'x' variable (parabolic function)
    return y  # The function return 'y' value

# Define the application path
current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()
st.subheader(":green[Integration by rectangles method]")  # Application subtitle
# Place the integrate method image
st.empty().image(Image.open(str(current_dir)+"/"+"Integrate.jpg"), width=700, caption='')

with st.form('Input Data'):  # Create input data form
    st.subheader(':green[Input Data]')   # Form subtitle
    col1, col2, col3 = st.columns(3)  # Insert containers laid out as side-by-side columns
    a = col1.number_input('Limit A ', value=1.0) # Control to input 'a' value
    b = col2.number_input('Limit B ', value=3.0) # Control to input 'b' value
    N = col3.number_input('N ', value=10) # Control to input 'N' value
    submit_button = st.form_submit_button('Calculate')  # Display a form submit button.
if submit_button:
    l=(b-a)/N # The interval length
    aria=0  ;  X=[]  ;  Y=[] # Initialize one variable 'aria' and two lists 'X', 'Y'
    for i in range(1,N+1):
        xi = a+(i-1)*l+l/2  ;  X.append(xi) # Calculate x-coordinate and append to list 'X'
        yi = funct(xi)  ;  Y.append(yi) # Calculate y-coordinate and add to list 'Y'
        adi = l * yi # Calculate area of 'i' rectangle
        aria = aria+adi # Sum area of 'i' rectangle to total 'aria'

    # Chart drawing
    Place_Chart = st.empty() # Inserts a container into app that can be used to hold a single element
    fig=plt.figure() # Create chart figure
    plt.grid(True) # Show the chart grid
    plt.title("Integration by rectangles method", fontsize=14,
        fontweight='bold',color='Black') # Define chart title
    lx=len(X) ; lx1=len(X)-1
    for i in range(0, N):
        plt.plot([X[i], X[i]],[Y[i], Y[i]], 'x', color='Red', linewidth=2) # Mark median curve points
        plt.plot([X[i], X[i]],[0, funct(X[i])], '--', color='Red',linewidth=1) # Median vertical lines
        x1=X[i]-l/2  ; y1= funct(x1)
        plt.plot(x1,y1, '.', color="Black", markersize=10) # Mark curve points
        plt.plot((x1,x1),(0, funct(x1)), '-', color="Black",linewidth=2.0) # Vertical lines
        plt.plot(b,funct(b), '.', color="Black", markersize=10) # Right vertical point
        plt.plot((b,b),(0,funct(b)), '-', color="Black",linewidth=2.0) # Right vertical line
        plt.plot((X[0]-l/2,X[0]+l/2),(funct(a),funct(X[0]+l/2)), '--', color="Blue",linewidth=1.0) # Left semicurve line
    for i in range(lx-1):
        plt.plot((X[i],X[i+1]),(Y[i],Y[i+1]), '--', color="Blue",linewidth=1.0)   # Curve lines
    plt.plot((X[lx1],X[lx1]+l/2),(funct(X[lx1]),funct(b)), '--', color="Blue",linewidth=1.0) # Right semicurve line
    for i in range(lx):  # Draw rectangle horizontal lines
        plt.plot((X[i]-l/2,X[i]+l/2),(funct(X[i]),funct(X[i])), '--', color="Red",linewidth=1.0)
    for i in range(lx): # Close red rectangles with left vertical lines
        plt.plot((X[i]-l/2,X[i]-l/2),(funct(X[i]),funct(X[i]-l/2)), '--', color="Red",linewidth=1.0)
    Place_Chart.write(fig)    Place_Chart.write(fig)

    # Chart dimensions & coordinates
    col1, col2 = st.columns([0.4,0.6]) # Insert containers laid out as side-by-side columns
    Exact_val = (b**3-a**3)/3 # Calculate exact area by mathematical formula
    col1.write("Exact area = "+'%0.6f' % Exact_val+" [mm2]") # Print exact area
    col2.write("Integrated value (approximate) = "+'%0.6f' % aria +" [mm2]") # Print integrated area
```

5. Integration by Gauss method

The general Gauss integration formula is:

$$\int_a^b f(x)dx = h_1 \cdot \sum_{i=1}^{16} a_i \cdot f(x_i)$$

$$x_i = h_1 * xg_i + h_2$$

$$h_1 = \frac{b-a}{2}$$

$$h_2 = \frac{b+a}{2}$$

so it convert the integral calculation into a sum of 16 terms, in which the values of the function **f** are computed in abscissa **x_i** imposed, the values coefficients **a_i** and **xg_i** are given in the right table. These coefficients are irrational numbers, calculated under the condition that the above formula is exact for any algebraic polynomial of degree (2n-1), where n=16.

i	**a_i**	**xg_i**
1.	0.02715245	-0.98940093
2.	0.06225352	-0.94457502
3.	0.09515851	-0.86563120
4.	0.12462897	-0.75540440
5.	0.14959598	-0.61787624
6.	0.16915651	-0.45801677
7.	0.18260341	-0.28160355
8.	0.18945061	-0.09501250
9.	0.18945061	0.09501250
10.	0.18260341	0.28160355
11.	0.16915651	0.45801677
12.	0.14959598	0.61787624
13.	0.12462897	0.75540440
14.	0.09515851	0.86563120
15.	0.06225352	0.94457502
16.	0.02715245	0.98940093

Numerical integration by Gauss method combines a high degree of accuracy with respect to relatively small number of points through calculates the integral and the particular simplicity of the computational procedure. The program was tested for numerical integration of the functions from **table 5.1**. The calculated values from **table 5.1** show a great accuracy of this integration method.

Listing 5.1 displays the Python & wxPython version of script, **Gauss.pyw** and **Listing 5.2** displays the Python & Streamlit version of the same script, **Gauss.py**.

Table 5.1

ID	**Function**	**Integration limits**		**Exact value**	**Value from Gauss integration**	
		a	**b**		**Gauss.pyw**	**Gauss.py**
1a	$f(x) = x^2$	1	3	8.6667	8.66666629571	8.666666295713735
1b		1	50	41.666,3333	41666.3313095	41666.33130952317
2	$\int_0^1 x^2\sqrt{1-x^2}dx$	0	0.9999	$\frac{\pi}{16}$ = 0.19634954	0.196378392758	0.19637839275823257
3	$\int_0^1 \frac{1}{1+x^2}dx$	0	1	$\frac{\pi}{4}$ = 0. 78539816	0.785398132586	0.7853981325857171
4	$\int_0^{\pi/2} SIN(x)dx$	0	$\frac{\pi}{2}$	1	0.999999963949	0.9999999639540322
5	$\int_0^{2\pi} x \cdot SIN(x)dx$	0	$2 \cdot \pi$	− 2π = - 6.2831853	-6.28318504761	-6.283185047606339
6	$\int_{-1}^1 \sqrt{1-x^2}dx$	-0.9999	0.9999	$\frac{\pi}{2}$ =1.5707963	1.5709695597	1.57096955970272
7	$\int_0^{\pi} \ln[1-8 \cdot COS(x)+16]dx$	0	π	2π ln(4) = 8.7103444	8.71034401767	8.710344017664525

The **Gauss.pyw** script (Python & wxPython version) include only one class: **Plot_Window** and 10 public functions (**funct1a**, **funct1b**, **funct2** … , **funct7**, **Put_Clipboard**, **Gauss**). The script begin with importing libraries: **wx** (the cross-platform GUI toolkit for the Python language), **wx.combo** (a combobox that displays bitmap in front of the list items; it currently only allows using bitmaps of one size and resizes itself so that a bitmap can be shown next to the text field), **os** (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.), **math** (module which provides access to the mathematical functions), **matplotlib** (library for creating static, animated and interactive visualizations in Python), **win32con** (this module contains constants related to Win32 programming) and **win32clipboard** (a module which supports the Windows Clipboard API).

The public functions **funct1a**, **funct1b**, **funct2** … , **funct7** define the tested function according to **table 5.1**.

The **Put_Clipboard** function receive a string **Sir_Clip** as parameter, open and empty the clipboard, place the string into clipboard and close the clipboard.

The **Gauss** function receive the integration limits **a** & **b** and the function **FUNCT** for integration as parameters. Line 47 initialize the **ag** and **xg** lists. Lines 48÷52 store the values of Gauss coefficients from indexes 9÷16. The **for** cycle (lines 54÷56) generate the values of Gauss coefficients from indexes 1÷8. Indexing in Python starts at 0, which means that the first element in a array has an index of 0, the second element has an index of 1 and so on. In this case, the **ag** and **xg** lists are dimensioned with 17 elements to avoid using index 0 of the list. Lines 57÷63 calculate the integral value according to Gauss integration method. A **return** statement is used to end the execution of the function call and returns the result (value of the expression following the return keyword) to the caller. Usually a function returns a single result. In our case we want to return three results stored in a single list **lst_return**: **suma_Gauss** which is the value of the integral and lists **x**, **y** which store the coordinates of Gauss integration points, which will be drawn in **OnCalculate** function of **Plot_Window** class.

The **Plot_Window** class create the interface, calculate the selected function integral, draw the function curve and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame) the **StatusBar** (placed at the frame bottom, were messages can be displayed); call the **_TOOLBAR** function; initializes top level container for all plot elements (line 77); add only one subplot area (line 78) and center the frame in the display.
- **_TOOLBAR** Create the **toolbar** (lines 82÷84) and the control **self.cb_ImageList** as **wx.combo.BitmapComboBox** to store the images and functions name like in **Figure 5.1**; lines 89÷91 create a list with function names and lines 92÷97 add to **self.cb_ImageList** control the function images and names. Line 88 connect **self.cb_ImageList** control with **Bind** statement to **OnFunction** function. Lines 99÷102 create the two **wx.TextCtrl** to input the integration limits; create the **Calculate** button (line 103) and connect with **Bind** statement to **OnCalculate** function (line 104); lines 106÷112 add these controls to the **ToolBar** and line 113÷114 generate and show the **ToolBar**.
- **OnFunction** Based on selected function from the **cb_ImageList** control (line 116) this class function set the function name in **FUNCT** variable and store in **self.txt_Lim_A** & **self.txt_Lim_B** text controls the values of integration limits.
- **MouseMotion** Display in the **StatusBar** the current values of X & Y mouse position in the chart.
- **Recreate_Axis** Initialize the chart (line 137), show the chart grid (line 138) and set the labels for **X** & **Y** axes of the chart (lines 139÷140).
- **OnCalculate** Read the the values of integration limits (lines 142÷143), generate **self.canvas** and **self.axes** areas for curve drawing, draw the curve drawing (lines 149÷159), call the **Gauss** public function (line 161), extract the selected function name in **sel** variable (line 163), put the integral value **Int_Gauss** into **self.axes** title and **StatusBar** (lines 164÷165) and plot the Gauss integration points with red colour (lines 168÷170).

Listing 5.1 Gauss.pyw – Integration by Gauss method – Python & wxPython version

```
from __future__ import division

import wx
import wx.combo
import os
import math

import matplotlib as mpl  # The tools are imported from the Matplotlib library
import matplotlib as plt
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas

import win32con # This module contains constants related to Win32 programming
import win32clipboard

def funct1a(x):
        y=x*x
        return y
def funct1b(x):
        y=x*x
        return y
def funct2(x):
        y=x*x*math.sqrt(1-x*x)
        return y
def funct3(x):
        y=1/(1+x*x)
        return y
def funct4(x):
        y=math.sin(x)
        return y
def funct5(x):
        y=x*math.sin(x)
        return y
def funct6(x):
        y=math.sqrt(1-x*x)
        return y
def funct7(x):
        y=math.log(1-8*math.cos(x)+16)
        return y

def Put_Clipboard(Sir_Clip):
        win32clipboard.OpenClipboard() # Open clipboard
        win32clipboard.EmptyClipboard() # Empty clipboard
        win32clipboard.SetClipboardData(win32con.CF_TEXT, Sir_Clip) # Set clipboard data
        win32clipboard.CloseClipboard()  # Close clipboard

def  Gauss(a, b, FUNCT):
        ag = [None] * 17  ;  xg = [None] * 17
        ag[9] =0.18945061 ; ag[10]=0.18260341 ; ag[11]=0.16915651 ; ag[12]=0.14959598
        ag[13]=0.12462897 ; ag[14]=0.09515851 ; ag[15]=0.06225352 ; ag[16]=0.02715245

        xg[9] =0.09501250 ; xg[10]=0.28160355 ; xg[11]=0.45801677 ; xg[12]=0.61787624
        xg[13]=0.75540440 ; xg[14]=0.86563120 ; xg[15]=0.94457502 ; xg[16]=0.98940093

        for i in range(1, 9):
                ag[i]=ag[17-i]
                xg[i]=-xg[17-i]
        h1=(b-a)/2; h2=(b+a)/2; suma_Gauss=0
        x=[]  ;  y=[]
        for i in range(1, 17):
                xi=h1*xg[i]+h2
                suma_Gauss=suma_Gauss+ag[i]*FUNCT(xi)
                x.append(xi)  ;  y.append(FUNCT(xi))
        suma_Gauss=suma_Gauss*h1
        lst_return=[suma_Gauss, x, y]
        return lst_return

class Plot_Window(wx.Frame):
        def __init__(self, parent, title):
                wx.Frame.__init__(self, parent, title=title, size=(650,600),
                        style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
                        wx.MAXIMIZE_BOX )) # Initialize the frame
                self.panel = wx.Panel(self)  #  The panel will hold the contents of the frame
```

Listing 5.1 Gauss.pyw – Integration by Gauss method – Python & wxPython version

```
        self.statusbar = self.CreateStatusBar() # Create the status bar
        self.statusbar.SetFieldsCount(1)
        self.statusbar.SetStatusText("This is the plot window",0)  # Add text to StatusBar
        self._TOOLBAR() # Create toolbar
        self.figure = mpl.figure.Figure()  # Initializes top level container for all plot elements
        self.axes=self.figure.add_subplot(111) # Add only one subplot area
        self.Center()  # Center the frame in display

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour("white")
        self.toolbar.SetToolBitmapSize((24,40))

        self.cb_ImageList = wx.combo.BitmapComboBox(self.toolbar, size=(220,-1))
        self.cb_ImageList.SetToolTipString("Select function to integrate !")
        self.Bind(wx.EVT_COMBOBOX, self.OnFunction,self.cb_ImageList)
        c=0 ; fct=["Functia1a","Functia1b"]
        for i in range(2,8):
            fct.append("Functia"+str(i))
        for filename in os.listdir(os.getcwd()):
          if filename.endswith(".jpg") :
                img = wx.Image(filename , wx.BITMAP_TYPE_ANY)  # load the image from the file
                sel_bmp=wx.BitmapFromImage(img.Scale(140,40))
                self.cb_ImageList.Append('%s' % fct[c], sel_bmp)
                c=c+1

        st1 = wx.StaticText(self.toolbar, -1, " Limit A value = ")
        self.txt_Lim_A = wx.TextCtrl(self.toolbar, value="0", size=(90,-1))
        st2 = wx.StaticText(self.toolbar, -1, " Limit B value = ")
        self.txt_Lim_B = wx.TextCtrl(self.toolbar, value="1", size=(90,-1))
        self.btn = wx.Button(self.toolbar, -1, "Calculate")
        self.btn.Bind(wx.EVT_BUTTON,self.OnCalculate)

        self.toolbar.AddControl(self.cb_ImageList )
        self.toolbar.AddControl(st1)
        self.toolbar.AddControl(self.txt_Lim_A)
        self.toolbar.AddControl(st2)
        self.toolbar.AddControl(self.txt_Lim_B)

        self.toolbar.AddControl(self.btn )
        self.toolbar.Realize()     # Toolbar generation
        self.toolbar.Show()        # Toolbar show
    def OnFunction(self, event):
        selectia=self.cb_ImageList.GetValue().strip()
        if selectia=="Functia1a" :
            self.FUNCT=funct1a  ;  self.txt_Lim_A.SetValue(str(1))  ;  self.txt_Lim_B.SetValue(str(3))
        elif selectia=="Functia1b" :
            self.FUNCT=funct1b  ;  self.txt_Lim_A.SetValue(str(1))  ;  self.txt_Lim_B.SetValue(str(50))
        elif selectia=="Functia2" :
            self.FUNCT=funct2  ;  self.txt_Lim_A.SetValue(str(0))  ;  self.txt_Lim_B.SetValue(str(0.9999))
        elif selectia=="Functia3" :
            self.FUNCT=funct3  ;self.txt_Lim_A.SetValue(str(0))  ;  self.txt_Lim_B.SetValue(str(1))
        elif selectia=="Functia4" :
            self.FUNCT=funct4  ;  self.txt_Lim_A.SetValue(str(0))  ;  self.txt_Lim_B.SetValue(str(math.pi/2))
        elif selectia=="Functia5" :
            self.FUNCT=funct5  ;  self.txt_Lim_A.SetValue(str(0))  ;  self.txt_Lim_B.SetValue(str(2*math.pi))
        elif selectia=="Functia6" :
            self.FUNCT=funct6  ; self.txt_Lim_A.SetValue(str(-0.9999))  ;  self.txt_Lim_B.SetValue(str(0.9999))
        else:
            self.FUNCT=funct7  ; self.txt_Lim_A.SetValue(str(0))  ;  self.txt_Lim_B.SetValue(str(math.pi))
    def MouseMotion(self, event):
        a1,b1=self.axes.transData.inverted().transform([event.x, event.y])
        self.statusbar.SetStatusText("X="+'%0.3f' % a1+",  Y="+'%0.3f' % b1,0)
    def Recreate_Axis(self):
        self.axes.cla()
        self.axes.grid(True)
        self.axes.set_xlabel('X', fontsize=20, fontweight='bold')
        self.axes.set_ylabel('Y', fontsize=20, fontweight='bold')
    def OnCalculate(self, event):
        a = float(self.txt_Lim_A.GetValue())
        b = float(self.txt_Lim_B.GetValue())
        self.canvas = FigureCanvas(self.panel,-1,self.figure)
```

Listing 5.1 Gauss.pyw – Integration by Gauss method – Python & wxPython version

```
        self.axes.figure.canvas.mpl_connect('motion_notify_event', self.MouseMotion)
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas, 1, wx.TOP | wx.LEFT | wx.EXPAND)
        self.Recreate_Axis()
        NrPct=100
        pas=(b-a)/NrPct
        xi=a ; X=[a] ; Y=[self.FUNCT(a)]
        for i in range(1, NrPct+1):
            xi=xi+pas
            X.append(xi)
            Y.append(self.FUNCT(xi))
        lx=len(X)-1
        for i in range(lx-1):
            self.axes.plot((X[i],X[i+1]),(Y[i],Y[i+1]), '-', color="Blue",linewidth=1.0)
        self.axes.plot((X[lx-1],X[lx]),(self.FUNCT(X[lx-1]),self.FUNCT(X[lx])), '-', color="Blue",linewidth=1.0)

        LST_RETURN= Gauss(a,b,self.FUNCT)
        Int_Gauss=LST_RETURN[0]
        sel=self.cb_ImageList.GetValue().strip()
        self.statusbar.SetStatusText("Value of the Gauss integral = "+str(Int_Gauss)+" for "+sel,0)
        self.axes.set_title("Value of the Gauss integral = "+str(Int_Gauss)+" for "+sel, fontsize=12)
        Put_Clipboard(str(Int_Gauss))

        xgauss=LST_RETURN[1]  ;  ygauss=LST_RETURN[2]
        for i, ( xplot, yplot) in enumerate(zip(xgauss, ygauss)):
            self.axes.plot(xplot,yplot, 'o', color="Red", markersize=4)
        self.axes.figure.canvas.draw()

        self.SetSizer(sizer)
        self.Fit()

app = wx.App(redirect=0)
window=Plot_Window(None, "Integration by Gauss method")
window.Show()
app.MainLoop()
```

The screen generated by Python & wxPython version of the **Gauss.pyw** script for the functions from **table 5.1** are displayed in the following figures: **Figure 5.1** for **function 1a**, **Figure 5.2** for **function 1b**, **Figure 5.3** for **function 2**, **Figure 5.4** for **function 3**, **Figure 5.5** for **function 4**, **Figure 5.6** for **function 5**, **Figure 5.7** for **function 6**, **Figure 5.8** for **function 7**. In this figures the blue curve is the function curve and the red points are the Gauss integration points.

The **Gauss.py** script (Python & Streamlit version) include the 9 public functions (**funct1a**, **funct1b**, **funct2** … , **funct7**, **Gauss**). The script begin with importing libraries: **streamlit** (create a great interactive web application with Python), **PIL** (Python Imaging Library), **matplotlib** (library for creating static, animated and interactive visualizations in Python), **math** (module which provides access to the mathematical functions), **pandas** (open source data analysis and manipulation tool) and **pathlib** (provide classes to represent abstract/concrete paths), This script contains 123 lines.

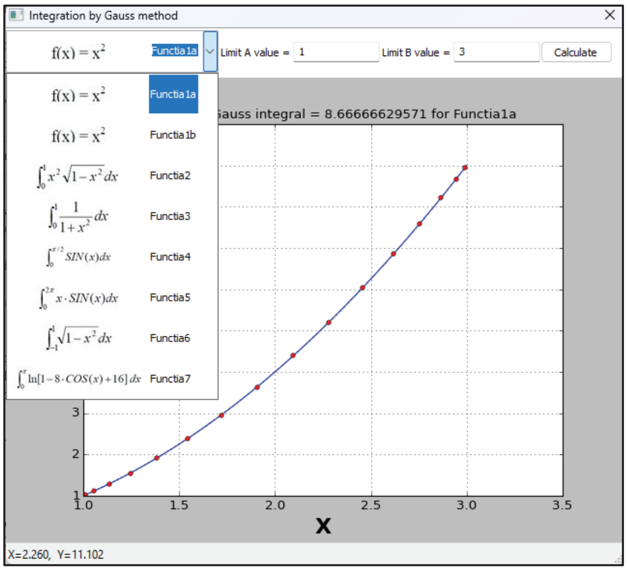
The public functions **funct1a**, **funct1b**, **funct2** … , **funct7** define the tested function according to **table 5.1**.

The **Gauss** function (lines 32÷50) is identical with the same function from **Gauss.pyw** (Python & wxPython version) script.

Line 52 define the title with **subheader** command and line 53 define the current script folder. Lines 55 and 61 create 4 column areas to place the images of the tested function through 56÷59 and 62÷65 lines. Line 67 create a list of function names used in line 70 to create a **selectbox** control from where the function name can be selected. Based on selected function from the **selectbox** control (line 70) the lines 73÷88 set the function name in **FUNCT** variable and store the values of integration limits in **AA** & **BB** variables.

The form created in line 89, wait to confirm or set the value of integration limits (lines 91÷92). Lines 95÷101 calculate 100 points of the function curve. Line 103 call the **Gauss** public function; lines 108÷116 draw the function curve with blue color, while lines 118÷120 draw the intergation Gausus points with red color.

Figure 5.9 show the screen generated by Python & Streamlit version of the **Gauss.py** script.


Integration by Gauss method

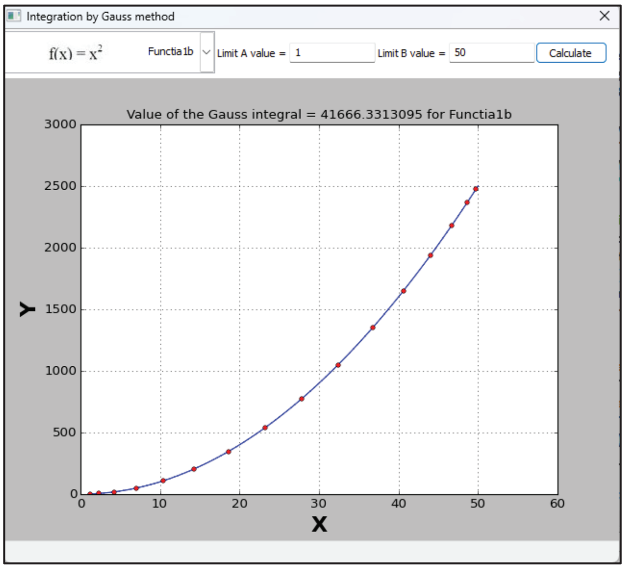
$f(x) = x^2$ Functia1a | Limit A value = 1 | Limit B value = 3 | Calculate

auss integral = 8.66666629571 for Functia1a

X=2.260, Y=11.102

$f(x) = x^2$ Functia1a
$f(x) = x^2$ Functia1b
$\int_0^1 x^2\sqrt{1-x^2}\,dx$ Functia2
$\int_0^1 \frac{1}{1+x^2}dx$ Functia3
$\int_0^{\pi/2} SIN(x)dx$ Functia4
$\int_0^{2\pi} x\cdot SIN(x)dx$ Functia5
$\int_{-1}^1 \sqrt{1-x^2}\,dx$ Functia6
$\int_0^{\pi} \ln[1-8\cdot COS(x)+16]\,dx$ Functia7
1
2
3
1.0
1.5
2.0
2.5
3.0
3.5
X

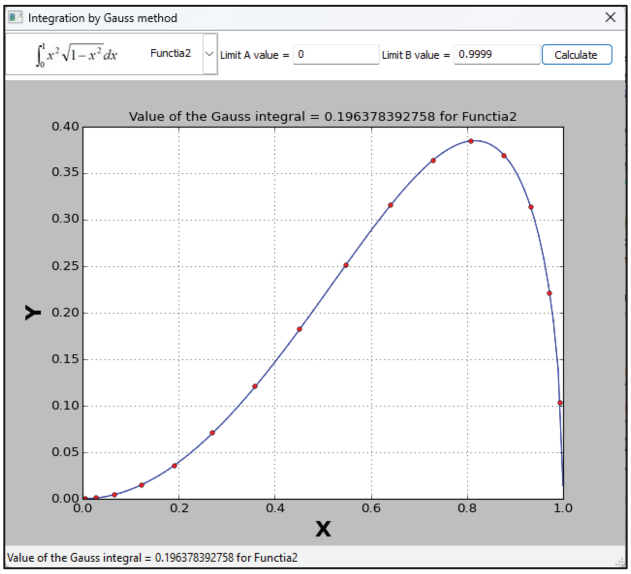
Figure 5.1 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 1a**


Integration by Gauss method

$f(x) = x^2$ Functia1b | Limit A value = 1 | Limit B value = 50 | Calculate

Value of the Gauss integral = 41666.3313095 for Functia1b

Figure 5.2 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 1b**

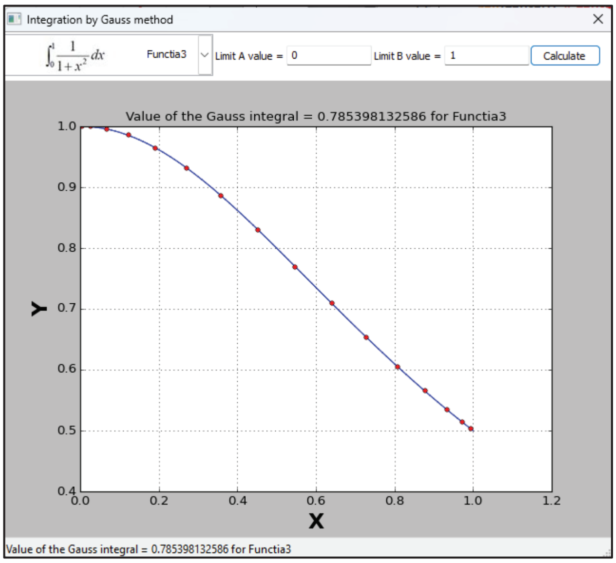

Integration by Gauss method

$\int_0^1 x^2\sqrt{1-x^2}\,dx$ Functia2 | Limit A value = 0 | Limit B value = 0.9999 | Calculate

Value of the Gauss integral = 0.196378392758 for Functia2

Value of the Gauss integral = 0.196378392758 for Functia2

Figure 5.3 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 2**

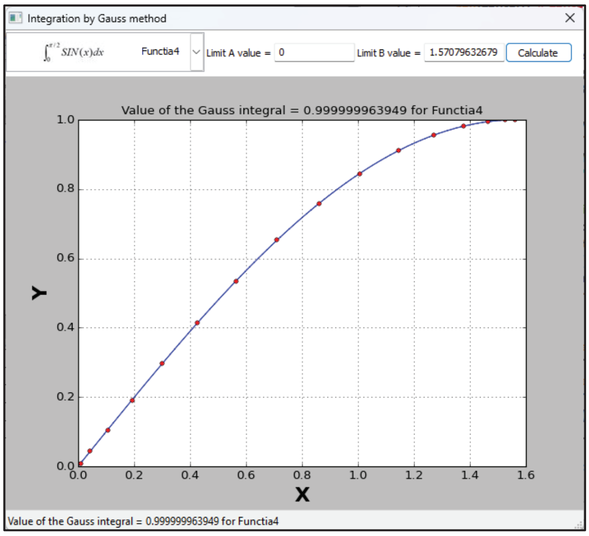

Integration by Gauss method

$\int_0^1 \frac{1}{1+x^2}dx$ Functia3 | Limit A value = 0 | Limit B value = 1 | Calculate

Value of the Gauss integral = 0.785398132586 for Functia3

Value of the Gauss integral = 0.785398132586 for Functia3

Figure 5.4 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 3**

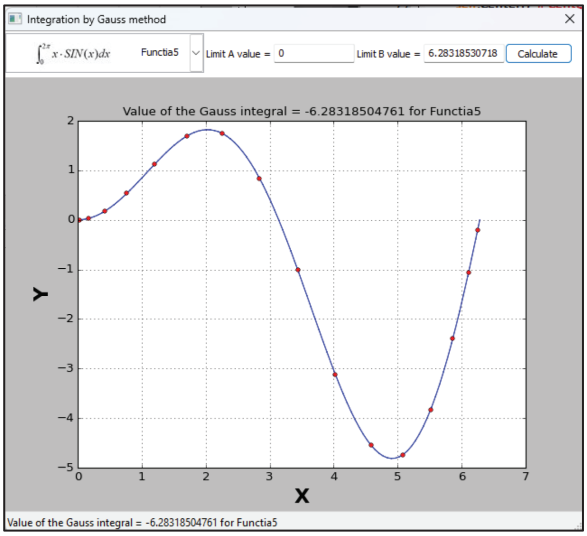

Integration by Gauss method

$\int_0^{\pi/2} SIN(x)dx$ Functia4 | Limit A value = 0 | Limit B value = 1.57079632679 | Calculate

Value of the Gauss integral = 0.999999963949 for Functia4

Value of the Gauss integral = 0.999999963949 for Functia4

Figure 5.5 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 4**

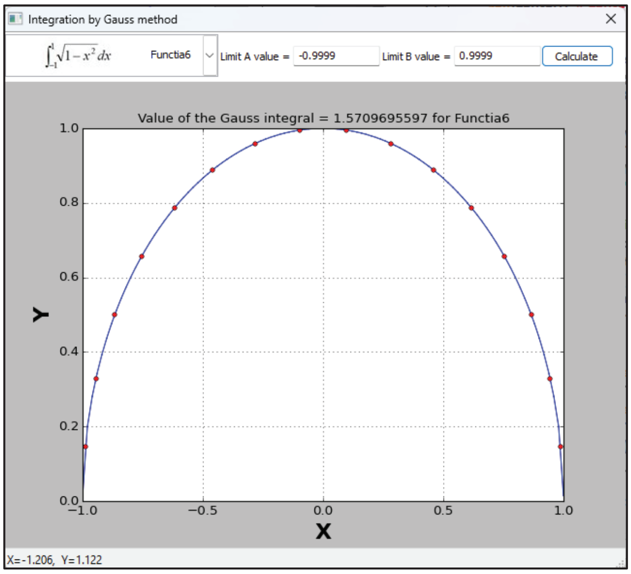

Integration by Gauss method

$\int_0^{2\pi} x\cdot SIN(x)dx$ Functia5 | Limit A value = 0 | Limit B value = 6.28318530718 | Calculate

Value of the Gauss integral = -6.28318504761 for Functia5

Value of the Gauss integral = -6.28318504761 for Functia5

Figure 5.6 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 5**

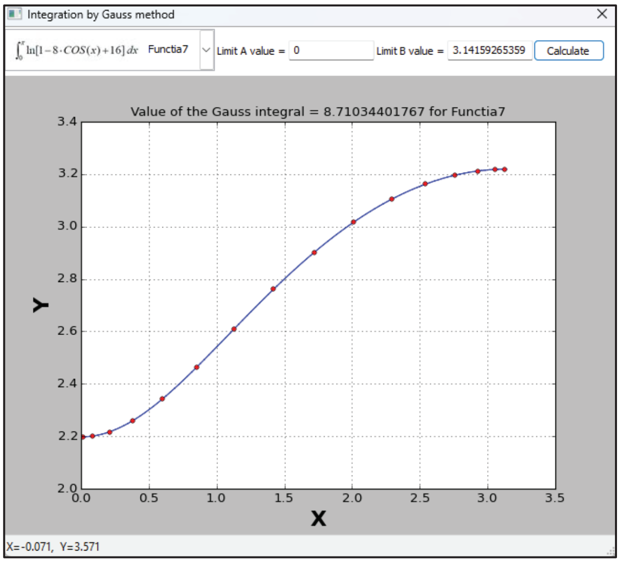

Integration by Gauss method

Limit A value = -0.9999

Limit B value = 0.9999

Value of the Gauss integral = 1.5709695597 for Functia6

X=-1.206, Y=1.122


Integration by Gauss method

Limit A value = 0

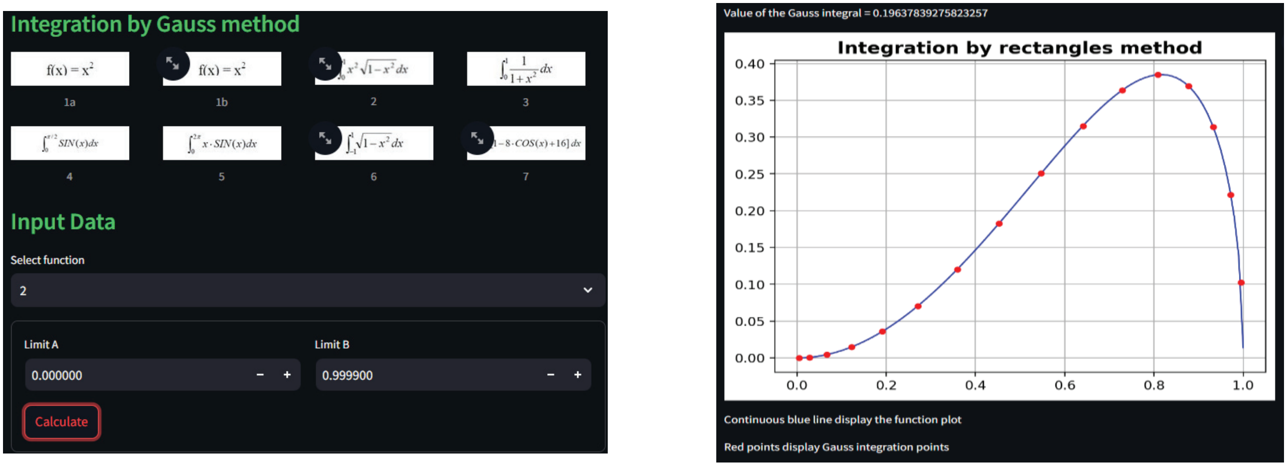
Limit B value = 3.14159265359

Value of the Gauss integral = 8.71034401767 for Functia7

X=-0.071, Y=3.571

Figure 5.7 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 6**

Figure 5.8 The screen generated by Python & wxPython version of the **Gauss.pyw** script - **function 7**


Integration by Gauss method

$f(x)=x^2$
1a
$f(x)=x^2$
1b
$\int_0^1 x^2\sqrt{1-x^2}\,dx$
2
$\int_0^1 \frac{1}{1+x^2}\,dx$
3
$\int_0^{\pi/2} SIN(x)dx$
4
$\int_0^{2\pi} x\cdot SIN(x)dx$
5
$\int_{-1}^{1}\sqrt{1-x^2}\,dx$
6
$1-8\cdot COS(x)+16]\,dx$
7
Calculate
Integration by rectangles method
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0.0
0.2
0.4
0.6
0.8
1.0

Input Data

Select function

2

Limit A

0.000000

Limit B

0.999900

Value of the Gauss integral = 0.19637839275823257

Continuous blue line display the function plot

Red points display Gauss integration points

Figure 5.9 The screen generated by Python & Streamlit version of the **Gauss.py** script for **function 2**

Listing 5.2 Gauss.py - Integration by Gauss method – Python & Streamlit version

```
import streamlit as st
from PIL import Image
import matplotlib.pyplot as plt
import math
from pathlib import Path

def funct1a(x):
        y=x*x
        return y
def funct1b(x):
        y=x*x
        return y
def funct2(x):
        y=x*x*math.sqrt(1-x*x)
        return y
def funct3(x):
        y=1/(1+x*x)
        return y
def funct4(x):
        y=math.sin(x)
        return y
def funct5(x):
        y=x*math.sin(x)
        return y
def funct6(x):
```

Listing 5.2 Gauss.py - Integration by Gauss method – Python & Streamlit version

```
        y=math.sqrt(1-x*x)
        return y
def funct7(x):
        y=math.log(1-8*math.cos(x)+16)
        return y

def  Gauss(a, b):
        ag = [None] * 17  ;  xg = [None] * 17
        ag[9] =0.18945061 ; ag[10]=0.18260341 ; ag[11]=0.16915651 ; ag[12]=0.14959598
        ag[13]=0.12462897 ; ag[14]=0.09515851 ; ag[15]=0.06225352 ; ag[16]=0.02715245

        xg[9] =0.09501250 ; xg[10]=0.28160355 ; xg[11]=0.45801677 ; xg[12]=0.61787624
        xg[13]=0.75540440 ; xg[14]=0.86563120 ; xg[15]=0.94457502 ; xg[16]=0.98940093
        for i in range(1, 9):
                ag[i]=ag[17-i]
                xg[i]=-xg[17-i];
        h1=(b-a)/2; h2=(b+a)/2; suma_Gauss=0
        x=[]  ;  y=[]
        for i in range(1, 17):
                xi=h1*xg[i]+h2
                suma_Gauss=suma_Gauss+ag[i]*FUNCT(xi)
                x.append(xi)  ;  y.append(FUNCT(xi))
        suma_Gauss=suma_Gauss*h1
        lst_return=[suma_Gauss, x, y]
        return lst_return

st.subheader(":green[Integration by Gauss method]")
current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()

col1, col2, col3, col4 = st.columns(4)
col1.image(Image.open(str(current_dir)+"/"+"Functia1a.jpg"), width=140, caption='1a')
col2.image(Image.open(str(current_dir)+"/"+"Functia1b.jpg"), width=140, caption='1b')
col3.image(Image.open(str(current_dir)+"/"+"Functia2.jpg"), width=140, caption='2')
col4.image(Image.open(str(current_dir)+"/"+"Functia3.jpg"), width=140, caption='3')

col1, col2, col3, col4 = st.columns(4)
col1.image(Image.open(str(current_dir)+"/"+"Functia4.jpg"), width=140, caption='4')
col2.image(Image.open(str(current_dir)+"/"+"Functia5.jpg"), width=140, caption='5')
col3.image(Image.open(str(current_dir)+"/"+"Functia6.jpg"), width=140, caption='6')
col4.image(Image.open(str(current_dir)+"/"+"Functia7.jpg"), width=140, caption='7')

lst_Functions=["Select function","1a","1b","2","3","4","5","6","7"]
st.subheader(':green[Input Data]')

selectia=st.selectbox("Select function ",options=lst_Functions)

if selectia != "Select function":
    if selectia=="1a" :
        FUNCT=funct1a  ;  AA=1.  ;  BB=3.
    elif selectia=="1b" :
        FUNCT=funct1b  ;  AA=1.  ;  BB=50.
    elif selectia=="2" :
        FUNCT=funct2  ;  AA=0.  ;  BB=0.9999
    elif selectia=="3" :
        FUNCT=funct3  ;               AA=0.  ;  BB=1.
    elif selectia=="4" :
        FUNCT=funct4  ;  AA=0.  ;  BB=math.pi/2
    elif selectia=="5" :
        FUNCT=funct5  ;  AA=0.  ;  BB=2*math.pi
    elif selectia=="6" :
        FUNCT=funct6  ; AA=-0.9999  ;  BB=0.9999
    else:
        FUNCT=funct7  ; AA=0.  ;  BB=math.pi
    with st.form('Input Data'):
        col1, col2 = st.columns(2)
        a = col1.number_input(label='Limit A ',value=AA,format="%.6f")
        b = col2.number_input(label='Limit B ',value=BB,format="%.6f")
        submit_button = st.form_submit_button('Calculate')
    if submit_button:
        NrPct=100
        pas=(b-a)/NrPct
        xi=a ; X=[a] ; Y=[FUNCT(a)]
```

Listing 5.2 Gauss.py - Integration by Gauss method – Python & Streamlit version

```
    for i in range(1, NrPct+1):
        xi=xi+pas
        X.append(xi)
        Y.append(FUNCT(xi))

    LST_RETURN= Gauss(a,b)
    Int_Gauss=LST_RETURN[0]
    st.write("Value of the Gauss integral = "+str(Int_Gauss))

    # Chart drawing
    fig=plt.figure()
    plt.grid(True)
    plt.title("Integration by rectangles method", fontsize=14,
        fontweight='bold',color='Black')
    lx=len(X)-1
    for i in range(lx-1):
        plt.plot((X[i],X[i+1]),(Y[i],Y[i+1]), '-', color="Blue",linewidth=1.0)
        print (i, X[lx-1])
    plt.plot((X[lx-1],X[lx]),(FUNCT(X[lx-1]),FUNCT(X[lx])), '-', color="Blue",linewidth=1.0)

    xgauss=LST_RETURN[1]  ;  ygauss=LST_RETURN[2]
    for i, ( xplot, yplot) in enumerate(zip(xgauss, ygauss)):
        plt.plot(xplot,yplot, 'o', color="Red", markersize=4)
    st.empty().write(fig)
    st.write("Continuous blue line display the function plot")
    st.write("Red points display Gauss integration points")
```

6. NACA hydrodynamic profile

This script is designed to draw a NACA profile defined through coordinates readed from an Excel file using wx.FileDialog control; then, in the same Excel file, the chart drawing of the NACA profile is exported. A new way to define toolbar with icons is used. Also, the profile coordinates are shown in an wx.lib.dialogs.ScrolledMessageDialog control and profile drawing will be saved as image file. The input data are the Excel file from which the profile coordinates are readed; the results of this script are the profile drawing exported to Excel file, the displayed of the profile coordinates and the saved image file of the profile.

Listing 6.1 displays the Python & wxPython version of script, NACA.pyw and Listing 6.2 displays the Python & Streamlit version of the same script, NACA.py. Figure 6.1 display the screens generated by Python & wxPython version of the script, while Figure 6.2 display the screens generated by Python & Streamlit version.

The NACA.pyw script (Python & wxPython version) include only one class: Plot_ Window. The script begin with importing libraries: wx (the cross-platform GUI toolkit for the Python language), os (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.), StringIO (the StringIO module is used to implement basic tasks on file-like objects), cStringIO (an optional module, which contains a faster implementation of the StringIO module), from win32com.client import Dispatch (the win32com. client package contains a number of modules to provide access to automation objects) and matplotlib (library for creating static, animated and interactive visualizations in Python). To use win32com.client the pywin32-214.win32-py2.7.exe file must be installed for python version 2.7; this will be used to connect with an Excel or Word file.

The Plot_ Window class create the interface, draw the profile and include the following functions:

- __init__ — Define the frame (a window whose size and position can be changed by the user), the panel (that will hold the contents of the frame) the StatusBar (placed at the frame bottom, were messages can be displayed); call the _TOOLBAR function, initializes top level container for all plot elements (the chart area self.figure & self.axes) and center the frame in the display.
- _TOOLBAR — Create the toolbar (lines 33÷35); create the toolbar icon based on open.png image (lines 37÷40) and connect with Bind statement (line 41) to OnOpenExcelFile function which open the Excel file with profile coordinates using wx.FileDialog; create the toolbar icon based on table.png image (lines 43÷47) and connect with Bind statement (line 48) to OnViewCoordinates function which display profile coordinates using wx.lib.dialogs.ScrolledMessageDialog; create the toolbar icon based on save.png image (lines 50÷53) and connect with Bind statement (line 54) to OnSave function which save the chart figure; create the toolbar icon based on Excel.png image (lines 56÷59) and connect with Bind statement (line 54) to OnExportExcel function which export the chart figure to Excel file; create the toolbar icon based on exit.png image (lines 62÷65) and connect with Bind statement (line 67) to OnExit function to exit from script; lines 68÷69 generate and show the ToolBar.
- OnOpenExcelFile — This function is called by toolbar icon. Through this function the user can select an Excel file with profile coordinates using wx.FileDialog dialog (lines 71÷74), Figure 6.1a. The Excel file name is intialized in line 30 and stored in the same variable self.Excel_File, line 75; lines 76÷79 connect and open the Excel file and define the sheet sht1 variable from where the coordinates X, Y are readed in self.X & self.Y lists (line 87) up to an empty empty cell (tested in line 84); line 88 store in self.XY list both X and Y values; this list variable will be used to show the coordinates in functiion OnViewCoordinates; finally, the Excel file is closed without saving and function DrawProfile is called.
- DrawProfile — This function create the canvas and drawing self.axes (lines 134÷136); create the sizer (line 137) and place the canvas inside the sizer (line 138);); line 139 call Recreate_Axis function;

lines 140÷143 plot the profile from **Figure 6.1b**; the chart is updated by the **draw** statement (line 144); line 145 display in **StatusBar** the name of the Excel file.

- **OnSave** This function is called by toolbar icon to save the profile drawing from **Figure 6.1b** as image file '**Profile.jpg**' (lines 94÷95); line 96 show an info message about the file name and path where the image is saved, using **wx.MessageBox** command.
- **OnViewCoordinates** This function is called by toolbar icon to show the profile coordinates readed from Excel file, **Figure 6.1c**.
- **OnExportExcel** This function is called by toolbar icon to export the profile drawing to the the same Excel file from where the profile coordinates where readed, **Figure 6.1d**; lines 105÷107 connect and open the Excel file and define the sheet **XLSheet** variable; line 110 write the text marked by quotation marks into cell line 1 column 4; lines 112÷113 save the drawing into '**Exported_Image.jpg**' image file, open by 114 line and converted to bitmap by 115÷116 lines; lines 118÷119 define the position (column **left** and line **top**) where the image will be placed in Excel file; the image is placed in Excel sheet by 120 command line and saved by 122 line.
- **MouseMotion** Display in the **StatusBar** the current values of X & Y mouse position in the chart.
- **Recreate_Axis** Initialize the chart (line 129), show the chart grid (line 130) and set the labels for X & Y axes of the chart (lines 131÷132).
- **OnExit** This function is called by toolbar icon; if the exit step is confirmed as a response of **wx.MessageDialog** (lines 149÷152), the main loop is stopped by line 153.

Listing 6.1 NACA.pyw – NACA profile – Python & wxPython version

```
1.  # The // operator will be available to request floor division unambiguously. This
2.  # statement will change the / operator to mean true division throughout the module.
3.  from __future__ import division
4.
5.  import wx    # import wx module
6.  import  wx.lib.dialogs
7.  import os
8.  import StringIO, cStringIO
9.
10. from win32com.client import Dispatch
11. import matplotlib as mpl  # The tools are imported from the Matplotlib library
12. import matplotlib.pyplot as plt
13. import matplotlib.image as img
14. from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
15. from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar
16.
17. class Plot_Window(wx.Frame):
18.         def __init__(self, parent, title):
19.                 wx.Frame.__init__(self, parent, title=title, size=(650,600),
20.                         style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
21.                         wx.MAXIMIZE_BOX )) # Initialize the frame
22.                 self.panel = wx.Panel(self)  #  The panel will hold the contents of the frame
23.                 self.statusbar = self.CreateStatusBar() # Create the status bar
24.                 self.statusbar.SetFieldsCount(2)
25.                 self.statusbar.SetStatusWidths([-15, -60])
26.                 self.statusbar.SetStatusText("This is the plot window",0)  # Add text to StatusBar
27.                 self._TOOLBAR() # Create toolbar
28.                 self.figure = mpl.figure.Figure()  # Initializes top level container for all plot elements
29.                 self.axes=self.figure.add_subplot(111) # Add only one subplot area
30.                 self.XY=""   ;  self.Excel_File=""
31.                 self.Center()  # Center the frame in display
32.         def _TOOLBAR(self):
33.                 self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
34.                 self.toolbar.SetBackgroundColour("white")
35.                 self.toolbar.SetToolBitmapSize((24,24))
36.
37.                 id_Open=wx.NewId()
38.                 img = wx.Image(os.getcwd()+'/open.png', wx.BITMAP_TYPE_ANY)
39.                 sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
40.                 self.toolbar.AddLabelTool( id_Open  , "Open", sel_bmp, shortHelp="Open Excel file to read NACA coordinates")
```

Listing 6.1 NACA.pyw – NACA profile – Python & wxPython version

```
        self.Bind(wx.EVT_TOOL, self.OnOpenExcelFile, id=id_Open)

        id_ViewCoordinates=wx.NewId()
        img = wx.Image(os.getcwd()+'/table.png', wx.BITMAP_TYPE_ANY)
        sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
        self.toolbar.AddLabelTool( id_ViewCoordinates  , "View", sel_bmp, shortHelp="View the NACA profile coordinates
extracted from Excel")
        self.Bind(wx.EVT_TOOL, self.OnViewCoordinates, id=id_ViewCoordinates)

        id_Save=wx.NewId()
        img = wx.Image(os.getcwd()+'/save.png', wx.BITMAP_TYPE_ANY)
        sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
        self.toolbar.AddLabelTool( id_Save  , "Save", sel_bmp, shortHelp="Save profile to image file")
        self.Bind(wx.EVT_TOOL, self.OnSave, id=id_Save)

        id_Excel=wx.NewId()
        img = wx.Image(os.getcwd()+'/Excel.png', wx.BITMAP_TYPE_ANY)
        sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
        self.toolbar.AddLabelTool( id_Excel  , "Excel", sel_bmp, shortHelp="Save profile image to Excel file")
        self.Bind(wx.EVT_TOOL, self.OnExportExcel, id=id_Excel)

        id_Exit=wx.NewId()
        img = wx.Image(os.getcwd()+'/exit.png', wx.BITMAP_TYPE_ANY)
        sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
        self.toolbar.AddLabelTool( id_Exit  , "Exit", sel_bmp, shortHelp="Exit application")

        self.Bind(wx.EVT_TOOL, self.OnExit, id=id_Exit)
        self.toolbar.Realize()      # Toolbar generation
        self.toolbar.Show()         # Toolbar show
    def OnOpenExcelFile(self, event):
        wildcard="Excel files (*.xlsx)|*.xlsx|Excel files (*.xsl)|*.xls"
        dialog = wx.FileDialog(None, "Open Excel file to read NACA coordinates", "", "", wildcard, wx.OPEN)
        if dialog.ShowModal() == wx.ID_CANCEL:
            return
        self.Excel_File= dialog.GetPath().strip().upper()
        xlApp = Dispatch("Excel.Application")   # Connect to Excel
        xlApp.Visible=False
        xlWb = xlApp.Workbooks.Open(self.Excel_File)
        sht1 = xlWb.Worksheets("Data")
        lex=2  ;  self.X=[]  ;  self.Y=[]
        while True:
            X=str(sht1.Cells(lex,1).Value).strip()
            Y=str(sht1.Cells(lex,2).Value).strip()
            if X=="" or X=="None":
                break
            else:
                self.X.append(float(X))  ;  self.Y.append(float(Y))
                self.XY=self.XY+"X="+'%0.6f' % float(X)+",   Y="+'%0.6f' % float(Y)+"\n"
                lex+=1
        xlWb.Close(SaveChanges=0)
        xlApp.Quit()
        self.DrawProfile()
    def OnSave(self, event):
            img_file = os.getcwd()+'/Profile.jpg'
            self.axes.figure.savefig(img_file)
            wx.MessageBox("Image saved in "+img_file, "Info", wx.OK | wx.ICON_INFORMATION)
    def OnViewCoordinates(self, event):
        if len(self.XY)==0:
            wx.MessageBox("Open an Excel file to read profile coordinates !",
                  "ERROR", wx.OK | wx.ICON_INFORMATION)
            return
        dlg = wx.lib.dialogs.ScrolledMessageDialog(self, self.XY, "View Coordinates")
        dlg.ShowModal()
    def OnExportExcel(self, event):
        xlApp = Dispatch("Excel.Application")   # Conexiune Excel
        xlApp.Visible=False
        xlWb = xlApp.Workbooks.Open(self.Excel_File)
        XLSheet=xlWb.Worksheets("DATA")

        XLSheet.Cells(1,4).Value="Profile image  exported from NACA.pyw script"

        PathImage = os.getcwd()+'/Exported_Image.jpg'
```

Listing 6.1 NACA.pyw – NACA profile – Python & wxPython version

```
        self.axes.figure.savefig(PathImage)
        data = open(PathImage, "rb").read()
        stream = cStringIO.StringIO(data)  # Convert to a data stream  Require to import  cStringIO & StringIO
        sel_bmp=wx.BitmapFromImage( wx.ImageFromStream( stream ))  # convert to a bitmap
        required_width=sel_bmp.GetWidth() ; required_height=sel_bmp.GetHeight()
        left=XLSheet.Cells(1,4).Left
        top=XLSheet.Cells(2,1).Top
        XLSheet.Shapes.AddPicture(PathImage, False, True,left, top,required_width,required_height)
        os.remove(PathImage)
        xlWb.Close(SaveChanges=1)
        xlApp.Quit()
        wx.MessageBox("Image saved in "+os.getcwd()+'/'+self.Excel_File, "Info", wx.OK | wx.ICON_INFORMATION)
    def MouseMotion(self, event):
        a1,b1=self.axes.transData.inverted().transform([event.x, event.y])
        self.statusbar.SetStatusText("X="+'%0.3f' % a1+",  Y="+'%0.3f' % b1   ,0)
    def Recreate_Axis(self):
        self.axes.cla()
        self.axes.grid(True)
        self.axes.set_xlabel('X', fontsize=20, fontweight='bold')
        self.axes.set_ylabel('Y', fontsize=20, fontweight='bold')
    def DrawProfile(self):
        self.canvas = FigureCanvas(self.panel,-1,self.figure)
        self.axes.figure.canvas.mpl_connect('motion_notify_event', self.MouseMotion)
        self.axes.axis('equal')
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas, 1, wx.TOP | wx.LEFT | wx.EXPAND)
        self.Recreate_Axis()
        for i, ( xplot, yplot) in enumerate(zip(self.X, self.Y)):
            self.axes.plot(xplot,yplot, 'o', color="Red", markersize=3)
            if i<>len(self.X)-1:
                self.axes.plot((self.X[i],self.X[i+1]),(self.Y[i],self.Y[i+1]), '-', color="Blue",linewidth=1.0)
        self.axes.figure.canvas.draw()
        self.statusbar.SetStatusText(self.Excel_File,1)
        self.SetSizer(sizer)
        self.Fit()
    def OnExit(self, event):
        d = wx.MessageDialog(self, "Confirmati application exit ?", caption = "Confirm", style = wx.YES_NO |
wx.ICON_QUESTION)
        response = d.ShowModal()
        if response == wx.ID_YES:
            app.ExitMainLoop()

app = wx.App(redirect=False)   # wx.App initializes the GUI toolkit for WxPython.
window=Plot_Window(None, "NACA profile")  # Call 'Plot_Window' class
window.Show()  # Show 'window' object
# This keeps the GUI in a continuous loop that is ready to receive key events
# from the user. It does not return until the program closes!
app.MainLoop()
```

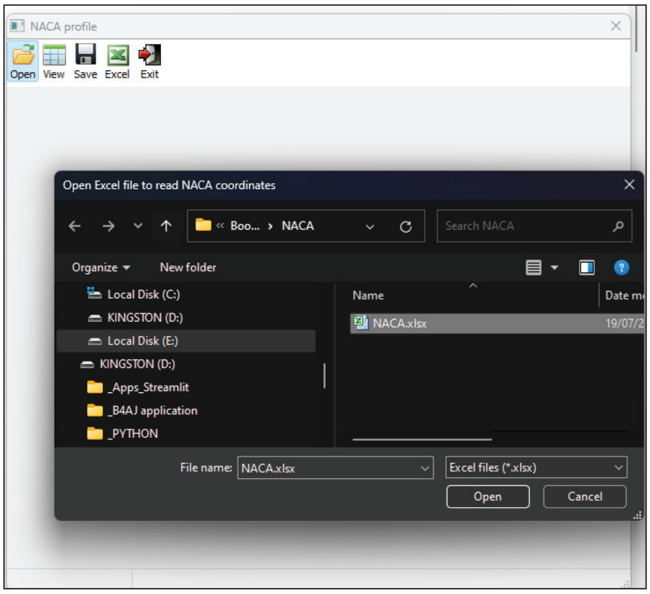
The **NACA.py** script (Python & Streamlit version) begin with importing libraries: **streamlit** (create a great interactive web application with Python), **pandas** (open source data analysis and manipulation tool), **matplotlib** (library for creating static, animated and interactive visualizations in Python), **pathlib** (provide classes to represent abstract/concrete paths) and **openpyxl** (Python library to read/write Excel files). This script contains 44 lines. Lines 1 to 6 import the required modules. Line 8 defines the current folder. Line 9 defines the script title with **subheader** command.

With line 11 the user can select the Excel file with profile coordinates; line 12 test if the file was uploaded; line 13 read the profile coordinates into **df** dataframe; line 14 show a table of coordinates; line 15 extract coordinates from dataframe to **X** list; line 16 extract coordinates from dataframe to **Y** list. Line 19 inserts a container element (with **st.empty()** command) into the script that can be used to hold the profile drawing created by lines 20÷29; line 30 define the path and name of future image file of the drawing, saved by line 33. Lines 37÷43 export, in the same Excel file uploaded in line 11, the image in D2 cell.

Figure 6.2 show the screens generated by Python & Streamlit version of the **NACA.py** script.

Listing 6.2 NACA.py – NACA profile – Python & Streamlit version

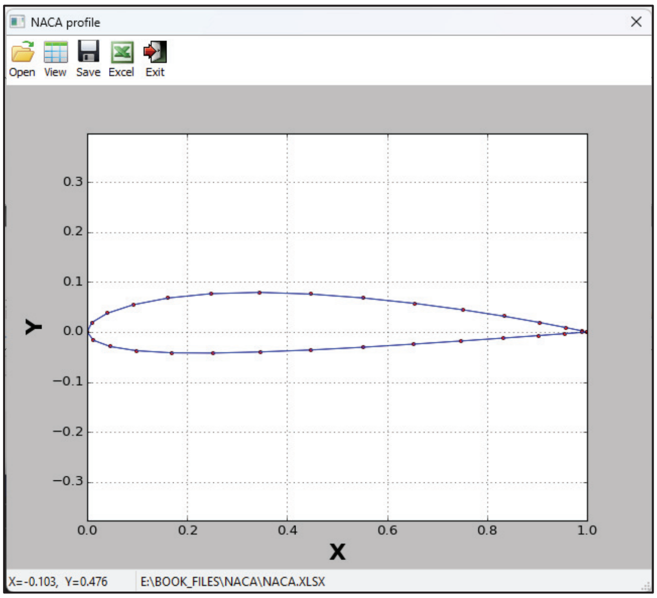
```
import streamlit as st
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path # This module provides classes to represent abstract/concrete paths
from openpyxl import load_workbook # Import Python library to read/write Excel files
from openpyxl.drawing.image import Image

current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()
st.subheader(":green[NACA profile]")
# Require to install openpyxl library with command 'pip install openpyxl'
uploaded_file = st.file_uploader("Choose a XLSX file with profile coordinates", type="xlsx")
if uploaded_file:
    df = pd.read_excel(uploaded_file)
    st.dataframe(df) # Display the table with profile coordinates
    X = df['X'].tolist() # Convert columns X from dataframe to list
    Y = df['Y'].tolist() # Convert columns Y from dataframe to list

    # Chart drawing
    Place_Chart = st.empty()
    fig=plt.figure() # Create chart figure
    plt.grid(True) # Show the chart grid
    plt.xlabel('X', fontsize=20, fontweight='bold')
    plt.ylabel('Y', fontsize=20, fontweight='bold')
    plt.axis('equal') # Set and adjust plots with equal axis aspect ratios
    plt.title("NACA profile", fontsize=14, fontweight='bold',color='Black') # Define chart title
    for i, ( xplot, yplot) in enumerate(zip(X, Y)):
        plt.plot(xplot,yplot, 'o', color="Red", markersize=3)
        if i != len(X)-1:
            plt.plot((X[i],X[i+1]),(Y[i],Y[i+1]), '-', color="Blue",linewidth=1.0)
    img_file = str(current_dir)+'/Profile.jpg'

    # Save image to an image file
    plt.savefig(img_file)
    st.write("Image saved in "+img_file)

    # Export image to Excel file
    wb = load_workbook(f"{uploaded_file.name}")
    ws = wb.active
    ws['D1'] = 'Profile image exported from NACA.py script'
    img=Image(img_file)
    ws.add_image(img,'D2')
    wb.save(f"{uploaded_file.name}")
    Place_Chart.write(fig)
    st.write("Image exported in Excel file "+f"{uploaded_file.name}")
```



File name: NACA.xlsx

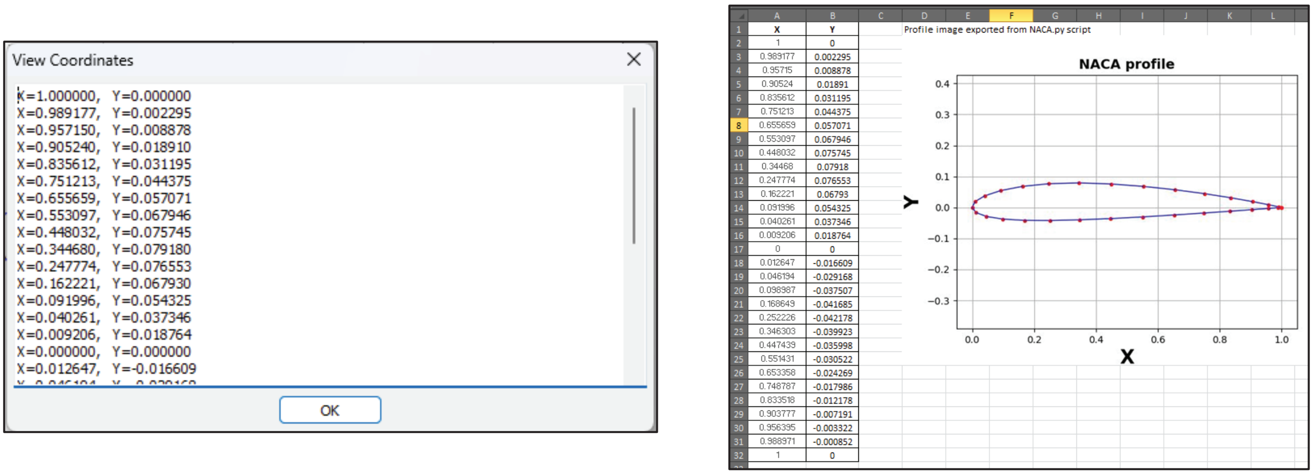
Excel files (*.xlsx)

Figure 6.1a



X=-0.103, Y=0.476 E:\BOOK_FILES\NACA\NACA.XLSX

Figure 6.1b

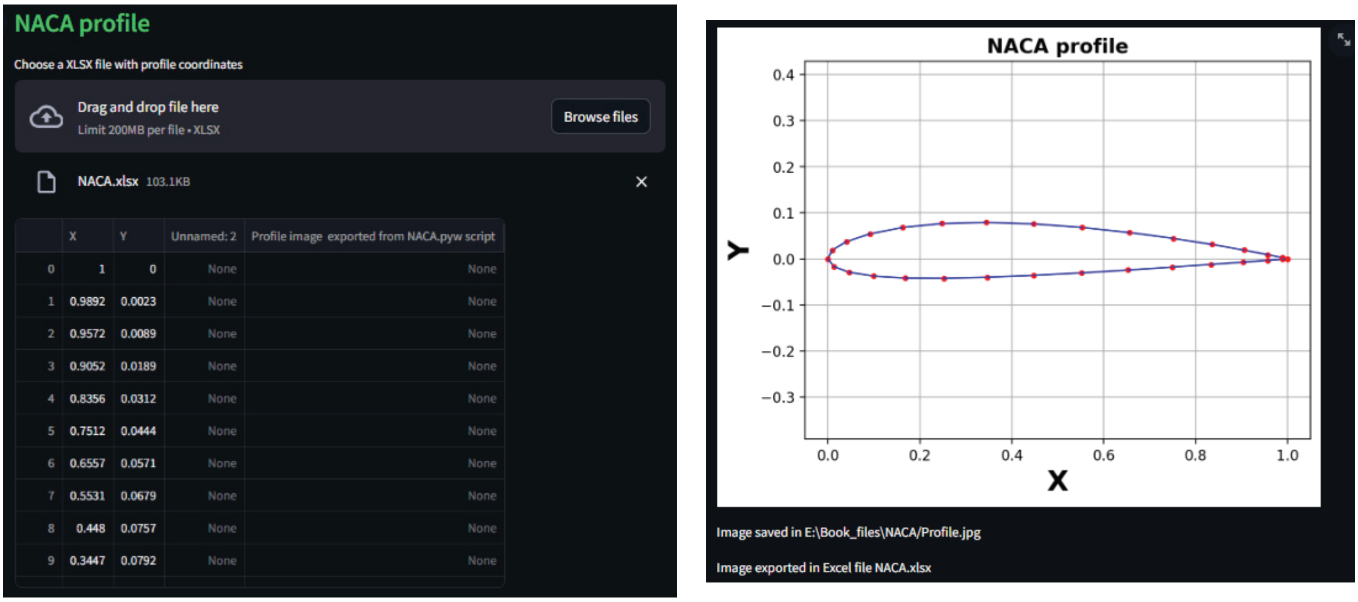


```
X=1.000000,  Y=0.000000
X=0.989177,  Y=0.002295
X=0.957150,  Y=0.008878
X=0.905240,  Y=0.018910
X=0.835612,  Y=0.031195
X=0.751213,  Y=0.044375
X=0.655659,  Y=0.057071
X=0.553097,  Y=0.067946
X=0.448032,  Y=0.075745
X=0.344680,  Y=0.079180
X=0.247774,  Y=0.076553
X=0.162221,  Y=0.067930
X=0.091996,  Y=0.054325
X=0.040261,  Y=0.037346
X=0.009206,  Y=0.018764
X=0.000000,  Y=0.000000
X=0.012647,  Y=-0.016609
```

	A	B	C	D
1	X	Y		Profile image exported from NACA.py script
2	1	0		
3	0.989177	0.002295		
4	0.95715	0.008878		
5	0.90524	0.01891		
6	0.835612	0.031195		
7	0.751213	0.044375		
8	0.655659	0.057071		
9	0.553097	0.067946		
10	0.448032	0.075745		
11	0.34468	0.07918		
12	0.247774	0.076553		
13	0.162221	0.06793		
14	0.091996	0.054325		
15	0.040261	0.037346		
16	0.009206	0.018764		
17	0	0		
18	0.012647	-0.016609		
19	0.046194	-0.029168		
20	0.098987	-0.037507		
21	0.168649	-0.041685		
22	0.252226	-0.042178		
23	0.346303	-0.039923		
24	0.447439	-0.035998		
25	0.551431	-0.030522		
26	0.653358	-0.024269		
27	0.748787	-0.017986		
28	0.833518	-0.012178		
29	0.903777	-0.007191		
30	0.956395	-0.003322		
31	0.988971	-0.000852		
32	1	0		

Figure 6.1c **Figure 6.1d**

Figure 6.1 The screens generated by Python & wxPython version of the **NACA.pyw** script



	X	Y	Unnamed: 2	Profile image exported from NACA.pyw script
0	1	0	None	None
1	0.9892	0.0023	None	None
2	0.9572	0.0089	None	None
3	0.9052	0.0189	None	None
4	0.8356	0.0312	None	None
5	0.7512	0.0444	None	None
6	0.6557	0.0571	None	None
7	0.5531	0.0679	None	None
8	0.448	0.0757	None	None
9	0.3447	0.0792	None	None

Image saved in E:\Book_files\NACA/Profile.jpg

Image exported in Excel file NACA.xlsx

Figure 6.2 The screens generated by Python & Streamlit version of the **NACA.py** script

7. Recursive functions

Recursion in programming is a special way of repeating groups of statements using functions that call themselves. Examples of recursive structures, where parts repeat the whole:

- In everyday life: a mirror image in a mirror or a screenshot showing the image of the camera itself;
- In mathematics: many definitions are recursive, e.g. the definition of a natural number (if n is a natural number, then n +1 is also a natural number);
- In nature: biological structures of trees and flowers, clouds, topographical structures, galaxies, etc.

Recursive functions are functions that call themselves. Program code written using recursive functions is shorter, but such functions require more time to execute.

The basic form of the definition of the recursive function is shown in **Figure 7.1**.

An example of a recursive function that does not return a result is the **countdown** function shown in **Figure 7.2**. The function prints to the screen all numbers from n, n-1, ...1. It calls itself recursively with a modified argument value. On the screen will be displayed the following numbers: 5, 4, 3, 2, 1.

```
def countdown (parameters):
    function block
    countdown(parameters):  # Self function call
```

Figure 7.1 The basic form of the recursive function

```
def countdown (n):
    print n
    if n > 1:
        countdown(n-1)  # Self function call
countdown(5)    # First function call
```

Figure 7.2 The **countdown** recursive function form

Vector graphics in Python is a way of creating graphical representations using basic geometric objects. A special approach to creating vector graphics is the so-called **turtle graphics**, where objects are drawn using relative coordinates and commands for drawing geometric objects that are issued to an imaginary pencil, the so-called **turtle**. The pen is displayed on the screen in the form of a small triangle that indicates the direction of drawing. Using such graphical commands, require to include the program library with the **import turtle** command.

Some of the basic functions of this library are:

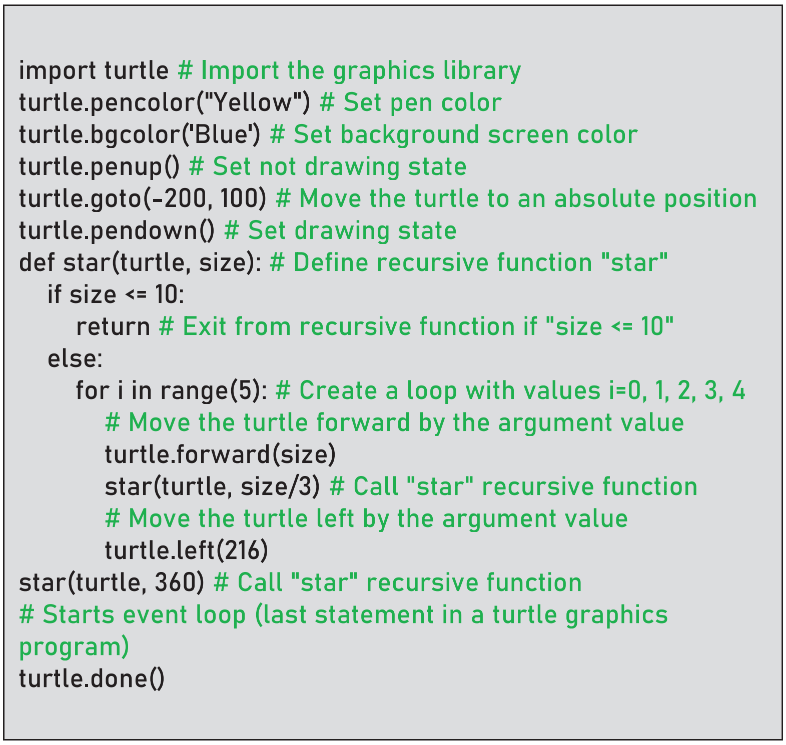
- **pendown()** - lowers the pen so that it draws a line as it moves;
- **penup()** - raises the pen so it doesn't leave a trail while moving, goto(x,y);
- **goto** - sets the (x,y) coordinates of the pen on the screen;
- **left** (corner), exactly (corner) - change of arrow direction for a given angle;
- **forward** (distance), back (distance) - moving the pen by a certain number of pixels in the direction of the arrow or the opposite direction;
- **pen** (color) - pen color adjustment;
- **fill color** (color) - adjusting the color of the interior of the figure, etc.

Drawing simple geometric figures is done by placing the pen on the selected starting point of the screen with coordinates (x,y) after which the lines are drawn by changing the direction and moving the pen in the appropriate direction by a certain number of pixels. If the initial position of the pen is not specified, the initial coordinates are assumed to be (0, 0). The script example from **Figure 7.3** illustrates the process of creating a star animation, the result of which is shown in **Figure 7.4**.

The next example compares two scripts (with nonrecursive and recursive functions) for generating a sequence of numbers based on the following algorithm: the following operations can be done with a natural number:

A) add the digit 4 at the end;
B) add the digit 0 at the end;
C) divide by 2 (if even).

For number N starting from the initial value 4 and applying the operations described above, to generate the number N and to display the transformation sequence intermediate transformations. Example: for the number N=5 the transformation sequence is: 4, 2, 1, 10, 5.



```
import turtle # Import the graphics library
turtle.pencolor("Yellow") # Set pen color
turtle.bgcolor('Blue') # Set background screen color
turtle.penup() # Set not drawing state
turtle.goto(-200, 100) # Move the turtle to an absolute position
turtle.pendown() # Set drawing state
def star(turtle, size): # Define recursive function "star"
    if size <= 10:
        return # Exit from recursive function if "size <= 10"
    else:
        for i in range(5): # Create a loop with values i=0, 1, 2, 3, 4
            # Move the turtle forward by the argument value
            turtle.forward(size)
            star(turtle, size/3) # Call "star" recursive function
            # Move the turtle left by the argument value
            turtle.left(216)
star(turtle, 360) # Call "star" recursive function
# Starts event loop (last statement in a turtle graphics
program)
turtle.done()
```

Figure 7.3 Creating a star animation with **turtle** library

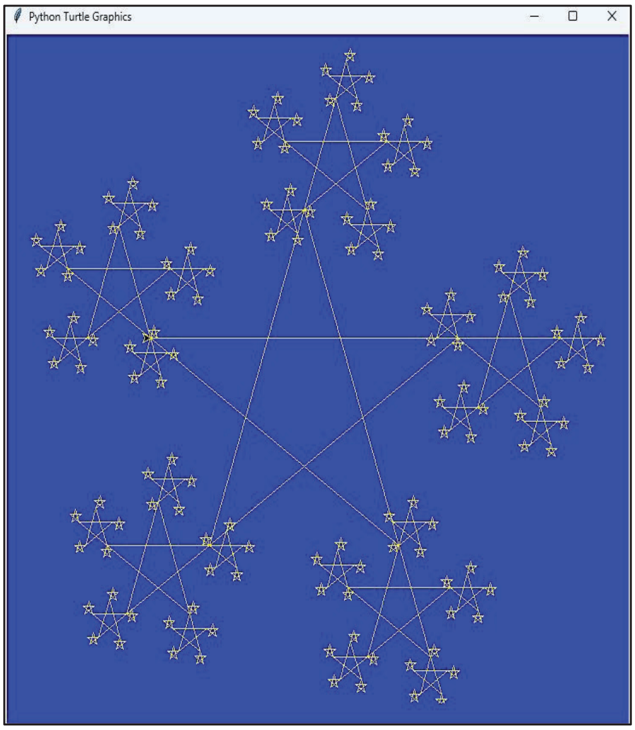


Figure 7.4 The final result of creating a star with **turtle** script

The problem does not pose any particular problems in terms of the operations to be applied, but generates uncertainties due to the choice of the type of operation to be applied (A or B), with implications for the multiple branching of the algorithm. For this reason, the solution relies on a method that provides a unique sequence generation: starting from the given number N, the inverse operations are applied, thus generating the sequence of inverse transformations; representing this sequence in reverse order gives the sequence that would have been obtained by applying the direct operations. The uniqueness of this solution lies in the deterministic nature of the operation to be applied, according to the following logic: if the last digit is 0 or 4, it is removed, otherwise the number is doubled. For example, starting from the number 5 and applying the inverse operations according to the above logic, the result is the sequence 5, 10, 1, 2, 4, which, displayed in reverse order, represents the sequence you are looking for.

Listing 7.1 displays the **Python** version of script **Generate_Sequence.pyw**. The script use two function: **NonRecursiv** which implement the nonrecursive algorithm and **Recursiv** function which implement the recursive algorithm. The results are identical, regardless of the used function. **Figure 7.5** exemplify the comparative graphical results of **Generate_Sequence.pyw** script.

Listing 7.1 Generate_Sequence.pyw – Sequence generated using an imposed algorithm – Python version

```
 import matplotlib.pyplot as plt
import numpy as np

def NonRecursiv(N):   # Nonrecursiv variant
    Lst=[N]
    while N<>4:
        x=str(N)     # Convert N to string
        if x[-1]=='0' or x[-1]=='4':
            x=x[:-1] # Remove the last string char
            N=int(x) # Convert string to N
        else:
            N=2*N    # Doubling N
        Lst.append(N)
    return Lst

def Recursiv(X): # Recursive variant
    x=str(X)
    if X==4:
```

Listing 7.1 Generate_Sequence.pyw – Sequence generated using an imposed algorithm – Python version

```
        return    # Exit from recursive calls
    elif x[-1]=='0' or x[-1]=='4':
        x=x[:-1] # Remove the last string char
        N=int(x) # Convert string to N
    else:
        N=2*X    # Doubling N
    Lst2.append(N)
    Recursiv(N)  # Self function call

# Main program ======================================
N=55
Lst1 = NonRecursiv(N) # Call nonrecursiv function
Lst2 =[N]
Recursiv(N)         # First recursive function call

fig, ax = plt.subplots()
plt.xlabel("Iteration", fontsize=16, fontweight='bold', color="Brown")
plt.ylabel("Generated numbers", fontsize=16, fontweight='bold', color="Brown")
plt.xticks(fontsize=16, fontweight='bold', color="Black")
plt.yticks(fontsize=16, fontweight='bold', color="Blue")

X_axis = np.arange(len(Lst1))
Lst1_string =",".join(str(element) for element in Lst1)
Lst2_string = ",".join(str(element) for element in Lst2)
ax.bar(X_axis - 0.2, Lst1, width=0.4, label = "Nonrecursiv: "+Lst1_string, color="Red")
ax.bar(X_axis + 0.2, Lst2, width=0.4, label = "Recursiv   : "+Lst2_string, color="Blue")

plt.title("Sequence generated using an imposed algorithm",      fontsize=16, fontweight='bold', color="Black")
plt.legend() ; plt.grid(True)
plt.show()
```

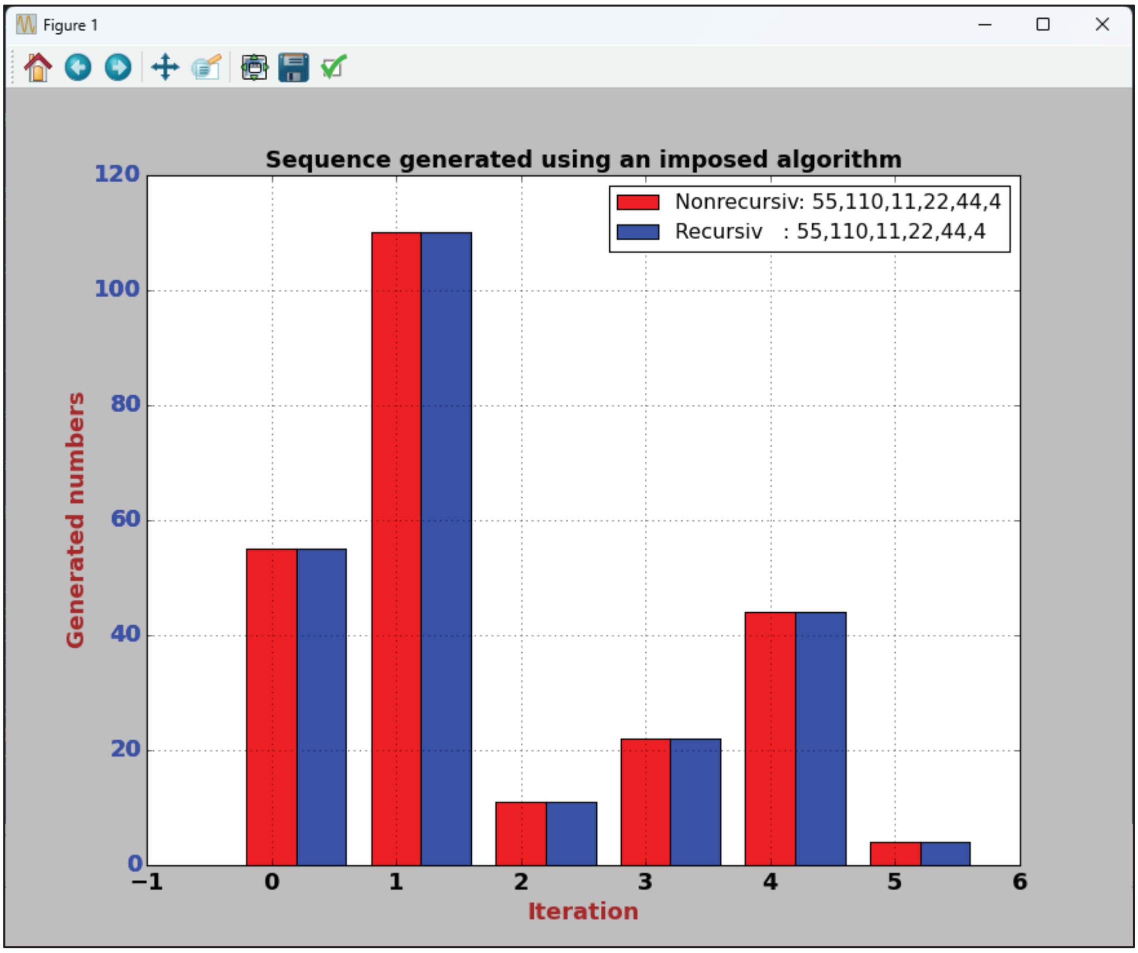


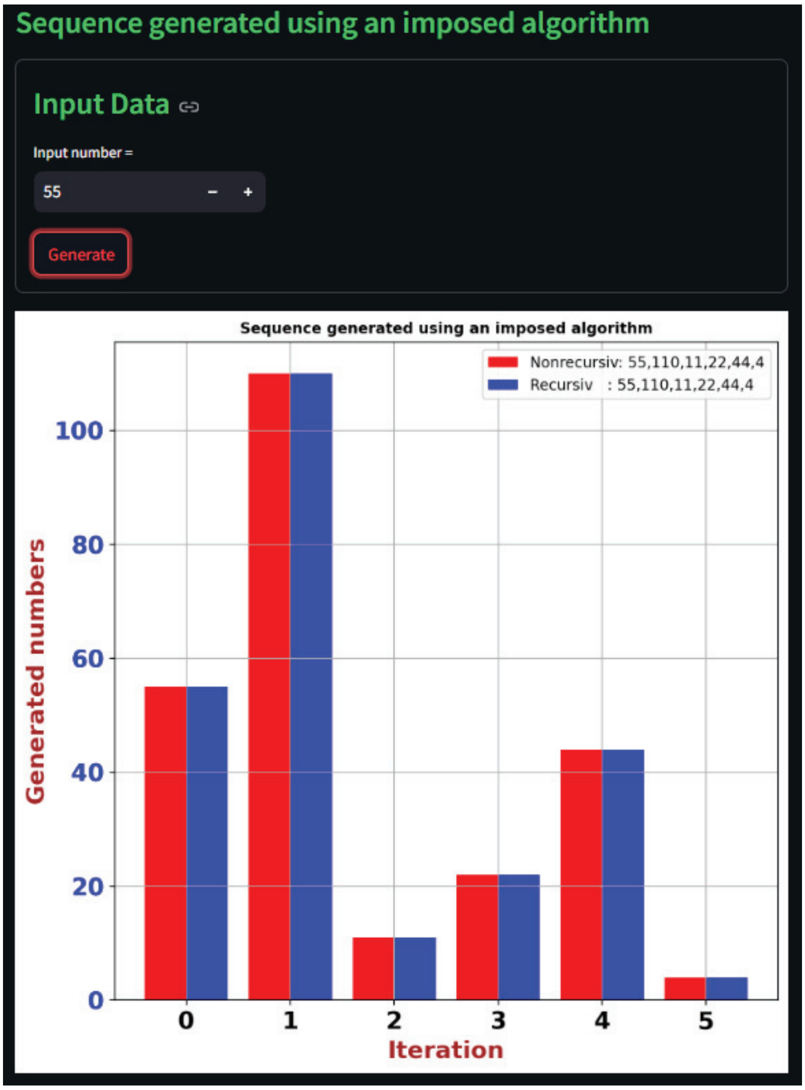
Figure 1
Sequence generated using an imposed algorithm
🟥 Nonrecursiv: 55,110,11,22,44,4
🟦 Recursiv   : 55,110,11,22,44,4
120
100
80
60
40
20
0
Generated numbers
−1
0
1
2
3
4
5
6
Iteration

Figure 7.5 The comparative results for **Generate_Sequence.pyw** script

Listing 7.2 displays the **Python & Streamlit** version of script **Generate_Sequence.py**. The script use the same functions: **NonRecursiv** and **Recursiv** like in **Generate_Sequence.pyw** script. **Figure 7.6** exemplify the comparative graphical results of **Generate_Sequence.py** script.

Listing 7.2 Generate_Sequence.py – Sequence generated using an imposed algorithm – Python & Streamlit version

```
import streamlit as st
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

def NonRecursiv(N):   # Nonrecursiv variant
    Lst=[N]
    while N != 4:
        x=str(N)     # Convert N to string
        if x[-1]=='0' or x[-1]=='4':
            x=x[:-1] # Remove the last string char
            N=int(x) # Convert string to N
        else:
            N=2*N    # Doubling N
        Lst.append(N)
    return Lst

def Recursiv(X): # Recursive variant
    x=str(X)
    if X==4:
        return    # Exit from recursive calls
    elif x[-1]=='0' or x[-1]=='4':
        x=x[:-1] # Remove the last string char
        N=int(x) # Convert string to N
    else:
        N=2*X    # Doubling N
    Lst2.append(N)
    Recursiv(N)  # Self function call

st.subheader(":green[Sequence generated using an imposed algorithm]")
with st.form('Input Data'):  # Create input data form
        st.subheader(':green[Input Data]')
        col1, col2, col3 = st.columns(3)
        N= col1.number_input('Input number  = ', value=55)
        submit_button = st.form_submit_button('Generate')
if submit_button:
     # Main program ======================================
    N=55
    Lst1 = NonRecursiv(N) # Call nonrecursiv function
    Lst2 =[N]
    Recursiv(N)          # First recursive function call

    placeholder1 = st.empty()
    fig = plt.figure(figsize=(8, 8)) ; ax = plt.axes()
    plt.xlabel("Iteration", fontsize=16, fontweight='bold', color="Brown")
    plt.ylabel("Generated numbers", fontsize=16, fontweight='bold', color="Brown")
    plt.xticks(fontsize=16, fontweight='bold', color="Black")
    plt.yticks(fontsize=16, fontweight='bold', color="Blue")

    X_axis = np.arange(len(Lst1))
    Lst1_string =",".join(str(element) for element in Lst1)
    Lst2_string = ",".join(str(element) for element in Lst2)
    ax.bar(X_axis - 0.2, Lst1, width=0.4, label = "Nonrecursiv: "+Lst1_string, color="Red")
    ax.bar(X_axis + 0.2, Lst2, width=0.4, label = "Recursiv   : "+Lst2_string, color="Blue")

    plt.title("Sequence generated using an imposed algorithm", fontsize=10, fontweight='bold', color="Black")
    plt.legend() ; plt.grid(True) ; plt.show()  ;  placeholder1.pyplot(fig)
```



Sequence generated using an imposed algorithm

Input Data

Input number = 55

−
+
Generate
Sequence generated using an imposed algorithm
Nonrecursiv: 55,110,11,22,44,4
Recursiv : 55,110,11,22,44,4
Generated numbers
0
20
40
60
80
100
Iteration
0
1
2
3
4
5

Figure 7.5 The comparative results for **Generate_Sequence.pyw** script

8. Cylinder unfold

This script is designed to unfold the cylinder intersected by two planes, Figure 8.1. The unfolded surface and the applied restrictions are displayed in Figure 8.2. The input data are: the cylinder diameter D, the cylinder height H, α and β angles; the results of this script are the unfolded surface, the main dimensions, area and coordinates of the upper side unfolded surface.

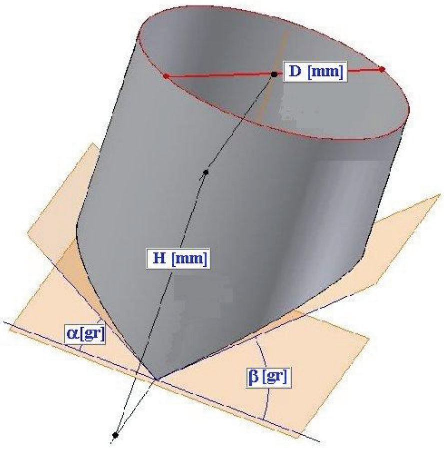
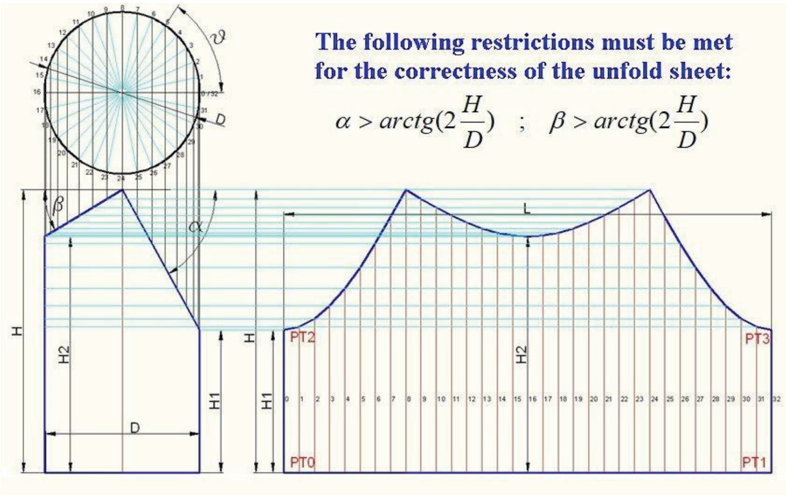


Figure 8.1 The cylinder 3D geometry



The following restrictions must be met for the correctness of the unfold sheet:
$\alpha > arctg(2\frac{H}{D})$; $\beta > arctg(2\frac{H}{D})$
ϑ
D
β
α
L
H
H2
H1
D
PT0
PT1
PT2
PT3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32

Figure 8.2 The unfolded cylinder

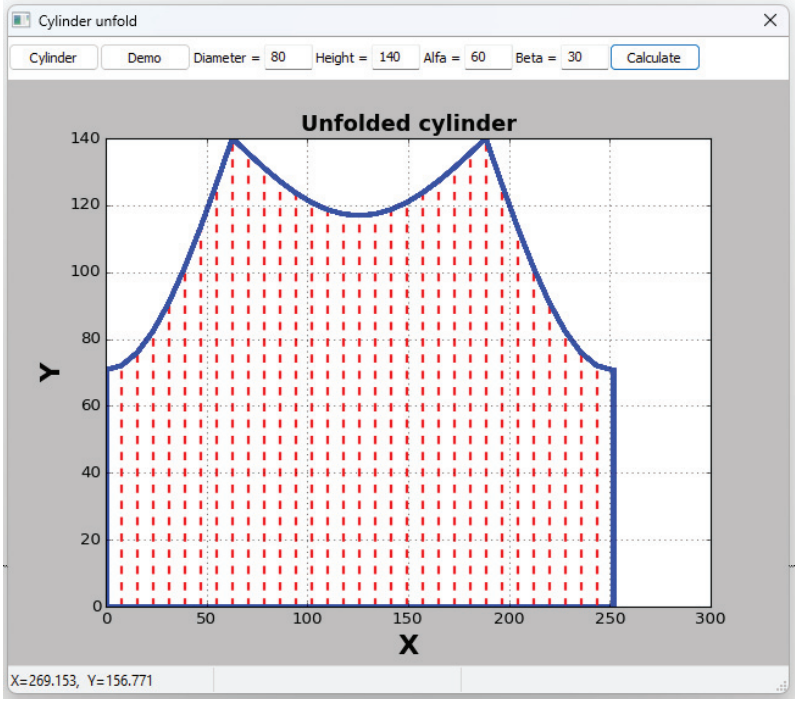
Listing 8.1 displays the Python & wxPython version of script, **Cylinder.pyw** and Listing 8.2 displays the Python & Streamlit version of the same script, **Cylinder.py**. Figure 8.3a & 8.3b display the screens generated by Python & wxPython version of the script, while Figure 8.4 display the screens generated by Python & Streamlit version.

The **Cylinder.pyw** script (Python & wxPython version) include only one class: **Plot_ Window**. The script begin with importing libraries: **wx** (the cross-platform GUI toolkit for the Python language), **os** (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.), **numpy** (the package for scientific computing with Python), **pandas** (open source data analysis and manipulation tool) and **matplotlib** (library for creating static, animated and interactive visualizations in Python). The **Plot_ Window** class create the interface, compute & draw the unfolded surface and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame) the **StatusBar** (placed at the frame bottom, were messages can be displayed); call the **_TOOLBAR** function, initializes top level container for all plot elements (the chart area **self.figure** & **self.axes**) and center the frame in the display.
- **_TOOLBAR** Create the **toolbar** (lines 31÷33), the **self.btn_Img1** button (lines 35÷36) and connect with **Bind** statement (line 37) to **OnImage1** function to display the picture from Figure 8.1, the **self.btn_Img2** button (lines 38÷39) and connect with **Bind** statement (line 40) to **OnImage2** function to display the picture from Figure 8.2; create four text fields (**wx.TextCtrl**) to input numerical values of input data variables (**Diameter**, **Height**, Alfa & **Beta** angles) (lines 42÷49); before text fields three **wx.StaticText** controls are defined as labels; create the **Calculate** button (line 51) and connect with **Bind** statement to **OnClicked** function (line 52). Lines 54÷65 add these controls to the **ToolBar** and lines 66÷67 generate and show the **ToolBar**.
- **OnImage1** Call **OnImage** function to display the picture from Figure 8.1 (line 69).
- **OnImage2** Call **OnImage** function to display the picture from Figure 8.2 (line 72).
- **OnImage** Display the picture defined in the **OnImage1** or **OnImage2** call.
- **MouseMotion** Display in the **StatusBar** the current values of X & Y mouse position in the chart.
- **Recreate_Axis** Initialize the chart (line 81), show the chart grid (line 82) and set the labels for X & Y axes of the chart (lines 83÷84).
- **OnClicked** The function is called by **Calculate** button; extract the input data values (**Diameter**, **Height**, Alfa

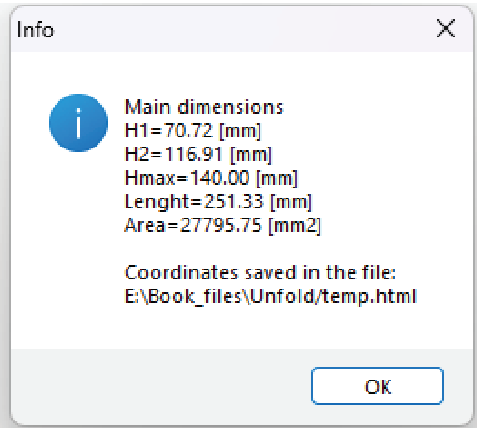
& **Beta** angles) from text fields (lines 86÷89), calculate the **PI** constant (line 90), verify restrictions from **Figure 8.2** (lines 91÷95), calculate the unfold surface main dimensions and coordinates (lines 97÷110); create the **canvas** and drawing **self.axes** (lines 112÷113); create the **sizer** (line 114) and place the **canvas** inside the **sizer** (line 115); line 103 call **Recreate_Axis** function; lines 118÷123 plot the chart elements from **Figure 8.3a**; the chart is updated by the **draw** statement (line 124); lines 128÷132creat dataframe element with coordinates: Teta angle, X and Y, saved into HTML file by line 134; the main dimension are placed into a string variable **sir** (lines 136÷143) displayed by **wx.MessageBox** command (line 144) **Figure 8.3b**.

Line 146 create the **app** which initializes the GUI toolkit for xPython; line 147 call **Plot_Window** class to create the window showed by line 148; finally, the line 151 enter into infinite continuous loop to receive key events from the user; the script can be interrupted with ☒ button of the frame.



Diameter = 80 Height = 140 Alfa = 60 Beta = 30

X=269.153, Y=156.771



Main dimensions
H1=70.72 [mm]
H2=116.91 [mm]
Hmax=140.00 [mm]
Lenght=251.33 [mm]
Area=27795.75 [mm2]

Coordinates saved in the file:
E:\Book_files\Unfold/temp.html

Figure 8.3a **Figure 8.3b**

Figure 8.3 The screens generated by Python & wxPython version of the **Cylinder.pyw** script

Listing 8.1 Cylinder.pyw - Cylinder unfold – Python & wxPython version

```
1.  # The // operator will be available to request floor division unambiguously. This
2.  # statement will change the / operator to mean true division throughout the module.
3.  from __future__ import division
4.  
5.  import wx    # import wx module
6.  import os
7.  import numpy as np
8.  import pandas as pd
9.  import matplotlib as mpl  # The tools are imported from the Matplotlib library
10. import matplotlib.pyplot as plt
11. import matplotlib.image as img
12. from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
13. from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar
14. 
15. class Plot_Window(wx.Frame):
16.         def __init__(self, parent, title):
17.                 wx.Frame.__init__(self, parent, title=title, size=(650,600),
18.                         style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
19.                         wx.MAXIMIZE_BOX )) # Initialize the frame
20.                 self.panel = wx.Panel(self)  #  The panel will hold the contents of the frame
21.                 self.statusbar = self.CreateStatusBar() # Create the status bar
22.                 self.statusbar.SetFieldsCount(3)
23.                 self.statusbar.SetStatusWidths([-25, -30, -40])
24.                 self.statusbar.SetStatusText("This is the plot window",0)  # Add text to StatusBar
```

Listing 8.1 Cylinder.pyw - Cylinder unfold – Python & wxPython version

```
        self._TOOLBAR() # Create toolbar
        self.figure = mpl.figure.Figure()  # Initializes top level container for all plot elements
        self.axes=self.figure.add_subplot(111) # Add only one subplot area
        self.Center()  # Center the frame in display
    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour("white")
        self.toolbar.SetToolBitmapSize((24,24))

        self.btn_Img1 = wx.Button(self.toolbar, -1, "Cylinder")
        self.btn_Img1.SetToolTipString("View cylinder geometry")
        self.btn_Img1.Bind(wx.EVT_BUTTON,self.OnImage1)
        self.btn_Img2 = wx.Button(self.toolbar, -1, "Demo")
        self.btn_Img2.SetToolTipString("View unfolding demo & restrictions")
        self.btn_Img2.Bind(wx.EVT_BUTTON,self.OnImage2)

        st1 = wx.StaticText(self.toolbar, -1, " Diameter = ")
        self.txt_Diameter = wx.TextCtrl(self.toolbar, value="80", size=(40,-1))
        st2 = wx.StaticText(self.toolbar, -1, " Height = ")
        self.txt_Height = wx.TextCtrl(self.toolbar, value="140", size=(40,-1))
        st3 = wx.StaticText(self.toolbar, -1, " Alfa = ")
        self.txt_Alfa = wx.TextCtrl(self.toolbar, value="60", size=(40,-1))
        st4 = wx.StaticText(self.toolbar, -1, " Beta = ")
        self.txt_Beta = wx.TextCtrl(self.toolbar, value="30", size=(40,-1))

        self.btn = wx.Button(self.toolbar, -1, "Calculate")
        self.btn.Bind(wx.EVT_BUTTON,self.OnClicked)

        self.toolbar.AddControl(self.btn_Img1)
        self.toolbar.AddControl(self.btn_Img2)
        self.toolbar.AddControl(st1)
        self.toolbar.AddControl(self.txt_Diameter)
        self.toolbar.AddControl(st2)
        self.toolbar.AddControl(self.txt_Height)
        self.toolbar.AddControl(st3)
        self.toolbar.AddControl(self.txt_Alfa)
        self.toolbar.AddControl(st4)
        self.toolbar.AddControl(self.txt_Beta)

        self.toolbar.AddControl(self.btn )
        self.toolbar.Realize()     # Toolbar generation
        self.toolbar.Show()        # Toolbar show
    def OnImage1(self, event):
        testImage = img.imread(os.getcwd()+'/Cylinder.jpg')
        self.OnImage('Cylinder.jpg')
    def OnImage2(self, event):
        self.OnImage('Cylinder_Demo.jpg')
    def OnImage(self, Image_File):
        testImage = img.imread(os.getcwd()+'/'+Image_File)
        plt.imshow(testImage)
        plt.show()
    def MouseMotion(self, event):
        a1,b1=self.axes.transData.inverted().transform([event.x, event.y])
        self.statusbar.SetStatusText("X="+'%0.3f' % a1+",  Y="+'%0.3f' % b1,0)
    def Recreate_Axis(self):
        self.axes.cla()
        self.axes.grid(True)
        self.axes.set_xlabel('X', fontsize=20, fontweight='bold')
        self.axes.set_ylabel('Y', fontsize=20, fontweight='bold')
    def OnClicked(self, event):
        Diameter = float(self.txt_Diameter.GetValue())
        Height = float(self.txt_Height.GetValue())
        Alfa = int(self.txt_Alfa.GetValue())
        Beta = int(self.txt_Beta.GetValue())
        PI=np.arctan(1)*4
        Umax= np.arctan(2 * Height / Diameter) * 180 / PI
        if Diameter=="" or Height==""  or Alfa==""  or Beta==""  :
            wx.MessageBox("Please fill all fields with numerical data !", "Info", wx.ICON_ERROR)
        elif (Alfa > Umax or Beta > Umax):
            wx.MessageBox("Alpha and Beta angles cannot be greater than "+str(Umax)+" grade", "Info",
                    wx.ICON_ERROR)
        else:
```

Listing 8.1 Cylinder.pyw - Cylinder unfold – Python & wxPython version

```
          NrPct = 8 ; ALFAR=Alfa * PI / 180 ; BETAR=Beta * PI / 180
          H1 = Height - Diameter * np.tan(ALFAR) / 2
          H2 = Height - Diameter * np.tan(BETAR) / 2
          Lungime = PI * Diameter # Unfolded length
          Aria = Diameter * (Height * PI - Diameter * (np.tan(ALFAR) + np.tan(BETAR)) / 2)
          XP=[] ; YP=[] ; Teta=[]
          for i in range(1, 4 * NrPct + 2):
            UT=(i - 1) * PI / 2 / NrPct
            Teta.append(UT*180/PI)
            XP.append(UT * Diameter / 2)
            if ((UT < PI / 2) or (UT > 3 * PI / 2)):
                YP.append(Height - Diameter / 2 * np.cos(UT) * np.tan(ALFAR))
            else:
                YP.append(Height + Diameter / 2 * np.cos(UT) * np.tan(BETAR))
        Xmax=XP[4 * NrPct] ; H1=YP[0]
        self.canvas = FigureCanvas(self.panel,-1,self.figure)
        self.axes.figure.canvas.mpl_connect('motion_notify_event', self.MouseMotion)
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas, 1, wx.TOP | wx.LEFT | wx.EXPAND)
        self.Recreate_Axis()
        self.axes.set_title("Unfolded cylinder", fontsize=16, fontweight='bold')
        for i in range(1, 4 * NrPct):
                self.axes.plot([XP[i], XP[i]],[0, YP[i]], '--', color='Red', linewidth=2)
                self.axes.plot(XP,YP, color='Blue', linewidth=3)
                self.axes.plot([0,0],[0,H1], color='Blue', linewidth=3)
                self.axes.plot([0,Xmax],[0,0], color='Blue', linewidth=3)
                self.axes.plot([Xmax,Xmax],[0,H1], color='Blue', linewidth=3)
        self.axes.figure.canvas.draw()
        self.SetSizer(sizer)
        self.Fit()

        df= pd.DataFrame(
                {'Teta [deg]': Teta,
                'X [mm]': XP,
                'Y [mm]': YP
                })
        file=os.getcwd()+'/'+'temp.html'
        df.to_html(file)

        sir="Main dimensions"
        sir=sir+"\nH1="+'%0.2f' % H1+" [mm]"
        sir=sir+"\nH2="+'%0.2f' % H2+" [mm]"
        sir=sir+"\nHmax="+'%0.2f' % Height+" [mm]"
        sir=sir+"\nLenght="+'%0.2f' % Lungime+" [mm]"
        sir=sir+"\nArea="+'%0.2f' % Aria+" [mm2]"
        sir=sir+"\n\nCoordinates saved in the file:"
        sir=sir+"\n"+file
        wx.MessageBox(sir, "Info", wx.OK)

app = wx.App(redirect=False)   # wx.App initializes the GUI toolkit for WxPython.
window=Plot_Window(None, "Cylinder unfold")  # Call 'Plot_Window' class
window.Show()  # Show 'window' object
# This keeps the GUI in a continuous loop that is ready to receive key events
# from the user. It does not return until the program closes!
app.MainLoop()
```

The **Cylinder.py** script (Python & Streamlit version) begin with importing libraries: **streamlit** (create a great interactive web application with Python), **PIL** (Python Imaging Library), **pandas** (open source data analysis and manipulation tool), **matplotlib** (library for creating static, animated and interactive visualizations in Python), **numpy** (the package for scientific computing with Python), **pathlib** (provide classes to represent abstract/concrete paths.

This script contains 87 lines. Lines 1 to 6 import the required modules. Line 8 defines the current folder. Line 10 defines the script title with **subheader** command. Line 12 inserts a container element (with **st.empty()** command) into the script that can be used to hold two columns defined in line 13; the **Cylinder.jpg** image (**Figure 8.4**) is placed on column 1 and also define the width of the image and no caption (lines 14÷15); the **Cylinder_Demo.jpg** image (**Figure 8.4**) is placed on column 2 and also define the width of the image and no caption (lines 16÷17). Line 19 inserts a container element (with **st.empty()** command) into the script that can be used to hold three columns defined in line 20; line 21÷22 open and read the video file **'Cylinder.mp4'**, while line 23 place the video into median column two.

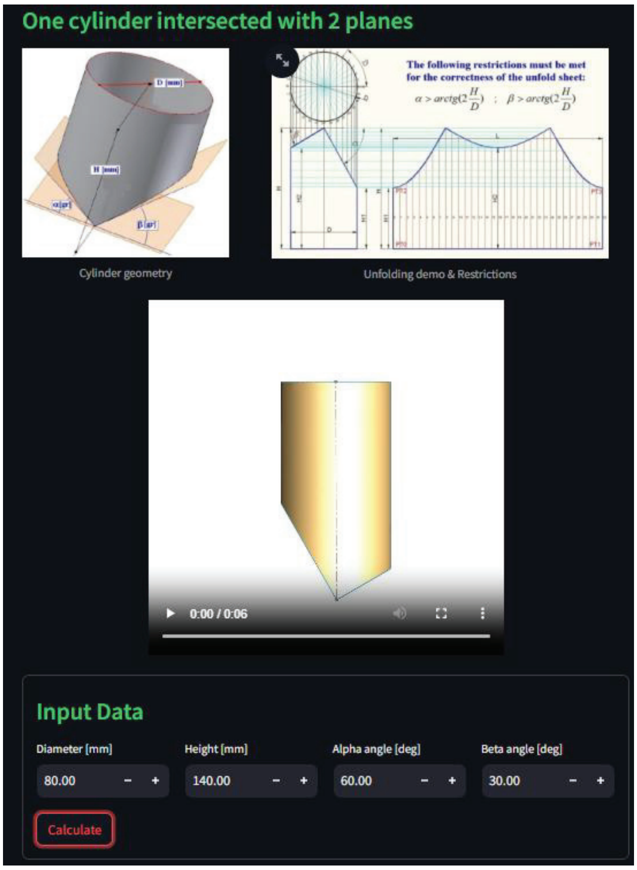
Line 25 creates a form where input data will be specified. Line 26 defines the form title with **subheader** command. Line 19 creates columns with four elements. Lines 28÷31 define the fields of input data (**Diameter**, **Height**, Alfa & **Beta** angles) into the previous columns created, using **number_input** command. The form require 4 numerical data and the **number_input** command specify the default values that can be changed by the user. Line 23 creates the button with the **form_submit_button** command and set the caption **Calculate**. When this button is clicked, all widget values inside the form will be sent to Streamlit in a batch. Every form must have a **form_submit_button**, which cannot exist outside a form. Line 33 verifies if the button was clicked and, only for **True** value, run the next lines; line 34 calculate the **PI** constant; lines 35÷39 verify restrictions from **Figure 8.2**; lines 41÷54 calculate the unfold surface main dimensions and coordinates. Lines 56÷69 plot the chart elements from **Figure 8.4**; **l**ines 71÷87display main dimensions and unfolded coordinate.

Listing 8.2 Cylinder.py - Cylinder unfold – Python & Streamlit version

```
21.   import streamlit as st
22.   from PIL import Image
23.   import pandas as pd
24.   import matplotlib.pyplot as plt
25.   import numpy as np
26.   from pathlib import Path
27.
28.   current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()
29.
30.   st.subheader(":green[One cylinder intersected with 2 planes - Version 1]")
31.
32.   Place_Images = st.empty()
33.   col1, col2 = Place_Images.columns([0.4,0.6])
34.   col1.image(Image.open(str(current_dir)+"/"+"Cylinder.jpg"),
35.               width=240, caption='Cylinder geometry')
36.   col2.image(Image.open(str(current_dir)+"/"+"Cylinder_Demo.jpg"),
37.               width=390, caption='Unfolding demo & Restrictions')
38.
39.   Place_Images = st.empty()
40.   col1, col2, col3  = Place_Images.columns([0.2,0.6,0.2])
41.   video_file = open(str(current_dir)+"/"+'Cylinder.mp4', 'rb')
42.   video_bytes = video_file.read()
43.   col2.video(video_bytes, format="mp4", start_time=0)
44.
45.   with st.form('Input_Data'):
46.       st.subheader(':green[Input Data]')
47.       col1, col2, col3, col4 = st.columns(4)
48.       Diameter = col1.number_input('Diameter [mm]', value=80.0)
49.       Height = col2.number_input('Height [mm]', value=140.0)
50.       Alfa = col3.number_input('Alpha angle [deg]', value=60.0)
51.       Beta = col4.number_input('Beta angle [deg]', value=30.0)
52.       submit_button = st.form_submit_button('Calculate')
53.   if submit_button:
54.       PI=np.arctan(1)*4
55.       Umax= np.arctan(2 * Height / Diameter) * 180 / PI
56.       if Diameter=="" or Height==""  or Alfa==""  or Beta==""  :
57.           st.warning("Please fill all fields with numerical data !")
58.       elif (Alfa > Umax or Beta > Umax):
59.           st.warning("Alpha and Beta angles cannot be greater than "+str(Umax)+" grade")
60.       else:   # st.success("Success !")
61.           NrPct = 8 ; ALFAR=Alfa * PI / 180 ; BETAR=Beta * PI / 180
62.           H1 = Height - Diameter * np.tan(ALFAR) / 2
63.           H2 = Height - Diameter * np.tan(BETAR) / 2
64.           Lungime = PI * Diameter # Lungime desfasurata
65.           Aria = Diameter * (Height * PI - Diameter * (np.tan(ALFAR) + np.tan(BETAR)) / 2)
66.           XP=[] ; YP=[] ; Teta=[]
67.           for i in range(1, 4 * NrPct + 2):
68.               UT=(i - 1) * PI / 2 / NrPct
69.               Teta.append(UT*180/PI)
70.               XP.append(UT * Diameter / 2)
71.               if ((UT < PI / 2) or (UT > 3 * PI / 2)):
72.                   YP.append(Height - Diameter / 2 * np.cos(UT) * np.tan(ALFAR))
73.               else:
74.                   YP.append(Height + Diameter / 2 * np.cos(UT) * np.tan(BETAR))
```

Listing 8.2 Cylinder.py - Cylinder unfold – Python & Streamlit version

```

        # Chart drawing
        Place_Chart = st.empty()
        Xmax=XP[4 * NrPct] ; H1=YP[0]
        fig=plt.figure()
        plt.title("Unfolded cylinder - Variant 1", fontsize=14,
                fontweight='bold',color='Black')
        plt.grid(True)
        for i in range(1, 4 * NrPct):
            plt.plot([XP[i], XP[i]],[0, YP[i]], color='Red', linewidth=1)
        plt.plot(XP,YP, color='Blue', linewidth=3)
        plt.plot([0,0],[0,H1], color='Blue', linewidth=3)
        plt.plot([0,Xmax],[0,0], color='Blue', linewidth=3)
        plt.plot([Xmax,Xmax],[0,H1], color='Blue', linewidth=3)
        Place_Chart.write(fig)

        # Chart dimensions & coordinates
        st.divider()
        col1, col2 = st.columns([0.4,0.6])
        col1.subheader(":green[Main dimensions]")
        col1.write("H1="+'%0.2f' % H1+" [mm]")
        col1.write("H2="+'%0.2f' % H2+" [mm]")
        col1.write("Hmax="+'%0.2f' % Height+" [mm]")
        col1.write("Lenght="+'%0.2f' % Lungime+" [mm]")
        col1.write("Area="+'%0.2f' % Aria+" [mm2]")
        df= pd.DataFrame(
            {'Teta [deg]': Teta,
            'X [mm]': XP,
            'Y [mm]': YP
            })
        col2.subheader(":green[Unfolded Coordinates]")
        col2.dataframe(df)
        st.divider()
```



One cylinder intersected with 2 planes

D [mm]
H [mm]
α [gr]
β [gr]
The following restrictions must be met for the correctness of the unfold sheet:
α > arctg(2H/D) ; β > arctg(2H/D)
0:00 / 0:06

Cylinder geometry

Unfolding demo & Restrictions

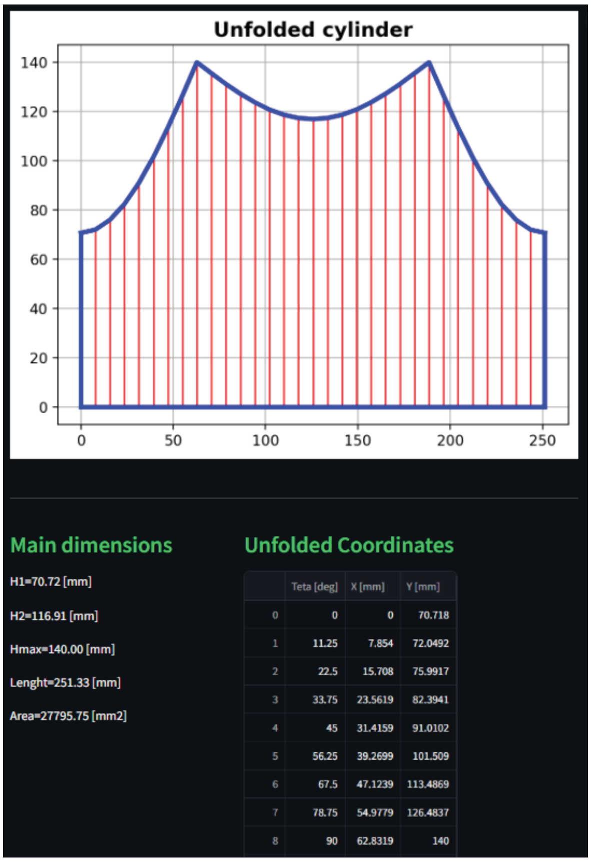
Input Data

Diameter [mm] 80.00 − +

Height [mm] 140.00 − +

Alpha angle [deg] 60.00 − +

Beta angle [deg] 30.00 − +



Unfolded cylinder

Main dimensions

H1=70.72 [mm]

H2=116.91 [mm]

Hmax=140.00 [mm]

Lenght=251.33 [mm]

Area=27795.75 [mm2]

Unfolded Coordinates

	Teta [deg]	X [mm]	Y [mm]
0	0	0	70.718
1	11.25	7.854	72.0492
2	22.5	15.708	75.9917
3	33.75	23.5619	82.3941
4	45	31.4159	91.0102
5	56.25	39.2699	101.509
6	67.5	47.1239	113.4869
7	78.75	54.9779	126.4837
8	90	62.8319	140

Figure 8.4 The screens generated by Python & Streamlit version of the **Cylinder.py** script

9. Elbow unfold

This script is designed to unfold the elbow geometry from **Figure 9.1**. The input data are: the elbow diameter D & radius R, the α & β angles; the results of this script are the elbow geometry and unfolded surfaces; also the main dimensions, the unfolded coordinates and the unfolded surfaces drawings exported into an Excel file using Windows clipboard.

Listing 9.1 displays the Python & wxPython version of script, **Elbow.pyw** and **Listing 9.2** displays the Python & Streamlit version of the same script, **Elbow.py**. **Figure 9.2** display interface & the screens generated by Python & wxPython version of the script and **Figure 9.3** display the Excel file content generated by **Elbow.pyw** script. **Figure 9.4** display the screens generated by Python & Streamlit version.

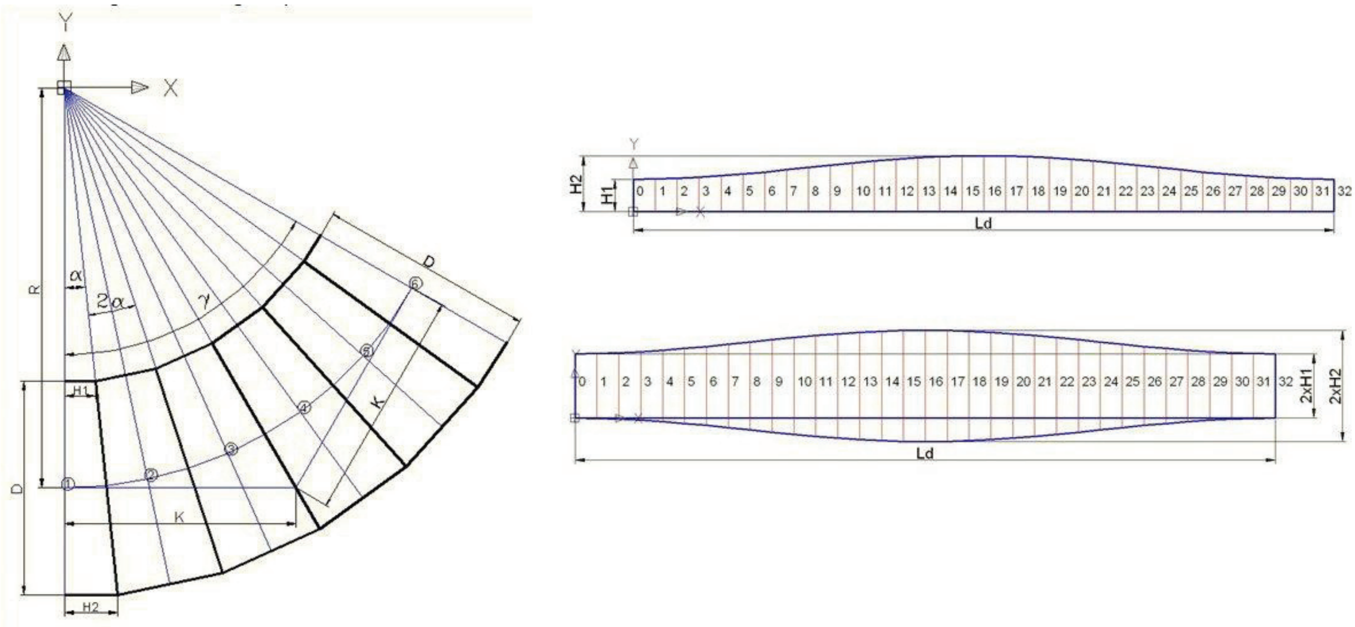


Figure 9.1 The elbow geometry and unfolded surfaces

The **Elbow.pyw** script (Python & wxPython version) include only one class: **Plot_ Window**. The script begin with importing libraries: **wx** (the cross-platform GUI toolkit for the Python language), **os** (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.), **numpy** (the package for scientific computing with Python), **pandas** (open source data analysis and manipulation tool) **StringIO** (the StringIO module is used to implement basic tasks on file-like objects), **cStringIO** (an optional module, which contains a faster implementation of the **StringIO** module), **from win32com.client import Dispatch** (the **win32com. client** package contains a number of modules to provide access to automation objects) and **matplotlib** (library for creating static, animated and interactive visualizations in Python). To use **win32com.client** the **pywin32-214.win32-py2.7.exe** file must be installed for python version 2.7; this will be used to connect with an Excel or Word file.

The **Plot_ Window** class create the interface, draw the elbow geometry and unfolded surfaces and include the following functions:

- **Put_Clipboard** Public function to copy content to Windows clipboard.
- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame) the **StatusBar** (placed at the frame bottom, were messages can be displayed); call the **_TOOLBAR** function, initializes top level container for all plot elements (the chart area **self.figure** & **self.axes1** ÷ **self.axes3**); center the frame in the display and call **Recreate_Axis** class function; **self.axes1** is used to draw the elbow geometry; **self.axes2** and **self.axes2** are used to draw the elbow unfolded surfaces.
- **_TOOLBAR** Create the **toolbar** (lines 46÷48); create the toolbar **Image** button (lines 50÷51) and connect with **Bind** statement (line 52) to **OnImage** class function to show the image from **Figure 9.1**; create **self.D** & **self.R** as **TextCtrl** to input the diameter and radius (lines

54÷58); create self.Gama & self.Alfa as Choice to select values of γ and α angles (lines 60÷68); line 68 put the self.Alfa control into a disabled state, which will be reversed in OnGama function; connect self. Gama control with Bind statement (line 64) to OnGama class function (line 64); connect self.Alfa control with Bind statement (line 64) to OnAlfa class function (line 67); create self.N as TextCtrl to display the numbert of elbow elements (lines 70÷71); line 72 put the self.N control into a permanent disabled state because the value will be calculated by the script in OnAlfa function; create the toolbar Calculate button (line 73) and connect with Bind statement (line 74) to OnClicked class function to calculate the elbow main dimensions, to draw the elbow and the unfolded surfaces, to export numerical results (main dimensions & coordinates) and charts to Excel.

- **Recreate_Axis** Initialize the axes1, axes 2, axes3, show the axes grid and set the axes titles (lines 86÷88).
- **OnImage** Display the picture from Figure 9.1 (lines 82÷84).
- **OnGama** Put the value of selected γ angle into Gama_Sel variable (line 90); define the posible values of α angle (line 91) into Alfa_Angles list; line 92 put the self.Alfa control into a enabled state; intialize variable Selected_Alfa list (line 93); because, for a selected γ value only certain α angle values can be selected, into Selected_Alfa list will be appended only the values that verify the condition from line 98; these values from Selected_Alfa list will be transferred to self.Alfa Choice control to permit the selection of one α angle.
- **OnAlfa** Put the value of selected γ angle into Gama_Sel variable (line 102); put the value of selected α angle into Alfa_Sel variable (line 103); calculate the number of elbow elements r1 based on the condition from line 106; put the r1 value into self.N control in toolbar.
- **OnClicked** Lines 109÷130 calculate the main dimension of elbow and drawings coordinates; lines 133÷154 draw the elbow geometry in self.axes1, Figure 9.2; lines 157÷165 draw the unfolded surface of element 1 in self.axes2, Figure 9.2; lines 168÷177 draw the unfolded surface of element 2 in self.axes3, Figure 9.2; lines 180÷183 connect, create and open the Excel file Elbow_Results.xlsx and define the sheet XLSheet variable; lines 185÷197 write the text marked by quotation marks into cells A1÷A13; lines 199÷205 define a dataframe df containing the coordinates of unfolded surfaces (element 1 & 2); line 206 connvert df dataframe to lis type; line 208 and 209 define the table headings as string, put this string in clipboard and paste in Excel sheet in starting with line 15 column 1; in a for cycle a string of coordinates are created (lines 211÷213), put this string String in clipboard and paste in Excel sheet in starting with line 16 column 1; lines 216÷217 save the chart into ' Charts.jpg' image file, open by 218 line and converted to bitmap by 219÷220 lines; line 222 define the position (column left and line top) where the image will be placed in Excel file; the image is placed in Excel sheet by 223 command line and saved by 229 line, Figure 9.3.

Line 234 create the app which initializes the GUI toolkit for xPython; line 235 call Plot_Window class to create the window showed by line 236; finally, the line 239 enter into infinite continuous loop to receive key events from the user; the script can be interrupted with ☒ button of the frame.

Listing 9.1 Elbow.pyw – Elbow unfold – Python & wxPython version

```
1.    # The // operator will be available to request floor division unambiguously. This
2.    # statement will change the / operator to mean true division throughout the module.
3.    from __future__ import division
4.
5.    import wx    # import wx module
6.    import os
7.    import numpy as np
8.    import pandas as pd
9.    import StringIO, cStringIO
10.   # The win32com.client package contains modules to provide access to automation objects
11.   from win32com.client import Dispatch
```

Listing 9.1 Elbow.pyw – Elbow unfold – Python & wxPython version

```
import win32con # This module contains constants related to Win32 programming
import win32clipboard
import matplotlib as mpl   # The tools are imported from the Matplotlib library
import matplotlib.pyplot as plt  # matplotlib.pyplot = collection of functions that make matplotlib work like MATLAB
import matplotlib.image as img # Import matplotlib.image  module
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as FigureCanvas
from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar

def Put_Clipboard(Sir_Clip):
  win32clipboard.OpenClipboard() # Open clipboard
  win32clipboard.EmptyClipboard() # Empty clipboard
  win32clipboard.SetClipboardData(win32con.CF_TEXT, Sir_Clip) # Set clipboard data
  win32clipboard.CloseClipboard() # Close clipboard

class Plot_Window(wx.Frame):
        def __init__(self, parent, title):
                wx.Frame.__init__(self, parent, title=title, size=(800,730),
                        style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
                        wx.MAXIMIZE_BOX )) # Initialize the frame
                self.panel = wx.Panel(self)  #  The panel will hold the contents of the frame
                self.statusbar = self.CreateStatusBar() # Create the status bar
                self.statusbar.SetFieldsCount(2)
                self.statusbar.SetStatusWidths([-15, -60])
                self.statusbar.SetStatusText("This is the plot window",0)  # Add text to StatusBar
                self._TOOLBAR() # Create toolbar
                self.figure = mpl.figure.Figure(figsize =(10, 8))  # Initializes top level container for all plot elements
                #   figsize define  Figure dimension (width, height) in inches
                self.axes1=self.figure.add_subplot(121)    ;   self.axes1.axis('equal')   # Add three subplot area
                self.axes2=self.figure.add_subplot(222)  # Add first subplot area
                self.axes3=self.figure.add_subplot(224)  # Add second subplot area
                self.Center()  # Center the frame in display
                self.canvas = FigureCanvas(self.panel,-1,self.figure)
                self.Recreate_Axis()
        def _TOOLBAR(self):
                self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
                self.toolbar.SetBackgroundColour("white")
                self.toolbar.SetToolBitmapSize((24,24))

                self.btn_Img = wx.Button(self.toolbar, -1, "Image", size=(50,-1))
                self.btn_Img.SetToolTipString("Show elbow image & unfold demo")
                self.btn_Img.Bind(wx.EVT_BUTTON,self.OnImage)

                st1 = wx.StaticText(self.toolbar, -1, " Diameter = ")
                self.D = wx.TextCtrl(self.toolbar, value="80", size=(40,-1))
                st2 = wx.StaticText(self.toolbar, -1, " Elbow Radius R= ")
                self.R = wx.TextCtrl(self.toolbar, value="150", size=(40,-1))
                st3 = wx.StaticText(self.toolbar, -1, " Gama angle = ")

                Gama_Angles = ["24","25","28","30","32","33","35","36","40","42","44","45","48","49","50","54","55",
                      "56","60","63","64","65","66","67.5","70","72","75","77","78","80","84","85","88","90"]
                Gama_Angles.reverse()
                self.Gama = wx.Choice(self.toolbar, -1, (100, 50), choices = Gama_Angles)
                self.Gama.Bind(wx.EVT_CHOICE, self.OnGama)
                st4 = wx.StaticText(self.toolbar, -1, " Alfa angle = ")
                self.Alfa = wx.Choice(self.toolbar, -1, size=(50, -1), choices = "")
                self.Alfa.Bind(wx.EVT_CHOICE, self.OnAlfa)
                self.Alfa.Enabled=False

                st5 = wx.StaticText(self.toolbar, -1, " No. of elbow elements = ")
                self.N = wx.TextCtrl(self.toolbar, value=" ", size=(40,-1))
                self.N.Enabled=False
                self.btn = wx.Button(self.toolbar, -1, "Calculate")
                self.btn.Bind(wx.EVT_BUTTON,self.OnClicked)

                lst_Control=[self.btn_Img, st1, self.D, st2, self.R, st3, self.Gama, st4,  self.Alfa, st5, self.N, self.btn ]
                for lc in lst_Control:
                    self.toolbar.AddControl(lc)
                self.toolbar.Realize()     # Toolbar generation
                self.toolbar.Show()        # Toolbar show
        def OnImage(self, event):
                testImage = img.imread(os.getcwd()+'/Elbow1.jpg')
                plt.imshow(testImage)
```

Listing 9.1 Elbow.pyw – Elbow unfold – Python & wxPython version

```
        plt.show()
    def Recreate_Axis(self):
        self.axes1.cla()  ;   self.axes1.grid(True)  ; self.axes1.set_title('Elbow geometry')
        self.axes2.cla()  ;   self.axes2.grid(True)  ; self.axes2.set_title('Chart drawing element 1')
        self.axes3.cla()  ;   self.axes3.grid(True)   ; self.axes3.set_title('Chart drawing element 2')
    def OnGama(self, event):
        Gama_Sel = self.Gama.GetStringSelection()
        Alfa_Angles=["2.5","3","3.5","4","5","5.5","6","7.5","8","10","11.25"]
        self.Alfa.Enabled=True
        Selected_Alfa=[]
        if Gama_Sel != "":
            for alf in Alfa_Angles:
                n = (float(Gama_Sel)/float(alf)+2) / 2
                r = np.floor(n)
                if n == r and n>2 and n<=19:
                    Selected_Alfa.append(alf)
        self.Alfa.SetItems( Selected_Alfa )
    def OnAlfa(self, event):
        Gama_Sel = self.Gama.GetStringSelection()
        Alfa_Sel = self.Alfa.GetStringSelection()
        n1 = (float(Gama_Sel)/float(Alfa_Sel)+2) / 2
        r1 = np.floor(n1)
        if n1 == r1:
            self.N.SetValue(str(int(r1)))
    def OnClicked(self, event):
        Diameter = float(self.D.GetValue())
        Raza = float(self.R.GetValue())
        Gama_Sel = self.Gama.GetStringSelection()
        Alfa_Sel = self.Alfa.GetStringSelection()
        No = self.N.GetValue()
        PI=np.arctan(1)*4
        NrPct = 8
        GamaR=float(Gama_Sel)*PI/180
        AlfaR=float(Alfa_Sel)*PI/180
        K = Raza * np.tan(GamaR / 2)
        t = Raza * np.tan(AlfaR)
        H1 = t - Diameter * np.tan(AlfaR) / 2
        H2 = t + Diameter * np.tan(AlfaR) / 2
        Ld = PI * Diameter   # Unfolded length
        Aria1 = PI * Diameter * (H2 - Diameter * (np.tan(AlfaR) + np.tan(0)) / 2)
        Aria2 = PI * Diameter * (2 * H2 - Diameter * (np.tan(AlfaR) + np.tan(0)) / 2)
        XP=[] ; Y1=[] ; Teta=[]
        for i in range(1, 4 * NrPct + 2):
                UT = (i - 1) * PI / 2 / NrPct
                Teta.append(UT)
                XP.append(UT * Diameter/ 2)
                Y1.append(Diameter / 2 * (1 - np.cos(UT)) * np.tan(AlfaR))

        # Elbow drawing
        self.Recreate_Axis()
        self.axes1.set_title("Elbow geometry with "+'%2d' %  float(No)+" elements")
        X11=[] ; Y11=[] ; X22=[] ; Y22=[]
        cnt=0 ; RpD2 = Raza + Diameter/2 ;  RmD2 = Raza - Diameter/2
        for i in range(1, 2 * int(float(No)) ):
            unghi = AlfaR * (i - 1)
            self.axes1.plot([0,RpD2 * np.sin(unghi)],[RpD2,RpD2-RpD2 * np.cos(unghi)], color='Red', linewidth=1)
            if i==1 or (i % 2) == 0 or i == (2 * int(float(No)) - 2) + 1 :
                x1=RpD2*np.sin(unghi) ; x2=RmD2*np.sin(unghi)
                y1=RpD2-RpD2*np.cos(unghi) ; y2=RpD2-RmD2*np.cos(unghi)
                self.axes1.plot([x1,x2],[y1, y2], color='Blue', linewidth=3)
                cnt+=1
                X11.append(x1) ; Y11.append(y1) ; X22.append(x2) ; Y22.append(y2)
        for i in range(0, cnt):
            if i==0:
                Xant1=X11[i] ; Yant1=Y11[i]
                Xant2=X22[i]  ; Yant2=Y22[i]
            else:
                self.axes1.plot([Xant1,X11[i]],[Yant1, Y11[i] ], color='Blue', linewidth=3)
                self.axes1.plot([Xant2,X22[i]],[Yant2, Y22[i] ], color='Blue', linewidth=3)
                Xant1=X11[i] ; Yant1=Y11[i]
                Xant2=X22[i]  ; Yant2=Y22[i]

```

Listing 9.1 Elbow.pyw – Elbow unfold – Python & wxPython version

```
# Chart drawing element 1
YH1=[]
for i in range(0, 4 * NrPct+ 1):
        YH1.append(H1+Y1[i])
        self.axes2.plot([XP[i], XP[i]],[0, YH1[i]], color='Red', linewidth=1)
self.axes2.plot(XP,YH1, color='Blue', linewidth=3)
self.axes2.plot([0,0],[0,H1], color='Blue', linewidth=3)
self.axes2.plot([0,Ld],[0,0], color='Blue', linewidth=3)
self.axes2.plot([Ld,Ld],[0,H1], color='Blue', linewidth=3)
self.axes2.figure.canvas.draw()

# Chart drawing element 2
YH2a=[] ; YH2b=[]
for i in range(0, 4 * NrPct+ 1):
    YH2a.append(H2-H1-Y1[i])
    YH2b.append(H2+H1+Y1[i])
    self.axes3.plot([XP[i], XP[i]],[YH2a[i], YH2b[i]], color='Red', linewidth=1)
self.axes3.plot(XP,YH2a, color='Blue', linewidth=3)
self.axes3.plot(XP,YH2b, color='Blue', linewidth=3)
self.axes3.plot([0, 0],[H2-H1, H2+H1], color='Blue', linewidth=3)
self.axes3.plot([Ld,Ld],[H2-H1, H2+H1], color='Blue', linewidth=3)
self.axes3.figure.canvas.draw()

# Export numerical results  (Main dimensions & coordinates) and charts to Excel
xlApp = Dispatch("Excel.Application")   # Conexiune Excel
wb_template = xlApp.Workbooks.Add()
XLSheet = wb_template.Worksheets(1)
Excel_file=os.getcwd()+'/'+'Elbow_Results.xlsx'

XLSheet.Range("A1").Value ="Main dimensions"
XLSheet.Range("A2").Value ="Diameter="+'%0.2f' % Diameter+" [mm]"
XLSheet.Range("A3").Value ="Elbow Radius="+'%0.2f' % Raza+" [mm]"
XLSheet.Range("A4").Value ="Gama angle="+'%0.2f' % float(Gama_Sel)+" [deg]"
XLSheet.Range("A5").Value ="Alfa angle="+'%0.2f' % float(Alfa_Sel)+" [deg]"
XLSheet.Range("A6").Value ="Elbow Elements No.="+'%2d' %  float(No)+" [-]"
XLSheet.Range("A7").Value ="K1 length="+'%0.2f' % K+" [mm]"
XLSheet.Range("A8").Value ="t length="+'%0.2f' % t+" [mm]"
XLSheet.Range("A9").Value ="H1="+'%0.2f' % H1+" [mm]"
XLSheet.Range("A10").Value ="H2="+'%0.2f' % H2+" [mm]"
XLSheet.Range("A11").Value ="Unfolded length Ld ="+'%0.2f' % Ld +" [mm]"
XLSheet.Range("A12").Value ="Area 1="+'%0.2f' % Aria1 +" [mm2]"
XLSheet.Range("A13").Value ="Area 2="+'%0.2f' % Aria2 +" [mm2]"

df= pd.DataFrame(
        {'Teta [deg]': Teta,
        'X [mm]': XP,
        'Y1 [mm]': YH1,
        'Y2a [mm]': YH2a,
        'Y2b [mm]': YH2b,
        })
coordinates = df.values.tolist()

String1="Teta"+"\t"+"X"+"\t"+"Y1"+"\t"+"Y2a"+"\t"+"Y2b"+"\t\n"
Put_Clipboard(String1)  ;  XLSheet.Cells(15,1).Select()  ;  XLSheet.Paste()

String=""
for teta, x, y1, y2a, y2b in coordinates:
        String=String+str(teta)+"\t"+str(x)+"\t"+str(y1)+"\t"+str(y2a)+"\t"+str(y2b)+"\t\n"
Put_Clipboard(String)  ;  XLSheet.Cells(16,1).Select()  ;  XLSheet.Paste()

self.figure.savefig("Charts.jpg", dpi=150)
PathImage = os.getcwd()+'/Charts.jpg'
data = open(PathImage, "rb").read()
stream = cStringIO.StringIO(data)   # Convert to a data stream  ;  require to import  cStringIO & StringIO
sel_bmp=wx.BitmapFromImage( wx.ImageFromStream( stream ))  # convert to a bitmap
required_width=sel_bmp.GetWidth() ; required_height=sel_bmp.GetHeight()
left=XLSheet.Cells(1,7).Left  ;  top=XLSheet.Cells(1,1).Top
XLSheet.Shapes.AddPicture(PathImage, False, True,left, top,required_width,required_height)
os.remove(PathImage)
XLSheet.Cells(1,1).Select()

if os.path.exists(Excel_file):
```

Listing 9.1 Elbow.pyw – Elbow unfold – Python & wxPython version

```
                        os.remove(Excel_file)
                wb_template.SaveAs(Excel_file)
                wb_template.Close()
                xlApp.Quit()
                self.statusbar.SetStatusText("Results saved in "+Excel_file,1)  # Add text to StatusBar

app = wx.App(redirect=False)   # wx.App initializes the GUI toolkit for WxPython.
window=Plot_Window(None, "The unfold components of an elbow")  # Call 'Plot_Window' class
window.Show()  # Show 'window' object
# This keeps the GUI in a continuous loop that is ready to receive key events
# from the user. It does not return until the program closes!
app.MainLoop()
```

The **Elbow.py** script (Python & Streamlit version) begin with importing libraries: **streamlit** (create a great interactive web application with Python), **PIL** (Python Imaging Library), **pandas** (open source data analysis and manipulation tool), **matplotlib** (library for creating static, animated and interactive visualizations in Python), **numpy** (the package for scientific computing with Python), **pathlib** (provide classes to represent abstract/concrete paths.

This script contains 145 lines. Lines 1 to 6 import the required modules. Line 8 defines the current folder. Line 9 defines the script title with **subheader** command. Line 11 inserts a container element (with **st.empty()** command) into the script that can be used to hold two columns defined in the same line 11; the **Elbow.jpg** image (**Figure 9.4**) is placed on column 1 and also define the width and caption of the image (lines 12); the **Elbow_Demo.jpg** image (**Figure 9.4**) is placed on column 2 and also define the width and caption of the image of the image (line 13).

Line 15 inserts a container element (with **st.empty()** command) into the script that can be used to hold three columns defined in line 15; line 16÷17 open and read the video file **Elbow.mp4'**, while line 18 place the video into median two column.

Line 20 define a list with possible angles of the elbow, list reversed in line 22; in first position of the list is inserted the text "**Select angle**".

Line 26 defines the title "**Input Data**" with **subheader** command. Line 27 creates columns with three elements. Lines 28÷29 define the fields of input data (**Diameter** & **Radius**) into the previous columns created (**col1** and **col2**), using **number_input** command.

Line 30 define a **selectbox control** from where the value of γ angle can be selected. Line 31 test if an γ angle is selected; after selection the following code will be executed.

Line 32 define the values of **α** angle. Line 33 initialize **Ualfa** list to "**Select angle**" value. Because, for a selected γ value only certain **α** angle values can be selected, into **Ualfa** list will be appended only the values that verify the condition from line 38; these values from **Ualfa** list will be transferred to **Alfa_Sel selectbox control** to permit the selection of one **α** angle.

Line 41 test if an **α** angle is selected; after selection the following code will be executed. Lines 42÷66 calculate the main dimension of elbow and drawings coordinates; lines 68÷92 draw the elbow geometry**, Figure 9.4**; lines 94÷105 draw the unfolded surface of element 1**, Figure 9.4**; lines 107÷119 draw the unfolded surface of element 2**, Figure 9.4**.

Line 122 creates columns with two elements, where main dimensions will be placed in **col1** (lines 123÷135). Lines 136÷142 define a dataframe **df** containing the coordinates of unfolded surfaces (element 1 & 2) placed in **col2** in line 144.

Listing 9.2 Elbow.py – Elbow unfold – Python & Streamlit version

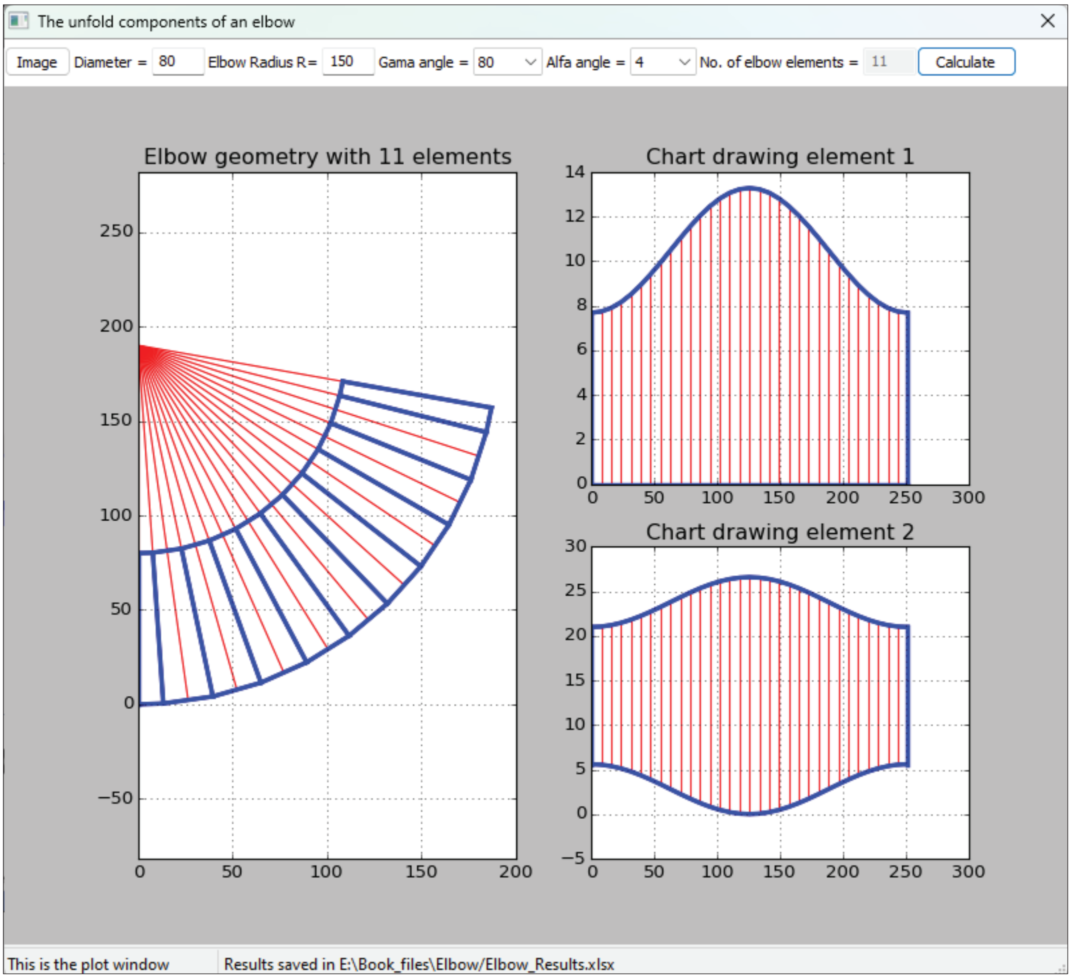
```
1.    import streamlit as st
2.    from PIL import Image
3.    import pandas as pd
4.    import matplotlib.pyplot as plt
5.    import numpy as np
6.    from pathlib import Path
7.
8.    current_dir = str(Path(__file__).parent if "__file__" in locals() else Path.cwd())
9.    st.subheader(":green[The unfold components of an elbow]")
10.
11.   col1, col2 = st.empty().columns([0.3,0.7])
```

Listing 9.2 Elbow.py – Elbow unfold – Python & Streamlit version

```
col1.image(Image.open(current_dir+"/Elbow.jpg"), width=200, caption='Cylindrical elbow')
col2.image(Image.open(current_dir+"/Elbow_unfolded.jpg"), width=440, caption='Unfolding demo')

col1, col2, col3 = st.empty().columns([0.1,0.8,0.1])
video_file = open(current_dir+'/Elbow.mp4', 'rb')
video_bytes = video_file.read()
col2.video(video_bytes, format="mp4", start_time=0)

Gama_Angles = ["24","25","28","30","32","33","35","36","40","42","44","45","48","49","50","54","55",
            "56","60","63","64","65","66","67.5","70","72","75","77","78","80","84","85","88","90"]
Gama_Angles.reverse()
Gama_Angles.insert(0, "Select angle")

st.divider()
st.subheader(':green[Input Data]')
col1, col2, col3 = st.columns(3)
Diameter = col1.number_input('Diameter [mm]', value=80.0)
Raza = col2.number_input('Elbow Radius R [mm]', value=150.0)
Gama_Sel=col3.selectbox("Gama angle [deg]",options=Gama_Angles)
if Gama_Sel != "Select angle":
    Alfa_Angles=["2.5","3","3.5","4","5","5.5","6","7.5","8","10","11.25"]
    Ualfa=["Select angle"]
    for alf in Alfa_Angles:
        n = (float(Gama_Sel)/float(alf)+2) / 2
        r = np.floor(n)
        if n == r and n>2 and n<=19:
            Ualfa.append(alf)
    col1, col2, col3 = st.columns(3)
    Alfa_Sel=col1.selectbox("Alfa angle [deg]",options=Ualfa)
    if Alfa_Sel != "Select angle":
        n1 = (float(Gama_Sel)/float(Alfa_Sel)+2) / 2
        r1 = np.floor(n1)
        if n1 == r1:
            No = col2.text_input('No. of elbow elements [-]', int(np.floor(n1)), disabled=True)
            col3.write("")
            s_btn = col3.button("Calculate",use_container_width=True)
            st.divider()
            if s_btn:   # Calculate elbow elements
                PI=np.arctan(1)*4
                NrPct = 8
                GamaR=float(Gama_Sel)*PI/180
                AlfaR=float(Alfa_Sel)*PI/180
                K = Raza * np.tan(GamaR / 2)
                t = Raza * np.tan(AlfaR)
                H1 = t - Diameter * np.tan(AlfaR) / 2
                H2 = t + Diameter * np.tan(AlfaR) / 2
                Ld = PI * Diameter   # Unfolded length
                Aria1 = PI * Diameter * (H2 - Diameter * (np.tan(AlfaR) + np.tan(0)) / 2)
                Aria2 = PI * Diameter * (2 * H2 - Diameter * (np.tan(AlfaR) + np.tan(0)) / 2)
                XP=[] ; Y1=[] ; Teta=[]
                for i in range(1, 4 * NrPct + 2):
                    UT = (i - 1) * PI / 2 / NrPct
                    Teta.append(UT)
                    XP.append(UT * Diameter/ 2)
                    Y1.append(Diameter / 2 * (1 - np.cos(UT)) * np.tan(AlfaR))

                fig=plt.figure()   # Elbow drawing
                plt.title("Elbow geometry with "+'%2d' %  float(No)+" elements", fontsize=14, fontweight='bold',color='Black')
                plt.grid(True)
                plt.axis('equal')
                X11=[] ; Y11=[] ; X22=[] ; Y22=[]
                cnt=0 ; RpD2 = Raza + Diameter/2 ;  RmD2 = Raza - Diameter/2
                for i in range(1, 2 * int(float(No)) ):
                    unghi = AlfaR * (i - 1)
                    plt.plot([0,RpD2 * np.sin(unghi)],[RpD2,RpD2-RpD2 * np.cos(unghi)], color='Red', linewidth=1)
                    if i==1 or (i % 2) == 0 or i == (2 * int(float(No)) - 2) + 1 :
                        x1=RpD2*np.sin(unghi) ; x2=RmD2*np.sin(unghi)
                        y1=RpD2-RpD2*np.cos(unghi) ; y2=RpD2-RmD2*np.cos(unghi)
                        plt.plot([x1,x2],[y1, y2], color='Blue', linewidth=3)
                        cnt+=1
                        X11.append(x1) ; Y11.append(y1) ; X22.append(x2) ; Y22.append(y2)
                for i in range(0, cnt):
```

Listing 9.2 Elbow.py – Elbow unfold – Python & Streamlit version

```
    if i==0:
        Xant1=X11[i] ; Yant1=Y11[i]
        Xant2=X22[i]  ; Yant2=Y22[i]
    else:
        plt.plot([Xant1,X11[i]],[Yant1, Y11[i] ], color='Blue', linewidth=3)
        plt.plot([Xant2,X22[i]],[Yant2, Y22[i] ], color='Blue', linewidth=3)
        Xant1=X11[i] ; Yant1=Y11[i]
        Xant2=X22[i]  ; Yant2=Y22[i]
st.empty().write(fig)

fig=plt.figure()  # Chart drawing element 1
plt.title("Unfolded element 1", fontsize=14, fontweight='bold',color='Black')
plt.grid(True)
YH1=[]
for i in range(0, 4 * NrPct+ 1):
    YH1.append(H1+Y1[i])
    plt.plot([XP[i], XP[i]],[0, YH1[i]], color='Red', linewidth=1)
plt.plot(XP,YH1, color='Blue', linewidth=3)
plt.plot([0,0],[0,H1], color='Blue', linewidth=3)
plt.plot([0,Ld],[0,0], color='Blue', linewidth=3)
plt.plot([Ld,Ld],[0,H1], color='Blue', linewidth=3)
st.empty().write(fig)

fig=plt.figure()   # Chart drawing element 2
plt.title("Unfolded element 2", fontsize=14, fontweight='bold',color='Black')
plt.grid(True)
YH2a=[] ; YH2b=[]
for i in range(0, 4 * NrPct+ 1):
    YH2a.append(H2-H1-Y1[i])
    YH2b.append(H2+H1+Y1[i])
    plt.plot([XP[i], XP[i]],[YH2a[i], YH2b[i]], color='Red', linewidth=1)
plt.plot(XP,YH2a, color='Blue', linewidth=3)
plt.plot(XP,YH2b, color='Blue', linewidth=3)
plt.plot([0, 0],[H2-H1, H2+H1], color='Blue', linewidth=3)
plt.plot([Ld,Ld],[H2-H1, H2+H1], color='Blue', linewidth=3)
st.empty().write(fig)

st.divider()   # Chart dimensions & coordinates
col1, col2 = st.columns([0.35,0.65])
col1.subheader(":green[Main dimensions]")
col1.write("Diameter="+'%0.2f' % Diameter+" [mm]")
col1.write("Elbow Radius="+'%0.2f' % Raza+" [mm]")
col1.write("Gama angle="+'%0.2f' % float(Gama_Sel)+" [deg]")
col1.write("Alfa angle="+'%0.2f' % float(Alfa_Sel)+" [deg]")
col1.write("Elbow Elements No.="+'%2d' %  float(No)+" [-]")
col1.write("K1 length="+'%0.2f' % K+" [mm]")
col1.write("t length="+'%0.2f' % t+" [mm]")
col1.write("H1="+'%0.2f' % H1+" [mm]")
col1.write("H2="+'%0.2f' % H2+" [mm]")
col1.write("Unfolded length Ld ="+'%0.2f' % Ld +" [mm]")
col1.write("Area 1="+'%0.2f' % Aria1 +" [mm2]")
col1.write("Area 2="+'%0.2f' % Aria2 +" [mm2]")
df= pd.DataFrame(
    {'Teta [deg]': Teta,
    'X [mm]': XP,
    'Y1 [mm]': YH1,
    'Y2a [mm]': YH2a,
    'Y2b [mm]': YH2b,
    })
col2.subheader(":green[Unfolded Coordinates]")
col2.dataframe(df)
st.divider()
```

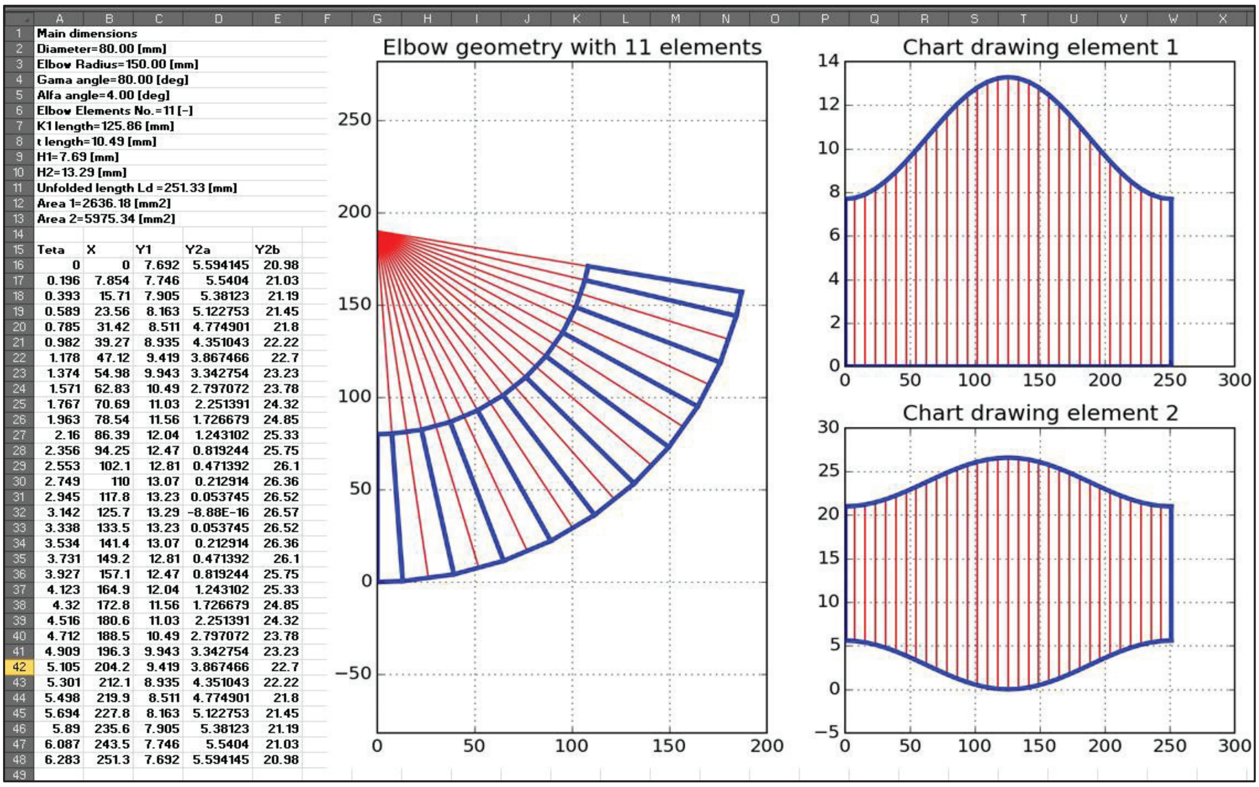


Diameter = 80 Elbow Radius R= 150 Gama angle = 80 Alfa angle = 4 No. of elbow elements = 11

This is the plot window

Results saved in E:\Book_files\Elbow/Elbow_Results.xlsx

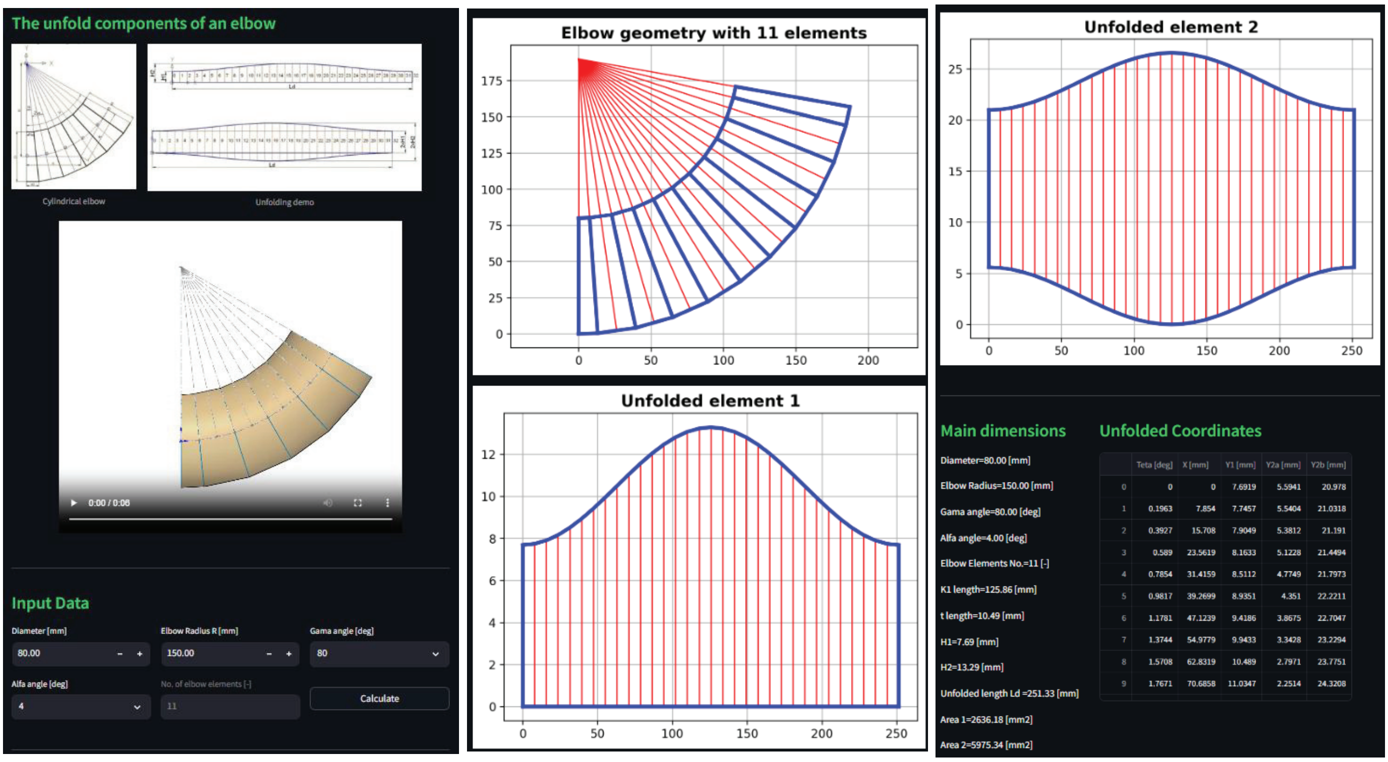
Figure 9.2 The interface and screen generated by Python & wxPython version of the **Elbow.pyw** script



Main dimensions
Diameter=80.00 [mm]
Elbow Radius=150.00 [mm]
Gama angle=80.00 [deg]
Alfa angle=4.00 [deg]
Elbow Elements No.=11 [-]
K1 length=125.86 [mm]
t length=10.49 [mm]
H1=7.69 [mm]
H2=13.29 [mm]
Unfolded length Ld =251.33 [mm]
Area 1=2636.18 [mm2]
Area 2=5975.34 [mm2]

Teta	X	Y1	Y2a	Y2b
0	0	7.692	5.594145	20.98
0.196	7.854	7.746	5.5404	21.03
0.393	15.71	7.905	5.38123	21.19
0.589	23.56	8.163	5.122753	21.45
0.785	31.42	8.511	4.774901	21.8
0.982	39.27	8.935	4.351043	22.22
1.178	47.12	9.419	3.867466	22.7
1.374	54.98	9.943	3.342754	23.23
1.571	62.83	10.49	2.797072	23.78
1.767	70.69	11.03	2.251391	24.32
1.963	78.54	11.56	1.726679	24.85
2.16	86.39	12.04	1.243102	25.33
2.356	94.25	12.47	0.819244	25.75
2.553	102.1	12.81	0.471392	26.1
2.749	110	13.07	0.212914	26.36
2.945	117.8	13.23	0.053745	26.52
3.142	125.7	13.29	-8.88E-16	26.57
3.338	133.5	13.23	0.053745	26.52
3.534	141.4	13.07	0.212914	26.36
3.731	149.2	12.81	0.471392	26.1
3.927	157.1	12.47	0.819244	25.75
4.123	164.9	12.04	1.243102	25.33
4.32	172.8	11.56	1.726679	24.85
4.516	180.6	11.03	2.251391	24.32
4.712	188.5	10.49	2.797072	23.78
4.909	196.3	9.943	3.342754	23.23
5.105	204.2	9.419	3.867466	22.7
5.301	212.1	8.935	4.351043	22.22
5.498	219.9	8.511	4.774901	21.8
5.694	227.8	8.163	5.122753	21.45
5.89	235.6	7.905	5.38123	21.19
6.087	243.5	7.746	5.5404	21.03
6.283	251.3	7.692	5.594145	20.98

Figure 9.3 The Excel file content generated by **Elbow.pyw** script



Input Data

Diameter [mm]: 80.00

Elbow Radius R [mm]: 150.00

Gama angle [deg]: 80

Alfa angle [deg]: 4

No. of elbow elements [-]: 11

Calculate

Main dimensions

Diameter=80.00 [mm]

Elbow Radius=150.00 [mm]

Gama angle=80.00 [deg]

Alfa angle=4.00 [deg]

Elbow Elements No.=11 [-]

K1 length=125.86 [mm]

t length=10.49 [mm]

H1=7.69 [mm]

H2=13.29 [mm]

Unfolded length Ld =251.33 [mm]

Area 1=2636.18 [mm2]

Area 2=5975.34 [mm2]

Unfolded Coordinates

	Teta [deg]	X [mm]	Y1 [mm]	Y2a [mm]	Y2b [mm]
0	0	0	7.6919	5.5941	20.978
1	0.1963	7.854	7.7457	5.5404	21.0318
2	0.3927	15.708	7.9049	5.3812	21.191
3	0.589	23.5619	8.1633	5.1228	21.4494
4	0.7854	31.4159	8.5112	4.7749	21.7973
5	0.9817	39.2699	8.9351	4.351	22.2211
6	1.1781	47.1239	9.4186	3.8675	22.7047
7	1.3744	54.9779	9.9433	3.3428	23.2294
8	1.5708	62.8319	10.489	2.7971	23.7751
9	1.7671	70.6858	11.0347	2.2514	24.3208

Figure 9.4 The interface and screen generated by Python & Streamlit version of the **Elbow.py** script

10. Flatten surface

The **Surfaces** script is an application to flatten a surface defined by two 3D curve. Engineering has many examples of using this type of 3D geometry, as in **Figure 10.1, Figure 10.2**.

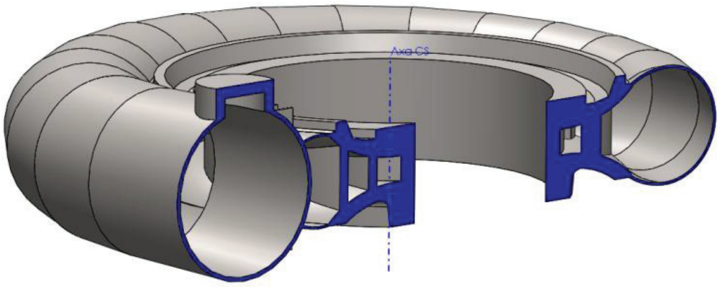


Figure 10.1 The spiral casing geometry of hydraulic turbine [10.1]

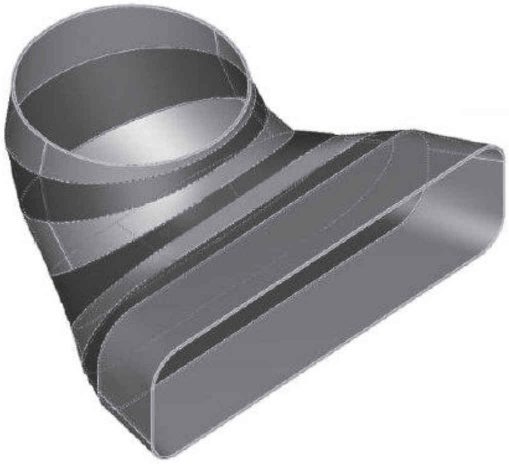


Figure 10.2 The draft tube geometry of hydraulic turbine [10.2]

The surface defined by two 3D curve with thickness=5 mm, shown in **Figure 10.3**, is symmetrical and has been cut in half by **Top plane** as in **Figure 10.4**. **Figure 10.5** show the mid surfaces, with thickness=0 mm, created between upper and lower faces. Technically, since such an area cannot be flatten, still the unfolded surface can be approximated by dividing the spatial area into triangles, **Figure 10.6**, and flattening them in the same plan.

The flatten steps are the following:

- Open an Excel file to read the coordinates of the two 3D curves; the template of the curve coordinates is shown in **table 10.1**.
- Based on these coordinates, a spatial spline interpolation is performed; this is done in order to obtain more points for a better approximation of the flattened surface;
- The script generates a 3D figure with the two curves joined by triangles covering the surface;
- The script calculate the coordinates of the flatten surface and draw, in a separate figure, the flatten geometry;
- The coordinates of the flatten surface and the two drawings can be exported to an Excel file.

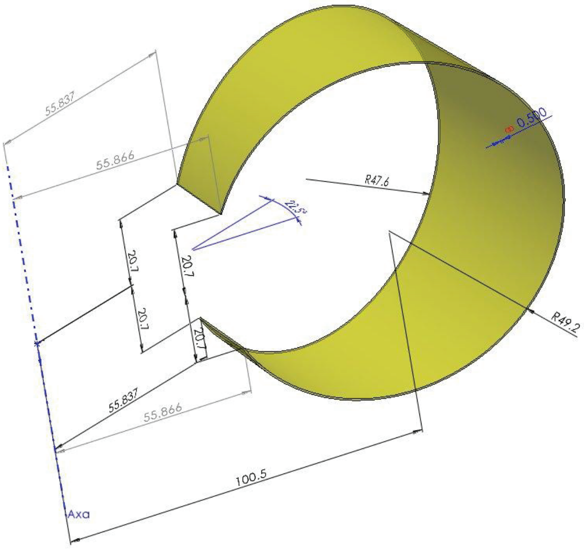


Figure 10.3 The surface defined by two 3D curve (thickness=5 mm)

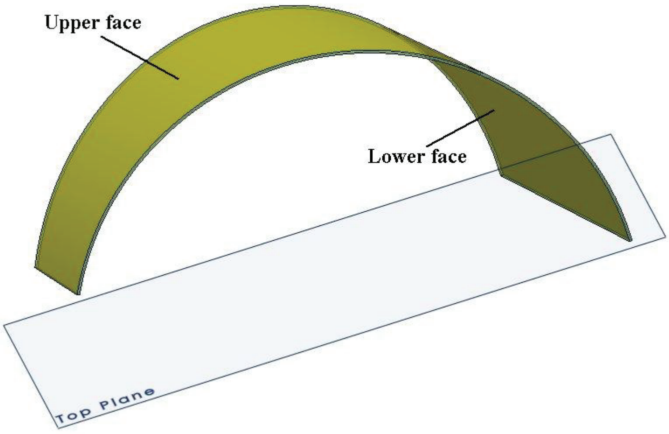


Figure 10.4 The half surface defined by two 3D curve (thickness=5 mm)

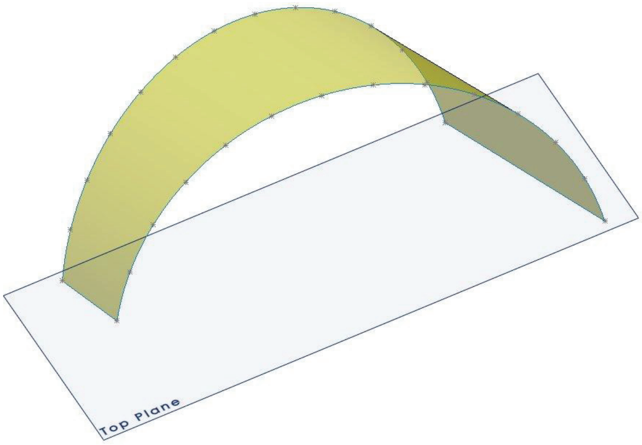


Figure 10.5 The mid surfaces between upper and lower faces (thickness=0 mm)

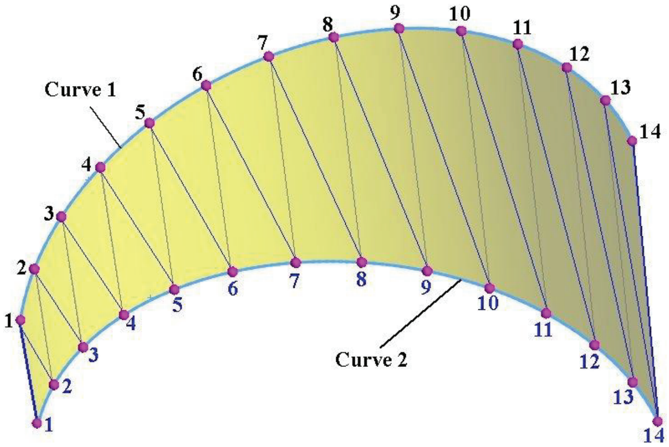


Figure 10.6 The mid surface covered by triangles

Table 10.1

	A	B	C	D	E	F	G	H	I
1	Section 1					Section 2			
2		X	Y	Z			X	Y	Z
3	1	51.36	-20.81	21.32		1	55.64	-20.80	-0.04
4	2	56.17	-29.22	23.31		2	60.91	-29.63	-0.04
5	3	62.47	-36.39	25.91		3	67.89	-37.17	-0.03
6	4	69.99	-41.99	29.02		4	76.28	-43.11	-0.02
7	5	78.43	-45.81	32.51		5	85.72	-47.19	-0.01
8	6	87.40	-47.68	36.22		6	95.79	-49.23	0.01
9	7	96.54	-47.50	39.99		7	106.07	-49.14	0.02
10	8	105.45	-45.29	43.67		8	116.11	-46.92	0.03
11	9	113.75	-41.15	47.10		9	125.47	-42.68	0.04
12	10	121.09	-35.25	50.13		10	133.76	-36.59	0.05
13	11	127.14	-27.84	52.64		11	140.60	-28.92	0.05
14	12	131.67	-19.25	54.51		12	145.72	-20.00	0.06
15	13	134.46	-9.84	55.66		13	148.88	-10.22	0.06
16	14	135.41	0.00	56.05		14	149.94	0.00	0.06

The **Surfaces** script use a database **Config.db** to store all icons loaded in frame **Toolbar**; this will avoid the existence of following icons files in the application folder: 📂, ▦ and ⇥. The **Config.db → objects** store icons for the script interface of **Python 2. 7 & wxPython** application version. Initially, the icons files are stored in the **Flatten surface→ Objects** application folder. The icons files were loaded into **Config.db → objects** as **Binary Large Object** (**BLOB**) through the **_copyimg.pyw** script displayed in **Listing 8.1**, from where they will be used into application interface through **Python 2.7 → Surfaces.pyw** script.

The **Surfaces.pyw** script (Python & wxPython version) include three classes: **Plot**, **PlotNotebook**, **Flatten_surface** and two public functions: **ExportImageToExcel**, **ExtragImageMemory**. The script begin with importing libraries (lines 3÷11).

The **Plot** class (lines 34÷44) create **self.figure**, **self.canvas**, **self.toolbar** and **sizer** entities for the class that calls it. The **PlotNotebook** class (lines 46÷60) creates several pages for the class that calls it, using the internal function **add**; through internal function **ChangePagina** the user can change the page. This class represents a notebook control, which manages multiple windows with associated tabs.

The **ExportImageToExcel** public function export an image Excel file. The parameters of the function are: the file **Path** – the path were the image file is saved, **Sheet** – the sheet were the image will be placed and **row** & **column** which identify the cell were the image will be placed in Excel sheet.

The **ExtragImageMemory** function (lines 28÷32) selecteaza din **SQLite database → Objects** an icon si create & returneaza the icon **img** from data in memory. This function is called from class **Flatten_surface → _TOOLBAR** function.

The **Flatten_surface** class create the interface (toolbar, grid, …), create the aplication main functionalities and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user); create the statusbar (lines 67÷69); line 71 create in **self.plotter** an instance of **PlotNotebook** class; lines 73, 74 create one 3D graphics subplot **self.Axa_3D**; line 75 create in **self.Axa2D_Flatten** a new instance of **PlotNotebook** class; lines 76, 77 create one 2D graphics subplot **self.Axa2D** that will be connected to **OnMouseMotion** class internal function (line 77); line 79 initialize the public variables of the class **self.Excel_File** and **self.xad**, **self.yad**, **self.xbd**, **self.ybd**, which are the coordinates of flatten surfaces; line 80 call **_TOOLBAR**; line 81 read the number of the spline interpolation points from **self.txt_Points** text control placed on **self.toolbar**; line 82 initialize the 3D curves coordinates **self.X1**, **self.Y1**, **self.Z1**, **self.X2**, **self.Y2**, **self.Z2**, that will be readed from an Excel file on **OnOpen** function; line 83 call **RecreateAxis** function;
  lines 84 center and show the frame in the display.
- **_TOOLBAR** Create the **toolbar** (line 87); set the **toolbar** background color (line 88) and icons size (line 89); create the connection and cursor using **Config.db** database (line 90); generate **Lst_label**, **Lst_icons** and **Lst_Help** lists (lines 91÷93), which store the **tooolbar** icons labels, the icon names that must be extracted from memory by calling **ExtragImageMemory** function and the associated icons help; for every line 95 associate an unique identificator ID; through **for** cycle from line 94 the **img** icon is extracted into memory (line 96); create the bitmap of **img** memory icon and scale at the imposed **toolbar** icons size (line 97); in this cycle, for every operation, the icon is added in **toolbar** (line 98) ; lines 100÷109 connect with **Bind** statement to associated functions; lines 102÷105 create **self.txt_Points** text control to input the number of the spline interpolation points; this control is connected to **EvtTextEnter** class internal function (line 106); line 110 generate and show the **toolbar;** line 111 close the connection with **Config.db** database.
- **EvtTextEnter** This function is activated from **self.txt_Points** text control after **Enter** key was pressed; lines 114÷117 verify if the curves coordinates was readed through **self.X1** variable; line 118 call **RecreateAxis** function; line 119 read the value from **self.txt_Points** text control in **self.Ninterp** public variable; line 119÷120 define and update the title of **self.Axa_3D**; line 122 call **Calculation** function to drawn the curves and triangles, based on the 3D curves coordinates.
- **RecreateAxis** This function define the parameters of the two graphics area: **self.Axa_3D** and **self.Axa2D**.
- **OnOpen** Open a dialog to load an Excel file (line 157) and store the filename in variable **Excel_File** (line 155); read the curve coordinates (lines 164÷169); close the Excel file (line 171) and call **Calculation** function. The content of Excel file is shown in **table 10.1**, where the coordinates start from line 3.
- **Calculation** This function compute the spline interpolation curve 1 & 2 (lines 189÷191 & 194÷195) and draw the following geometry in **self.Axa_3D**: section 2 curve, points & text (lines 176÷180), section 1 curve, points & text (lines 182÷186), spline interpolate curve 1 & 2 (lines 192, 196) , straight bending lines (lines 198, 199) and diagonal bending lines (lines 201÷204); the left pressed mouse button rotate the 3D geometry, while the right pressed mouse button redraw the geometry at different levels of zoom; call **Flatten** function.
- **ALFA** This function calculates the α angle of triangles used in flattening calculations (**Flatten** function) to correctly align one triangle with the next.
- **Flatten** This is the function that calculates the coordinates of the flattened surface: **self.xad**,

self.yad, **self.xbd**, **self.ybd**.

- **Draw_Flatten** This function draw the flatten surface in **self.Axa2D** chart: section 1, 2 curve, points, stright bending lines and diagonal bending lines.
- **OnMouseMotion** Identify the current mouse position (line 162); only for **self.Axa2D** write these coordinates in statusbar (line 292).
- **OnExcel** Lines 296÷299 verify if the curves coordinates was readed through **self.xbd** variable; open **self.Excel_File** file (line 301) and activate sheet 1 (line 302) ; create Excel table coordinates headings (line 303÷304) and write in a **for** loop the flatten surface coordinates (**self.yad**, **self.xbd**, **self.ybd**) (lines 306÷309); lines 311÷313 save the **self.Axa_3D** and **self.Axa2D** as image files which is exported to Excel file (lines 315÷316) using **ExportImageToExcel** function; line 317 save and close the Excel file and reopen & shown by line 318. The saving of the coordinates and drawings will be performed on Excel sheet 2; for this reason, the file must have at least two sheets: the coordinates of the curves are read from the first sheet and the numerical & graphical results are stored in the second sheet.
- **OnClose** Exit from application.

The interface of **Surfaces.pyw** script is displayed in **Figure 10.7** and **Figure 10.8**. **Listing 10.1** displays the **Python & wxPython** version of script **Surfaces.pyw**.

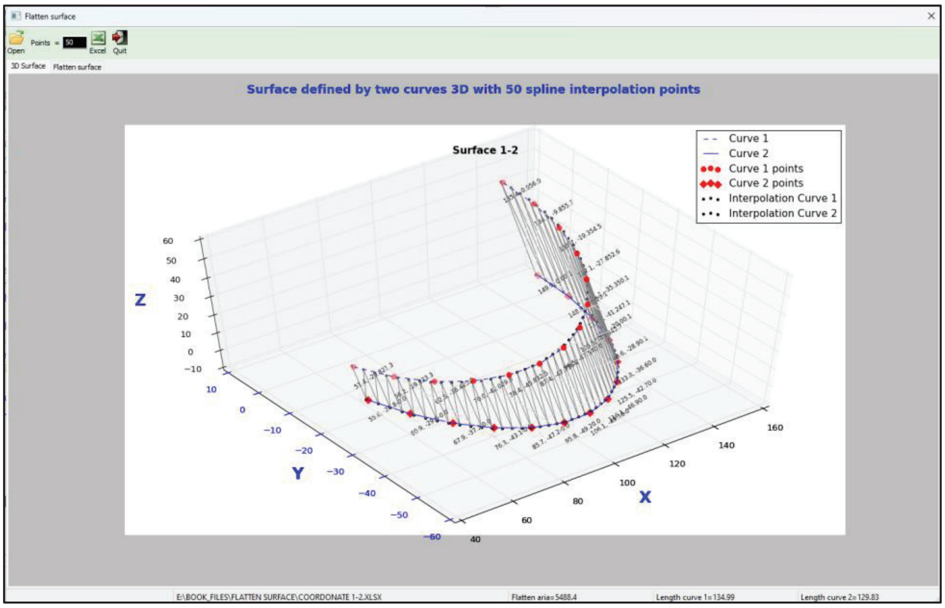
The **Surface.py** script (Python & Streamlit version) contains 171 lines and begin with importing libraries (lines 1÷8). Lines 10÷15 configures the title settings of the script and the max-width of the page (70 rem = 1120 px, where 1 rem = 16px). Lines 17 define the the script path.

Lines 19÷27 define **ALFA** function which calculate the α angle of triangles used in flattening calculations to correctly align one triangle with the next.

Line 31 wait to select an Excel file by the user to read the curves points coordinates in 6 lists (line 44÷54). The coordinates readed from Excel will be stored in a dataframe **df** (line 55) and displayed as table and the Excel file name is also displayed (**Figure 10.9**).

Line 58 create a form to input a value **Ninterp variable** (number of spline interpolation points - lines 59÷61) Also, a **submit_button** is created (line 62). If the button is clicked, lines 64÷95 draw the following geometry in **fig**: section 1 curve, points & text, section 2 curve, points & text, spline interpolate curve 1 & 2, straight bending lines and diagonal bending lines. Lines 97÷144 compute the flatten surface coordinates. Lines 145÷163 draw the flatten surface. Lines 164÷171 show the numerical results.

The interface of **Surface.py** script is displayed in **Figure 10.9**. **Listing 10.2** displays the **Python & Streamlit** version of script **Surface.py**.



E:\BOOK_FILES\FLATTEN SURFACE\COORDONATE 1-2.XLSX Flatten aria= 5488.4 Length curve 1= 134.99 Length curve 2= 129.83

Figure 10.7 The 3D geometry of curves and triangles

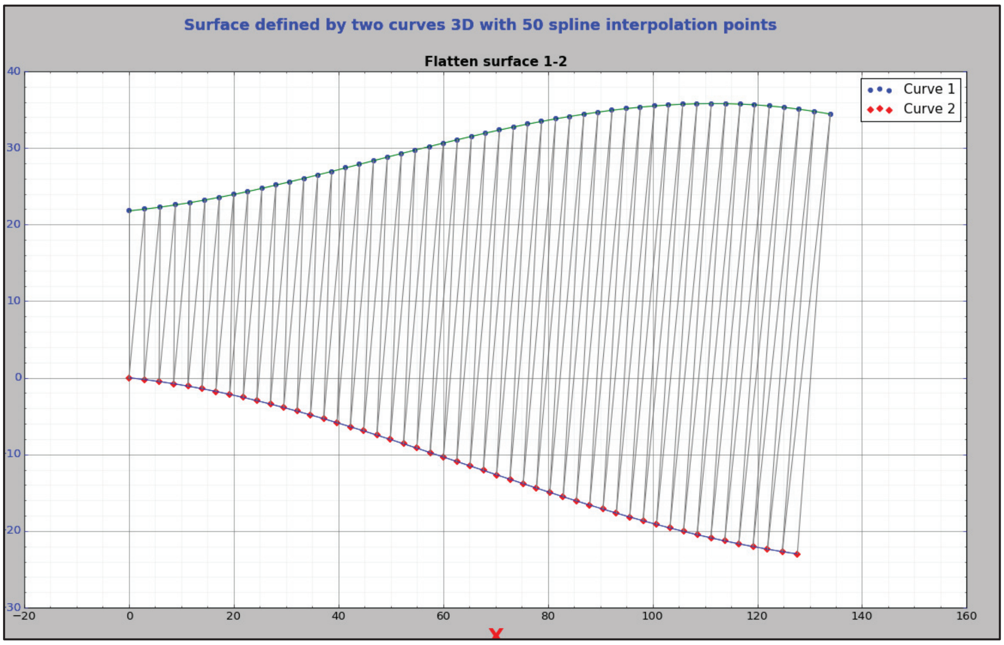


Figure 10.8 The flatten surface

Listing 10.1 Surfaces.pyw – Flatten surface – Python & wxPython version

```
 from __future__ import division

from win32com.client import Dispatch
from pysqlite2 import dbapi2 as sqlite
from scipy import interpolate
import StringIO,cStringIO
import wx , os, math
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as Canvas
from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar

def ExportImageToExcel(PathImage, Sheet, row, column):
        data = open(PathImage, "rb").read()
        stream = cStringIO.StringIO(data)   # convert to a data stream
        sel_bmp=wx.BitmapFromImage( wx.ImageFromStream( stream ))  # convert to a bitmap
        width=sel_bmp.GetWidth() ; height=sel_bmp.GetHeight()
        left=Sheet.Cells(1,column).Left  ;  top=Sheet.Cells(row,1).Top
        Sheet.Shapes.AddPicture(PathImage, False, True,left, top,width,height)
        os.remove(PathImage)
        return

def ExtragImageMemory(cursor, NumeImage):
        cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
        blob=cursor.fetchone()[0]
        img = wx.ImageFromStream(StringIO.StringIO(blob))
        return img

class Plot(wx.Panel):
        def __init__(self, parent, id = -1, dpi = None, **kwargs):
                wx.Panel.__init__(self, parent, id=id, **kwargs)
                self.figure = mpl.figure.Figure(dpi=dpi)
                self.canvas = Canvas(self, -1, self.figure)
                self.toolbar = Toolbar(self.canvas)
                self.toolbar.Realize() ; self.toolbar.Hide()
                sizer = wx.BoxSizer(wx.VERTICAL) ;
                sizer.Add(self.canvas,1,wx.EXPAND)
                sizer.Add(self.toolbar, 0 , wx.LEFT | wx.EXPAND)
                self.SetSizer(sizer)

class PlotNotebook(wx.Panel):
        def __init__(self, parent, id = -1):
```

Listing 10.1 Surfaces.pyw – Flatten surface – Python & wxPython version

```
        wx.Panel.__init__(self, parent, id=id)
        self.nb = wx.Notebook(self)
        sizer = wx.BoxSizer() ; sizer.Add(self.nb, 1, wx.EXPAND)
        self.SetSizer(sizer)
    def add(self,name="plot"):
        page = Plot(self.nb)
        self.nb.AddPage(page,name)
        return page.figure
    def add_PAGE(self,name):
        Pagina=wx.Panel(self.nb) ; self.nb.AddPage(Pagina, name)
        return Pagina
    def hide_PAGE(self,number):
        self.nb.RemovePage(number)
    def ChangePagina(self,INDEX):
        self.nb.ChangeSelection(INDEX)

class Flatten_surface(wx.Frame):  # ===============================================
    def __init__(self):
        displaySize= self.DS=wx.DisplaySize()
        wx.Frame.__init__(self, None, -1, title="Flatten surface",
                size=(displaySize[0]*0.75, displaySize[1]*0.85),
                style=wx.DEFAULT_FRAME_STYLE ^ ( wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
        self.SB=self.SB = self.CreateStatusBar()   # StatusBar
        self.SB.SetFieldsCount(5) ; self.SB.SetStatusWidths([-35,-70,-30, -30, -30])
        self.SB.SetStatusText("",1)

        self.plotter = PlotNotebook(self)  # Create Notebook
        mpl.rcParams['legend.fontsize'] = 14
        self.Axa_3D_Surface = self.plotter.add("3D Surface").gca()
        self.Axa_3D = self.Axa_3D_Surface.figure.add_subplot(1, 1, 1, projection='3d')
        self.Axa2D_Flatten = self.plotter.add("Flatten surface").gca()
        self.Axa2D = self.Axa2D_Flatten.figure.add_subplot(1, 1, 1)
        self.Axa2D.figure.canvas.mpl_connect('motion_notify_event', self.OnMouseMotion)

        self.Excel_File="" ; self.xad=[] ; self.yad=[] ; self.xbd=[] ; self.ybd=[]
        self._TOOLBAR()
        self.Ninterp=int(self.txt_Points.GetValue().strip())
        self.X1=[] ; self.Y1=[] ; self.Z1=[] ; self.X2=[] ; self.Y2=[] ; self.Z2=[]
        self.RecreateAxis()
        self.CenterOnScreen() ; self.Show() # self.Maximize(True)

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))
        self.toolbar.SetToolBitmapSize((24,24))
        conn = sqlite.connect(os.getcwd()+'\Config.db') ; cursor = conn.cursor()
        Lst_label=["Open","Excel","Quit"]
        Lst_icons=["open.png","Excel.png", "exit.png"]
        Lst_Help=["Open image file.","Export coordinates & charts to Excel","Quit application"]
        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            ID_tool= wx.NewId()
            img = ExtragImageMemory(cursor, icon)
            sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
            self.toolbar.AddLabelTool(ID_tool, label, sel_bmp, shortHelp=HELP)
            if label=="Open":
                self.Bind(wx.EVT_TOOL, self.OnOpen, id=ID_tool)
                st1 = wx.StaticText(self.toolbar, -1, "  Points  = ")
                self.txt_Points = wx.TextCtrl(self.toolbar, value="50", size=(40,-1))
                self.txt_Points.SetBackgroundColour('Black')
                self.txt_Points.SetForegroundColour("White")
                self.txt_Points.SetToolTipString("No. of spline interpolation points\nChange and hit ENTER key
!")
                self.Bind(wx.EVT_TEXT_ENTER, self.EvtTextEnter, self.txt_Points)
                self.toolbar.AddControl(st1) ; self.toolbar.AddControl(self.txt_Points)
            if label=="Excel": self.Bind(wx.EVT_TOOL, self.OnExcel, id=ID_tool)
            if label=="Quit": self.Bind(wx.EVT_TOOL, self.OnClose, id=ID_tool)
        self.toolbar.Realize() ; self.toolbar.Show()  # Generate toolbar
        cursor.close() ; conn.close()

    def EvtTextEnter(self, event):
        if len(self.X1)==0:
            mesaj = "Please open an Excel file with sections coordinates !"
```

Listing 10.1 Surfaces.pyw – Flatten surface – Python & wxPython version

```
            wx.MessageBox(mesaj, "Info",style=wx.OK|wx.ICON_EXCLAMATION)
            return
        self.RecreateAxis()
        self.Ninterp=int(self.txt_Points.GetValue().strip())
        sir="Surface defined by two curves 3D with "+str(self.Ninterp)+" spline interpolation points"
        self.Axa_3D.figure.suptitle(sir, fontsize=16, fontweight='bold', color="Blue")
        self.Calculation()

    def RecreateAxis(self):
        self.plotter.ChangePagina(0) ; self.Ninterp=int(self.txt_Points.GetValue().strip())
        self.Axa_3D.clear() ; param=dict()
        self.Axa_3D.tick_params(axis='y', colors='Blue')
        self.Axa_3D.grid(b=True, which='major', color='#666666', linestyle='-')
        self.Axa_3D.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
        self.Axa_3D.minorticks_on()
        self.Axa_3D.set_xlabel("X", fontsize=22, fontweight='bold',  color="Blue")
        self.Axa_3D.set_ylabel("Y", fontsize=22, fontweight='bold',  color="Blue")
        self.Axa_3D.set_zlabel("Z", fontsize=22, fontweight='bold',  color="Blue")
        self.Axa_3D.set_title("Surface 1-2", fontsize=14, fontweight='bold')
        self.Axa_3D.view_init(elev=55., azim=-127)
        self.Axa_3D.figure.canvas.draw()
        self.Axa2D.clear()
        #  self.Axa2D.set_xticklabels([]) ; self.Axa2D.set_yticklabels([])
        self.Axa2D.tick_params(axis='y', colors='Blue')
        self.Axa2D.grid(b=True, which='major', color='#666666', linestyle='-')
        self.Axa2D.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
        self.Axa2D.minorticks_on()
        self.Axa2D.set_xlabel("X", fontsize=22, fontweight='bold', color="Red")
        self.Axa2D.set_ylabel("Y", fontsize=22, fontweight='bold', color="Red")
        self.Axa2D.set_title("Flatten surface 1-2", fontsize=14, fontweight='bold')
        sir="Surface defined by two curves 3D with "+str(self.Ninterp)+" spline interpolation points"
        self.Axa_3D.figure.suptitle(sir, fontsize=16, fontweight='bold', color="Blue")
        self.Axa2D.figure.suptitle(sir, fontsize=16, fontweight='bold', color="Blue")
        self.Axa2D.figure.canvas.draw()

    def OnOpen(self,event):
        wildcard="Excel files (*.xlsx)|*.xlsx|Excel files (*.xls)|*.xls"
        dialog = wx.FileDialog(None, "Open Excel file to read sections coordinates", "", "", wildcard, wx.OPEN)
        if dialog.ShowModal() == wx.ID_CANCEL: return
        self.Excel_File= dialog.GetPath().strip().upper()
        self.SB.SetStatusText(self.Excel_File,1)
        xlApp = Dispatch("Excel.Application")   # Connect to Excel
        xlApp.Visible=False ; xlWb = xlApp.Workbooks.Open(self.Excel_File)
        sht1 = xlWb.Worksheets(1) ;  lex=3
        while True:
            if str(sht1.Cells(lex,2).Value).strip()=="None":
                break
            else:
                self.X2.append( round(float(str(sht1.Cells(lex,2).Value)),3) )
                self.Y2.append( round(float(str(sht1.Cells(lex,3).Value)),3) )
                self.Z2.append( round(float(str(sht1.Cells(lex,4).Value)),3) )
                self.X1.append( round(float(str(sht1.Cells(lex,7).Value)),3) )
                self.Y1.append( round(float(str(sht1.Cells(lex,8).Value)),3) )
                self.Z1.append( round(float(str(sht1.Cells(lex,9).Value)),3) )
                lex+=1
        xlWb.Close(SaveChanges=1) ; xlApp.Quit()
        self.Calculation()

    def Calculation(self):
        # Draw section 1 curve, points & text
        self.Axa_3D.plot(self.X2, self.Y2, self.Z2, 'b--', linewidth=1, label='Curve 2')
        self.Axa_3D.scatter(self.X2, self.Y2, self.Z2, s=50, marker="8", color="Red", label='Curve 2 points')
        for i, (Xt, Yt, Zt) in enumerate(zip(self.X2, self.Y2, self.Z2)):
            sir= '%0.1f' % Xt +", "+ '%0.1f' % Yt + '%0.1f' % Zt
            self.Axa_3D.text(Xt, Yt, Zt, sir, (1,0,0), size=8, zorder=1, color='k')
        # Draw section 2 curve, points & text
        self.Axa_3D.plot(self.X1, self.Y1, self.Z1, 'b-', linewidth=1, label='Curve 1')
        self.Axa_3D.scatter(self.X1, self.Y1, self.Z1, s=50, marker="D", color="Red", label='Curve 1 points')
        for i, (Xt, Yt, Zt) in enumerate(zip(self.X1, self.Y1, self.Z1)):
            sir= '%0.1f' % Xt +", "+ '%0.1f' % Yt + '%0.1f' % Zt
            self.Axa_3D.text(Xt, Yt, Zt, sir, (1,0,0), size=8, zorder=1, color='k')
            #   bbox = dict(boxstyle = 'round,pad=0.3', fc = 'yellow', alpha = 0.3))
```

Listing 10.1 Surfaces.pyw – Flatten surface – Python & wxPython version

```
        # Spline interpolate curve 2
        pnts = np.linspace(0,1,self.Ninterp)
        tck2, u2 = interpolate.splprep([self.X2, self.Y2, self.Z2], s=2)
        self.Xs2, self.Ys2, self.Zs2 = interpolate.splev(pnts, tck2)
        self.Axa_3D.scatter(self.Xs2, self.Ys2, self.Zs2, 'o', c="Black", s=10,label="Interpolation Curve 2")
        # Spline interpolate curve 1
        tck1, u1 = interpolate.splprep([self.X1, self.Y1, self.Z1], s=2)
        self.Xs1, self.Ys1, self.Zs1 = interpolate.splev(pnts, tck1)
        self.Axa_3D.scatter(self.Xs1, self.Ys1, self.Zs1, 'o', c="Black", s=10,label="Interpolation Curve 1")
        # Straight bending lines
        for i, (x1, y1, z1, x2, y2, z2) in enumerate(zip(self.Xs1,self.Ys1,self.Zs1,self.Xs2,self.Ys2,self.Zs2)):
            self.Axa_3D.plot([x1,x2],[y1,y2],[z1,z2], color="Gray")
        # Diagonal bending lines
        for i, (x1a, y1a, z1a) in enumerate(zip(self.Xs1,self.Ys1,self.Zs1)):
            if i< len(self.Xs2) - 1:
                x2b, y2b, z2b = self.Xs2[i+1],self.Ys2[i+1],self.Zs2[i+1]
                self.Axa_3D.plot([x1a,x2b],[y1a,y2b],[z1a,z2b], color="Gray")
        self.Axa_3D.legend(fontsize=12)  ; self.Axa_3D.legend(loc='best')
        self.Axa_3D.figure.canvas.draw() ; self.Axa2D.figure.canvas.draw()
        self.Flatten()

    def ALFA(self, xdsf1, xdsf2, ydsf1, ydsf2):
        if (xdsf2-xdsf1)==0:
            alfaperp=0
        else:
            mcom=(ydsf2-ydsf1)/(xdsf2-xdsf1) ; mperp=-1/mcom ; alfaperp=math.atan(mperp)
        if alfaperp<0:  alfaperp=math.pi-abs(alfaperp)
        coefi=1
        if alfaperp*180/math.pi>90: alfaperp=math.pi-alfaperp ; coefi=-1
        return [coefi, alfaperp]

    def Flatten(self):
        self.ARIA=0
        for i, (x1, y1, z1, x2, y2, z2) in enumerate(zip(self.Xs1,self.Ys1,self.Zs1,self.Xs2,self.Ys2,self.Zs2)):
            if i==0:
                lcom=math.sqrt((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)
                xdsf1=0 ; ydsf1=0 ; xdsf2=0 ; ydsf2=lcom ; yaxa=lcom
                self.xad.append(0) ; self.yad.append(0) ; self.xbd.append(0) ; self.ybd.append(lcom)
            else: #--------------------------------
                xdsf1=self.xad[i-1] ; ydsf1=self.yad[i-1] ; xdsf2=self.xbd[i-1] ; ydsf2=self.ybd[i-1]
                lcom=math.sqrt((xdsf1-xdsf2)**2+(ydsf1-ydsf2)**2)
                l22=math.sqrt((x1-self.Xs1[i-1])**2+(y1-self.Ys1[i-1])**2+(z1-self.Zs1[i-1])**2)
                l21=math.sqrt((self.Xs2[i-1]-x1)**2+(self.Ys2[i-1]-y1)**2+(self.Zs2[i-1]-z1)**2)
                sp=(lcom+l22+l21)/2 ; aria=math.sqrt(abs(sp*(sp-lcom)*(sp-l22)*(sp-l21)))
                h=2*aria/lcom ; lh2=math.sqrt(abs(l22**2-h**2)) ; lh1=math.sqrt(abs(l21**2-h**2))
                kpunct=lh1/lh2    ; self.ARIA=self.ARIA+aria
                if (lh1+lh2)-lcom>0.0001: kpunct=-kpunct
                xk=(xdsf1+kpunct*xdsf2)/(1+kpunct) ; yk=(ydsf1+kpunct*ydsf2)/(1+kpunct)
                coefi, alfaperp = self.ALFA(xdsf1, xdsf2, ydsf1, ydsf2)
                xalt=xk-h*math.cos(alfaperp) ; yalt=yk-coefi*h*math.sin(alfaperp)
                self.xbd.append( round(xk+h*math.cos(alfaperp),2) )
                self.ybd.append( round(yk+coefi*h*math.sin(alfaperp),2) )
                if i>2: Val_com = math.sqrt(abs((self.xad[i-2]-self.xbd[i])**2+(self.yad[i-2]-self.ybd[i])**2))
                if i>2 and Val_com < math.sqrt((self.xad[i-2]-xalt)**2+(self.yad[i-2]-yalt)**2):
                    ele1 = self.xbd.pop() ; ele2 = ybd() ;
                    self.xbd.append( round(xalt,2) ); self.ybd.append( round(yalt,2) )
                #--------------------------------
                xdsf1=self.xad[i-1] ; ydsf1=self.yad[i-1] ; xdsf2=self.xbd[i] ; ydsf2=self.ybd[i]
                lcom=l21
                l22=math.sqrt((self.Xs2[i]-self.Xs2[i-1])**2+(self.Ys2[i]-self.Ys2[i-1])**2+
                     (self.Zs2[i]-self.Zs2[i-1])**2)
                l21=math.sqrt((self.Xs2[i]-self.Xs1[i])**2+(self.Ys2[i]-self.Ys1[i])**2+(self.Zs2[i]-self.Zs1[i])**2)
                sp=(lcom+l22+l21)/2 ; aria=math.sqrt(abs(sp*(sp-lcom)*(sp-l22)*(sp-l21)))
                self.ARIA=self.ARIA+aria
                h=2*aria/lcom ; lh2=math.sqrt(l22**2-h**2) ; lh1=math.sqrt(l21**2-h**2) ; kpunct=lh1/lh2
                if (lh1+lh2)-lcom>0.0001: kpunct=-kpunct
                xk=(xdsf2+kpunct*xdsf1)/(1+kpunct) ; yk=(ydsf2+kpunct*ydsf1)/(1+kpunct)
                coefi, alfaperp = self.ALFA(xdsf1, xdsf2, ydsf1, ydsf2)
                xalt=xk-h*math.cos(alfaperp) ; yalt=yk-coefi*h*math.sin(alfaperp)
                self.xad.append( round(xk+h*math.cos(alfaperp),2) )
                self.yad.append( round(yk+coefi*h*math.sin(alfaperp),2) )
                if i>2: Val_com = math.sqrt((self.xbd[i-1]-self.xad[i])**2+(self.ybd[i-1]-self.yad[i])**2)
```

Listing 10.1 Surfaces.pyw – Flatten surface – Python & wxPython version

```
                if i>2 and Val_com < math.sqrt((self.xbd[i-1]-xalt)**2+(self.ybd[i-1]-yalt)**2):
                    ele1 = self.xad.pop() ; ele2 = yad() ;
                    self.xad.append( round(xalt,2) ) ; self.yad.append( round(yalt,2) )
        LengthA=0 ; LengthB=0
        for i, (xa, ya, xb, yb) in enumerate(zip(self.xad,self.yad, self.xbd, self.ybd)):
            if i>0:
                LengthA = LengthA+math.sqrt((self.xbd[i-1] - xb) ** 2 + (self.ybd[i-1] - yb) ** 2)
                LengthB = LengthB+math.sqrt((self.xad[i-1] - xa) ** 2 + (self.yad[i-1] - ya) ** 2)
        self.SB.SetStatusText("Length curve 2="+str(round(LengthA,2)),3)
        self.SB.SetStatusText("Length curve 1="+str(round(LengthB,2)),4)
        self.Draw_Flatten()

    def Draw_Flatten(self):
        # Draw section 1, 2 curve, points
        self.Axa2D.plot(self.xad, self.yad)  ;  self.Axa2D.plot(self.xbd, self.ybd)
        self.Axa2D.scatter(self.xbd, self.ybd, marker="8", s=20, color="Blue", label="Curve 2")
        self.Axa2D.scatter(self.xad, self.yad, marker="D", s=20, color="Red", label="Curve 1")
        # Straight bending lines
        for i, (xi1, yi1, xi2, yi2) in enumerate(zip(self.xad, self.yad,self.xbd, self.ybd)):
            self.Axa2D.plot([xi1,xi2],[yi1,yi2], color="Gray")
        # Diagonal bending lines
        for i, (xbla, ybla) in enumerate(zip(self.xad, self.yad)):
            if i< len(self.xad) - 1:
                xblb, yblb = self.xbd[i+1],self.ybd[i+1]
                self.Axa2D.plot([xbla,xblb],[ybla,yblb], color="Gray")
        self.Axa2D.legend(fontsize=12)  ; self.Axa2D.legend(loc='best')
        self.Axa2D.figure.canvas.draw()
        self.SB.SetStatusText("Flatten aria="+str(round(self.ARIA,2)),2)

    def OnMouseMotion(self, event):
        xMouse,yMouse = event.xdata, event.ydata
        if xMouse<>None and yMouse<> None:
            SirReal="X point="+'%0.4f' % xMouse + ",  Y point="+'%0.4f' % yMouse
            self.SB.SetStatusText( SirReal,0)

    def OnExcel(self, event):
        if len(self.xbd)==0:
            mesaj = "No point defined for surface.\n Open an Excel file with curves coordinates !"
            wx.MessageBox(mesaj, "Info",style=wx.OK|wx.ICON_EXCLAMATION)
            return
        xlApp = Dispatch("Excel.Application")
        xlWb = xlApp.Workbooks.Open(self.Excel_File) ; sheets = xlWb.Sheets
        XLSheet=xlWb.Worksheets(2) ; sheets(2).Activate()
        XLSheet.Cells(1,1).Value = "X1" ; XLSheet.Cells(1,2).Value = "Y1"
        XLSheet.Cells(1,3).Value = "X2" ; XLSheet.Cells(1,4).Value = "Y2"
        SirClip1="" ; lex=2
        for i, (xb, yb, xa, ya) in enumerate(zip(self.xbd, self.ybd, self.xad, self.yad)):
            XLSheet.Cells(lex,1).Value = xb ; XLSheet.Cells(lex,2).Value = yb
            XLSheet.Cells(lex,3).Value = xa ; XLSheet.Cells(lex,4).Value = ya
            lex+=1
        Fig1=os.getcwd()+'\\'+"3D_Surface.jpg" ; Fig2=os.getcwd()+'\\'+"Flatten surface.jpg"
        self.Axa_3D.figure.savefig(Fig1, orientation='landscape', bbox_inches=None,
            transparent=False, pad_inches=0, format ="jpg", quality = 95)
        self.Axa2D.figure.savefig(Fig2, orientation='landscape', bbox_inches=None,
            transparent=False, pad_inches=0, format ="jpg", quality = 95)
        ExportImageToExcel(Fig1, xlWb.Worksheets(2), 1, 6)
        ExportImageToExcel(Fig2, xlWb.Worksheets(2), 60, 1)
        xlWb.Close(SaveChanges=1) ; xlApp.Quit()
        xlWb = xlApp.Workbooks.Open(self.Excel_File) ; xlApp.Visible=True


    def OnClose(self, event):
        self.Destroy()

if __name__ == '__main__':   #==================================
    app = wx.App(0)
    Flatten_surface()
    app.MainLoop()
```

Listing 10.2 Surfaces.py – Flatten surface – Python & Streamlit version

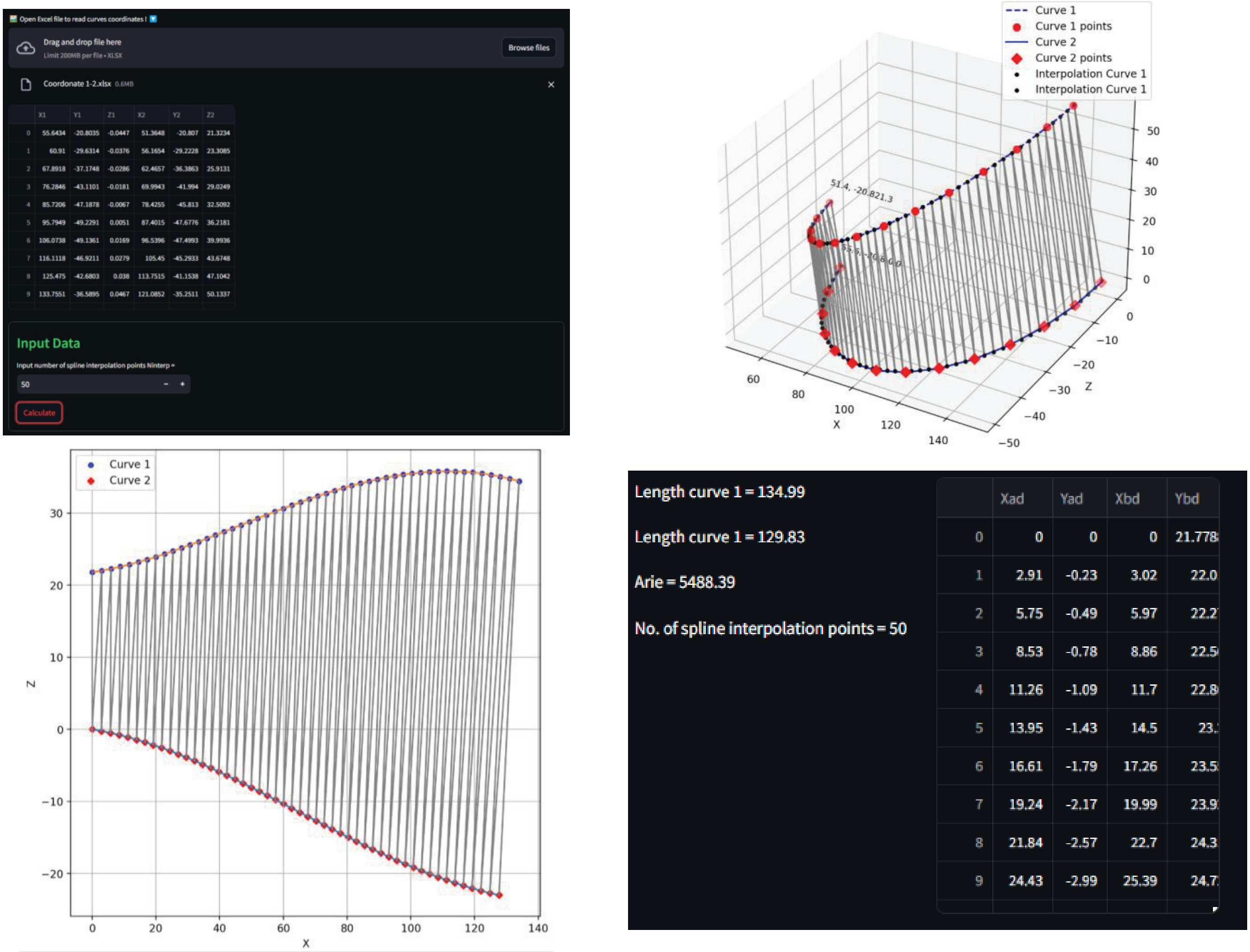
```
 import streamlit as st
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from pathlib import Path
from scipy import interpolate   # pip install scipy
import openpyxl
import math

st.set_page_config(page_title="Surface flatten")
css='''
<style>
    section.main > div {max-width:70rem}  # 1rem = 16px ; 70rem = 1120 px
</style>
'''
st.markdown(css, unsafe_allow_html=True)
current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()

def ALFA(xdsf1, xdsf2, ydsf1, ydsf2):
    if (xdsf2-xdsf1)==0:
        alfaperp=0
    else:
        mcom=(ydsf2-ydsf1)/(xdsf2-xdsf1) ; mperp=-1/mcom ; alfaperp=math.atan(mperp)
    if alfaperp<0:  alfaperp=math.pi-abs(alfaperp)
    coefi=1
    if alfaperp*180/math.pi>90: alfaperp=math.pi-alfaperp ; coefi=-1
    return [coefi, alfaperp]

st.subheader(":green[Surface flatten]")
MSJ=":frame_with_picture: Open Excel file to read curves coordinates ! :arrow_down_small:"
uploaded_file = st.file_uploader(MSJ, type="xlsx")
if uploaded_file:
    X1=[] ; Y1=[] ; Z1=[] ; X2=[] ; Y2=[] ; Z2=[]
    xad=[] ; yad=[] ; xbd=[] ; ybd=[]
    wb = openpyxl.load_workbook(uploaded_file)
    sheet  = wb.worksheets[0]
    # Count number of Excel rows filled with coordinates
    count_lex = 0
    for row in sheet:
        if not all([cell.value == None for cell in row]):
            count_lex += 1
    lex=3
    # Read curve coordinates from Excel file
    while True:
        if lex > count_lex:
            break
        else:
            X2append( sheet.cell(lex,2).value )
            Y2.append( sheet.cell(lex,3).value )
            Z2.append( sheet.cell(lex,4).value )
            X1.append( sheet.cell(lex,7).value )
            Y1.append( sheet.cell(lex,8).value )
            Z1.append( sheet.cell(lex,9).value )
            lex+=1
    df = pd.DataFrame( {'X1': X1, 'Y1': Y1,'Z1': Z1, 'X2': X2, 'Y2': Y2,'Z2': Z2} )
    st.dataframe(df)

    with st.form('Input Data'):  # Create input data form
        st.subheader(':green[Input Data]')
        col1, col2, col3 = st.columns(3)
        Ninterp = col1.number_input('Input number of spline interpolation points Ninterp = ', value=50)
        submit_button = st.form_submit_button('Calculate')
    if submit_button:
        placeholder = st.empty()
        fig = plt.figure(figsize=(8, 8))  ;  ax = plt.axes(projection="3d")
        ax.set_xlabel("X") ; ax.set_ylabel("Y") ; ax.set_ylabel("Z")
        # Draw section 1 curve, points & text
        ax.plot(X2, Y2, Z2, 'b--',  label='Curve 2)
        ax.scatter(X2, Y2, Z2, s=50, marker="8", color="Red", label='Curve 2points')
        sir= '%0.1f' % X1[0] +", "+ '%0.1f' % Y1[0] + '%0.1f' % Z1[0]
        ax.text(X1[0], Y1[0], Z1[0], sir, (1,0,0), size=8, zorder=1, color='k')
        # Draw section 1 curve, points & text
```

Listing 10.2 Surfaces.py – Flatten surface – Python & Streamlit version

```
        ax.plot(X1, Y1, Z1, 'b-',  label='Curve 1')
        ax.scatter(X1, Y1, Z1, s=50, marker="D", color="Red", label='Curve 1 points')
        sir= '%0.1f' % X2[0] +", "+ '%0.1f' % Y2[0] + '%0.1f' % Z2[0]
        ax.text(X2[0], Y2[0], Z2[0], sir, (1,0,0), size=8, zorder=1, color='k')
        # Spline interpolate curve 2
        pnts = np.linspace(0,1,Ninterp)
        tck2, u2 = interpolate.splprep([X2, Y2, Z2], s=2)
        Xs2, Ys2, Zs2 = interpolate.splev(pnts, tck2)
        ax.scatter(Xs2, Ys2, Zs2, 'o', c="Black", s=10,label="Interpolation Curve 2")
        # Spline interpolate curve 1
        tck1, u1 = interpolate.splprep([X1, Y1, Z1], s=2)
        Xs1, Ys1, Zs1 = interpolate.splev(pnts, tck1)
        ax.scatter(Xs1, Ys1, Zs1, 'o', c="Black", s=10,label="Interpolation Curve 1")
        # Straight bending lines
        for i, (x1, y1, z1, x2, y2, z2) in enumerate(zip(Xs1,Ys1,Zs1,Xs2,Ys2,Zs2)):
            ax.plot([x1,x2],[y1,y2],[z1,z2], color="Gray")
        # Diagonal bending lines
        for i, (x1a, y1a, z1a) in enumerate(zip(Xs1,Ys1,Zs1)):
            if i< len(Xs2) - 1:
                x2b, y2b, z2b = Xs2[i+1],Ys2[i+1],Zs2[i+1]
                ax.plot([x1a,x2b],[y1a,y2b],[z1a,z2b], color="Gray")
        ax.figure.canvas.draw() ; ax.legend(fontsize=10)  ; ax.legend(loc='best')
        plt.show()  ;  placeholder.pyplot(fig)
        # Compute flatten surface coordinates
        ARIA=0
        for i, (x1, y1, z1, x2, y2, z2) in enumerate(zip(Xs1,Ys1,Zs1,Xs2,Ys2,Zs2)):
            if i==0:
                lcom=math.sqrt((x1-x2)**2+(y1-y2)**2+(z1-z2)**2)
                xdsf1=0 ; ydsf1=0 ; xdsf2=0 ; ydsf2=lcom ; yaxa=lcom
                xad.append(0) ; yad.append(0) ; xbd.append(0) ; ybd.append(lcom)
            else: #--------------------------------
                xdsf1=xad[i-1] ; ydsf1=yad[i-1] ; xdsf2=xbd[i-1] ; ydsf2=ybd[i-1]
                lcom=math.sqrt((xdsf1-xdsf2)**2+(ydsf1-ydsf2)**2)
                l22=math.sqrt((x1-Xs1[i-1])**2+(y1-Ys1[i-1])**2+(z1-Zs1[i-1])**2)
                l21=math.sqrt((Xs2[i-1]-x1)**2+(Ys2[i-1]-y1)**2+(Zs2[i-1]-z1)**2)
                sp=(lcom+l22+l21)/2 ; aria=math.sqrt(abs(sp*(sp-lcom)*(sp-l22)*(sp-l21)))
                h=2*aria/lcom ; lh2=math.sqrt(abs(l22**2-h**2)) ; lh1=math.sqrt(abs(l21**2-h**2))
                kpunct=lh1/lh2  ;       ARIA=ARIA+aria
                if (lh1+lh2)-lcom>0.0001: kpunct=-kpunct
                xk=(xdsf1+kpunct*xdsf2)/(1+kpunct) ; yk=(ydsf1+kpunct*ydsf2)/(1+kpunct)
                coefi, alfaperp = ALFA(xdsf1, xdsf2, ydsf1, ydsf2)
                xalt=xk-h*math.cos(alfaperp) ; yalt=yk-coefi*h*math.sin(alfaperp)
                xbd.append( round(xk+h*math.cos(alfaperp),2) )
                ybd.append( round(yk+coefi*h*math.sin(alfaperp),2) )
                if i>2: Val_com = math.sqrt(abs((xad[i-2]-xbd[i])**2+(yad[i-2]-ybd[i])**2))
                if i>2 and Val_com < math.sqrt((xad[i-2]-xalt)**2+(yad[i-2]-yalt)**2):
                    ele1 = xbd.pop() ; ele2 = ybd() ;
                    xbd.append( round(xalt,2) ); ybd.append( round(yalt,2) )
                #-------------------------------
                xdsf1=xad[i-1] ; ydsf1=yad[i-1] ; xdsf2=xbd[i] ; ydsf2=ybd[i]
                lcom=l21
                l22=math.sqrt((Xs2[i]-Xs2[i-1])**2+(Ys2[i]-Ys2[i-1])**2+
                        (Zs2[i]-Zs2[i-1])**2)
                l21=math.sqrt((Xs2[i]-Xs1[i])**2+(Ys2[i]-Ys1[i])**2+(Zs2[i]-Zs1[i])**2)
                sp=(lcom+l22+l21)/2 ; aria=math.sqrt(abs(sp*(sp-lcom)*(sp-l22)*(sp-l21)))
                ARIA=ARIA+aria
                h=2*aria/lcom ; lh2=math.sqrt(l22**2-h**2) ; lh1=math.sqrt(l21**2-h**2) ; kpunct=lh1/lh2
                if (lh1+lh2)-lcom>0.0001: kpunct=-kpunct
                xk=(xdsf2+kpunct*xdsf1)/(1+kpunct) ; yk=(ydsf2+kpunct*ydsf1)/(1+kpunct)
                coefi, alfaperp = ALFA(xdsf1, xdsf2, ydsf1, ydsf2)
                xalt=xk-h*math.cos(alfaperp) ; yalt=yk-coefi*h*math.sin(alfaperp)
                xad.append( round(xk+h*math.cos(alfaperp),2) )
                yad.append( round(yk+coefi*h*math.sin(alfaperp),2) )
                if i>2: Val_com = math.sqrt((xbd[i-1]-xad[i])**2+(ybd[i-1]-yad[i])**2)
                if i>2 and Val_com < math.sqrt((xbd[i-1]-xalt)**2+(ybd[i-1]-yalt)**2):
                    ele1 = xad.pop() ; ele2 = yad() ;
                    xad.append( round(xalt,2) ) ; yad.append( round(yalt,2) )
        LengthA=0 ; LengthB=0
        for i, (xa, ya, xb, yb) in enumerate(zip(xad,yad, xbd, ybd)):
            if i>0:
                LengthA = LengthA+math.sqrt((xbd[i-1] - xb) ** 2 + (ybd[i-1] - yb) ** 2)
                LengthB = LengthB+math.sqrt((xad[i-1] - xa) ** 2 + (yad[i-1] - ya) ** 2)
```

Listing 10.2 Surfaces.py – Flatten surface – Python & Streamlit version

```python
# Draw FLATTEN SURFACE
placeholder1 = st.empty()
fig1 = plt.figure(figsize=(8, 8))
ax1 = plt.axes()
ax1.set_xlabel("X") ; ax1.set_ylabel("Y") ; ax1.set_ylabel("Z")
# Draw section 1, 2 flaten curve, points
ax1.plot(xad, yad)  ;  ax1.plot(xbd, ybd)
ax1.scatter(xbd, ybd, marker="8", s=20, color="Blue", label="Curve 2")
ax1.scatter(xad, yad, marker="D", s=20, color="Red", label="Curve 1")
# Straight bending lines
for i, (xi1, yi1, xi2, yi2) in enumerate(zip(xad, yad,xbd, ybd)):
    ax1.plot([xi1,xi2],[yi1,yi2], color="Gray")
# Diagonal bending lines
for i, (xbla, ybla) in enumerate(zip(xad, yad)):
    if i< len(xad) - 1:
        xblb, yblb = xbd[i+1],ybd[i+1]
        ax1.plot([xbla,xblb],[ybla,yblb], color="Gray")
ax1.legend(fontsize=10)  ; ax1.legend(loc='best') ; ax1.figure.canvas.draw()
ax1.grid(True) ;  placeholder1.pyplot(fig1)
col1, col2, col3, col4 = st.columns(4)
col1.write("Length curve 2= "+str(round(LengthA,2)))
col2.write("Length curve 1 = "+str(round(LengthB,2)))
col3.write("Arie = "+str(round(ARIA,2)))
col4.write("No. of spline interpolation points = "+str(Ninterp))
# Flatten surface coordonates
df1 = pd.DataFrame( {'Xad': xad, 'Yad': yad,'Xbd': xbd, 'Ybd': ybd} )
st.dataframe(df1)
```



Open Excel file to read curves coordinates !

	X1	Y1	Z1	X2	Y2	Z2
0	55.6434	-20.8035	-0.0447	51.3648	-20.807	21.3234
1	60.91	-29.6314	-0.0376	56.1654	-29.2228	23.3085
2	67.8918	-37.1748	-0.0286	62.4657	-36.3863	25.9131
3	76.2846	-43.1101	-0.0181	69.9943	-41.994	29.0249
4	85.7206	-47.1878	-0.0067	78.4255	-45.813	32.5092
5	95.7949	-49.2291	0.0051	87.4015	-47.6776	36.2181
6	106.0738	-49.1361	0.0169	96.5396	-47.4993	39.9936
7	116.1118	-46.9211	0.0279	105.45	-45.2933	43.6748
8	125.475	-42.6803	0.038	113.7515	-41.1538	47.1042
9	133.7551	-36.5895	0.0467	121.0852	-35.2511	50.1337

Input Data

Input number of spline interpolation points Ninterp = 50

Length curve 1 = 134.99

Length curve 1 = 129.83

Arie = 5488.39

No. of spline interpolation points = 50

	Xad	Yad	Xbd	Ybd
0	0	0	0	21.778
1	2.91	-0.23	3.02	22.0
2	5.75	-0.49	5.97	22.2
3	8.53	-0.78	8.86	22.5
4	11.26	-1.09	11.7	22.8
5	13.95	-1.43	14.5	23.
6	16.61	-1.79	17.26	23.5
7	19.24	-2.17	19.99	23.9
8	21.84	-2.57	22.7	24.3
9	24.43	-2.99	25.39	24.7

Figure 10.9 The interface of **Surfaces.py** script

11. Rotating a three-dimensional geometry

This script is designed to rotate a 3D geometry which consist in a house designed with points and lines, **Figure 11.1**. The house coordinates are presented in **Figure 11.2** and readed from **Coordonate.xlsx** file.

Listing 11.1 displays the Python & wxPython version of script, **Rotate.pyw** and **Listing 11.2** displays the Python & Streamlit version of the same script, **Rotate.py**. **Figure 11.3** display interface & the screens generated by Python & wxPython version of the script and **Figure 11.4** display display the screens generated by Python & Streamlit version.

The **Rotate.pyw** script (Python & wxPython version) include only one class: **Class_Rotire_3D**. The script begin with importing libraries: **wx** (the cross-platform GUI toolkit for the Python language), **os** (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.), **pandas** (open source data analysis and manipulation tool) and **matplotlib** (library for creating static, animated and interactive visualizations in Python).

The **Class_Rotire_3D** class create the interface, draw the 3D geometry of the house and include the following functions:

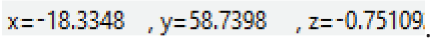
- **__init__** Define the **self.ax** area where the 3D geometry of the house will be drawn (line 12); define the grid color (line 13); lines 14÷16 define the axes titles; line 17 set the initial view of the 3D geometry by imposing the values of the elevation and azimuth; call the class functions **Read_Data** and **Draw**; finally show the plot (line 20).
- **Read_Data** Create **df** dataframe by reading geometry coordinates from the Excel file **Coordonate.xlsx** (line 23); convert columns X, Y, Z from dataframe to individual lists: **self.X, self.Y**, **self.Z** respectively (lines 24, 25, 26); line 27 convert columns **ID_Face** from dataframe to list of float numbers **ID_Pct**; line 28 convert list of float numbers **ID_Pct** to list of strings **self.ID_Point**.
- **Draw** Line 31 draw the points of the house using **scatter** command; line 32 draw the house lines using **wireframe** command; lines 14÷16 draw the identification points numbers in a **for** cycle.

Line 36 create the **app** which initializes the GUI toolkit for xPython; line 37 call **Class_Rotire_3D** class to create the window showed in **Figure 1.3**; finally, the line 38 enter into infinite continuous loop to receive key events from the user; the script can be interrupted with ☒ button of the frame.

A number of icons are automatically placed on the top toolbar, with following functions:

🏠	Reset original view.
←	Back to previous view.
→	Forward to next view.
✥	Pan axes with left mouse, zoom with right.
🔍	Zoom to rectangle.
⚙	Configure subplots.
💾	Save the figure into an image file (jpg, png, tiff, svg,…).
✓	Edit curves line and axes parameters.

To rotate the house geometry no icon must be selected and , with left mouse pressed, move the mouse over the 3D chart. During rotation the current values of the elevation and azimuth are displayed in the left location of the **StatusBar**: azimuth=139 deg, elevation=28 deg.

Moving the mouse over the chart without left mouse pressed will generate the current mouse position in the left location of the **StatusBar** showing the X, Y & Z coordinates:  x=-18.3348 , y=58.7398 , z=-0.75109.

Listing 11.1 Rotate.pyw – Rotating a three-dimensional geometry – Python & wxPython version

```
1.   from __future__ import division
2.
3.   import wx
4.   import os
5.   import pandas as pd
6.
7.   import mpl_toolkits.mplot3d.axes3d   # The tools are imported from the Matplotlib library
8.   import matplotlib.pyplot as plt
9.
10.  class Class_Rotire_3D():
11.          def __init__(self, parent, Title):
12.                  self.ax = plt.gca(projection='3d', title=Title)
13.                  self.ax.xaxis._axinfo['grid'].update({'color': 'Gray', 'linewidth': 2})
14.                  self.ax.set_xlabel('X', fontsize=18, fontweight='bold', color="Red")
15.                  self.ax.set_ylabel('Y', fontsize=18, fontweight='bold', color="Red")
16.                  self.ax.set_zlabel('Z', fontsize=18, fontweight='bold', color="Red")
17.                  self.ax.view_init(elev=20., azim=310)
18.                  self.Read_Data()
19.                  self.Draw()
20.                  plt.show()
21.
22.          def Read_Data(self):
23.                  df = pd.read_excel(os.getcwd()+"/Coordonate.xlsx")
24.                  self.X = df['X'].tolist() # Convert columns X from dataframe to list
25.                  self.Y = df['Y'].tolist() # Convert columns Y from dataframe to list
26.                  self.Z = df['Z'].tolist() # Convert columns Y from dataframe to list
27.                  ID_Pct=df['ID_Point'].tolist() # Convert columns ID_Face from dataframe to list of float numbers
28.                  self.ID_Point = [str(x) for x in ID_Pct] # Convert list of float numbers to list of strings
29.
30.          def Draw(self):
31.                  self.ax.scatter(self.X, self.Y, self.Z, edgecolors="Red", marker='o', s=60, facecolor="White")
32.                  self.ax.plot_wireframe(self.X, self.Y, self.Z, color="Blue", linewidth=2)
33.                  for i, ( lbl) in enumerate(zip(self.ID_Point)):
34.                          self.ax.text(self.X[i], self.Y[i], self.Z[i],lbl[0], color="Black", fontsize=20)
35.
36.  app = wx.App(redirect=False)
37.  frm_Rotire_3D=Class_Rotire_3D(None, "Rotate 3D home geometry")
38.  app.MainLoop()
```

To display the house 3D geometry on the browser window a new free and open source library will be used: **Plotly's Python** graphing library makes interactive, publication-quality graphs. This library can draw line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts and bubble charts.

The **Rotate.py** script (Python & Streamlit version) begin with importing libraries: **plotly.graph_objects** (module typically imported as **go** contains an automatically-generated hierarchy of Python classes which represent non-leaf nodes in the figure schema; the term **"graph objects"** refers to instances of these classes), **pandas** (open source data analysis and manipulation tool), **os** (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.).

This script contains 29 lines. Line 5create **df** dataframe by reading geometry coordinates from the Excel file **Coordonate.xlsx**; convert columns X, Y, Z from dataframe to individual lists: **self.X, self.Y**, **self.Z** respectively (lines 6, 7, 8); line 10 convert columns **ID_Face** from dataframe to list of float numbers **ID_Pct**; line 28 convert list of float numbers **ID_Pct** to list of strings **self.ID_Point**.

Lines 12÷20 create the 3D house geometry, **Figure 11.4**, with the following parameters: specifying the coordinates X, Y, Z (line 13), drawing modes (draw lines, markers and text – line 14), specifying text values associated to points as **ID_Point** list (line 15), text position (line 16) and size & opacity of the markers (lines 17÷19).

Lines 23÷26 define the default parameters which are used when **layout.scene.camera** is not provided. Line 28 update the properties of the figure's layout. Line 29 place the 3D geometry on browser window.

Listing 11.2 Rotate.py – Rotating a three-dimensional geometry – Python & Streamlit version

```
import plotly.graph_objects as go
import pandas as pd
import os

df = pd.read_excel(os.getcwd()+"/Coordonate.xlsx")
X = df['X'].tolist() # Convert columns X from dataframe to list
Y = df['Y'].tolist() # Convert columns Y from dataframe to list
Z = df['Z'].tolist() # Convert columns Y from dataframe to list
ID_Pct=df['ID_Point'].tolist() # Convert columns ID_Face from dataframe to list of float numbers
ID_Point = [str(x) for x in ID_Pct] # Convert list of float numbers to list of strings

fig = go.Figure(data=go.Scatter3d(
    x=X, y=Y,z=Z,
    mode='lines+markers+text',
    text=ID_Point,
    textposition="top center",
    marker=dict(
        size=8,
        opacity=0.5)
    ))

# Default parameters which are used when `layout.scene.camera` is not provided
camera = dict(
    up=dict(x=0, y=0, z=1),
    center=dict(x=0, y=0, z=0),
    eye=dict(x=1.25, y=1.25, z=1.25))

fig.update_layout(scene_camera=camera)
fig.show()
```

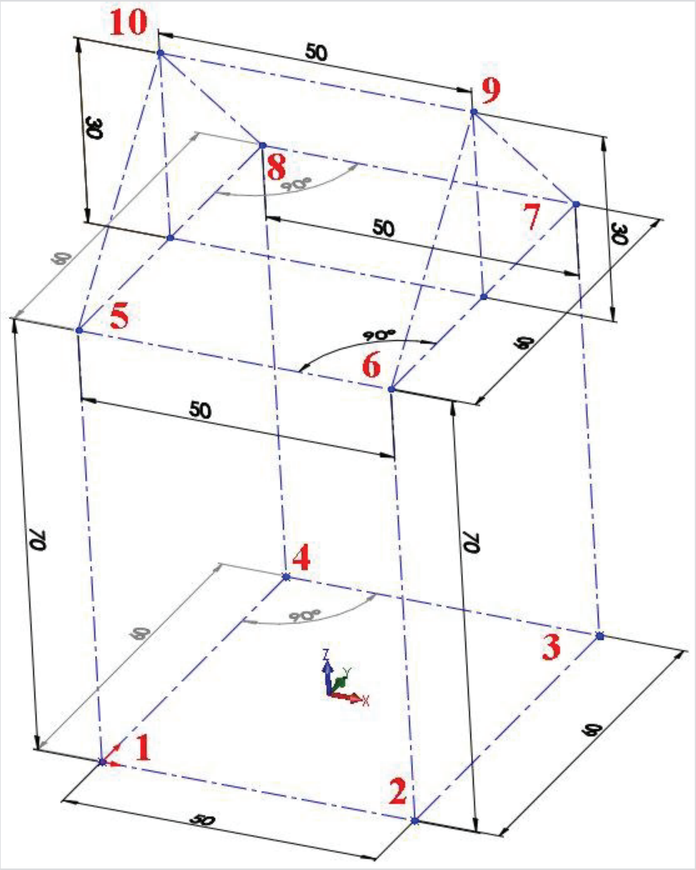
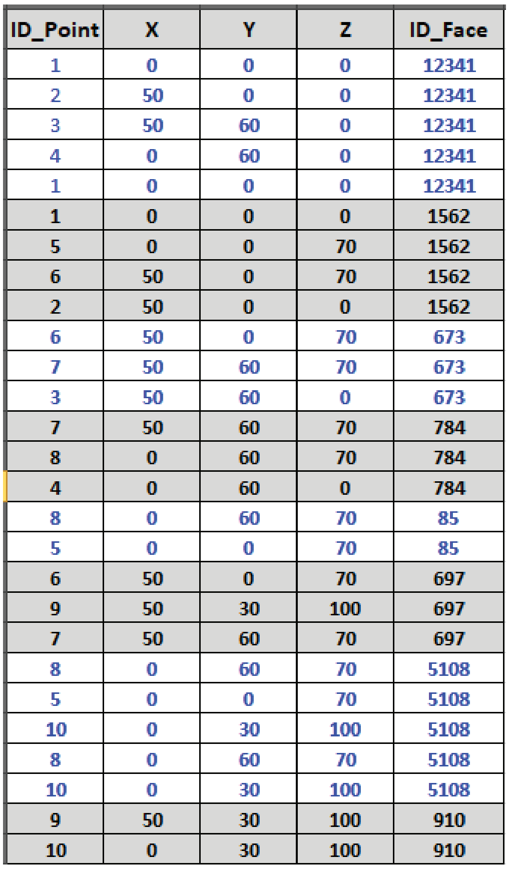


Figure 11.1 The house geometry



ID_Point	X	Y	Z	ID_Face
1	0	0	0	12341
2	50	0	0	12341
3	50	60	0	12341
4	0	60	0	12341
1	0	0	0	12341
1	0	0	0	1562
5	0	0	70	1562
6	50	0	70	1562
2	50	0	0	1562
6	50	0	70	673
7	50	60	70	673
3	50	60	0	673
7	50	60	70	784
8	0	60	70	784
4	0	60	0	784
8	0	60	70	85
5	0	0	70	85
6	50	0	70	697
9	50	30	100	697
7	50	60	70	697
8	0	60	70	5108
5	0	0	70	5108
10	0	30	100	5108
8	0	60	70	5108
10	0	30	100	5108
9	50	30	100	910
10	0	30	100	910

Figure 11.2 The house coordinates

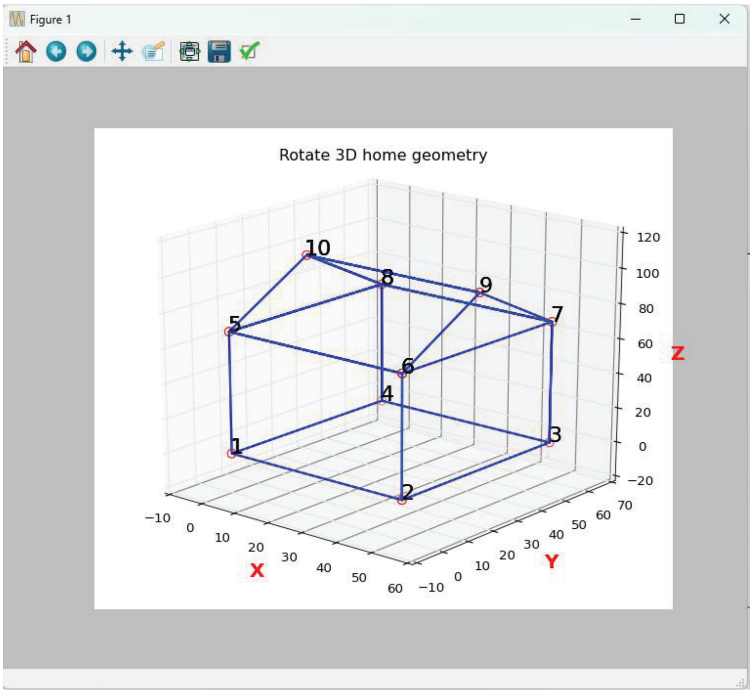


Figure 11.3 The interface and screen generated by Python & wxPython version of the **Rotate.pyw** script

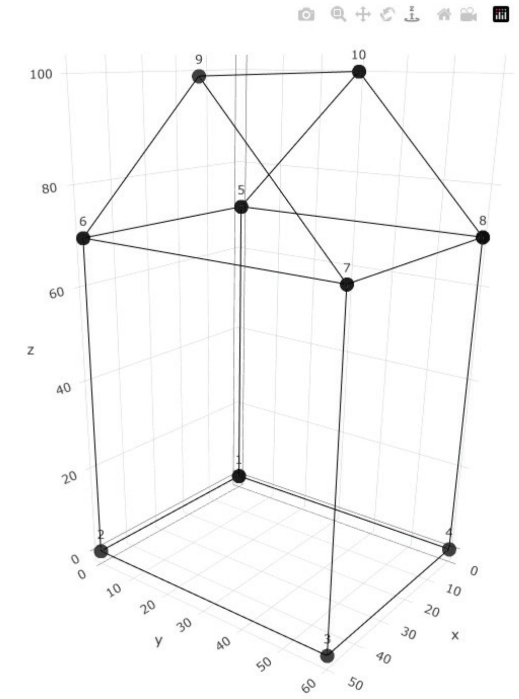


Figure 11.4 The interface and screen generated by Python & Streamlit version of the **Rotate.py** script

A number of icons are automatically placed on the top location of the browser window, figure **xx5.4**, with following functions:

📷	Download plot as png.	⚓	Turntable rotation.
🔍	Zoom.	🏠	Reset camera to default.
✥	Pan.	🎥	Reset camera to last view.
⟳	Orbital rotation.	▦	Plotly-logomart.

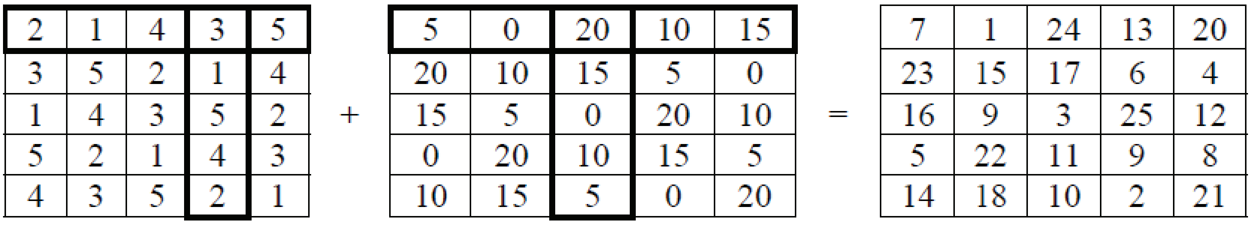
To rotate the house geometry move the mouse over the 3D chart with left mouse pressed.

12. Generate the magic square

A magic square is a matrix for which the sum of the elements on the rows, columns and the two diagonals (main and secondary) are equal. The algorithm for generating magic squares is:

- from the string 1,2,3,4,..., N, randomly choose N numbers (which can be multiples) and deposit them in first line of the first square; the rest of the lines (2nd to last) are filled starting with the number in the first column after the middle of the line and continuing with the first;
- from the string 0xN, 1xN, 2xN,...., (N-1)xN, N numbers (which may also be multiple) and are deposited in the first line of the second square; the remaining lines are filled in as in the first square, but starting from the middle of the line;
- by adding the two squares element by element, the result is a magic square, **Figure 12.1**.

Listing 12.1 displays the Python & wxPython version of script, **Magic_Square.pyw** and **Listing 12.2** displays the Python & Streamlit version of the same script, **Magic_Square.py**. **Figure 12.2** display interface & the screens generated by Python & wxPython version of the script and **Figure 12.3** display display the screens generated by Python & Streamlit version.



2	1	4	3	5
3	5	2	1	4
1	4	3	5	2
5	2	1	4	3
4	3	5	2	1

+

5	0	20	10	15
20	10	15	5	0
15	5	0	20	10
0	20	10	15	5
10	15	5	0	20

=

7	1	24	13	20
23	15	17	6	4
16	9	3	25	12
5	22	11	9	8
14	18	10	2	21

First square Second square The magic square

Figure 12.1 The algorithm for generating magic squares

The **Magic_Square.pyw** script (Python & wxPython version) include only one class: **Magic_Square**. The script begin with importing libraries: **wx** (the cross-platform GUI toolkit for the Python language), **os** (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.), **wx.grid** (classes are used for displaying and editing tabular data) and **random** (this module implements pseudo-random number generators for various distributions).

The **Class_Rotire_3D** class create the interface, draw the 3D geometry of the house and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame) the **StatusBar** (placed at the frame bottom, were messages can be displayed); define public class font **self.font (line 15)**; call the **_TOOLBAR** function; define public variable **self.N1**; call **CreateGrids** class function to generate the three grids: **First square**, **Second square** and **Magic square**; set the frame title and center the frame in the display.
- **_TOOLBAR** Create the **toolbar** (lines 24÷27), the **StaticText** control (lines 29÷30); create **wx.Choice** control (line 33) with the values defined in **lst_Choice** control (line 32) and place the first value **“5”** as first option to be selected (lines 34÷35); create the **Generate** button (line 37) and connect with **Bind** statement to **OnGenerate** function (line 38); lines 40÷42 add these controls to the **ToolBar** and line 44 generate and show the **ToolBar**.
- **CreateGrids** Class function to generate the three grids: **First square**, **Second square** and **Magic square**; define the size of grid columns and rows (line48); calculate **self.Width** and **self.Heigth** variables (line 49); create **StaticText** with label **First square**, blue color and font **self.font** (lines 52÷53) define first square as **self.grid1** and place in **self.panel** in **pos** position (line 55); create **self.grid1** grid and impose disable properties (cannot be modified) (line 57); also, impose horizontal and vertical text alignment in grid cells (line 58); line 60 impose the head rows heigth; line 61 impose the head columns width; inside the cycle from line 62, line 63 define the columns labels, line 64 define cells rows heigth and line 65 define cells columns

width; line 67 impose the **self.grid1** size; line 68 repaint the grid based on previous settings. Lines 71÷87 and 90÷106 create **Second square** and **Magic** square in the same manner. Line 108 impose the frame **Width** and **Heigth**.

- **OnGenerate** Regenerate the grids according to value selected from **Choice** control; first destroy **self.st1÷ self.st3** and **self.grid1÷ self.grid3** (lines 111-112); call **CreateGrids** function to regenerate grids (line 115); inside a **for** cycle from the string 1,2,3,4,..., N, randomly choose N numbers (which can be multiples) and deposit them in first line of the first square **self.grid1** (lines 118÷119); inside a **for** cycle from the string 0xN, 1xN, 2xN,...., (N-1)xN, randomly choose N numbers (which can be multiples) and deposit them in first line of the second square **self.grid2** (lines 120÷121); call **Calc_Grid** function for **Col_Start= self.N1 / 2** value (line 123); call **Calc_Grid** function for **Col_Start= self.N1 / 2 - 1** value (line 124); call **Sum_Grid** function (line 125).
- **Calc_Grid** For **self.grid1 t**his function complete with values the rest of the lines (2nd to last) starting with the number in the first column after the middle of the line and continuing with the first;
  for **self.grid2 t**his function complete with values the rest of the lines (2nd to last) starting with the number from the middle of the line and continuing with the first; the **opt** parameter set **GRID** variable to **self.grid1** for value **1** and **self.grid2** for **value 2**.
- **Sum_Grid** Calculate the sum of the elements on the rows, columns and the two diagonals (main and secondary) to verify that magic square conditions. Inside two **for** cycle lines 141÷149 add cells value from **self.grid1** & **self.grid2** (line 144) and put the sum in the same lin & col in **self.grid3** (line 146); also sum cells from every row (line 145) and put the results at the row end (line 147) with red colour (line 148) and yellow background (line 149). Lines 151÷157 compute the sum of every column and put the value in the last row cell. Lines 159÷165 compute the two diagonals (main and secondary) and put the value in the last diagonals cell.

Line 169 create the **app** which initializes the GUI toolkit for xPython; line 170 call **Magic_Square** class to create the window; finally, the line 171 enter into infinite continuous loop to receive key events from the user; the script can be interrupted with ☒ button of the frame.

Listing 12.1 Magic_Square.pyw – Magic_Square – Python & wxPython version

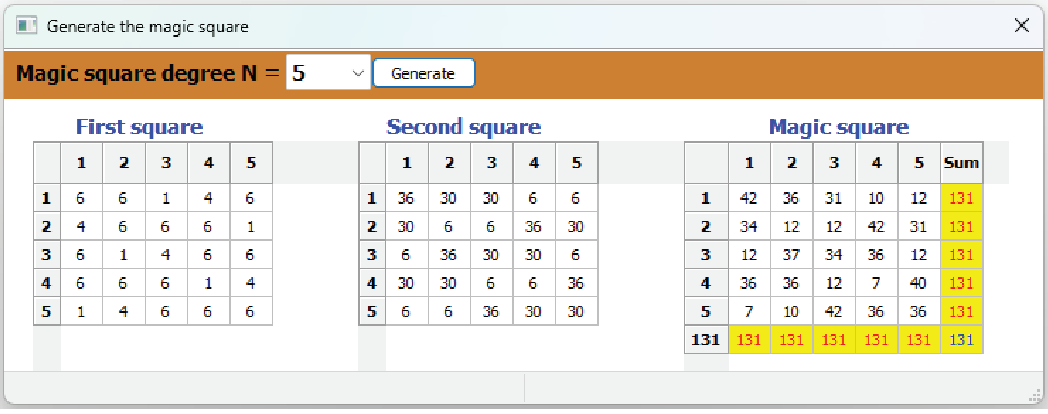
```
1.  import wx
2.  import wx.grid
3.  import random  # imports random module
4.
5.  class Magic_Square(wx.Frame):
6.
7.      def __init__(self,parent, id):
8.          wx.Frame.__init__(self, parent, id,style=
9.              wx.DEFAULT_FRAME_STYLE ^ (wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX ))
10.         self.panel = wx.Panel(self)  ;  self.panel.SetBackgroundColour("White")
11.         # StatusBar
12.         self.statusbar = self.CreateStatusBar()
13.         self.statusbar.SetFieldsCount(2)
14.         self.statusbar.SetStatusWidths([-10, -10])
15.         self.font = wx.Font(11, wx.DEFAULT, wx.NORMAL, wx.BOLD)
16.         self._TOOLBAR()
17.         self.N1=6
18.         self.CreateGrids()
19.         self.SetTitle("Generate the magic square")
20.         self.CenterOnScreen()
21.         self.Show()
22.
23.     def _TOOLBAR(self):
24.         self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
25.         self.toolbar.SetBackgroundColour("white")
26.         self.toolbar.SetToolBitmapSize((24,24))
27.         self.toolbar.SetBackgroundColour("GOLD")
28.
29.         st4 = wx.StaticText(self.toolbar, -1, "  Magic square degree N = ")
```

Listing 12.1 Magic_Square.pyw – Magic_Square – Python & wxPython version

```
        st4.SetFont(self.font)

        lst_Choice=["5", "7", "11", "13", "17"]
        self.txt_N = wx.Choice(self.toolbar, -1, choices=lst_Choice, size=(60, -1))
        self.txt_N.SetSelection( self.txt_N.FindString('5')  )
        self.txt_N.SetFont(self.font)

        self.btn = wx.Button(self.toolbar, -1, "Generate")
        self.btn.Bind(wx.EVT_BUTTON,self.OnGenerate)

        self.toolbar.AddControl(st4)
        self.toolbar.AddControl(self.txt_N)
        self.toolbar.AddControl(self.btn)

        self.toolbar.Realize()  ;  self.toolbar.Show()       # Toolbar show

    def CreateGrids(self):

        self.Col_size=30  ;  self.Row_size=20
        self.Width=self.Row_size+(self.N1+1)*self.Col_size ; self.Heigth=(self.N1+1)*self.Row_size

        # Create 'First square' grid
        self.st1=wx.StaticText(self.panel, -1, "First square", pos=(50,10))
        self.st1.SetForegroundColour('BLUE')  ;  self.st1.SetFont(self.font)

        self.grid1 = wx.grid.Grid(self.panel, pos=(20,30))
        self.grid1.CreateGrid(self.N1-1,self.N1-1)
        self.grid1.EnableEditing(0)
        self.grid1.SetDefaultCellAlignment(wx.ALIGN_CENTRE,wx.ALIGN_CENTRE)

        self.grid1.SetRowLabelSize(self.Row_size)
        self.grid1.SetColLabelSize(self.Col_size)
        for i in range(self.N1-1):
            self.grid1.SetColLabelValue(i,str(i+1))      # Headlines in self.grid1
            self.grid1.SetRowSize(i,self.Row_size)   # Define table row heigth
            self.grid1.SetColSize(i,self.Col_size)   # Define table column widths
        # Table head height and table numbering column width
        self.grid1.SetSize((self.Width, self.Heigth+80))
        self.grid1.ForceRefresh()

        # Create 'Second square' grid
        self.st2=wx.StaticText(self.panel, -1, "Second square", pos=(self.Width+40,10))
        self.st2.SetForegroundColour('BLUE')  ;  self.st2.SetFont(self.font)

        self.grid2 = wx.grid.Grid(self.panel, pos=(self.Width+20,30))
        self.grid2.CreateGrid(self.N1-1,self.N1-1)
        self.grid2.EnableEditing(0)
        self.grid2.SetDefaultCellAlignment(wx.ALIGN_CENTRE,wx.ALIGN_CENTRE)

        self.grid2.SetRowLabelSize(self.Row_size)
        self.grid2.SetColLabelSize(self.Col_size)
        for i in range(self.N1-1):
            self.grid2.SetColLabelValue(i,str(i+1))      # Headlines in self.grid2
            self.grid2.SetRowSize(i,self.Row_size)   # Define table row heigth
            self.grid2.SetColSize(i,self.Col_size)   # Define table column widths
        # Table head height and table numbering column width
        self.grid2.SetSize((self.Width, self.Heigth+80))
        self.grid2.ForceRefresh()

        # Create 'Magic square' grid
        self.st3=wx.StaticText(self.panel, -1, "Magic square", pos=(2*self.Width+80,10))
        self.st3.SetForegroundColour('BLUE')  ;  self.st3.SetFont(self.font)

        self.grid3 = wx.grid.Grid(self.panel, pos=(2 * self.Width+20,30))
        self.grid3.CreateGrid(self.N1,self.N1)
        self.grid3.EnableEditing(0)
        self.grid3.SetDefaultCellAlignment(wx.ALIGN_CENTRE,wx.ALIGN_CENTRE)

        self.grid3.SetRowLabelSize(self.Row_size*1.6)
        self.grid3.SetColLabelSize(self.Col_size)
        for i in range(self.N1):
            self.grid3.SetColLabelValue(i,str(i+1))      # Headlines in self.grid3
```

Listing 12.1 Magic_Square.pyw – Magic_Square – Python & wxPython version

```
                self.grid3.SetRowSize(i,self.Row_size)   # Define table row heigth
                self.grid3.SetColSize(i,self.Col_size)   # Define table column widths
            # Table head height and table numbering column width
            self.grid3.SetSize((self.Width, self.Heigth+80))
            self.grid3.ForceRefresh()

            self.SetSize((3 * (self.Width+20) ,self.Heigth+180))

    def OnGenerate(self, event):
            self.st1.Destroy()  ;  self.st2.Destroy()  ;  self.st3.Destroy()
            self.grid1.Destroy()  ;  self.grid2.Destroy()  ;  self.grid3.Destroy()

            self.N1 =int(self.txt_N.GetStringSelection().strip())+1
            self.CreateGrids()

            for col in range(self.N1-1):
                r1 = random.randint(1, self.N1)     # Generate values in line 0 of self.grid1
                self.grid1.SetCellValue(0,col,  str(r1))
                r2 = self.N1*random.randint(1, self.N1) ;    # Generate values in line 0 of self.grid2
                self.grid2.SetCellValue(0,col, str(r2))

            self.Calc_Grid(1, self.N1 / 2)
            self.Calc_Grid(2, self.N1 / 2-1)
            self.Sum_Grid()

    def Calc_Grid(self, opt, Col_Start):
            GRID=self.grid1
            if opt==2: GRID=self.grid2
            for lin in range(1, self.N1-1):
                counter=Col_Start
                for col in range(0, self.N1-1):
                    V = GRID.GetCellValue(lin-1, counter)
                    GRID.SetCellValue(lin,col,  V)
                    counter=counter+1
                    if counter >= self.N1-1:  counter  = 0

    def Sum_Grid(self):
            self.grid3.SetColLabelValue(self.N1-1,'Sum')

            for lin in range(0, self.N1-1):
                sum_lin=0
                for col in range(0, self.N1-1):
                    V = int(self.grid1.GetCellValue(lin, col)) + int(self.grid2.GetCellValue(lin, col))
                    sum_lin=sum_lin+V
                    self.grid3.SetCellValue(lin,col,  str(V))
                self.grid3.SetCellValue(lin,self.N1-1,  str(sum_lin))
                self.grid3.SetCellTextColour(lin,self.N1-1, 'Red')
                self.grid3.SetCellBackgroundColour(lin,self.N1-1, 'Yellow' )

            for col in range(0, self.N1-1):
                sum_col=0
                for lin in range(0, self.N1-1):
                    sum_col=sum_col+int(self.grid3.GetCellValue(lin, col))
                self.grid3.SetCellValue(self.N1-1,col,  str(sum_col))
                self.grid3.SetCellTextColour(self.N1-1,col, 'Red')
                self.grid3.SetCellBackgroundColour(self.N1-1,col, 'Yellow' )

            sum_DP=0  ;  sum_DS=0
            for lin in range(0, self.N1-1):
                sum_DP=sum_DP+int(self.grid3.GetCellValue(lin, lin))
                sum_DS=sum_DS+int(self.grid3.GetCellValue(lin, self.N1-2-lin))
            self.grid3.SetCellValue(self.N1-1,self.N1-1,  str(sum_DP))
            self.grid3.SetCellTextColour(self.N1-1,self.N1-1, 'Blue')
            self.grid3.SetCellBackgroundColour(self.N1-1,self.N1-1, 'Yellow' )

            self.grid3.SetRowLabelValue(self.N1-1,str(sum_DS))

app = wx.App(redirect=0)
frm=Magic_Square(None, -1)
app.MainLoop()
```



Generate the magic square

Magic square degree N = 5

First square

	1	2	3	4	5
1	6	6	1	4	6
2	4	6	6	6	1
3	6	1	4	6	6
4	6	6	6	1	4
5	1	4	6	6	6

Second square

	1	2	3	4	5
1	36	30	30	6	6
2	30	6	6	36	30
3	6	36	30	30	6
4	30	30	6	6	36
5	6	6	36	30	30

Magic square

	1	2	3	4	5	Sum
1	42	36	31	10	12	131
2	34	12	12	42	31	131
3	12	37	34	36	12	131
4	36	36	12	7	40	131
5	7	10	42	36	36	131
131	131	131	131	131	131	131

Figure 12.2 The interface and screen generated by Python & wxPython version of the **Magic_Square.pyw** script

The **Magic_Square.py** script (Python & Streamlit version) include the public **Calc_DataFrame** function. The script begin with importing libraries: **streamlit** (create a great interactive web application with Python), **pandas** (open source data analysis and manipulation tool), **matplotlib** (library for creating static, animated and interactive visualizations in Python), **numpy** (the fundamental package for scientific computing with Python) and **random** (this module implements pseudo-random number generators for various distributions). This script contains 63 lines.

Lines 1 to 4 import the required modules. Lines 6 to 13 define the **Calc_DataFrame** function; for dataframe **df1** this function complete with values the rest of the lines (2nd to last) starting with the number in the first column after the middle of the line and continuing with the first; for dataframe **df2** this function complete with values the rest of the lines (2nd to last) starting with the number from the middle of the line and continuing with the first; the **DFrame** parameter receive the value **df1** or **df2** when **Calc_DataFrame** function is called in lines 35 or 36.

The **Calc_DataFrame** procedure generates the elements of the dataframe from line 2 to the last line, according to the logic imposed by the algorithm; the procedure receives as input the dataframe **df1** or **df2** transfered into **DFrame** variable, the size of the dataframe **N1** and the reference column for the translation of the values **Col_Start**. The logic for generating the dataframe values is as follows: the current element of the dataframe takes the values from the previous row and the column specified by **counter** variable; the fetching starts from the reference column, whose value is stored initially stored in the **counter** variable, the value of which increases by 1 at each retrieval; if, by is incremented by 1, the value of the counter variable exceeds the number of dataframe columns (the dataframe boundary has been exceeded), then the **counter** variable is initialized by 0.

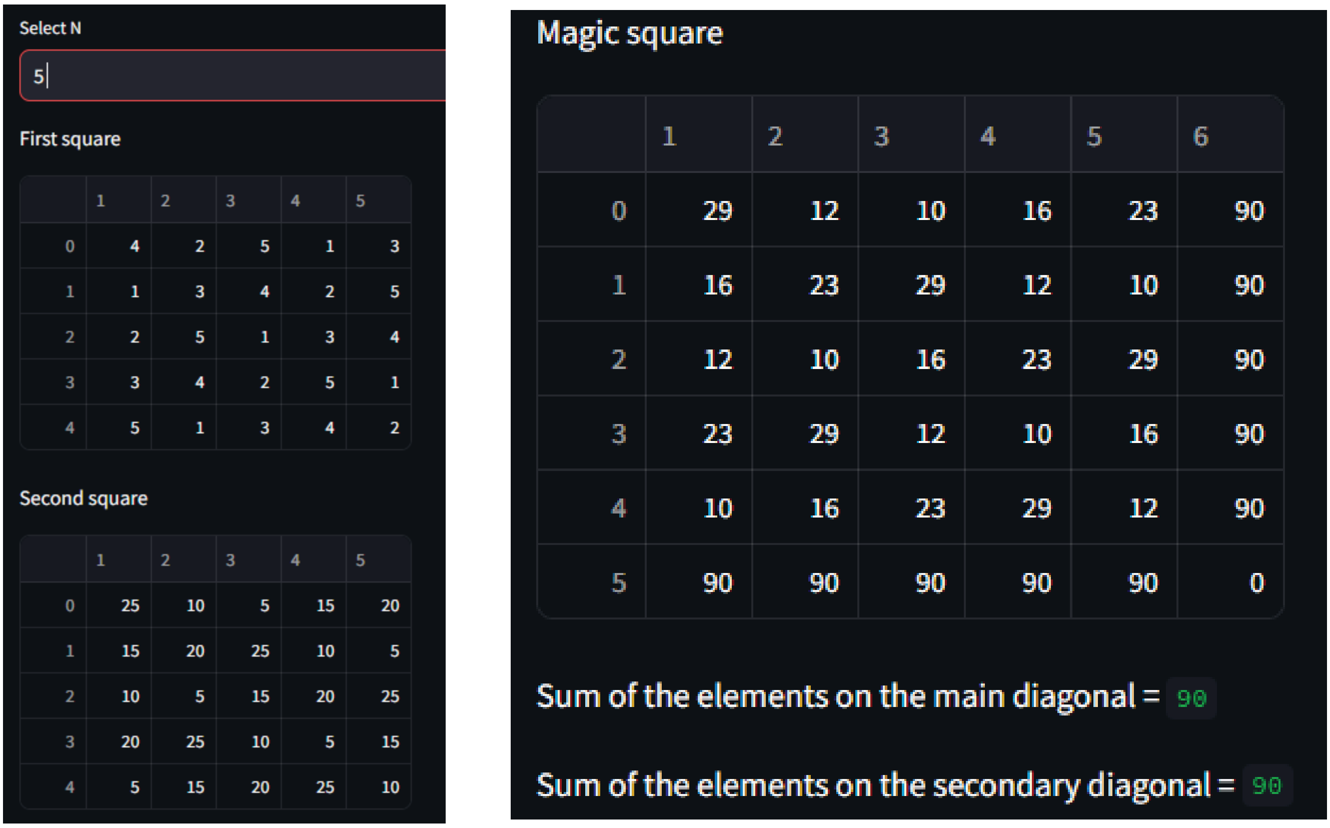
Line 15 defines the script title with **subheader** command. Line 17 define **lst_Choice** list variable; line 18 create a **selectbox** control based on **lst_Choice** list; line 19 verifies if a selectin was made from the **selectbox** control; lines 21÷ 23 generate **col_list** list with store the numerical ID's of datframe columns; lines 24÷25 create datframes **df1** and **df2**; line 26 extend the col_list for **df3** dataframe with one position, since the **Magic Square** dataframe **df3** will show also the sum of rows and columns.

Inside a **for** cycle from the string 1,2,3,4,..., N, randomly choose N numbers (which can be multiples) and deposit them in first line of the first square dataframe **df1** (lines 29÷30); inside a **for** cycle from the string 0xN, 1xN, 2xN,...., (N-1)xN, randomly choose N numbers (which can be multiples) and deposit them in first line of the second square dataframe **df2** (lines 31÷32); call **Calc_DataFrame** function for **Col_Start= int((N1-1) / 2) + 1** value (line 35); call **Calc_DataFrame** function for **Col_Start= int((N1-1) / 2)** value (line 36); line 38 display the "**First square**"title; lines 39-41 display dataframes **df1** and **df2**; line 40 display the "**Second square**"title; ; line 43 display the "**Magic square**"title.

Inside two **for** cycle lines 141÷149 add cells value from **df1** & **df2** and put the sum in the same lin & col in **df3** dataframe (line 47); also sum cells from every row (line 48) and put the results at the row end (line 49). Lines 51÷55 compute the sum of every column and put the value in the last row cell. Line 57 initialize with 0 the **sum_DP** and **sum_DS** variables to store the sum of the elements on the main and secondary diagonals. Inside a **for** cycle lines lines 58÷60 compute the sum of two diagonals (main and secondary) and show the values in lines 62÷63.

Listing 12.2 Magic_Square.py – Magic_Square– Python & Streamlit version

```
import streamlit as st
import pandas as pd
import numpy as np
import random  # imports random module

def Calc_DataFrame(DFrame, N1, Col_Start):
    for lin in range(1, N1):
        counter = int((N1-1) / 2) +1
        for col in range(0, N1):
            V = DFrame.iat[lin-1,counter]
            DFrame.at[lin,col+1] = V
            counter=counter+1
            if counter >= N1:  counter  = 0

st.subheader(":green[Magic square degree N = ]")

lst_Choice=["Select N", "5", "7", "11", "13", "17"]
N=st.selectbox("Select N",options=lst_Choice)
if N != "Select N":
    N1=int(N)
    col_list = []
    for i in range(1,N1+1):
        col_list.append(i)
    df1 = pd.DataFrame(0, index=np.arange(N1), columns=col_list)
    df2 = pd.DataFrame(0, index=np.arange(N1), columns=col_list)
    col_list.append(N1+1)
    df3 = pd.DataFrame(0, index=np.arange(N1+1), columns=col_list)
    for col in range(1,N1+1):
        r1 = random.randint(1, N1)
        df1.at[0,col] = r1
        r2 = N1 * random.randint(1, N1)
        df2.at[0,col] = r2

    Col_Start = int((N1-1) / 2) +1
    Calc_DataFrame(df1, N1, Col_Start)
    Calc_DataFrame(df2, N1, Col_Start-1)

    st.write("First square")
    df1
    st.write("Second square")
    df2

    st.write("Magic square")
    for lin in range(0, N1):
        sum=0
        for col in range(0, N1):
            df3.iat[lin,col] = df1.iat[lin,col]+df2.iat[lin,col]
            sum=sum+df3.iat[lin,col]
        df3.at[lin,N1+1]=sum

    for col in range(0, N1):
        sum=0
        for lin in range(0, N1):
            sum=sum+df3.iat[lin,col]
        df3.at[N1,col+1]=sum

    sum_DP=0  ;  sum_DS=0
    for lin in range(0, N1):
            sum_DP=sum_DP+df3.iat[lin,lin]
            sum_DS=sum_DS+df3.iat[lin,N1-lin-1]
    df3
    st.write("Sum of the elements on the main diagonal = ", sum_DP)
    st.write("Sum of the elements on the secondary diagonal = ", sum_DS)
```



Select N

5

First square

	1	2	3	4	5
0	4	2	5	1	3
1	1	3	4	2	5
2	2	5	1	3	4
3	3	4	2	5	1
4	5	1	3	4	2

Second square

	1	2	3	4	5
0	25	10	5	15	20
1	15	20	25	10	5
2	10	5	15	20	25
3	20	25	10	5	15
4	5	15	20	25	10

Magic square

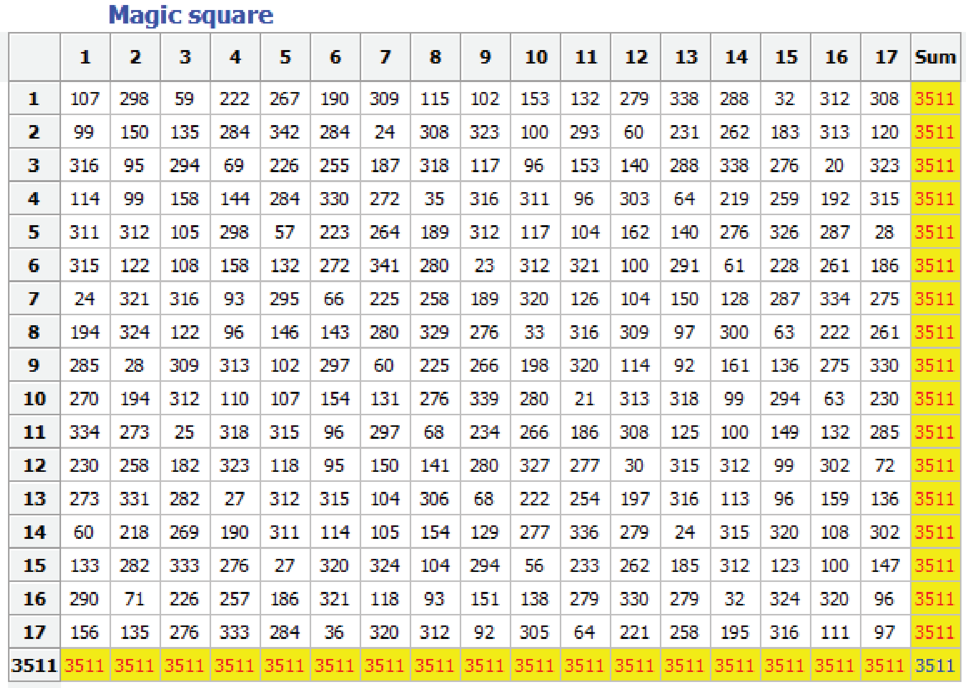
	1	2	3	4	5	6
0	29	12	10	16	23	90
1	16	23	29	12	10	90
2	12	10	16	23	29	90
3	23	29	12	10	16	90
4	10	16	23	29	12	90
5	90	90	90	90	90	0

Sum of the elements on the main diagonal = 90

Sum of the elements on the secondary diagonal = 90

Figure 12.3 The interface and screen generated by Python & Streamlit version of the **Magic_Square.py** script

Figure 12.4 display the **Magic Square** generated for 17 lines & columns by Python & wxPython version of the **Magic_Square.pyw** script.



Magic square

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Sum
1	107	298	59	222	267	190	309	115	102	153	132	279	338	288	32	312	308	3511
2	99	150	135	284	342	284	24	308	323	100	293	60	231	262	183	313	120	3511
3	316	95	294	69	226	255	187	318	117	96	153	140	288	338	276	20	323	3511
4	114	99	158	144	284	330	272	35	316	311	96	303	64	219	259	192	315	3511
5	311	312	105	298	57	223	264	189	312	117	104	162	140	276	326	287	28	3511
6	315	122	108	158	132	272	341	280	23	312	321	100	291	61	228	261	186	3511
7	24	321	316	93	295	66	225	258	189	320	126	104	150	128	287	334	275	3511
8	194	324	122	96	146	143	280	329	276	33	316	309	97	300	63	222	261	3511
9	285	28	309	313	102	297	60	225	266	198	320	114	92	161	136	275	330	3511
10	270	194	312	110	107	154	131	276	339	280	21	313	318	99	294	63	230	3511
11	334	273	25	318	315	96	297	68	234	266	186	308	125	100	149	132	285	3511
12	230	258	182	323	118	95	150	141	280	327	277	30	315	312	99	302	72	3511
13	273	331	282	27	312	315	104	306	68	222	254	197	316	113	96	159	136	3511
14	60	218	269	190	311	114	105	154	129	277	336	279	24	315	320	108	302	3511
15	133	282	333	276	27	320	324	104	294	56	233	262	185	312	123	100	147	3511
16	290	71	226	257	186	321	118	93	151	138	279	330	279	32	324	320	96	3511
17	156	135	276	333	284	36	320	312	92	305	64	221	258	195	316	111	97	3511
3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511	3511

Figure 12.4 The **Magic Square** generated for 17 lines & columns by Python & wxPython version of the **Magic_Square.pyw** script

13. Spline Interpolation

Interpolation is a method for generating points between given points. Spline interpolation based on cubic splines creates an interpolating curve from cubic functions with matching first and second derivatives. A cubic spline is a piecewise cubic function interpolating a set of data points and ensuring smoothness at the data points. The purpose of this application is to compare, for a set of points, the interpolation results generated by different spline functions.

The script use a database Config.db to store all icons loaded in frame Toolbar; this will avoid the existence of following icons files in the application folder: , , , , , , and . The Config.db → objects store icons for the script interface of **Python 2. 7 & wxPython** application version. Initially, the icons files are stored in the Spline Interpolation→ Objects application folder. The icons files were loaded into Config.db → objects as Binary Large Object (BLOB) through the _copyimg.pyw script displayed in Listing 8.1, from where they will be used into application interface through **Python 2.7** → Interpolation.pyw script. For **Python & Streamlit** → Interpolation.py script version icon files have not been used.

The Interpolation.pyw script (Python & wxPython version) include three classes: Plot, PlotNotebook & Interpolation and two public functions: ExtragImageMemory & Put_Clipboard. The script starts by importing the libraries (lines 3÷20). Lines 19÷20 import external modules: KIUSALAAS and KOMASARU, stored in the application folder, to calculate a 2-D cubic spline function.

The ExtragImageMemory function (lines 22÷26) selecteaza din Interpolation → Objects folder an icon and create & returneaza the icon img from data in memory. This function is called from class Interpolation → _TOOLBAR function. The Put_Clipboard function (lines 28÷32) receive a string Sir_Clip as parameter, open and empty the clipboard, place the string into clipboard and close the clipboard.

The Plot class (lines 34÷46) create self.figure, self.canvas, self.toolbar and sizer entities for the class that calls it. The PlotNotebook class (lines 48÷63) creates several pages for the class that calls it, using the internal function add; through internal function ChangePagina the user can change the page. This class represents a notebook control, which manages multiple windows with associated tabs.

The Interpolation class create the interface, the aplication main functionalities and include the following functions:

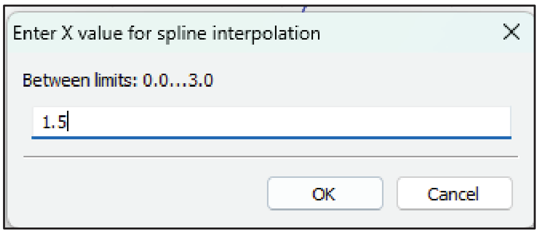
- __init__ Define the frame (a window whose size and position can be changed by the user); create the statusbar (lines 72÷74); line 76 create in self.plotter an instance of PlotNotebook class; line 77 create self.AXA page, were two graphics subplot self.axes1 & self.axes2 will be created side by side (lines 79, 81); both will be connected with Bind statement to OnMouseMotion class internal function (lines 80, 82); line 84 call Recreate_Axis class internal function; line 84 initializes the public variables of the class X and Y, which store the coordinates of the points that are initial data for the spline functions; line 86 call _TOOLBAR function; lines 87÷88 center and show the frame in the display.
- _TOOLBAR Create the toolbar (line 101); set the toolbar icons size (line 103) and background color (line 104); create the connection and cursor using Config.db database (line 106); generate Lst_label, Lst_icons and Lst_Help lists (lines 107÷110), which store the tooolbar icons labels, the icon names that must be extracted from memory by calling ExtragImageMemory function and the associated icons help; for every icon line 113 associate an unique identificator ID; through for cycle from line 112 the img icon is extracted into memory (ine 114); create the bitmap of img memory icon and scale at the imposed toolbar icons size (line 115); in this cycle, for every operation, the icon is added in toolbar (line 116) ; lines 117÷124 connect with Bind statement to associated functions; line 126 generate and show the toolbar; line 127 close the connection with Config.db database.
- CHARTS Call Recreate_Axis, DrawInitialPoints, Kiusalaas, KOMASARU, splprep, interp1d, interp and LEGEND functions to calculate and draw the spline interpolation curve.
- OnOpenExcelFile Open a dialog to load an Excel file (line 23) and store the filename in variable self.Excel_File line (26); open Excel file (line 29) and read the sample points coordinates

from cells in a **while** loop (line 32) until a cell content is empty (line 34); store the coordinates in lists **self.X** and **self.Y** variables (line 37); close the Excel file (line 39) and call **CHARTS** function.

- **DrawInitialPoints** Draw in both **axes1** and **axes2** the initial sample points readed from an Excel file by **OnOpenExcelFile** function.
- **interp** This function calculate one-dimensional linear interpolation for monotonically increasing sample points [**13.1**].The function create X echidistant 50 points and draw the interpolation curve in **axes2**.
- **interp1d** This function interpolate a 1-D function for monotonically increasing sample points [**13.1**]. The function create X echidistant 50 points and draw the interpolation curve in **axes2**.
- **Kiusalaas** This function interpolate a 2-D cubic spline function for monotonically increasing sample points [**13.2**] - pag. 114. The calculated spline interpolation curve is drawn in **axes1**.
- **KOMASARU** This function interpolate a 2-D cubic spline function for monotonically increasing sample points [**13.3**]. The calculated spline interpolation curve is drawn in **axes1**.
- **splprep** Calculate the B-spline representation of an N-D curve [**13.4**]. The calculated spline interpolation curve is drawn in **axes2**.
- **LEGEND** For both axes, this functions set the charts legend fontsize, position and redraw axes. The function is called by **OnInterpolate** and **CHARTS** functions.
- **OnInterpolate** This function calculate the value of spline interpolation for only one point imposed by the user through a **wx.TextEntryDialog**; initially, the function tests if the coordinates of the points have been read (lines 138÷143); in the dialog is proposed the median value **Vmed** between **min(self.X)** and **max(self.X),** but the user can change this value, **Figure 13.1**, (line 144÷145); the user X value is stored in **Xinterp** variable (line 148) and, for all type of spline interpolation, the Y values is calculated through lines 150÷168; the calculated point (Xinterp, Y) is marked for every spline function (except linear **interp** function) to both axes (lines 153, 159, 164, 168); finally, all Y values are deposited to **Clipboard** (line 170) and **LEGEND** function is called.
- **OnMouseMotion** Display in the **StatusBar** the current values of X & Y mouse position in the chart.
- **OnReload** Call **CHARTS** function.
- **OnSave** Save both axes charts in a single file named **Interpolation.jpg** in the application folder.
- **Recreate_Axis** This function clear both axes, set the grid and charts labels parameters. The function is called by **__init__** and **CHARTS** functions.
- **Home** Redraw the charts to fit in the axes spaces.
- **Pan** Pan the charts.
- **Zoom** Create a rectangular zoom in the selected chart.
- **OnClose** This function exits the application.

Figure 13.2 The show the table of initial sample points readed from an Excel file and **Figure 13.3** show the Excel chart of these points. **Figure 13.4** show the application interface, the spline curves of the initial sample points and marked with X=1.5 point (red color) calculated by spline functions. As can be seen from the charts, the interpolated values for X=1.5 are: **KOMASARU** point=1.500, 1.3250, **Kiusalaas** point=1.500, 1.3250, **splprep** point=1.500, 1.3125 and **interp1d** point=1.500, 1.3125. The **KOMASARU** and **Kiusalaas** spline curves are identical, while **splprep** and **interp1d** has slightly differences at the beginning and the end. The **interp1d** linear curve pass through the points with lines while the rest are drawn with cubic function. All curves pass through the initial sample points. For other sample points the differences between spline curves can be smaller.

Listing 13.1 displays the **Python & wxPython** version of script, **Interpolation.pyw** and **Listing 13.2** displays the Python & Streamlit version of the same script, **Interpolation.py**.



Enter X value for spline interpolation

Between limits: 0.0...3.0

1.5

Figure 13.1 The dialog to input the X value of a point for spline interpolation

X	Y
0	0
1	0.5
2	2
3	1.5

Figure 13.2 The initial sample points readed from an Excel file

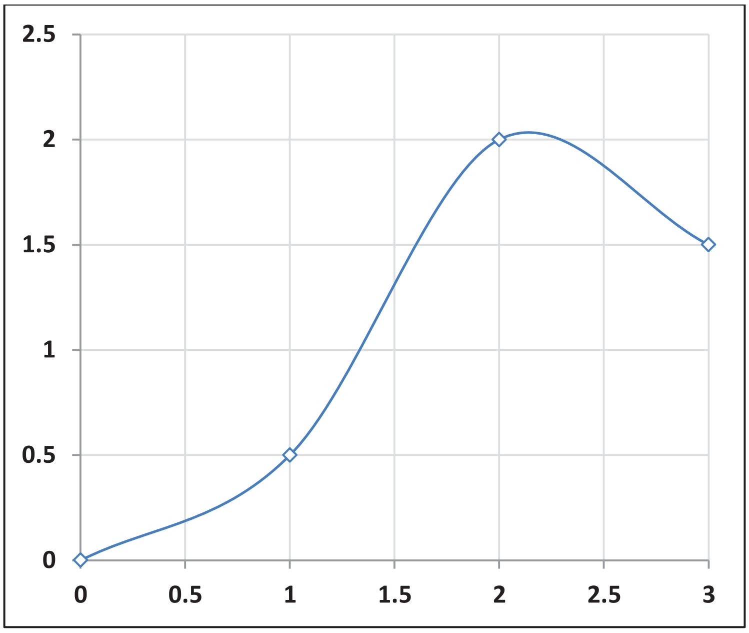
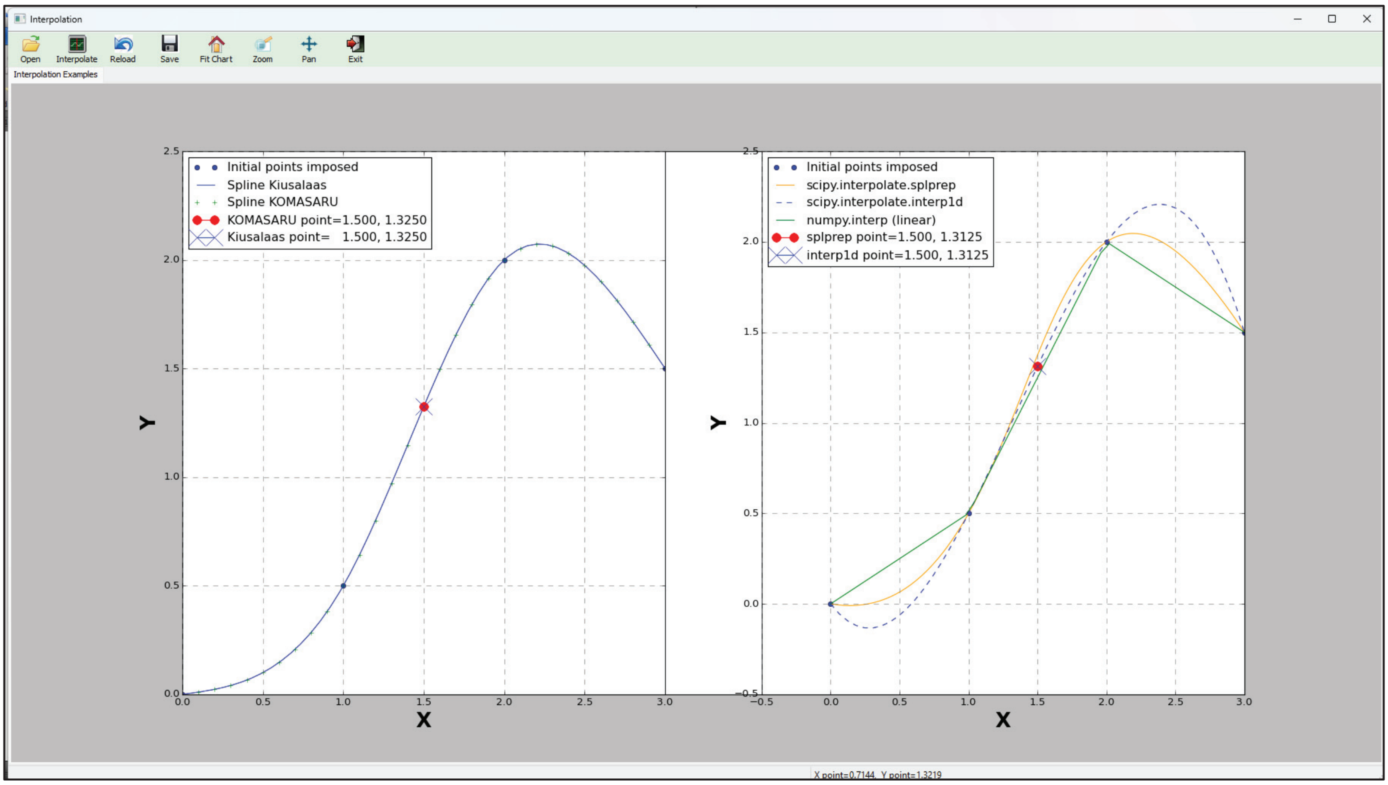


Figure 13.3 The Excel chart of the initial sample points



X point=0.7144, Y point=1.3219

Figure 13.4 The application interface, the spline curves of the initial sample points and marked with X=1.5 point (red color) calculated by spline functions

Listing 13.1 Interpolation.pyw – Spline Interpolation – Python & wxPython version

```
from __future__ import division

import os
import wx
import matplotlib.pyplot as plt
from sqlite3 import dbapi2 as sqlite
import numpy as np
from numpy import array
from scipy import interpolate
from scipy.interpolate import interp1d
import pandas as pd
import matplotlib as mpl
from win32com.client import Dispatch
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as Canvas
from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar
import StringIO
import win32con # Constants related to Win32 programming
import win32clipboard
import KIUSALAAS
import KOMASARU

def ExtragImageMemory(cursor, NumeImage):
    cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
    blob=cursor.fetchone()[0]
    img = wx.ImageFromStream(StringIO.StringIO(blob))
    return img

def Put_Clipboard(Sir_Clip):
    win32clipboard.OpenClipboard() # Open clipboard
    win32clipboard.EmptyClipboard() # Empty clipboard
    win32clipboard.SetClipboardData(win32con.CF_TEXT, Sir_Clip) # Set clipboard data
    win32clipboard.CloseClipboard()  # Close clipboard

class Plot(wx.Panel):
    def __init__(self, parent, id = -1, dpi = None, **kwargs):
        wx.Panel.__init__(self, parent, id=id, **kwargs)
        self.figure = mpl.figure.Figure(dpi=dpi, figsize=(2,2))
        self.canvas = Canvas(self, -1, self.figure)
        self.toolbar = Toolbar(self.canvas)
        self.toolbar.Realize()
        self.toolbar.Hide()

        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas,1,wx.EXPAND)
        sizer.Add(self.toolbar, 0 , wx.LEFT | wx.EXPAND)
        self.SetSizer(sizer)

class PlotNotebook(wx.Panel):
    def __init__(self, parent, id = -1):
        wx.Panel.__init__(self, parent, id=id)
        self.nb = wx.Notebook(self)
        sizer = wx.BoxSizer()
        sizer.Add(self.nb, 1, wx.EXPAND)
        self.SetSizer(sizer)

    def add(self,name="plot"):
        page = Plot(self.nb)
        self.nb.AddPage(page,name)
        return page.figure

    def ChangePagina(self,INDEX):
      # Change page in Notebook
      self.nb.ChangeSelection(INDEX)

# ==============================================================
class Interpolation(wx.Frame):
    # Interpolation is a method for generating points between given points.
    def __init__(self):
        displaySize= wx.DisplaySize()
        wx.Frame.__init__(self, None, -1, title="Interpolation",
            size=(displaySize[0]*0.95, displaySize[1]*0.95))
        self.statusbar = self.CreateStatusBar()  # Define statusBar
```

Listing 13.1 Interpolation.pyw – Spline Interpolation – Python & wxPython version

```
        self.statusbar.SetFieldsCount(2)
        self.statusbar.SetStatusWidths([-70, -50])

        self.plotter = PlotNotebook(self)  # Create Notebook
        self.AXA = self.plotter.add("Interpolation Examples").gca()
        self.AXA.set_xticklabels([]) ; self.AXA.set_yticklabels([])
        self.axes1 = self.AXA.figure.add_subplot(1, 2, 1)
        self.axes1.figure.canvas.mpl_connect('motion_notify_event', self.OnMouseMotion)
        self.axes2 = self.AXA.figure.add_subplot(1, 2, 2)
        self.axes2.figure.canvas.mpl_connect('motion_notify_event', self.OnMouseMotion)

        self.Recreate_Axis()
        self.X, self.Y = [], []
        self._TOOLBAR()
        self.CenterOnScreen()
        self.Show()

    def Recreate_Axis(self):
        self.AXE=[self.axes1, self.axes2]
        for ax in self.AXE:
            ax.cla()
            ax.grid(b=True, which='major', color='#666666', linestyle='--')
            ax.grid(b=True, which='minor', color='#999999', linestyle='--')
            ax.set_xlabel("X", fontsize=24, fontweight='bold', color="Black")
            ax.set_ylabel("Y", fontsize=24, fontweight='bold', color="Black")
        self.statusbar.SetStatusText("",0)

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.NO_BORDER | \
                                wx.TB_FLAT | wx.TB_3DBUTTONS | wx.TB_TEXT) )
        self.toolbar.SetToolBitmapSize((24,24))
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))

        connection = sqlite.connect(os.getcwd()+'\Config.db') ; cursor = connection.cursor()
        Lst_label=["Open", "Interpolate", "Reload", "Save", "Fit Chart", "Zoom", "Pan", "Exit"]
        Lst_icons=["open.png", "Interpolate.png", "Back.jpg", "save.png", "home.png", "zoom.png", "pan.png", "exit.png"]
        Lst_Help=["Open Excel file to read coordinates", "Spline interpolation for X imposed", "Reload the chart",
            "Save charts", "Fit chart in window", "Zoom chart rectangular region","Translate chart", "Exit application"]

        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            ID_tool=i # wx.NewId()
            img = ExtragImageMemory(cursor, icon)
            sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
            self.toolbar.AddLabelTool(ID_tool, label, sel_bmp, shortHelp=HELP)
            if label=="Open": self.Bind(wx.EVT_TOOL, self.OnOpenExcelFile, id=ID_tool)
            if label=="Interpolate": self.Bind(wx.EVT_TOOL, self.OnInterpolate, id=ID_tool)
            if label=="Reload": self.Bind(wx.EVT_TOOL, self.OnReload, id=ID_tool)
            if label=="Save": self.Bind(wx.EVT_TOOL, self.OnSave, id=ID_tool)
            if label=="Fit Chart": self.Bind(wx.EVT_TOOL, self.Home, id=ID_tool)
            if label=="Zoom": self.Bind(wx.EVT_TOOL, self.Zoom, id=ID_tool)
            if label=="Pan": self.Bind(wx.EVT_TOOL, self.Pan, id=ID_tool)
            if label=="Exit": self.Bind(wx.EVT_TOOL, self.OnClose, id=ID_tool)

        self.toolbar.Realize()   ; self.toolbar.Show()
        cursor.close() ; connection.close()

    def OnSave(self, event):
            img_file = os.getcwd()+'/Interpolation.jpg'
            self.axes1.figure.savefig(img_file)
            wx.MessageBox("Image saved in "+img_file, "Info", wx.OK | wx.ICON_INFORMATION)

    def OnReload(self, event):
        self.CHARTS()

    def OnInterpolate(self, event):
        if len(self.X)==0:
            MSJ="Please select an Excel file with X, Y coordinates !"
            dlg = wx.MessageDialog(self, MSJ, "Error", wx.OK | wx.ICON_ERROR)
            dlg.ShowModal()
            dlg.Destroy()
            return
        Vmed=str((max(self.X)-min(self.X))/2 + min(self.X))
```

Listing 13.1 Interpolation.pyw – Spline Interpolation – Python & wxPython version

```
        dlg = wx.TextEntryDialog(self, 'Between limits: '+str(min(self.X))+"..."+str(max(self.X)),
                'Enter X value for spline interpolation', Vmed)
        if dlg.ShowModal() == wx.ID_OK:
            Xinterp=float(str(dlg.GetValue()))

            si = KOMASARU.SplineInterpolation(self.X, self.Y )
            yK = si.interpolate(Xinterp)
            LBL1="KOMASARU point="+'%0.3f' % Xinterp+", "+'%0.4f' % yK
            self.axes1.plot(Xinterp, yK, marker=".", markersize=20, color="Red", label=LBL1)

            xData=array(self.X)  ;  yData=array(self.Y)
            kspline = KIUSALAAS.curvatures_Kiusalaas(xData, yData)
            Yspline =KIUSALAAS.evalSpline_Kiusalaas(xData, yData,kspline,Xinterp)
            LBL2="Kiusalaas point=   "+'%0.3f' % Xinterp+", "+'%0.4f' % Yspline
            self.axes1.plot(Xinterp, Yspline, marker="x", markersize=20, color="Blue", label=LBL2)

            splines = interpolate.splrep(self.X, self.Y )
            y2=interpolate.splev(Xinterp,splines)
            LBL3="splprep point="+'%0.3f' % Xinterp+", "+'%0.4f' % y2
            self.axes2.plot(Xinterp,y2, marker=".", markersize=20, color="Red", label=LBL3)

            f2 = interp1d(self.X, self.Y , kind='cubic')
            LBL4="interp1d point="+'%0.3f' % Xinterp+", "+'%0.4f' % f2(Xinterp)
            self.axes2.plot(Xinterp, f2(Xinterp), marker="x", markersize=20, color="Blue", label=LBL4)

            Put_Clipboard(LBL1+"\n"+LBL2+"\n"+LBL3+"\n"+LBL4)
            self.statusbar.SetStatusText("The interpolated values have been copied to Clipboard",0)
            self.LEGEND()

        dlg.Destroy()

    def LEGEND(self):
        for ax in self.AXE:
            ax.legend(fontsize=10)  ; ax.legend(loc='best') ; ax.figure.canvas.draw()

    def DrawInitialPoints(self, x,y):
        # Initial points imposed
        self.axes1.plot(x, y, 'ob', label="Initial points imposed")
        self.axes2.plot(x, y,'ob', label="Initial points imposed")

    def splprep(self, x,y):            # scipy.interpolate.splprep
        data = np.array((x,y))
        tck,u = interpolate.splprep(data, s=0)
        # s=0  forcing the spline fit to pass through all the input points
        unew = np.arange(0, 1.01, 0.01)
        out = interpolate.splev(unew, tck)
        self.axes2.plot(out[0], out[1], color='orange', label="scipy.interpolate.splprep")

    def Kiusalaas(self, x,y):  # Spline Kiusalaas
        xData=array(x)  ;  yData=array(y) # conversion from  'list' to 'vector'
        xspline,  yspline  = KIUSALAAS.Divizare_Spline_Kiusalaas(xData, yData, 50)
        self.axes1.plot(xspline, yspline, '-b', label="Spline Kiusalaas")

    def KOMASARU(self, x,y):          # Spline KOMASARU
        S   = 0.1       # Step for interpolation
        S_1 = 1 / S      # Inverse of S
        xs_K, ys_K = [], []  # List for graph
        si = KOMASARU.SplineInterpolation(x, y)
        for xK in [x / S_1 for x in range(int(x[0] / S), int(x[-1] / S) + 1)]:
            yK = si.interpolate(xK)
            xs_K.append(xK)
            ys_K.append(yK)
        self.axes1.plot(xs_K, ys_K, '+', color='green',label="Spline KOMASARU")

    def interp1d(self, x,y):
        f2 = interp1d(x, y, kind='cubic')
        xnew = np.linspace(min(x), max(x), num=50)
        self.axes2.plot(xnew, f2(xnew), '--', label="scipy.interpolate.interp1d ")

    def interp(self, x,y):
        xnew = np.linspace(min(x), max(x), num=50)
        ynew=[]
```

Listing 13.1 Interpolation.pyw – Spline Interpolation – Python & wxPython version

```
        for xx in xnew:
            ynew.append(np.interp(xx, x, y))
        self.axes2.plot(xnew, ynew, '-', color='green',label="numpy.interp (linear)")

    def OnOpenExcelFile(self, event):
        wildcard="Excel files (*.xlsx)|*.xlsx|Excel files (*.xls)|*.xls"
        dialog = wx.FileDialog(None, "Open Excel file to read curve coordinates", "", "", wildcard, wx.OPEN)
        if dialog.ShowModal() == wx.ID_CANCEL:
            return
        self.Excel_File= dialog.GetPath().strip().upper()
        xlApp = Dispatch("Excel.Application")   # Connect to Excel
        xlApp.Visible=False
        xlWb = xlApp.Workbooks.Open(self.Excel_File)
        sht1 = xlWb.Worksheets(1)
        lex=2  ;  self.X=[]  ;  self.Y=[]
        while True:
            xx=str(sht1.Cells(lex,1).Value).strip()  ;  yy=str(sht1.Cells(lex,2).Value).strip()
            if xx=="" or yy=="None":
                break
            else:
                self.X.append(float(xx))  ;  self.Y.append(float(yy))
                lex+=1
        xlWb.Close(SaveChanges=0)
        xlApp.Quit()
        self.CHARTS()

    def CHARTS(self):
        self.Recreate_Axis()
        self.DrawInitialPoints(self.X, self.Y )
        self.Kiusalaas(self.X, self.Y )
        self.KOMASARU(self.X, self.Y )
        self.splprep(self.X, self.Y )

        self.interp1d(self.X, self.Y )
        self.interp(self.X, self.Y )

        self.LEGEND()

    def Home(self,event):
        self.axes1.figure.canvas.toolbar.home()

    def Pan(self,event):
        self.axes1.figure.canvas.toolbar.pan()

    def Zoom(self,event):
        self.axes1.figure.canvas.toolbar.zoom()

    def OnMouseMotion(self, event):
        xMouse,yMouse = event.xdata, event.ydata
        if xMouse<>None and yMouse<> None:
            SirReal="X point="+'%0.4f' % xMouse + ",  Y point="+'%0.4f' % yMouse
            self.statusbar.SetStatusText( SirReal,1)

    def OnClose(self, event):
        self.Close(True)
        self.Destroy()

#-------------------------------
if __name__ == '__main__':
    app = wx.App(redirect=0)
    Interpolation()
    app.MainLoop()
#-------------------------------
```

The **Interpolation.py** script (Python & Streamlit version) contains 146 lines and begin with importing libraries (lines 1÷9). Lines 11÷16 configures the title settings of the script and the max-width of the page (70 rem = 1120 px, where 1 rem = 16px). Lines 18 define the the script path.

Lines 20÷27 define **splprep** function which calculate the B-spline representation of an N-D curve [**13.4**]. Line 31 wait to select an Excel file image by the user to read the sample points coordinates in a dataframe (line

33÷34). The coordinates readed from Excel will be displayed as table and the Excel file name is also displayed (**Figure 13.5**). Lines 35÷36 convert columns X & Y from dataframe to list. Line 37 create a form where is displayed **min(X)** and **max(X)** and wait to input a value **XXX** (lines 42÷43) where spline functions will interpolate the Y coordinate; in the form is proposed the median value **Vmed** between **min(X)** and **max(X),** but the user can change this value, **Figure 13.5**. Also, a **submit_button** is created (line 44). If the button is clicked, two columns are created by line 47. Lines 50÷53 define **fig1** figure, set the grid and title. Line 55 plot the sample points coordinates in **fig1**. Line 57 call **splprep** function which calculate the B-spline curve plotted by line 59 in **fig1**. Lines 61÷62 interpolate the **YP_splev** value in **XXX** point by **splprep** function. This point (**XXX**, **YP_splev**) will be marked in **fig1** by line 64. Line 66 place **fig1** in **col1** (top-left position of **Figure 13.6**). Lines 69÷77 calculate the spline interpolated curves of sample data points using **scipy** library type splines: **CubicSpline** & **akima** & **pchip**.

The **CubicSpline** type function "*interpolate data with a piecewise cubic polynomial which is twice continuously differentiable*" [13.5].

"*The **Akima** interpolation function fit piecewise cubic polynomials, for given vectors x and y. The interpolation method by **Akima** uses a continuously differentiable sub-spline built from piecewise cubic polynomials. The resultant curve passes through the given data points and will appear smooth and natural*" [13.6].

"***PCHIP** 1-D monotonic cubic interpolation; x and y are arrays of values used to approximate some function f, with y = f(x). The interpolant uses monotonic cubic splines to find the value of new points. (**PCHIP** stands for Piecewise Cubic Hermite Interpolating Polynomial)* " [13.7].

Lines 80÷83 extract **X_Interp, Y_spline**, **Y_akima**, **Y_pchip** interpolated list values from the three interpolation: ***CubicSpline*** & ***akima*** & ***pchip***. Lines

Lines 86÷99 calculate **Point_spline**, **Point_akima**, **Point_pchip** point interpolation for **XXX** value using **CubicSpline** & **akima** & **pchip** functions.

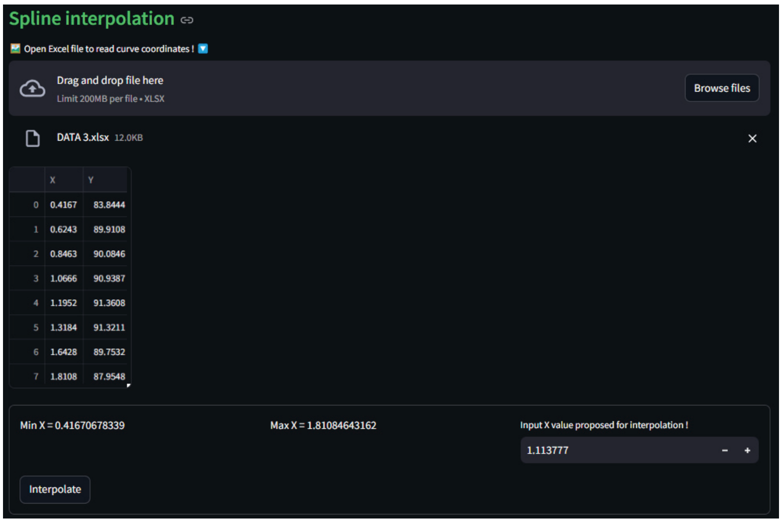
Lines 102÷105 define **fig2** figure, set the grid and title. Line 106 plot the sample points coordinates in **fig2**. Line 107 plot **CubicSpline** curve and line 108 mark (**XXX**, **Point_spline**) point. Line 110 place **fig2** in **col2** (top-right position of **Figure 13.6**).

Next, two columns are created by line 112. Lines 114÷117 define **fig3** figure, set the grid and title. Line 118 plot the sample points coordinates in **fig3**. Line 119 plot **Akima** curve and line 129 mark (**XXX**, **Point_akima**) point. Line 122 place **fig3** in **col1** (bottom-left position of **Figure 13.6**).

Lines 124÷127 define **fig4** figure, set the grid and title. Line 128 plot the sample points coordinates in **fig4**. Line 129 plot **pchip** curve and line 130mark (**XXX**, **Point_ pchip**) point. Line 132 place **fig3** in **col2** (bottom-right position of **Figure 13.6**).

Line 135 create 4 columns, where are placed **X_Interp, Y_spline**, **Y_akima**, **Y_pchip** lists in table format.

Line 142 create 4 columns, where are placed **XXX** and **LBL2, LBL3, LB4 labels** of the interpolated points values, created in lines 90, 95 and 100.



Spline interpolation

Open Excel file to read curve coordinates !

DATA 3.xlsx 12.0KB

	X	Y
0	0.4167	83.8444
1	0.6243	89.9108
2	0.8463	90.0846
3	1.0666	90.9387
4	1.1952	91.3608
5	1.3184	91.3211
6	1.6428	89.7532
7	1.8108	87.9548

Min X = 0.41670678339

Max X = 1.81084643162

Input X value proposed for interpolation !

1.113777

Figure 13.5
Upload Excel file with the initial sample points, show the coordinates table and wait for an user input for point interpolate

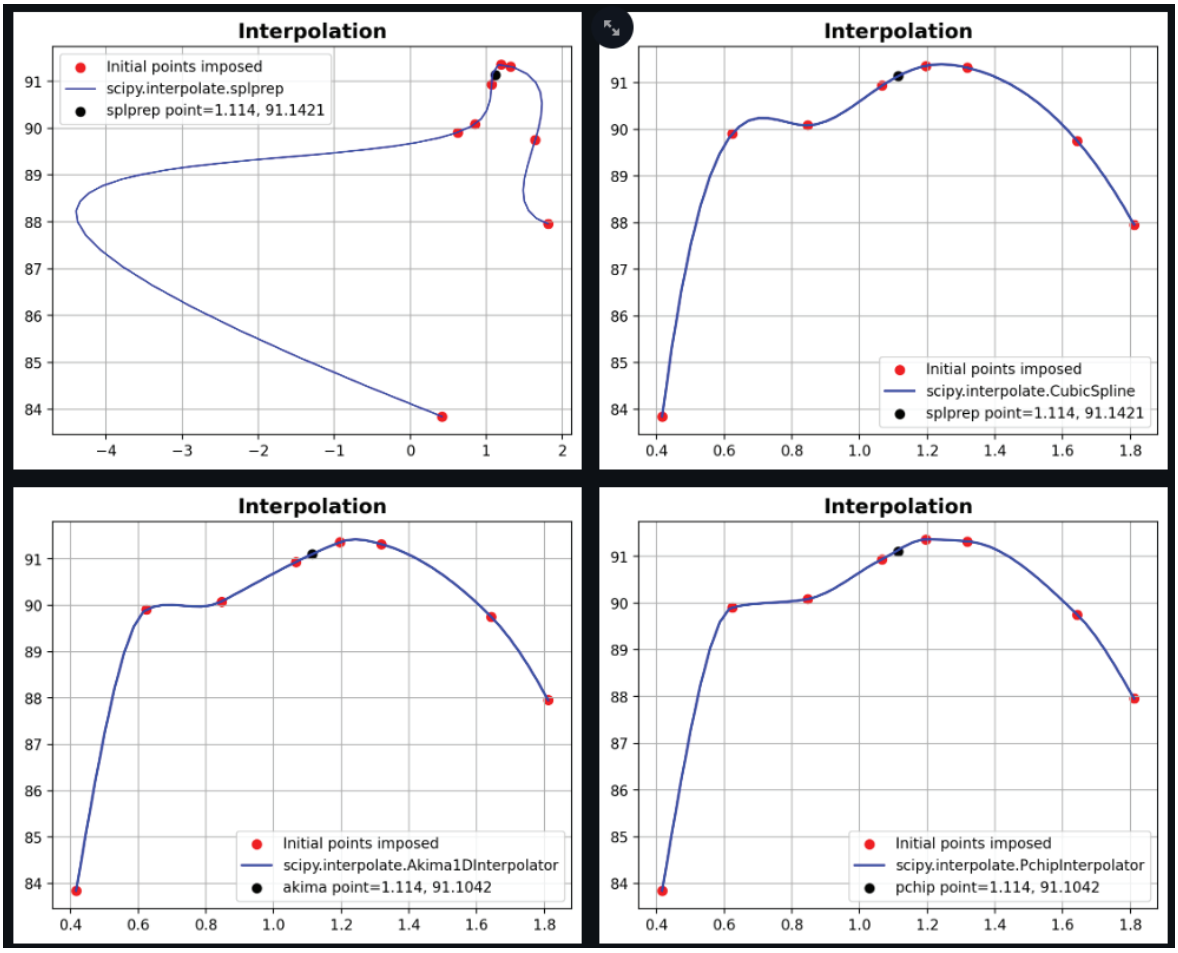
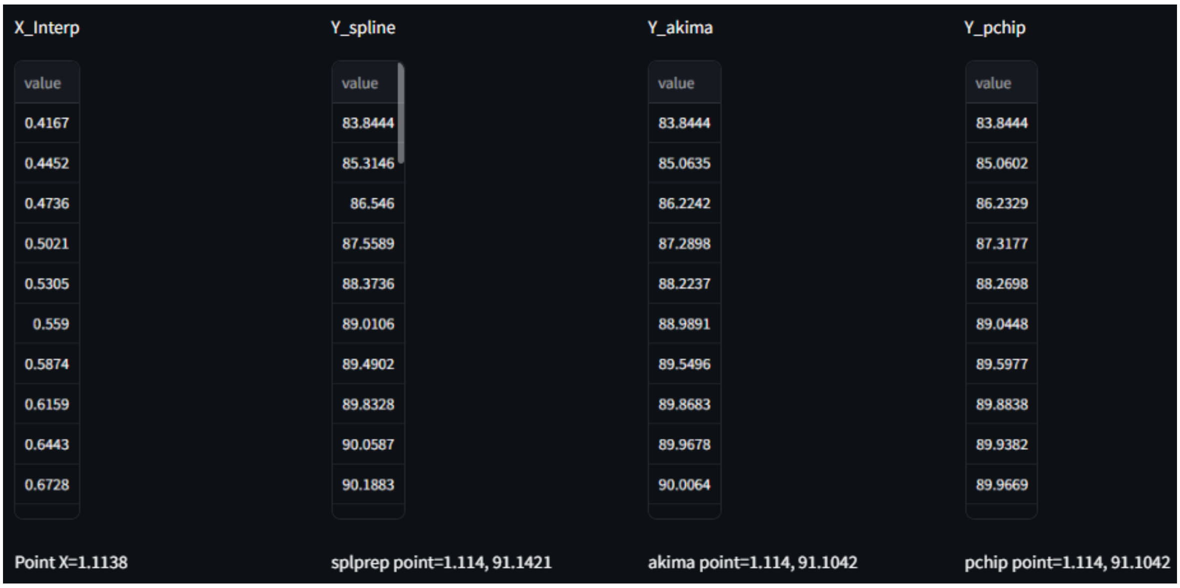


Figure 13.6 The spline interpolated curves



X_Interp	Y_spline	Y_akima	Y_pchip
value	value	value	value
0.4167	83.8444	83.8444	83.8444
0.4452	85.3146	85.0635	85.0602
0.4736	86.546	86.2242	86.2329
0.5021	87.5589	87.2898	87.3177
0.5305	88.3736	88.2237	88.2698
0.559	89.0106	88.9891	89.0448
0.5874	89.4902	89.5496	89.5977
0.6159	89.8328	89.8683	89.8838
0.6443	90.0587	89.9678	89.9382
0.6728	90.1883	90.0064	89.9669

Point X=1.1138 splprep point=1.114, 91.1421 akima point=1.114, 91.1042 pchip point=1.114, 91.1042

Figure 13.7 The spline interpolated tables and interpolated points

From **Figure 13.6** results that the shape of the spline curves is similar, except for the **splprep** variant, which shows an exaggerated deformation on the initial portion. As can be seen from the charts, the interpolated point values for **XXX**=1.1138 are: **splprep** point=1.1138, 91.1421, **CubicSpline** point=1.1138, 911421, **akima** point=1.1138, 91.1042, and **pchip** point=1.1138, 91.1042.

In conclusion, the use of spline functions offers a precise interpolation possibility, but the graphical visualization of the curve is a must to avoid situations like the one in the **Figure 13.6**.

Figure 13.8 The show the table of initial sample points readed from an Excel file and **Figure 13.9** show the Excel chart of these points.

X	Y
0.416707	83.84439
0.624285	89.91082
0.84631	90.08463
1.066597	90.93865
1.195182	91.36078
1.318358	91.32108
1.642826	89.75317
1.810846	87.95477

Figure 13.8 The initial sample points readed from an Excel file

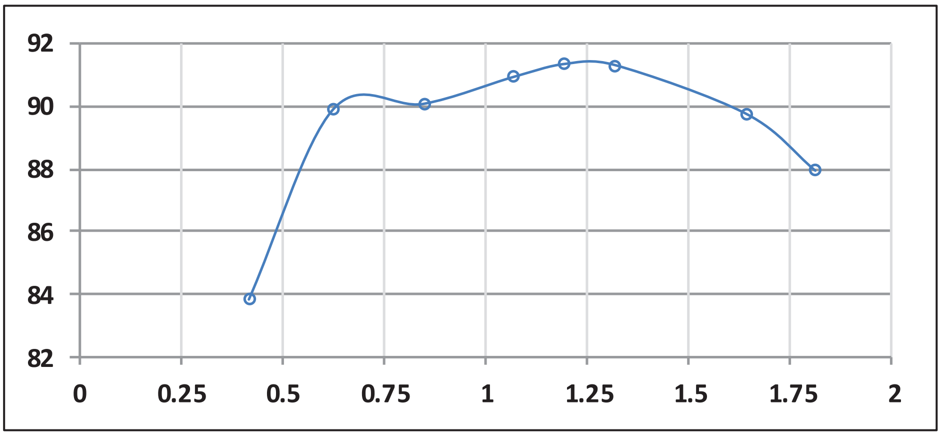


Figure xx1310.9 The Excel chart of the initial sample points

Listing 13.2 Interpolation.py – Spline Interpolation – Python & Streamlit version

```python
import streamlit as st
import numpy as np
from numpy import array
import matplotlib.pyplot as plt
import pandas as pd
from pathlib import Path
import scipy
from scipy import interpolate   # pip install scipy
from scipy.interpolate import interp1d

st.set_page_config(page_title="Spline interpolation")
css='''
<style>
    section.main > div {max-width:70rem}  # 1rem = 16px ; 70rem = 1120 px
</style>
'''
st.markdown(css, unsafe_allow_html=True)
current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()

def splprep(x,y):        # scipy.interpolate.splprep
    data = np.array((x,y))
    tck,u = interpolate.splprep(data, s=0)
    # s=0  forcing the spline fit to pass through all the input points
    unew = np.arange(0, 1.02, 0.02)
    out = interpolate.splev(unew, tck)
    lst_return=[out[0],  out[1]]
    return lst_return

st.subheader(":green[Spline interpolation]")
MSJ=":frame_with_picture: Open Excel file to read curve coordinates ! :arrow_down_small:"
uploaded_file = st.file_uploader(MSJ, type="xlsx")
if uploaded_file:
    df = pd.read_excel(uploaded_file)
    st.dataframe(df) # Display the table with profile coordinates
    X = df['X'].tolist() # Convert columns X from dataframe to list
    Y = df['Y'].tolist() # Convert columns Y from dataframe to list
    with st.form('Input Data'):
        col1, col2, col3 = st.columns(3)
        col1.write('Min X = '+str(min(X)))
        col2.write('Max X = '+str(max(X)))
        Vmed=(max(X)-min(X))/2 + min(X)
        XXX = col3.number_input(label='Input X value proposed for interpolation !' \
                ,value=Vmed,format="%0.6f")
        submit_button = st.form_submit_button('Interpolate')
    if submit_button:
```

Listing 13.2 Interpolation.py – Spline Interpolation – Python & Streamlit version

```
    # Charts drawing
    col1, col2 = st.columns(2)
    # cm = 1/2.54  # centimeters in inches
    # fig1=plt.figure(figsize=(6*cm, 6*cm))
    fig1=plt.figure()
    plt.grid(True)
    plt.title("Interpolation", fontsize=14,
        fontweight='bold',color='Black') # Define chart title
    # Initial points imposed
    plt.scatter(X, Y, color='Red', label="Initial points imposed")
    # Curve interpolation - splprep
    LST_RETURN=splprep(X, Y)
    xspline = LST_RETURN[0]  ;  yspline = LST_RETURN[1]
    plt.plot(xspline, yspline, '-b', linewidth=1, label="scipy.interpolate.splprep")
    # Point interpolation - splprep & splev
    splines = interpolate.splrep(X, Y)
    YP_splev=interpolate.splev(XXX,splines)
    LBL1="splprep point="+'%0.3f' % XXX+", "+'%0.4f' % YP_splev
    plt.scatter(XXX, YP_splev, color='Black', label=LBL1)
    plt.legend(fontsize=6)  ; plt.legend(loc='best')
    col1.write(fig1)

    # Curve interpolation scipy.interpolate: CubicSpline & akima & pchip
    data = {"x": X, "y": Y}
    interpolated = {"x": np.linspace(min(X), max(X), 50)}
    # interpolated["linear"] = np.interp(interpolated["x"], data["x"], data["y"])
    interpolated["spline"] = scipy.interpolate.CubicSpline(data["x"], data["y"])(
        interpolated["x"] )
    interpolated["akima"] = scipy.interpolate.Akima1DInterpolator(data["x"], data["y"])(
        interpolated["x"] )
    interpolated["pchip"] = scipy.interpolate.PchipInterpolator(data["x"], data["y"])(
        interpolated["x"] )

    # Extract list values from the three interpolation: CubicSpline & akima & pchip
    X_Interp = interpolated["x"]
    Y_spline = interpolated["spline"]
    Y_akima = interpolated["akima"]
    Y_pchip = interpolated["pchip"]

    # Point interpolation scipy.interpolate: CubicSpline & akima & pchip
    inter_Point = {"x": XXX}
    interpolated["spline"] = scipy.interpolate.CubicSpline \
        (data["x"], data["y"])(inter_Point["x"] )
    Point_spline = interpolated["spline"]
    LBL2="splprep point="+'%0.3f' % XXX+", "+'%0.4f' % Point_spline

    interpolated["akima"] = scipy.interpolate.Akima1DInterpolator \
        (data["x"], data["y"])(inter_Point["x"] )
    Point_akima = interpolated["akima"]
    LBL3="akima point="+'%0.3f' % XXX+", "+'%0.4f' % Point_akima

    interpolated["pchip"] = scipy.interpolate.Akima1DInterpolator \
        (data["x"], data["y"])(inter_Point["x"] )
    Point_pchip = interpolated["pchip"]
    LBL4="pchip point="+'%0.3f' % XXX+", "+'%0.4f' % Point_pchip

    fig2=plt.figure() # Create chart figure
    plt.grid(True) # Show the chart grid
    plt.title("Interpolation", fontsize=14,
    fontweight='bold',color='Black') # Define chart title
    plt.scatter(X, Y, color='Red', marker="o", label="Initial points imposed")
    plt.plot(X_Interp, Y_spline, color='Blue', label="scipy.interpolate.CubicSpline")
    plt.scatter(XXX, Point_spline, color='Black', label=LBL2)
    plt.legend(fontsize=6)  ; plt.legend(loc='best')
    col2.write(fig2)

    col1, col2 = st.columns(2)

    fig3=plt.figure() # Create chart figure
    plt.grid(True) # Show the chart grid
    plt.title("Interpolation", fontsize=14,
    fontweight='bold',color='Black') # Define chart title
```

Listing 13.2 Interpolation.py – Spline Interpolation – Python & Streamlit version

```
plt.scatter(X, Y, color='Red', marker="o", label="Initial points imposed")
plt.plot(X_Interp, Y_akima, color='Blue', label="scipy.interpolate.Akima1DInterpolator")
plt.scatter(XXX, Point_akima, color='Black', label=LBL3)
plt.legend(fontsize=6)  ; plt.legend(loc='best')
col1.write(fig3)

fig4=plt.figure() # Create chart figure
plt.grid(True) # Show the chart grid
plt.title("Interpolation", fontsize=14,
fontweight='bold',color='Black') # Define chart title
plt.scatter(X, Y, color='Red', marker="o", label="Initial points imposed")
plt.plot(X_Interp, Y_pchip, color='Blue', label="scipy.interpolate.PchipInterpolator")
plt.scatter(XXX, Point_pchip, color='Black', label=LBL4)
plt.legend(fontsize=6)  ; plt.legend(loc='best')
col2.write(fig4)

# Show interpolated list values
col1, col2, col3, col4 = st.columns(4)
col1.write("X_Interp")  ;  col1.write(X_Interp)
col2.write("Y_spline")  ;  col2.write(Y_spline)
col3.write("Y_akima")  ;  col3.write(Y_akima)
col4.write("Y_pchip")  ;  col4.write(Y_pchip)

# Show interpolated point values
col1, col2, col3, col4 = st.columns(4)
col1.write("Point X="+ '%0.4f' % XXX)
col2.write(LBL2)
col3.write(LBL3)
col4.write(LBL4)
```

14. Curve vectorization

The **Vectorization** script is a simple and educational application to vectorize one curve and to obtaing numerical points on it. This script is a shorter version of an complex script PyDigitizer [**14.1**], [**14.2**] that can be download from [**14.3**]. A similar script is presented in [**14.4**], [**14.5**]. The vectorization steps are the following:

○ open an image file containing the curve to vectorize;

○ in the calibration window enter the real coordinates of the calibration points bottom-left respectively top–right;

○ after saving, two graphical calibration points must be specified, by left mouse click in the graphic area;

○ next, place multiple points above the curve by left mouse click, starting from left to right X axis;;

○ it is possible to delete or move points points: middle mouse click - delete near mouse point (without confirmation) while right mouse click - move or exit move near mouse points.

○ at the end of placement points it is possible to visualize the curve passing through the placed points;

○ the placed points can also be exported in Excel together with the graphic image over which the points have been placed.

The script use a database **Config.db** to store all icons loaded in frame **Toolbar**; this will avoid the existence of following icons files in the application folder: 📂, 👁, 📊 and 🚪. The **Config.db → objects** store icons for the script interface of **Python 2. 7 & wxPython** application version. Initially, the icons files are stored in the **Curve vectorization → Objects** application folder. The icons files were loaded into **Config.db → objects** as **Binary Large Object** (**BLOB**) through the **_copyimg.pyw** script displayed in **Listing 8.1**, from where they will be used into application interface through **Python 2.7 → Vectorization pyw** script.

The **Vectorization.pyw** script (Python & wxPython version) include four classes: **Plot**, **PlotNotebook**, **CalibrareAxe** & **Vectorization** and three public functions: **ExportImageToExcel**, **ExtragTextMemory**, **ExtragImageMemory**. The script begin with importing libraries (lines 3÷11).

The **Plot** class (lines 34÷44) create **self.figure**, **self.canvas**, **self.toolbar** and **sizer** entities for the class that calls it. The **PlotNotebook** class (lines 46÷60) creates several pages for the class that calls it, using the internal function **add**; through internal function **ChangePagina** the user can change the page. This class represents a notebook control, which manages multiple windows with associated tabs.

The **ExportImageToExcel** public function export an image Excel file. The parameters of the function are: the file **Path** – the path were the image file is saved, **Sheet** – the sheet were the image will be placed and **row** & **column** which identify the cell were the image will be placed in Excel sheet.

The **ExtragImageMemory** function (lines 28÷32) selecteaza din **SQLite database → Objects** an icon si create & returneaza the icon **img** from data in memory. This function is called from class **Vectorization → _TOOLBAR** function.

The **ExtragTextMemory**function (lines 23÷26) selecteaza din **SQLite database → Objects** a file and returneaza this file as **BLOB** object. This function is called from class **Vectorization → OnExcel** function.

The **Vectorization** class create the interface (toolbar, grid, …), create the aplication main functionalities and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user); create the statusbar (lines 68÷73); line 74 create in **self.plotter** an instance of **PlotNotebook** class; line 75 create one graphics subplot **self. Axa** that will be connected to **OnClick and OnMouseMotion** class internal function (lines 76, 77); lines 78, 79 initialize the public variables of the class **ExcelFile** and **self.FOLDER_FILE** (folder where is located **Config.db** file); line 80 call **_TOOLBAR** and **INITIALIZARE** function; lines 81 center and show the frame in the display.
- **_TOOLBAR** Create the **toolbar** (line 230); set the **toolbar** background color (line 231) and icons size (line 232); create the connection and cursor using **Config.db** database (line 233); generate **Lst_label**, **Lst_icons** and **Lst_Help** lists (lines 234÷237), which store the **tooolbar** icons labels, the icon names that must be extracted from memory by calling **ExtragImageMemory** function and the associated icons help; for every line 239 associate an unique identificator ID; through **for** cycle from line 238 the **img** icon is extracted into memory (ine 240); create the bitmap of **img** memory icon and scale at the imposed **toolbar** icons size (line 241); in

this cycle, for every operation, the icon is added in **toolbar** (line 242) ; lines 243÷247 connect with **Bind** statement to associated functions; line 248 generate and show the **toolbar;** line 249 close the connection with **Config.db** database.

- **OnOpen** Open a dialog to load an image file (line 96) and store the filename in variable **image_file** line (100); read the image file and get the height and width (line 102); line 103 set the value of **self.ExcelFile** public variable; lines 104÷106 define and set the frame title; line 107 put the image in **self.axa** graphic area; finally, the **CalibrareAxe** class is called to calibrate the image; the parameter **self** is send to this class in order to be able to access from inside the **CalibrareAxe** class variable of the **Vectorization** class, more precisely the coordinates of two points in the graph, which define the location bottom-left and top-right points of the image; also the **FOLDER_FILE** variable is send to **CalibrareAxe** class.
- **Calibrare** Call **CalibrareAxe** class (line 252) to calibrate the image; the correspondence between calibration points (graphical and real) is show in **Figure 14.1**. The constants **Kx** & Ky in relation (1) result from the correspondence of the points in the graphic (pixel) and the real plane; through relations (2) the coordinates from the real plane can be computed as a function of the coordinates in the graphical plane; through relations (3) the coordinates from the graphical plane can be computed as a function of the coordinates in the real plane. In the script the graphical (mouse) coordinates are stored in (**XXX, YYY**) public variables; first two positions in **XXX** & **YYY** list are occupied by calibration points and in the next position are stored the click mouse curve coordinates; the real points curve coordinates are stored in (**Crd_X**, **Crd_Y**) public variables.
- **Pixel_To_Graph** Convert coordinates from the graphic plane to coordinate in real plane with relations (2)
- **Graph_To_Pixel** Convert coordinates from the real plane to coordinate in graphic plane with relations (3).
- **DeletePoint** Delete the point closest to the mouse cursor (lines 188÷201). Lines 188 initialize the local variables **MINIM**, **Xdel**, **Ydel**. Inside the **for** loop from line 189 the **MINIM** minimum distance between mouse points stored in (**XXX, YYY**) public variables are computed together with position **INDEX** and the coordinates **Xdel** & **Ydel** (line 192). The graphical point is removed from **ARTISTS** list (line 193) and the chart is redraw (line 194); the **ARTISTS** list store all points entities placed on graphical area. With **pop** command (lines 195÷196) are removed the values from **ARTISTS**, **XXX** and **YYY** lists; if is identified a point to be deleted (line 197) this pixel point is converted to real coordinates (line 198) to be displayed into statusbar (line 199) together with current points number **No_Points** (line 200÷201).
- **INITIALIZARE** Initialize the class public variables (lines 84, 87, 88, 90), delete and redraw graphical area (**self.axa –** line 85, 86), clear the status bar in areas 2 & 3 (line 91) and write in status bar arfeas 4 the value of **No_Points**=0. This function is called by **__init__** and **OnOpen** **Vectorization** class functions. The variables **XRcal1**, **YRcal1**, **XRcal2**, **YRcal2** store the values of the real calibration points coordinates (bottom-left respectively top–right).
- **OnClick** Identify the current mouse position (line 113); if the left mouse was clicked and the move operation is not active (line 115) is tested if the calibration is active by line 116 case in which both variables **X_is_between** & **Y_is_between** are set to **True**; if calibration operation is not active (line 119) lines 120, 121 test if the mouse coordinate X & Y are between calibration points; is both condition are **True** (line 122) a point is marked in chart area (lines 123, 124), the **ARTIST**, **XXX** & **YYY** list are appended with this mouse point; if calibration points exist (line 128) the current points real coordinates are computed (line 129) and the point is appended to **Crd_X**, **Crd_Y** lists (line 130); if the middle mouse is clicked (line 134) the is called the **DeletePoint** function; for the right mouse click (line 136) is tested if move operation is not active (line 137) and the variable **MOVE** is set to **MOVE START** while variable **Calc_MINIM** is set to **DA** (line 138); this means that the current mouse point is activated for move operation in **OnMouseMotion** function; if move operation is active (line 140) the move operation is closed by setting **MOVE** variable to **""** (line 141).
- **OnExcel** Verify if points are placed on graphical area (lines 204÷207); connect to **Config.db** database to extract the Excel template file **Template_Results.xls** (line 210) using **ExtragTextMemory** function and write this file on disk with **write(blob)** command (line 211); Open saved file

(line 213) and activate **Chart_Data** sheet (line 215) ; create Excel table coordinates headings (line 216) and write in a **for** loop the real coordinates (**Crd_X**, **Crd_Y**) (lines 218÷220); lines 223÷224 save the current chart as image file which is exported to Excel file (line 225) using **ExportImageToExcel** function; line 226 save and close the Excel file (line 226) and reopen & shown by line 227.

- **OnMouseMotion** Identify the current mouse position (line 162); if a real point is clicked and is not active calibration phase (line 163) the real coordinates of the clicked point are calculated (line 164);
  If these real coordinates values falls between X and Y real calibration points (line 165, 166, 168) the status bar is updated with them (line 169-170); if move point operation is activated in **OnClick** function (**MOVE=="MOVE START"** - line 171) the closest point to cursor mouse is identified (lines 173÷178) and the point index is stored in **INDEX** variable; from **ARTIST** list is extracted this point and the current mouse coordinates are assigned to it (181÷182); the chart area is updated to see the point movement in the chart (line 183).
- **OnView** Verify if points are placed on graphical area (lines 255÷258); create a new figure (line 259), set the X, Y labels and title (lines 260÷262); inside a **for** loop (line 263) the real points coordinates are placed as text (lines 264÷266); line 267 plot the curve.
- **OnClose** Exit from application.

The interface of **Vectorization** class is displayed in **Figure 14.2**. **Listing 14.1** displays the **Python & wxPython** version of script, **Vectorization.pyw**.

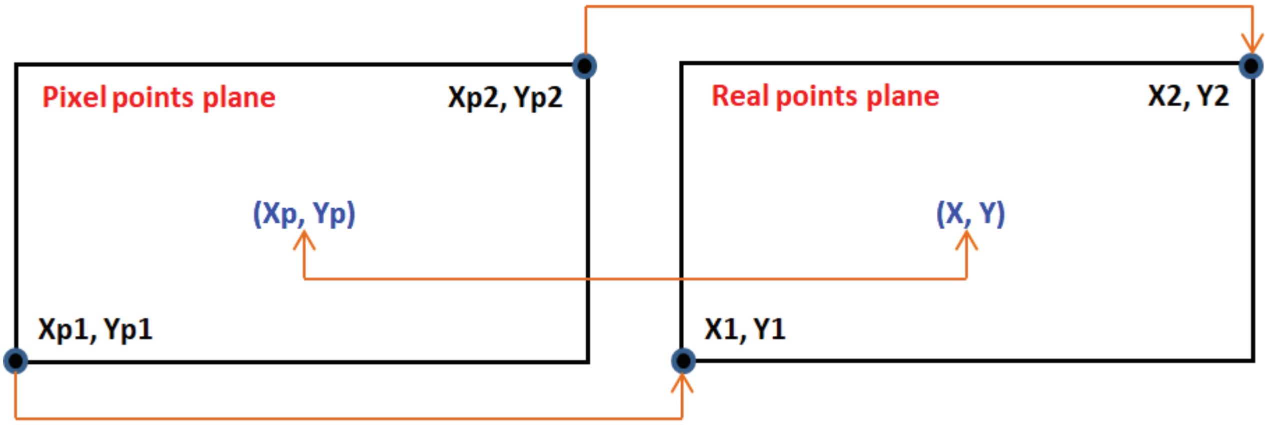
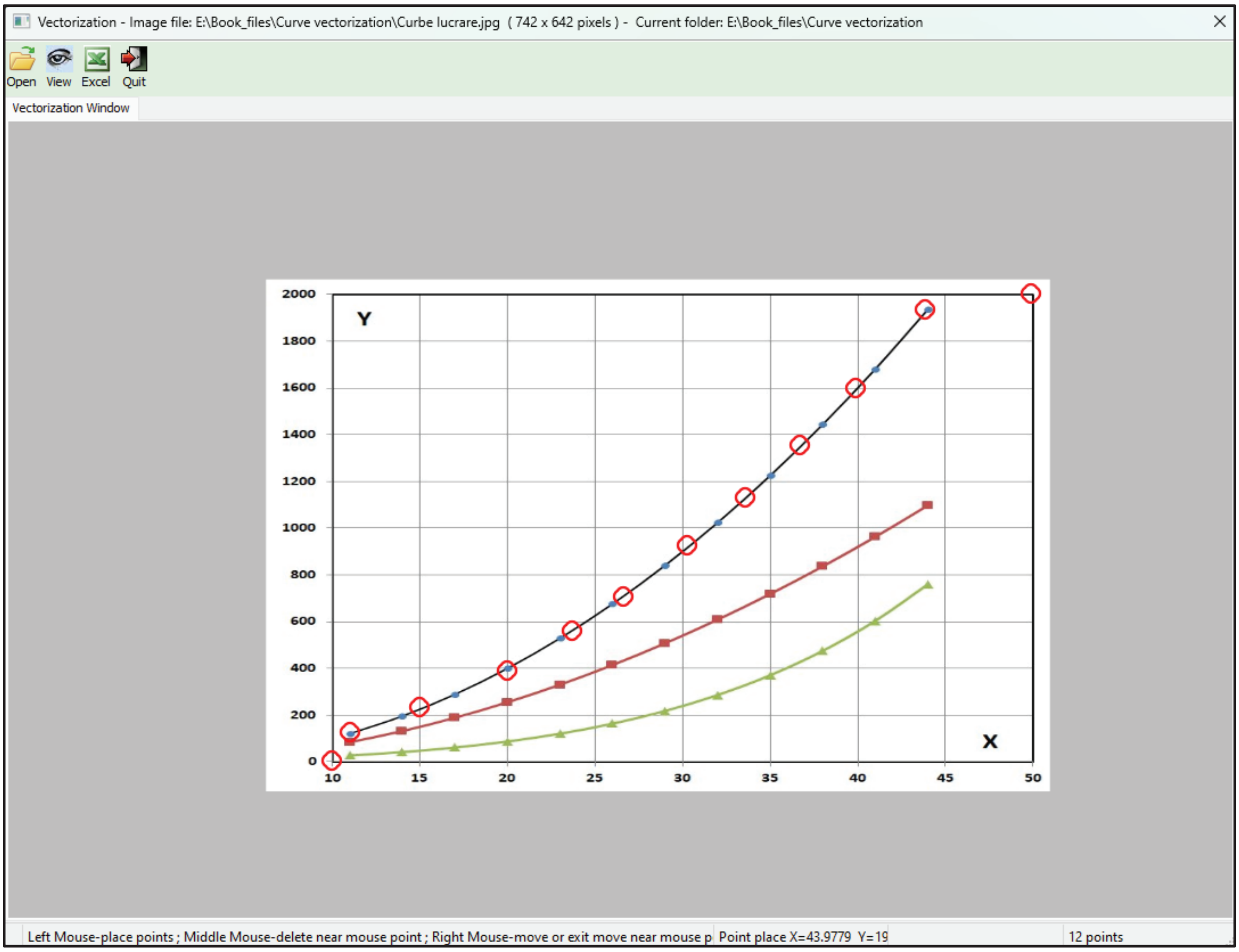


Figure 14.1 The correspondence between calibration points (graphical and real)

$$\begin{cases} X_{p2} - X_{p1} = K_x \cdot (X_2 - X_1) \rightarrow & K_x = \dfrac{X_{p2} - X_{p1}}{(X_2 - X_1)} \\ Y_{p2} - Y_{p1} = K_y \cdot (Y_2 - Y_1) \rightarrow & K_y = \dfrac{Y_{p2} - Y_{p1}}{(Y_2 - Y_1)} \end{cases} \qquad (1)$$

$$\begin{cases} X = X_1 + (X_p - X_{p1})/K_x \\ Y = Y_1 + (Y_p - Y_{p1})/K_y \end{cases} \qquad (2)$$

$$\begin{cases} X_p = X_{p1} + (X - X_1) \cdot K_x \\ Y_p = Y_{p1} + (Y - Y) \cdot K_y \end{cases} \qquad (3)$$



Vectorization - Image file: E:\Book_files\Curve vectorization\Curbe lucrare.jpg (742 x 642 pixels) - Current folder: E:\Book_files\Curve vectorization

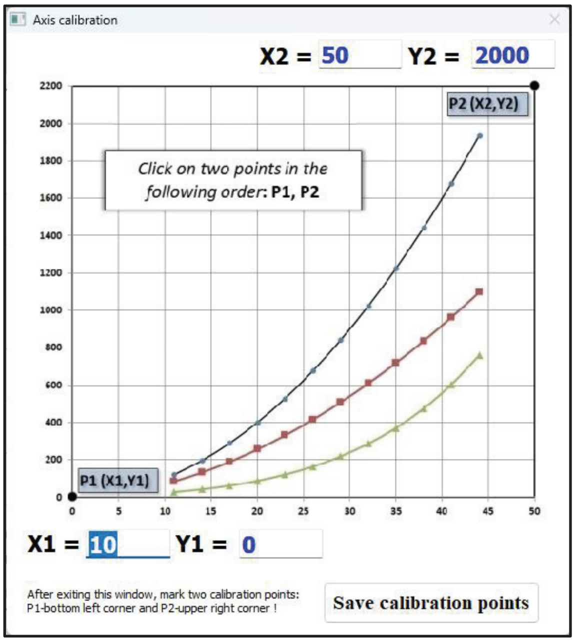
Left Mouse-place points ; Middle Mouse-delete near mouse point ; Right Mouse-move or exit move near mouse p Point place X=43.9779 Y=19 12 points

Figure 14.2 The Vectorization class interface

The CalibrareAxeclass wait to input the real coordinates of the calibration points and has two functions:

- __init__ Define the frame (a window whose size and position can be changed by the user); this class receive from Vectorization class two parameters: frm_parinte which is the class that called it and FOLDER_FILE which is the folder where Config.db is located. Create the connection and cursor using Config.db database (line 280) to extract Demo Calibrare.jpg image (Figure 14.3); create four text controls to input the real calibration coordinates : XC1_Graph, YC1_Graph, XC2_Graph, YC2_Graph (lines 286, 289, 294, 296), the Info StaticText to display the message from bottom-left window and the save button connected through Bind with OnExitCalibrare function; the user must input and save the P1 & P2 real calibration coordinates; lines 306 center and show the frame in the display; also this line make modal the window.
- OnExitCalibrare Read the real calibration coordinates values from text controls and save these values to XRcal1, YRcal1, XRcal2, YRcal2 public variables of the Vectorization class (lines 309÷312); set
MakeModal to False and close this window.

Figure 14.4 show the OnView plot of Vectorization class. Figure 14.5 show the OnExcel result of Vectorization class.



X2 = 50 Y2 = 2000

Click on two points in the following order: **P1, P2**

X1 = 10 Y1 = 0

After exiting this window, mark two calibration points: P1-bottom left corner and P2-upper right corner !

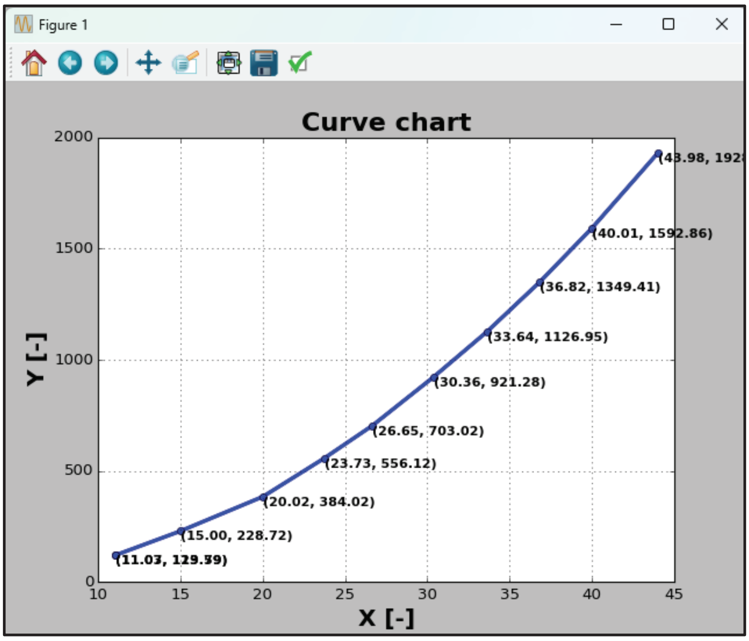
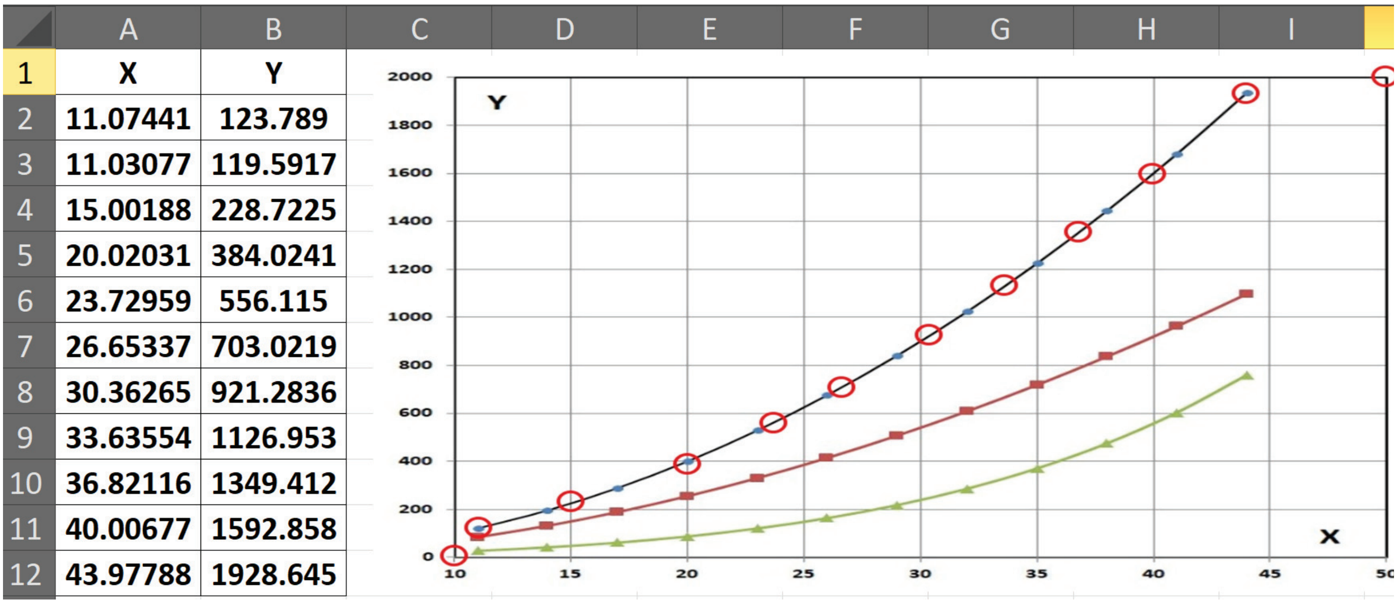


Figure 14.3 The **CalibrareAxe** class interface

Figure 14.4 The **OnView** plot of **Vectorization** class



	A	B
1	X	Y
2	11.07441	123.789
3	11.03077	119.5917
4	15.00188	228.7225
5	20.02031	384.0241
6	23.72959	556.115
7	26.65337	703.0219
8	30.36265	921.2836
9	33.63554	1126.953
10	36.82116	1349.412
11	40.00677	1592.858
12	43.97788	1928.645

Figure 14.5 The **OnExcel** result of **Vectorization** class

Listing 14.1 Vectorization.pyw – Vectorization curve – Python & wxPython version

```
 from __future__ import division

from win32com.client import Dispatch
from pysqlite2 import dbapi2 as sqlite
import StringIO,cStringIO
import wx , os, math
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as Canvas
from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar

def ExportImageToExcel(PathImage, Sheet, row, column):
        data = open(PathImage, "rb").read()
        stream = cStringIO.StringIO(data)   # convert to a data stream
```

Listing 14.1 Vectorization.pyw – Vectorization curve – Python & wxPython version

```
    sel_bmp=wx.BitmapFromImage( wx.ImageFromStream( stream ))  # convert to a bitmap
    width=sel_bmp.GetWidth() ; height=sel_bmp.GetHeight()
    left=Sheet.Cells(1,column).Left  ;  top=Sheet.Cells(row,1).Top
    Sheet.Shapes.AddPicture(PathImage, False, True,left, top,width,height)
    os.remove(PathImage)
    return

def ExtragTextMemory(cursor, NumeText):
    cursor.execute("SELECT fisier FROM objects where nume='"+NumeText+"'")
    blob=str(cursor.fetchone()[0])
    return blob

def ExtragImageMemory(cursor, NumeImage):
    cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
    blob=cursor.fetchone()[0]
    img = wx.ImageFromStream(StringIO.StringIO(blob))
    return img

class Plot(wx.Panel):
    def __init__(self, parent, id = -1, dpi = None, **kwargs):
        wx.Panel.__init__(self, parent, id=id, **kwargs)
        self.figure = mpl.figure.Figure(dpi=dpi)
        self.canvas = Canvas(self, -1, self.figure)
        self.toolbar = Toolbar(self.canvas)
        self.toolbar.Realize() ; self.toolbar.Hide()
        sizer = wx.BoxSizer(wx.VERTICAL) ;
        sizer.Add(self.canvas,1,wx.EXPAND)
        sizer.Add(self.toolbar, 0 , wx.LEFT | wx.EXPAND)
        self.SetSizer(sizer)

class PlotNotebook(wx.Panel):
    def __init__(self, parent, id = -1):
        wx.Panel.__init__(self, parent, id=id)
        self.nb = wx.Notebook(self)
        sizer = wx.BoxSizer() ; sizer.Add(self.nb, 1, wx.EXPAND)
        self.SetSizer(sizer)
    def add(self,name="plot"):
        page = Plot(self.nb)
        self.nb.AddPage(page,name)
        return page.figure
    def add_PAGE(self,name):
        Pagina=wx.Panel(self.nb) ; self.nb.AddPage(Pagina, name)
        return Pagina
    def hide_PAGE(self,number):
        self.nb.RemovePage(number)

class Vectorization(wx.Frame):  # ==============================================
    def __init__(self):
        displaySize= self.DS=wx.DisplaySize()
        wx.Frame.__init__(self, None, -1, title="Vectorization",
            size=(displaySize[0]*0.75, displaySize[1]*0.75),
            style=wx.DEFAULT_FRAME_STYLE ^ ( wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
        self.SB=self.SB = self.CreateStatusBar()   # StatusBar
        self.SB.SetFieldsCount(5)
        self.SB.SetStatusWidths([-1,-40, -10, -10, -10])
        sir="Left Mouse-place points ; Middle Mouse-delete near mouse point"
        sir=sir+" ; Right Mouse-move or exit move near mouse points"
        self.SB.SetStatusText(sir,1)
        self.plotter = PlotNotebook(self)
        self.axa = self.plotter.add("Vectorization Window").gca()
        self.axa.figure.canvas.mpl_connect('button_press_event', self.OnClick)
        self.axa.figure.canvas.mpl_connect('motion_notify_event', self.OnMouseMotion)
        self.ExcelFile=""
        self.FOLDER_FILE=os.getcwd()+'\Config.db'
        self._TOOLBAR() ; self.INITIALIZARE()
        self.CenterOnScreen()  ;  self.Show()

    def INITIALIZARE(self):
        self.MOVE="" ; self.Calc_MINIM="NU" ; self.INDEX=-9999 ; self.No_Points=0
        self.axa.cla() ; self.axa.grid(False) ; self.axa.axis('off')
        self.axa.figure.canvas.draw()
        self.ARTISTS=[] ; self.Crd_X=[] ; self.Crd_Y=[]  # Real coordinates
```

Listing 14.1 Vectorization.pyw – Vectorization curve – Python & wxPython version

```
        self.XXX=[] ; self.YYY=[]  # Coordinates in pixels image
        # Real coordinates calibration points
        self.XRcal1=-1e9 ; self.YRcal1=-1e9 ; self.XRcal2=-1e9 ; self.YRcal2=-1e9
        self.SB.SetStatusText("",2) ; self.SB.SetStatusText("",3)
        self.SB.SetStatusText(str(self.No_Points)+ " points",4)

    def OnOpen(self,event):
        wildcard="Image files (*.jpg)|*.jpg|Image files (*.jpeg)|*.jpeg|Image files (*.png)|*.png"
        dlg = wx.FileDialog(self, "Select image file", os.getcwd(), "",wildcard , wx.OPEN)
        result = dlg.ShowModal() ; dlg.Destroy()
        if result == wx.ID_OK:
            image_file=dlg.GetDirectory()+'\\'+dlg.GetFilename()
            file_name = image_file[0 : image_file.find(".")]
            IMG_NAME=dlg.GetFilename()[0 : dlg.GetFilename().find(".")]
            image = plt.imread(image_file) ; height, width, _ = image.shape
            self.ExcelFile=IMG_NAME+".xls"
            TITLE="Vectorization - Image file: " + str(image_file) + "  ( "+str(width)+" x "
            TITLE=TITLE+str(height)+" pixels )"+" -  Current folder: " + str(os.getcwd())
            self.SetTitle( TITLE )  ;  self.INITIALIZARE()
            self.axa.imshow(image, aspect='auto') ; self.axa.figure.canvas.draw()
            CalibrareAxe(None, -1, self, self.FOLDER_FILE)

    def OnClick(self, event):
        self.SB.SetStatusText("",2)
        if event.inaxes is None: return
        Btn_MOUSE=event.button  ;  X_MOUSE, Y_MOUSE=event.xdata, event.ydata
        # Left mouse - put a marker on image, if the 'MOVE' operation is not active
        if Btn_MOUSE==1 and not("MOVE" in self.MOVE):
            if len(self.ARTISTS)<2 :
                curba="Calibrare"
                X_is_between = True ; Y_is_between = True
            elif len(self.ARTISTS)>=2:
                X_is_between = self.XXX[0] <= X_MOUSE <= self.XXX[1]
                Y_is_between = self.YYY[1] <= Y_MOUSE <= self.YYY[0]
            if X_is_between==True and Y_is_between==True:   # Place calibration point
                MARKER, = self.axa.plot(X_MOUSE, Y_MOUSE, 'o', markersize=14,
                    mfc='none', markeredgecolor="Red", markeredgewidth=2)
                self.ARTISTS.append(MARKER,) ; self.XXX.append(X_MOUSE)
                self.YYY.append(Y_MOUSE)  ;  self.axa.figure.canvas.draw()
                self.MOVE=""  ;  self.Calc_MINIM="DA"
                if len(self.ARTISTS)>2: # If calibration points exist
                    Xreal, Yreal = self.Pixel_To_Graph (X_MOUSE, Y_MOUSE)
                    self.Crd_X.append(Xreal) ; self.Crd_Y.append(Yreal)
                    self.SB.SetStatusText("Point place X="+'%0.4f' % Xreal+"  Y="+'%0.4f' % Yreal,2)
                self.No_Points=len(self.XXX)
                self.SB.SetStatusText(str(self.No_Points)+ " points",4)
        if Btn_MOUSE==2 and len(self.ARTISTS) > 2 : # Middle mouse - disable move marker
            self.DeletePoint(X_MOUSE, Y_MOUSE) ;  self.MOVE=""
        if Btn_MOUSE==3: # Right mouse - activate muve marker
            if self.MOVE=="":
                self.MOVE="MOVE START"  ; self.Calc_MINIM="DA"
                self.SB.SetStatusText("Move START",2)
            else:
                self.MOVE="" ; self.SB.SetStatusText("Move STOP",2)

    def Pixel_To_Graph(self, x, y):
        # Real coordinates
        X1=self.XRcal1 ; Y1=self.YRcal1 ; X2=self.XRcal2 ; Y2=self.YRcal2
        # Coordinates in pixels image
        Xp1=self.XXX[0] ; Yp1=self.YYY[0] ; Xp2=self.XXX[1] ; Yp2=self.YYY[1]
        kx=(Xp2-Xp1)/(X2-X1) ; ky=(Yp2-Yp1)/(Y2-Y1)
        Xreal=X1+(x-Xp1)/kx ; Yreal=Y1+(y-Yp1)/ky
        return Xreal, Yreal

    def Graph_To_Pixel(self, Xreal, Yreal):
        # Real coordinates
        X1=self.XRcal1 ; Y1=self.YRcal1 ; X2=self.XRcal2 ; Y2=self.YRcal2
        # Coordinates in pixels image
        Xp1=self.XXX[0] ; Yp1=self.YYY[0] ; Xp2=self.XXX[1] ; Yp2=self.YYY[1]
        kx=(Xp2-Xp1)/(X2-X1) ; ky=(Yp2-Yp1)/(Y2-Y1)
        x =(Xreal-X1)*kx+Xp1 ; y =(Yreal-Y1)*ky+Yp1
        return x, y
```

Listing 14.1 Vectorization.pyw – Vectorization curve – Python & wxPython version

```

    def OnMouseMotion(self, event):
        x,y = event.xdata, event.ydata
        if x<>None and y<> None and len(self.ARTISTS)>=2:
            Xreal, Yreal = self.Pixel_To_Graph (x, y)
            X_is_between = self.XRcal1 <= Xreal <= self.XRcal2
            Y_is_between = self.YRcal1 <= Yreal<= self.YRcal2
            self.SB.SetStatusText( "", 3)
            if X_is_between==True and Y_is_between==True:
                SirReal="X chart="+'%0.4f' % Xreal + ",  Y chart="+'%0.4f' % Yreal
                self.SB.SetStatusText( SirReal,3)
        if self.MOVE=="MOVE START":
            # Point identification by minimum distance
            if x<>None and y<> None and self.Calc_MINIM=="DA":
                MINIM=1e7 ; self.INDEX=-999
                for i, (XP, YP) in enumerate(zip(self.XXX, self.YYY )):
                    distanta=math.sqrt((XP-x)**2+(YP-y)**2)
                    if distanta<MINIM:
                        MINIM=distanta  ;  self.INDEX= i
                self.Calc_MINIM=="NU" # Point identification done only for MouseMotion
            if len(self.ARTISTS)<>0 and self.INDEX <> -999:
                punct=self.ARTISTS[self.INDEX]  ;  punct.set_data(x,y)
                self.XXX[self.INDEX]=x  ;  self.YYY[self.INDEX]=y
                self.axa.figure.canvas.draw()
                # When the point starts to move, it no longer needs to be identified
                self.Calc_MINIM="NU"

    def DeletePoint(self,x,y):
        MINIM=1e7 ; XDel=-1e7 ; YDel=-1e7
        for i, (XP, YP, PCT) in enumerate(zip(self.XXX, self.YYY, self.ARTISTS )):
            distanta=math.sqrt((XP-x)**2+(YP-y)**2)
            if distanta<MINIM:
                MINIM=distanta ; INDEX = i ; XDel=XP ; YDel=YP
        punct=self.ARTISTS[INDEX]  ;  punct.remove()   # Remove marker from graph
        self.axa.figure.canvas.draw()
        self.ARTISTS.pop(INDEX)  # Remove point from lists
        self.XXX.pop(INDEX) ; self.YYY.pop(INDEX)
        if XDel<>-1e7:
            Xreal, Yreal = self.Pixel_To_Graph (XDel, YDel)
            self.SB.SetStatusText("Point deleted X="+'%0.4f' % Xreal+"  Y="+'%0.4f' % Yreal,2)
        self.No_Points=len(self.XXX)
        self.SB.SetStatusText(str(self.No_Points)+ " points",4)

    def OnExcel(self, event):
        if len(self.Crd_X)==0:
            mesaj = "No point defined on chart !"
            wx.MessageBox(mesaj, "Info",style=wx.OK|wx.ICON_EXCLAMATION)
            return
        fex1=self.ExcelFile
        conn = sqlite.connect(self.FOLDER_FILE) ; cursor = conn.cursor()
        blob=ExtragTextMemory(cursor, 'Template_Results.xls')
        cursor.close() ; conn.close()  ;  open(fex1, 'wb').write(blob)
        xlApp = Dispatch("Excel.Application")
        xlWb = xlApp.Workbooks.Open(os.getcwd()+'\\'+fex1) ; sheets = xlWb.Sheets
        XLSheet=xlWb.Worksheets("Chart_Data")  # Export chart points
        sheets("Chart_Data").Activate()
        XLSheet.Cells(1,1).Value = "X" ; XLSheet.Cells(1,1).Value = "Y"
        SirClip1="" ; lex=2
        for i, (xt, yt) in enumerate(zip(self.Crd_X, self.Crd_Y)):
            XLSheet.Cells(lex,1).Value = xt ; XLSheet.Cells(lex,2).Value = yt
            lex+=1
        XLSheet.Cells(lex,1).Select()
        CaleFig=os.getcwd()+'\\'+"ChartFigure.jpg"
        self.axa.figure.savefig(CaleFig, orientation='landscape', bbox_inches=None,
            transparent=True, pad_inches=0, format ="jpg", quality = 95)
        ExportImageToExcel(CaleFig, xlWb.Worksheets("Chart_Data"), 1, 6)
        xlWb.Close(SaveChanges=1) ; xlApp.Quit()
        xlWb = xlApp.Workbooks.Open(os.getcwd()+'\\'+fex1) ; xlApp.Visible=True

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))
```

Listing 14.1 Vectorization.pyw – Vectorization curve – Python & wxPython version

```
        self.toolbar.SetToolBitmapSize((24,24))
        conn = sqlite.connect(self.FOLDER_FILE) ; cursor = conn.cursor()
        Lst_label=["Open","View","Excel","Quit"]
        Lst_icons=["open.png","Show.jpg","Excel.png", "exit.png"]
        Lst_Help=["Open image file.","View current curve chart",
                    "Export coordinates to Excel","Quit application"]
        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            ID_tool= wx.NewId()
            img = ExtragImageMemory(cursor, icon)
            sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
            self.toolbar.AddLabelTool(ID_tool, label, sel_bmp, shortHelp=HELP)
            if label=="Open": self.Bind(wx.EVT_TOOL, self.OnOpen, id=ID_tool)
            if label=="View": self.Bind(wx.EVT_TOOL, self.OnView, id=ID_tool)
            if label=="Erase": self.Bind(wx.EVT_TOOL, self.OnErase, id=ID_tool)
            if label=="Excel": self.Bind(wx.EVT_TOOL, self.OnExcel, id=ID_tool)
            if label=="Quit": self.Bind(wx.EVT_TOOL, self.OnClose, id=ID_tool)
        self.toolbar.Realize() ; self.toolbar.Show()  # Generate toolbar
        cursor.close() ; conn.close()

    def Calibrare(self, event):
        frm_parinte=self  ;  CalibrareAxe(None, -1, frm_parinte)

    def OnView(self, event):  # Plotting the Graph
        if len(self.XXX)<3:
            mesaj = "At least 3 points must be digitized for the selected curve !"
            wx.MessageBox(mesaj, "Info",style=wx.OK|wx.ICON_EXCLAMATION)
            return
        plt.figure(figsize=(8,6)) ; plt.grid(True)
        plt.xlabel("X [-]", fontsize=18, fontweight='bold')
        plt.ylabel("Y [-]", fontsize=18, fontweight='bold')
        plt.title("Curve chart",color='Black', fontsize=20, fontweight='bold')
        for i, (xt, yt) in enumerate(zip(self.Crd_X, self.Crd_Y)):
            sirtxt='('+'%.2f' % xt+', '+'%.2f' %  yt+')'
            plt.text(xt, yt, sirtxt, ha='left', va= 'top',
                color='Black', fontsize=10, fontweight='bold')
        plt.plot(self.Crd_X, self.Crd_Y, '-o', linewidth=3) ; plt.show()

    def OnClose(self, event):
            self.Destroy()

class CalibrareAxe(wx.Frame):
    def __init__(self,parent, id, frm_parinte, FOLDER_FILE):
        wx.Frame.__init__(self, None, -1, "Axis calibration", size=(540, 600),
                style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
                wx.MAXIMIZE_BOX | wx.CLOSE_BOX ) )
        self.frm_parinte=frm_parinte  ;  self.FOLDER_FILE = FOLDER_FILE
        self.SetBackgroundColour("White") ; font=wx.Font(16, wx.DEFAULT, wx.NORMAL, wx.BOLD)
        connection = sqlite.connect(self.FOLDER_FILE) ; cursor = connection.cursor()
        img = ExtragImageMemory(cursor, 'Demo Calibrare.jpg')
        png=wx.BitmapFromImage(img.Scale(490,426))
        cursor.close() ; connection.close()
        self.Imagine=wx.StaticBitmap(self, -1, png, (20, 38), (png.GetWidth(), png.GetHeight()))
        XC1_Graph=wx.StaticText(self, -1, 'X1 =', pos=(20, 471)) ; XC1_Graph.SetFont(font)
        self.XC1_Graph=wx.TextCtrl(self, -1, "10", pos=(75, 470), size=(80, 28), style=wx.TE_LEFT)
        self.XC1_Graph.SetForegroundColour('Blue') ; self.XC1_Graph.SetFont(font)
        YC1_Graph=wx.StaticText(self, -1, 'Y1 = ', pos=(160, 471)) ; YC1_Graph.SetFont(font)
        self.YC1_Graph=wx.TextCtrl(self, -1, "0", pos=(220, 470), size=(80, 28), style=wx.TE_LEFT)
        self.YC1_Graph.SetForegroundColour('Blue')
        self.YC1_Graph.SetFont(font) ; self.YC1_Graph.SetInsertionPoint(0)
        XC2_Graph=wx.StaticText(self, -1, 'X2 =', pos=(240, 9)) ; XC2_Graph.SetFont(font)
        self.XC2_Graph=wx.TextCtrl(self, -1, "50", pos=(295, 6), size=(80, 28), style=wx.TE_LEFT)
        self.XC2_Graph.SetForegroundColour('Blue') ; self.XC2_Graph.SetFont(font)
        YC2_Graph=wx.StaticText(self, -1, 'Y2 = ', pos=(380, 9)) ; YC2_Graph.SetFont(font)
        self.YC2_Graph=wx.TextCtrl(self, -1, "2000", pos=(440, 6), size=(80, 28), style=wx.TE_LEFT)
        self.YC2_Graph.SetForegroundColour('Blue')
        self.YC2_Graph.SetFont(font) ; self.YC2_Graph.SetInsertionPoint(0)
        sir="After exiting this window, mark two calibration points:\n"
        sir=sir+"P1-bottom left corner and P2-upper right corner !"
        Info=wx.StaticText(self, -1,sir , pos=(20, 525), size=(280,100))
        btn_ExitInfo= wx.Button(self, -1, "Save calibration points", pos=(300, 520), size=(205,40))
        btn_ExitInfo.SetForegroundColour('Black')
        btn_ExitInfo.SetFont(wx.Font(14, wx.ROMAN, wx.NORMAL, wx.BOLD))
```

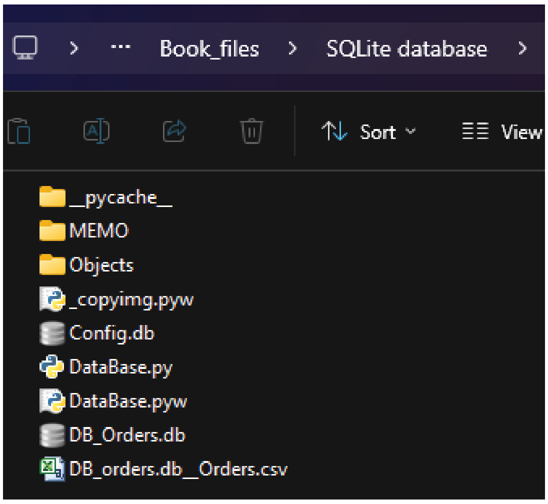
Listing 14.1 Vectorization.pyw – Vectorization curve – Python & wxPython version

```
        self.Bind(wx.EVT_BUTTON, self.OnExitCalibrare, btn_ExitInfo)
        self.CenterOnScreen() ; self.MakeModal(True) ; self.Show()
    def OnExitCalibrare(self, event):
        # Save real coordinates of calibration points P1 & P2
        self.frm_parinte.XRcal1 = float(self.XC1_Graph.GetValue())
        self.frm_parinte.YRcal1 = float(self.YC1_Graph.GetValue())
        self.frm_parinte.XRcal2 = float(self.XC2_Graph.GetValue())
        self.frm_parinte.YRcal2 = float(self.YC2_Graph.GetValue())
        self.MakeModal(False) ; self.Close(True)

if __name__ == '__main__':   #====================================
    app = wx.App(0)
    Vectorization()
    app.MainLoop()
```

15. SQLite database

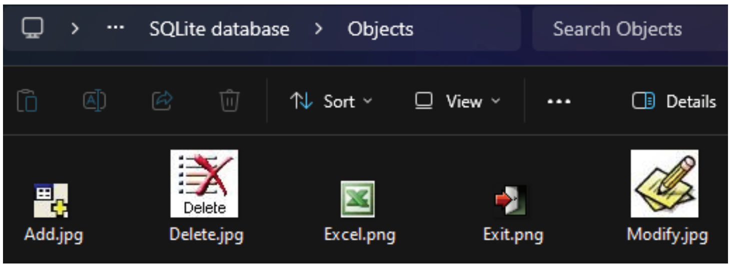
This script saved in **SQLite database** folder, **Figure 15.1**, is designed to manage a SQLite database for the following operations: add, delete, modify record and export database to Excel.

The name of main database is **DB_orders.db** with only one table **Orders**, extracted from original database **northwind.db** [15.1]. Only 7 fields have been extracted for the **Orders** table to avoid increasing the script in number of instructions. The **SQL** to generate **Orders** table is displayed below, **Figure 15.3**. Another database **Config.db** is used to store all icons loaded in frame **Toolbar**; this will avoid the existence of following icons files in the application folder: , Delete , , and . The **SQL** to generate **objects** table is displayed in **Figure 15.4**. The **DB_orders.db** → **Orders** memorize the usefull informations for the script and **Config.db** → **objects** memorize icons for the script interface for **Python 2.7 & wxPython** application version. Initially, the icons files are stored in the **SQLite database** → **Objects** application folder, **Figure 15.2**. The icons files were loaded into **SQLite database** → **Objects** as **Binary Large Object** (**BLOB**) through the **_copyimg.pyw** script displayed in **Listing 8.1**, from where they will be used into application interface through **Python 2.7** → **DataBase.pyw** script. For **Python & Streamlit** → **DataBase.py** script version icon files have not been used.



- _pycache_
- MEMO
- Objects
- _copyimg.pyw
- Config.db
- DataBase.py
- DataBase.pyw
- DB_Orders.db
- DB_orders.db__Orders.csv

Figure 15.1 The content of application folder **SQLite database**

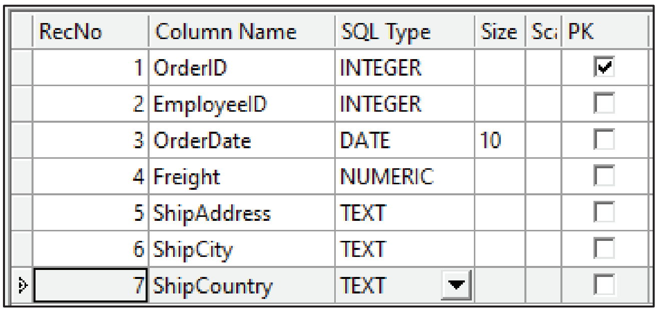


- Add.jpg
- Delete.jpg
- Excel.png
- Exit.png
- Modify.jpg

Figure 15.2 The content of icons folder **SQLite database** → **Objects**

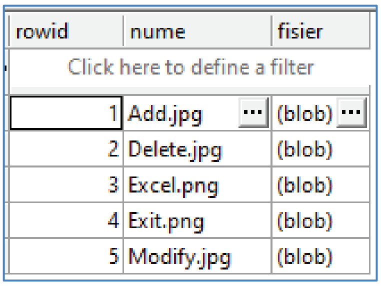
```
CREATE TABLE "Orders"(
        [OrderID] INTEGER PRIMARY KEY AUTOINCREMENT,
        [EmployeeID] INTEGER, [OrderDate] DATE(10),
        [Freight] NUMERIC, [ShipAddress] TEXT,
        [ShipCity] TEXT, [ShipCountry] TEXT);
```

```
CREATE TABLE 'objects' (
        [nume] CHAR(20) NOT NULL,
        [fisier] BLOB NOT NULL);
```



RecNo	Column Name	SQL Type	Size	Sc	PK
1	OrderID	INTEGER			☑
2	EmployeeID	INTEGER			☐
3	OrderDate	DATE	10		☐
4	Freight	NUMERIC			☐
5	ShipAddress	TEXT			☐
6	ShipCity	TEXT			☐
7	ShipCountry	TEXT ▼			☐

Figure 15.3 The **SQL** to generate **Orders** table in **DB_orders.db** database



rowid	nume	fisier
Click here to define a filter		
1	Add.jpg ···	(blob) ···
2	Delete.jpg	(blob)
3	Excel.png	(blob)
4	Exit.png	(blob)
5	Modify.jpg	(blob)

Figure 15.4 The **SQL** to generate **objects** table in **Config.db** database:

Listing 15.1 _copyimg.pyw - Load images from OBJECTS folder to Config.db → objects

```
# Python 2.7 version
from pysqlite2 import dbapi2 as sqlite
import os, glob
con = sqlite.connect(os.getcwd()+'\Config.db')
cur = con.cursor()
cur.execute( "DELETE from objects")
con.commit()
Sursa_fisiere=os.getcwd()+"/OBJECTS/*"
counter=0
for name in glob.glob(Sursa_fisiere):
   nf= os.path.basename(name)
   counter=counter+1
   fisier=os.path.basename(name)
   blobdata = open(os.getcwd()+'\\OBJECTS\\'+fisier,'rb').read()
   cur.execute("Insert into objects(nume, fisier) values (?,?)",(fisier,sqlite.Binary(blobdata)))
   print counter, '     ',name
con.commit() ; cur.close() ; con.close()
```

To operate with databases outside the scripts, to create databases, tables or to view/modify/verify their contents, a number of freeware applications are available on the Internet: **SQLite Expert Personal,** which is freeware and does not have an expiration date [**15.2**];

- **DB Browser for SQLite,** which is freeware and does not have an expiration date [**15.3**];
- **SQLiteStudio,** which is free to use for any purpose [**15.4**].

Listing 15.2 displays the **Python & wxPython** version of script, **DataBase.pyw** and **Listing 15.3** displays the **Python & Streamlit** version of the same script, **DataBase.py**.

The **DataBase.pyw** script (Python & wxPython version) include only two classes: **Class_DB**, **AddModifyRecord** and 1 public function (**ExtragImageMemory**). The script begin with importing libraries: **os** (this module provides a portable way of using operating system dependent functionality), **sqlite** (a C-language library that implements a small, fast, self-contained, high-reliability, full-featured), **wx** (the cross-platform GUI toolkit for the Python language), **wx. grid** (this module and its related classes are used for displaying and editing tabular data.), **wx.calendar** (a calendar control class), **pandas** (a fast, powerful, flexible and easy to use open source data analysis and manipulation tool).

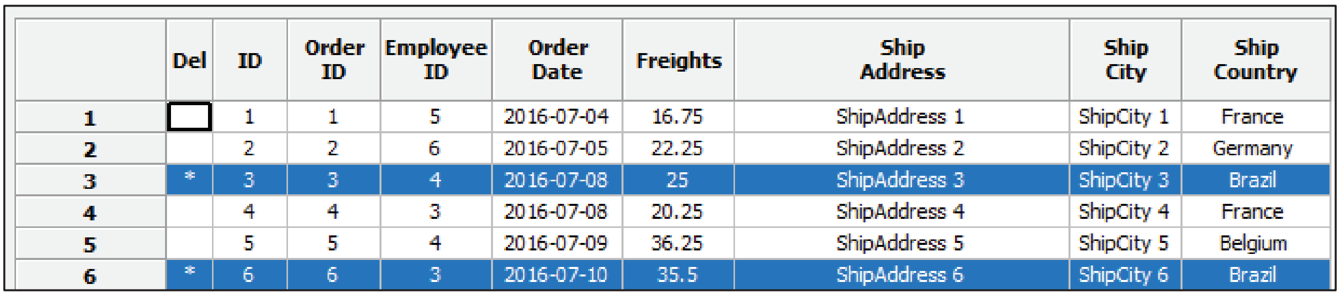
The **ExtragImageMemory** function (lines 11÷15) selecteaza din **SQLite database → Objects** an icon si create & returneaza the icon **img** from data in memory. This function is called from class **Class_DB → _TOOLBAR** function.

The **Class_DB** class create the interface (toolbar, grid, …), create the database main functionalities and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame), the **StatusBar** (placed at the frame bottom, were messages can be displayed); initializes the public variables of the class (lines 29÷33); call the **_TOOLBAR**, **CreateGrid** and **OnSelect** functions; create the sizer (a **wx.Sizer** control that can lay out items in a virtual grid); add **self.grid** in sizer created by **CreateGrid** function (line 40) and center the frame in the display.
- **_TOOLBAR** Create the **toolbar** (line 46); set the **toolbar** icons size (line 47) and background color (line 48); create the connection and cursor using **Config.db** database (line 49); generate **Lst_label**, **Lst_icons** and **Lst_Help** lists (lines 50÷53), which store the **tooolbar** icons labels, the icon names that must be extracted from memory by calling **ExtragImageMemory** function and the associated icons help; for every operation (add record, delete, modify, Excel export and exit) line 54 associate an unique identificator ID; through this ID the **Delete** and **Modify** icons can be activated or deactivated, because these operations can be executed only when in **self.grid** control are selected one or more lines, through **OnGridRangeSelect** function; through **for** cycle from line 55 the **img** icon is extracted into memory. Line 56 create the bitmap of **img** memory icon and scale at the imposed **toolbar**

icons size (line 57); in this cycle, for every operation, the icon is added in **toolbar** (lines 59, 62, 65, 68 & 71) and connect with **Bind** statement to associated functions (lines 60, 63, 66, 69 & 72); line 73 close the connection with **Config.db** database (line73); lines 74 deactivate **Delete** and **Modify** icons, because no line in **self.grid** control is selected; line 76 generate and show the **toolbar**.

- **OnToolClick** It recognizes a mouse click on one of the **toolbar** icons (line 79) and associates specific functions (lines 80÷84).
- **CreateGrid** This function create **self.grid** control (lines 87÷88) where database records are displayed; connect by **Bind** statement (line 89) with **OnGridRangeSelect** function based on **wx.grid.EVT_GRID_RANGE_SELECT** event; this event notify about a range of cells being selected when the user uses the mouse for selection; to operate directly on grid is not permitted (line 90); lines 91÷112 set the grid parameters: default centre cell alignment (line 91), columns label size (line 92), label value for grid columns (lines 94÷102), columns size (lines 101÷112).
- **OnGridRangeSelect** This function is called from **CreateGrid** function (line 89), when the user select one or more grid lines like in **Figure 15.5**; the selection can be made with left mouse click on first grid column to the respective line; to select more lines **CTRL** keyboard taste must be pressed; the selection lines are marked with blue color and a ***** simbol is placed in **Del** grid column; multiple continuous lines can be selected with pressed **Shift** keyboard taste; to deselect selected line/lines a click in the unselected white grid area must be performed. If a selection is detected by line 116, the index of selected line is appended in the **currentSelection** public list (line 119), put the ***** simbol in **Del** grid column (line 121) and refreshing the data in the grid (line 122). If a click is detected in the unselected white grid area (**else** branch from line 123) the index is removed from **currentSelection** list (line 126), put a space character in **Del** grid column (line 128) and refreshing the data in the grid (line 129). Lines 130÷131 deactivate **Delete** and **Modify** toolbar icons. Lines 132÷134 count in **csel** variable the number of indexes stored in **currentSelection** list; if multiple selection are detected (line 135) the **Delete** toolbar icons is activated (line136); if, only one selection & selection process was made, are detected by line 137, the index of selected line are stored in **LinGridSelect** variable; also the value from the grid cell identified by **LinGridSelect** line & **self.COL_rowid** column is stored in **self.CellGridSelect** variable (line 140); if the **self.CellGridSelect** variable is not empty (line 141) then **Modify** toolbar icon is activated (line 142); line 143 ensures more processing on the script.



	Del	ID	Order ID	Employee ID	Order Date	Freights	Ship Address	Ship City	Ship Country
1		1	1	5	2016-07-04	16.75	ShipAddress 1	ShipCity 1	France
2		2	2	6	2016-07-05	22.25	ShipAddress 2	ShipCity 2	Germany
3	*	3	3	4	2016-07-08	25	ShipAddress 3	ShipCity 3	Brazil
4		4	4	3	2016-07-08	20.25	ShipAddress 4	ShipCity 4	France
5		5	5	4	2016-07-09	36.25	ShipAddress 5	ShipCity 5	Belgium
6	*	6	6	3	2016-07-10	35.5	ShipAddress 6	ShipCity 6	Brazil

Figure 15.5 The selection of more lines in grid control

- **OnDelete** This function deletes selected lines from the grid with confirmation. Lines 146÷148 count the number of selected grid lines and ask for delete confirmation (lines149÷151); if the answer is **YES**, the connection with **self.DATABASE="DB_orders.db"** is made and for all selections (line 155) the function go through the lines of the table marked with ***** symbol and delete them (lines 156-161); lines 163÷164 initializes the selection variables and line 165 **ForceRefresh** of the grid; the **Delete** icon from the **toolbar** is deactivated (line

166); finallyh, the **OnSelect** function is called (line 167) to reload the grid with database records, where the delete lines are excluded.

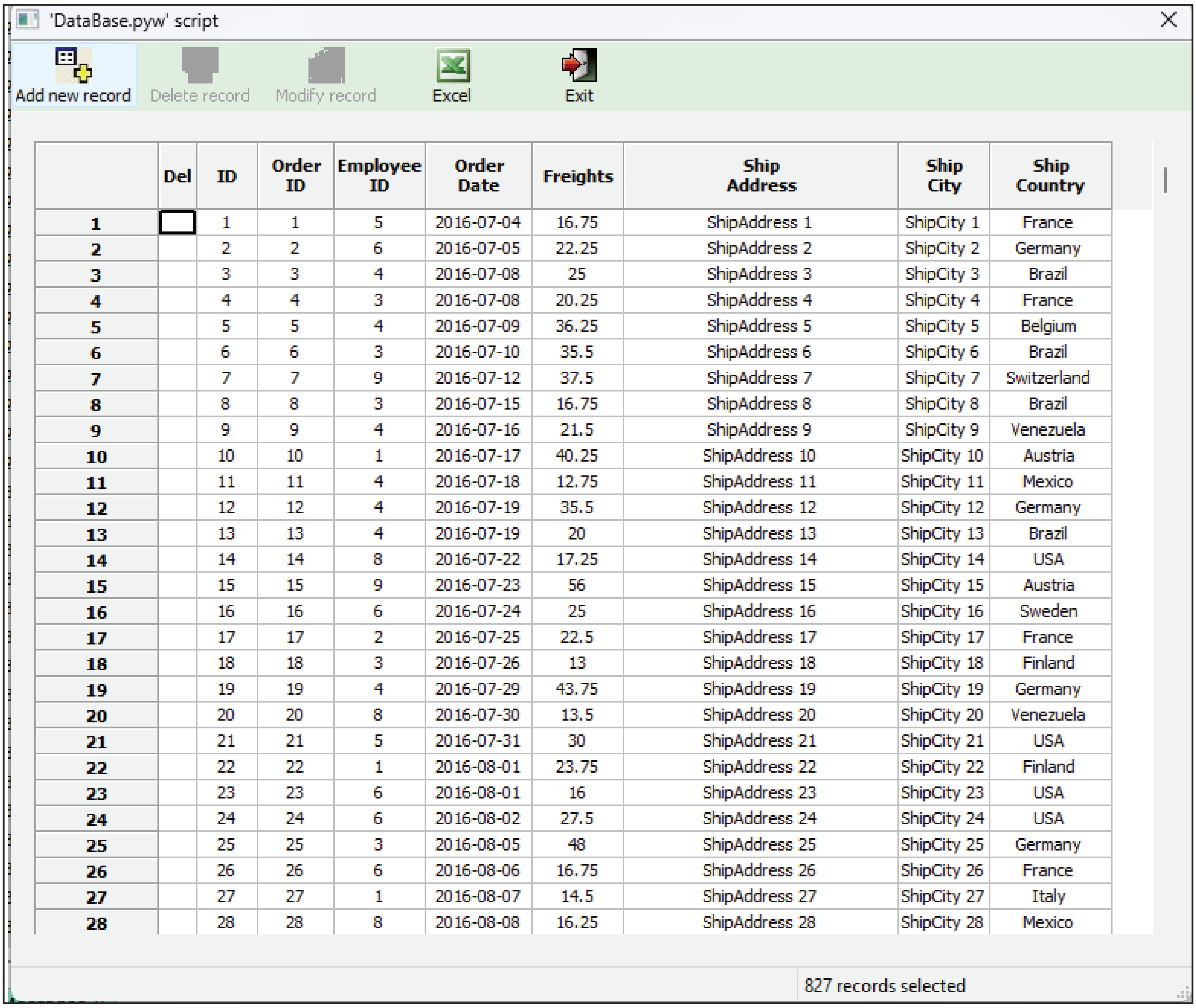
- **OnSelect** This function is called from **Class_DB class** functions at **init** event (lines 137), after **Delete** operation (line 167) and also from **AddModifyRecord** class function after add or modify record (line 325). If the grid is not empty (line 170) line 171 delete all grid lines; the connection with **self.DATABASE="DB_orders.db"** is made (lines 172÷173) and all records are loaded into **cursor** variable, which is used for fetching data from table in row-by-row manner; in **for** cycle (line 178) every record is extracted from **cursor** variable and put in grid cells (lines 179÷188); also the number of records are stored in **RECCOUNT** variable (lines 191÷192) and displayed in status bar (lines 194÷195).
- **OnAdd** Lines 198÷199 deactivate **Delete** and **Modify** toolbar icons. Call **AddModifyRecord** class with **Add Record** option (line 200) and make modal the class frame (line 201) to create a nested event loop so **MakeModal** will not return until the class is finished and input to the other windows in the application will be blocked. Also, the database & table names and the empty string “” are sent to the **AddModifyRecord** class; the empty string “” is transmitted as parameter, because it is not necessary to select a line in the grid to add a record, as it is the case for modify a record.
- **OnModify** Lines 204÷205 deactivate **Delete** and **Modify** toolbar icons. Call **AddModifyRecord** class with **Modify Record** option (line 206) and make modal the class frame (line 207) to create a nested event loop so **MakeModal** will not return until the class is finished and input to the other windows in the application will be blocked. Also, the database & table names and the variable **self.CellGridSelect** are sent to the **AddModifyRecord** class; the variable **self.CellGridSelect** store the value of the cell selected to modify and is created in **OnGridRangeSelect** function.
- **OnExcel** This function export the content of database **B_orders.db** table **Orders** to an Excel file using **pandas** library.
- **On_Exit** This function exits the application.

The **AddModifyRecord** class implement operations to add or modify a record and include the following functions:

- **__init__** • Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame), the **Sizer** and create the **font** variable. The function receive from **Class_DB → OnAdd** and **Class_DB → OnModify** the following parameters: **title**, **DATABASE**, **TABLE**, **Parent_Frame** and **CellGridSelect**. The **title** variable receive two values: **Add Record** transmitted by **Class_DB → OnAdd** function and Modify Record transmitted by **Class_DB → OnModify** function. Lines 229÷232 create the public variables in **AddModifyRecord** class. The **self.Parent_Frame** store a reference to **Class_DB** from where the class **AddModifyRecord** was called. The **self.CellGridSelect** variable store the cell content of the single line selected from the grid to be modified; in the case of adding a new record the value of this parameter is empty, because it is not necessary to select a line in the grid to add a record, as it is the case for modify a record. Lines 233÷236 define the labels of the controls placed on the **panel**. Lines 238÷252 define define the text controls where the record values will be put or modified. Line 243 define a **CalendarCtrl** control to select the date and line 244 connect by **Bind** statement with **OnCalSelChanged** function. Lines 254÷260 add text controls to the **sizer**. Lines 262÷271 create **Save** and **Cancel** buttons and bind these controls to **OnSave** and **OnCloseMe** functions. If the variable **title** has the **Modify Record** value (line 273) then coonection with **DB_orders.db → Orders** table to extract the fileds values from the record identified by **(_ROWID_= " + str(CellGridSelect)** condition (line 276); these values were placed in text controls (lines 278÷285) created before. Lines 289÷290 center and show the frame in the display.

- **OnCalSelChanged** • This function is called by **CalendarCtrl** to modify the selected data in yyyy-mm-dd format.
- **OnSave** Create the connection to **DB_orders.db → Orders** table and define **Fields** list. If the **title** is **Add Record** lines 303÷310 define a complex string that store the text controls values and line 311 create the **SELECT** command; if the **title** is **Modify Record** lines 313÷321 create direct the **SELECT** command. Line 322 execute one of those **SELECT** command and close the connection. Through the **self.Parent_Frame** variable is called **OnSelect** functions from **Class_DB** class to reload the grid with database records.
- **OnCloseMe** Call the **CLOSE** function
- **CLOSE** Cancel **Make Modal** status of **AddModifyRecord** class and close the window.

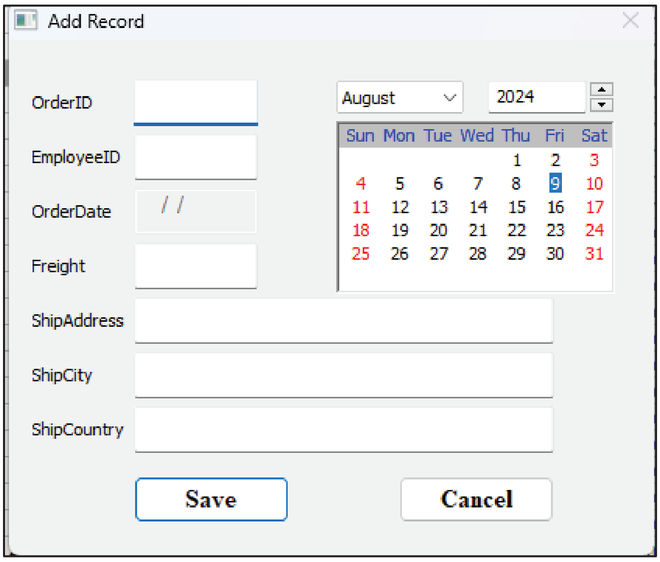
Figure 15.6 show the main interface of **DataBase.pyw** script. **Figures 15.7** & **15.8** show the interface of **Add Record** & **Modify Record** functions.



	Del	ID	Order ID	Employee ID	Order Date	Freights	Ship Address	Ship City	Ship Country
1	□	1	1	5	2016-07-04	16.75	ShipAddress 1	ShipCity 1	France
2		2	2	6	2016-07-05	22.25	ShipAddress 2	ShipCity 2	Germany
3		3	3	4	2016-07-08	25	ShipAddress 3	ShipCity 3	Brazil
4		4	4	3	2016-07-08	20.25	ShipAddress 4	ShipCity 4	France
5		5	5	4	2016-07-09	36.25	ShipAddress 5	ShipCity 5	Belgium
6		6	6	3	2016-07-10	35.5	ShipAddress 6	ShipCity 6	Brazil
7		7	7	9	2016-07-12	37.5	ShipAddress 7	ShipCity 7	Switzerland
8		8	8	3	2016-07-15	16.75	ShipAddress 8	ShipCity 8	Brazil
9		9	9	4	2016-07-16	21.5	ShipAddress 9	ShipCity 9	Venezuela
10		10	10	1	2016-07-17	40.25	ShipAddress 10	ShipCity 10	Austria
11		11	11	4	2016-07-18	12.75	ShipAddress 11	ShipCity 11	Mexico
12		12	12	4	2016-07-19	35.5	ShipAddress 12	ShipCity 12	Germany
13		13	13	4	2016-07-19	20	ShipAddress 13	ShipCity 13	Brazil
14		14	14	8	2016-07-22	17.25	ShipAddress 14	ShipCity 14	USA
15		15	15	9	2016-07-23	56	ShipAddress 15	ShipCity 15	Austria
16		16	16	6	2016-07-24	25	ShipAddress 16	ShipCity 16	Sweden
17		17	17	2	2016-07-25	22.5	ShipAddress 17	ShipCity 17	France
18		18	18	3	2016-07-26	13	ShipAddress 18	ShipCity 18	Finland
19		19	19	4	2016-07-29	43.75	ShipAddress 19	ShipCity 19	Germany
20		20	20	8	2016-07-30	13.5	ShipAddress 20	ShipCity 20	Venezuela
21		21	21	5	2016-07-31	30	ShipAddress 21	ShipCity 21	USA
22		22	22	1	2016-08-01	23.75	ShipAddress 22	ShipCity 22	Finland
23		23	23	6	2016-08-01	16	ShipAddress 23	ShipCity 23	USA
24		24	24	6	2016-08-02	27.5	ShipAddress 24	ShipCity 24	USA
25		25	25	3	2016-08-05	48	ShipAddress 25	ShipCity 25	Germany
26		26	26	6	2016-08-06	16.75	ShipAddress 26	ShipCity 26	France
27		27	27	1	2016-08-07	14.5	ShipAddress 27	ShipCity 27	Italy
28		28	28	8	2016-08-08	16.25	ShipAddress 28	ShipCity 28	Mexico

827 records selected

Figure 15.6 The main interface of **DataBase.pyw** script



Add Record

OrderID

EmployeeID

OrderDate / /

Freight

ShipAddress

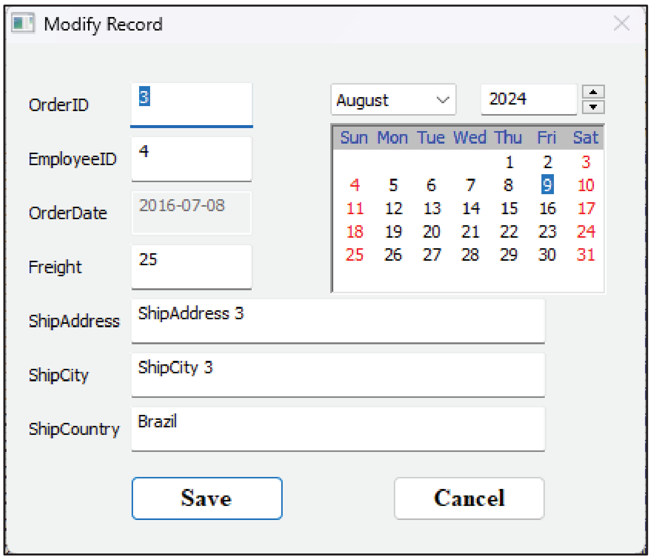
ShipCity

ShipCountry

August 2024

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Figure 15.7 The interface of **Add Record** function



Modify Record

OrderID 3

EmployeeID 4

OrderDate 2016-07-08

Freight 25

ShipAddress ShipAddress 3

ShipCity ShipCity 3

ShipCountry Brazil

August 2024

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Figure 15.8 The interface of **Modify Record** function

Listing 15.2 DataBase.pyw – SQLite Database – Python & wxPython version

```
from __future__ import division

import os
from pysqlite2 import dbapi2 as sqlite
import StringIO
import wx
import wx.grid
import  wx.calendar
import pandas as pd

def ExtragImageMemory(cursor, NumeImage):
    cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
    blob=cursor.fetchone()[0]
    img = wx.ImageFromStream(StringIO.StringIO(blob))
    return img

class Class_DB(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "'DataBase.pyw' script", size=(780, 660),
            style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
        self.panel = wx.Panel(self, -1)

        self.statusbar = self.CreateStatusBar()
        self.statusbar.SetFieldsCount(2)
        self.statusbar.SetStatusWidths([-4, -2])
        self.statusbar.SetStatusText("",0)
        self.statusbar.SetStatusText("",1)

        self.DATABASE="DB_orders.db"
        self.TABLE="Orders"
        self.currentSelection=[]
        self.COL_del=0    # The grid column with marking for delete record
        self.COL_rowid=1 # The grid column where the ROWID from the database is placed

        self._TOOLBAR()
        self.CreateGrid()
        self.OnSelect()

        sizer= wx.GridBagSizer(hgap=5, vgap=0)
        sizer.Add(self.grid,pos=(1,1),span=(1,8))
        self.panel.SetSizerAndFit(sizer)
        self.CenterOnScreen()
        self.Show()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  |wx.TB_FLAT | wx.TB_3DBUTTONS | wx.TB_TEXT) )
```

Listing 15.2 DataBase.pyw – SQLite Database – Python & wxPython version

```
        ToolBitmapSize = 24
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))
        connection = sqlite.connect(os.getcwd()+'\Config.db') ; cursor = connection.cursor()
        Lst_label=[ "Add new record", "Delete record", "Modify record", "Excel", "Exit"]
        Lst_icons=[ "Add.jpg", "Delete.jpg", "Modify.jpg","Excel.png",'Exit.png']
        Lst_Help=["Add new record in database","Delete selected record",
                "Modify selected record","Export table in Excel",  "Exit"]
        self.Add=100 ; self.Delete=200 ; self.Modify=300 ; self.Excel=400 ; self.Exit=500
        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            img = ExtragImageMemory(cursor, icon)
                sel_bmp=wx.BitmapFromImage(img.Scale(ToolBitmapSize,ToolBitmapSize))
                if label=="Add new record":
                    self.toolbar.AddLabelTool(self.Add, label, sel_bmp, shortHelp=HELP)
                    self.Bind(wx.EVT_TOOL, self.OnToolClick, id=self.Add)
                if label=="Delete record":
                    self.toolbar.AddLabelTool(self.Delete, label, sel_bmp, shortHelp=HELP)
                    self.Bind(wx.EVT_TOOL, self.OnToolClick, id=self.Delete)
                if label=="Modify record":
                    self.toolbar.AddLabelTool(self.Modify, label, sel_bmp, shortHelp=HELP)
                    self.Bind(wx.EVT_TOOL, self.OnToolClick, id=self.Modify)
                if label=="Excel":
                    self.toolbar.AddLabelTool(self.Excel, label, sel_bmp, shortHelp=HELP)
                    self.Bind(wx.EVT_TOOL, self.OnToolClick, id=self.Excel)
                if label=="Exit":
                    self.toolbar.AddLabelTool(self.Exit, label, sel_bmp, shortHelp=HELP)
                    self.Bind(wx.EVT_TOOL, self.OnToolClick, id=self.Exit)
            cursor.close() ; connection.close()
            self.toolbar.EnableTool(self.Delete, False)  # Delete
            self.toolbar.EnableTool(self.Modify, False)  # Modify
            self.toolbar.Realize()  ;   self.toolbar.Show()

    def OnToolClick(self, event):
            tb = event.GetEventObject()
            if event.GetId()==self.Add:      self.OnAdd()
            if event.GetId()==self.Delete:   self.OnDelete()
            if event.GetId()==self.Modify:  self.OnModify()
            if event.GetId()==self.Excel:     self.OnExcel()
            if event.GetId()==self.Exit:       self.On_Exit()

    def CreateGrid(self):
        self.grid = wx.grid.Grid(self.panel, size=(750, 520))
        self.grid.CreateGrid( 100, 9)
        self.Bind(wx.grid.EVT_GRID_RANGE_SELECT,self.OnGridRangeSelect)
        self.grid.EnableEditing(0)
        self.grid.SetDefaultCellAlignment(wx.ALIGN_CENTRE,wx.ALIGN_CENTRE)
        self.grid.SetColLabelSize(45)

        self.grid.SetColLabelValue(self.COL_del,'Del')
        self.grid.SetColLabelValue(self.COL_rowid,'ID')
        self.grid.SetColLabelValue(2,'Order\nID')
        self.grid.SetColLabelValue(3,'Employee\nID')
        self.grid.SetColLabelValue(4,'Order\nDate')
        self.grid.SetColLabelValue(5,'Freights')
        self.grid.SetColLabelValue(6,'Ship\nAddress')
        self.grid.SetColLabelValue(7,'Ship\nCity')
        self.grid.SetColLabelValue(8,'Ship\nCountry')

        self.grid.SetColSize(self.COL_del,25)
        self.grid.SetColSize(self.COL_rowid,40)
        self.grid.SetColSize(2,50)
        self.grid.SetColSize(3,60)
        self.grid.SetColSize(4,70)
        self.grid.SetColSize(5,60)
        self.grid.SetColSize(6,180)
        self.grid.SetColSize(7,80)
        self.grid.SetColSize(7,60)

    def OnGridRangeSelect( self, event ):
        # Ncol=self.grid.GetNumberCols()
        if event.Selecting():
            for index in range( event.GetTopRow(), event.GetBottomRow()+1):
                if index not in self.currentSelection:
```

Listing 15.2 DataBase.pyw – SQLite Database – Python & wxPython version

```
                    self.currentSelection.append( index )
                    # Mark selected line in 'self.COL_del' column with '*' key
                    self.grid.SetCellValue(index,self.COL_del,'*')
                    self.grid.ForceRefresh()
        else:
                for index in range( event.GetTopRow(), event.GetBottomRow()+1):
                    while index in self.currentSelection:
                        self.currentSelection.remove( index )
                        # Deselect selected line by deleting the '*' key in the 'self.COL_del'
column
                        self.grid.SetCellValue(index,self.COL_del,'')
                        self.grid.ForceRefresh()
        self.toolbar.EnableTool(self.Delete, False)
        self.toolbar.EnableTool(self.Modify, False)
        csel=0
        for x in self.currentSelection:
            csel=csel+1
        if ((csel > 0)):
            self.toolbar.EnableTool(self.Delete, True)
        if csel == 1 and event.Selecting():
            coordonate= event.GetTopLeftCoords(), event.GetBottomRightCoords()
            LinGridSelect= coordonate[0][0]
            self.CellGridSelect=self.grid.GetCellValue(LinGridSelect,self.COL_rowid )
            if self.CellGridSelect.strip() !='':
                self.toolbar.EnableTool(self.Modify, True)
        event.Skip()

    def OnDelete(self):
        csel=0
        for x in self.currentSelection:
            csel=csel+1
        d = wx.MessageDialog(self, 'Confirm deletion of '+str(csel)+' selected records ?', \
            caption = "Confirm", style = wx.YES_NO | wx.ICON_QUESTION)
        response = d.ShowModal()
        if response == wx.ID_YES:
            connection = sqlite.connect(os.getcwd()+'\\'+self.DATABASE)
            cursor = connection.cursor()
            for x in range(csel):    # Deleting marked records from the table
                for lin in range(self.grid.GetNumberRows()):
                    if self.grid.GetCellValue(lin,self.COL_del)=='*':
                      del_ID=str(self.grid.GetCellValue(lin,self.COL_rowid))
                      cursor.execute( "DELETE from "+self.TABLE+" WHERE _ROWID_ ='"+del_ID+"'")
                      connection.commit()
                      break
            cursor.close() ; connection.close()
            self.grid.ClearSelection()
            self.currentSelection=[]
            self.grid.ForceRefresh()
            self.toolbar.EnableTool(self.Delete, False)
            self.OnSelect()

    def OnSelect(self):
        if self.grid.GetNumberRows()>0:
            self.grid.DeleteRows(0, self.grid.GetNumberRows())
        connection = sqlite.connect(os.getcwd()+"\\"+self.DATABASE)
        cursor = connection.cursor()
        Fields ="_ROWID_,OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry"
        SELECT="select  "+Fields+" from "+self.TABLE
        cursor.execute(SELECT)
        lin=0
        for (rowid,OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry) in cursor.fetchall():
            self.grid.AppendRows(1)
            self.grid.SetCellValue(lin,self.COL_del,'') # Del
            self.grid.SetCellValue(lin,self.COL_rowid,str(rowid)) # ROWID
            self.grid.SetCellValue(lin,2,str(OrderID))
            self.grid.SetCellValue(lin,3,str(EmployeeID) )
            self.grid.SetCellValue(lin,4,str(OrderDate))
            self.grid.SetCellValue(lin,5,str(Freight))
            self.grid.SetCellValue(lin,6,ShipAddress)
            self.grid.SetCellValue(lin,7,ShipCity)
            self.grid.SetCellValue(lin,8,ShipCountry)
            lin=lin+1
```

Listing 15.2 DataBase.pyw – SQLite Database – Python & wxPython version

```
        self.grid.MakeCellVisible(0,0)
        cursor.execute( "SELECT count(*) FROM "+self.TABLE)
        RECCOUNT=cursor.fetchone()[0]
        cursor.close()  ;  connection.close()
        self.statusbar.SetStatusText("Database: " +self.DATABASE+"  Table:   "+self.TABLE,0)
        self.statusbar.SetStatusText(str(RECCOUNT)+' records selected',1)

    def OnAdd(self):
        self.toolbar.EnableTool(self.Delete, False)
        self.toolbar.EnableTool(self.Modify, False)
        AMR=AddModifyRecord(None, 'Add Record', self, self.DATABASE, self.TABLE, "")
        AMR.MakeModal(True)

    def OnModify(self):
        self.toolbar.EnableTool(self.Delete, False)
        self.toolbar.EnableTool(self.Modify, False)
        MMR=AddModifyRecord(None, 'Modify Record', self, self.DATABASE, self.TABLE, self.CellGridSelect)
        MMR.MakeModal(True)

    def OnExcel(self):
        connection = sqlite.connect(os.getcwd()+"\\"+self.DATABASE)
        SELECT="SELECT * FROM "+self.TABLE
        df = pd.read_sql(SELECT,connection)
        file_name=os.getcwd()+'/'+self.DATABASE+"__"+self.TABLE+".csv"
        df.to_csv(file_name)
        wx.MessageBox( "File saved to:\n\n"+file_name)

    def On_Exit(self):
        self.Destroy()

# =================================
class AddModifyRecord(wx.Frame):
    def __init__(self, parent, title, Parent_Frame, DATABASE, TABLE, CellGridSelect):
        wx.Frame.__init__(self, parent, title=title, size=(420,360),
            style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
            wx.MAXIMIZE_BOX | wx.CLOSE_BOX )) # Initialize the frame
            panel = wx.Panel(self)  # The panel will hold the contents of the frame
            sizer= wx.GridBagSizer(hgap=5, vgap=5)
            font = wx.Font(12, wx.ROMAN, wx.NORMAL, wx.BOLD)
            self.DATABASE=DATABASE
            self.TABLE=TABLE
            self.Parent_Frame=Parent_Frame
            self.CellGridSelect=CellGridSelect
            Lst_Labels=['OrderID', 'EmployeeID', 'OrderDate', 'Freight','ShipAddress','ShipCity','ShipCountry']
            for i, lbl in enumerate(Lst_Labels):
                self.st=wx.StaticText(panel, -1, lbl)
                sizer.Add(self.st,pos=(i+1,1),flag=wx.ALIGN_CENTER_VERTICAL | wx.ALIGN_LEFT)

            self.txt_OrderID = wx.TextCtrl(panel, -1, "", size=(80, 30))
            self.txt_OrderID .SetToolTipString("Input an integer number")
            self.txt_EmployeeID = wx.TextCtrl(panel, -1, "", size=(80, 30))
            self.txt_EmployeeID .SetToolTipString("Input an integer number")
            self.txt_OrderDate = wx.TextCtrl(panel, -1, "    /  /  ", size=(80, 30))
            calendar = wx.calendar.CalendarCtrl(panel, -1, wx.DateTime_Now(), pos=(210,26))
            self.Bind(wx.calendar.EVT_CALENDAR_SEL_CHANGED, self.OnCalSelChanged, calendar)
            self.txt_Freight = wx.TextCtrl(panel, -1, "", size=(80, 30))
            self.txt_Freight .SetToolTipString("Input an float number")
            self.txt_ShipAddress = wx.TextCtrl(panel, -1, "", size=(270, 30))
            self.txt_ShipAddress .SetToolTipString("Input text")
            self.txt_ShipCity = wx.TextCtrl(panel, -1, "", size=(270, 30))
            self.txt_ShipCity .SetToolTipString("Input text")
            self.txt_ShipCountry = wx.TextCtrl(panel, -1, "", size=(270, 30))
            self.txt_ShipCountry .SetToolTipString("Input text")

            sizer.Add(self.txt_OrderID,pos=(1,2))
            sizer.Add(self.txt_EmployeeID,pos=(2,2))
            sizer.Add(self.txt_OrderDate,pos=(3,2)) ; self.txt_OrderDate.Enabled=False
            sizer.Add(self.txt_Freight,pos=(4,2))
            sizer.Add(self.txt_ShipAddress,pos=(5,2))
            sizer.Add(self.txt_ShipCity,pos=(6,2))
            sizer.Add(self.txt_ShipCountry,pos=(7,2))

```

Listing 15.2 DataBase.pyw – SQLite Database – Python & wxPython version

```
        btn_Save= wx.Button(panel, -1, "Save", size=(100, 30), pos=(80,280))
        self.Bind(wx.EVT_BUTTON, self.OnSave, btn_Save)
        btn_Save.SetToolTipString("Save record to database")
        btn_Save.SetFont(wx.Font(12, wx.ROMAN, wx.NORMAL, wx.BOLD))
        btn_Save.SetDefault()

        btn_Exit = wx.Button(panel, -1, "Cancel", size=(100, 30), pos=(250,280))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe,btn_Exit)
        btn_Exit.SetToolTipString("Close window without saving changes")
        btn_Exit.SetFont(wx.Font(12, wx.ROMAN, wx.NORMAL, wx.BOLD))

        if self.Title=='Modify Record':
            connection = sqlite.connect(os.getcwd()+'\\'+self.DATABASE)
            cursor = connection.cursor()
            SELECT ="select * from "+self.TABLE+" where (_ROWID_= " + str(CellGridSelect) +")"
            cursor.execute(SELECT)  ;  CRS=cursor.fetchall()
            for (OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry) in CRS:
                self.txt_OrderID.SetValue(str(OrderID))
                self.txt_EmployeeID.SetValue(str(EmployeeID))
                self.txt_OrderDate.SetValue(str(OrderDate).replace("/", "-", 2))
                self.txt_Freight.SetValue(str(Freight))
                self.txt_ShipAddress.SetValue(str(ShipAddress))
                self.txt_ShipCity.SetValue(str(ShipCity))
                self.txt_ShipCountry.SetValue(str(ShipCountry))
                cursor.close()  ;  connection.close()

        panel.SetSizerAndFit(sizer)
        self.CenterOnScreen()
        self.Show()

    def OnCalSelChanged(self, event):
        cal = event.GetEventObject()
        str_Data=str(cal.GetDate())[:8]
        month, day, year = map(str, str_Data.split('/'))
        self.txt_OrderDate.SetValue( "20"+year+"-"+month+"-"+day )

    def OnSave(self, event):
        connection = sqlite.connect(os.getcwd()+"\\"+self.DATABASE)
        cursor = connection.cursor()
        Fields ="OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry"
        if self.Title=='Add Record':
            str_Fields=''
            str_Fields=str_Fields+"'"+self.txt_OrderID.GetLineText(0).strip()+"',"
            str_Fields=str_Fields+"'"+self.txt_EmployeeID.GetLineText(0).strip()+"',"
            str_Fields=str_Fields+"'"+self.txt_OrderDate.GetLineText(0).strip()+"',"
            str_Fields=str_Fields+"'"+self.txt_Freight.GetLineText(0).strip()+"',"
            str_Fields=str_Fields+"'"+self.txt_ShipAddress.GetLineText(0).strip()+"',"
            str_Fields=str_Fields+"'"+self.txt_ShipCity.GetLineText(0).strip()+"',"
            str_Fields=str_Fields+"'"+self.txt_ShipCountry.GetLineText(0).strip()+"'"
            SELECT="INSERT INTO "+self.TABLE+"  ("+Fields+") VALUES ( "+str_Fields+")"
        if self.Title=='Modify Record':
            SELECT="UPDATE "+self.TABLE+" SET "+ \
                "OrderID="+"'"+self.txt_OrderID.GetLineText(0)+"',"+ \
                 "EmployeeID="+"'"+self.txt_EmployeeID.GetLineText(0)+"',"+ \
                "OrderDate="+"'"+self.txt_OrderDate.GetLineText(0)+"',"+ \
                "Freight="+"'"+self.txt_Freight.GetLineText(0)+"',"+ \
                "ShipAddress="+"'"+self.txt_ShipAddress.GetLineText(0)+"',"+ \
                "ShipCity="+"'"+self.txt_ShipCity.GetLineText(0)+"',"+ \
                "ShipCountry="+"'"+self.txt_ShipCountry.GetLineText(0)+"'"+ \
                 " where (_ROWID_= " + str(self.CellGridSelect) +")"
        cursor.execute( SELECT)
        connection.commit()
        cursor.close()  ;  connection.close()
        self.Parent_Frame.OnSelect()
        self.CLOSE()

    def OnCloseMe(self, event):
        self.CLOSE()

    def CLOSE(self):
        self.MakeModal(False)
        self.Close(True)
```

Listing 15.2 DataBase.pyw – SQLite Database – Python & wxPython version

```
                        self.Destroy()

app = wx.App(redirect=False)
frm_DB=Class_DB()
app.MainLoop()
```

The **DataBase.py** script (Python & Streamlit version) contains 151 lines and begin with importing libraries: **sqlite3** (a C-language library that implements a small, fast, self-contained, high-reliability, full-featured), **streamlit** (create a great interactive web application with Python), **pathlib** (provide classes to represent abstract/concrete paths) and **pandas** (open source data analysis and manipulation tool).

Lines 6÷9 configures the default settings of the page (title, page icon and **sidebar** state). Lines 11÷21 define the parameters of the page **dark** mode (primary / background / text color and font). Line 22 impose the **dark** mode. Lines 28÷32 define the the script path and the public variables of the script.

The script contain the following 6 functions: **COUNT**, **Browse_Records**, **Add_Record**, **Modify_Record**, **Delete_Record**, **OnExcel**:

- **COUNT** This function count the number of records selected for an imposed condition and return this value through **RECCOUNT** variable.
- **Browse_Records** Display the table records through a dataframe, **Figure 15.9**. at the table top there are 3 icons: ⤓ 🔍 ⛶. First icon download the table content to a CSV file; the second launch a search in the table; last icon place the table on full screen.
- **Add_Record** Add a new record in the table.
- **Modify_Record** Modify a record selected through **OrderID** value.
- **Delete_Record** Delete a record from the table selected through **OrderID** value.
- **OnExcel** This function export the content of database table to an Excel file using **pandas** library.

Lines 121÷151 are part of the main script. Lines 121÷133 define the controls to input values of the associated table fields, **Figure 15.10**. **Figure 15.11** show the **sidebar** of **DataBase.py** script. Lines 136÷151 implements the functionalities associated with the **sidebar** buttons.

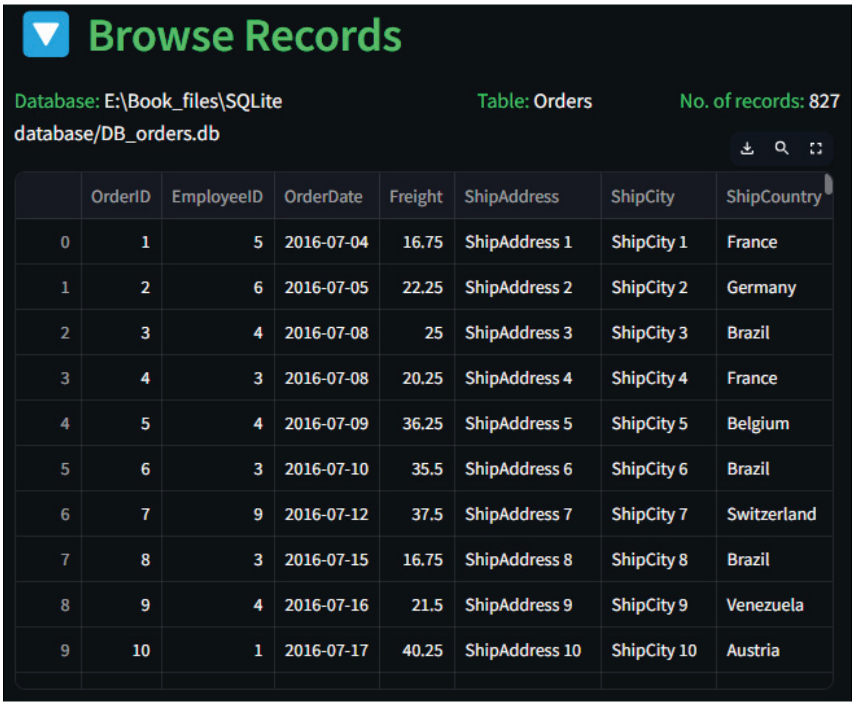
*"The streamlit library has a unique way to execute the scripts. As soon as a script is launched, a local **Streamlit** server will spin up and the application will open in a new tab in the default web browser. The Streamlit architecture allows writing applications in the same way as Python scripts. To unlock this, Streamlit apps have a unique data flow: any time when something must be updated on the screen, Streamlit reruns entire Python script from top to bottom. This can happen in two situations:*

- *whenever the app's source code is modified.*
- *whenever a user interacts with widgets in the app (exemple: clicking a button)."* [15.5]

For this reason, the controls for entering the values of the associated table fields are placed in the main section of the script. So, for **Add Record** operation, all fields must be filled with informations and later select **Add Record** button from **sidebar**.

For **Modify** or **Delete** operation, the **Order ID** field must be filled with one from existing in the table and later select the corresponding button from **sidebar**.

For the **Modify** option, after filling the **Order ID** field, the script should fill the associated controls with the values from the record identified by the **Order ID** key, which is not implemented in this script.

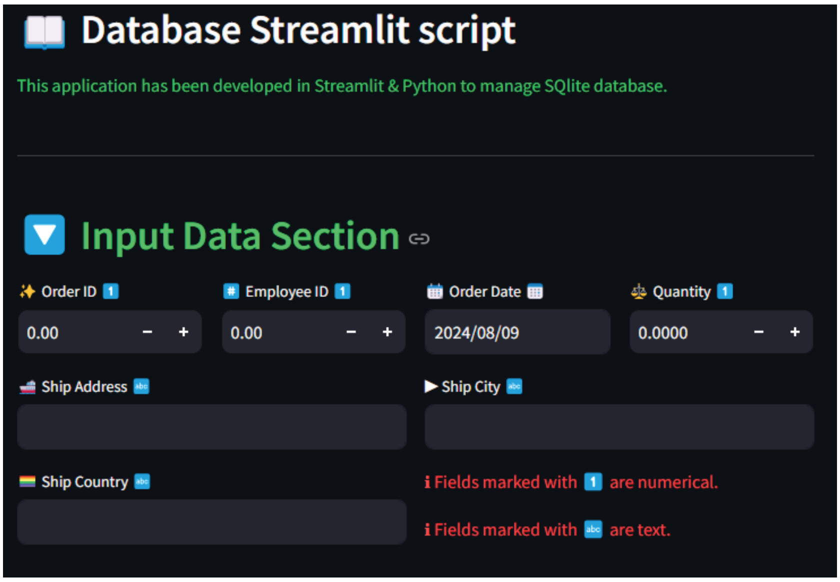


🔽 Browse Records

Database: E:\Book_files\SQLite database/DB_orders.db   Table: Orders   No. of records: 827

	OrderID	EmployeeID	OrderDate	Freight	ShipAddress	ShipCity	ShipCountry
0	1	5	2016-07-04	16.75	ShipAddress 1	ShipCity 1	France
1	2	6	2016-07-05	22.25	ShipAddress 2	ShipCity 2	Germany
2	3	4	2016-07-08	25	ShipAddress 3	ShipCity 3	Brazil
3	4	3	2016-07-08	20.25	ShipAddress 4	ShipCity 4	France
4	5	4	2016-07-09	36.25	ShipAddress 5	ShipCity 5	Belgium
5	6	3	2016-07-10	35.5	ShipAddress 6	ShipCity 6	Brazil
6	7	9	2016-07-12	37.5	ShipAddress 7	ShipCity 7	Switzerland
7	8	3	2016-07-15	16.75	ShipAddress 8	ShipCity 8	Brazil
8	9	4	2016-07-16	21.5	ShipAddress 9	ShipCity 9	Venezuela
9	10	1	2016-07-17	40.25	ShipAddress 10	ShipCity 10	Austria

Figure 15.9 The controls of **DataBase.py** script



📖 Database Streamlit script

This application has been developed in Streamlit & Python to manage SQlite database.

🔽 Input Data Section

✨ Order ID 1️⃣: 0.00

#️⃣ Employee ID 1️⃣: 0.00

📅 Order Date 📅: 2024/08/09

⚖ Quantity 1️⃣: 0.0000

🚢 Ship Address 🔤

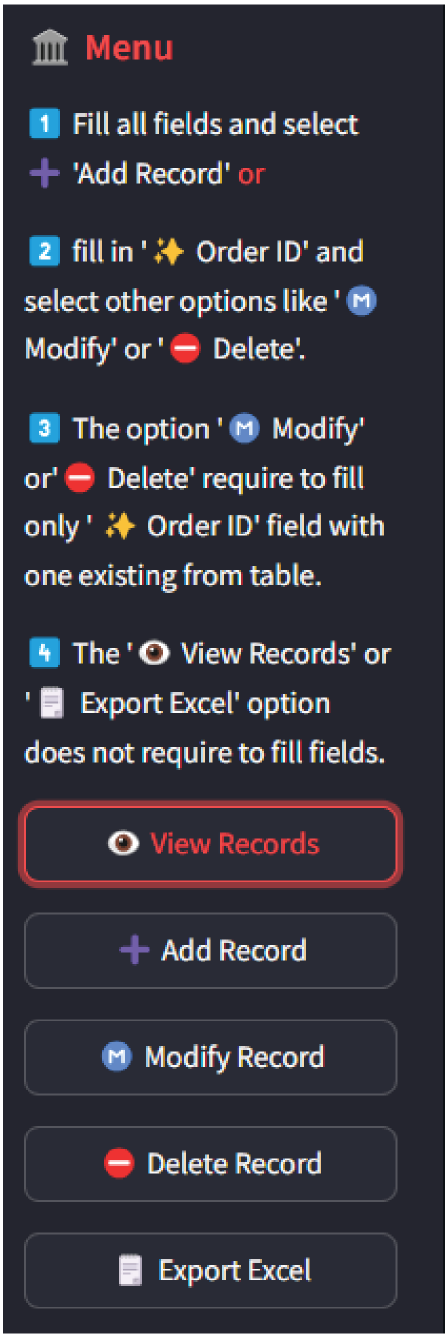
▶ Ship City 🔤

🏳️‍🌈 Ship Country 🔤

ℹ Fields marked with 1️⃣ are numerical.

ℹ Fields marked with 🔤 are text.

Figure 15.10 The controls of **DataBase.py** script



🏛 Menu

1️⃣ Fill all fields and select ➕ 'Add Record' or

2️⃣ fill in ' ✨ Order ID' and select other options like ' Ⓜ️ Modify' or ' ⛔ Delete'.

3️⃣ The option ' Ⓜ️ Modify' or' ⛔ Delete' require to fill only ' ✨ Order ID' field with one existing from table.

4️⃣ The ' 👁 View Records' or ' 📄 Export Excel' option does not require to fill fields.

Figure 15.11 The **sidebar**

Listing 15.3 DataBase.py – SQLite Database – Python & Streamlit version

```
import sqlite3
import streamlit as st
from pathlib import Path
import pandas as pd

st.set_page_config(
        page_title="Database Streamlit script",
        page_icon="□",  # layout="wide",
        initial_sidebar_state="expanded" )

dark = '''
        <style>
                .stApp {
                primaryColor="Red"
                backgroundColor="Black"
                secondaryBackgroundColor="Black"
```

Listing 15.3 DataBase.py – SQLite Database – Python & Streamlit version

```
                textColor="Gray"
                font="sans serif"
                }
        </style>
'''
st.markdown(dark, unsafe_allow_html=True)
st.header(":book: Database Streamlit script")
st.markdown (""":green[This application has been developed in
    Streamlit & Python to manage SQlite database.]""")
st.divider()

current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()
DATABASE="DB_orders.db"
TABLE="Orders"
DB_NAME=str(current_dir)+"/"+DATABASE
FIELDS ="OrderID,EmployeeID,OrderDate,Freight,ShipAddress,ShipCity,ShipCountry"

def COUNT(OrderID):
    connection = sqlite3.connect(DB_NAME)
    CURSOR = connection.cursor()
    CURSOR.execute("SELECT COUNT(*) FROM "+TABLE+" WHERE OrderID= " + str(int(OrderID)))
    RECCOUNT=CURSOR.fetchone()[0]
    return RECCOUNT

def Browse_Records():
    st.header(":green[:arrow_down_small: Browse Records]")
    connection = sqlite3.connect(DB_NAME)
    CURSOR = connection.cursor()
    CURSOR.execute( "SELECT count(*) FROM "+TABLE)
    RECCOUNT=CURSOR.fetchone()[0]
    CRS = CURSOR.execute('select * from '+TABLE)
    COLUMNS = [column[0] for column in CURSOR.description]
    col1, col2, col3= st.columns([0.7,0.3,0.4])
    col1.write(":green[Database: ] "+DB_NAME)
    col2.write(":green[Table: ] "+TABLE)
    col3.write(":green[No. of records: ] "+str(RECCOUNT))
    # col1, col2, col3= st.columns([0.9,0.3,0.4])
    # col1.write(":green[Table fields:]")  ;  col2.write(FIELDS)
    df = pd.DataFrame(CRS, columns=COLUMNS)
    st.dataframe(df)
    CURSOR.close()  ;  connection.close()
    st.divider()

def Add_Record(OrderID, EmployeeID, OrderDate, Freight, ShipAddress, ShipCity, ShipCountry):
    connection = sqlite3.connect(DB_NAME)
    CURSOR = connection.cursor()
    if COUNT(OrderID)>0:
        connection.close()
        MSJ="This is an error. Value of Primary Key 'OrderID="
        MSJ=MSJ+str(int(OrderID))+"' exist in table."
        st.error(MSJ, icon="□")
        return
    str_Fields=''
    str_Fields=str_Fields+"'"+str(int(OrderID))+"',"
    str_Fields=str_Fields+"'"+str(int(EmployeeID))+"',"
    str_Fields=str_Fields+"'"+str(OrderDate)+"',"
    str_Fields=str_Fields+"'"+str(Freight)+"',"
    str_Fields=str_Fields+"'"+ShipAddress+"',"
    str_Fields=str_Fields+"'"+ShipCity+"',"
    str_Fields=str_Fields+"'"+ShipCountry+"'"
    SELECT="INSERT INTO "+TABLE+" ("+FIELDS+") VALUES ( "+str_Fields+")"
    CURSOR.execute( SELECT)
    connection.commit() ; connection.close()
    st.success("Record added to database: '"+DATABASE+"'    Table: '"+TABLE+"'")

def Modify_Record(OrderID, EmployeeID, OrderDate, Freight, ShipAddress, ShipCity, ShipCountry):
    connection = sqlite3.connect(DB_NAME)
    CURSOR = connection.cursor()
    SELECT="UPDATE "+TABLE+" SET "+ \
        "EmployeeID="+"'"+str(int(EmployeeID))+"',"+ \
        "OrderDate="+"'"+str(OrderDate)+"',"+ \
        "Freight="+"'"+str(Freight)+"',"+ \
```

Listing 15.3 DataBase.py – SQLite Database – Python & Streamlit version

```
        "ShipAddress="+"'"+str(ShipAddress)+"',"+ \
        "ShipCity="+"'"+str(ShipCity)+"',"+ \
        "ShipCountry="+"'"+str(ShipCountry)+"'"+ \
        " where (OrderID= " + str(int(OrderID)) +")"
    CURSOR.execute(SELECT)
    connection.commit() ; connection.close()
    st.success("Record modified in database: '"+DATABASE+"'    Table: '"+TABLE+"'")

def Delete_Record(OrderID):
    connection = sqlite3.connect(DB_NAME)
    CURSOR = connection.cursor()
    if COUNT(OrderID)==0:
        connection.close()
        MSJ="This is an error. Value of Primary Key 'OrderID="
        MSJ=MSJ+str(int(OrderID))+"' not exist in table."
        st.error(MSJ, icon="□")
        return
    CURSOR.execute("DELETE FROM "+TABLE+" WHERE OrderID= " + str(int(OrderID)))
    connection.commit() ; connection.close()
    MSJ="The record with value of Primary Key 'OrderID='"
    MSJ=MSJ+str(int(OrderID))+" erased from table."
    st.success(MSJ, icon="□")

def OnExcel():
    connection = sqlite3.connect(DB_NAME)
    CURSOR = connection.cursor()
    SELECT="SELECT * FROM "+TABLE
    df = pd.read_sql(SELECT,connection)
    file_name=DB_NAME+"__"+TABLE+".csv"
    df.to_csv(file_name)
    st.success("File saved to:\n\n"+file_name)

st.header(":green[:arrow_down_small: Input Data Section]")
col1, col2, col3, col4 = st.columns(4)
OrderID = col1.number_input(":sparkles: Order ID :one:")
EmployeeID = col2.number_input(":hash: Employee ID :one:")
OrderDate = col3.date_input(":calendar: Order Date :date:")
Freight = col4.number_input(":scales: Quantity :one:", format="%2.4f")
col1, col2 = st.columns(2)
ShipAddress = col1.text_input(":ship: Ship Address :abc:")
ShipCity = col2.text_input(":arrow_forward: Ship City :abc:")
col1, col2 = st.columns(2)
ShipCountry = col1.text_input(":rainbow-flag: Ship Country :abc:")
col2.write(":red[:information_source: Fields marked with :one: are numerical.]")
col2.write(":red[:information_source: Fields marked with :abc: are text.]")
st.divider()

if st.sidebar.header(":red[:classical_building: Menu]"):
    st.sidebar.write(":one: Fill all fields and select :heavy_plus_sign: 'Add Record' :red[or]")
    st.sidebar.write(":two: fill in ':sparkles: Order ID' and select other options like ':m: Modify' or ':no_entry: Delete'.")
    st.sidebar.write(":three: The option ':m: Modify' or':no_entry: Delete' require to fill only ' \
                    :sparkles: Order ID' field with one existing from table.")
    st.sidebar.write(":four: The ':eye: View Records' or ':spiral_note_pad: Export Excel' option does not require to fill fields.")
if st.sidebar.button(":eye: View Records", use_container_width=True):
        Browse_Records()
if st.sidebar.button(":heavy_plus_sign: Add Record", use_container_width=True):
        Add_Record(OrderID, EmployeeID, OrderDate, Freight, ShipAddress, ShipCity, ShipCountry)
if st.sidebar.button(":m: Modify Record", use_container_width=True):
        Modify_Record(OrderID, EmployeeID, OrderDate, Freight, ShipAddress, ShipCity, ShipCountry)
if st.sidebar.button(":no_entry: Delete Record", use_container_width=True):
        Delete_Record(OrderID)
if st.sidebar.button(":spiral_note_pad: Export Excel", use_container_width=True):
        OnExcel()
```

16. MySQL Database

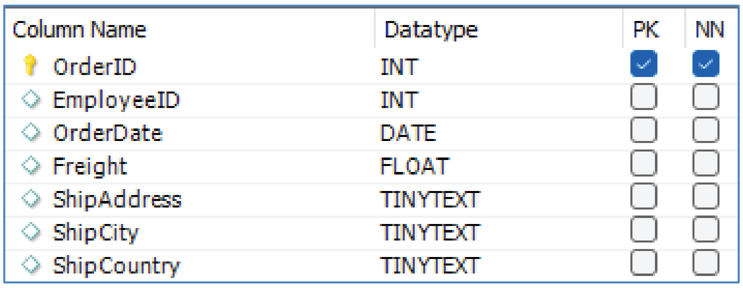
SQLite is a lightweight solution used for small projects with only one user, while MySQL works for major projects and supports multiple users. This application will focus on **MySQL** database type with standard operation (**Add**, **Delete**, **Modify** records), export table to **Word** & **Excel**. To operate directly with database (create databases, tables, browse records, SQL commands) will use **MySQL Workbench**, a visual database administration tool that provides data modelling, SQL development and administration tools for server configuration, user management, backup and more.

This application uses the same **DB_Orders.db** database as in the **& 15. SQLite Database** application, which we will copy into **MySQL database** folder of the current application. After installing **MySQL** and **MySQL Workbench** and running **Transfer.pyw** script (**Listing 16.1**) the database records in SQLite format will be transferred to the **MySQL db_orders** database. The operating procedure is described in the following steps:

❶ Install **MySQL** server & **MySQL Workbench (table 16.1**);

❷ Open **MySQL Workbench** and create **db_orders** database & **orders** table with the following fields (**Figure 16.1**):

```
CREATE TABLE `db_orders`.`orders` (
`OrderID` INT NOT NULL,
`EmployeeID` INT NULL,
`OrderDate` DATE NULL,
`Freight` FLOAT NULL,
`ShipAddress` TEXT(60) NULL,
`ShipCity` TEXT(30) NULL,
`ShipCountry` TEXT(30) NULL,
PRIMARY KEY (`OrderID`));
```



Column Name	Datatype	PK	NN
OrderID	INT	☑	☑
EmployeeID	INT	□	□
OrderDate	DATE	□	□
Freight	FLOAT	□	□
ShipAddress	TINYTEXT	□	□
ShipCity	TINYTEXT	□	□
ShipCountry	TINYTEXT	□	□

Figure 16.1 Fields of **orders** table from **db_orders** database

❸ copy **DB_Orders.db** database from **& 15. SQLite Database DB_Orders.db** folder to **MySQL database** folder of the current application;

❹ run **Transfer.pyw** (**Listing 16.1**) to transfer the database records from SQLite format to the MySQL database;

❺ run **MySQL DataBase.pyw** script; **Listing 16.1** displays the **Python & wxPython** script version.

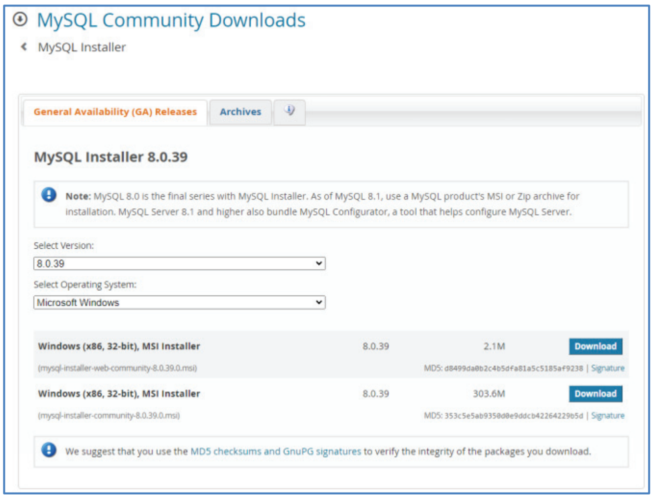
To install **MySQL Server** & **MySQL Workbench,** you need to perform the steps from **table 16.1**:

Table 16.1

Action	Command	Figure
Download **mysql-installer-community-8.0.39.0** file (~ 303 Mb)	Click **Download** button	**Figure 16.2**
Confirm download, save the installer on the computer and open the saved file	Click **No thanks, just start my download** button	**Figure 16.3**
Select **Custom** option	Select **Custom** & click **Next**	**Figure 16.4**
Expand options like in the figure, select optiom and transfer to right window using green right arrow	Move **MySQL Server 8.0.39 – X64** & **MySQL Workbench 8.0.39 – X64** to **Products to Be Installed** & click **Next**	**Figure 16.5**
Confirm installation	Click **Execute**	**Figure 16.6**
Review installation	Click **Next**	**Figure 16.7**
Confirm **MySQL Server 8.0.39** configuration	Click **Next**	**Figure 16.8**
Confirm **Type and Networking** settings	Click **Next**	**Figure 16.9**
Confirm **Authentification Method** settings	Click **Next**	**Figure 16.10**
Input two identical password: **password**;	Click **Next**	**Figure 16.11**
Confirm **Windows Service**	Click **Next**	**Figure 16.12**

Table 16.1

Action	Command	Figure
Confirm **Server File Permission**	Click **Execute**	**Figure 16.13**
Review **Apply Configuration**	Click **Execute**	**Figure 16.14**
Review completed **Apply Configuration**	Click **Finish**	**Figure 16.15**
Confirm **Product Configuration**	Click **Next**	**Figure 16.16**
Review **Installation Complete**	Optional, click on **Copy Log to Clipboard** button; click **Finish**	**Figure 16.17**
Enter on **MySQL Workbench**	Click on **Local instance MySQL80** gray area & Enter password & Save password to vault & Click **OK**	**Figure 16.18**
View **MySQL Workbench** interface + insert 2 x **Query** command: to create **database db_orders** and table **orders**	Enter first SQL command and click ⚡ icon Enter second SQL command and click ⚡ icon	**Figure 16.19**
Expand left area to view **orders** table name and associated icons	Click ▦ icon	**Figure 16.20**



MySQL Community Downloads

‹ MySQL Installer

General Availability (GA) Releases | Archives

MySQL Installer 8.0.39

Note: MySQL 8.0 is the final series with MySQL Installer. As of MySQL 8.1, use a MySQL product's MSI or Zip archive for installation. MySQL Server 8.1 and higher also bundle MySQL Configurator, a tool that helps configure MySQL Server.

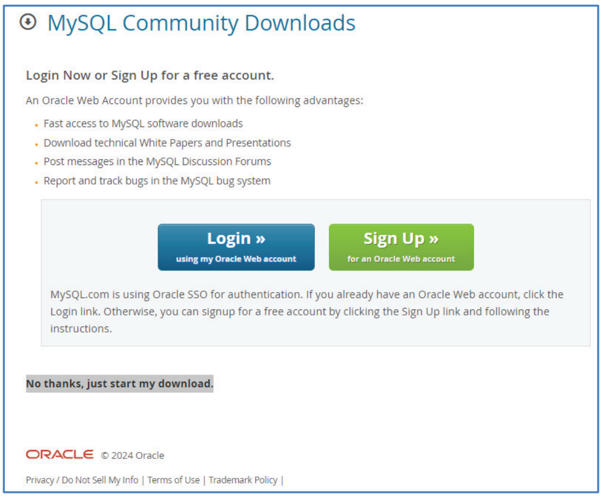
Select Version: 8.0.39

Select Operating System: Microsoft Windows

Windows (x86, 32-bit), MSI Installer (mysql-installer-web-community-8.0.39.0.msi)	8.0.39	2.1M	Download MD5: d8499da0b2c4b5dfa81a5c5185af9238 | Signature
Windows (x86, 32-bit), MSI Installer (mysql-installer-community-8.0.39.0.msi)	8.0.39	303.6M	Download MD5: 353c5e5ab9350d0e9ddcb42264229b5d | Signature

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

Figure 16.2 Download the MySQL installer



MySQL Community Downloads

Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

Login » using my Oracle Web account
Sign Up » for an Oracle Web account

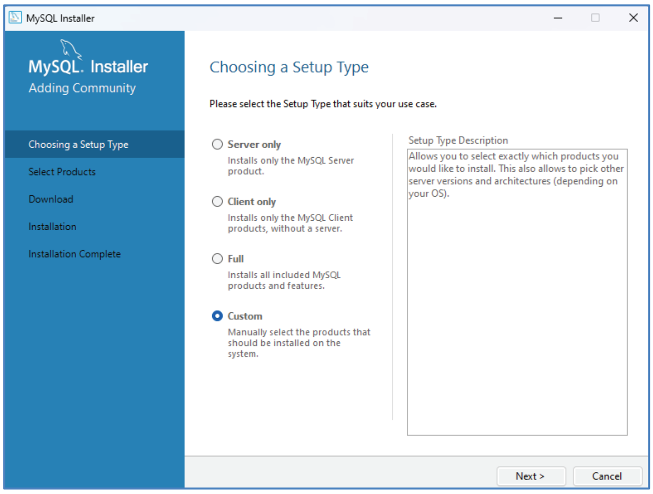
MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

No thanks, just start my download.

ORACLE © 2024 Oracle

Privacy / Do Not Sell My Info | Terms of Use | Trademark Policy |

Figure 16.3 Confirm download



Choosing a Setup Type

Please select the Setup Type that suits your use case.

○ **Server only** Installs only the MySQL Server product.

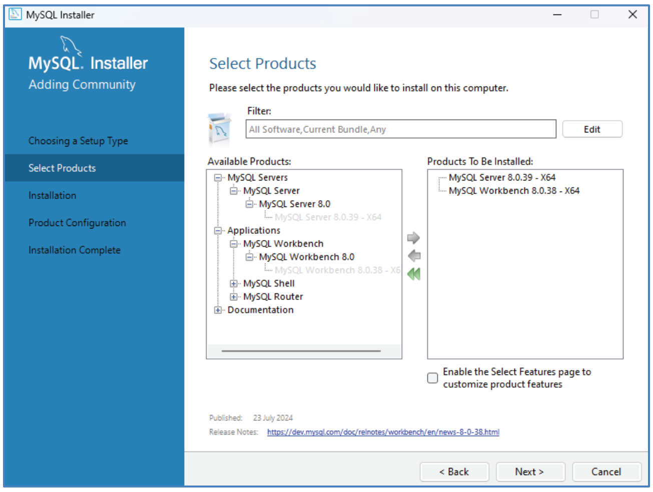
○ **Client only** Installs only the MySQL Client products, without a server.

○ **Full** Installs all included MySQL products and features.

● **Custom** Manually select the products that should be installed on the system.

Setup Type Description: Allows you to select exactly which products you would like to install. This also allows to pick other server versions and architectures (depending on your OS).

Figure 16.4 Select **Custom** option



Select Products

Please select the products you would like to install on this computer.

Filter: All Software,Current Bundle,Any

Available Products:

```
⊟ MySQL Servers
   ⊟ MySQL Server
      ⊟ MySQL Server 8.0
         └ MySQL Server 8.0.39 - X64
⊟ Applications
   ⊟ MySQL Workbench
      ⊟ MySQL Workbench 8.0
         └ MySQL Workbench 8.0.38 - X6
   ⊞ MySQL Shell
   ⊞ MySQL Router
⊞ Documentation
```

Products To Be Installed:

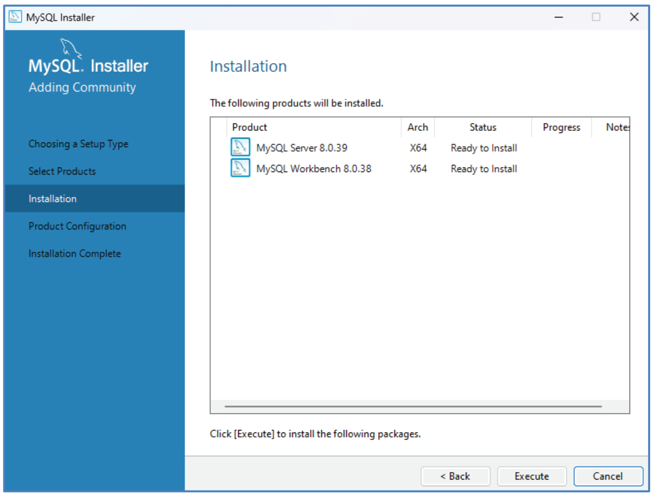
```
MySQL Server 8.0.39 - X64
MySQL Workbench 8.0.38 - X64
```

□ Enable the Select Features page to customize product features

Published: 23 July 2024

Release Notes: https://dev.mysql.com/doc/relnotes/workbench/en/news-8-0-38.html

Figure 16.5 Select files to install



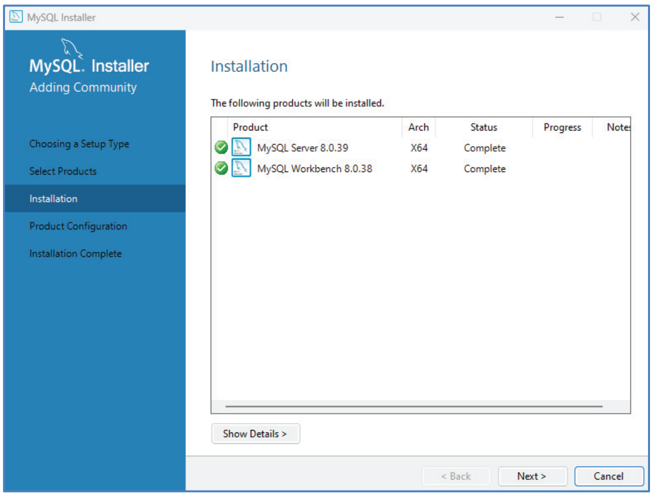
Installation

The following products will be installed.

Product	Arch	Status	Progress	Notes
MySQL Server 8.0.39	X64	Ready to Install		
MySQL Workbench 8.0.38	X64	Ready to Install		

Click [Execute] to install the following packages.

Figure 16.6 Confirm installation

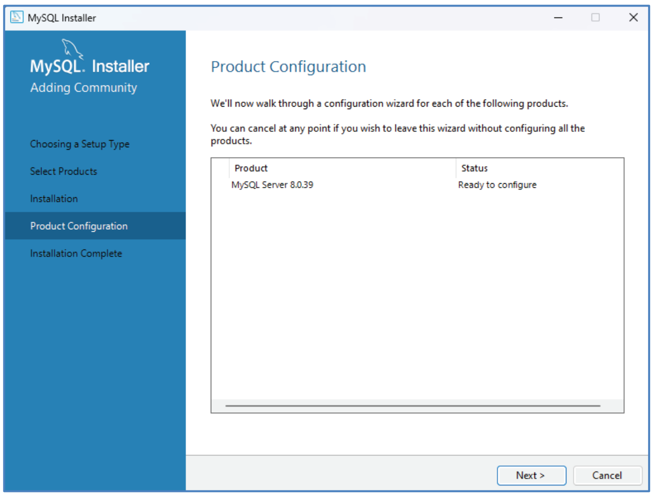


Installation

The following products will be installed.

Product	Arch	Status	Progress	Notes
✓ MySQL Server 8.0.39	X64	Complete		
✓ MySQL Workbench 8.0.38	X64	Complete		

Figure 16.7 Review installation

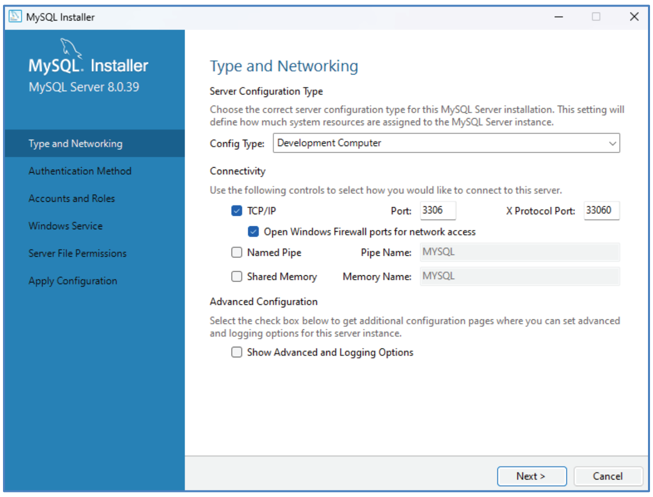


Product Configuration

We'll now walk through a configuration wizard for each of the following products.

You can cancel at any point if you wish to leave this wizard without configuring all the products.

Product	Status
MySQL Server 8.0.39	Ready to configure



Type and Networking

Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type: Development Computer

Connectivity

Use the following controls to select how you would like to connect to this server.

☑ TCP/IP Port: 3306 X Protocol Port: 33060

☑ Open Windows Firewall ports for network access

☐ Named Pipe Pipe Name: MYSQL

☐ Shared Memory Memory Name: MYSQL

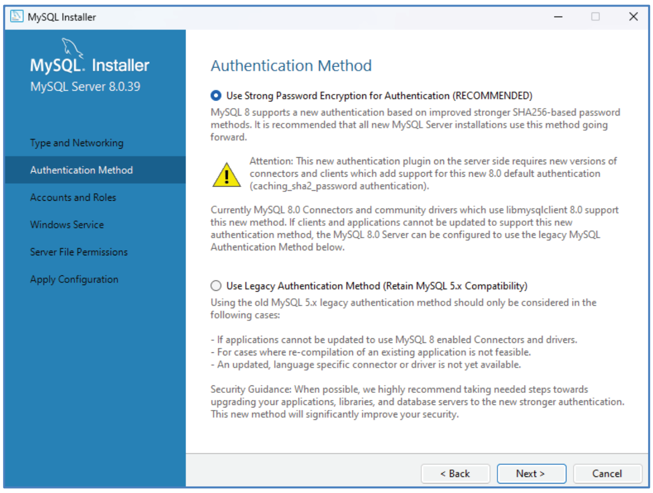
Advanced Configuration

Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.

☐ Show Advanced and Logging Options

Figure 16.8 Confirm configuration

Figure 16.9 Confirm settings



Authentication Method

● Use Strong Password Encryption for Authentication (RECOMMENDED)

MySQL 8 supports a new authentication based on improved stronger SHA256-based password methods. It is recommended that all new MySQL Server installations use this method going forward.

Attention: This new authentication plugin on the server side requires new versions of connectors and clients which add support for this new 8.0 default authentication (caching_sha2_password authentication).

Currently MySQL 8.0 Connectors and community drivers which use libmysqlclient 8.0 support this new method. If clients and applications cannot be updated to support this new authentication method, the MySQL 8.0 Server can be configured to use the legacy MySQL Authentication Method below.

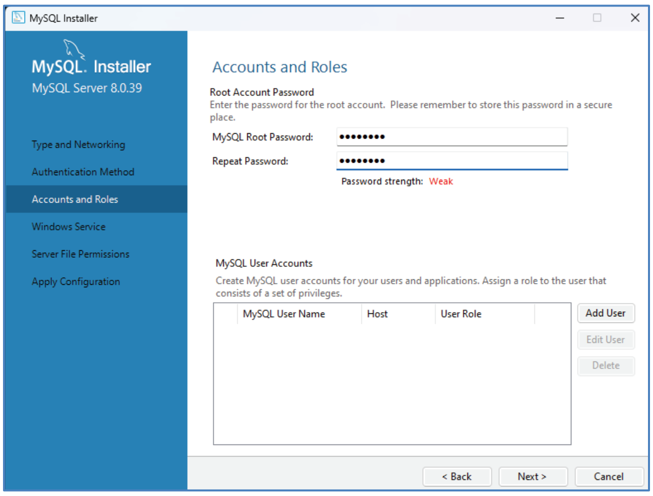
○ Use Legacy Authentication Method (Retain MySQL 5.x Compatibility)

Using the old MySQL 5.x legacy authentication method should only be considered in the following cases:

- If applications cannot be updated to use MySQL 8 enabled Connectors and drivers.
- For cases where re-compilation of an existing application is not feasible.
- An updated, language specific connector or driver is not yet available.

Security Guidance: When possible, we highly recommend taking needed steps towards upgrading your applications, libraries, and database servers to the new stronger authentication. This new method will significantly improve your security.

Figure 16.10 Confirm settings



Accounts and Roles

Root Account Password

Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password: ••••••••

Repeat Password: ••••••••

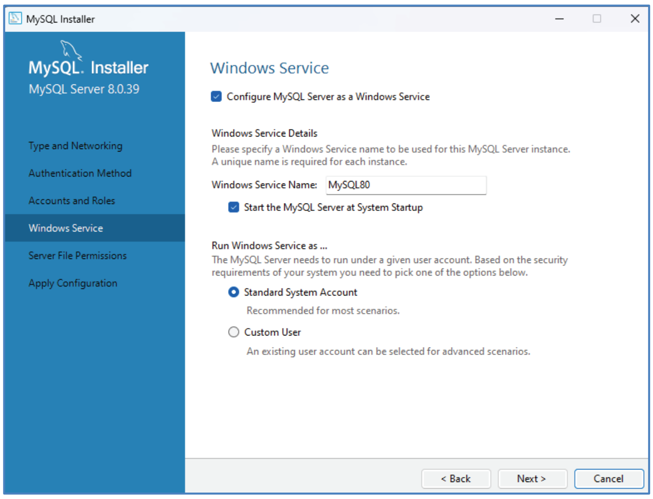
Password strength: Weak

MySQL User Accounts

Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL User Name	Host	User Role

Figure 16.11 Input two identical password



Windows Service

☑ Configure MySQL Server as a Windows Service

Windows Service Details

Please specify a Windows Service name to be used for this MySQL Server instance. A unique name is required for each instance.

Windows Service Name: MySQL80

☑ Start the MySQL Server at System Startup

Run Windows Service as ...

The MySQL Server needs to run under a given user account. Based on the security requirements of your system you need to pick one of the options below.

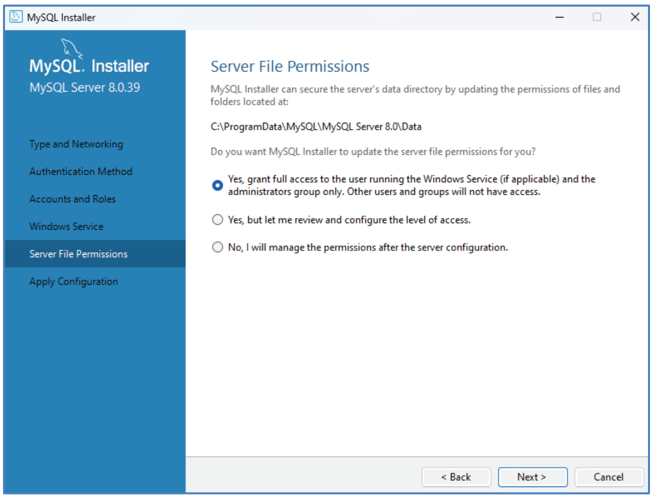
● Standard System Account

Recommended for most scenarios.

○ Custom User

An existing user account can be selected for advanced scenarios.

Figure 16.12 Confirm settings



Server File Permissions

MySQL Installer can secure the server's data directory by updating the permissions of files and folders located at:

C:\ProgramData\MySQL\MySQL Server 8.0\Data

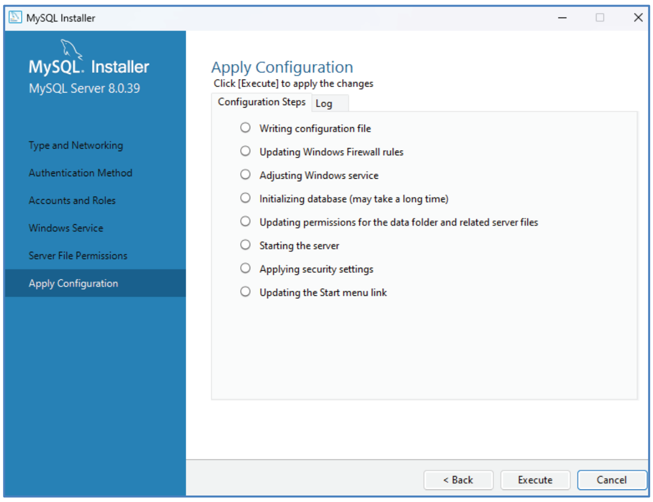
Do you want MySQL Installer to update the server file permissions for you?

● Yes, grant full access to the user running the Windows Service (if applicable) and the administrators group only. Other users and groups will not have access.

○ Yes, but let me review and configure the level of access.

○ No, I will manage the permissions after the server configuration.

Figure 16.13 Confirm settings

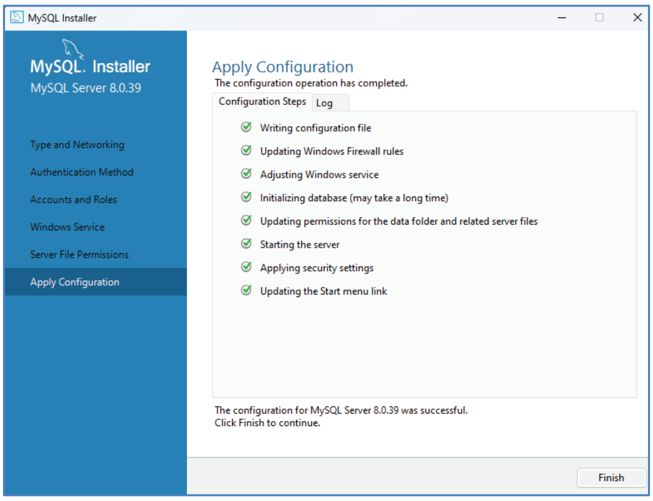


Apply Configuration

Click [Execute] to apply the changes

○ Writing configuration file
○ Updating Windows Firewall rules
○ Adjusting Windows service
○ Initializing database (may take a long time)
○ Updating permissions for the data folder and related server files
○ Starting the server
○ Applying security settings
○ Updating the Start menu link

Figure 16.14 Review **Apply Configuration**



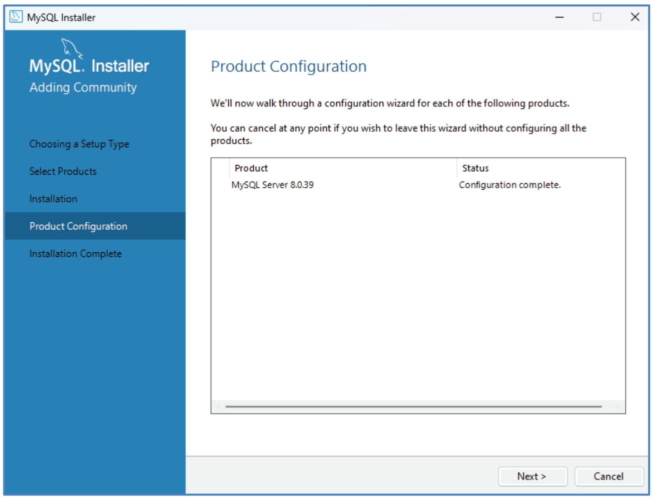
Apply Configuration

The configuration operation has completed.

✓ Writing configuration file
✓ Updating Windows Firewall rules
✓ Adjusting Windows service
✓ Initializing database (may take a long time)
✓ Updating permissions for the data folder and related server files
✓ Starting the server
✓ Applying security settings
✓ Updating the Start menu link

The configuration for MySQL Server 8.0.39 was successful.
Click Finish to continue.

Figure 16.15 Review completed configuration



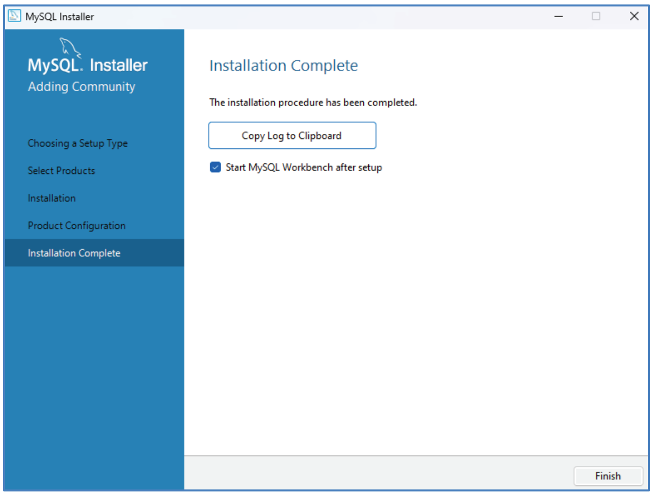
Product Configuration

We'll now walk through a configuration wizard for each of the following products.

You can cancel at any point if you wish to leave this wizard without configuring all the products.

Product	Status
MySQL Server 8.0.39	Configuration complete.

Figure 16.16 Confirm **Product Configuration**

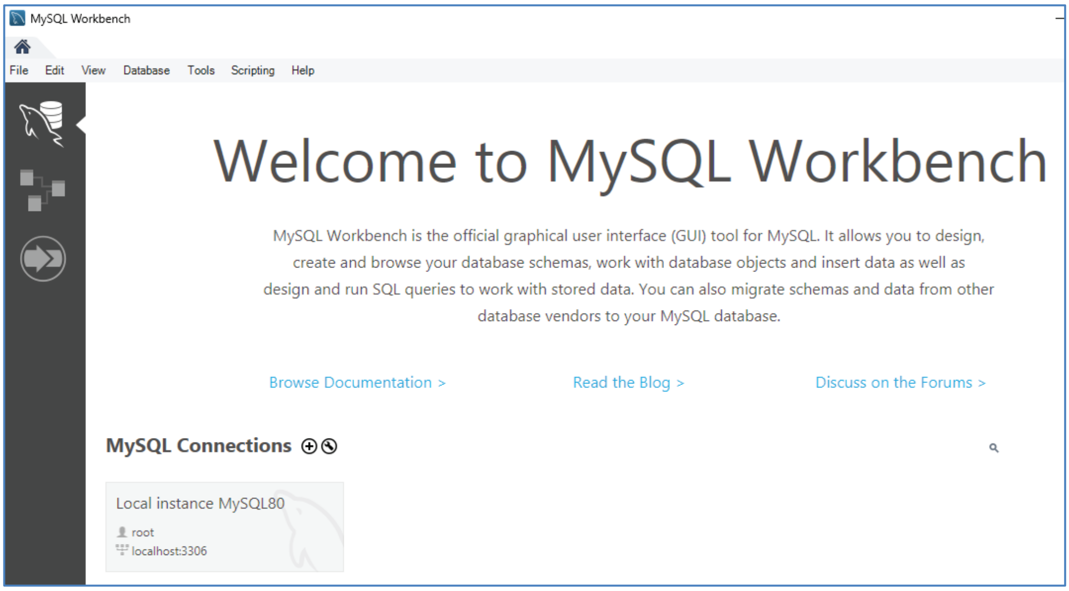


Installation Complete

The installation procedure has been completed.

☑ Start MySQL Workbench after setup

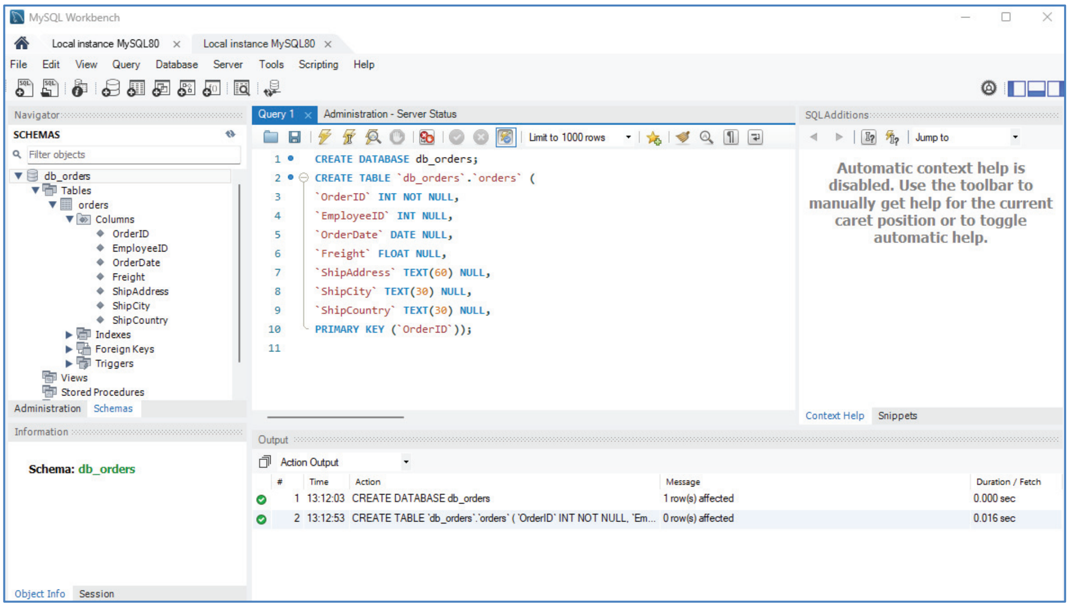
Figure 16.17 Review **Installation Complete**



Welcome to MySQL Workbench

MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

Figure 16.18 Welcome to **MySQL Workbench**



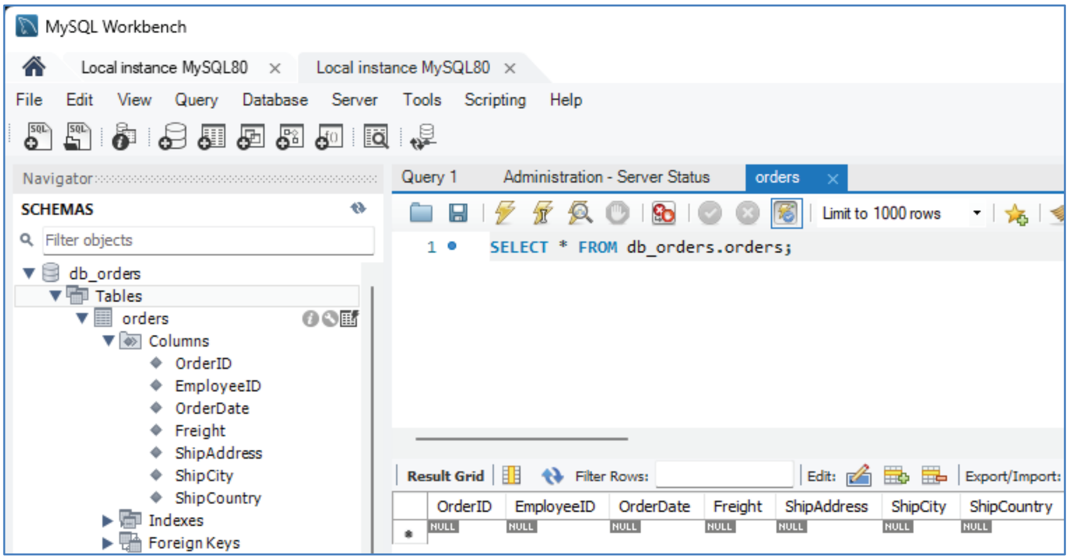
```
CREATE DATABASE db_orders;
CREATE TABLE `db_orders`.`orders` (
`OrderID` INT NOT NULL,
`EmployeeID` INT NULL,
`OrderDate` DATE NULL,
`Freight` FLOAT NULL,
`ShipAddress` TEXT(60) NULL,
`ShipCity` TEXT(30) NULL,
`ShipCountry` TEXT(30) NULL,
PRIMARY KEY (`OrderID`));
```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Schema: db_orders

	#	Time	Action	Message	Duration / Fetch
✓	1	13:12:03	CREATE DATABASE db_orders	1 row(s) affected	0.000 sec
✓	2	13:12:53	CREATE TABLE 'db_orders'.'orders' ('OrderID' INT NOT NULL, 'Em...	0 row(s) affected	0.016 sec

Figure 16.19 View **MySQL Workbench** interface & execute two **Queries**



```
SELECT * FROM db_orders.orders;
```

OrderID	EmployeeID	OrderDate	Freight	ShipAddress	ShipCity	ShipCountry
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure 16.20 View **orders** table

This script saved in **MySQL database** folder is designed to manage a **MySQL** database for the following operations: add, delete, modify record and export database table to Word & Excel.

The name of main database is **db_orders** with only one table **orders**, extracted from original database **northwind.db** [15.1]. Only 7 fields have been extracted for the **Orders** table to avoid increasing the script in number of instructions. Another database **Config.db** is used to store all icons loaded in frame **Toolbar**; this will avoid the existence of following icons files in the application folder: , Delete , , , , , , and . The **SQL** to generate **objects** table is displayed in **Figure 15.4**. The **db_orders.db → orders** memorize the usefull informations for the script and **Config.db → objects** memorize icons for the script interface for **Python 2.7 & wxPython** application version. Initially, the icons files are stored in the **MySQL database → Objects** application folder. The icons files were loaded into **Config.db → Objects** as **Binary Large Object** (**BLOB**) through the **_copyimg.pyw** script displayed in **Listing 15.1**, from where they will be used into application interface through **Python 2.7 → MySQL DataBase.pyw** script.

The **MySQL DataBase.pyw** script (Python & wxPython version) include only two classes: **Class_DB**, **AddModifyRecord** and two public functions (**ExtragImageMemory** & **EXPORT_WORD**). The script begin with importing libraries (lines 2÷8).

Line 8 import **mysql.connector** which is provide connectivity to the MySQL server for client programs. To install the **mysql-connector-python** package, after downloading, type the following command:

```
pip install mysql-connector-python
```

The **ExtragImageMemory** function (lines 10÷14) selecteaza din **Config.db → Objects** an icon si create & returneaza the icon **img** from data in memory. This function is called from class **Class_DB → _TOOLBAR** function.

The **EXPORT_WORD** public function (lines 16÷52) select from **orders** table only a limited records number (No_LIMIT=10 ,line 33) to be exported to Word, but this number can be modified. Anyway, the **Word** export process is much longer the **Excel** export. For this reason the number of exported records was limited.

The **Class_DB** class create the interface (toolbar, grid, …), the database main functionalities and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame), the **StatusBar** (placed at the frame bottom, were messages can be displayed); initializes the public variables of the class (lines 64÷66); call the **_TOOLBAR**, **CreateGrid** and **OnSelect** functions; create the sizer (a **wx.Sizer** control that can lay out items in a virtual grid); add **self.grid** in sizer created by **CreateGrid** function (line 71) and center the frame in the display.
- **_TOOLBAR** Create the **toolbar** (line 75); set the **toolbar** background color (line 76); create the connection and cursor using **Config.db** database (line 77); generate **Lst_label**, **Lst_icons** and **Lst_Help** lists (lines 78÷81), which store the **tooolbar** icons labels, the icon names that must be extracted from memory by calling **ExtragImageMemory** function and the associated icons help; for every operation lines 82÷83 associate an unique identificator ID; through this ID the **Delete** and **Modify** icons can be activated or deactivated, because these operations can be executed only when in **self.grid** control are selected one or more lines, through **OnGridRangeSelect** function; through **for** loop from line 84 the **img** icon is extracted into memory. Line 85 create the bitmap of **img** memory icon and scale at the imposed **toolbar** icons size; in this loop, for every operation, the icon is added in **toolbar** (lines 87, 90, 93, 96, 99, 102, 105 & 108) and connect with **Bind** statement to associated functions (lines 88, 91, 94, 97, 100, 103, 106 &109); line 110 close the connection with **Config.db** database; lines 111, 112 deactivate **Delete** and **Modify** icons, because no line in **self.grid** control is selected; line 113 generate and show the **toolbar**.
- **CreateGrid** This function create **self.grid** control (lines 116÷117) where database records are displayed; connect by **Bind** statement (line 118) with **OnGridRangeSelect** function based on **wx.grid.EVT_GRID_RANGE_SELECT** event; this event notify about a range of cells being selected when the user uses the mouse for selection; to operate directly on grid is not permitted (line 119); lines 120÷138 set the grid parameters: default centre cell alignment (line 120), columns label size (line 121), label value for grid

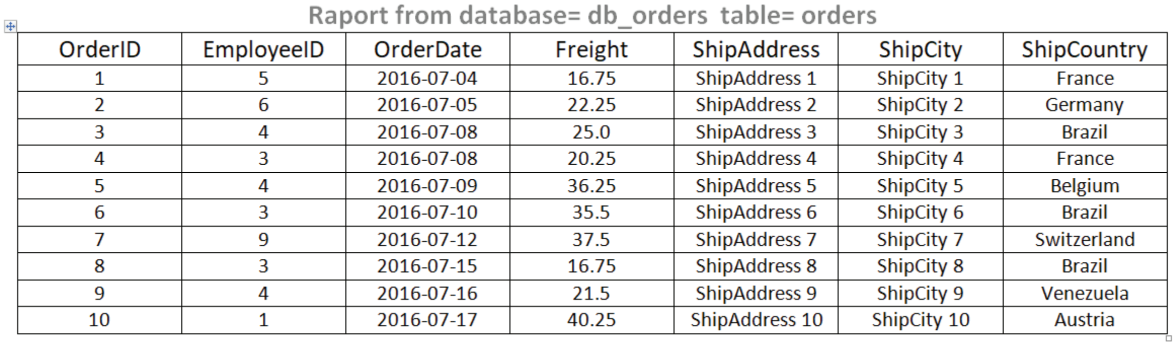
columns (lines 123÷131), columns size (lines 133÷138).

- **OnGridRangeSelect** This function is called from **CreateGrid** function (line 118), when the user select one or more grid; the selection can be made with left mouse click on first grid column to the respective line; to select more lines **CTRL** keyboard taste must be pressed; the selection lines are marked with blue color and a ***** simbol is placed in **Del** grid column; multiple continuous lines can be selected with pressed **Shift** keyboard taste; to deselect selected line/lines a click in the unselected white grid area must be performed. If a selection is detected by line 142, the index of selected line is appended in the **currentSelection** public list (line 145), put the ***** simbol in **Del** grid column (line 147) and refreshing the data in the grid (line 148). If a click is detected in the unselected white grid area (**else** branch from line 149) the index is removed from **currentSelection** list (line 152), put a space character in **Del** grid column (line 154) and refreshing the data in the grid (line 155). Line 156 deactivate **Delete** and **Modify** toolbar icons. Lines 157÷159 count in **csel** variable the number of indexes stored in **currentSelection** list; if multiple selection are detected (line 160) the **Delete** toolbar icons is activated (line 161); if, only one selection & selection process was made, are detected by line 162, the index of selected line are stored in **LinGridSelect** variable; also the value from the grid cell identified by **LinGridSelect** line & **self.COL_rowid** column is stored in **self.CellGridSelect** variable (line 165); if the **self.CellGridSelect** variable is not empty (line 166) then **Modify** toolbar icon is activated (line 167); line 168 ensures more processing on the script.
- **OnDelete** This function deletes selected lines from the grid with confirmation. Lines 180÷182 count the number of selected grid lines and ask for delete confirmation (lines 183÷185); if the answer is **YES**, the connection with **db_orders** database is made and for all selections (line 190) the function go through the lines of the table marked with ***** symbol and delete them (lines 192÷195); line 198 initializes the selection variables and line 199 **ForceRefresh** of the grid; the **Delete** icon from the **toolbar** is deactivated (line 199); finallyh, the **OnSelect** function is called (line 200) to reload the grid with database records, where the delete lines are excluded.
- **OnSelect** This function is called from **Class_DB class** functions at **init** event, after **Delete** operation and also from **AddModifyRecord** class function after add or modify record. If the grid is not empty (line 203) line 204 delete all grid lines; the connection with **db_orders** database is made (lines 205÷206) and all records are loaded into **cursor** variable, which is used for fetching data from table in row-by-row manner; in **for** loop (line 210) every record is extracted from **cursor** variable and put in grid cells (lines 211÷221); also the number of records are stored in **RECCOUNT** variable (lines 223÷224) and displayed in status bar (lines 226÷227).
- **OnAdd** Line 230 deactivate **Delete** and **Modify** toolbar icons. Call **AddModifyRecord** class with **Add Record** option (line 231) and make modal the class frame (line 232) to create a nested event loop so **MakeModal** will not return until the class is finished and input to the other windows in the application will be blocked. Also, the database & table names and the empty string “” are sent to the **AddModifyRecord** class; the empty string “” is transmitted as parameter, because it is not necessary to select a line in the grid to add a record, as it is the case for modify a record.
- **OnModify** Line 235 deactivate **Delete** and **Modify** toolbar icons. Call **AddModifyRecord** class with **Modify Record** option (line 236) and make modal the class frame (line 238) to create a nested event loop so **MakeModal** will not return until the class is finished and input to the other windows in the application will be blocked. Also, the database & table names and the variable **self.CellGridSelect** are sent to the **AddModifyRecord** class; the variable **self.CellGridSelect** store the value of the cell selected to modify and is created in **OnGridRangeSelect** function.
- **OnFirst** This function send the grid position to the first line.
- **OnLast** This function send the grid position to the last line.

- **OnWord** This function export a limited number of records from **orders** table to an **Word** file through public function **EXPORT_WORD**, **Figure 16.21**.
- **OnExcel** This function export the content of database **db_orders.db** table **orders** to an Excel file using **pandas** library.
- **On_Exit** This function exits the application.

The **AddModifyRecord** class implement operations to add or modify a record and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel** (that will hold the contents of the frame), the **Sizer** and create the **font** variable. The function receive from **Class_DB → OnAdd** and **Class_DB → OnModify** the following parameters: **title**, **DATABASE**, **TABLE**, **Parent_Frame** and **CellGridSelect**. The **title** variable receive two values: **Add Record** transmitted by **Class_DB → OnAdd** function and **Modify Record** transmitted by **Class_DB → OnModify** function. Lines 257÷258 create the public variables in **AddModifyRecord** class. The **self.Parent_Frame** store a reference to **Class_DB** from where the class **AddModifyRecord** was called. The **self.CellGridSelect** variable store the cell content of the single line selected from the grid to be modified; in the case of adding a new record the value of this parameter is empty, because it is not necessary to select a line in the grid to add a record, as it is the case for modify a record. Lines 260÷262 define the labels of the controls placed on the **panel**. Lines 269÷286 define define the text controls where the record values will be put or modified. Line 277 define a **CalendarCtrl** control to select the date and line 278 connect by **Bind** statement with **OnCalSelChanged** function. Lines 288÷294 add text controls to the **sizer**. Lines 296÷305 create **Save** and **Exit** buttons and bind these controls to **OnSave** and **OnCloseMe** functions. If the variable **title** has the **Modify Record** value (line 307) then coonection with **db_orders.db → orders** table to extract the fileds values from the record identified by **(_ROWID_= " + str(CellGridSelect)** condition (line 311); these values were placed in text controls (lines 313÷320) created before. Line 322 center and show the frame in the display.
- **OnCalSelChanged** This function is called by **CalendarCtrl** to modify the selected data in yyyy-mm-dd format.
- **OnSave** Create the connection to **db_orders.db → orders** table and define **Fields** list. If the **title** is **Add Record** lines 335÷342 define a complex string that store the text controls values and line 343 create the **SELECT** command; if the **title** is **Modify Record** lines 345÷352 create direct the **SELECT** command. Line 353 execute one of those **SELECT** command and close the connection. Through the **self.Parent_Frame** variable is called **OnSelect** functions from **Class_DB** class to reload the grid with database records.
- **OnCloseMe** Call the **CLOSE** function
- **CLOSE** Cancel **Make Modal** status of **AddModifyRecord** class and close the window.

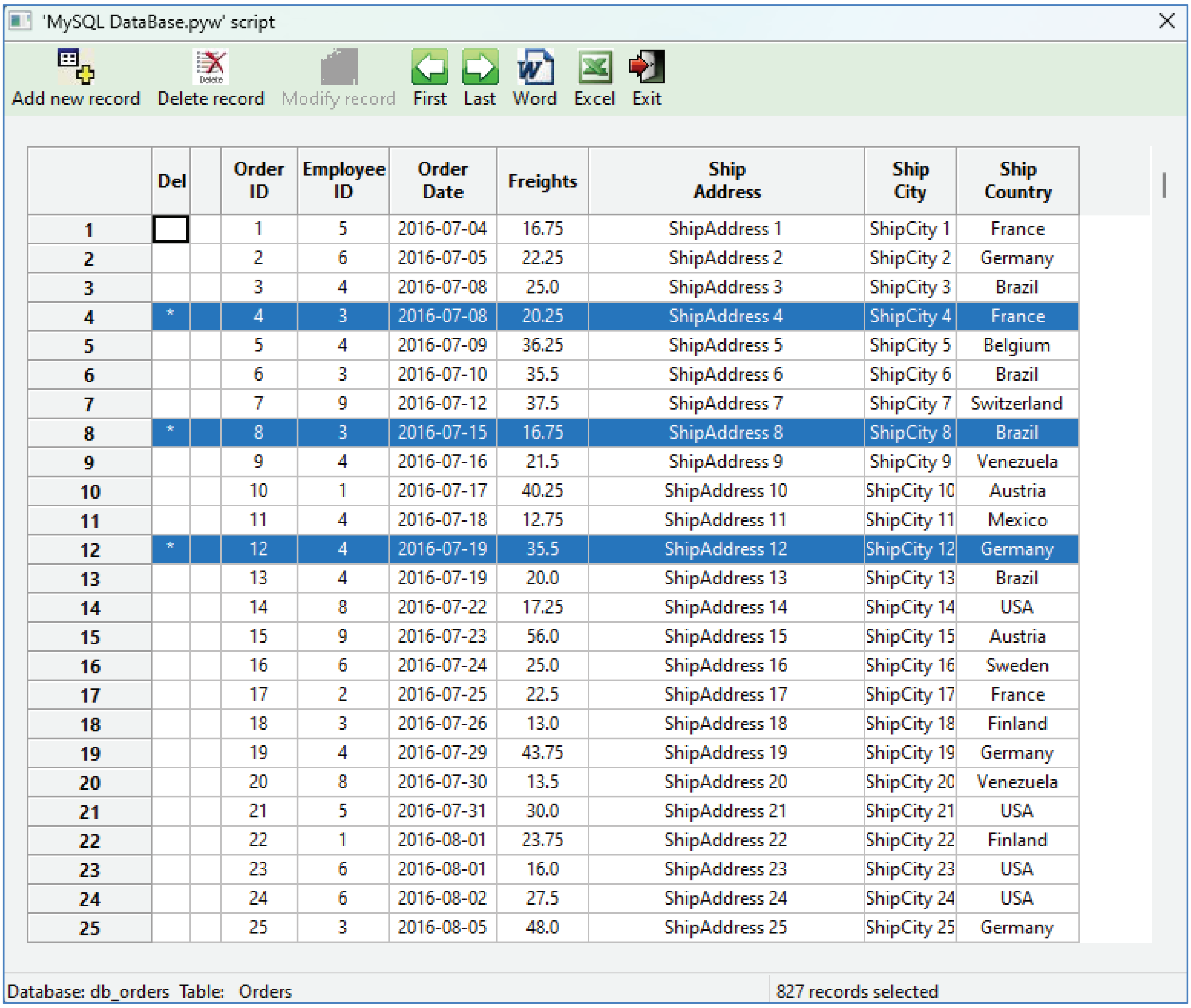


Raport from database= db_orders table= orders

OrderID	EmployeeID	OrderDate	Freight	ShipAddress	ShipCity	ShipCountry
1	5	2016-07-04	16.75	ShipAddress 1	ShipCity 1	France
2	6	2016-07-05	22.25	ShipAddress 2	ShipCity 2	Germany
3	4	2016-07-08	25.0	ShipAddress 3	ShipCity 3	Brazil
4	3	2016-07-08	20.25	ShipAddress 4	ShipCity 4	France
5	4	2016-07-09	36.25	ShipAddress 5	ShipCity 5	Belgium
6	3	2016-07-10	35.5	ShipAddress 6	ShipCity 6	Brazil
7	9	2016-07-12	37.5	ShipAddress 7	ShipCity 7	Switzerland
8	3	2016-07-15	16.75	ShipAddress 8	ShipCity 8	Brazil
9	4	2016-07-16	21.5	ShipAddress 9	ShipCity 9	Venezuela
10	1	2016-07-17	40.25	ShipAddress 10	ShipCity 10	Austria

Figure 16.21 The records exported to Word

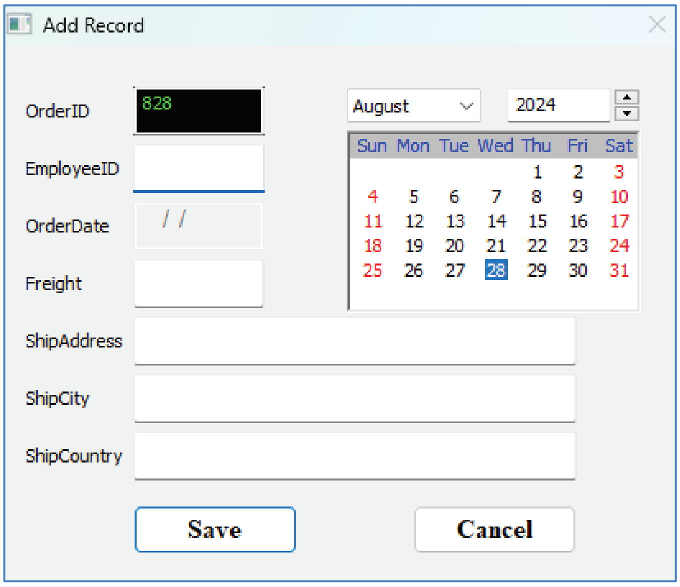
Listing 16.2 displays the **Python & wxPython** version of script, **MySQL DataBase.pyw**. **Figure 16.22** display the main interface of **MySQL DataBase.pyw** script. **Figure 16.23** and **Figure 16.24** display the interface of **Add Record** & **Modify Record** functions.



	Del		Order ID	Employee ID	Order Date	Freights	Ship Address	Ship City	Ship Country
1	□		1	5	2016-07-04	16.75	ShipAddress 1	ShipCity 1	France
2			2	6	2016-07-05	22.25	ShipAddress 2	ShipCity 2	Germany
3			3	4	2016-07-08	25.0	ShipAddress 3	ShipCity 3	Brazil
4	*		4	3	2016-07-08	20.25	ShipAddress 4	ShipCity 4	France
5			5	4	2016-07-09	36.25	ShipAddress 5	ShipCity 5	Belgium
6			6	3	2016-07-10	35.5	ShipAddress 6	ShipCity 6	Brazil
7			7	9	2016-07-12	37.5	ShipAddress 7	ShipCity 7	Switzerland
8	*		8	3	2016-07-15	16.75	ShipAddress 8	ShipCity 8	Brazil
9			9	4	2016-07-16	21.5	ShipAddress 9	ShipCity 9	Venezuela
10			10	1	2016-07-17	40.25	ShipAddress 10	ShipCity 10	Austria
11			11	4	2016-07-18	12.75	ShipAddress 11	ShipCity 11	Mexico
12	*		12	4	2016-07-19	35.5	ShipAddress 12	ShipCity 12	Germany
13			13	4	2016-07-19	20.0	ShipAddress 13	ShipCity 13	Brazil
14			14	8	2016-07-22	17.25	ShipAddress 14	ShipCity 14	USA
15			15	9	2016-07-23	56.0	ShipAddress 15	ShipCity 15	Austria
16			16	6	2016-07-24	25.0	ShipAddress 16	ShipCity 16	Sweden
17			17	2	2016-07-25	22.5	ShipAddress 17	ShipCity 17	France
18			18	3	2016-07-26	13.0	ShipAddress 18	ShipCity 18	Finland
19			19	4	2016-07-29	43.75	ShipAddress 19	ShipCity 19	Germany
20			20	8	2016-07-30	13.5	ShipAddress 20	ShipCity 20	Venezuela
21			21	5	2016-07-31	30.0	ShipAddress 21	ShipCity 21	USA
22			22	1	2016-08-01	23.75	ShipAddress 22	ShipCity 22	Finland
23			23	6	2016-08-01	16.0	ShipAddress 23	ShipCity 23	USA
24			24	6	2016-08-02	27.5	ShipAddress 24	ShipCity 24	USA
25			25	3	2016-08-05	48.0	ShipAddress 25	ShipCity 25	Germany

Database: db_orders Table: Orders   827 records selected

Figure 16.22 The main interface of **MySQL DataBase.pyw** script



OrderID: 828

EmployeeID:

OrderDate: / /

Freight:

ShipAddress:

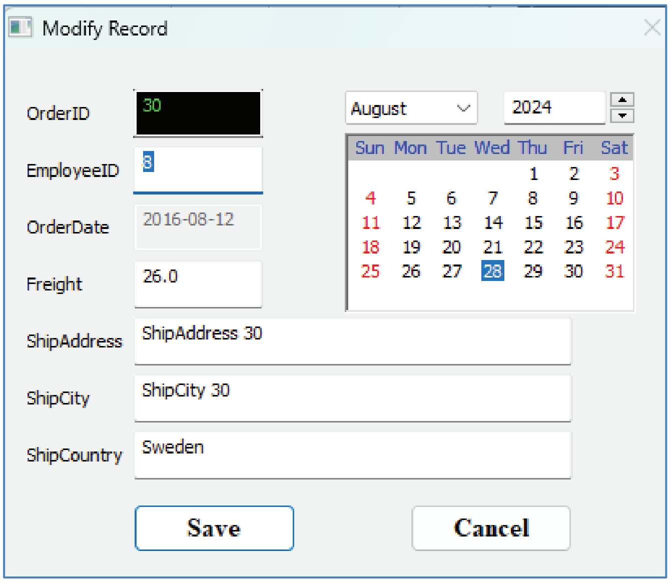
ShipCity:

ShipCountry:

August 2024

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Figure 16.23 The interface of **Add Record**



OrderID: 30

EmployeeID: 3

OrderDate: 2016-08-12

Freight: 26.0

ShipAddress: ShipAddress 30

ShipCity: ShipCity 30

ShipCountry: Sweden

August 2024

Sun	Mon	Tue	Wed	Thu	Fri	Sat
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Figure 16.24 The interface of **Modify Record**

Listing 16.1 Transfer.pyw – Database transfer from SQLite to MySQL – Python

Listing 16.1 Transfer.pyw – Database transfer from SQLite to MySQL – Python

```
# Transfer records from SQLite database to MySQL database
import os
from pysqlite2 import dbapi2 as sqlite # For Python version 2
import mysql.connector
# import sqlite3 as sqlite  # For Python version 3 and print argument must be put in paranthesis

connection = sqlite.connect(os.getcwd()+"\\DB_Orders.db") # Open SQLite Database from current folder
cursor_SQLite = connection.cursor()
Fields ="OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry"
SELECT="select  "+Fields+" from Orders"
CRS = cursor_SQLite.execute(SELECT).fetchall()
cursor_SQLite.execute( "SELECT count(*) FROM Orders")
RECCOUNT=cursor_SQLite.fetchone()[0]
print "\nSQLite databes 'DB_Orders' has: "+str(RECCOUNT)+" records\n "

cnx = mysql.connector.connect(host="localhost" , user="root", passwd="Alfa#Beta_1900",db="db_orders")
cursor_MySQL = cnx.cursor( )

tmp_DELETE ="DELETE FROM orders"
cursor_MySQL.execute( tmp_DELETE)
cnx.commit()

Counter = 0
for (OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry) in CRS:
    # print OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry
    tmp_INSERT="INSERT INTO orders (OrderID, EmployeeID, OrderDate, "
    tmp_INSERT=tmp_INSERT+"Freight,ShipAddress,ShipCity,ShipCountry) VALUES ("
    tmp_INSERT=tmp_INSERT+"'"+str(OrderID)+"','"+str(EmployeeID)+"','"+str(OrderDate)+"','"
    tmp_INSERT=tmp_INSERT+str(Freight)+"','"+ShipAddress+"','"+ShipCity+"','"+ShipCountry+"')"
    # print tmp_INSERT
    cursor_MySQL.execute( tmp_INSERT)
    Counter = Counter + 1
cnx.commit()
cursor_SQLite.close()  ;  connection.close()

cursor_MySQL.execute( "Select Max(OrderID) from orders")
CRS_Max_OrderID=cursor_MySQL.fetchall()
Max_OrderID=CRS_Max_OrderID[0][0]

print "MySQL databes last record ID = ", Max_OrderID
print "MySQL databes 'DB_Orders' has: "+str(Counter)+" records"
cursor_MySQL.execute( "Select * from orders")
lst_NAMES=[]
for elem in cursor_MySQL.description:
    lst_NAMES.append(str(elem[0]))
print "List of fields names = ",lst_NAMES
cnx.close()
```

Listing 16.2 MySQL DataBase.pyw – MySQL Database – Python & wxPython version

```
from __future__ import division
import os, wx, StringIO
from pysqlite2 import dbapi2 as sqlite
import wx.grid, wx.calendar
import pandas as pd
import win32com.client
from win32com.client import Dispatch
import mysql.connector

def ExtragImageMemory(cursor, NumeImage):
        cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
        blob=cursor.fetchone()[0]
        img = wx.ImageFromStream(StringIO.StringIO(blob))
        return img

def EXPORT_WORD(DATABASE, TABLE):
        # 1) Creating a word application object
        wordApp = win32com.client.gencache.EnsureDispatch('Word.Application')
        wordApp.Visible = True # Show Word application
        doc = wordApp.Documents.Add() # Create a new document
```

Listing 16.2 MySQL DataBase.pyw – MySQL Database – Python & wxPython version

```
    # 2) Formating the document
    doc.PageSetup.RightMargin = 20 ; doc.PageSetup.LeftMargin = 20
    doc.PageSetup.Orientation = win32com.client.constants.wdOrientLandscape
    doc.PageSetup.PageWidth = 595   ; doc.PageSetup.PageHeight = 842
    header_range= doc.Sections(1).Headers(win32com.client.constants.wdHeaderFooterPrimary).Range
    header_range.ParagraphFormat.Alignment = win32com.client.constants.wdAlignParagraphCenter
    header_range.Font.Bold = True ; header_range.Font.Size = 12
    header_range.Text = "Raport from database= "+DATABASE+"  table= "+TABLE
    # 3) Connect with DATABASE
    connection = mysql.connector.connect(host="localhost" , user="root", passwd="Alfa#Beta_1900",db=DATABASE)
    Fields ="OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry"
    # 4) Inserting Table
    No_LIMIT=10 ; total_column = 7 ; total_row = No_LIMIT+1 ; rng = doc.Range(0,0)
    rng.ParagraphFormat.Alignment = win32com.client.constants.wdAlignParagraphCenter
    table = doc.Tables.Add(rng,total_row, total_column) ; table.Borders.Enable = True
    cursor = connection.cursor() ; cursor.execute( "select "+Fields+" from "+TABLE ) ; lst_NAMES=[]
    for elem in cursor.description: lst_NAMES.append(str(elem[0]))
    for col in range(1,8): table.Cell(1, col).Range.InsertAfter(lst_NAMES[col-1])
    connection.close()
    connection1 = mysql.connector.connect(host="localhost" , user="root", passwd="Alfa#Beta_1900",db=DATABASE)
    cursor1 = connection1.cursor() ; lin=2
    SELECT="select "+Fields+" from "+TABLE+" LIMIT "+str(No_LIMIT) ; cursor1.execute(SELECT)
    # if total_column > 1:  table.Columns.DistributeWidth()
    for (OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry) in cursor1.fetchall():
        lst=[OrderID,EmployeeID,OrderDate,Freight,ShipAddress,ShipCity,ShipCountry]
        for col in range(1,8):
            cell_range= table.Cell(lin, col).Range ; cell_range.Font.Size = 9
            cell_range.ParagraphFormat.LineSpacingRule=win32com.client.constants.wdLineSpaceSingle
            cell_range.ParagraphFormat.SpaceBefore = 0 ; cell_range.ParagraphFormat.SpaceAfter = 0
            table.Cell(lin, col).Range.InsertAfter(str(lst[col-1]))
        lin=lin+1
    connection.close()

class Class_DB(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, "'MySQL DataBase.pyw' script", size=(780, 660),
            style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
        self.panel = wx.Panel(self, -1)

        self.statusbar = self.CreateStatusBar() ; self.statusbar.SetFieldsCount(2)
        self.statusbar.SetStatusWidths([-4, -2]) ;self.statusbar.SetStatusText("",0)
        self.statusbar.SetStatusText("",1)

        self.DATABASE="db_orders" ; self.TABLE="Orders" ; self.currentSelection=[]
        self.COL_del=0      # The grid column with marking for delete record
        self.COL_rowid=2 # The grid column where the ROWID from the database is placed

        self._TOOLBAR() ; self.CreateGrid() ; self.OnSelect()

        sizer= wx.GridBagSizer(hgap=5, vgap=0)
        sizer.Add(self.grid,pos=(1,1),span=(1,8))
        self.panel.SetSizerAndFit(sizer) ; self.CenterOnScreen() ; self.Show()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  |wx.TB_FLAT | wx.TB_3DBUTTONS | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))
        connection = sqlite.connect(os.getcwd()+'\Config.db') ; cursor = connection.cursor()
        Lst_label=[ "Add new record", "Delete record", "Modify record", "First","Last","Word","Excel", "Exit"]
        Lst_icons=[ "Add.jpg", "Delete.jpg", "Modify.jpg","First.png","Last.png","MsWord.jpg","Excel.png","Exit.png"]
        Lst_Help=["Add new record in database","Delete selected record","Modify selected record",
            "Go to FIRST record","Go to LAST record","Export table to Word","Export table to Excel","Exit"]
        self.Add=100 ; self.Delete=200 ; self.Modify=300 ; self.First=400 ; self.Last=500
        self.MsWord=600 ; self.Excel=700 ; self.Exit=800
        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            img = ExtragImageMemory(cursor, icon) ; sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
            if label=="Add new record":
                self.toolbar.AddLabelTool(self.Add, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.OnAdd, id=self.Add)
            if label=="Delete record":
                self.toolbar.AddLabelTool(self.Delete, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.OnDelete, id=self.Delete)
            if label=="Modify record":
```

Listing 16.2 MySQL DataBase.pyw – MySQL Database – Python & wxPython version

```
                self.toolbar.AddLabelTool(self.Modify, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.OnModify, id=self.Modify)
            if label=="First":
                self.toolbar.AddLabelTool(self.First, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.OnFirst, id=self.First)
            if label=="Last":
                self.toolbar.AddLabelTool(self.Last, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.OnLast, id=self.Last)
            if label=="Word":
                self.toolbar.AddLabelTool(self.MsWord, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.OnWord, id=self.MsWord)
            if label=="Excel":
                self.toolbar.AddLabelTool(self.Excel, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.OnExcel, id=self.Excel)
            if label=="Exit":
                self.toolbar.AddLabelTool(self.Exit, label, sel_bmp, shortHelp=HELP)
                self.Bind(wx.EVT_TOOL, self.On_Exit, id=self.Exit)
            cursor.close() ; connection.close()
            self.toolbar.EnableTool(self.Delete, False)  # Delete
            self.toolbar.EnableTool(self.Modify, False)  # Modify
            self.toolbar.Realize() ; self.toolbar.Show()

    def CreateGrid(self):
        self.grid = wx.grid.Grid(self.panel, size=(750, 520))
        self.grid.CreateGrid( 100, 9)
        self.Bind(wx.grid.EVT_GRID_RANGE_SELECT,self.OnGridRangeSelect)
        self.grid.EnableEditing(0)
        self.grid.SetDefaultCellAlignment(wx.ALIGN_CENTRE,wx.ALIGN_CENTRE)
        self.grid.SetColLabelSize(45)

        self.grid.SetColLabelValue(self.COL_del,'Del')
        self.grid.SetColLabelValue(1,'')
        self.grid.SetColLabelValue(2,'Order\nID')
        self.grid.SetColLabelValue(3,'Employee\nID')
        self.grid.SetColLabelValue(4,'Order\nDate')
        self.grid.SetColLabelValue(5,'Freights')
        self.grid.SetColLabelValue(6,'Ship\nAddress')
        self.grid.SetColLabelValue(7,'Ship\nCity')
        self.grid.SetColLabelValue(8,'Ship\nCountry')

        self.grid.SetColSize(self.COL_del,25) ; self.grid.SetColSize(1,20)
        self.grid.SetColSize(self.COL_rowid,40)
        self.grid.SetColSize(2,50) ; self.grid.SetColSize(3,60)
        self.grid.SetColSize(4,70) ; self.grid.SetColSize(5,60)
        self.grid.SetColSize(6,180) ; self.grid.SetColSize(7,80)
        self.grid.SetColSize(7,60)

    def OnGridRangeSelect( self, event ):
        # Ncol=self.grid.GetNumberCols()
        if event.Selecting():
            for index in range( event.GetTopRow(), event.GetBottomRow()+1):
                if index not in self.currentSelection:
                    self.currentSelection.append( index )
                    # Mark selected line in 'self.COL_del' column with '*' key
                    self.grid.SetCellValue(index,self.COL_del,'*')
                    self.grid.ForceRefresh()
        else:
            for index in range( event.GetTopRow(), event.GetBottomRow()+1):
                while index in self.currentSelection:
                    self.currentSelection.remove( index )
                     # Deselect selected line by deleting the '*' key in the 'self.COL_del' column
                    self.grid.SetCellValue(index,self.COL_del,'')
                    self.grid.ForceRefresh()
        self.toolbar.EnableTool(self.Delete, False) ; self.toolbar.EnableTool(self.Modify, False)
        csel=0
        for x in self.currentSelection:
            csel=csel+1
        if ((csel > 0)):
            self.toolbar.EnableTool(self.Delete, True)
        if csel == 1 and event.Selecting():
            coordonate= event.GetTopLeftCoords(), event.GetBottomRightCoords()
            LinGridSelect= coordonate[0][0]
```

Listing 16.2 MySQL DataBase.pyw – MySQL Database – Python & wxPython version

```
            self.CellGridSelect=self.grid.GetCellValue(LinGridSelect,self.COL_rowid )
            if self.CellGridSelect.strip() !='':
                self.toolbar.EnableTool(self.Modify, True)
        event.Skip()

    def OnFirst(self, event):
        self.grid.MakeCellVisible(0,0)

    def OnLast(self, event):
        self.grid.MakeCellVisible(self.grid.GetNumberRows()-1,0)

    def OnWord(self, event):
        EXPORT_WORD(self.DATABASE, self.TABLE)

    def OnDelete(self, event):
        csel=0
        for x in self.currentSelection:
            csel=csel+1
        d = wx.MessageDialog(self, 'Confirm deletion of '+str(csel)+' selected records ?', \
            caption = "Confirm", style = wx.YES_NO | wx.ICON_QUESTION)
        response = d.ShowModal()
        if response == wx.ID_YES:
            connection = mysql.connector.connect(host="localhost" , user="root",
                passwd="Alfa#Beta_1900",db=self.DATABASE)
            cursor = connection.cursor( )
            for x in range(csel):       # Deleting marked records from the table
                for lin in range(self.grid.GetNumberRows()):
                    if self.grid.GetCellValue(lin,self.COL_del)=='*':
                        del_ID=str(self.grid.GetCellValue(lin,self.COL_rowid))
                         SLCT="DELETE from "+self.TABLE+" WHERE OrderID='"+del_ID+"'"
                        cursor.execute(SLCT) ; connection.commit()
                        break
            cursor.close() ; connection.close()
        self.grid.ClearSelection() ; self.currentSelection=[]
        self.grid.ForceRefresh() ; self.toolbar.EnableTool(self.Delete, False)
        self.OnSelect()

    def OnSelect(self):
        if self.grid.GetNumberRows()>0:
            self.grid.DeleteRows(0, self.grid.GetNumberRows())
        connection = mysql.connector.connect(host="localhost" , user="root",
            passwd="Alfa#Beta_1900",db=self.DATABASE)
        cursor = connection.cursor( )
        Fields ="OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry"
        cursor.execute("select "+Fields+" from "+self.TABLE) ; lin=0
        for (OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry) in cursor.fetchall():
                self.grid.AppendRows(1)
                self.grid.SetCellValue(lin,self.COL_del,'') # Del
                self.grid.SetCellValue(lin,self.COL_rowid,"") # ROWID
                self.grid.SetCellValue(lin,2,str(OrderID))
                self.grid.SetCellValue(lin,3,str(EmployeeID) )
                self.grid.SetCellValue(lin,4,str(OrderDate))
                self.grid.SetCellValue(lin,5,str(Freight))
                self.grid.SetCellValue(lin,6,ShipAddress)
                self.grid.SetCellValue(lin,7,ShipCity)
                self.grid.SetCellValue(lin,8,ShipCountry)
                lin=lin+1
        self.grid.MakeCellVisible(0,0)
        cursor.execute( "SELECT count(*) FROM "+self.TABLE)
        RECCOUNT=cursor.fetchone()[0]
        cursor.close()  ;  connection.close()
        self.statusbar.SetStatusText("Database: " +self.DATABASE+"  Table:   "+self.TABLE,0)
        self.statusbar.SetStatusText(str(RECCOUNT)+' records selected',1)

    def OnAdd(self, event):
        self.toolbar.EnableTool(self.Delete, False) ; self.toolbar.EnableTool(self.Modify, False)
        AMR=AddModifyRecord(None, 'Add Record', self, self.DATABASE, self.TABLE, "")
        AMR.MakeModal(True)

    def OnModify(self, event):
        self.toolbar.EnableTool(self.Delete, False) ; self.toolbar.EnableTool(self.Modify, False)
        MMR=AddModifyRecord(None, 'Modify Record', self, self.DATABASE, self.TABLE, self.CellGridSelect)
```

Listing 16.2 MySQL DataBase.pyw – MySQL Database – Python & wxPython version

```
        # wx.MessageBox( str(self.CellGridSelect) )
        MMR.MakeModal(True)

    def OnExcel(self, event):
        connection = mysql.connector.connect(host="localhost" , user="root", passwd="Alfa#Beta_1900",db="db_orders")
        df = pd.read_sql("SELECT * FROM "+self.TABLE, connection)
        file_name=os.getcwd()+'/'+self.DATABASE+"__"+self.TABLE+".csv"
        df.to_csv(file_name) ; wx.MessageBox( "File saved to:\n\n"+file_name)

    def On_Exit(self, event):
        self.Destroy()

# =============================
class AddModifyRecord(wx.Frame):
    def __init__(self, parent, title, Parent_Frame, DATABASE, TABLE, CellGridSelect):
        wx.Frame.__init__(self, parent, title=title, size=(420,360),
            style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
            wx.MAXIMIZE_BOX | wx.CLOSE_BOX)) # Initialize the frame
        panel = wx.Panel(self)  # The panel will hold the contents of the frame
        sizer= wx.GridBagSizer(hgap=5, vgap=5) ; font = wx.Font(12, wx.ROMAN, wx.NORMAL, wx.BOLD)
        self.DATABASE=DATABASE ; self.TABLE=TABLE ;
        self.Parent_Frame=Parent_Frame ; self.CellGridSelect=CellGridSelect
        Lst_Labels=['OrderID', 'EmployeeID', 'OrderDate', 'Freight','ShipAddress','ShipCity','ShipCountry']
        for i, lbl in enumerate(Lst_Labels):
            self.st=wx.StaticText(panel, -1, lbl)
            sizer.Add(self.st,pos=(i+1,1),flag=wx.ALIGN_CENTER_VERTICAL | wx.ALIGN_LEFT)
        connection = mysql.connector.connect(host="localhost" , user="root",
            passwd="Alfa#Beta_1900",db=self.DATABASE)
        cursor = connection.cursor( )
        cursor.execute( "Select Max(OrderID) from orders")
        CRS_Max_OrderID=cursor.fetchall() ; self.Max_OrderID=CRS_Max_OrderID[0][0]+1
        cursor.close()  ;  connection.close()
        self.txt_OrderID = wx.TextCtrl(panel, -1, str(self.Max_OrderID), size=(80, 30))
        self.txt_OrderID.SetToolTipString("Input an integer number")
        self.txt_OrderID.SetEditable(False)
        self.txt_OrderID.SetBackgroundColour('Black')
        self.txt_OrderID.SetForegroundColour("Green")
        self.txt_EmployeeID = wx.TextCtrl(panel, -1, "", size=(80, 30))
        self.txt_EmployeeID.SetToolTipString("Input an integer number")
        self.txt_OrderDate = wx.TextCtrl(panel, -1, "    /  /  ", size=(80, 30))
        calendar = wx.calendar.CalendarCtrl(panel, -1, wx.DateTime_Now(), pos=(210,26))
        self.Bind(wx.calendar.EVT_CALENDAR_SEL_CHANGED, self.OnCalSelChanged, calendar)
        self.txt_Freight = wx.TextCtrl(panel, -1, "", size=(80, 30))
        self.txt_Freight.SetToolTipString("Input an float number")
        self.txt_ShipAddress = wx.TextCtrl(panel, -1, "", size=(270, 30))
        self.txt_ShipAddress.SetToolTipString("Input text")
        self.txt_ShipCity = wx.TextCtrl(panel, -1, "", size=(270, 30))
        self.txt_ShipCity.SetToolTipString("Input text")
        self.txt_ShipCountry = wx.TextCtrl(panel, -1, "", size=(270, 30))
        self.txt_ShipCountry.SetToolTipString("Input text")

        sizer.Add(self.txt_OrderID,pos=(1,2))
        sizer.Add(self.txt_EmployeeID,pos=(2,2))
        sizer.Add(self.txt_OrderDate,pos=(3,2)) ; self.txt_OrderDate.Enabled=False
        sizer.Add(self.txt_Freight,pos=(4,2))
        sizer.Add(self.txt_ShipAddress,pos=(5,2))
        sizer.Add(self.txt_ShipCity,pos=(6,2))
        sizer.Add(self.txt_ShipCountry,pos=(7,2))

        btn_Save= wx.Button(panel, -1, "Save", size=(100, 30), pos=(80,280))
        self.Bind(wx.EVT_BUTTON, self.OnSave, btn_Save)
        btn_Save.SetToolTipString("Save record to database")
        btn_Save.SetFont(wx.Font(12, wx.ROMAN, wx.NORMAL, wx.BOLD))
        btn_Save.SetDefault()

        btn_Exit = wx.Button(panel, -1, "Cancel", size=(100, 30), pos=(250,280))
        self.Bind(wx.EVT_BUTTON, self.OnCloseMe,btn_Exit)
        btn_Exit.SetToolTipString("Close window without saving changes")
        btn_Exit.SetFont(wx.Font(12, wx.ROMAN, wx.NORMAL, wx.BOLD))

        if self.Title=='Modify Record':
            connection = mysql.connector.connect(host="localhost" , user="root", \
```

Listing 16.2 MySQL DataBase.pyw – MySQL Database – Python & wxPython version

```
                    passwd="Alfa#Beta_1900",db=self.DATABASE)
            cursor = connection.cursor( )
            SELECT ="select * from "+self.TABLE+" where (OrderID = " + str(CellGridSelect) +")"
            cursor.execute(SELECT)  ;  CRS=cursor.fetchall()
            for (OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry) in CRS:
                self.txt_OrderID.SetValue(str(OrderID))
                self.txt_EmployeeID.SetValue(str(EmployeeID))
                self.txt_OrderDate.SetValue(str(OrderDate).replace("/", "-", 2))
                self.txt_Freight.SetValue(str(Freight))
                self.txt_ShipAddress.SetValue(str(ShipAddress))
                self.txt_ShipCity.SetValue(str(ShipCity))
                self.txt_ShipCountry.SetValue(str(ShipCountry))
            cursor.close()  ;  connection.close()
            panel.SetSizerAndFit(sizer) ; self.CenterOnScreen() ; self.Show()

        def OnCalSelChanged(self, event):
            cal = event.GetEventObject()  ;  str_Data=str(cal.GetDate())[:8]
            month, day, year = map(str, str_Data.split('/'))
            self.txt_OrderDate.SetValue( "20"+year+"-"+month+"-"+day )

        def OnSave(self, event):
            connection = mysql.connector.connect(host="localhost" , user="root",
                passwd="Alfa#Beta_1900",db=self.DATABASE)
            cursor = connection.cursor( )
            Fields ="OrderID, EmployeeID, OrderDate, Freight,ShipAddress,ShipCity,ShipCountry"
            if self.Title=='Add Record':
                str_Fields=''
                str_Fields=str_Fields+"'"+str(self.Max_OrderID)+"',"
                str_Fields=str_Fields+"'"+self.txt_EmployeeID.GetLineText(0).strip()+"',"
                str_Fields=str_Fields+"'"+self.txt_OrderDate.GetLineText(0).strip()+"',"
                str_Fields=str_Fields+"'"+self.txt_Freight.GetLineText(0).strip()+"',"
                str_Fields=str_Fields+"'"+self.txt_ShipAddress.GetLineText(0).strip()+"',"
                str_Fields=str_Fields+"'"+self.txt_ShipCity.GetLineText(0).strip()+"',"
                str_Fields=str_Fields+"'"+self.txt_ShipCountry.GetLineText(0).strip()+"'"
                SELECT="INSERT INTO "+self.TABLE+"  ("+Fields+") VALUES ( "+str_Fields+")"
            if self.Title=='Modify Record':
                SELECT="UPDATE "+self.TABLE+" SET "+ \
                    "EmployeeID="+"'"+self.txt_EmployeeID.GetLineText(0)+"',"+ \
                    "OrderDate="+"'"+self.txt_OrderDate.GetLineText(0)+"',"+ \
                    "Freight="+"'"+self.txt_Freight.GetLineText(0)+"',"+ \
                    "ShipAddress="+"'"+self.txt_ShipAddress.GetLineText(0)+"',"+ \
                    "ShipCity="+"'"+self.txt_ShipCity.GetLineText(0)+"',"+ \
                    "ShipCountry="+"'"+self.txt_ShipCountry.GetLineText(0)+"'"+ \
                    " where (OrderID= " + str(self.CellGridSelect) +")"
            cursor.execute( SELECT)  ; connection.commit()
            cursor.close()  ;  connection.close()
            self.Parent_Frame.OnSelect() # Call 'OnSelect' function from Parent_Frame='Class_Dbclass' class
            self.CLOSE()

        def OnCloseMe(self, event):
            self.CLOSE()

        def CLOSE(self):
            self.MakeModal(False) ; self.Close(True) ; self.Destroy()

app = wx.App(redirect=0)
frm_DB=Class_DB()
app.MainLoop()
```

17. Image Viewer – Matplotlib & OpenCV

This script is a simple application to manage a images. The script use a database **Config.db** to store all icons loaded in frame **Toolbar**; this will avoid the existence of following icons files in the application folder: ↶, G, B, C, ⇄, ↘, 🖼 and ➡. The **Config.db → objects** store icons for the script interface of **Python 2. 7 & wxPython** application version. Initially, the icons files are stored in the **Image_Viewer → Objects** application folder. The icons files were loaded into **Config.db → objects** as **Binary Large Object** (**BLOB**) through the **_copyimg.pyw** script displayed in **Listing 15.1**, from where they will be used into application interface through **Python 2.7 → Image_Viewer.pyw** script. For **Python & Streamlit → Image_Viewer.py** script version icon files have not been used.

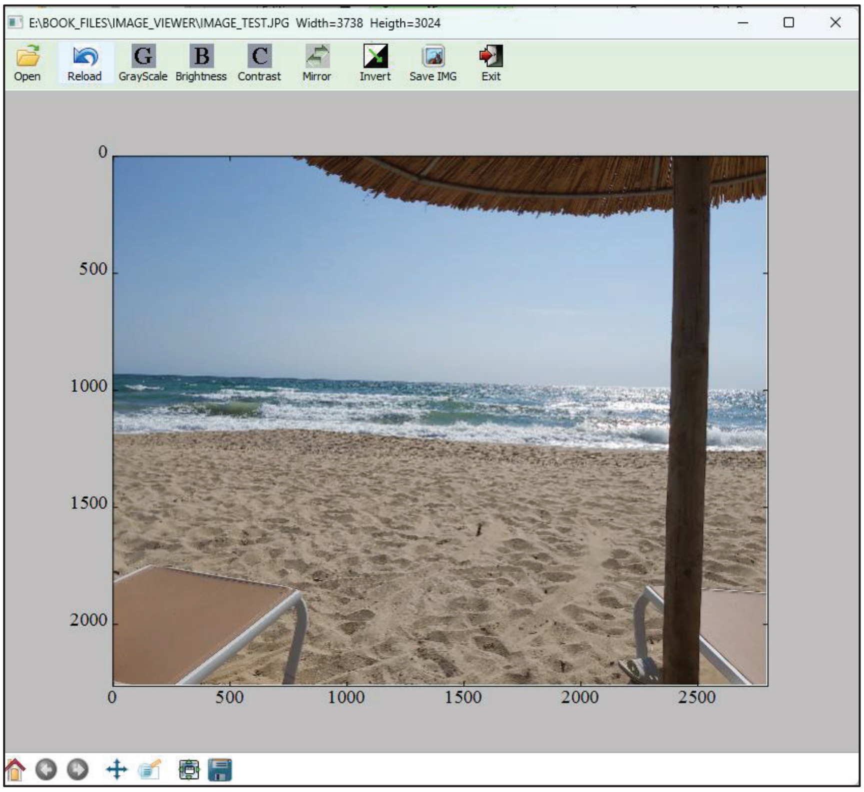
The **Image_Viewer.pyw** script (Python & wxPython version) include only one classe: **PictureFrame**. The script begin with importing libraries: **wx** (the cross-platform GUI toolkit for the Python language), **os** (this module provides a portable way of using operating system dependent functionality), **StringIO** (the StringIO module is used to implement basic tasks on file-like objects), **sqlite** (a C-language library that implements a small, fast, self-contained, high-reliability, full-featured), **matplotlib** (library for creating static, animated and interactive visualizations in Python) and **PIL** (adds image processing capabilities to the Python interpreter.).

The **PictureFrame** class create the interface (toolbar, grid, …), create the aplication main functionalities and include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user); create the statusbar (lines 13÷16); initializes the public variables of the class (lines 18÷22); call the **_TOOLBAR** function (line 24); create the **canvas** (line 25) and the sizer (line 26, a **wx.Sizer** control that can lay out items in a virtual grid); add **self. canvas** in sizer (line 27); add the bottom **toolbar** (line 28); center and show the frame in the display.
- **_TOOLBAR** Create the **toolbar** (line 63); set the **toolbar** icons size (line 65) and background color (line 66); create the connection and cursor using **Config.db** database (line 67); generate **Lst_label**, **Lst_icons** and **Lst_Help** lists (lines 68÷81), which store the **tooolbar** icons labels, the icon names that must be extracted from memory by calling **ExtragImageMemory** function and the associated icons help; for every line 84 associate an unique identificator ID; through **for** cycle from line 83 the **img** icon is extracted into memory (ine 85); create the bitmap of **img** memory icon and scale at the imposed **toolbar** icons size (line 86); in this cycle, for every operation, the icon is added in **toolbar** (line 87) ; lines 88÷96 connect with **Bind** statement to associated functions; line 97 generate and show the **toolbar;** line 98 close the connection with **Config.db** database.
- **ExtragImageMemory** The **ExtragImageMemory** function (34÷38) selecteaza din **Image_Viewer → Objects** an icon and create & returneaza the icon **img** from data in memory. This function is called from class **PictureFrame → _TOOLBAR** function.
- **OnOpenImageFile** Open a dialog to load an image file (line 40) and store the image name in variable **self.filename** line (45); call **Open_Image** function (line 46).
- **Open_Image** Load the selected image (line 49) and obtain the image width and heigth (line 50); set the frame title with image parameters (name, width and heigth, line 51); if the width or heigth are greather then 2800, the image is scaled to **MaxSize = 2800** (lines 54÷59); show the scaled image (line 60) and redraw the figure (line 61).
- **add_toolbar** Create **self.toolbar** from **NavigationToolbar2Wx**; add **self.toolbar** in sizer at the bottom of the frame, **Figure 17.1** (line 104).
- **Reload** Show original image (line 108); redraw the figure (line 109) and set variables **contrast** and **Brightness** to the initial values (lines 110÷111).
- **OnSaveIMG** Open a dialog to save the image as JPG, JPEG, PNG or BMP file.
- **OnContrast** Modify the image contrast (line 128), increase the **contrast** variable value (line 129), display the modified image (line 131) and redraw the figure (line 132).
- **OnGrayScale** Convert the image to grayscale (line 135), display the converted image (line 136) and

redraw the figure (line 137).

- **OnBrightness** Modify the image brightness (lines 141÷142), increase the **Brightness** variable value (line 143), display the modified image (line 145) and redraw the figure (line 146).
- **OnMirror** Mirror the image (line 149), display the mirrored image (line 150) and redraw the figure (line 151).
- **OnInvert** Invert the image colors (line 154), display the inverted image (line 155) and redraw the figure (line 156).
- **On_Exit** Exit from application.



E:\BOOK_FILES\IMAGE_VIEWER\IMAGE_TEST.JPG Width=3738 Heigth=3024

Figure 17.1 The interface of **Image_Viewer.pyw** script

Listing 17.1 displays the **Python & wxPython** version of script, **Image_Viewer.pyw** and **Listing 17.2** displays the **Python & Streamlit** version of the same script, **Image_Viewer.py**.

Listing 17.1 Image_Viewer.pyw – Image Viewer – Matplotlib & OpenCV – Python & wxPython version

```
1.  import wx, os
2.  import StringIO
3.  from pysqlite2 import dbapi2 as sqlite
4.  from matplotlib.backends.backend_wx import FigureCanvasWx as FigureCanvas
5.  from matplotlib.backends.backend_wx import NavigationToolbar2Wx
6.  from matplotlib.figure import Figure
7.  from PIL import Image, ImageFilter, ImageChops, ImageEnhance, ImageOps
8.  
9.  class PictureFrame(wx.Frame):
10.     def __init__(self):
11.         wx.Frame.__init__(self, None, -1)
12. 
13.         self.statusbar = self.CreateStatusBar()
14.         self.statusbar.SetFieldsCount(1)
15.         self.statusbar.SetStatusWidths([-10])
16.         self.statusbar.SetStatusText("",0)
17. 
18.         self.filename = ""
19.         self.figure = Figure()
20.         self.axes = self.figure.add_subplot(111)
21.         self.contrast=2
22.         self.Brightness=1.1
23. 
```

Listing 17.1 Image_Viewer.pyw – Image Viewer – Matplotlib & OpenCV – Python & wxPython version

```
        self._TOOLBAR()
        self.canvas = FigureCanvas(self, -1, self.figure)
        self.sizer = wx.BoxSizer(wx.VERTICAL)
        self.sizer.Add(self.canvas, 1, wx.LEFT | wx.TOP | wx.EXPAND)
        self.add_toolbar()
        self.SetSizer(self.sizer)
        self.SetSize(wx.Size(800, 800))
        self.Center()
        self.Show()

    def ExtragImageMemory(self, cursor, NumeImage):
        cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
        blob=cursor.fetchone()[0]
        img = wx.ImageFromStream(StringIO.StringIO(blob))
        return img

    def OnOpenImageFile(self, event):
        wildcard="Images files (*.jpg)|*.jpg|Images files (*.png)|*.png"
        dialog = wx.FileDialog(None, "Open Image file", "", "", wildcard, wx.OPEN)
        if dialog.ShowModal() == wx.ID_CANCEL:
            return
        self.filename = dialog.GetPath().strip().upper()
        self.Open_Image()

    def Open_Image(self):
        self.img = Image.open(self.filename)
        img_w, img_h = self.img.size
        self.Title=self.filename+"  Width="+str(img_w)+"  Heigth="+str(img_h)
        MaxSize = 2800
        self.Scalare = ""
        if img_w>MaxSize:
            self.img = self.img.resize((MaxSize,  img_h*MaxSize/img_w))
            self.Scalare="Imagine scalata la: "+str(MaxSize)+"/"+str(img_h*MaxSize/img_w)+" pixeli"
        elif img_h>MaxSize:
            self.img = self.img.resize((img_w3800/img_h,  MaxSize))
            self.Scalare="Imagine scalata la: "+str(img_w3800/img_h)+"/"+str(MaxSize)+" pixeli"
        self.axes.imshow(self.img)
        self.axes.figure.canvas.draw()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        ToolBitmapSize = 24
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))
        connection = sqlite.connect(os.getcwd()+'\Config.db') ; cursor = connection.cursor()
        Lst_label=[ "Open", "Reload", "GrayScale", "Brightness", "Contrast",
            "Mirror", "Invert", "Save IMG", "Exit"]
        Lst_icons=[ "open.png", "Back.jpg", "GrayScale.png", "Brightness.png",
            "Contrast.png", "Mirror.jpg", "Invert.png","Save.png","Exit.png"]
        Lst_Help=["Open an image file","Reload the original image",
            "Convert image into different shades of gray.",
            "Modify the brilliance of an image.\nPossible values between 1.1  \
                and 1.9 generated by successive clicks on the icon.",
            "Modify the range of brightness, from lightest to darkest in image. \n\
                With magnification, colors are more vivid.\n \
                Possible values between 2 and 5 generated by successive clicks on the icon.",
            "Mirror image: turn the image left and right (horizontally).",
            "Negative-positive color inversion of an image (reversing pixel values).",
            "Save current image.",  "Exit"]

        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            ID_tool=i # wx.NewId()
            img = self.ExtragImageMemory(cursor, icon)
            sel_bmp=wx.BitmapFromImage(img.Scale(ToolBitmapSize,ToolBitmapSize))
            self.toolbar.AddLabelTool(ID_tool, label, sel_bmp, shortHelp=HELP)
            if label=="Open": self.Bind(wx.EVT_TOOL, self.OnOpenImageFile, id=ID_tool)
            if label=="Reload": self.Bind(wx.EVT_TOOL, self.Reload, id=ID_tool)
            if label=="GrayScale": self.Bind(wx.EVT_TOOL, self.OnGrayScale, id=ID_tool)
            if label=="Brightness": self.Bind(wx.EVT_TOOL, self.OnBrightness, id=ID_tool)
            if label=="Contrast": self.Bind(wx.EVT_TOOL, self.OnContrast, id=ID_tool)
            if label=="Mirror": self.Bind(wx.EVT_TOOL, self.OnMirror, id=ID_tool)
            if label=="Invert": self.Bind(wx.EVT_TOOL, self.OnInvert, id=ID_tool)
            if label=="Save IMG": self.Bind(wx.EVT_TOOL, self.OnSaveIMG, id=ID_tool)
```

Listing 17.1 Image_Viewer.pyw – Image Viewer – Matplotlib & OpenCV – Python & wxPython version

```
            if label=="Exit": self.Bind(wx.EVT_TOOL, self.On_Exit, id=ID_tool)
        self.toolbar.Realize()  ;           self.toolbar.Show()
        cursor.close() ; connection.close()

    def add_toolbar(self):
        self.toolbar = NavigationToolbar2Wx(self.canvas)
        self.toolbar.Realize()
        # Adding 'self.toolbar' in sizer at the bottom of the frame
        self.sizer.Add(self.toolbar, 0, wx.LEFT | wx.EXPAND)
        self.toolbar.update()   # Update the axes menu on the toolbar

    def Reload(self, event):
        self.axes.imshow(self.img )
        self.axes.figure.canvas.draw()
        self.contrast=2
        self.Brightness=1.1

    def OnSaveIMG(self, event):
        wildcard = "JPG file(*.jpg)|*.jpg|"    \
                   "JPEG file (*.jpeg)|*.jpeg|" \
                   "PNG file (*.png)|*.png|"   \
                   "BMP file (*.bmp)|*.bmp"
        dlg = wx.FileDialog(
          self, message="Save file as ...", defaultDir=os.getcwd(),
          defaultFile="", wildcard=wildcard, style=wx.SAVE)
        if dlg.ShowModal() == wx.ID_OK:
            path = dlg.GetPath()
            self.figure.savefig(path)
        dlg.Destroy()

    def OnContrast(self, event):   # Enhance contrast
        self.statusbar.SetStatusText("Contrast = "+str(self.contrast),0)
        img1 = ImageEnhance.Contrast(self.img ).enhance(self.contrast)
        self.contrast+=1
        if self.contrast>5: self.contrast=2
        self.axes.imshow(img1)
        self.axes.figure.canvas.draw()

    def OnGrayScale(self, event):  # GrayScale
        img1 = self.img.convert('LA')
        self.axes.imshow(img1)
        self.axes.figure.canvas.draw()

    def OnBrightness(self, event): # Brightness
        self.statusbar.SetStatusText("Brightness = "+str(self.Brightness),0)
        enhancer  = ImageEnhance.Brightness(self.img )
        img1 = enhancer.enhance(self.Brightness)
        self.Brightness+=0.2
        if self.Brightness>1.9: self.Brightness=1.1
        self.axes.imshow(img1)
        self.axes.figure.canvas.draw()

    def OnMirror(self, event):    # Mirroring
        img1 = ImageOps.mirror(self.img )
        self.axes.imshow(img1)
        self.axes.figure.canvas.draw()

    def OnInvert(self, event):    # Invert colors
        img1 = ImageChops.invert(self.img )
        self.axes.imshow(img1)
        self.axes.figure.canvas.draw()

    def On_Exit(self, event):
      self.Destroy()

app = wx.App(1)
frame = PictureFrame()
app.MainLoop()
```

The **Image_Viewer.py** script (Python & Streamlit version) contains 52 lines and begin with importing libraries: **streamlit** (create a great interactive web application with Python), **PIL** (Python Imaging Library), **numpy** (the fundamental package for scientific computing with Python), **pathlib** (provide classes to represent abstract/concrete paths) and **cv2** (a biggest computer vision library).

Lines 9÷14 configures the title settings of the script. Lines 16 define the the script path. Line 20 wait to select an file image by the user. Line 21 store the image name into **NAME** variable. Line 23 create two columns on the same row: one for the original image and one for the future modified image. Initially, the original image is placed in both columns (lines 24÷26). Line 28 create a form with options to modify the image based on the options from lines 29÷30: **Grayscale**,**Vertical Mirror**, **Horizontal Mirror**, **Edge detection** and **Invert**; also, a **submit_button** is created (line 31).

If the button is clicked, the current image path is stored in **IMAGE_NAME** variable (line 34), the future modified image path is stored in **MODIFIED_NAME** variable (line 35) and the original image is loaded in **IMAGE** variable.

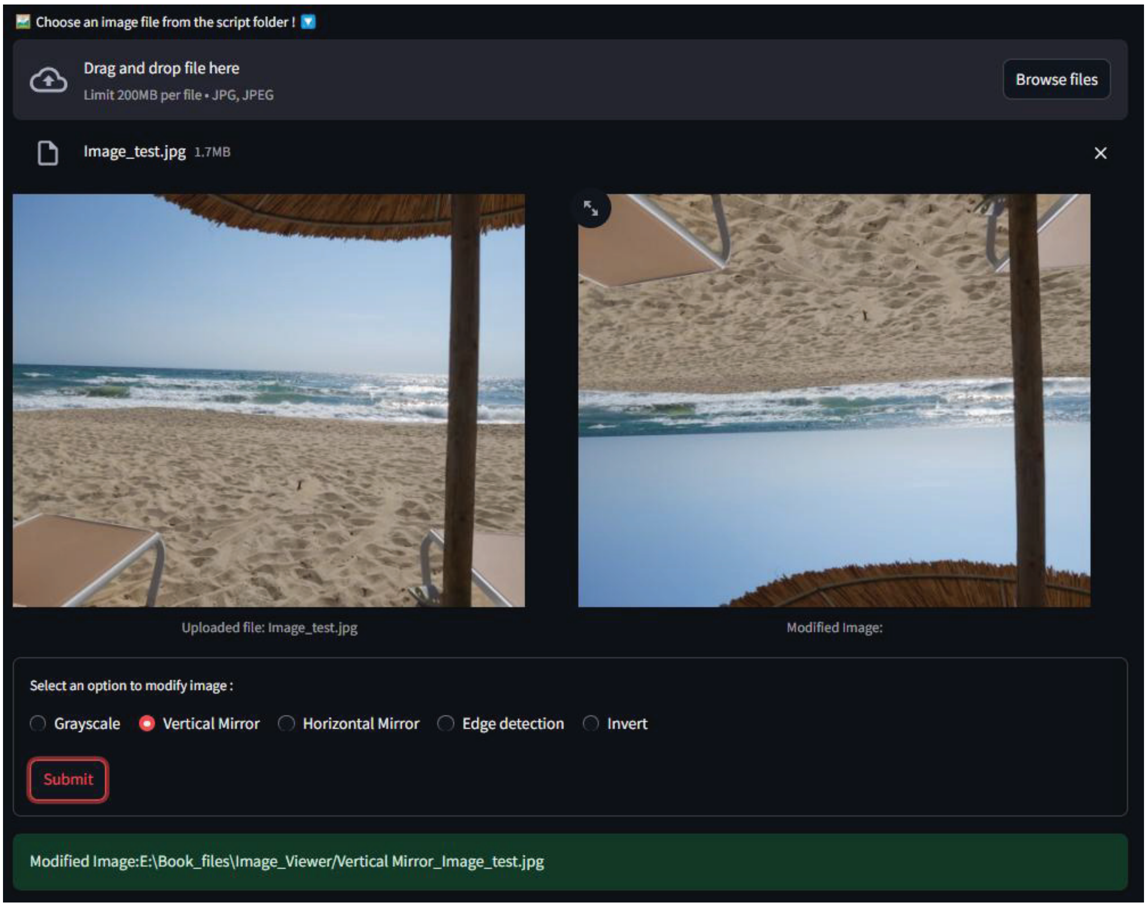
The image is modified based on option selected in the previous form (lines 38÷48); line 49 write the modified image as file on disk. Line 50 empty the **Place_Image** area and line 51 load the modified image from the disk. A success message is displayed by line 52, **Figure 17.2**.

Listing 17.2 Image_Viewer.py – Image Viewer – Matplotlib & OpenCV – Python & Streamlit version

```
1.  import streamlit as st
2.  from PIL import Image
3.  from PIL.ImageFilter import *
4.  import numpy as np
5.  from pathlib import Path
6.  import cv2
7.  # Install OpenCV with command:  pip install opencv-python
8.
9.  st.set_page_config(page_title="Image Viewer")
10. css='''
11. <style>
12.     section.main > div {max-width:70rem}  # 1rem = 16px ; 70rem = 1120 px
13. </style>
14. '''
15. st.markdown(css, unsafe_allow_html=True)
16. current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()
17.
18. st.subheader(":green[Image Viewer]")
19. MSJ=":frame_with_picture: Choose an image file from the script folder ! :arrow_down_small:"
20. uploaded_file = st.file_uploader(MSJ, type="jpg")
21. if uploaded_file:
22.     NAME=uploaded_file.name
23.     col1, col2 = st.columns(2)
24.     col1.image(Image.open(NAME), width=500, caption="Uploaded file: "+NAME)
25.     Place_Image = col2.empty()
26.     Place_Image.image(Image.open(NAME), width=500, caption='Original Image ')
27.
28.     with st.form(key='columns_in_form'):
29.             lst_Select=["Grayscale","Vertical Mirror","Horizontal Mirror", "Edge detection", "Invert"]
30.             selected=st.radio('Select an option to modify image :',lst_Select,horizontal=True)
31.             submit_button = st.form_submit_button('Submit')
32.     if submit_button:
33.         # cv_version=cv2.version
34.         IMAGE_NAME = str(current_dir)+"/"+NAME
35.         MODIFIED_NAME = str(current_dir)+"/"+selected+"_"+NAME
36.         IMAGE = cv2.imread(IMAGE_NAME)
37.
38.         if selected=="Grayscale":
39.             image2 = cv2.imread(IMAGE_NAME, cv2.IMREAD_GRAYSCALE)
40.         if selected=="Vertical Mirror":
41.             image2 = cv2.flip(IMAGE, 0)
42.         if selected=="Horizontal Mirror":
43.             image2 = cv2.flip(IMAGE, 1)
44.         if selected=="Edge detection":
45.             gray_img = cv2.cvtColor(IMAGE,cv2.COLOR_BGR2GRAY)  # convert to grayscale
46.             image2 = cv2.Canny(gray_img, 50, 240)
47.         if selected=="Invert":
48.             image2 = cv2.bitwise_not(IMAGE)
49.         cv2.imwrite(MODIFIED_NAME, image2)
50.         Place_Image.empty()
51.         Place_Image.image(Image.open(MODIFIED_NAME), width=500, caption='Modified Image:')
```

Listing 17.2 Image_Viewer.py – Image Viewer – Matplotlib & OpenCV – Python & Streamlit version

```
        st.success('Modified Image:'+MODIFIED_NAME)
```



🖼 Choose an image file from the script folder ! ⬇

Drag and drop file here
Limit 200MB per file • JPG, JPEG

Image_test.jpg 1.7MB

Uploaded file: Image_test.jpg

Modified Image:

Select an option to modify image :

○ Grayscale ● Vertical Mirror ○ Horizontal Mirror ○ Edge detection ○ Invert

Modified Image:E:\Book_files\Image_Viewer/Vertical Mirror_Image_test.jpg

Figure 17.2 The interface of **Image_Viewer.py** script

18. Image warp with OpenCV

Image warping is the process of digitally manipulating an image in such a way that all of the shapes depicted in the image have been significantly distorted. It can be used to correct distorted images or for creative purposes like morphing.

The script use a database Config.db to store all icons loaded in frame Toolbar; this will avoid the existence of following icons files in the application folder:📂, 🏠, 📋, ✥ and 🚪 . The Config.db → objects store icons for the script interface of **Python 2. 7 & wxPython** application version. Initially, the icons files are stored in the Image Warp → Objects application folder. The icons files were loaded into Config.db → objects as Binary Large Object (BLOB) through the _copyimg.pyw script displayed in Listing 15.1, from where they will be used into application interface through **Python 2.7** → Image Warp.pyw script.

The Image Warp.pyw script (Python & wxPython version) include three classes: Plot, PlotNotebook & Interpolation and one public functions: ExtragImageMemory. The script starts by importing the libraries (lines 3÷15).

The ExtragImageMemory function (lines 17÷21) selecteaza din Image Warp → Objects folder an icon and create & returneaza the icon img from data in memory. This function is called from class Image_Wrap → _TOOLBAR function. The Plot class (lines 23÷34) create self.figure, self.canvas, self.toolbar and sizer entities for the class that calls it. The PlotNotebook class (lines 36÷48) creates several pages for the class that calls it, using the internal function add; through internal function ChangePagina the user can change the page. This class represents a notebook control, which manages multiple windows with associated tabs.

The Image_Wrap class create the interface, the aplication main functionalities and include the following functions:

- __init__ Define the frame (a window whose size and position can be changed by the user); create the statusbar with 6 places (lines 56÷57); line 59 create in self.plotter an instance of PlotNotebook class; line 60 create self.AXA page, were two graphics subplot self.axes1 & self.axes2 will be created side by side (lines 62, 65); both will be connected with Bind statement to OnMouseMotion class internal function (lines 64, 66); self.axes1 will also be connected with Bind statement to OnClick class internal function (line 63); lines 68÷72 initialize the publi8c variables of the class; self.POINTS list variable memorize the point coordinates clicked on self.axes1; these four points define the area to be deformed from the original image; self.CIRCLES_ARTIST list variable memorize the circles entities placed on self.axes1 to highlight the coordinates of the selected points; line 74 call Recreate_Axis class internal function; line 75 call _TOOLBAR function; lines 76÷77 center and show the frame in the display.
- _TOOLBAR Create the toolbar (line 93÷94); set the toolbar icons size (line 95) and background color (line 96); create the connection and cursor using Config.db database (line 98); generate Lst_label, Lst_icons and Lst_Help lists (lines 99÷102), which store the tooolbar icons labels, the icon names that must be extracted from memory by calling ExtragImageMemory function and the associated icons help; for every icon line 104 associate an unique identificator ID; through for cycle from line 103 the img icon is extracted into memory (ine 105); create the bitmap of img memory icon and scale at the imposed toolbar icons size (line 106); in this cycle, for every operation, the icon is added in toolbar (line 107) ; lines 108÷112 connect with Bind statement to associated functions; line 113 generate and show the toolbar; line 114 close the connection with Config.db database.
- OnOpenImageFile Open a dialog to load an image file (line 118) and store the filename in variable self.filename line (121); call Recreate_Axis function (line 122); open and read image file (line 124); obtain the width and the heigth of the image by line 125; define the title and set the frame title on lines 126, 128; the original image is place on self.axes1 by line 127 and redraw this axes.
- Recreate_Axis This function clear both axes, set the grid and charts labels parameters. The function is called by __init__ and OnOpenImageFile functions; also, set info texts in places 1…4 of the statusbar.

- **OnClick** Line 161 store the mouse button pressed in **Btn_MOUSE** variable (line 161); calculate the number of the point coordinates clicked on **self.axes1**if in **LP** variable (line 163); if the right mouse button was pressed (line 165), initialize **self. POINTS** list (line 166), clear the first 5 places on statusbar (line 167), set the title of **self.axes1** & **self.axes2** (line 168,169), call **Remove_Marker** function (line170), set the X label of **self.axes2** and redraw it. If the left mouse button was pressed and on **self.axes1** (line 165) the number of the point coordinates clicked is smaller or equal with 4, lines 174÷214 are executed. To define the area subject of deformation process a number of 4 points must be defined on **self.axes1** by left click mouse. The coordinates of each point are stored in **Top_Left**, **Top_Right**, **Bottom_Left** and **Bottom_Right** variables (lines 176, 181, 186 and 191) and displayed in status bar (lines 177, 182, 187 and 192). Also, these coordinates are appended in self.POINTS list (lines 178, 183, 188 and 193). The clicked mouse point is marked with a circle through **PlaceMarker** function (lines 179, 184, 189 and 194). When all 4 points are clicked on **self.axes1**, the area marked by these points are wraped (lines 197÷203). Both images from **self.axes1** and **self.axes2** are saved in **Wrap_Image2** file (line 206). Only the wraped image from **self.axes2** is saved in **Wrap_Image1** file (line 211). If a point or more are wrong defined, a right click mouse will reinitiate the marker operation.
- **OnMouseMotion** Display in place 0 of the **StatusBar** the current values of X & Y mouse position in the chart.
- **PlaceMarker** Place a marker on **self.axes1** chart and redraw the chart.
- **Remove_Marker** Remove the markers placed on image on **self.axes1** chart (lines 153÷154) and redraw the chart (line 155); clear and redraw **self.axes2** (lines 156, 157); initialize **self.CIRCLES_ARTIST** list.
- **Home** Redraw the charts to fit in the axes spaces.
- **Pan** Pan the charts.
- **Zoom** Create a rectangular zoom in the selected chart.
- **OnClose** This function exits the application.

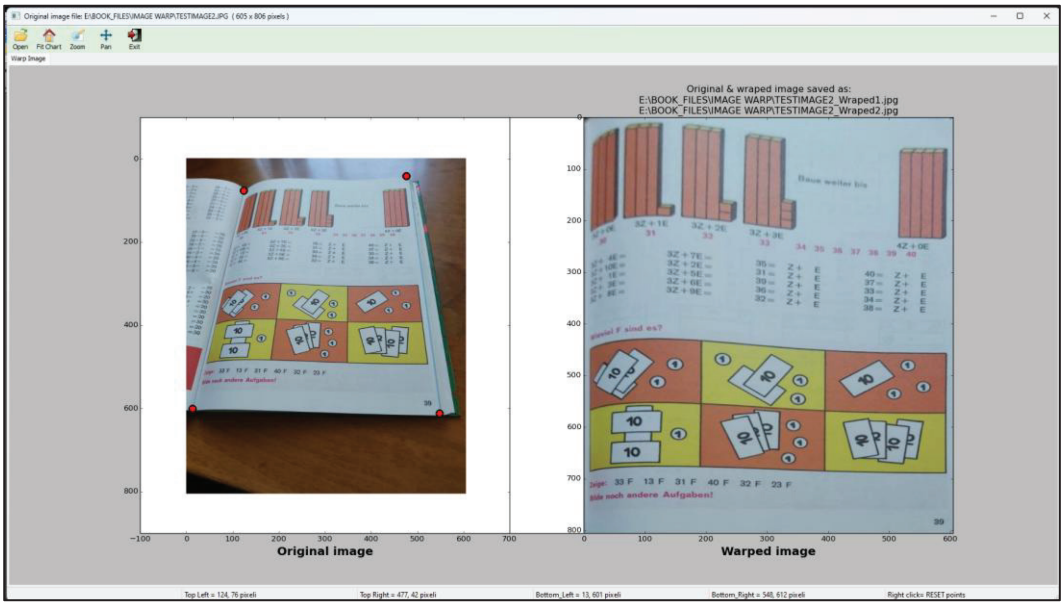
Figure 18.1 display the interface of the **Image Warp.pyw** script. The operating procedure following the next steps:

❶ Open an image file, loaded in left window part;

❷ On left part window define 4 points by left click mouse to define the regionthat will be deformed; these points is marked with circles;

❸ When all 4 points are clicked, the warp deformation is calculated and images are saved.

❹ If a new region must be deformed on the same image, a right click mouse will reinitiate the marker operation.



Original image file: E:\BOOK_FILES\IMAGE WARP\TESTIMAGE2.JPG (605 x 806 pixels)

Original & wraped image saved as:
E:\BOOK_FILES\IMAGE WARP\TESTIMAGE2_Wraped1.jpg
E:\BOOK_FILES\IMAGE WARP\TESTIMAGE2_Wraped2.jpg

Top Left = 124, 76 pixeli | Top Right = 477, 42 pixeli | Bottom_Left = 13, 601 pixeli | Bottom_Right = 548, 612 pixeli | Right click= RESET points

Figure 18.1 The interface of the **Image Warp.pyw** script

Listing 18.1 displays the **Python & wxPython** version of script, **Image Warp.pyw**.

Listing 18.1 Image Warp.pyw – Image Warp – Python & wxPython version

```
from __future__ import division

import os
import wx
import StringIO
import cv2
from sqlite3 import dbapi2 as sqlite
import numpy as np   #from numpy import array
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib.backends.backend_wxagg import FigureCanvasWxAgg as Canvas
from matplotlib.backends.backend_wxagg import NavigationToolbar2Wx as Toolbar
from matplotlib import image
import matplotlib.image as mpimg
from PIL import Image

def ExtragImageMemory(cursor, NumeImage):
        cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
        blob=cursor.fetchone()[0]
        img = wx.ImageFromStream(StringIO.StringIO(blob))
        return img

class Plot(wx.Panel):
    def __init__(self, parent, id = -1, dpi = None, **kwargs):
        wx.Panel.__init__(self, parent, id=id, **kwargs)
        self.figure = mpl.figure.Figure(dpi=dpi, figsize=(2,2))
        self.canvas = Canvas(self, -1, self.figure)
        self.toolbar = Toolbar(self.canvas)
        self.toolbar.Realize()
        self.toolbar.Hide()
        sizer = wx.BoxSizer(wx.VERTICAL)
        sizer.Add(self.canvas,1,wx.EXPAND)
        sizer.Add(self.toolbar, 0 , wx.LEFT | wx.EXPAND)
        self.SetSizer(sizer)

class PlotNotebook(wx.Panel):
    def __init__(self, parent, id = -1):
        wx.Panel.__init__(self, parent, id=id)
        self.nb = wx.Notebook(self)
        sizer = wx.BoxSizer()
        sizer.Add(self.nb, 1, wx.EXPAND)
        self.SetSizer(sizer)
    def add(self,name="plot"):
        page = Plot(self.nb)
        self.nb.AddPage(page,name)
        return page.figure
    def ChangePagina(self,INDEX):
      self.nb.ChangeSelection(INDEX)

# ==================================================================
class Image_Wrap(wx.Frame):
    def __init__(self):
        displaySize= wx.DisplaySize()
        frame=wx.Frame.__init__(self, None, -1, title="WARP Image",
                size=(displaySize[0]*0.95, displaySize[1]*0.95))
        self.statusbar = self.CreateStatusBar()  # Define statusBar
        self.statusbar.SetFieldsCount(6)

        self.plotter = PlotNotebook(self)  # Create Notebook
        self.AXA = self.plotter.add("Warp Image").gca()
        self.AXA.set_xticklabels([]) ; self.AXA.set_yticklabels([])
        self.axes1 = self.AXA.figure.add_subplot(1, 2, 1)
        self.axes1.figure.canvas.mpl_connect('button_press_event', self.OnClick)
        self.axes1.figure.canvas.mpl_connect('motion_notify_event', self.OnMouseMotion)
        self.axes2 = self.AXA.figure.add_subplot(1, 2, 2)
        self.axes2.figure.canvas.mpl_connect('motion_notify_event', self.OnMouseMotion)

        self.POINTS=[]
```

Listing 18.1 Image Warp.pyw – Image Warp – Python & wxPython version

```
        self.CIRCLES_ARTIST=[]
        self.info = "Select 4 points with left mouse click in following order:\n"
        self.info = self.info + "Top_Left, Top_Right, Bottom_Left, Bottom_Right"
        self.Top_Left, self.Top_Right, self.Bottom_Left, self.Bottom_Right = [0,0],[0,0],[0,0],[0,0]

        self.Recreate_Axis()
        self._TOOLBAR()
        self.CenterOnScreen()
        self.Show()

    def Recreate_Axis(self):
        self.AXE=[self.axes1, self.axes2]
        for i, ax in enumerate(self.AXE):
            ax.cla() ; ax.grid(False)
            Xlbl="Original image"
            if i==1: Xlbl="Warped image"
            ax.set_xlabel(Xlbl, fontsize=18, fontweight='bold', color="Black")
        self.statusbar.SetStatusText("",0)
        self.statusbar.SetStatusText( "Top_Left",1)
        self.statusbar.SetStatusText( "Top_Right",2)
        self.statusbar.SetStatusText( "Bottom_Left",3)
        self.statusbar.SetStatusText( "Bottom_Right",4)

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.NO_BORDER | \
                                  wx.TB_FLAT | wx.TB_3DBUTTONS | wx.TB_TEXT) )
        self.toolbar.SetToolBitmapSize((24,24))
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))

        connection = sqlite.connect(os.getcwd()+'\Config.db') ; cursor = connection.cursor()
        Lst_label=["Open", "Fit Chart", "Zoom", "Pan", "Exit"]
        Lst_icons=["open.png", "home.png", "zoom.png", "pan.png", "exit.png"]
        Lst_Help=["Open Excel file to read coordinates", "Fit chart in window",
            "Zoom chart rectangular region","Translate chart", "Exit from application"]
        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            ID_tool=i # wx.NewId()
            img = ExtragImageMemory(cursor, icon)
            sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
            self.toolbar.AddLabelTool(ID_tool, label, sel_bmp, shortHelp=HELP)
            if label=="Open": self.Bind(wx.EVT_TOOL, self.OnOpenImageFile, id=ID_tool)
            if label=="Fit Chart": self.Bind(wx.EVT_TOOL, self.Home, id=ID_tool)
            if label=="Zoom": self.Bind(wx.EVT_TOOL, self.Zoom, id=ID_tool)
            if label=="Pan": self.Bind(wx.EVT_TOOL, self.Pan, id=ID_tool)
            if label=="Exit": self.Bind(wx.EVT_TOOL, self.OnClose, id=ID_tool)
        self.toolbar.Realize()   ; self.toolbar.Show()
        cursor.close() ; connection.close()

    def OnOpenImageFile(self, event):
        wildcard="Images files (*.jpg)|*.jpg|Images files (*.png)|*.png"
        dialog = wx.FileDialog(None, "Open Image file", "", "", wildcard, wx.OPEN)
        if dialog.ShowModal() == wx.ID_CANCEL:
            return
        self.filename = dialog.GetPath().strip().upper()
        self.Recreate_Axis()
        # image_file=dialog.GetDirectory()+'\\'+dialog.GetFilename()
        image = plt.imread(self.filename)
        self.height, self.width, _ = image.shape
        self.Title="Original image file: " + str(self.filename ) + "  ( "+str(self.width)+" x "+str(self.height)+" pixels )"
        self.axes1.imshow(image, aspect='auto')
        self.axes1.set_title(self.info)
        self.axes1.figure.canvas.draw()

    def Home(self,event):
        self.axes1.figure.canvas.toolbar.home()

    def Pan(self,event):
        self.axes1.figure.canvas.toolbar.pan()

    def Zoom(self,event):
        self.axes1.figure.canvas.toolbar.zoom()

    def OnMouseMotion(self, event):
```

Listing 18.1 Image Warp.pyw – Image Warp – Python & wxPython version

```
        xMouse,yMouse = event.xdata, event.ydata
        if xMouse<>None and yMouse<> None:
            SirReal="X point="+'%0.4f' % xMouse + ",  Y point="+'%0.4f' % yMouse
            self.statusbar.SetStatusText( SirReal,0)

    def PlaceMarker(self, Xmouse, Ymouse):  # Place  marker on image
        MARKER,  = self.axes1.plot(Xmouse, Ymouse, 'o', markersize=11,  \
            mfc='Red', markeredgecolor="Black", markeredgewidth=2)
        self.CIRCLES_ARTIST.append(MARKER, )
        self.axes1.figure.canvas.draw()

    def Remove_Marker(self):   # Remove markers placed on image
        for i in range(len(self.CIRCLES_ARTIST)):
            punct=self.CIRCLES_ARTIST[i]  ;  punct.remove()
        self.axes1.figure.canvas.draw()
        self.axes2.cla()
        self.axes2.figure.canvas.draw()
        self.CIRCLES_ARTIST=[]

    def OnClick(self, event):
        Btn_MOUSE=event.button  ;  X_MOUSE, Y_MOUSE=event.xdata, event.ydata
        XM=int(X_MOUSE) ; YM=int(Y_MOUSE)  ;  MSJ=str(XM)+", "+str(YM)+" pixeli"
        LP=len(self.POINTS)

        if Btn_MOUSE==3: # Right mouse
            self.POINTS=[]
            for i in range(5): self.statusbar.SetStatusText("", i)
            self.axes1.set_title(self.info)
            self.axes2.set_title("")
            self.Remove_Marker()
            self.axes2.set_xlabel("Warped image", fontsize=18, fontweight='bold', color="Black")
            self.axes2.figure.canvas.draw()
        if Btn_MOUSE==1 and LP<=4:   # Left mouse
            self.statusbar.SetStatusText("Right click= RESET points", 5)
            if LP==0:
                self.Top_Left = [X_MOUSE, Y_MOUSE]
                self.statusbar.SetStatusText("Top Left = "+MSJ, 1)
                self.POINTS.append(self.Top_Left)
                self.PlaceMarker(XM,YM)
            elif LP==1:
                self.Top_Right =  [X_MOUSE, Y_MOUSE]
                self.statusbar.SetStatusText("Top Right = "+MSJ, 2)
                self.POINTS.append(self.Top_Right)
                self.PlaceMarker(XM,YM)
            elif LP==2:
                self.Bottom_Left = [X_MOUSE, Y_MOUSE]
                self.statusbar.SetStatusText("Bottom_Left = "+MSJ, 3)
                self.POINTS.append(self.Bottom_Left)
                self.PlaceMarker(XM,YM)
            elif LP==3:
                self.Bottom_Right = [X_MOUSE, Y_MOUSE]
                self.statusbar.SetStatusText("Bottom_Right = "+MSJ, 4)
                self.POINTS.append(self.Bottom_Right)
                self.PlaceMarker(XM,YM)

                # Warp image by Open CV
                Top_Left= self.POINTS[0] ; Top_Right = self.POINTS[1]
                Bottom_Left = self.POINTS[2]  ;  Bottom_Right= self.POINTS[3]
                point_matrix = np.float32([Top_Left,Top_Right,Bottom_Left, Bottom_Right])
                converted_points = np.float32([[0,0], [self.width,0], [0,self.height],[self.width,self.height]])
                perspective_transform = cv2.getPerspectiveTransform(point_matrix,converted_points)
                img_WARP= plt.imread(self.filename)
                img_Output = cv2.warpPerspective(img_WARP ,perspective_transform,(self.width,self.height))
                Wrap_Image1 = self.filename[:-4]+"_Wraped1.jpg"
                Wrap_Image2 = self.filename[:-4]+"_Wraped2.jpg"
                cv2. imwrite(Wrap_Image2, img_Output)
                # cv2.imshow("Output Image", img_Output) ; cv2.waitKey(0)

                self.axes1.set_title("")
                self.axes2.imshow(img_Output, aspect='auto')
                fig= self.axes1.figure.savefig(Wrap_Image1)
                self.axes2.set_title("Original & wraped image saved as:\n"+Wrap_Image1+"\n"+Wrap_Image2)
```

Listing 18.1 Image Warp.pyw – Image Warp – Python & wxPython version

```
                self.axes1.figure.canvas.draw()
                self.axes2.figure.canvas.draw()

    def OnClose(self, event):
        self.Close(True)
        self.Destroy()

#--------------------------------
if __name__ == '__main__':
    app = wx.App(redirect=0)
    Image_Wrap()
    app.MainLoop()
```

19. Detect faces & eyes with Open CV

Open CV is a complex library for building computer vision applications, running algorithms in real time, highly optimised and available on many platforms.

The **Detect_faces_eyes.py** script is an application to detect faces & eyes with Open CV. The **Detect_faces_eyes.py**script (Python & Streamlit version) contains 76 lines and begin with importing libraries (lines 1÷5). Lines 7÷12 configures the title settings of the script and the max-width of the page (70 rem = 1120 px, where 1 rem = 16px). Lines 14 define the the script path.

Lines 16÷33 define **Detect_FACES** function which detect faces on a image. This function return the name of the image saved with detected faces and the original image width & height scaled for maximum 1000 pixels for width. Lines 35÷53 define **Detect_EYES** function which detect eyes on a image. This function return the name of the image saved with detected eyes.

Line 55 write the script title. Line 57 wait to select an Excel file by the user to read an image file, **Figure 19.1**. Lines 60, 61 call **Detect_FACES** and **Detect_EYES** functions on the original images.

Lines 63÷64 place the selected (original) image in **PlaceImage1**.

Lines 65 create a form to select a value from three options, **Figure 19.2**: view only faces image, view only eyes image or both. Also, a **submit_button** is created (line 68). If the button is clicked, lines 73 and 76 place images in **PlaceImage2** or **PlaceImage3**.

Listing 19.1 displays the **Python & Streamlit** version of script **Detect_faces_eyes.py**.

Listing 19.1 Detect_faces_eyes.py – Detect faces & eyes with Open CV – Python & Streamlit version

```
1.  import streamlit as st
2.  from PIL import Image
3.  import numpy as np
4.  from pathlib import Path
5.  import cv2
6.
7.  st.set_page_config(page_title=" Detect faces & eyes ")
8.  css='''
9.  <style>
10.     section.main > div {max-width:70rem}  # 1rem = 16px ; 70rem = 1120 px
11. </style>
12. '''
13. st.markdown(css, unsafe_allow_html=True)
14. current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()
15.
16. def Detect_FACES(NAME):
17.     img = cv2.imread(NAME)
18.     H,W,x = img.shape
19.     ImageWidth = W ; Imageheight = H
20.     if W>1000:
21.         ImageWidth=1000
22.         Imageheight = int(H*ImageWidth/W)
23.     gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
24.     face_classifier = cv2.CascadeClassifier(
25.             cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
26.     faces = face_classifier.detectMultiScale(
27.             gray_image, scaleFactor=1.05, minNeighbors=5, minSize=(50, 50) )
28.     for (x, y, w, h) in faces:
29.             cv2.rectangle(img, (x, y), (x + w, y + h), (0, 255, 0), 3)
30.     img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
31.     FileName = NAME[:-4]+"_Faces" + NAME[-4:]
32.     cv2.imwrite(FileName, img_rgb)
33.     return [FileName, ImageWidth, Imageheight]
34.
35. def Detect_EYES(NAME):
36.     img = cv2.imread(NAME)
37.     gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
38.     face_classifier = cv2.CascadeClassifier(
39.                 cv2.data.haarcascades + "haarcascade_frontalface_default.xml")
40.     eye_cascade = cv2.CascadeClassifier(
41.                 cv2.data.haarcascades + "haarcascade_eye_tree_eyeglasses.xml")
42.     faces = face_classifier.detectMultiScale(
43.                 gray_image, scaleFactor=1.5, minNeighbors=5, minSize=(50, 50) )
44.     for (x, y, w, h) in faces:
```

Listing 19.1 Detect_faces_eyes.py – Detect faces & eyes with Open CV – Python & Streamlit version

```
45.         face_roi = gray_image[y:y+h, x:x+w] # Get the face ROI
46.         eyes = eye_cascade.detectMultiScale(face_roi)  #  Detect eyes in the face ROI
47.         for (ex, ey, ew, eh) in eyes:  # Loop over the eyes
48.             # Draw a rectangle around the eyes
49.             cv2.rectangle(img, (x+ex, y+ey), (x+ex+ew, y+ey+eh), (0, 255, 0), 2)
50.     img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
51.     FileName = NAME[:-4]+"_Eyes" + NAME[-4:]
52.     cv2.imwrite(FileName, img_rgb)
53.     return FileName
54.
55. st.subheader(":green[Detect faces / eyes]")
56. MSJ=":frame_with_picture: Choose an image file ! :arrow_down_small:"
57. uploaded_file = st.file_uploader(MSJ, type="jpg")
58. if uploaded_file:
59.     NAME=uploaded_file.name
60.     FileName1, ImageWidth, ImageHeigth = Detect_FACES(NAME)
61.     FileName2 = Detect_EYES(NAME)
62.     PlaceImage1= st.empty()
63.     PlaceImage1.image(Image.open(NAME), width=ImageWidth, caption="Uploaded file: "+
64.               NAME+" Width/Heigth = "+str(ImageWidth)+"/"+str(ImageHeigth))
65.     with st.form(key='form'):
66.         lst_Select=["Faces","Eyes","Both"]
67.         selected=st.radio('Select an option :',lst_Select,horizontal=True)
68.         submit_button = st.form_submit_button('Submit')
69.     if submit_button:
70.         if submit_button:
71.             if selected=="Faces" or selected=="Both":
72.                 PlaceImage2= st.empty()
73.                 PlaceImage2.image(Image.open(FileName1), width=ImageWidth, caption='Detected faces on: '+FileName1)
74.             if selected=="Eyes" or selected=="Both":
75.                 PlaceImage3= st.empty()
76.                 PlaceImage3.image(Image.open(FileName2), width=ImageWidth, caption='Detected eyes on: '+FileName2)
```

One of the methods used today for face detection is using the Python program and its libraries. OpenCV is the most popular computer vision library. Originally written for C/C++, it can now also be used for Python. OpenCV uses machine learning algorithms to search for faces within an image. Because faces are so complicated, there is no single simple test that will prove whether a face has been found or not. Instead, there are thousands of small patterns and features that must match. Algorithms break down the task of facial recognition into thousands of smaller, bite-sized tasks, each easily solvable. These tasks are also called classifiers. For something like a face, there are 6000 or more classifiers, all of which must match to detect the face (within errors of course) [19.1], [19.2].

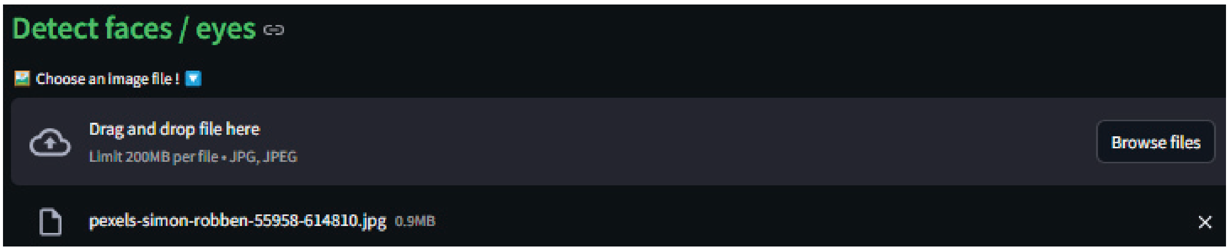
With the use of OpenCV, which aids in object detection, you can identify objects such as faces and eyes and track them in real-time using the Haar Cascade Algorithm. A Haar Cascade is a classifier that detects the object for which it has been trained [19.3]. On Google, it is possible to find a number of different hair cascades of things that can be detected by Haar Cascade algorithm.

The first step in using OpenCV for face and eye detection is to load the Haar cascade classifiers (lines 24÷25, 40÷41). The classifier is a pre-trained machine learning model that has been trained to detect faces and eyes [19.4]. The next step is to detect faces in the image using the **detectMultiScale** function of the cascade classifier. This function returns a list of rectangles representing the locations of the detected faces (lines 26÷27, 46)

Parameters of **detectMultiScale** in OpenCV using Python are [19.5], [19.6]:

- **image** – matrix containing an image where objects are detected;
- **scaleFactor** - rescaling the original image transforms a larger face into a smaller one that the algorithm can recognise; the parameter specifying how much the image size is reduced at each image scale.
- **minNeighbors** – this parameter specifying how many neighbors each candidate rectangle should have to retain it; this parameter will affect the quality of the detected faces; higher value results in fewer detections but with higher quality. 3~6 is a good value for it.
- **minSize** – minimum possible object size; objects smaller than that are ignored.
- **maxSize** – maximum possible object size; objects larger than that are ignored.

For eyes detection, line 45 define **face_roi** variable, where **roi** represents a region of interest described by an array of Boolean values; it can crop images or restrict data (such as feature localisation data) to a specified region. Lines 28, 44 loop over the detected faces and lines 29 & 49 call the rectangle function to draw rectangles around detected the faces / eyes. Figures 15.3÷15.8 exemplify the results for faces and eyes detection, based on **pexels-olly-3807758.jpg** and **pexels-simon-robben-55958-614810.jpg** images [19.7].



Detect faces / eyes

Choose an image file !

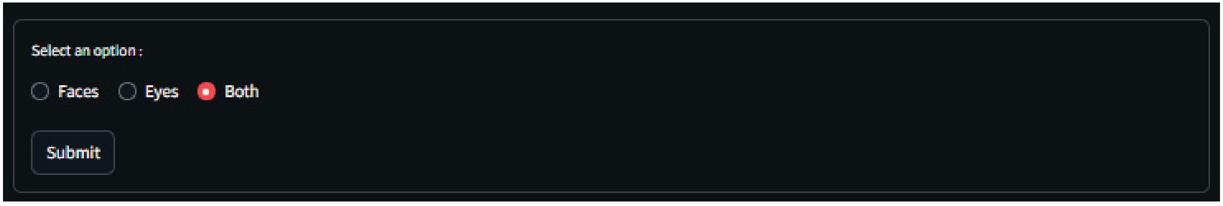
Drag and drop file here

Limit 200MB per file • JPG, JPEG

Browse files

pexels-simon-robben-55958-614810.jpg 0.9MB ✕

Figure 19.1 Select an Excel file by the user to read an image file



Select an option :

○ Faces ○ Eyes ● Both

Submit

Figure 19.2 Form to select a value from three options

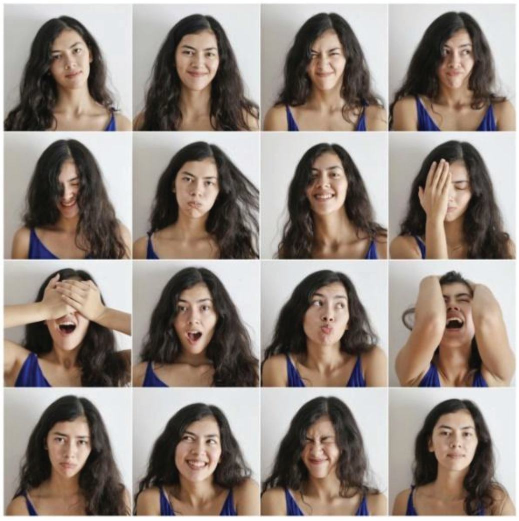


Figure 19.3 The original image

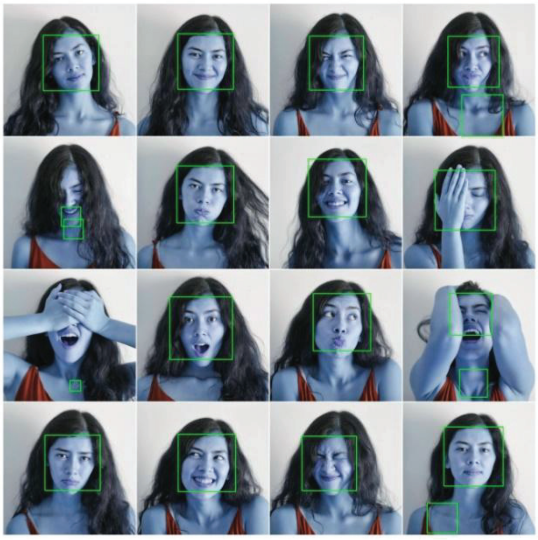


Figure 19.4 The image with detected faces

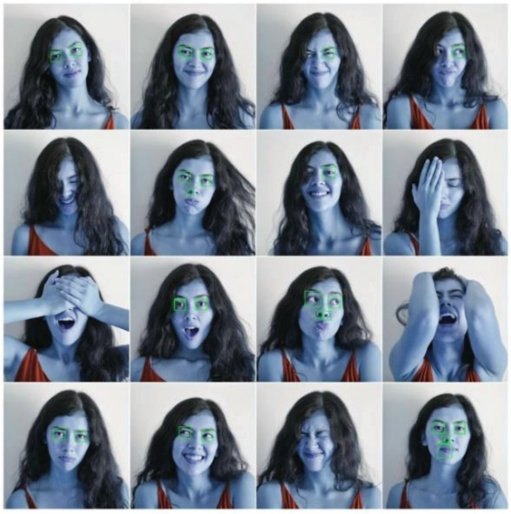


Figure 19.5 The image with detected eyes

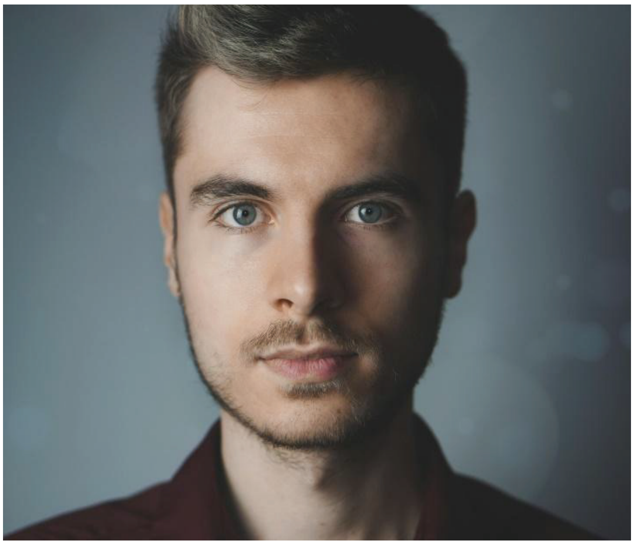


Figure 19.6 The original image

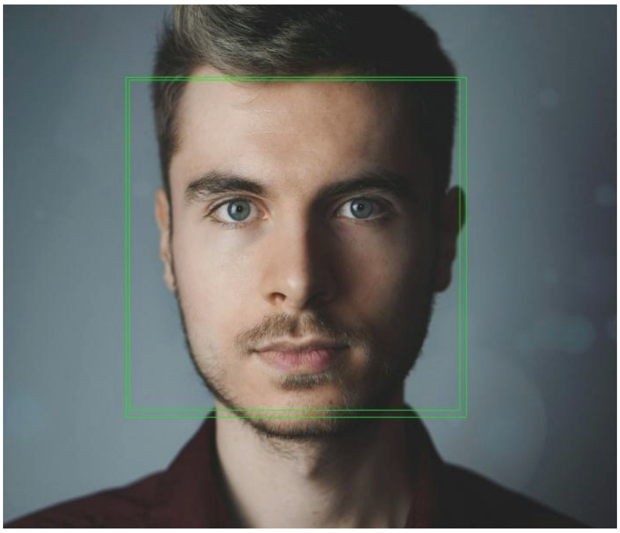


Figure 19.7 The image with detected faces

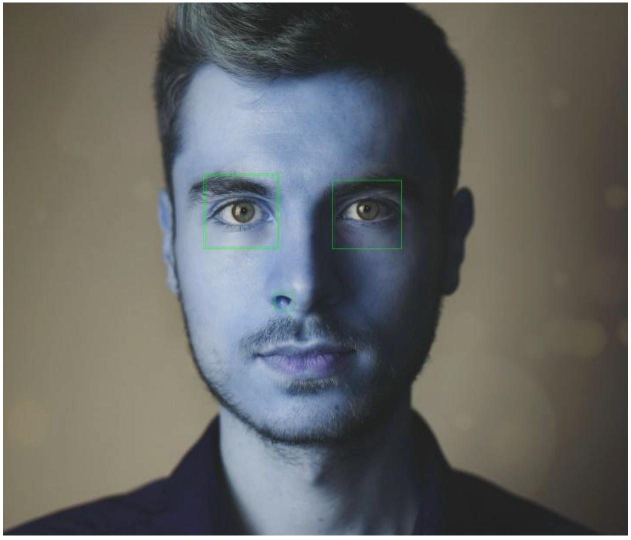


Figure 19.8 The image with detected eyes

20. PDF file operations

Pyhon has some libraries to manage PDF file operations. In this script saved in **PDF files** folder we will use reportlab [**20.1**] and pyPdf [**20.2**]. Due to space limitations and without exhausting the possibilities of working with these libraries, in this script we will limit ourselves to illustrating only the following operations, **Figure 20.1**:

❶ convert TXT or CSV files to PDF;

❷ open a PDF file;

❸ merge two or more PDF files, **Figure 20.2**.

Other possible operations with PDF files are [**20.2**]: splitting, cropping, retrieve text, add passwords or transforming the pages of PDF files.

The **Config.db** database is used to store all icons loaded in frame **Toolbar**; this will avoid the existence of following icons files in the application folder: , , , and . The **SQL** to generate **objects** table is displayed in **Figure 15.4**. The **Config.db** → **objects** memorize icons for the script interface for **Python 2.7 & wxPython** application version. Initially, the icons files are stored in the **PDF files** → **Objects** application folder. The icons files were loaded into **Config.db** → **Objects** as **Binary Large Object** (**BLOB**) through the **_copyimg.pyw** script displayed in **Listing 15.1**, from where they will be used into application interface through **Python 2.7** → **PDF operations.pyw** script.

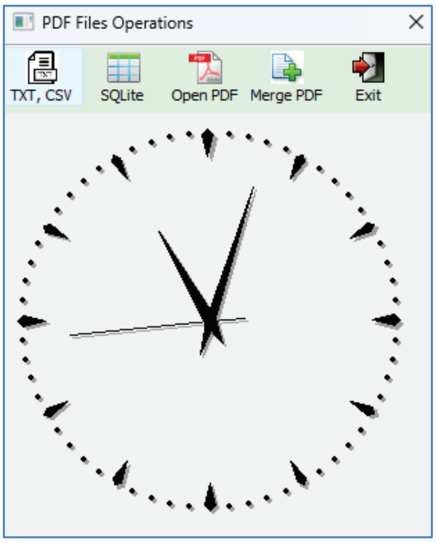
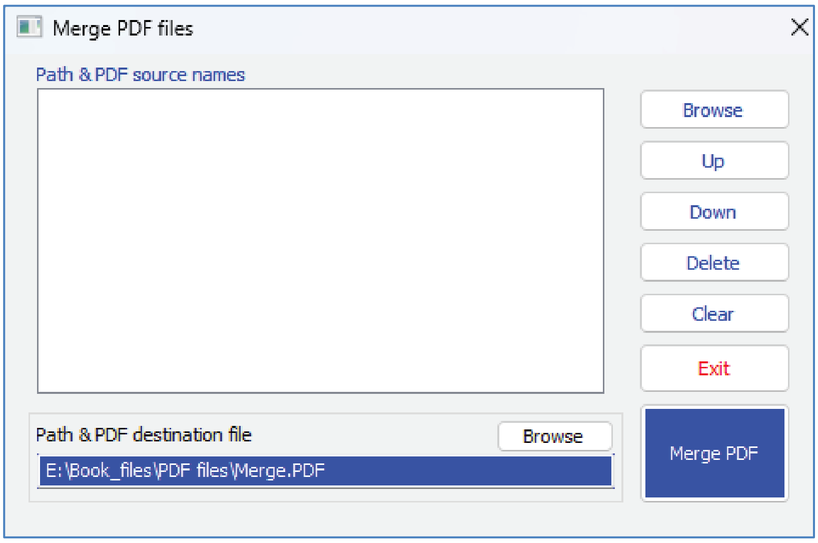


Figure 20.1 The **PDF operations.pyw** interface



E:\Book_files\PDF files\Merge.PDF

Figure 20.2 The **Merge PDF files** interface

The **PDF files.pyw** script (Python & wxPython version) include three classes: **PDF_Files**, **Merge_PDF** and **PanelPDF** and three public functions (**ExtragImageMemory, DrawPage** & **Generate_PDF**). The script begin with importing libraries (lines 3÷12).

The **DrawPage** public function (lines 14÷19) prepare canvas page for the PDF file.

The **Generate_PDF** public function (lines 21÷55) prepare canvas page for the PDF file. The functions parameter are: the selected file name and path to be converted to PDF file, font size, font type, page orientation, and data that will be converted into PDF file.

The **ExtragImageMemory** function (lines 57÷61) selecteaza din **Config.db** → **Objects** an icon si create & returneaza the icon **img** from data in memory. This function is called from class **PDF_Files** → **_TOOLBAR** function.

The **PDF_Files** class create the interface (toolbar) and include the following functions:

- **__init__** Define the **frame** and the **panel** (that will hold the contents of the frame); create the sizer (line 68), a functional visual clock (line 69) and place the clock inside the sizer (line 70); initializes the public variable of the class (line 72); call the **_TOOLBAR** and center & show the frame in the display (line 73).
- **_TOOLBAR** Create the **toolbar** (line 76); set the **toolbar** background color (line 77) and icon size (line

78); create the connection and cursor using **Config.db** database (line 79); generate **Lst_label**, **Lst_icons** and **Lst_Help** lists (lines 80÷83), which store the **tooolbar** icons labels, the icon names that must be extracted from memory by calling **ExtragImageMemory** function and the associated icons help; for every operation lines 84÷85 associate an unique identificator ID; through **for** loop from line 84 the **img** icon is extracted into memory (line 86). Line 87 create the bitmap of **img** memory icon and scale at the imposed **toolbar** icons size; in this loop, for every operation, the icon is added in **toolbar** (line 88) and connect with **Bind** statement to associated functions (lines 89÷93); line 94 generate and show the **toolbar**; line 95 close the connection with **Config.db** database.

- **OnConvert_TXT** Create a dialog to select TXT or CSV file (lines 98÷100), call the **Generate_PDF** public function (line 103) and show a info message (line 104). The data that will be converted into PDF file send to **Generate_PDF** function is a empty string because the function will read the data directly from the file.
- **OnConvert_SQLite** Create a dialog to select a **SQLite** databse file (lines 107÷109); connect with selected database file (line 112) and retrieve in **TABLES** list the tables existing in databases (line 114); initialize the **SQLite_DATA** list variable (line 115); for every table, the loop from line 116, count the records number in **RECCOUNT** variable (line 117), add the database, table name and no. of records to the list (line 118÷119) and select all table records (line 119); in for loop from line 122 all records are appended to **SQLite_DATA** list; line 126 close the connection; line 127 call the **Generate_PDF** public function and show a info message (line 128). The data that will be converted into PDF file send to **Generate_PDF** function is **SQLite_DATA** list.
- **OnOpenPDF** Create a dialog to select a **SQLite** databse file (lines 131÷132); the function try (line 134) to read the PDF file through a PDF reader software installed on Windows (line 135); if on Windows is not installed a PDF reader software lines 137÷139 are executed to create a frame where the PDf file will be displayed in **PanelPDF** class.
- **OnMergePDF** Call **MergePDF** class to merge two or more PDF files into one PDF file.
- **On_Close** This function exits the application.

The **Merge_PDF** class, **Figure 20.2**, include the following functions:

- **__init__** Define the **frame** (a window whose size and position can be changed by the user), the **panel**, the **BoxSizer** (line 182). Lines 183÷212 define the interface controls (buttons, texts) and bind the buttons to associate functions. Lines 214÷218 place these controls inside **BoxSizer**. Line 219 center and show the frame in the display.
- **OnBROWSE** This function is called by the **Browse** button located at the top of the windows. This function create a dialog (lines 222÷225) and append PDF names to **self.listbox** controls (line 229). The PDF files with names placed in **self.listbox** list will merged into one PDF file when Merge PDF button will be clicked.
- **OnUp** Move up the PDF name in **self.listbox** list.
- **OnDown** Move down the PDF name in **self.listbox** list.
- **OnDelete** Delete the PDF name from **self.listbox** list.
- **OnClear** Clear the **self.listbox** list.
- **OnCaleNumePDF** This function create a dialog (lines 233÷235) to select the path and the name of the destination PDF file were all PDF with names from **self.listbox** list will merged into one PDF file. This function is called by the **Browse** button located at the bottom of the windows.
- **OnMerge** Create the PDF file from the PDF files with names placed in **self.listbox** list. Line 282 get the name of the destination PDF file. Lines 283, 284 verify if at least two name of PDF files exists in **self.listbox** list; if this condition is not satisfied line 284 exit from function. Line 285 test if a PDF name file with those proposed name exist; if the user confirm, the process will continue with lines 291÷299 to merge PDF files; line 300÷302 write the PDF file on disk; lines 303÷307 try (line 303) to read the PDF file through a PDF reader

software installed on Windows (line 304); if on Windows is not installed a PDF reader software lines 306÷307 are executed to create a frame where the PDf file will be displayed in **PanelPDF** class.

- **OnExit** Close the window.

The **PanelPDF** class include the following functions:

- **__init__** Define the **panel** (line 149) and the **Sizer** (line 152). Line 153 create the **self.pdf** window where the PDF file content will be placed; line 154 add the **self.pdf** window to sizer. Lines 155÷162 create two buttons to call **OnPrevPageButton** or **OnNextPageButton** functions and add these buttons to the sizer (lines 157, 160).
- **OnPrevPageButton** Move to the previous page in PDF file.
- **OnNextPageButton** Move to the next page in PDF file.

Listing 20.1 displays the **Python & wxPython** version of script, **PDF operations.pyw**.

Listing 20.1 PDF operations.pyw – PDF file operations – Python & wxPython version

```
706.  from __future__ import division
707.
708.  from pysqlite2 import dbapi2 as sqlite
709.  import StringIO,cStringIO
710.  import wx, sys, os, math
711.  from wx.lib.pdfwin import PDFWindow
712.  import wx.lib.analogclock as ac
713.  from reportlab.pdfgen import canvas
714.  from reportlab.lib import units, pagesizes
715.  from reportlab.lib.units import inch, cm
716.  import pyPdf
717.  import win32api
718.
719.  def DrawPage(canv, font_SIZE, font_TYPE, TOP, BOTTOM, LEFT, RIGHT, pageWidth, pageHeight, TITLE, HEADER):
720.      canv.setFont(font_TYPE, font_SIZE)
721.      canv.drawCentredString(0.5 * pageWidth, TOP +25, TITLE)
722.      canv.line(LEFT, TOP+20, RIGHT, TOP+20) ; canv.drawString(LEFT, TOP + 6, HEADER)
723.      canv.line(LEFT, TOP, RIGHT, TOP) ; canv.line(LEFT, BOTTOM+7, RIGHT, BOTTOM+7)
724.      canv.drawCentredString(0.5*pageWidth, 0.5 * inch,"Pag. %d" % canv.getPageNumber())
725.
726.  def Generate_PDF(file_SOURCE, TITLE, font_SIZE, font_TYPE, format_PAGINA, DATA):
727.      if format_PAGINA=='Portrait': pageWidth, pageHeight = pagesizes.A4
728.      if format_PAGINA=='Landscape': pageWidth, pageHeight = pagesizes.landscape(pagesizes.A4)
729.      file_PDF = file_SOURCE[:-3]+"pdf"
730.      if DATA<>"":
731.          data=DATA
732.          file_PDF = file_SOURCE[:-2]+"pdf"
733.      bordura=0.65
734.      TOP = pageHeight- 0.75*inch  ; RIGHT = pageWidth - bordura*inch
735.      BOTTOM = bordura*inch ; LEFT = bordura*inch
736.      canv = canvas.Canvas(file_PDF, pagesize=(pageWidth,pageHeight))
737.      canv.setPageCompression(0)
738.      DrawPage(canv, font_SIZE, font_TYPE, TOP, BOTTOM, LEFT,
739.              RIGHT, pageWidth, pageHeight, TITLE, file_PDF)
740.      canv.setFont(font_TYPE, font_SIZE)
741.      tx = canv.beginText(LEFT, TOP - 0.15*inch)
742.      if DATA=="":
743.          data = open(file_SOURCE, 'r').readlines()
744.      for i in xrange(len(data)):  # Removing all linefeeds
745.          if data[i][-1]=='\n':  # if data[i][-1]=='\012':
746.              data[i] = data[i][:-1]
747.      for line in data:
748.          tx.textLine(line)
749.          y = tx.getY()
750.          if y < BOTTOM + 0.15*inch:
751.              canv.drawText(tx) ; canv.showPage()
752.              DrawPage(canv, font_SIZE, font_TYPE, TOP, BOTTOM, LEFT,  \
753.                  RIGHT, pageWidth, pageHeight, TITLE, file_PDF)
754.              canv.setFont(font_TYPE, font_SIZE)
755.              tx = canv.beginText(LEFT, TOP - 0.15*inch) ; pg = canv.getPageNumber()
756.      if tx:
```

Listing 20.1 PDF operations.pyw – PDF file operations – Python & wxPython version

```
        canv.drawText(tx) ; canv.showPage()
        DrawPage(canv, font_SIZE, font_TYPE, TOP, BOTTOM, LEFT,  \
                    RIGHT, pageWidth, pageHeight, TITLE, file_PDF)
    canv.save()

def ExtragImageMemory(cursor, NumeImage):
    cursor.execute("SELECT fisier FROM objects where nume='"+NumeImage+"'")
    blob=cursor.fetchone()[0]
    img = wx.ImageFromStream(StringIO.StringIO(blob))
    return img

class PDF_Files(wx.Frame):  # ==============================================
    def __init__(self):
        wx.Frame.__init__(self, None, -1, title="PDF Files Operations", size=(330,400),
            style=wx.DEFAULT_FRAME_STYLE ^ ( wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
        panel = wx.Panel(self,-1,)
        sizer= wx.GridBagSizer(hgap=10, vgap=40)
        ceas = ac.AnalogClock(panel,size=(280,280), pos=(10,10))
        sizer.Add(ceas,pos=(1,1) )

        self.CONFIG_DB=os.getcwd()+'\Config.db'
        self._TOOLBAR() ; self.CenterOnScreen()  ;  self.Show()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour(wx.Colour(224,238,224))
        self.toolbar.SetToolBitmapSize((24,24))
        conn = sqlite.connect(self.CONFIG_DB) ; cursor = conn.cursor()
        Lst_label=["TXT, CSV","SQLite","Open PDF","Merge PDF","Exit"]
        Lst_icons=["TXT.png","table.png","PDF.png", "Plus.png","exit.png"]
        Lst_Help=["Convert TXT or CSV file to PDF file","Convert SQLite table to PDF file",
                    "Open PDF file","Merge PDF files", "Quit application"]
        for i, ( label, icon, HELP) in enumerate(zip(Lst_label, Lst_icons, Lst_Help )):
            ID_tool= wx.NewId()
            img = ExtragImageMemory(cursor, icon)
            sel_bmp=wx.BitmapFromImage(img.Scale(24,24))
            self.toolbar.AddLabelTool(ID_tool, label, sel_bmp, shortHelp=HELP)
            if label=="TXT, CSV": self.Bind(wx.EVT_TOOL, self.OnConvert_TXT, id=ID_tool)
            if label=="SQLite": self.Bind(wx.EVT_TOOL, self.OnConvert_SQLite, id=ID_tool)
            if label=="Open PDF": self.Bind(wx.EVT_TOOL, self.OnOpenPDF, id=ID_tool)
            if label=="Merge PDF": self.Bind(wx.EVT_TOOL, self.OnMergePDF, id=ID_tool)
            if label=="Exit": self.Bind(wx.EVT_TOOL, self.OnClose, id=ID_tool)
        self.toolbar.Realize() ; self.toolbar.Show()  # Generate toolbar
        cursor.close() ; conn.close()

    def OnConvert_TXT(self, event):
        wildcard="All files (*.*)|*.*|TXT files (*.txt)|*.txt|CSV files (*.csv)|*.csv"
        dlg = wx.FileDialog(self, "Select TXT or CSV file", os.getcwd(), "",wildcard , wx.OPEN)
        result = dlg.ShowModal() ; dlg.Destroy()
        if result == wx.ID_OK:
            PATH_File = dlg.GetDirectory()+'\\'+dlg.GetFilename() ; NAME_File=dlg.GetFilename()
            Generate_PDF(NAME_File, PATH_File, 11, 'Courier', 'Portrait', "")
            wx.MessageBox("File:\n\n"+PATH_File+"\n\nconverted to:\n\n"+os.getcwd()+"\\"+NAME_File[:-3]+"pdf")

    def OnConvert_SQLite(self, event):
        wildcard="SQLite database (*.db)|*.db"
        dlg = wx.FileDialog(self, "Select SQLite database file", os.getcwd(), "",wildcard , wx.OPEN)
        result = dlg.ShowModal() ; dlg.Destroy()
        if result == wx.ID_OK:
            PATH_File = dlg.GetDirectory()+'\\'+dlg.GetFilename() ; NAME_File=dlg.GetFilename()
            CNX = sqlite.connect(PATH_File) ; cursor = CNX.cursor() ; cursor1 = CNX.cursor()
            cursor.execute("SELECT name FROM sqlite_master WHERE type='table';")
            TABLES  = [ t[0] for t in cursor.fetchall() if t[0] != "sqlite_sequence"]
            SQLite_DATA=[]
            for table in TABLES:
                cursor1.execute( "SELECT count(*) FROM "+table) ; RECCOUNT=cursor1.fetchone()[0]
                sir="Database = "+PATH_File+"    Table = "+table+"   Records = "+str(RECCOUNT)+"\n"
                SQLite_DATA.append(sir ) ; SELECT="select * from "+table  ;  cursor.execute(SELECT)
                # fields_names = [description[0] for description in cursor.description]
                rows = cursor.fetchall()
                for row in rows:
                    x =",".join(str(elem) for elem in row)
```

Listing 20.1 PDF operations.pyw – PDF file operations – Python & wxPython version

```
                    x = x.encode('ascii', 'ignore').decode()
                    SQLite_DATA.append(x+"\n")
            cursor.close()  ;  CNX.close()
            Generate_PDF(NAME_File, PATH_File, 11, 'Courier', 'Landscape', SQLite_DATA)
            wx.MessageBox("File:\n\n"+PATH_File+"\n\nconverted to:\n\n"+os.getcwd()+"\\"+NAME_File[:-3]+"pdf")

    def OnOpenPDF(self, event):
        dlg = wx.FileDialog(self, wildcard="*.pdf")
        result = dlg.ShowModal() ; FISIER = dlg.GetPath() ; dlg.Destroy()
        if result == wx.ID_OK:
            try:
                win32api.ShellExecute (  0,  "open",  FISIER,  None,  ".",  0)
            except:
                frame = wx.Frame(None, -1, "Open PDF file", size = (900, 700))
                PanelPDF(frame, FISIER)
                frame.Show(True)

    def OnMergePDF(self, event):
        Merge_PDF(None, -1, 'Merge PDF files')

    def OnClose(self, event):
        self.Destroy()

class PanelPDF(wx.Panel): #==================================
    def __init__(self, parent,FISIER):
        wx.Panel.__init__(self, parent, id=-1)
        self.pdf = None

        sizer = wx.BoxSizer(wx.VERTICAL) ; btnSizer = wx.BoxSizer(wx.HORIZONTAL)
        self.pdf = PDFWindow(self, style=wx.SUNKEN_BORDER)
        sizer.Add(self.pdf, proportion=1, flag=wx.EXPAND)
        btn = wx.Button(self, wx.NewId(), "Previous Page")
        self.Bind(wx.EVT_BUTTON, self.OnPrevPageButton, btn)
        btnSizer.Add(btn, proportion=1, flag=wx.EXPAND|wx.ALL, border=5)
        btn = wx.Button(self, wx.NewId(), "Next Page")
        self.Bind(wx.EVT_BUTTON, self.OnNextPageButton, btn)
        btnSizer.Add(btn, proportion=1, flag=wx.EXPAND|wx.ALL, border=5)
        btnSizer.Add((50,-1), proportion=2, flag=wx.EXPAND)
        sizer.Add(btnSizer, proportion=0, flag=wx.EXPAND)

        if FISIER.strip()!="":  self.pdf.LoadFile(FISIER)
        self.SetSizer(sizer) ; self.SetAutoLayout(True)

    def OnPrevPageButton(self, event):
        self.pdf.gotoPreviousPage()

    def OnNextPageButton(self, event):
        self.pdf.gotoNextPage()

class Merge_PDF(wx.Frame): # ==============================================
 def __init__(self, parent, id, title):
  wx.Frame.__init__(self, parent, id, "Merge PDF files", size=(500, 330),
        style=wx.DEFAULT_FRAME_STYLE ^ ( wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
    panel = wx.Panel(self, -1) ; hbox = wx.BoxSizer(wx.HORIZONTAL)
    self.listbox = wx.ListBox(panel, -1, size=(600, 180))
    self.listbox.SetForegroundColour("BLUE")
    hbox.Add(self.listbox, 1,  wx.ALL, 20)
    btnPanel = wx.Panel(panel, -1)
    vbox = wx.BoxSizer(wx.VERTICAL)
    st1=wx.StaticText(panel, -1, 'Path && PDF source names', pos=(20, 5))
    st1.SetForegroundColour("BLUE")
    btn_BROWSE = wx.Button(btnPanel, wx.NewId(), "Browse", size=(90, 25))
    btn_BROWSE.SetForegroundColour("BLUE")
    btn_BROWSE.SetToolTipString ('Select PDF file source path && name')
    self.Bind(wx.EVT_BUTTON, self.OnBROWSE, btn_BROWSE)
    btn_UP = wx.Button(btnPanel, wx.NewId(), "Up", size=(90, 25))
    btn_UP.SetForegroundColour("BLUE") ; self.Bind(wx.EVT_BUTTON, self.OnUp, btn_UP)
    btn_DOWN = wx.Button(btnPanel, wx.NewId(), "Down", size=(90, 25))
    btn_DOWN.SetForegroundColour("BLUE") ; self.Bind(wx.EVT_BUTTON, self.OnDown, btn_DOWN)
    btn_DELETE = wx.Button(btnPanel, wx.NewId(), "Delete", size=(90, 25))
    btn_DELETE.SetForegroundColour("BLUE") ; self.Bind(wx.EVT_BUTTON, self.OnDelete, btn_DELETE)
    btn_CLEAR = wx.Button(btnPanel, wx.NewId(), "Clear", size=(90, 25))
```

Listing 20.1 PDF operations.pyw – PDF file operations – Python & wxPython version

```
                btn_CLEAR.SetForegroundColour("BLUE") ; self.Bind(wx.EVT_BUTTON, self.OnClear, btn_CLEAR)
                btn_EXIT = wx.Button(btnPanel, wx.NewId(), "Exit", size=(90, 30))
                btn_EXIT.SetForegroundColour("RED") ; self.Bind(wx.EVT_BUTTON, self.OnExit, btn_EXIT)
                btn_Unire = wx.Button(btnPanel, wx.NewId(), "Merge PDF", size=(90, 60))
                btn_Unire.SetBackgroundColour('Blue') ; btn_Unire.SetForegroundColour('White')
                btn_Unire.SetToolTipString ('Merge PDF files into one PDF file')
                self.Bind(wx.EVT_BUTTON, self.OnMerge, btn_Unire)
                box = wx.StaticBox(panel, -1, "", pos=(15, 205), size=(350,60))
                st3=wx.StaticText(panel, -1, 'Path && PDF destination file', pos=(20, 217))
                st3.SetForegroundColour("Black")
                btn_CaleNumePDF = wx.Button(panel, wx.NewId(), "Browse", pos=(290, 215), size=(70, 20))
                btn_CaleNumePDF.SetToolTipString ('Select PDF file destination')
                btn_CaleNumePDF.SetForegroundColour('Black') ; btn_CaleNumePDF.SetBackgroundColour('White')
                self.Bind(wx.EVT_BUTTON, self.OnCaleNumePDF, btn_CaleNumePDF)
                self.nume_fisier= wx.TextCtrl(panel, -1, os.getcwd()+"\Merge.PDF",
                                pos=(20, 235), size=(340, -1), style=wx.TE_LEFT)
                self.nume_fisier.SetForegroundColour('White') ; self.nume_fisier.SetBackgroundColour('Blue')

                vbox.Add((-1, 20)) ; vbox.Add(btn_BROWSE)
                vbox.Add(btn_UP, 0, wx.TOP, 5) ; vbox.Add(btn_DOWN, 0, wx.TOP, 5)
                vbox.Add(btn_DELETE, 0, wx.TOP, 5) ; vbox.Add(btn_CLEAR, 0, wx.TOP, 5)
                vbox.Add(btn_EXIT, 0, wx.TOP, 5) ; vbox.Add(btn_Unire, 0, wx.TOP, 5)
                btnPanel.SetSizer(vbox) ; hbox.Add(btnPanel, 0.6, wx.EXPAND | wx.RIGHT, 20)
                panel.SetSizer(hbox) ; self.Centre() ; self.Show(True)

    def OnBROWSE(self, event):
        wildcard = "PDF (*.pdf)|*.pdf"
        dlg = wx.FileDialog(
            self, message="Choose a file", defaultDir=os.getcwd(),
            defaultFile="", wildcard=wildcard, style=wx.OPEN | wx.CHANGE_DIR )
        if dlg.ShowModal() == wx.ID_OK:
            path = dlg.GetPath()
            if path.strip() != '':
                self.listbox.Append(path)
        dlg.Destroy()

    def OnCaleNumePDF(self, event):
        dlg = wx.FileDialog(self, message="Path && PDF file name",
                defaultDir=os.getcwd(),defaultFile="",wildcard="*.pdf",
                                style=wx.OPEN | wx.CHANGE_DIR)
        if dlg.ShowModal() == wx.ID_OK:
                paths= dlg.GetPaths()
                for path in paths:
                                self.nume_fisier.SetValue(str(path))

    def OnUp(self, event):
        if self.listbox.GetCount()<=1:
                return
        sel = self.listbox.GetSelection()       # Selected index option
        nume_sel=self.listbox.GetString(sel)    # Selected index name=Path+name PDF file
        lst=[]                                  # Taking options from 'listbox' control
        if sel != -1 and sel-1>=0:
            for i in range(self.listbox.GetCount()):
                    lst.append(self.listbox.GetString(i))
            del lst[sel]                        # Remove from selected item list
            lst.insert(sel-1,nume_sel)          # Insert selected item in the the correct position list
            self.listbox.Clear()                # Clear 'listbox' control
            for i in lst:                       # Adding options back in 'listbox' control
                    self.listbox.Append(i)
            self.listbox.SetSelection(sel-1)    # Reselect the selected list element

    def OnDown(self, event):
        if self.listbox.GetCount()<=1:
                return
        sel = self.listbox.GetSelection()       # Selected index option
        nume_sel=self.listbox.GetString(sel)    # Selected index name=Path+name PDF file
        lst=[]                                  # Taking options from 'listbox' control
        if sel != -1 and sel+1<=self.listbox.GetCount()-1:
            for i in range(self.listbox.GetCount()):
                    lst.append(self.listbox.GetString(i))
            del lst[sel]                        # Remove from selected item list
            lst.insert(sel+1,nume_sel)          # Insert selected item in the the correct position list
```

Listing 20.1 PDF operations.pyw – PDF file operations – Python & wxPython version

```
            self.listbox.Clear()              # Clear 'listbox' control
            for i in lst:                # Adaugarea optiuni inapoi in control 'listbox'
                self.listbox.Append(i)
            self.listbox.SetSelection(sel+1)   # Reselect the selected list element

    def OnDelete(self, event):
        sel = self.listbox.GetSelection()
        if sel != -1:
            self.listbox.Delete(sel)

    def OnClear(self, event):
        self.listbox.Clear()

    def OnMerge(self, event):
        numePDF=self.nume_fisier.GetValue().strip()
        if self.listbox.GetCount()<2 or numePDF=="":
                                             return
        if os.path.exists(numePDF):
                msg = "File "+numePDF+" exist.\n" ; msg=msg+"Confirm overwrite ?"
                dlg = wx.MessageDialog(None, msg, 'Confirm', wx.YES_NO |wx.ICON_EXCLAMATION)
                if dlg.ShowModal()==wx.ID_NO:
                                dlg.Destroy()
                                return
        output = pyPdf.PdfFileWriter()
        for i in range(self.listbox.GetCount()):
            f_PDF=self.listbox.GetString(i).strip()
            PDF_File=pyPdf.PdfFileReader(file(f_PDF, "rb"))
            Nr_Pag=PDF_File.getNumPages()
            cnt=1
            for page in range(Nr_Pag-1):
                output.addPage(PDF_File.getPage(page))
                cnt+=1
        outputStream = file(numePDF, "wb")
        output.write(outputStream)
        outputStream.close()
        try:
                win32api.ShellExecute (  0,  "open",  numePDF,  None,  ".",  0)
       except:
                frame = wx.Frame(None, -1, numePDF, size = (900, 700))
                PanelPDF(frame, numePDF) ; frame.Show(True)

    def OnExit(self,event):
        self.Close(True)  ;  self.Destroy()

if __name__ == '__main__':   #==================================
                app = wx.App(1)
                PDF_Files()
                app.MainLoop()
```

21. WEB Tools

This script is focus on the following operations:
❶ accesing WEB pages from Python script;
❷ convert database table to HTML format and display in **wx.lib.iewin.IEHtmlWindow**;
❸ accesing country and cities maps and display in browser (only for **Python 3.11.9** version).

The **Python 3.11.9** version installer **python-3.11.9-amd64.exe** (~25 Mb) can be found at [**21.1**]; click on downloaded file to install. Add Python 3.11.9 to PATH environment variables:

PYTHON 3.11 INCLUDE	**PYTHON 3.11 LIB**
C:\Program Files\Python311\include	**C:\Program Files\Python311\libs\python311.lib**

The **WEB Tools 3.11.pyw** script import the following libraries: **sqlite3**, **wxPython**, **webbrowser** (included in **wxPython)**, **mysql.connector** and **folium**.

To install **sqlite3** goto [**20.2**], write **sqlite3** in **Search** control and select **SQLite3-0611 0.0.1** version to download **SQLite3_0611-0.0.1-py3-none-any.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip3.11 install SQLite3_0611-0.0.1-py3-none-any.whl
```

To install **wxPython** goto [**20.2**], write **wxPython** in **Search** control and select **wxPython 4.2.1** version to download **wxPython-4.2.1-cp311-cp311-win_amd64.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip3.11 install wxPython-4.2.1-cp311-cp311-win_amd64.whl
```

But **wxPython** require **comtypes** library; to install **comtypes** goto [**20.2**], write **comtypes** in **Search** control and select **comtypes 1.4.6** version to download **comtypes-1.4.6-py3-none-any.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip3.11 install comtypes-1.4.6-py3-none-any.whl
```

To install **mysql.connector** goto [**20.2**], write **mysql.connector python 3** in **Search** control and select **mysql-connector-python 9.0.0** version to download **mysql_connector_python-9.0.0-cp311-cp311-win_amd64.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip3.11 install mysql_connector_python-9.0.0-cp311-cp311-win_amd64.whl
```

To access WEB pages from Python script the library **webbrowser** is used. This library is part of the python standard library, so you don't have to install a separate package to use it.

The **wx.lib.iewin.IEHtmlWindow** display rich content pages (either local file or downloaded via HTTP protocol) in a window based on a subset of the HTML standard and is part of **wxPython** package.

To acces country and cities maps the **folium** library, [**21.2**], must be installed for **Python 3.11;** goto [**20.2**], write **folium** in **Search** control and select **folium 0.17.0** version to download **folium-0.17.0-py2.py3-none-any.whl**file; open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip3 install folium-0.17.0-py2.py3-none-any.whl
```

The **folium** library makes it easy to display maps and data from Python on an interactive leaflet map. The **WEB Tools 3.11.pyw** script use **folium** library to display country and city maps on the current browser.

The **WEB Tools 3.11.pyw** script interface is displayed in **Figure 21.1**. The **WEB Tools 2.7.pyw** script interface is displayed in **Figure 21.2**. For both version the **WEB Pages** menu is displayed in **Figure 21.3**. Only for **WEB Tools 3.11.pyw** script, the select country or city menu is displayed in **Figure 21.4**; this option is not available for **WEB Tools 2.7.pyw** script because **folium** library require **Python** >=3.8.

For **WEB Tools 2.7.pyw** script the **Config.db** database is used to store all icons loaded in frame **Toolbar**; this will avoid the existence of following icons files in the application folder: and . The **SQL** to generate **objects** table is displayed in **Figure 15.4**. The **Config.db → objects** memorize icons for the script interface for **Python 2.7**

& wxPython application version. Initially, the icons files are stored in the **WEB tools\Python 2.7 → Objects** application folder. The icons files were loaded into **Config.db → Objects** as **Binary Large Object** (**BLOB**) through the **_copyimg.pyw** script displayed in **Listing 15.1**, from where they will be used into application interface through **Python 2.7 → WEB Tools 2.7.pyw** script. For **WEB Tools 3.11.pyw** script and icons were placed directly on **WEB tools** folder.

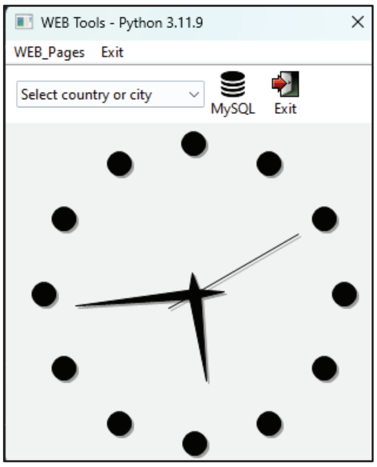


Figure 21.1 The **WEB Tools 3.11.pyw** interface

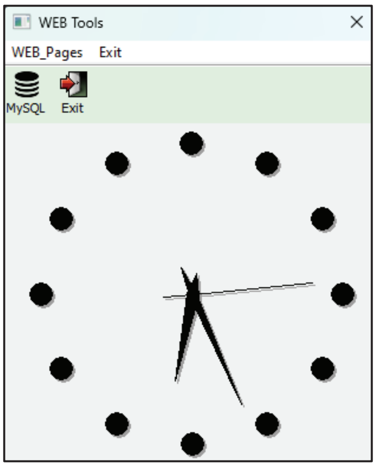


Figure 21.2 The **WEB Tools 2.7.pyw** interface

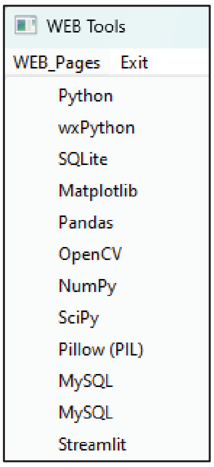


Figure 21.3 WEB Pages menu

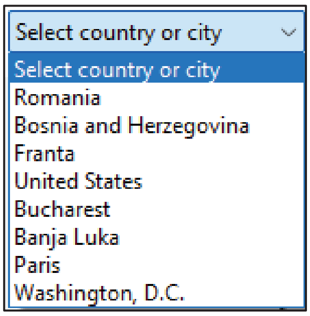


Figure 21.4 Select country or city menu

Listing 21.1 displays the **Python & wxPython** version of script, **WEB Tools 3.11.pyw**.

The **WEB Tools 3.11.pyw** script (Python 3.11.9 & wxPython version) include three classes: **MyExplorer**, **CreateMap** and **WEB_Tools**. The script begin with importing libraries (lines 1÷7).

The **MyExplorer** class create the interface to display rich content pages (either local file or downloaded via HTTP protocol):

- **__init__** Define the **frame** (line 10) and **self.PDFPanel** variable (line 12) using the **wx.lib.iewin.IEHtmlWindow**; this class receive the following parameters: **title** and **fileName** to display; line 13 display the content of **fileName** file.

- **__init__** Define the **frame** (line 10) and **self.PDFPanel** variable (line 12) using the **wx.lib.iewin.IEHtmlWindow**; this class recive the following parameters: **title** and **fileName** to display; line 13 display the content of **fileName** file.

The **CreateMap** class create the interface to display the maps in browser:

- **__init__** This class recive the following parameters: **center** (the maps location to be displayed: **Latitude** & **Longitude** coordinates) and **zoom_start to**, which is the zoom factor to display the map. The maps coordinates can be obtained for every country or city from [21.3]. Line 17 use **folium** library to create the map at center location and at zoom factor. Line 18 place an icon at location using **cloud** icon. Line 19 save the map as HTML file with the **map.html** name. This class is called by **EvtChoice** function from **WEB_Tools** class.

The **WEB_Tools** class include the following functions:

- **__init__** Define the **frame** (line 23÷24) and the **panel** (line 25); create the sizer (line 25), a functional visual clock (lines 26÷29) and place the clock inside the sizer (line 30); line 31 define the public class variable; line 32 call **_Gen_MENU** function to create a menu; line 33 call **_TOOLBAR** function to create the toolbar; line 34 center and show the frame in the display.
- **_TOOLBAR** Line 37 create the toolbar; lines 38 define the toolbar background and the icon size; line 40 add a separator in toolbar; line 41 store, in **sir** variable, the options of the **self.Choice_map** control defined in lines 43, 44; line 45 place the **Select country or city** text as first option in **self.Choice_map** control. Line 47 add **self.Choice_map** control

inside the toolbar; line 48 connect **self.Choice_map** control with **self.EvtChoice** function. Lines 50÷52 create the **MySQL** icon and connect with **self.HTML** function. Lines 55÷58 create the **Exit** icon and connect with **self.OnExit**function; line 60 generate and show the toolbar.

- **EvtChoice** Get the option selected in **self.Choice_map** control in **select** variable; lines 65÷73 define the maps coordinate and zoom factor for the selected option; line 75 call **CreateMap** class to create and save the map as **map.html** file; line 76 open the map from **map.html** file through webbrowser library.
- **_Gen_MENU** This function create the interface menu: **WEB Pages** and **Exit**.
- **On_Python** Open the **Python** page through **webbrowser** library.
- **On_wxPython** Open the **wxPython** page through **webbrowser** library.
- **On_Sqlite** Open the **Sqlite** page through **webbrowser** library.
- **On_Matplotlib** Open the **Matplotlib** page through **webbrowser** library.
- **On_Pandas** Open the **Pandas** page through **webbrowser** library.
- **On_OpenCV** Open the **OpenCV** page through **webbrowser** library.
- **On_NumPy** Open the **NumPy** page through **webbrowser** library.
- **On_SciPy** Open the **SciPy** page through **webbrowser** library.
- **On_Pillow** Open the **Pillow** page through **webbrowser** library.
- **On_MySQL** Open the **MySQL** page through **webbrowser** library.
- **On_Streamlit** Open the **Streamlit** page through **webbrowser** library.
- **HTML** Connect with database and for every table read the records and create the HTML file saved as **fis_HTML** (line 137).
- **OnExit** Close the window.

Figure 21.5 show the maps of France and **Figure 21.6** show the maps of Washington. This maps can be zoomed in or out with mouse like any map.

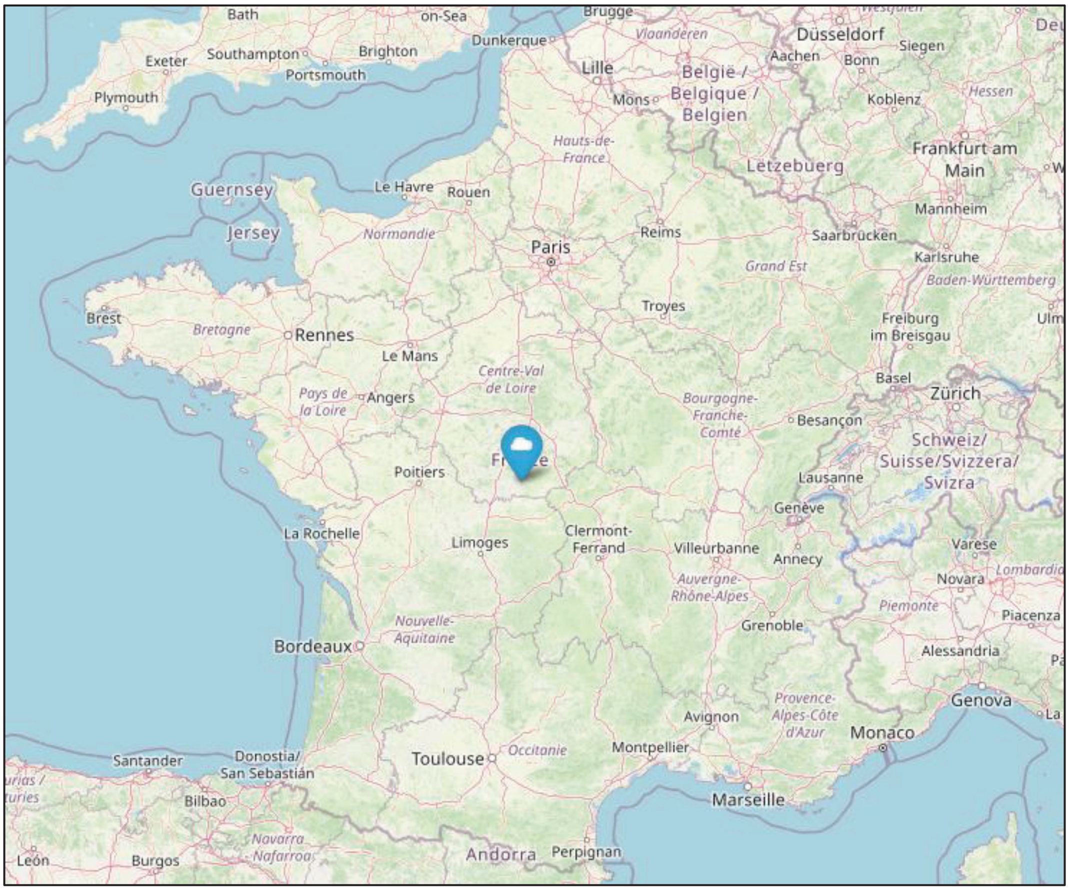


Figure 21.5 The maps of France

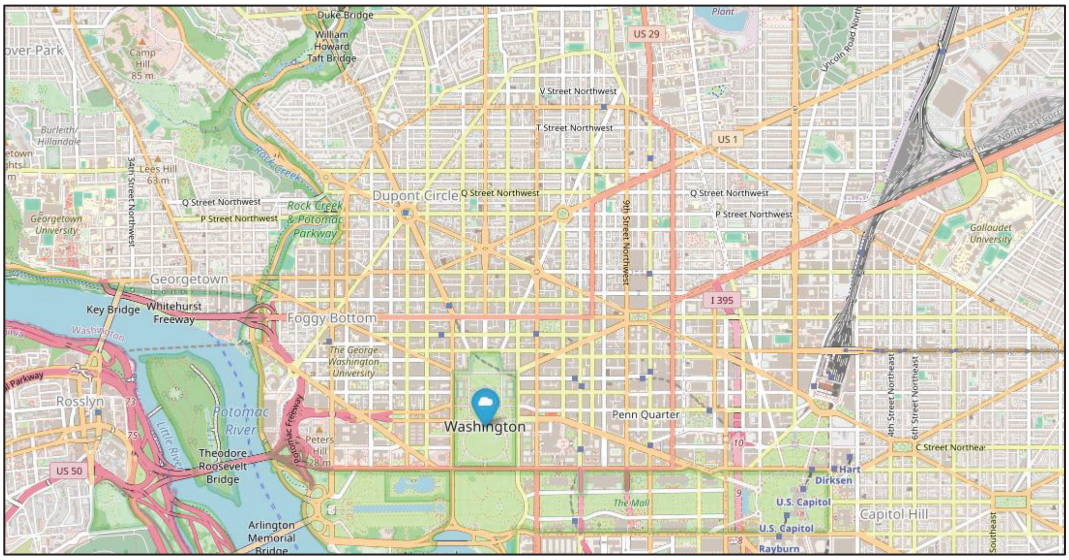


Figure 21.6 The maps of Washington

Listing 21.1 WEB Tools 3.11.pyw – WEB tools – Python & wxPython version

```
import wx, os
from sqlite3 import dbapi2 as sqlite
import wx.lib.analogclock as ac
import wx.lib.iewin
import webbrowser
import mysql.connector
import folium

class MyExplorer(wx.Frame):  # Must import:  import wx.lib.iewin
    def __init__(self, parent,  title, fileName):
        wx.Frame.__init__(self, parent, -1, title)
        self.PDFPanel = wx.lib.iewin.IEHtmlWindow(self, -1, style = wx.NO_FULL_REPAINT_ON_RESIZE)
        self.PDFPanel.LoadUrl(fileName)

class CreateMap: # ==============================================
    def __init__(self, center, zoom_start):
        MAP = folium.Map(location = center, zoom_start = zoom_start, control_scale=True)
        folium.Marker(location=center,icon=folium.Icon(icon="cloud"),).add_to(MAP)
        MAP.save("map.html")

class WEB_Tools(wx.Frame):  # ==============================================
    def __init__(self):
        wx.Frame.__init__(self, None, -1, title="WEB Tools - Python 3.11.9", size=(330,400),
            style=wx.DEFAULT_FRAME_STYLE ^ ( wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
        panel = wx.Panel(self,-1,) ; sizer= wx.GridBagSizer(hgap=10, vgap=40)
        ceas = ac.AnalogClock(panel, hoursStyle=ac.TICKS_CIRCLE, \
            clockStyle=ac.SHOW_HOURS_TICKS|ac.SHOW_HOURS_HAND| \
            ac.SHOW_MINUTES_HAND| ac.SHOW_SECONDS_HAND| ac.SHOW_SHADOWS, \
            size=(280,280), pos=(20,5))
        sizer.Add(ceas,pos=(1,1) )
        self.CONFIG_DB=os.getcwd()+'\Config.db'
        self._Gen_MENU()
        self._TOOLBAR()
        self.CenterOnScreen() ; self.Show()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour("white") ; self.toolbar.SetToolBitmapSize((24,24))

        self.toolbar.AddSeparator()
        sir=["Select country or city", "Romania", "Bosnia and Herzegovina","Franta", "United States",
            "Bucharest","Banja Luka","Paris","Washington, D.C."]
        self.Choice_map = wx.Choice(self.toolbar, -1, size=(160, 50), choices = sir)
        self.Choice_map.SetToolTip("Select the location's map !\nhttps://www.gps-coordinates.net/map/country/BA")
        x=self.Choice_map.FindString('Select country or city') ; self.Choice_map.SetSelection(x)

        self.toolbar.AddControl( self.Choice_map)
        self.Bind(wx.EVT_CHOICE, self.EvtChoice, self.Choice_map)
```

Listing 21.1 WEB Tools 3.11.pyw – WEB tools – Python & wxPython version

```

        img = wx.Image(os.getcwd()+'/MySQL.png', wx.BITMAP_TYPE_ANY)
        id_MySQL=91 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool(id_MySQL  , "MySQL", sel_bmp, shortHelp="Convert SQLite table to HTML file")
        self.Bind(wx.EVT_TOOL, self.HTML, id=id_MySQL)

        img = wx.Image(os.getcwd()+'/exit.png', wx.BITMAP_TYPE_ANY)
        id_Exit=93 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_Exit , "Exit", sel_bmp, shortHelp="Exit application")
        self.Bind(wx.EVT_TOOL, self.OnExit, id=id_Exit)

        self.toolbar.Realize() ; self.toolbar.Show()       # Toolbar show

    def EvtChoice(self, event):  # https://www.gps-coordinates.net/map/country/BA
        Select = event.GetString()

        if Select == "Romania": CRD=[45.9852129, 24.6859225] ; ZOOM=7
        if Select == "Bosnia and Herzegovina": CRD=[43.8926525, 17.67058399999999] ; ZOOM=8
        if Select == "Franta": CRD=[46.603354,1.8883335] ; ZOOM=6
        if Select == "United States": CRD=[39.7837304,-100.445882] ; ZOOM=5

        if Select == "Bucharest": CRD=[44.4361414, 26.1027202] ; ZOOM=12
        if Select == "Banja Luka": CRD=[44.7720845,17.1917651] ; ZOOM=14
        if Select == "Paris": CRD=[48.8588897,2.320041] ; ZOOM=14
        if Select == "Washington, D.C.": CRD=[38.8950368,-77.0365427] ; ZOOM=14

        map = CreateMap(center=CRD, zoom_start = ZOOM)
        webbrowser.open("map.html")

    def _Gen_MENU(self):
        WEB_Menu= wx.Menu()
        menu_EXIT = wx.Menu()

        item_Python= WEB_Menu.Append(1,"Python", "View Python page")
        self.Bind(wx.EVT_MENU, self.On_Python, item_Python)
        item_wxPython= WEB_Menu.Append(2,"wxPython", "View wxPython page")
        self.Bind(wx.EVT_MENU, self.On_wxPython, item_wxPython)
        item_SQLite= WEB_Menu.Append(3,"SQLite", "View SQLite page")
        self.Bind(wx.EVT_MENU, self.On_Sqlite, item_SQLite)
        item_Matplotlib= WEB_Menu.Append(4,"Matplotlib", "View Matplotlib page")
        self.Bind(wx.EVT_MENU, self.On_Matplotlib, item_Matplotlib)
        item_Pandas= WEB_Menu.Append(5,"Pandas", "View Pandas page")
        self.Bind(wx.EVT_MENU, self.On_Pandas, item_Pandas)
        item_OpenCV= WEB_Menu.Append(6,"OpenCV", "View OpenCV page")
        self.Bind(wx.EVT_MENU, self.On_OpenCV, item_OpenCV)
        item_NumPy= WEB_Menu.Append(7,"NumPy", "View NumPy page")
        self.Bind(wx.EVT_MENU, self.On_OpenCV, item_NumPy)
        item_SciPy= WEB_Menu.Append(8,"SciPy", "View SciPy page")
        self.Bind(wx.EVT_MENU, self.On_SciPy, item_SciPy)
        item_Pillow= WEB_Menu.Append(9,"Pillow (PIL)", "View Pillow (PIL) page")
        self.Bind(wx.EVT_MENU, self.On_Pillow, item_Pillow)
        item_MySQL= WEB_Menu.Append(10,"MySQL", "View MySQL page")
        self.Bind(wx.EVT_MENU, self.On_MySQL, item_MySQL)
        item_Streamlit= WEB_Menu.Append(12,"Streamlit", "View Streamlit page")
        self.Bind(wx.EVT_MENU, self.On_Streamlit, item_Streamlit)

        item_exit = menu_EXIT.Append(-1, "Exit", 'Close application')
        self.Bind(wx.EVT_MENU, self.OnExit, item_exit)

        menuBar = wx.MenuBar()
        menuBar.Append(WEB_Menu, "WEB_Pages")
        menuBar.Append(menu_EXIT, "Exit")
        self.SetMenuBar(menuBar)

    def On_Python(self, event):
        webbrowser.open("http://www.python.org/")
    def On_wxPython(self, event):
        webbrowser.open("http://www.wxpython.org/")
    def On_Sqlite(self, event):
        webbrowser.open("http://www.sqlite.org/")
    def On_Matplotlib(self, event):
            webbrowser.open("https://matplotlib.org/")
```

Listing 21.1 WEB Tools 3.11.pyw – WEB tools – Python & wxPython version

```
    def On_Pandas(self, event):
        webbrowser.open("https://pandas.pydata.org/")
    def On_OpenCV(self, event):
        webbrowser.open("https://opencv.org/")
    def On_NumPy(self, event):
        webbrowser.open("https://numpy.org/")
    def On_SciPy(self, event):
        webbrowser.open("https://scipy.org/")
    def On_Pillow(self, event):
        webbrowser.open("https://pillow.readthedocs.io/en/stable/")
    def On_MySQL(self, event):
        webbrowser.open("https://www.mysql.com/")
    def On_Streamlit(self, event):
        webbrowser.open("https://streamlit.io/")

    def HTML(self, event):
        fis_HTML=os.getcwd()+'\HTML_Report.html' ; f=open(fis_HTML,'w')
        conn = mysql.connector.connect(host="localhost" , user="root", passwd="password")
        cursor = conn.cursor() ; lst_DATABASES=[] ; cursor.execute("show databases")
        for (databases) in cursor:
            if databases[0]!="information_schema" and databases[0]!="mysql" and \
                databases[0]!="performance_schema" and databases[0]!="sys":
                    lst_DATABASES.append(str(databases[0]))
        conn.close()
        for DATABASE in lst_DATABASES:
            lst_TABLES=[]
            conn = mysql.connector.connect(host="localhost" , user="root", passwd="password",db=DATABASE)
            cursor = conn.cursor() ; cursor.execute("Show tables") ; TABLES = cursor.fetchall()
            for table in TABLES:
                lst_TABLES.append(str(table[0]))
            for TABLE in lst_TABLES:
                conn = mysql.connector.connect(host="localhost" , user="root",
                       passwd="password",db=DATABASE)
                cursor = conn.cursor()
                cursor.execute( "SELECT count(*) FROM "+TABLE) ;  RECCOUNT=cursor.fetchone()[0]
                cursor.execute("select * from "+TABLE) ; Crs_All=cursor.fetchall()
                lst_Fields=[]
                for elem in cursor.description:
                    lst_Fields.append(elem[0])
                titlu="Database:  "+DATABASE+"  Table:  "+TABLE+"  Records number:  "+str(RECCOUNT)
                sir="<CENTER>"+titlu+"</CENTER>" ; f.write(sir)
                sir="<TABLE BORDER='1' CELLSPACING='1' CELLPADDING='4'  "
                sir=sir+"WIDTH='100%' style='font-size:10pt '>"+' \n'
                f.write(sir) ; sir=' \n'+"<TR>"+' \n'  ;  f.write(sir)
                for FIELD in lst_Fields:
                    f.write("<TD>"+FIELD+"</TD>"+' \n')
                sir="</TR>"+' \n' ; f.write(sir)
                for row in Crs_All:
                    sir=' \n'+"<TR>"+' \n' ; f.write(sir)
                    for elem in row:
                        sir="<TD>"+str(elem)+"</TD>" ;  f.write(sir)
                    sir="</TR>"+' \n' ; f.write(sir)
                sir="</TABLE>"+' \n' ; f.write(sir)
                conn.close()
        f.close()
        frm = MyExplorer(None, "Show records from Databases & Tables", fis_HTML)
        frm.Show()

    def OnExit(self, event):
        self.Close(True) ; self.Destroy()

if __name__ == '__main__':   #====================================
    app = wx.App(0)
    WEB_Tools()
    app.MainLoop()
```

22. Optical Character Recognition (OCR)

The chapter exemplifies the optical character recognition process using **Tesseract's OCR** engine & **pytesseract** on Python 3.11. The script is **OCR.pyw** saved in **OCR** folder, with interface from **Figure 22.1**.

❶ To install **Tesseract's OCR** go to [**22.1**], download **tesseract-ocr-w64-setup-5.4.0.20240606.exe** file (~48 Mb) and click on it to install in proposed **C:\Program Files\Tesseract-OCR** path.

❷ Add **Tesseract** path to your System Environment.

❸ To install **pytesseract** go to [**20.2**], write **pytesseract** in **Search** control and select **pytesseract 0.3.13** version to download je_open_cv-0.0.22-py3-none-any.whl file; then open command prompt, go to folder where the **whl** was saved and write the following command:

pip3.11 install pytesseract-0.3.13-py3-none-any.whl

❸ To install **wxPython** library for **Python 3.11** go to [**20.2**], write **wxPython** in **Search** control and select **wxPython 4.2.1** version to download **wxPython-4.2.1-cp311-cp311-win_amd64.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

pip3.11 install wxPython-4.2.1-cp311-cp311-win_amd64.whl

❹ To install **Open CV** library for **Python 3.11** go to [**20.2**], write **open cv** in **Search** control and select **je-open-cv 0.0.22** version to download **je_open_cv-0.0.22-py3-none-any.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

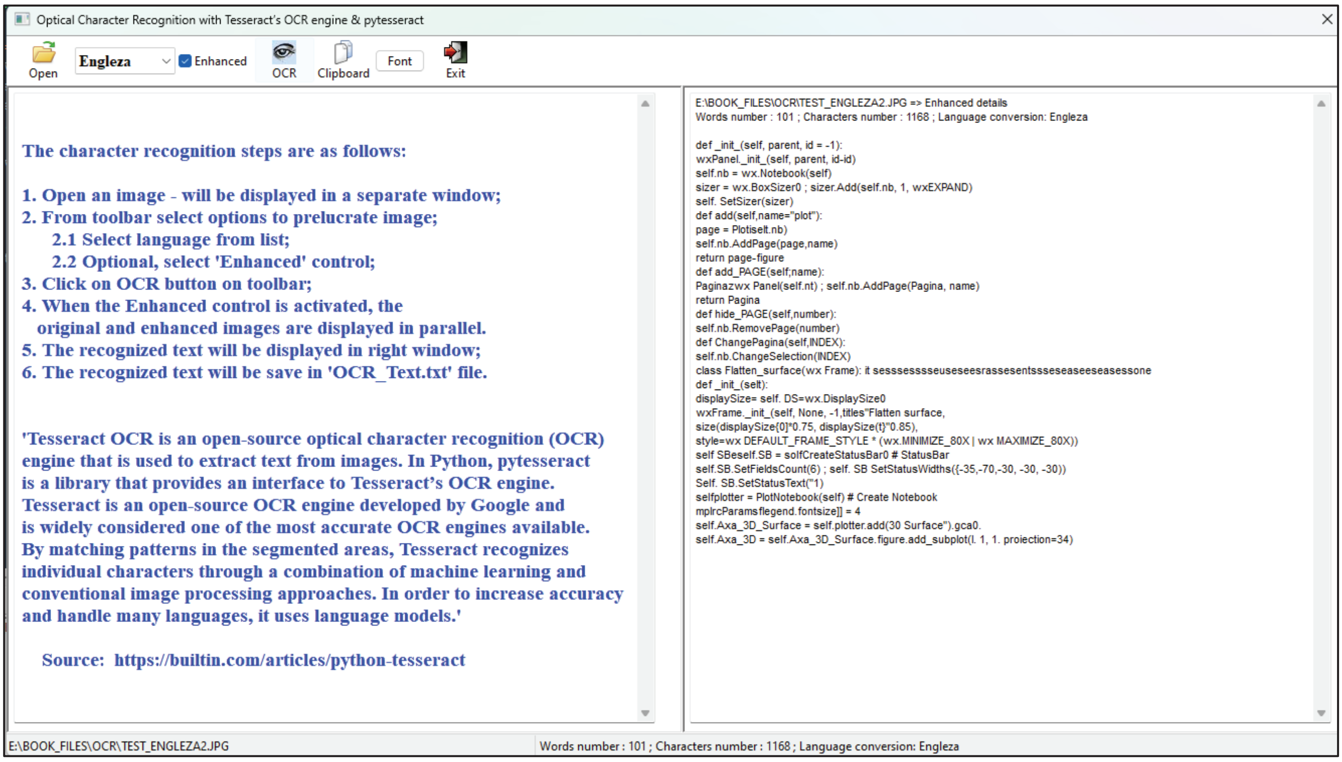
pip3.11 install je_open_cv-0.0.22-py3-none-any.whl

❺ To install **matplotlib** library for **Python 3.11** open **command prompt** and place next command:

pip3.11 install matplotlib

❻ To install **PILLOW (PIL)** library for **Python 3.11** open **command prompt** and place next command:

pip3 install pillow


Optical Character Recognition with Tesseract's OCR engine & pytesseract
Open
Engleza
☑ Enhanced
OCR
Clipboard
Font
Exit

The character recognition steps are as follows:

1. Open an image - will be displayed in a separate window;
2. From toolbar select options to prelucrate image;
2.1 Select language from list;
2.2 Optional, select 'Enhanced' control;
3. Click on OCR button on toolbar;
4. When the Enhanced control is activated, the original and enhanced images are displayed in parallel.
5. The recognized text will be displayed in right window;
6. The recognized text will be save in 'OCR_Text.txt' file.

'Tesseract OCR is an open-source optical character recognition (OCR) engine that is used to extract text from images. In Python, pytesseract is a library that provides an interface to Tesseract's OCR engine. Tesseract is an open-source OCR engine developed by Google and is widely considered one of the most accurate OCR engines available. By matching patterns in the segmented areas, Tesseract recognizes individual characters through a combination of machine learning and conventional image processing approaches. In order to increase accuracy and handle many languages, it uses language models.'

Source: https://builtin.com/articles/python-tesseract

```
E:\BOOK_FILES\OCR\TEST_ENGLEZA2.JPG => Enhanced details
Words number : 101 ; Characters number : 1168 ; Language conversion: Engleza

def _init_(self, parent, id = -1):
wxPanel._init_(self, parent, id-id)
self.nb = wx.Notebook(self)
sizer = wx.BoxSizer0 ; sizer.Add(self.nb, 1, wxEXPAND)
self. SetSizer(sizer)
def add(self,name="plot"):
page = Plotiselt.nb)
self.nb.AddPage(page,name)
return page-figure
def add_PAGE(self;name):
Paginazwx Panel(self.nt) ; self.nb.AddPage(Pagina, name)
return Pagina
def hide_PAGE(self,number):
self.nb.RemovePage(number)
def ChangePagina(self,INDEX):
self.nb.ChangeSelection(INDEX)
class Flatten_surface(wx Frame): it sesssesssseuseseesrassesentssseseaseeseasessone
def _init_(selt):
displaySize= self. DS=wx.DisplaySize0
wxFrame._init_(self, None, -1,titles"Flatten surface,
size(displaySize{0]*0.75, displaySize(t}"0.85),
style=wx DEFAULT_FRAME_STYLE * (wx.MINIMIZE_80X | wx MAXIMIZE_80X))
self SBeself.SB = solfCreateStatusBar0 # StatusBar
self.SB.SetFieldsCount(6) ; self. SB SetStatusWidths({-35,-70,-30, -30, -30))
Self. SB.SetStatusText("1)
selfplotter = PlotNotebook(self) # Create Notebook
mplrcParamsflegend.fontsize]] = 4
self.Axa_3D_Surface = self.plotter.add(30 Surface").gca0.
self.Axa_3D = self.Axa_3D_Surface.figure.add_subplot(l. 1, 1. proiection=34)
```

E:\BOOK_FILES\OCR\TEST_ENGLEZA2.JPG

Words number : 101 ; Characters number : 1168 ; Language conversion: Engleza

Figure 22.1 The **OCR.pyw** interface

Listing 22.1 displays the **Python & wxPython** version of script, **WEB Tools 3.11.pyw**.

The **OCR.pyw** script (Python 3.11.9 & wxPython version) include one class: **OCR** and one public function **Put_Clipboard**. The script begin with importing libraries (lines 3÷15).

The public function **Put_Clipboard** (lines 17÷22) receive a string from caller function and place that string in **Clipboard**.

The **OCR** class create the script interface:

- **__init__** Define the **frame** (line 28) and **statusbar** (line 31÷33). Line 35 define public variable s of the class. Line 37 create **main_panel**. Lines 38÷40 define **panel1** (left) and **panel2** (right) and their background. Lines 38÷40 define the text content that will be placed in **panel1** in **self.text1** control. Lines 62÷66 define **self.text1** control, his font and text blue color. Lines 68÷73 define **self.text2** control and his font; in this control will be placed the recognized text. Lines 75 define sizers to place **panel1** and **panel2**. Line 82 call **_TOOLBAR** function and center the frame in display.
- **_TOOLBAR** Lines 106, 107 define the **toolbar**, the background and the icons size. Line 111-114 define **Open** control and Bind with **OnOpenImage** class function to open an image that will be used to recognize characters. Line 116-126 define a choice control **self.Lang** with some languages used to recognize characters, but manny other languages are available [22.2]; the **Lang1** list store the real names of the language; the **Lang2** list store the abbreviated names of the language; the **self.Lang** control will show only the values from **Lang1** list. Lines 128÷130 define self.Enhanced **Check box** control as an option to enhance or not the original image. Lines 132÷136 define a control connected to **OnOCR** class function to start the OCR process. Lines 138÷141 define a control connected to **OnClipboard** class function to put a string on **Clipboard**. Lines 143÷147 define a control connected to **OnSelectFont** class function to change font type, color and size of the recognized text from **panel2**. Lines 149÷152 define a control to exit from main window and close the application.
- **OnClipboard** This class function (lines 156÷163) place on **Clipboard** the text content of **self.Clipboard** variable.
- **OnOpenImage** This function (lines 84÷95) open a dialog to select an image with **self.filename** name and call **ShowImage** class function to display the selected image.
- **ShowImage** This function (lines 97÷103) display the original image selected on **OnOpenImage** function.
- **OnOCR** Verify if an image file is open (lines 176÷179); select the language abbreviation from **self.Lang2** list through **INDEX** variable of **self.Lang2** list (lines 181÷183). If **self.Enhanced** control was activated by the user display in a separate window the original and enhanced images (lines 187÷196); lines 201÷202 convert image to characters using **pytesseract** module; line 210 put the **TITLE** (line 189 or 198) in **Clipboard**; lines 212÷213 save the recognized text to **OCR_Text.txt** file.
- **OnSelectFont** This class function (lines 165÷173) open a dialog to select and change the font type, color and size of the recognized text from **panel2**.
- **OnExit** Close the application.

Listing 22.1 OCR.pyw – Optical Character Recognition – Python & wxPython version

```
1.	from __future__ import division
2.	
3.	import wx, os
4.	from matplotlib.figure import Figure
5.	import matplotlib.image as mpimg
6.	import matplotlib.pyplot as plt
7.	from matplotlib.backends.backend_wx import FigureCanvasWx as FigureCanvas
8.	try:
9.		import Image
10.	except ImportError:
11.		from PIL import Image
12.	import win32clipboard
13.	import pytesseract
14.	pytesseract.pytesseract.tesseract_cmd = 'C:\\Program Files\\Tesseract-OCR\\tesseract.exe'
15.	import cv2
```

Listing 22.1 OCR.pyw – Optical Character Recognition – Python & wxPython version

```

def Put_Clipboard(String_Clip):
    String_Clip1 = str(String_Clip)
    win32clipboard.OpenClipboard()
    win32clipboard.EmptyClipboard()
    win32clipboard.SetClipboardText(String_Clip1)
    win32clipboard.CloseClipboard()

class OCR(wx.Frame):
    def __init__(self, parent, ID, title):
        self.CW=0.7 ; self.CH=0.7
        W,H = wx.DisplaySize() ; W=int(W*self.CW) ; H=int(H*self.CH)
        self.frame=wx.Frame.__init__(self, parent, ID, title, size=(W, H),
            style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER|wx.MINIMIZE_BOX |wx.MAXIMIZE_BOX))

        self.statusbar = self.CreateStatusBar() # Create the status bar
        self.statusbar.SetFieldsCount(2)
        self.statusbar.SetStatusWidths([-40, -60])

        self.filename="" ; self.COLOR="Black" ; self.FONT=8

        main_panel = wx.Panel(self)
        self.panel1 = wx.Panel(main_panel,-1, size=(int(W/2), int(H)),style=wx.SUNKEN_BORDER)
        self.panel2 = wx.Panel(main_panel,-1, size=(int(W/2), int(H)),style=wx.SUNKEN_BORDER)
        self.panel1.SetBackgroundColour("White") ; self.panel2.SetBackgroundColour("White")
        sir_Text1 = 2*"\n"+"The character recognition steps are as follows:\n\n"
        sir_Text1 = sir_Text1 + "1. Open an image - will be displayed in a separate window;\n"
        sir_Text1 = sir_Text1 + "2. From toolbar select options to prelucrate image;\n"
        sir_Text1 = sir_Text1 + "      2.1 Select language from list;\n"
        sir_Text1 = sir_Text1 + "      2.2 Optional, select 'Enhanced' control;\n"
        sir_Text1 = sir_Text1 + "3. Click on OCR button on toolbar;\n"
        sir_Text1 = sir_Text1 + "4. When the Enhanced control is activated, the \n"
        sir_Text1 = sir_Text1 + "   original and enhanced images are displayed in parallel.\n"
        sir_Text1 = sir_Text1 + "5. The recognized text will be displayed in right window;\n"
        sir_Text1 = sir_Text1 + "6. The recognized text will be save in 'OCR_Text.txt' file.\n"
        sir_Text1a = 2*"\n"+"Tesseract OCR is an open-source optical character recognition (OCR)\n"
        sir_Text1a = sir_Text1a + "engine that is used to extract text from images. In Python, pytesseract\n"
        sir_Text1a = sir_Text1a + "is a library that provides an interface to Tesseract's OCR engine.\n"
        sir_Text1a = sir_Text1a + "Tesseract is an open-source OCR engine developed by Google and\n"
        sir_Text1a = sir_Text1a + "is widely considered one of the most accurate OCR engines available.\n"
        sir_Text1a = sir_Text1a + "By matching patterns in the segmented areas, Tesseract recognizes\n"
        sir_Text1a = sir_Text1a + "individual characters through a combination of machine learning and\n"
        sir_Text1a = sir_Text1a + "conventional image processing approaches. In order to increase accuracy\n"
        sir_Text1a = sir_Text1a + "and handle many languages, it uses language models.'\n"
        sir_Text1a = sir_Text1a + "\n    Source:  https://builtin.com/articles/python-tesseract"

        self.text1 = wx.TextCtrl(self.panel1, -1,sir_Text1+sir_Text1a,
                size=(int(0.95*W/2), int(0.83*H)), pos=(5,5), style=wx.TE_MULTILINE)
        font1 = wx.Font(14, wx.ROMAN, wx.NORMAL, wx.BOLD)
        self.text1.SetFont(font1)
        self.text1.SetForegroundColour("Blue")

        self.sir_Text2="\nThe recognized text will displayed here."
        self.text2 = wx.TextCtrl(self.panel2, -1,self.sir_Text2,
            size=(int(0.95*W/2), int(0.83*H)), pos=(5,5),
            style=wx.TE_MULTILINE|wx.TE_PROCESS_ENTER)
        font2 = wx.Font(self.FONT, wx.DEFAULT, wx.NORMAL, wx.NORMAL)
        self.text2.SetFont(font2)

        mainsizer = wx.BoxSizer(wx.VERTICAL) # mainsizer.AddStretchSpacer()
        box = wx.BoxSizer(wx.HORIZONTAL)
        box.Add(self.panel1, 1, wx.EXPAND)
        box.Add(self.panel2, 1, wx.EXPAND)
        mainsizer.Add(box, 1, wx.EXPAND)
        main_panel.SetSizer(mainsizer)

        self._TOOLBAR() ; self.Center()

    def OnOpenImage(self,event):
        self.text2.SetValue("")
        wildcard = "JPG file (*.jpg)|*.jpg|" \
                "PNG file (*.png)|*.png|" \
```

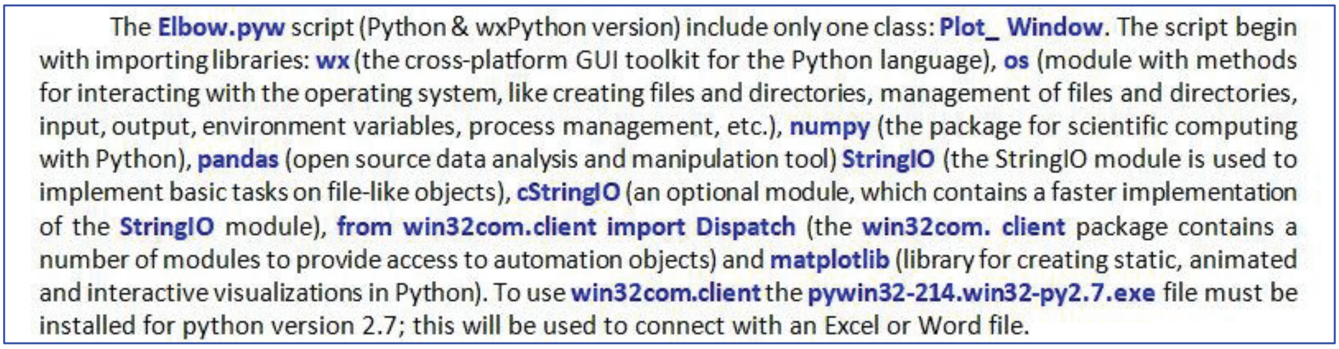
Listing 22.1 OCR.pyw – Optical Character Recognition – Python & wxPython version

```
                "TIF file (*.tif)|*.tif|" \
                "BMP file (*.bmp)|*.bmp"
        dialog = wx.FileDialog(None, "Open Image file", "", "", wildcard)
        if dialog.ShowModal() == wx.ID_CANCEL:
            return
        self.filename = dialog.GetPath().strip().upper()
        self.statusbar.SetStatusText(str(self.filename), 0)
        self.ShowImage()

    def ShowImage(self):
        img=mpimg.imread(self.filename)
        plt.title("Original image: "+self.filename)
        imgplot = plt.imshow(img)
        fig = plt.gcf()
        plt.tight_layout()
        plt.show()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour("white") ; self.toolbar.SetToolBitmapSize((24,24))

        self.toolbar.AddSeparator()

        img = wx.Image(os.getcwd()+'/_Icons/open.png', wx.BITMAP_TYPE_ANY)
        id_Open=90 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_Open , "Open", sel_bmp, shortHelp="Open image for character recognition")
        self.Bind(wx.EVT_TOOL, self.OnOpenImage, id=id_Open)

        self.Lang1=["Engleza","French", "German", "Italian", "Croatian", "Hungarian", "Serbian", "Romanian"]
        self.Lang2=["eng","fra", "deu", "ita", "hrv", "hun", "srp", "ron"]
        self.Lang= wx.Choice(self.toolbar, -1, size=(100, 50), choices = self.Lang1)
        x=self.Lang.FindString('Engleza')
        self.Lang.SetSelection(x)
        sir="https://tesseract-ocr.github.io/tessdoc/Data-Files-in-different-versions.html"
        self.Lang.SetToolTip("Select character recognition language\n"+sir)
        font = wx.Font(12, wx.ROMAN, wx.NORMAL, wx.BOLD)
        self.Lang.SetFont(font)
        self.Lang.SetForegroundColour("Blue")
        self.toolbar.AddControl(self.Lang)

        self.Enhanced = wx.CheckBox(self.toolbar, -1, "Enhanced")
        self.Enhanced.SetToolTip("Image enhancement involves refining an image to improve its quality")
        self.toolbar.AddControl(self.Enhanced)

        img = wx.Image(os.getcwd()+'/_Icons/Show.jpg', wx.BITMAP_TYPE_ANY)
        id_OCR=91 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        sir="https://pypi.org/project/pytesseract/"
        self.toolbar.AddTool( id_OCR , "OCR", sel_bmp, shortHelp="Start optical character recognition\n"+sir)
        self.Bind(wx.EVT_TOOL, self.OnOCR, id=id_OCR)

        img = wx.Image(os.getcwd()+'/_Icons/copy.png', wx.BITMAP_TYPE_ANY)
        id_COPY=92 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_COPY , "Clipboard", sel_bmp, shortHelp="Copy recognized text to Cliboard")
        self.Bind(wx.EVT_TOOL, self.OnClipboard, id=id_COPY)

        id_BTN = 88
        FONT_button = wx.Button(self.toolbar, id_BTN, "Font", size=(50, -1))
        FONT_button.SetToolTip("Select font type, size and color")
        self.toolbar.AddControl( FONT_button )
        self.Bind(wx.EVT_BUTTON, self.OnSelectFont, FONT_button )

        img = wx.Image(os.getcwd()+'/_Icons/exit.png', wx.BITMAP_TYPE_ANY)
        id_Exit=94 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_Exit , "Exit", sel_bmp, shortHelp="Exit application")
        self.Bind(wx.EVT_TOOL, self.OnExit, id=id_Exit)

        self.toolbar.Realize() ; self.toolbar.Show()

    def OnClipboard(self, evt):
        if self.filename=="":
            wx.MessageBox("Open an image file to recognize characters !",
                "ERROR", wx.OK | wx.ICON_INFORMATION)
```

Listing 22.1 OCR.pyw – Optical Character Recognition – Python & wxPython version

```
            return
        Put_Clipboard(self.Clipboard)
        wx.MessageBox("The recognize characters are placed in Clipboard !",
                "Info", wx.OK | wx.ICON_INFORMATION)

    def OnSelectFont(self, evt):
            data = wx.FontData()
            data.EnableEffects(True)
            data.SetColour(self.COLOR)
            dlg = wx.FontDialog(self, data)
            if dlg.ShowModal() == wx.ID_OK:
                data = dlg.GetFontData()
                font = data.GetChosenFont() ; self.text2.SetFont(font)
                colour = data.GetColour() ; self.text2.SetForegroundColour(colour)

    def OnOCR(self, evt):
        if self.filename=="":
            wx.MessageBox("Open an image file to recognize characters !",
                "ERROR", wx.OK | wx.ICON_INFORMATION)
            return

        language=self.Lang.GetStringSelection().strip()
        INDEX=self.Lang1.index(language)
        lng=self.Lang2[INDEX]

        ENHANCED=self.Enhanced.GetValue()
        if ENHANCED==True:
            IMG = cv2.imread(self.filename)
            img = cv2.detailEnhance(IMG, sigma_s=10, sigma_r=0.15)
            TITLE=self.filename+" => Enhanced details\n"
            titles = ['Original Image', 'Enhanced details']
            images = [IMG, img]
            for i in range(2):
                plt.subplot(1,2,i+1),plt.imshow(images[i])
                plt.title(titles[i])
                plt.xticks([]),plt.yticks([])
            plt.show()
        else:
            TITLE="Converted file: " + str(self.filename)+"\n"
            img = cv2.imread(self.filename)
        self.text2.SetValue(TITLE)
        custom_config = r'--oem 3 --psm 6 lang=lng' # Adding custom options
        TEXT=pytesseract.image_to_string(img, config=custom_config)
        TEXT = TEXT.encode('ascii', 'ignore').decode()

        N_words = len(TEXT.split())
        msj="Words number : " + str(N_words)
        msj=msj+" ; "+"Characters number : " + str(len(TEXT))
        msj=msj+" ; "+"Language conversion: " + language + 2*"\n"

        self.Clipboard=TITLE+msj+TEXT
        self.text2.SetValue(self.Clipboard)
        with open(os.getcwd()+"\\"+"OCR_Text.txt","w") as output_file:
            output_file.write(TITLE+TEXT)
        self.statusbar.SetStatusText(msj, 1)

    def OnExit(self, evt):
        self.Close()

app = wx.App()
tit="Optical Character Recognition with Tesseract's OCR engine & pytesseract"
frame = OCR(None, -1, tit)
frame.Show()
app.MainLoop()
```

Figure 22.2 ÷ 22.15 show images in different languages and recognised texts. Of course, what I have presented does not cover all the possibilities offered by **Tesseract's OCR** engine & **pytesseract l**ibraries. Reference [**22.3**] offers many more of these possibilities.

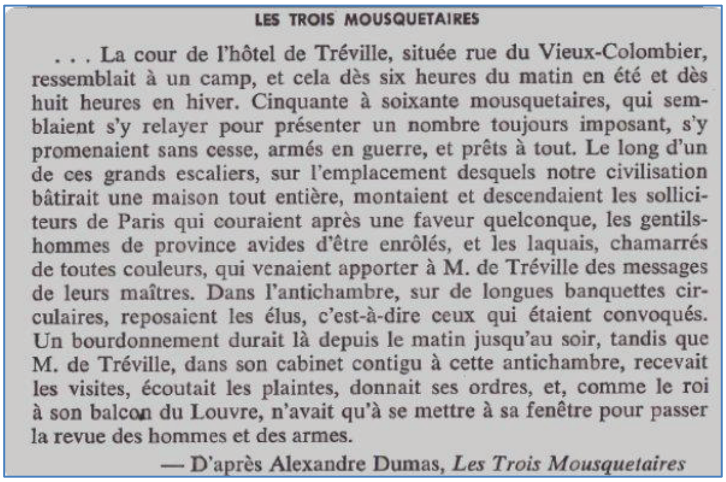


The **Elbow.pyw** script (Python & wxPython version) include only one class: **Plot_ Window**. The script begin with importing libraries: **wx** (the cross-platform GUI toolkit for the Python language), **os** (module with methods for interacting with the operating system, like creating files and directories, management of files and directories, input, output, environment variables, process management, etc.), **numpy** (the package for scientific computing with Python), **pandas** (open source data analysis and manipulation tool) **StringIO** (the StringIO module is used to implement basic tasks on file-like objects), **cStringIO** (an optional module, which contains a faster implementation of the **StringIO** module), **from win32com.client import Dispatch** (the **win32com. client** package contains a number of modules to provide access to automation objects) and **matplotlib** (library for creating static, animated and interactive visualizations in Python). To use **win32com.client** the **pywin32-214.win32-py2.7.exe** file must be installed for python version 2.7; this will be used to connect with an Excel or Word file.

Figure 22.2 Test_English1.jpg image

```
E:\BOOK_FILES\OCR\TEST_ENGLEZA1.JPG => Enhanced details
Words number : 153 ; Characters number : 1085 ; Language conversion: Engleza

The Elbow.pyw script (Python & wxPython version) include only one class: Plot_ Window. The script begin
with importing libraries: wx (the cross-platform GUI toolkit for the Python language), os (module with methods
for interacting with the operating system, like creating files and directories, management of files and directories,
input, output, environment variables, process management, etc.), numpy (the package for scientific computing
with Python), pandas (open source data analysis and manipulation tool) StringlO (the StringlO module is used to
implement basic tasks on file-ike objects), eStringlO (an optional module, which contains a faster implementation
of the StringlO module), from win32com.client import Dispatch (the win32com. client package contains a
number of modules to provide access to automation objects) and matplotlib (library for creating static, animated
and interactive visualizations in Python). To use win32com.client the pywin32-214.win32-py2.7.exe file must be
installed for python version 2.7; this will be used to connect with an Excel or Word file.
```

Figure 22.3 Text recognized from **Test_ English1.jpg** image

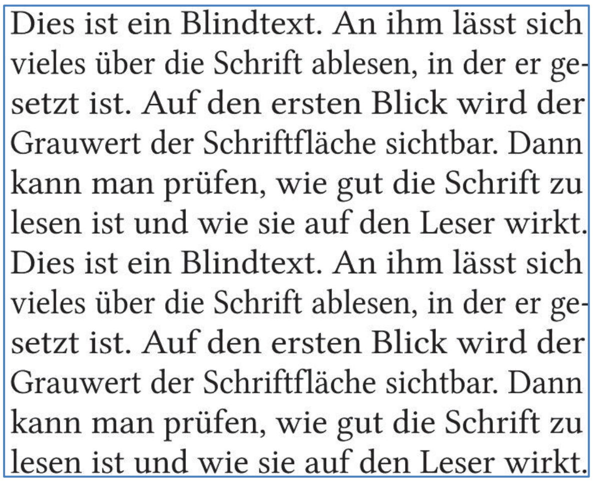


LES TROIS MOUSQUETAIRES

. . . La cour de l'hôtel de Tréville, située rue du Vieux-Colombier, ressemblait à un camp, et cela dès six heures du matin en été et dès huit heures en hiver. Cinquante à soixante mousquetaires, qui semblaient s'y relayer pour présenter un nombre toujours imposant, s'y promenaient sans cesse, armés en guerre, et prêts à tout. Le long d'un de ces grands escaliers, sur l'emplacement desquels notre civilisation bâtirait une maison tout entière, montaient et descendaient les solliciteurs de Paris qui couraient après une faveur quelconque, les gentilshommes de province avides d'être enrôlés, et les laquais, chamarrés de toutes couleurs, qui venaient apporter à M. de Tréville des messages de leurs maîtres. Dans l'antichambre, sur de longues banquettes circulaires, reposaient les élus, c'est-à-dire ceux qui étaient convoqués. Un bourdonnement durait là depuis le matin jusqu'au soir, tandis que M. de Tréville, dans son cabinet contigu à cette antichambre, recevait les visites, écoutait les plaintes, donnait ses ordres, et, comme le roi à son balcon du Louvre, n'avait qu'à se mettre à sa fenêtre pour passer la revue des hommes et des armes.

— D'après Alexandre Dumas, *Les Trois Mousquetaires*

Figure 22.4 Test_French1.jpg image



Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt.

Figure 22.5 Test_German1.jpg image

```
E:\BOOK_FILES\OCR\TEST_FRENCH1.JPG => Enhanced details
Words number : 194 ; Characters number : 1202 ; Language conversion:
French

LES TROIS MOUSQUETAIRES

... La cour de Phtel de Trville, situe rue du Vieux-Colombier,
ressemblait A un camp, et cela ds six heures du matin en t et ds
huit heures en hiver. Cinquante 4 soixante mousquetaires, qui sem-
blaient sy relayer pour prsenter un nombre toujours imposant, s'y
promenaient sans cesse, arms en guerre, et prts 4 tout. Le long dun
de ces grands escaliers, sur l'emplacement desquels notre civilisation
batirait une maison tout entigre, montaient et descendaient les sollici-
```

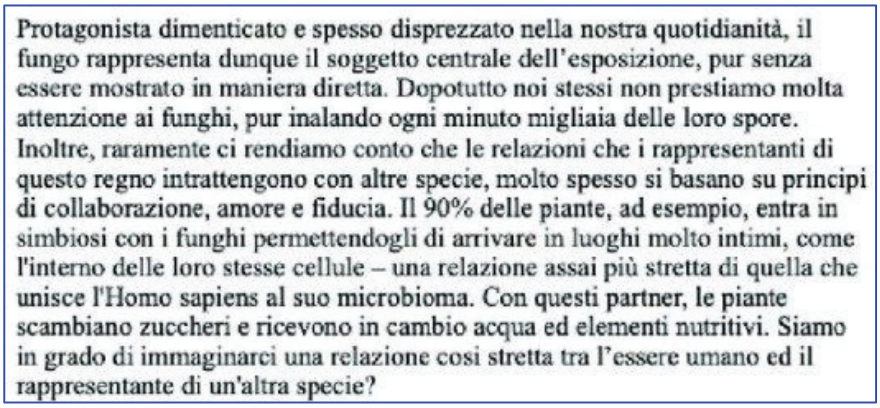
```
E:\BOOK_FILES\OCR\TEST_GERMAN1.JPG => Enhanced
details
Words number : 94 ; Characters number : 514 ; Language
conversion: German

Dies ist ein Blindtext. An ihm lasst sich
vieles tiber die Schrift ablesen, in der er ge-
setzt ist. Auf den ersten Blick wird der
Grauwert der Schriftflache sichtbar. Dann
kann man priifen, wie gut die Schrift zu
lesen ist und wie sie auf den Leser wirkt.
Dies ist ein Blindtext. An ihm lasst sich
vieles itber die Schrift ablesen, in der er ge-
```

```
teurs de Paris qui couraient aprs une faveur quelconque, les gentils-
hommes de province avides dtre enrdls, et les laquais, chamarrs
de toutes couleurs, qui venaient apporter 4 M. de Trville des messages
de leurs maitres. Dans l'antichambre, sur de longues banquettes cir-
culaires, reposaicnt les lus, cest-d-dire ceux qui taient convoqus.
Un bourdonnement durait 14 depuis le matin jusquau soir, tandis que
M. de Trville, dans son cabinet contigu A cette antichambre, recevait
les visites, coutait les plaintes, donnait ses ordres, et, comme le roi
A son balcon du Louvre, navait quA se mettre 4 sa fentre pour passer
la reyue des hommes et des armes.

Daprs Alexandre Dumas, Les Trois Mousquetaires
```

Figure 22.6 Text recognized from **Test_French1.jpg** image

```
setzt ist. Auf den ersten Blick wird der
Grauwert der Schriftflache sichtbar. Dann
kann man priifen, wie gut die Schrift zu
lesen ist und wie sie auf den Leser wirkt.
```

Figure 22.7 Text recognized from **Test_German1.jpg** image

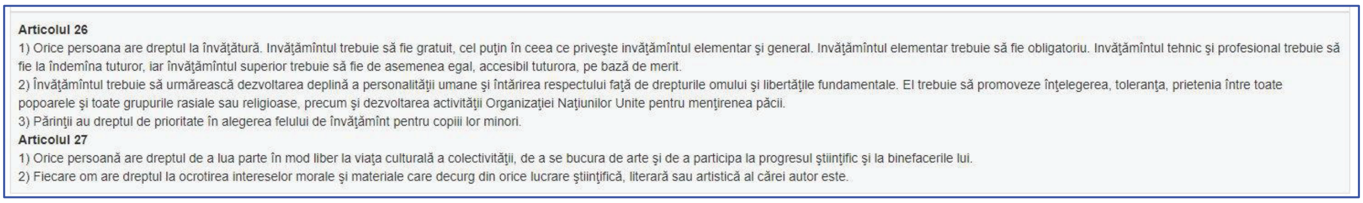


Protagonista dimenticato e spesso disprezzato nella nostra quotidianità, il fungo rappresenta dunque il soggetto centrale dell'esposizione, pur senza essere mostrato in maniera diretta. Dopotutto noi stessi non prestiamo molta attenzione ai funghi, pur inalando ogni minuto migliaia delle loro spore. Inoltre, raramente ci rendiamo conto che le relazioni che i rappresentanti di questo regno intrattengono con altre specie, molto spesso si basano su principi di collaborazione, amore e fiducia. Il 90% delle piante, ad esempio, entra in simbiosi con i funghi permettendogli di arrivare in luoghi molto intimi, come l'interno delle loro stesse cellule – una relazione assai più stretta di quella che unisce l'Homo sapiens al suo microbioma. Con questi partner, le piante scambiano zuccheri e ricevono in cambio acqua ed elementi nutritivi. Siamo in grado di immaginarci una relazione così stretta tra l'essere umano ed il rappresentante di un'altra specie?

Figure 22.8 Test_Italian1.jpg image

```
Converted file: E:\BOOK_FILES\OCR\TEST_ITALIAN1.JPG
Words number : 142 ; Characters number : 951 ; Language conversion: German

Protagonista dimenticato e spesso disprezzato nella nostra quotidianita, il
fungo rappresenta dunque il soggetto centrale dell esposizione, pur senza
essere mostrato in maniera diretta, Depotutto noi stessi non prestiamo molta
attenzione ai funghi, pur inalando ogni minuto migliaia delle loro spore.
Inoltre, raramente ci rendiamo conto che le relazioni che i rappresentanti di
questo regno intrattengono con altre specie, molto spesso si basano su principi
di collaborazione, amore e fiducia. Il 90% delle piante, ad esempio, entra in
simbiosi con i funghi permettendogli di arrivare in luoghi molto intimi, come
interno delle loro stesse cellule  una relazione assai pid stretta di quella che
unisce l'Homo sapiens al suo microbioma. Con questi partner, le piante
scambiano zuccheri e ricevono in cambio acqua ed elementi nuttitivi. Siamo
in grado di immaginarci una relazione cosi stretta tra lessere umano ed il
tappresentante di unaltra specie?
```

Figure 22.9 Text recognized from **Test_Italian1.jpg** image



Articolul 26

1) Orice persoana are dreptul la învăţătură. Invăţămîntul trebuie să fie gratuit, cel puţin în ceea ce priveşte invăţămîntul elementar şi general. Invăţămîntul elementar trebuie să fie obligatoriu. Invăţămîntul tehnic şi profesional trebuie să fie la îndemîna tuturor, iar învăţămîntul superior trebuie să fie de asemenea egal, accesibil tuturora, pe bază de merit.

2) Învăţămîntul trebuie să urmărească dezvoltarea deplină a personalităţii umane şi întărirea respectului faţă de drepturile omului şi libertăţile fundamentale. El trebuie să promoveze înţelegerea, toleranţa, prietenia între toate popoarele şi toate grupurile rasiale sau religioase, precum şi dezvoltarea activităţii Organizaţiei Naţiunilor Unite pentru menţirenea păcii.

3) Părinţii au dreptul de prioritate în alegerea felului de învăţămînt pentru copiii lor minori.

Articolul 27

1) Orice persoană are dreptul de a lua parte în mod liber la viaţa culturală a colectivităţii, de a se bucura de arte şi de a participa la progresul ştiinţific şi la binefacerile lui.

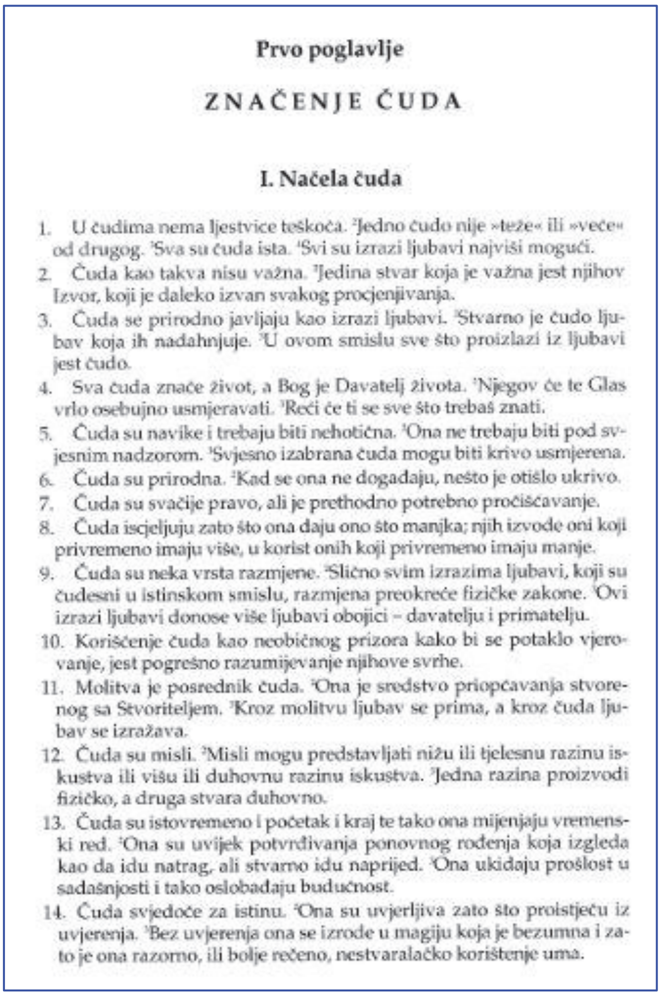
2) Fiecare om are dreptul la ocrotirea intereselor morale şi materiale care decurg din orice lucrare ştiinţifică, literară sau artistică al cărei autor este.

Figure 22.10 Test_Romanian1.jpg image

```
Converted file: E:\BOOK_FILES\OCR\TEST_ROMANIAN1.JPG
Words number : 176 ; Characters number : 1198 ; Language conversion: Romanian

Articolul 26

1) Orice persoana are dreptul la invataturd. Invatamintul trebuie sa fie gratuit, cel putin
in ceea ce priveste invatmintul elementar si general. Invatmintul elementar trebuie s fie
obligatoriu. Invat4mintul tehnic si profesional trebuie s&
```

```
fle fa indemina tuturor, iar invatamintul superior trebuie sd fie de asemenea egal,
accesibil tuturora, pe baz de merit.

2) invatamintul trebuie s urmareasca dezvoltarea deplina a personalitatii umane si
intarirea respectului fata de drepturile omului si libertatile fundamentale. El trebuie s4
promoveze intelegerea, toleranta, prietenia intre toate
Popoarele si toate grupurile rasiale sau religioase, precum si dezvoltarea activitatii
Organizatiel Natiunilor Unite pentru mentirenea paci.

3) P&rintji au dreptul de prioritate in alegerea felului de invatamint pentru copill lor
minori

Articolul 27

1) Orice persoand are dreptul de a lua parte in mod liber la viata cutturala a colectivitti,
de a se bucura de arte si de a participa la progresul stiintific si la binefacerile Iu

2) Fiecare om are dreptul la ocrotirea intereselor morale si materiale care decurg din
orice lucrare stiintificd, literara sau artistic al c4rei autor este.
```

Figure 22.11 Text recognized from **Test_Romanian1.jpg** image



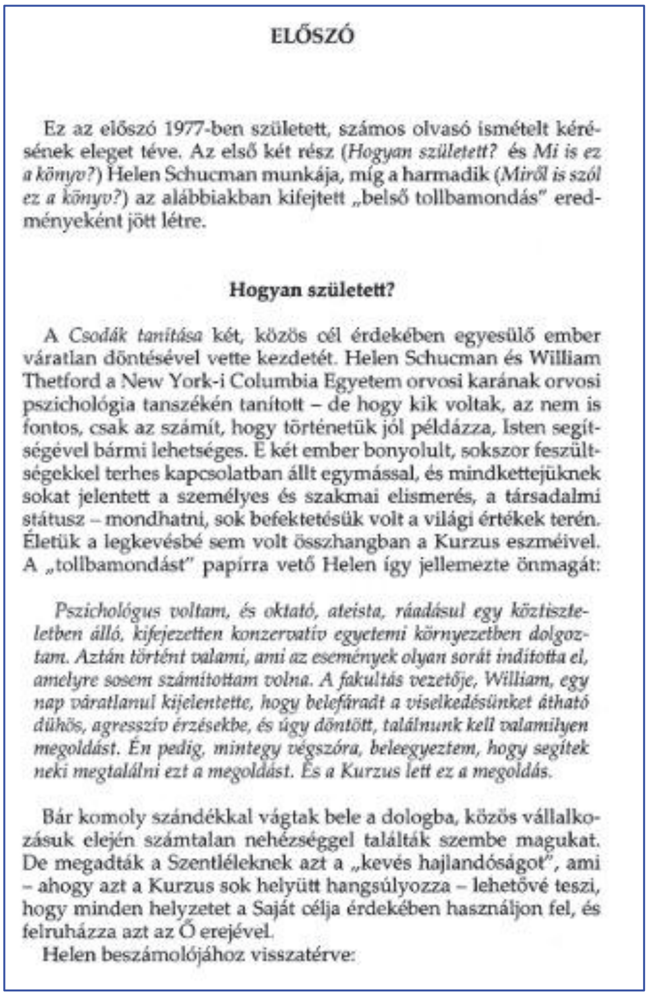
Prvo poglavlje

ZNAČENJE ČUDA

I. Načela čuda

1. U čudima nema ljestvice teškoća. [2]Jedno čudo nije »teže« ili »veće« od drugog. [3]Sva su čuda ista. [4]Svi su izrazi ljubavi najviši mogući.
2. Čuda kao takva nisu važna. [2]Jedina stvar koja je važna jest njihov Izvor, koji je daleko izvan svakog procjenjivanja.
3. Čuda se prirodno javljaju kao izrazi ljubavi. [2]Stvarno je čudo ljubav koja ih nadahnjuje. [3]U ovom smislu sve što proizlazi iz ljubavi jest čudo.
4. Sva čuda znače život, a Bog je Davatelj života. [2]Njegov će te Glas vrlo osebujno usmjeravati. [3]Reći će ti se sve što trebaš znati.
5. Čuda su navike i trebaju biti nehotična. [2]Ona ne trebaju biti pod svjesnim nadzorom. [3]Svjesno izabrana čuda mogu biti krivo usmjerena.
6. Čuda su prirodna. [2]Kad se ona ne događaju, nešto je otišlo ukrivo.
7. Čuda su svačije pravo, ali je prethodno potrebno pročišćavanje.
8. Čuda iscjeljuju zato što ona daju ono što manjka; njih izvode oni koji privremeno imaju više, u korist onih koji privremeno imaju manje.
9. Čuda su neka vrsta razmjene. [2]Slično svim izrazima ljubavi, koji su čudesni u istinskom smislu, razmjena preokreće fizičke zakone. [3]Ovi izrazi ljubavi donose više ljubavi obojici – davatelju i primatelju.
10. Korišćenje čuda kao neobičnog prizora kako bi se potaklo vjerovanje, jest pogrešno razumijevanje njihove svrhe.
11. Molitva je posrednik čuda. [2]Ona je sredstvo priopćavanja stvorenog sa Stvoriteljem. [3]Kroz molitvu ljubav se prima, a kroz čuda ljubav se izražava.
12. Čuda su misli. [2]Misli mogu predstavljati nižu ili tjelesnu razinu iskustva ili višu ili duhovnu razinu iskustva. [3]Jedna razina proizvodi fizičko, a druga stvara duhovno.
13. Čuda su istovremeno i početak i kraj te tako ona mijenjaju vremenski red. [2]Ona su uvijek potvrđivanja ponovnog rođenja koja izgleda kao da idu natrag, ali stvarno idu naprijed. [3]Ona ukidaju prošlost u sadašnjosti i tako oslobađaju budućnost.
14. Čuda svjedoče za istinu. [2]Ona su uvjerljiva zato što proistječu iz uvjerenja. [3]Bez uvjerenja ona se izrode u magiju koja je bezumna i zato je ona razorno, ili bolje rečeno, nestvaralačko korištenje uma.

Figure 22.12 Test_Croatian1.jpg image



ELŐSZÓ

Ez az előszó 1977-ben született, számos olvasó ismételt kérésének eleget téve. Az első két rész (*Hogyan született?* és *Mi is ez a könyv?*) Helen Schucman munkája, míg a harmadik (*Miről is szól ez a könyv?*) az alábbiakban kifejtett „belső tollbamondás" eredményeként jött létre.

Hogyan született?

A *Csodák tanítása* két, közös cél érdekében egyesülő ember váratlan döntésével vette kezdetét. Helen Schucman és William Thetford a New York-i Columbia Egyetem orvosi karának orvosi pszichológia tanszékén tanított – de hogy kik voltak, az nem is fontos, csak az számít, hogy történetük jól példázza, Isten segítségével bármi lehetséges. E két ember bonyolult, sokszor feszültségekkel terhes kapcsolatban állt egymással, és mindkettejüknek sokat jelentett a személyes és szakmai elismerés, a társadalmi státusz – mondhatni, sok befektetésük volt a világi értékek terén. Életük a legkevésbé sem volt összhangban a Kurzus eszméivel. A „tollbamondást" papírra vető Helen így jellemezte önmagát:

> *Pszichológus voltam, és oktató, ateista, ráadásul egy köztiszteletben álló, kifejezetten konzervatív egyetemi környezetben dolgoztam. Aztán történt valami, ami az események olyan sorát indította el, amelyre sosem számítottam volna. A fakultás vezetője, William, egy nap váratlanul kijelentette, hogy belefáradt a viselkedésünket átható dühös, agresszív érzésekbe, és úgy döntött, találnunk kell valamilyen megoldást. Én pedig, mintegy végszóra, beleegyeztem, hogy segítek neki megtalálni ezt a megoldást. És a Kurzus lett ez a megoldás.*

Bár komoly szándékkal vágtak bele a dologba, közös vállalkozásuk elején számtalan nehézséggel találták szembe magukat. De megadták a Szentléleknek azt a „kevés hajlandóságot", ami – ahogy azt a Kurzus sok helyütt hangsúlyozza – lehetővé teszi, hogy minden helyzetet a Saját célja érdekében használjon fel, és felruházza azt az Ő erejével.

Helen beszámolójához visszatérve:

Figure 22.13 Test_Hungarian1.jpg image

```
Converted file: E:\BOOK_FILES\OCR\TEST_CROATIAN1.JPG
Words number : 272 ; Characters number : 1334 ; Language
conversion: Croatian

Prvo poglavije
ZNACENJE CUDA
L Nagela duds
```

```
Converted file: E:\BOOK_FILES\OCR\TEST_HUNGARIAN1.JPG
Words number : 244 ; Characters number : 1538 ; Language
conversion: Hungarian

H16820

Ex az eld 1977-ben sziletot,szimos olvas ismtle ki
sink lp te, Al tn ag sale? MI
```

```
1. Ueudima nama svc thts no cdo ie tte vee
od drugeg. Sap Cet vious ub aj og
2.Goa ta taka als vada, Teina svat kj vata fest iho
ov je dahon vag prepay
3. Gat se ptrino jvina kao ter ibn. Stat je Co
tov keh nada ovom smi eve So pro Iba
etendo.
Sea cada znate vot 3 Bog e Davatel vot, "Neg de te las
rigescbujn winernall-Recet wwe so eben
"Cuca su navike tcbeju bi ncn Ona ne reba bi po
ienim nadzorm Sijesnoabrana Cuda mg evo serena
0 Cada su pons Kad ona ne dogada neta oti kv.
7 Gada su vate rave, lj pthc pti proses ane
Cora cea oon don so alah vod
preremeno trojuise ute nih kl prvremena ia er
oF Gada una vt came "Sco win zai bo, es
des usta sms raza preokct ice akone: Ox
{tra jubavi donee ie ubavebope = date primate.
10. Keienje ed Ka nobienog pista ako Bs potalo 0
anf jst Pogo rarumjovanp nove see
11, Mais esr Cs. J sredsv prapavanj von
Kegs Soria Kror motu ub se primar tre ude
12 Cada su mi. Mik mogu presi i ii lean rain i
ust iui duboviocne kos, Jodn rina proved
aetna drug stare duhovno.
15, Cada pu stvremenal posta kn tako oa mien eens
tise Ona au cvles porvane ponomog sets foe peda
tao dau nara, stearo sat ape, Ga lids prot
SSusnt ko flats Dnt
1 Gadnsjece 2a ists Ona sev zt Ho pre i
Toforefa er urea nase iro sg ko eben
tofeone razon ibe mono, retro korn
```

Figure 22.14 Text recognized from **Test_Croatian1.jpg** image

```
"nyo? Helen Schucrnan munkaj, nia harmadli (Mir iss)
za kinyo?) az alabbiakban bite ,bels6tllbamonds"ered-
mnyeknt jt te

Hogyan szleteu?

A Coodik tavtisa hit, Kas fl rdehbon egyesilS ember
wtatlan dootsvel vette kezdett. Helen Schucman s William
Thetford a New York-i Columbia Egyetem orvosi Karnak orvosi
psvicholgia tanszken tanitott~ de hogy kik volta, az nem is
fontos, sak az zim hogy tortnetk jl peldazza inten segi-
scgeve bir iehetwges, Eke ember bonyolult,sokszor fest
Sczehkel teres Kapeselaiban st egymassal, s mindketjuknck
Skat jelentot a szemlyes Gs seakmai elismers, a tirsadalmi
Stitusz - mondhetn, sk beektetistk volta vig kek tern.
let a lekevsb som volt dsszhangban a Kurzus eszmeivel
A tolbamondist papinra vet Helen igy fellemezte Grumagat

Pscholgus volta,  oka, leit, rads egy hice.
lee al, ejection lonzcratioegyletkirnyecelben doles:
tam. Astin rt alo a ex emerge lyn sora indo
tele soem stom cara A flats ext Wiliam,
ap diratlana ijletete, hogy beeiadt a viselkedistinkt that
th agregto estate, & gy dnt, alu el eaayen
epi En pi mint eis, ile hy et
nk meg ext 8 egos. Esa Karas let xo megldis.

Bir komoly szdndkkal vglak bele a dologha, kis vilalho-
zsuk clef sedmtalan nchersdggel tlt szembe maguht
De megadtak a Szentileknek azt a kevshojandsagot, ami
~ahogy azta Kurzus sok heat hangsiyozza ~lehetOv fszi,
hogy inden helyzott a Sot ea erdckaben hasenljon fl,
felruhizza sat az  revel

Helen beszimolgjhog visszatrve:
```

Figure 22.15 Text recognized from **Test_Hungarian1.jpg** image

23. Open 3D

"Open3D is an open-source library that supports rapid development of software that deals with 3D data. The Open3D frontend exposes a set of carefully selected data structures and algorithms in both C++ and Python. The backend is highly optimized and is set up for parallelization. Open3D was developed from a clean slate with a small and carefully considered set of dependencies. It can be set up on different platforms and compiled from source with minimal effort. The code is clean, consistently styled, and maintained via a clear code review mechanism. Open3D has been used in a number of published research projects and is actively deployed in the cloud" [23.1], [23.2].

The chapter exemplifies a simple mesh & point cloud script through **Open_3D_wx_Python.pyw** saved in **Open 3D** folder, with interface from **Figure 23.1.**

A **mesh** is a three-dimensional model used to create 3D objects & environments and is made of points, lines and surfaces.

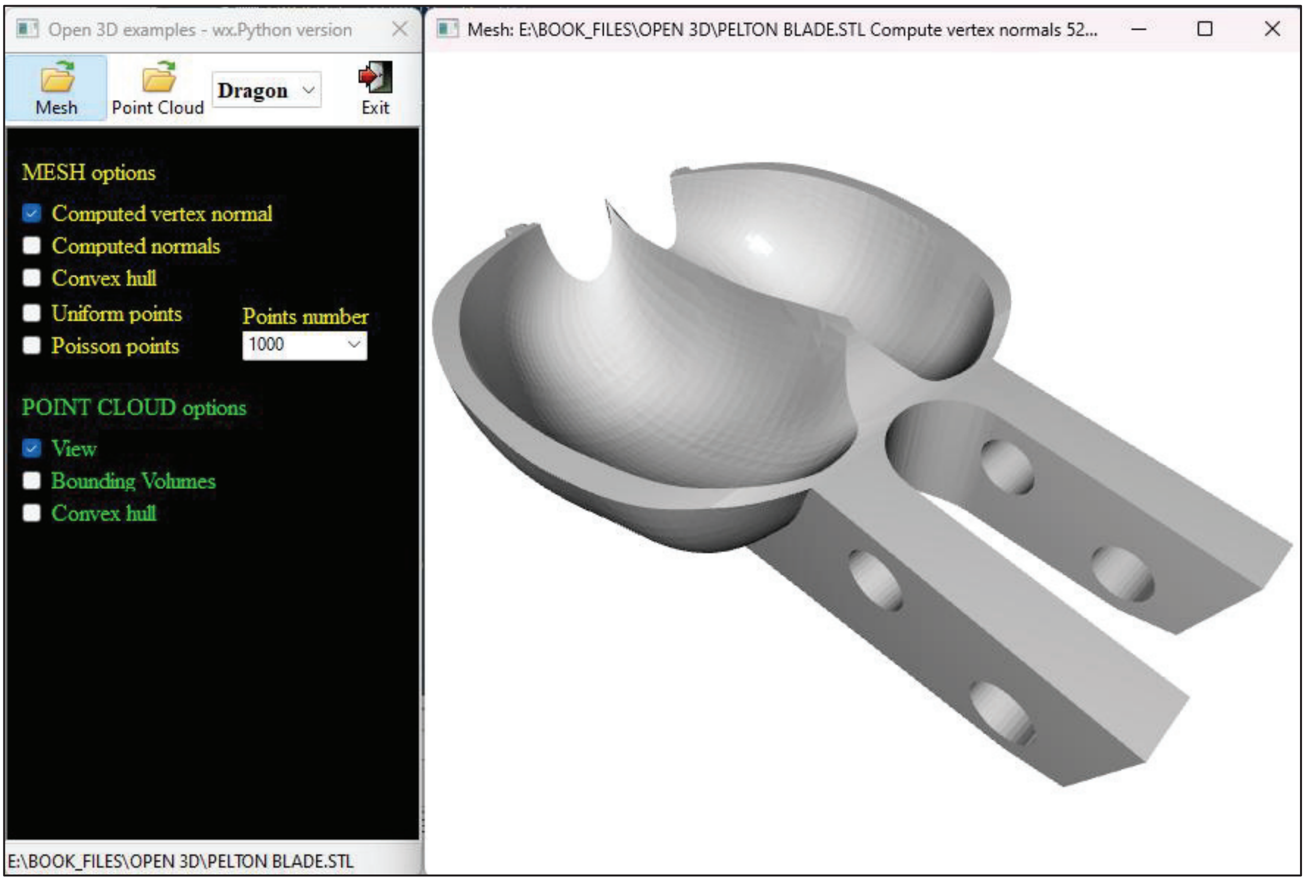
The **STL** file format approximates the surfaces of a solid model with triangles.

A **point cloud** is a discrete set of data points in space. The points can represent a 3D shape or object. Each point location has its own set of Cartesian coordinates.

The **convex hull** of a mesh or point cloud is the smallest convex set that contains all points.

The function to **Uniform points** or **Poisson points from** convert mesh into a point cloud.

The left side of the interface contain the toolbar and the some options for mesh & point cloud. The right side of the interface display the 3D geometry in concordance with selected options.


Open 3D examples - wx.Python version
Mesh
Point Cloud
Dragon
Exit
Mesh: E:\BOOK_FILES\OPEN 3D\PELTON BLADE.STL Compute vertex normals 52...

MESH options

☑ Computed vertex normal
□ Computed normals
□ Convex hull
□ Uniform points
□ Poisson points

Points number: 1000

POINT CLOUD options

☑ View
□ Bounding Volumes
□ Convex hull

E:\BOOK_FILES\OPEN 3D\PELTON BLADE.STL

Figure 23.1 The **Open_3D_wx_Python.pyw** interface

❶ To install **Open 3D** go to [23.3], write **Open 3D** in **Search** control and select **ope3d 0.18.0** version to download **open3d-0.18.0-cp311-cp311-win_amd64.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip3.11 install open3d-0.18.0-cp311-cp311-win_amd64.whl
```

The 3D geometry can be rotated in right window with left mouse pressed, panned with right mouse pressed and zoomed with central mouse wheel. The right window can be redimensioned with left mouse placed on the window border. The right window and 3D geometry can be redimensioned with left mouse placed on corner window.

Listing 23.1 displays the **Python & wxPython** version of script, **Open_3D_wx_Python.pyw**.

The line **#!/usr/bin/env** placed at the top script ensures that the interpreter will use the first installed version on the environment's **$PATH**. The **Open_3D_wx_Python.pyw** script (Python 3.11.9 & wxPython version) include one class: **OPEN_3D**. The script begin with importing libraries (lines 4÷7).

The **OPEN_3D** class create the script interface and define functions to display 3D geometry:

- **__init__** Calculate the width **W** and heigth **H** of the display, the frame dimensions (**Wframe**, **Hframe**) and left/ top position (**poz_WND** / **poz_HND**); define the **frame** (lines 12÷13), the **panel** (lines 15÷16) and **statusbar** (line 17÷18), line 20 define the right window dimensions (**W_width**, **H_height**) and left/ top position (**left**/ **top**); lines 22, 23 call **_TOOLBAR** and **_CONTROLS** functions.
- **_TOOLBAR** Lines 26, 27 define the **toolbar** and the background. Lines 29÷31 define **Mesh** control and **Bind** with **OnOpen_MESH** class function (line 32) to open an mesh file. Lines 34÷37 define **Point Cloud** control and **Bind** with **OnOpen_POINT_CLOUD** class function (line 38) to open an point cloud file. Lines 40÷49 define a choice control **self.Opt_Demo** with some **STL** files that can be loaded as demo; this control is connected through **Bind** command (line 48) with **OnDemo** class function. Lines 51÷53 define **Exit** icon connected through **Bind** command (line 54) with **OnExit**class function.
- **_CONTROLS** This class function create, through **wx.CheckBox** command, controls to select options for mesh and point cloud (placed in left interface window). The **lst_n_pts** control is created to select the points number used for **Uniform points** or **Poisson points** options.
- **Open_FILE** This function open a dialog to select an mesh or point cloud file in **self.filename** name. The **wildcard** is different for mesh or point cloud.
- **OnOpen_MESH** This function call on **OPEN_MESH** function with **Mesh** option.
- **OnDemo** This function call on **OPEN_MESH** function with **STL** option.
- **OPEN_MESH_STL** Open a mesh or a **STL** file for processing; lines 159÷163 display an info about selected file; lines 165÷202 display the 3D mesh/STL geometry according with the mesh options selected from the left window.
- **OnOpen_POINT_CLOUD** Open a point cloud file for processing; lines 212÷215 display an info about selected file; lines 217÷235 display the point cloud geometry according with the point cloud options selected from the left window.
- **OnExit** Close the application.

Listing 23.1 Open_3D_wx_Python.pyw – Open 3D – Python & wxPython version

```
224.  #! /usr/bin/env python
225.  #coding=utf-8
226.
227.  import wx
228.  import os
229.  import numpy as np
230.  import open3d as o3d
231.
232.  class OPEN_3D(wx.Frame):
233.      def __init__(self, parent, ID, title):
234.          W,H = wx.DisplaySize() ; Wframe=300 ; Hframe=600 ; poz_WND=100 ; poz_HND=int(H-Hframe)/2
235.          self.frame=wx.Frame.__init__(self, parent, ID, title, size=(Wframe, Hframe), pos=(poz_WND,poz_HND),
236.              style=wx.DEFAULT_FRAME_STYLE ^ ( wx.RESIZE_BORDER | wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
237.
238.          self.panel = wx.Panel(self,-1,style=wx.SUNKEN_BORDER)
239.          self.panel.SetBackgroundColour("Black")
240.          self.statusbar = self.CreateStatusBar() # Create the status bar
```

Listing 23.1 Open_3D_wx_Python.pyw – Open 3D – Python & wxPython version

```
        self.statusbar.SetFieldsCount(1)

        self.W_width=600 ; self.H_height=Hframe-50 ; self.left=poz_WND+Wframe ; self.top=int(poz_HND)

        self._TOOLBAR()
        self._CONTROLS()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour("white")

        img = wx.Image(os.getcwd()+'/_Icons/open.png', wx.BITMAP_TYPE_ANY)
        id_Open_MESH=90 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_Open_MESH , "Mesh", sel_bmp, shortHelp="Open MESH file: .ply .stl .obj")
        self.Bind(wx.EVT_TOOL, self.OnOpen_MESH, id=id_Open_MESH)

        img1 = wx.Image(os.getcwd()+'/_Icons/open.png', wx.BITMAP_TYPE_ANY)
        id_Open_PCD=91 ; sel_bmp1=wx.Bitmap(img1.Scale(24,24))
        sh="Open Point Cloud file: .xyz .xyzn .xyzrgb .ply .pcd .pts"
        self.toolbar.AddTool( id_Open_PCD , "Point Cloud", sel_bmp1, shortHelp=sh)
        self.Bind(wx.EVT_TOOL, self.OnOpen_POINT_CLOUD, id=id_Open_PCD)

        lst_Demo=["Dragon"]
        for i in range(1, 12):
            lst_Demo.append("STL "+str(i))
        self.Opt_Demo= wx.Choice(self.toolbar, -1, size=(75, 50), choices = lst_Demo)
        self.Opt_Demo.SetSelection(0)
        self.Opt_Demo.SetToolTip("Select demo STL file to load !")
        font = wx.Font(11, wx.ROMAN, wx.NORMAL, wx.BOLD) ; self.Opt_Demo.SetFont(font)
        self.Opt_Demo.SetForegroundColour("Blue")
        self.Bind(wx.EVT_CHOICE, self.OnDemo, self.Opt_Demo)
        self.toolbar.AddControl(self.Opt_Demo)

        img = wx.Image(os.getcwd()+'/_Icons/exit.png', wx.BITMAP_TYPE_ANY)
        id_Exit=94 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_Exit , "Exit", sel_bmp, shortHelp="Exit application")
        self.Bind(wx.EVT_TOOL, self.OnExit, id=id_Exit)

        self.toolbar.Realize() ; self.toolbar.Show()

    def _CONTROLS(self):
        LEFT=10
        font = wx.Font(12, wx.ROMAN, wx.NORMAL, wx.NORMAL)

        self.st_MESH_options=wx.StaticText(self.panel, -1, 'MESH options', pos=(LEFT,20), size=(140,-1))
        self.st_MESH_options.SetForegroundColour('Yellow') ; self.st_MESH_options.SetFont(font)

        self.chk_Computed_Vertex_Normals= wx.CheckBox(self.panel, -1, " Computed vertex normals ",
                pos=(LEFT,50), size=(170,-1))
        self.chk_Computed_Vertex_Normals.SetForegroundColour('Yellow')
        self.chk_Computed_Vertex_Normals.SetFont(font)
        self.chk_Computed_Vertex_Normals.SetValue(True)

        self.chk_Computed_Normals= wx.CheckBox(self.panel, -1, " Computed normals ", pos=(LEFT,72), size=(170,-1))
        self.chk_Computed_Normals.SetForegroundColour('Yellow') ; self.chk_Computed_Normals.SetFont(font)

        self.chk_Convex_hull = wx.CheckBox(self.panel, -1, " Convex hull ", pos=(LEFT,94), size=(130,-1))
        self.chk_Convex_hull.SetForegroundColour('Yellow') ; self.chk_Convex_hull.SetFont(font)
        sir="The convex hull of a point cloud is the smallest convex set that contains all points."
        self.chk_Convex_hull.SetToolTip(sir)

        self.chk_Uniform_points= wx.CheckBox(self.panel, -1, " Uniform points ", pos=(LEFT,118), size=(130,-1))
        self.chk_Uniform_points.SetForegroundColour('Yellow') ; self.chk_Uniform_points.SetFont(font)
        sir="Number of sampling points to convert mesh into a point cloud"
        self.chk_Uniform_points.SetToolTip(sir)

        self.chk_Poisson_points= wx.CheckBox(self.panel, -1, " Poisson points ", pos=(LEFT,140), size=(130,-1))
        self.chk_Poisson_points.SetForegroundColour('Yellow') ; self.chk_Poisson_points.SetFont(font)
        self.chk_Poisson_points.SetToolTip(sir)

        self.st_n_pts=wx.StaticText(self.panel, -1, 'Points number', pos=(160,118), size=(140,-1))
        self.st_n_pts.SetForegroundColour('Yellow') ; self.st_n_pts.SetFont(font)
```

Listing 23.1 Open_3D_wx_Python.pyw – Open 3D – Python & wxPython version

```
        choices_n_pts=('1000', '10000', '50000', '100000', )
        self.lst_n_pts = wx.Choice(self.panel, -1, choices=choices_n_pts, pos=(160,135), size=(85, -1))
        self.lst_n_pts.SetSelection(0)

        self.st_PCD_options=wx.StaticText(self.panel, -1, 'POINT CLOUD options', pos=(LEFT,180), size=(140,-1))
        self.st_PCD_options.SetForegroundColour('Green') ; self.st_PCD_options.SetFont(font)

        self.chk_PCD_View= wx.CheckBox(self.panel, -1, " View ", pos=(LEFT,210), size=(170,-1))
        self.chk_PCD_View.SetForegroundColour('Green') ; self.chk_PCD_View.SetFont(font)
        self.chk_PCD_View.SetToolTip("View the Point Cloud")
        self.chk_PCD_View.SetValue(True)

        self.chk_Bounding_Volumes= wx.CheckBox(self.panel, -1, " Bounding Volumes ", pos=(LEFT,232), size=(170,-1))
        self.chk_Bounding_Volumes.SetForegroundColour('Green') ; self.chk_Bounding_Volumes.SetFont(font)
        sir="A 3D enclosure used in computer graphics to enclose elements within a hierarchical structure."
        self.chk_Bounding_Volumes.SetToolTip(sir)

        self.chk_PCD_Convex_hull= wx.CheckBox(self.panel, -1, " Convex hull ", pos=(LEFT,254), size=(170,-1))
        self.chk_PCD_Convex_hull.SetForegroundColour('Green') ; self.chk_PCD_Convex_hull.SetFont(font)
        sir="The convex hull of a point cloud is the smallest convex set that contains all points."
        self.chk_PCD_Convex_hull.SetToolTip(sir)

    def Open_FILE(self, OPT):
        self.filename=""
        if OPT=="Mesh":
            title="Choose MESH file to load"
            matches = ["ply", "stl.", "obj"]
            Wildcard1 = "All files (*.*)|*.*|" \
                    "PLY file (*.ply)|*.ply|" \
                    "STL file (*.stl)|*.stl|" \
                    "OBJ file (*.obj)|*.obj"
            dialog = wx.FileDialog(None, title, "", "", wildcard=Wildcard1)
        if OPT=="PCD":
            title="Choose Point Cloud file to load"
            matches = ["xyz", "xyzn", "xyzrgb", "ply", "pcd", "pts"]
            # xyz=ASCII point cloud files ; xyzn=ASCII point cloud with normals
            # xyzrgb=ASCII point cloud files with colors ; pcd=Point Cloud Data files
            # pts=3D Points files
            Wildcard2 = "All files (*.*)|*.*|" \
                    "XYZ file (*.xyz)|*.xyz|" \
                    "XYZN file (*.xyzn)|*.xyzn|" \
                    "XYZRGB file (*.xyzrgb)|*.xyzrgb|" \
                    "PLY file (*.ply)|*.ply|" \
                    "PCD file (*.pcd)|*.pcd|" \
                    "PTS file (*.pts)|*.pts"
            dialog = wx.FileDialog(None, title, "", "", wildcard=Wildcard2)
        if dialog.ShowModal() == wx.ID_CANCEL:
            return
        self.filename = dialog.GetPath().strip().upper()
        self.statusbar.SetStatusText(self.filename,0)

    def OnOpen_MESH(self,event):
        self. OPEN_MESH_STL ("Mesh")
    def OnDemo(self, events):
        self. OPEN_MESH_STL ("STL")

    def OPEN_MESH_STL (self,Optiune):
        if Optiune=="Mesh":
            self.Open_FILE("Mesh")
            if self.filename=="":
                wx.MessageBox("Please select a correct type file !")
                return
            FISIER = self.filename
        if Optiune=="STL":
            STL = self.Opt_Demo.GetStringSelection().strip().replace(" ", "")
            FISIER=os.getcwd()+"/STL/"+STL+".stl"
            self.statusbar.SetStatusText(FISIER,0)
        mesh = o3d.io.read_triangle_mesh(FISIER)

        sir="Mesh "+str(FISIER)+"\n"
        sir=sir+str(mesh)+"\n"
        sir=sir+'\nVertices'+"\n"+str(np.asarray(mesh.vertices))
```

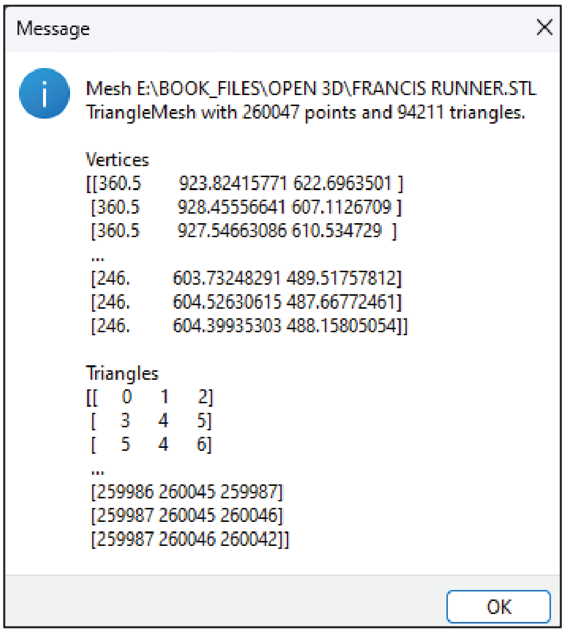
Listing 23.1 Open_3D_wx_Python.pyw – Open 3D – Python & wxPython version

```
        sir=sir+'\n\nTriangles'+"\n"+str(np.asarray(mesh.triangles))
        wx.MessageBox(sir)

        Vert_Tri=" "+str(len(mesh.vertices))+" vertices & "+str(len(mesh.triangles))+" triangles"
        if self.chk_Computed_Vertex_Normals.IsChecked():
            mesh.compute_vertex_normals()
            o3d.visualization.draw_geometries([mesh],
                window_name='Mesh: '+FISIER+' Compute vertex normals'+Vert_Tri,
                width=self.W_width, height=self.H_height, left=self.left, top=self.top)

        if self.chk_Computed_Normals.IsChecked():
            o3d.visualization.draw_geometries([mesh], window_name='Mesh: '+FISIER+
              '; Computed Normals'+Vert_Tri,
              width=self.W_width, height=self.H_height, left=self.left, top=self.top,
             mesh_show_wireframe=True)
        n_pts=int(self.lst_n_pts.GetStringSelection().strip())
        # n_pts = uniformly sampling points to convert mesh into a point cloud.
        pcd1 = mesh.sample_points_uniformly(n_pts)
        pcd2 = mesh.sample_points_poisson_disk(n_pts)

        if self.chk_Convex_hull.IsChecked():
            mesh.compute_vertex_normals()
            # The convex hull of a point cloud is the smallest convex set that contains all points.
            No_Points = 20000
            pcl = mesh.sample_points_poisson_disk(number_of_points=No_Points)
            hull, _ = pcl.compute_convex_hull()
            hull_ls = o3d.geometry.LineSet.create_from_triangle_mesh(hull)
            hull_ls.paint_uniform_color((0, 0, 0))
            o3d.visualization.draw_geometries([pcl, hull_ls],
                window_name='Mesh: '+FISIER+ " Convex hull with points number = "+str(No_Points),
                width=self.W_width, height=self.H_height, left=self.left, top=self.top)

        if self.chk_Uniform_points.IsChecked():
            o3d.visualization.draw_geometries([pcd1], window_name='Mesh: '+FISIER+
                " Sample uniform points: "+str(n_pts),
                width=self.W_width, height=self.H_height, left=self.left, top=self.top)

        if self.chk_Poisson_points.IsChecked():
            o3d.visualization.draw_geometries([pcd2], window_name='Mesh: '+FISIER+
            " Sample Poisson points: "+str(n_pts),
            width=self.W_width, height=self.H_height, left=self.left, top=self.top)

    def OnOpen_POINT_CLOUD(self, event):
        self.Open_FILE("PCD")
        if self.filename=="":
            wx.MessageBox("Please select a correct type file !")
            return
        FISIER = self.filename
        pcd = o3d.io.read_point_cloud(FISIER)

        sir="Point Cloud "+str(FISIER)+"\n"
        sir=sir+str(pcd)+"\n"
        sir=sir+'\nVertices'+"\n"+str(np.asarray(pcd.points))
        wx.MessageBox(sir)

        if self.chk_PCD_View.IsChecked():
            o3d.visualization.draw_geometries([pcd],
                window_name='Point cloud: '+FISIER,
                width=self.W_width, height=self.H_height, left=self.left, top=self.top)

        if self.chk_Bounding_Volumes.IsChecked():
            aabb = pcd.get_axis_aligned_bounding_box() ; aabb.color = (1, 0, 0)
            obb = pcd.get_oriented_bounding_box() ; obb.color = (0, 1, 0)
            o3d.visualization.draw_geometries([pcd, aabb, obb],
                window_name='Point cloud: '+FISIER+" Bounding Volumes",
                width=self.W_width, height=self.H_height, left=self.left, top=self.top)

        if self.chk_PCD_Convex_hull.IsChecked():
            hull, _ = pcd.compute_convex_hull()
            hull_ls = o3d.geometry.LineSet.create_from_triangle_mesh(hull)
            hull_ls.paint_uniform_color((1, 0, 0))
            o3d.visualization.draw_geometries([pcd, hull_ls],
```

Listing 23.1 Open_3D_wx_Python.pyw – Open 3D – Python & wxPython version

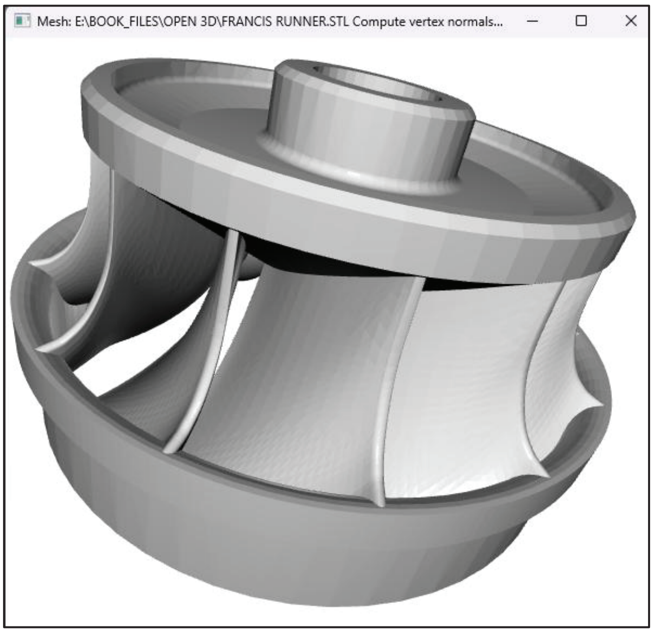
```
                                    window_name='Point cloud: '+FISIER+" Convex hull",
                                    width=self.W_width, height=self.H_height, left=self.left, top=self.top)

    def OnExit(self, evt):
        self.Close()

if __name__ == '__main__':
    app = wx.App(0)
    frame = OPEN_3D(None, -1, "Open 3D examples - wx.Python version")
    frame.Show()
    app.MainLoop()

```

Figures 23.2 ÷ 23.7 show images of **Francis runner.STL** file created by **Open_3D_wx_Python.pyw** script.



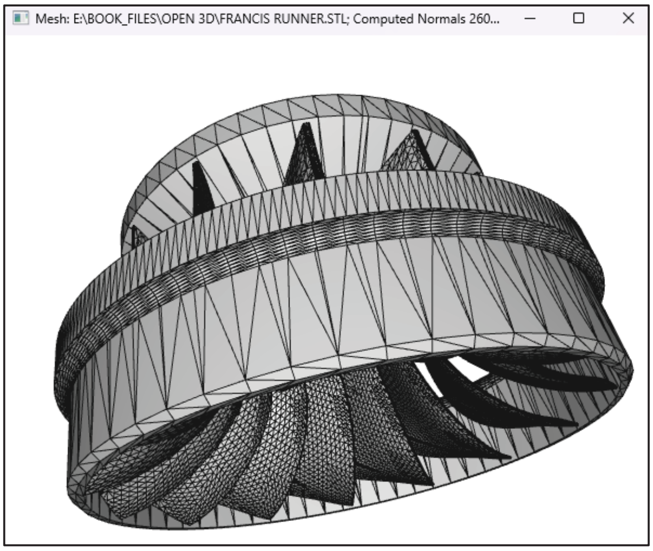
```
Mesh E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL
TriangleMesh with 260047 points and 94211 triangles.

Vertices
[[360.5      923.82415771 622.6963501 ]
 [360.5      928.45556641 607.1126709 ]
 [360.5      927.54663086 610.534729  ]
 ...
 [246.       603.73248291 489.51757812]
 [246.       604.52630615 487.66772461]
 [246.       604.39935303 488.15805054]]

Triangles
[[     0      1      2]
 [     3      4      5]
 [     5      4      6]
 ...
 [259986 260045 259987]
 [259987 260045 260046]
 [259987 260046 260042]]
```

Figure 23.2 The **Info** for **Francis runner.STL** file 260047 points & 94211 triangles



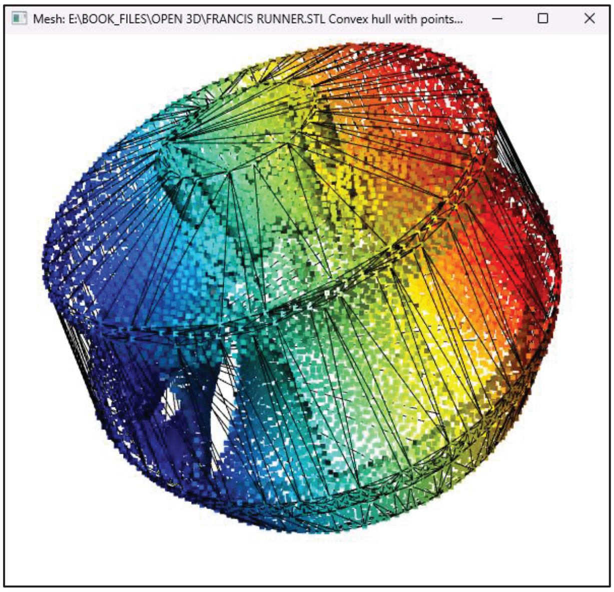
Mesh: E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL Compute vertex normals...

Figure 23.3 The **Compute vertex normals** for **Francis runner.STL** file



Mesh: E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL; Computed Normals 260...

Figure 23.4 The **Compute normals** for **Francis runner.STL** file



Mesh: E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL Convex hull with points...

Figure 23.5 The **Convex hull** for **Francis runner.STL** file

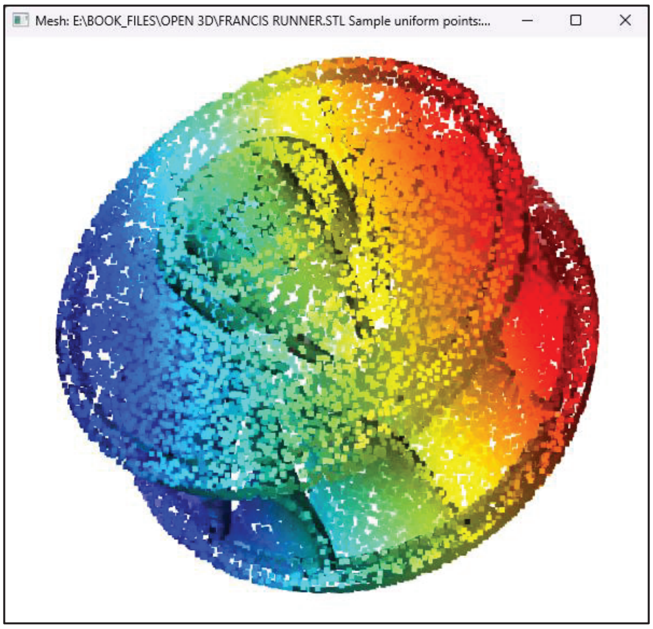

Mesh: E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL Sample uniform points:...

Figure 23.6 The **Uniform points** for **Francis runner.STL** file (50000 points)

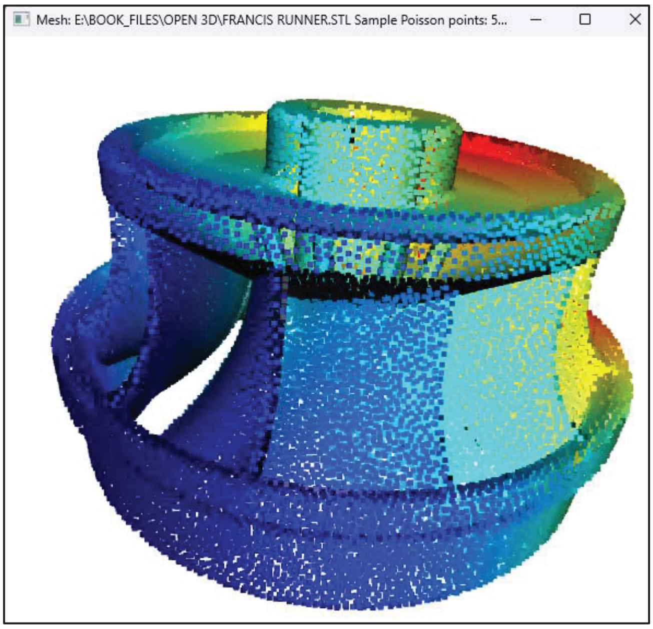
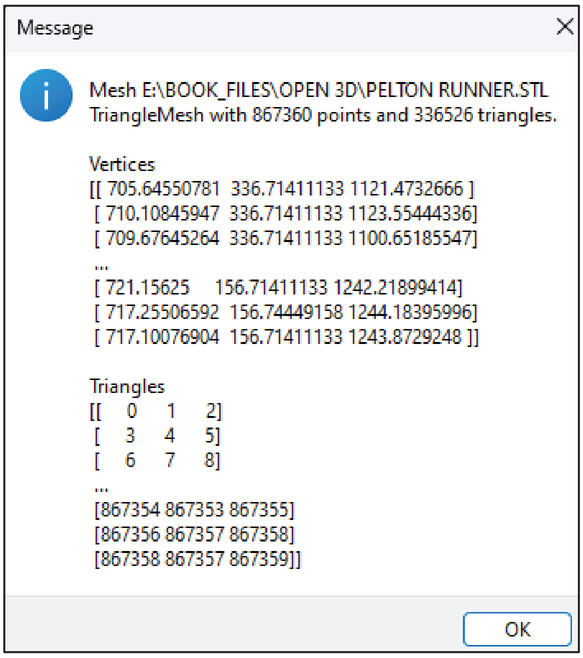

Mesh: E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL Sample Poisson points: 5...

Figure 23.7 The **Poissonpoints** for **Francis runner.STL** file (50000 points)

Figures 23.8 ÷ 23.11 show images of **Pelton runner.STL** file created by **Open_3D_wx_Python.pyw** script.
Figures 23.12 ÷ 23.13 show images of **Horse.stl** file created by **Open_3D_wx_Python.pyw** script.
Figure 23.14 show images of **Worker.xyzrgb** file created by **Open_3D_wx_Python.pyw** script.
Figures 23.15 ÷ 23.16 show images of **Dragon.stl** file created by **Open_3D_wx_Python.pyw** script [**23.4**].
Figures 23.17 ÷ 23.27 show images of some **STL** files created by **Open_3D_wx_Python.pyw** script [**23.4**].



```
Mesh E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL
TriangleMesh with 867360 points and 336526 triangles.

Vertices
[[ 705.64550781  336.71411133 1121.4732666 ]
 [ 710.10845947  336.71411133 1123.55444336]
 [ 709.67645264  336.71411133 1100.65185547]
 ...
 [ 721.15625     156.71411133 1242.21899414]
 [ 717.25506592  156.74449158 1244.18395996]
 [ 717.10076904  156.71411133 1243.8729248 ]]

Triangles
[[      0       1       2]
 [      3       4       5]
 [      6       7       8]
 ...
 [867354 867353 867355]
 [867356 867357 867358]
 [867358 867357 867359]]
```

Figure 23.8 The **Info** for **Pelton runner.STL** file (867360 points & 336526 triangles)

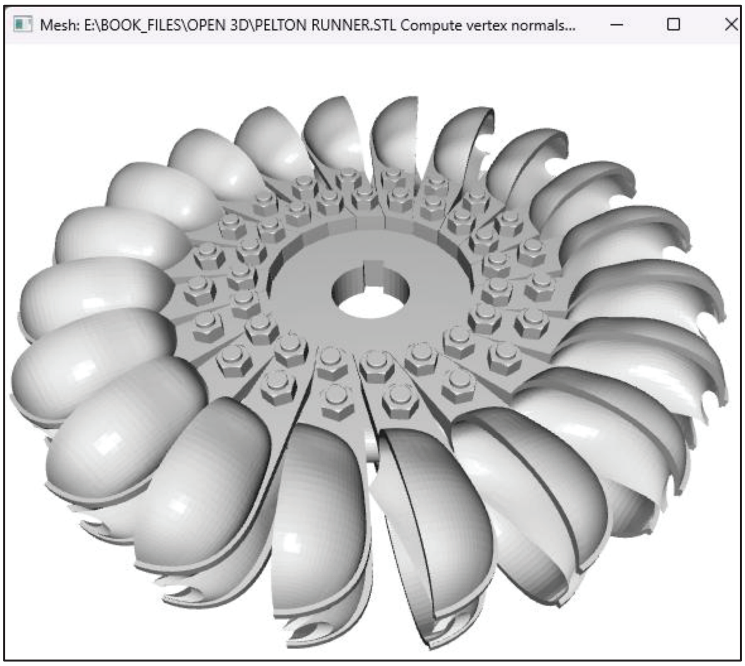
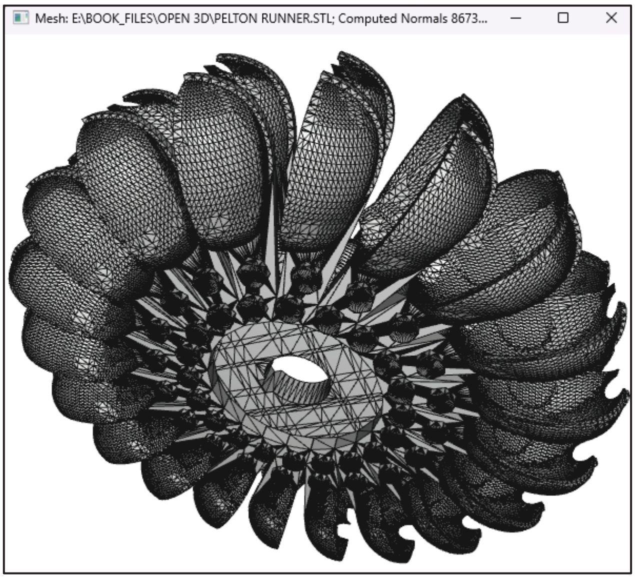
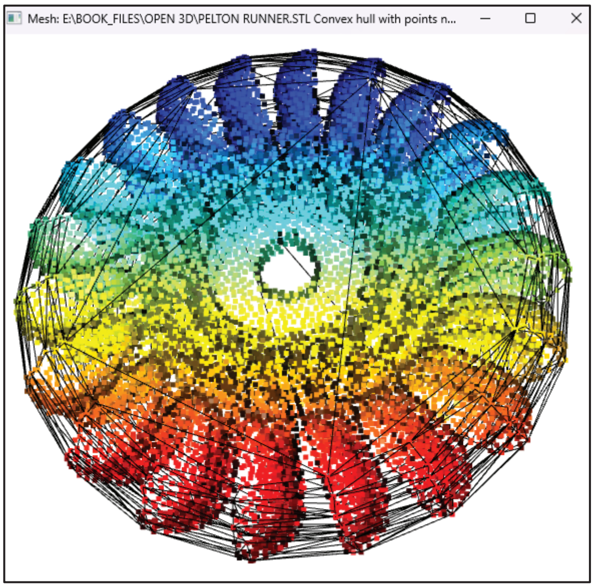

Mesh: E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL Compute vertex normals...

Figure 23.9 The **Compute vertex normals** for **Pelton runner.STL** file



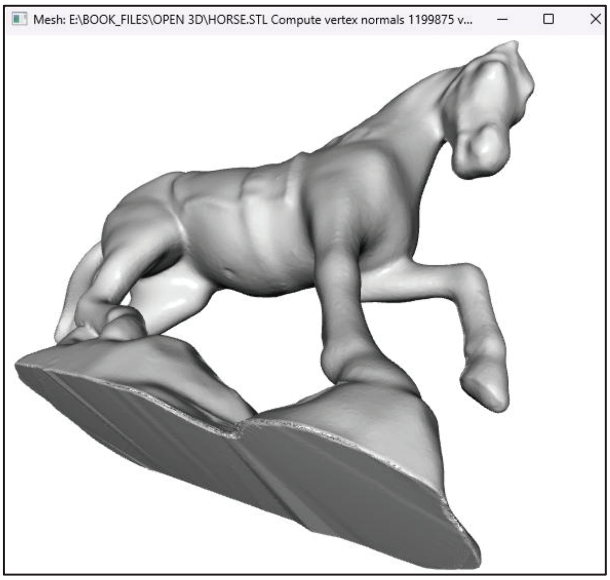
Mesh: E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL; Computed Normals 8673...

Figure 23.10 The **Compute normals** for **Pelton runner.STL** file



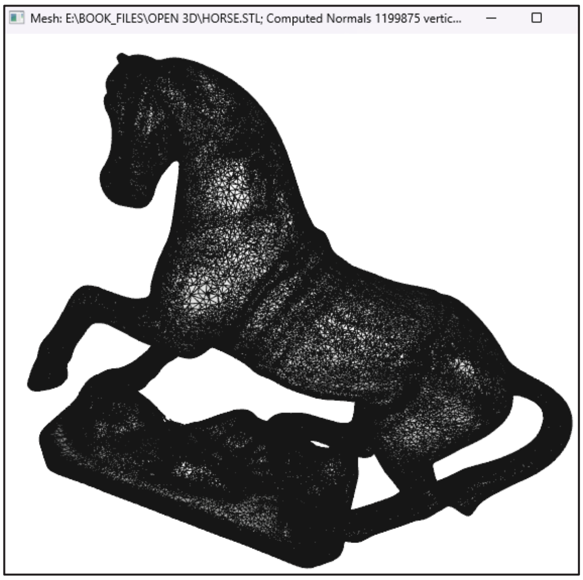
Mesh: E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL Convex hull with points n...

Figure 23.11 The **Convex hull** for **Pelton runner.STL** file



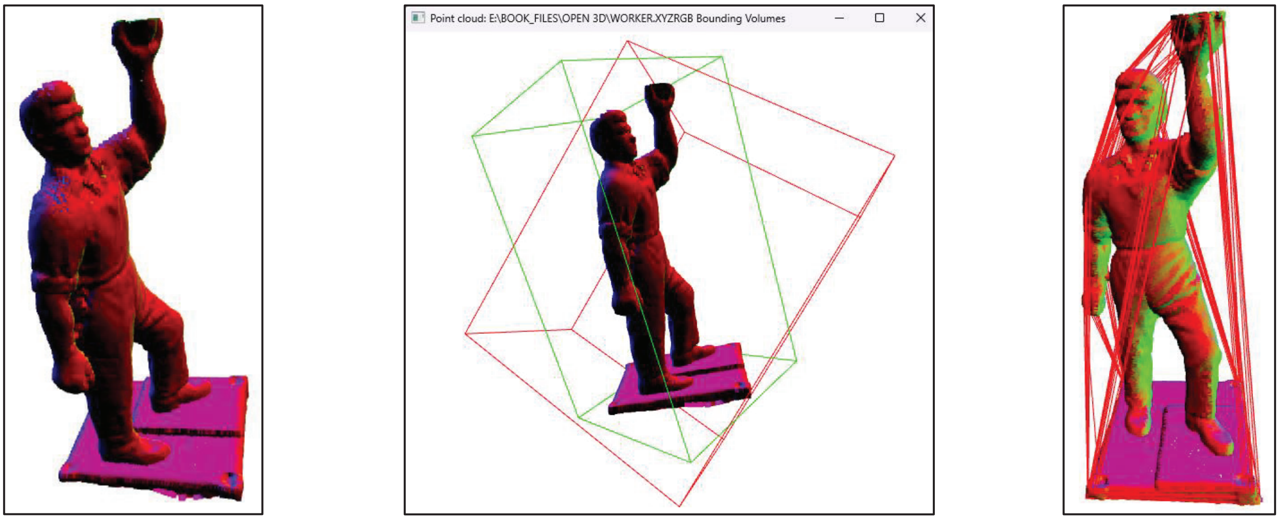
Mesh: E:\BOOK_FILES\OPEN 3D\HORSE.STL Compute vertex normals 1199875 v...

Figure 23.12 The **Compute vertex normals** for **Horse.stl** file



Mesh: E:\BOOK_FILES\OPEN 3D\HORSE.STL; Computed Normals 1199875 vertic...

Figure 23.13 The **Compute Normals** for **Horse.stl** file 1199875 points & 399999 triangles



Point cloud: E:\BOOK_FILES\OPEN 3D\WORKER.XYZRGB Bounding Volumes

Figure 23.14 The **View, Bounding Volumes** and **Convex hull** options for **Worker.xyzrgb** file (24305 points)

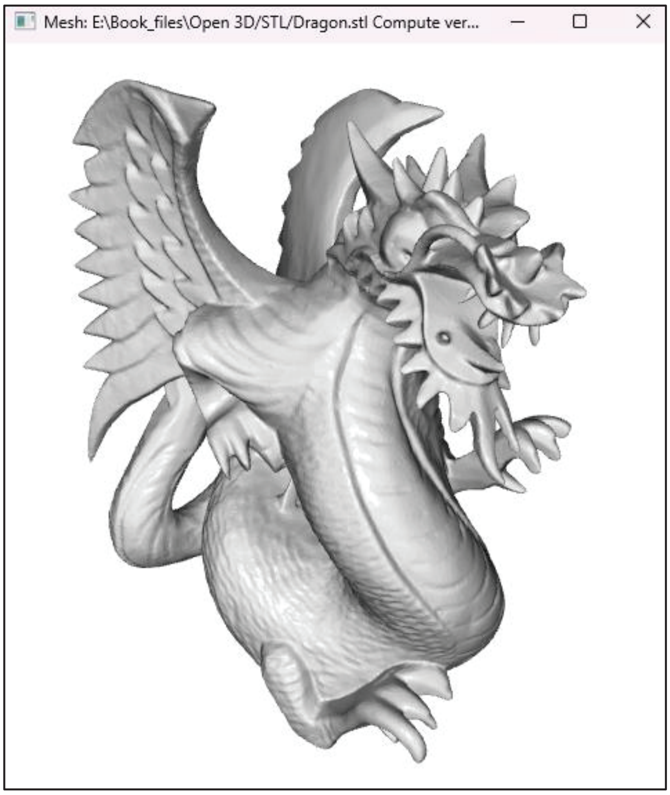


Figure 23.15 The **Compute vertex normals** for **Dragon.stl** file

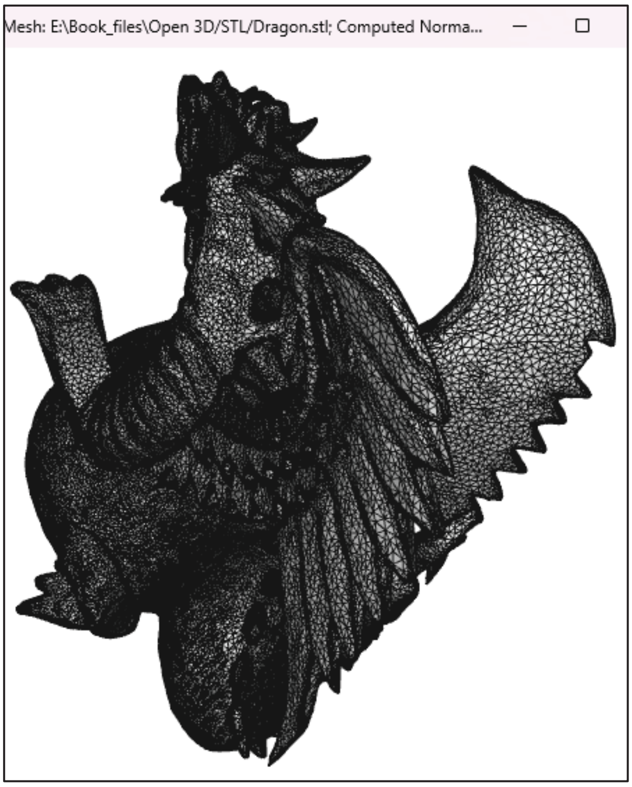

Mesh: E:\Book_files\Open 3D/STL/Dragon.stl; Computed Norma...

Figure 23.16 The **Compute Normals** for **Dragon.stl** file (629644 points & 209898 triangles)

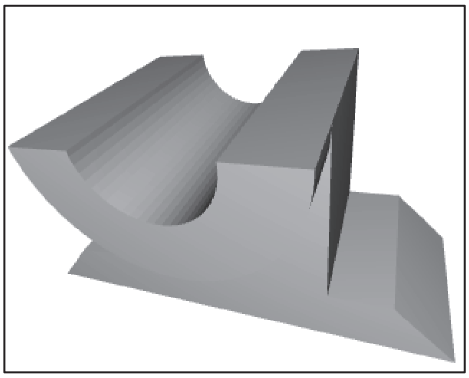


Figure 23.17 The **Compute vertex normals** for **STL1.stl** file

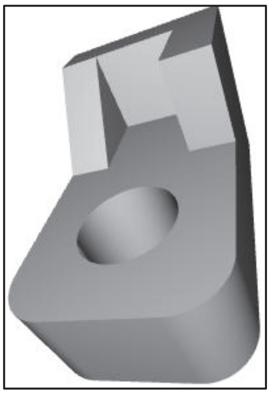


Figure 23.18 The **Compute vertex normals** for **STL2.stl** file

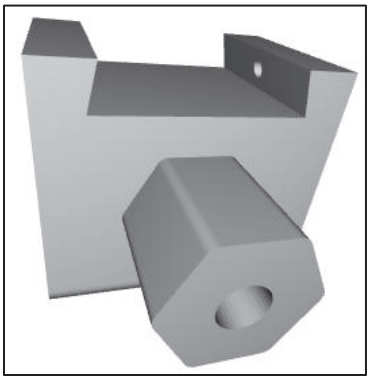


Figure 23.19 The **Compute vertex normals** for **STL3.stl** file

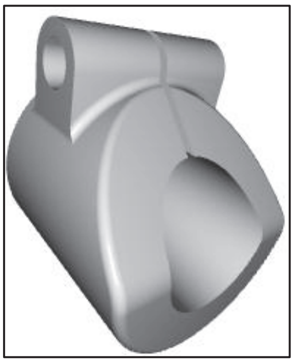


Figure 23.20 The **Compute vertex normals** for **STL4.stl** file

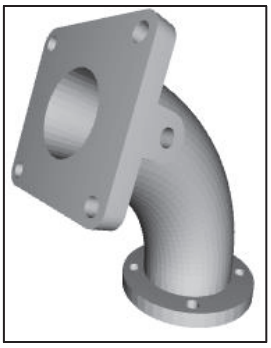


Figure 23.21 The **Compute vertex normals** for **STL5.stl** file

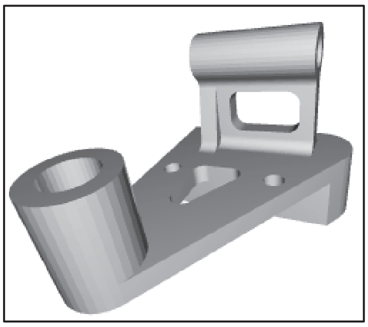


Figure 23.22 The **Compute vertex normals** for **STL6.stl** file

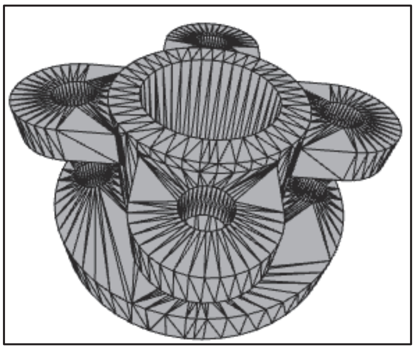


Figure 23.23 The **Compute normals** for **STL7.stl** file

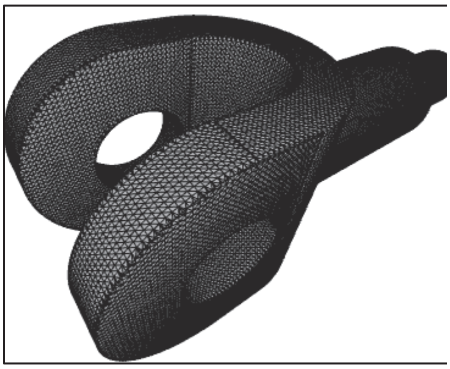


Figure 23.24 The **Compute normals** for **STL8.stl** file

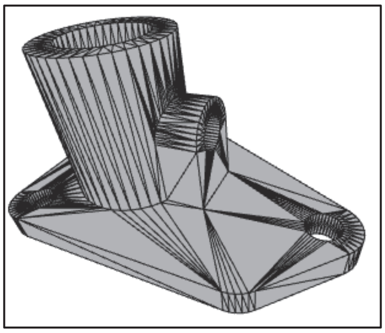


Figure 23.25 The **Compute normals** for **STL9.stl** file

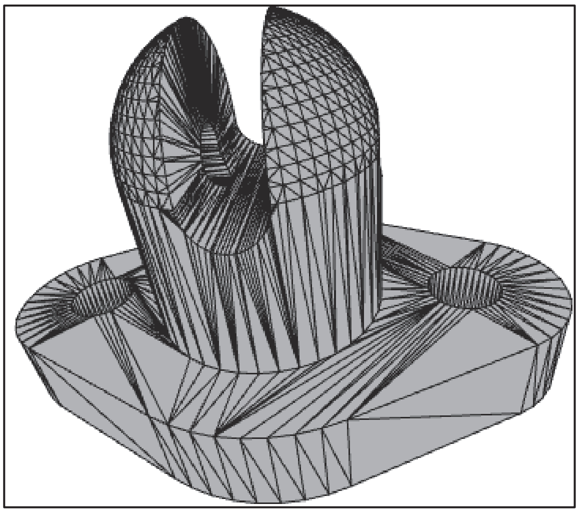


Figure 23.26 The **Compute normals** for **STL10.stl** file

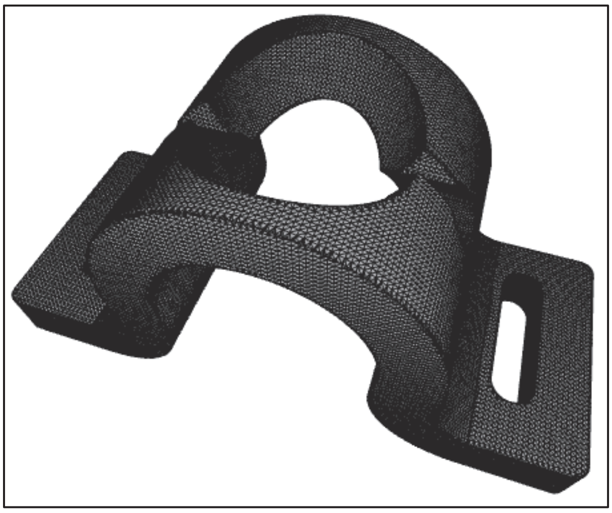


Figure 23.27 The **Compute normals** for **STL11.stl** file

24. VTK

"The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, modeling, image processing, volume rendering, scientific visualization, and 2D plotting. It supports a wide variety of visualization algorithms and advanced modeling techniques, and it takes advantage of both threaded and distributed memory parallel processing for speed and scalability, respectively." [24.1].

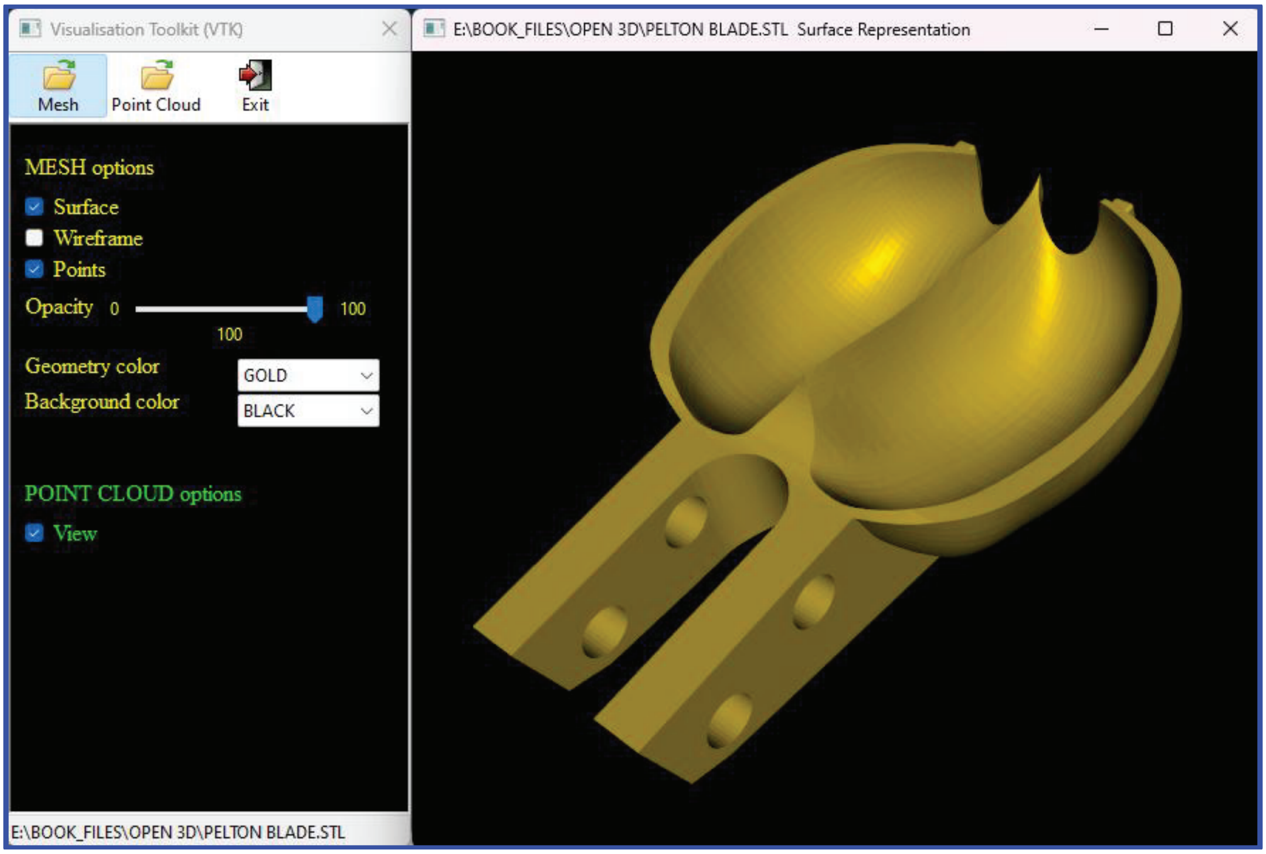
The chapter exemplifies a simple mesh & point cloud script through **VTK.pyw** saved in **VTK** folder, with interface from **Figure 24.1**.

A **mesh** is a three-dimensional model used to create 3D objects & environments and is made of points, lines and surfaces.

The **STL** file format approximates the surfaces of a solid model with triangles.

A **point cloud** is a discrete set of data points in space. The points can represent a 3D shape or object. Each point location has its own set of Cartesian coordinates.

The left side of the interface contain the toolbar and the some options for mesh & point cloud. The right side of the interface display the 3D geometry in concordance with selected options.



E:\BOOK_FILES\OPEN 3D\PELTON BLADE.STL Surface Representation

MESH options

☑ Surface

□ Wireframe

☑ Points

Opacity 0 100

100

Geometry color GOLD

Background color BLACK

POINT CLOUD options

☑ View

E:\BOOK_FILES\OPEN 3D\PELTON BLADE.STL

Figure 24.1 The **VTK.pyw** interface

❶ To install **VTK** go to [**23.3**], write **VTK** in **Search** control and select **VTK 9.3.1** version to download **vtk-9.3.1-cp311-cp311-win_amd64.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip3.11 install vtk-9.3.1-cp311-cp311-win_amd64.whl
```

The 3D geometry can be rotated in right window with left mouse pressed, panned with center mouse wheel pressed and zoomed with center mouse wheel. The right window can be redimensioned with left mouse placed on the window border.

Listing 24.1 displays the **Python & wxPython** version of script, **VTK.pyw**.

The **VTK.pyw** script (Python 3.11.9 & wxPython version) include one class: **VTK**. The script begin with importing libraries (lines 4÷7).

The **VTK** class create the script interface and define functions to display 3D geometry:

- **__init__** Calculate the width **W** and heigth **H** of the display, the frame dimensions (**Wframe**, **Hframe**) and left/ top position (**poz_WND** / **poz_HND**); define the **frame** (lines 25÷26), the **panel** (lines 28÷29) and **statusbar** (line 30÷31), line 33 define the right window dimensions (**W_width**, **H_height**) and left/ top position (**left**/ **top**); lines 35, 36 call **_TOOLBAR** and **_CONTROLS** functions.
- **_TOOLBAR** Lines 39, 40 define the **toolbar** and the background. Lines 42÷44 define **Mesh** control and **Bind** with **OnOpen_MESH** class function (line 45) to open an mesh file. Lines 47÷50 define **Point Cloud** control and **Bind** with **OnOpen_POINT_CLOUD** class function (line 51) to open an point cloud file. Lines 53÷55 define **Exit** icon connected through **Bind** command (line 56) with **OnExit**class function.
- **_CONTROLS** This class function create, through **wx.CheckBox, wx.Slider** commands, controls to select options for mesh and point cloud (placed in left interface window).
- **Open_FILE** This function open a dialog to select an mesh or point cloud file in **self.filename** name. The **wildcard** is different for mesh or point cloud.
- **OnOpen_MESH** This function call on **OPEN_MESH** function with **Mesh** option.
- **OPEN_MESH_STL** Open a mesh or a **STL** file for processing; lines 166÷170 display an info about selected file; lines 172÷177 display the 3D mesh/STL geometry through **VTK_MESH** window according with the mesh options selected from the left window.
- **VTK_MESH** Display the 3D geometry in right window using VTK commands.
- **OnOpen_POINT_CLOUD** Open a point cloud file for processing; lines 187÷190 display an info about selected file; line 191 call **VTK_PCD** function to display the point cloud geometry according with the point cloud options selected from the left window.
- **VTK_PCD** The point geometry is readed through **Read_TXT** function (line 265) and displayed in right window using VTK commands.
- **OnCOLOR** Change the geometry color.
- **OnCOL_BACK** Change the right window background color.
- **On_Opacitate** Change the opacity of the geometry.
- **OnExit** Close the application.

Listing 24.1 VTK.pyw – VTK – Python & wxPython version

```
1.  #! /usr/bin/env python
2.  #coding=utf-8
3.
4.  import wx, os
5.  import numpy as np
6.  import open3d as o3d
7.  import vtk
8.
9.  import vtkmodules.vtkInteractionStyle
10. import vtkmodules.vtkRenderingOpenGL2
11. from vtkmodules.vtkCommonColor import vtkNamedColors
12. from vtkmodules.vtkRenderingCore import (
13.     vtkActor,
14.     vtkPolyDataMapper,
15.     vtkRenderWindow,
16.     vtkRenderWindowInteractor,
17.     vtkRenderer)
18. from vtkmodules.vtkIOGeometry import (
19.     vtkOBJReader,
20.     vtkSTLReader)
21. from vtk import vtkPLYReader
22. class VTK(wx.Frame):
23.     def __init__(self, parent, ID, title):
24.         W,H = wx.DisplaySize() ; Wframe=300 ; Hframe=600 ; poz_WND=100 ; poz_HND=int(H-Hframe)/2
25.         self.frame=wx.Frame.__init__(self, parent, ID, title, size=(Wframe, Hframe), pos=(poz_WND,poz_HND),
26.             style=wx.DEFAULT_FRAME_STYLE ^ ( wx.RESIZE_BORDER | wx.MINIMIZE_BOX | wx.MAXIMIZE_BOX))
27.
28.         self.panel = wx.Panel(self,-1,style=wx.SUNKEN_BORDER)
```

Listing 24.1 VTK.pyw – VTK – Python & wxPython version

```
        self.panel.SetBackgroundColour("Black")
        self.statusbar = self.CreateStatusBar() # Create the status bar
        self.statusbar.SetFieldsCount(1)

        self.W_width=600 ; self.H_height=Hframe-35 ; self.left=poz_WND+Wframe ; self.top=int(poz_HND)-35

        self._TOOLBAR()
        self._CONTROLS()

    def _TOOLBAR(self):
        self.toolbar = self.CreateToolBar( ( wx.TB_HORIZONTAL  | wx.TB_TEXT) )
        self.toolbar.SetBackgroundColour("white")

        img = wx.Image(os.getcwd()+'/_Icons/open.png', wx.BITMAP_TYPE_ANY)
        id_Open_MESH=90 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_Open_MESH , "Mesh", sel_bmp, shortHelp="Open MESH file: .ply .stl .obj")
        self.Bind(wx.EVT_TOOL, self.OnOpen_MESH, id=id_Open_MESH)

        img1 = wx.Image(os.getcwd()+'/_Icons/open.png', wx.BITMAP_TYPE_ANY)
        id_Open_PCD=91 ; sel_bmp1=wx.Bitmap(img1.Scale(24,24))
        sh="Open Point Cloud file: .xyz .xyzn .xyzrgb .ply .pcd .pts"
        self.toolbar.AddTool( id_Open_PCD , "Point Cloud", sel_bmp1, shortHelp=sh)
        self.Bind(wx.EVT_TOOL, self.OnOpen_POINT_CLOUD, id=id_Open_PCD)

        img = wx.Image(os.getcwd()+'/_Icons/exit.png', wx.BITMAP_TYPE_ANY)
        id_Exit=94 ; sel_bmp=wx.Bitmap(img.Scale(24,24))
        self.toolbar.AddTool( id_Exit , "Exit", sel_bmp, shortHelp="Exit application")
        self.Bind(wx.EVT_TOOL, self.OnExit, id=id_Exit)

        self.toolbar.Realize() ; self.toolbar.Show()

    def _CONTROLS(self):
        LEFT=10
        font = wx.Font(12, wx.ROMAN, wx.NORMAL, wx.NORMAL)

        self.st_MESH_options=wx.StaticText(self.panel, -1, 'MESH options', pos=(LEFT,20), size=(140,-1))
        self.st_MESH_options.SetForegroundColour('Yellow') ; self.st_MESH_options.SetFont(font)

        self.chk_Surfaces= wx.CheckBox(self.panel, -1, " Surface ", pos=(LEFT,50), size=(100,-1))
        self.chk_Surfaces.SetForegroundColour('Yellow')
        self.chk_Surfaces.SetFont(font)
        self.chk_Surfaces.SetValue(True)

        self.chk_Wireframe= wx.CheckBox(self.panel, -1, " Wireframe ", pos=(LEFT,72), size=(100,-1))
        self.chk_Wireframe.SetForegroundColour('Yellow') ; self.chk_Wireframe.SetFont(font)

        self.chk_Points = wx.CheckBox(self.panel, -1, " Points ", pos=(LEFT,94), size=(130,-1))
        self.chk_Points.SetForegroundColour('Yellow') ; self.chk_Points.SetFont(font)
        sir="The convex hull of a point cloud is the smallest convex set that contains all points."
        self.chk_Points.SetToolTip(sir)

        stOpac=wx.StaticText(self.panel, -1, "Opacity", (LEFT, 118))
        stOpac.SetForegroundColour('Yellow') ; stOpac.SetFont(font)
        self.sld_Opac = wx.Slider( self.panel, 0, 100, 0, 100, (LEFT+60,118), (180, -1), wx.SL_HORIZONTAL | wx.SL_LABELS)
        self.sld_Opac.SetForegroundColour('Yellow')
        self.Bind(wx.EVT_SLIDER, self.On_Opacitate, self.sld_Opac)

        COLORS = ["BLUE","BROWN","CYAN","GOLD","GREY","GREEN","MAGENTA",
                           "NAVY","PINK","RED","VIOLET","DarkOliveGreen","YELLOW","BLACK"]
        stCol=wx.StaticText(self.panel, -1, "Geometry color", (LEFT, 160))
        stCol.SetForegroundColour('Yellow') ; stCol.SetFont(font)
        self.lst_COLORS = wx.Choice(self.panel, -1, choices=COLORS, pos=(160,165), size=(100, -1))
        self.lst_COLORS.SetSelection(0)
        self.Bind(wx.EVT_CHOICE, self.OnCOLOR, self.lst_COLORS)

        stCol1=wx.StaticText(self.panel, -1, "Background color", (LEFT, 185))
        stCol1.SetForegroundColour('Yellow') ; stCol1.SetFont(font)
        self.lst_COL_BACK = wx.Choice(self.panel, -1, choices=COLORS, pos=(160,190), size=(100, -1))
        x=self.lst_COL_BACK.FindString('GREY')
        self.lst_COL_BACK.SetSelection(x)
        self.Bind(wx.EVT_CHOICE, self.OnCOL_BACK, self.lst_COL_BACK)

```

Listing 24.1 VTK.pyw – VTK – Python & wxPython version

```
        self.st_PCD_options=wx.StaticText(self.panel, -1, 'POINT CLOUD options', pos=(LEFT,250), size=(140,-1))
        self.st_PCD_options.SetForegroundColour('Green') ; self.st_PCD_options.SetFont(font)

        self.chk_PCD_View= wx.CheckBox(self.panel, -1, " View ", pos=(LEFT,280), size=(170,-1))
        self.chk_PCD_View.SetForegroundColour('Green') ; self.chk_PCD_View.SetFont(font)
        self.chk_PCD_View.SetToolTip("View the Point Cloud")
        self.chk_PCD_View.SetValue(True)

    def OnCOLOR(self,event):
        try:
            colors = vtkNamedColors()
            COLOR=self.lst_COLORS.GetStringSelection().strip()
            self.actor.GetProperty().SetColor(colors.GetColor3d(COLOR))
            self.ren.GetRenderWindow().Render()
        except:
            pass

    def OnCOL_BACK(self,event):
        try:
            colors = vtkNamedColors()
            COLOR=self.lst_COL_BACK.GetStringSelection().strip()
            self.ren.SetBackground(colors.GetColor3d(COLOR))
            self.ren.GetRenderWindow().Render()
        except:
            pass

    def On_Opacitate(self,event):
        try:
            val_SLD=float(self.sld_Opac.GetValue())/100
            self.actor.GetProperty().SetOpacity(val_SLD)
            self.ren.GetRenderWindow().Render()
        except:
            pass

    def Open_FILE(self, OPT):
        self.filename=""
        if OPT=="Mesh":
            title="Choose MESH file to load"
            matches = ["ply", "stl.", "obj"]
            Wildcard1 = "All files (*.*)|*.*|" \
                    "PLY file (*.ply)|*.ply|" \
                    "STL file (*.stl)|*.stl|" \
                    "OBJ file (*.obj)|*.obj"
            dialog = wx.FileDialog(None, title, "", "", wildcard=Wildcard1)
        if OPT=="PCD":
            title="Choose Point Cloud file to load"
            matches = ["xyz"]   # xyz=ASCII point cloud files
            Wildcard2 = "XYZ file (*.xyz)|*.xyz"
            dialog = wx.FileDialog(None, title, "", "", wildcard=Wildcard2)
        if dialog.ShowModal() == wx.ID_CANCEL:
            return
        self.filename = dialog.GetPath().strip().upper()
        self.statusbar.SetStatusText(self.filename,0)

    def OnOpen_MESH(self,event):
        self.OPEN_MESH_STL("Mesh")

    def OPEN_MESH_STL(self,Optiune):
        self.Open_FILE("Mesh")
        if self.filename=="":
            wx.MessageBox("Please select a correct type file !")
            return
        FISIER = self.filename
        mesh = o3d.io.read_triangle_mesh(FISIER)

        sir="Mesh "+str(FISIER)+"\n"
        sir=sir+str(mesh)+"\n"
        sir=sir+'\nVertices'+"\n"+str(np.asarray(mesh.vertices))
        sir=sir+'\n\nTriangles'+"\n"+str(np.asarray(mesh.triangles))
        wx.MessageBox(sir)

        if self.chk_Surfaces.IsChecked():
```

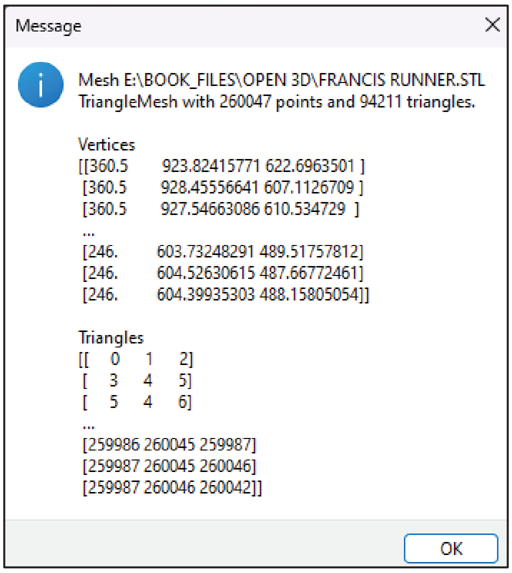
Listing 24.1 VTK.pyw – VTK – Python & wxPython version

```
            self.VTK_MESH(FISIER, "Surface")
        if self.chk_Wireframe.IsChecked():
            self.VTK_MESH(FISIER, "Wireframe")
        if self.chk_Points.IsChecked():
            self.VTK_MESH(FISIER, "Points")

    def OnOpen_POINT_CLOUD(self, event):
        self.Open_FILE("PCD")
        if self.filename=="":
            wx.MessageBox("Please select a correct type file !")
            return
        FISIER = self.filename
        pcd = o3d.io.read_point_cloud(FISIER)

        sir="Point Cloud "+str(FISIER)+"\n"
        sir=sir+str(pcd)+"\n"
        sir=sir+'\nVertices'+"\n"+str(np.asarray(pcd.points))
        wx.MessageBox(sir)
        self.VTK_PCD(FISIER)

    def VTK_MESH(self,filename, OPTIUNE):
        colors = vtkNamedColors()

        if "STL" in filename.upper():
            reader = vtkSTLReader()
        elif "PLY" in filename.upper():
            reader = vtkPLYReader()
        elif "OBJ" in filename.upper():
            reader = vtkOBJReader()
        reader.SetFileName(filename)
        reader.Update()

        mapper = vtkPolyDataMapper()
        mapper.SetInputConnection(reader.GetOutputPort())

        actor = vtkActor() ; self.actor=actor
        actor.SetMapper(mapper)
        actor.GetProperty().SetDiffuse(0.8)
        actor.GetProperty().SetDiffuseColor(colors.GetColor3d('LightSteelBlue'))
        actor.GetProperty().SetSpecular(0.3)
        actor.GetProperty().SetSpecularPower(60.0)

        COLOR=self.lst_COLORS.GetStringSelection().strip()
        actor.GetProperty().SetColor(colors.GetColor3d( COLOR ))

        # Create a rendering window and renderer
        ren = vtkRenderer() ;  self.ren=ren
        renWin = vtkRenderWindow()
        renWin.AddRenderer(ren)
        renWin.SetWindowName(filename+"  "+OPTIUNE+" Representation")

        renWin.SetSize(self.W_width, self.H_height)
        renWin.SetPosition(self.left, self.top)

        # Create a renderwindowinteractor
        iren = vtkRenderWindowInteractor()
        iren.SetRenderWindow(renWin)

        # Assign actor to the renderer
        if OPTIUNE=="Surface":
            actor.GetProperty().SetRepresentationToSurface()
        if OPTIUNE=="Wireframe":
            actor.GetProperty().SetRepresentationToWireframe()
        if OPTIUNE=="Points":
            actor.GetProperty().SetRepresentationToPoints()
        val_SLD=int(self.sld_Opac.GetValue())/100
        actor.GetProperty().SetOpacity(val_SLD) #  Opacity

        COLOR=self.lst_COL_BACK.GetStringSelection().strip()
        ren.SetBackground(colors.GetColor3d(COLOR))
        ren.AddActor(actor)

```

Listing 24.1 VTK.pyw – VTK – Python & wxPython version

```
        # Enable user interface interactor
        iren.Initialize() ; renWin.Render() ; iren.Start()

    def Read_TXT(self, file):
        FILE = open(file)
        counter=0
        for line in FILE: counter+=1
        FILE.close()

        FILE = open(file)
        i=0 ; points = np.zeros((counter,3))
        for line in FILE:
            x1,y1,z1 = line.split()
            x=float(x1) ; y=float(y1) ; z=float(z1)
            points[i,0] = x ; points[i,1] = y ; points[i,2] = z
            i+=1
        FILE.close()
        return [points, counter]

    def VTK_PCD(self,filename):
        pts, N= self.Read_TXT(filename)
        sphere = vtk.vtkSphereSource() ; sphere.SetRadius(0.01)

        mapper = vtk.vtkPolyDataMapper()
        mapper.SetInputConnection(sphere.GetOutputPort())
        # Assign actor to the renderer
        renderer = vtk.vtkRenderer() ;  self.ren=renderer
        actor = [] ; self.actor=actor
        for i in range(N):
            actor.append(vtk.vtkActor())
            actor[-1].SetMapper(mapper)
            actor[-1].SetPosition(pts[i])
            # actor[-1].GetProperty().SetColor(pts[i,2], 1.0-pts[i,2], 0.0)
            actor[-1].GetProperty().SetColor(1, 1, 1)
            renderer.AddActor(actor[-1])
        renderer.ResetCamera()

        window = vtk.vtkRenderWindow() ; window.AddRenderer(renderer)

        window.SetSize(self.W_width, self.H_height)
        window.SetPosition(self.left, self.top)

        istyle = vtk.vtkInteractorStyleSwitch()
        istyle.SetCurrentStyleToTrackballCamera()
        # Create a renderwindowinteractor
        interactor = vtk.vtkRenderWindowInteractor()
        interactor.SetRenderWindow(window)
        interactor.SetInteractorStyle(istyle)
        # Enable user interface interactor
        interactor.Initialize()
        interactor.Start()

    def OnExit(self, evt):
        self.Close()

if __name__ == '__main__':
    app = wx.App(0)
    frame = VTK(None, -1, "Visualisation Toolkit (VTK)")
    frame.Show()
    app.MainLoop()
```

Figures 24.2 ÷ **24.5** show images of **Francis runner.STL** file created by **VTK.pyw** script.
Figures 24.6 ÷ **24.7** show images of **Dragon.stl** file created by **VTK.pyw** script [**23.4**].
Figures 24.8 ÷ **24.11** show images of **Pelton runner.STL** file created by **VTK.pyw** script.



```
Mesh E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL
TriangleMesh with 260047 points and 94211 triangles.

Vertices
[[360.5      923.82415771 622.6963501 ]
 [360.5      928.45556641 607.1126709 ]
 [360.5      927.54663086 610.534729  ]
 ...
 [246.       603.73248291 489.51757812]
 [246.       604.52630615 487.66772461]
 [246.       604.39935303 488.15805054]]

Triangles
[[     0      1      2]
 [     3      4      5]
 [     5      4      6]
 ...
 [259986 260045 259987]
 [259987 260045 260046]
 [259987 260046 260042]]
```

Figure 24.2 The **Info** for **Francis runner.STL** file 260047 points & 94211 triangles

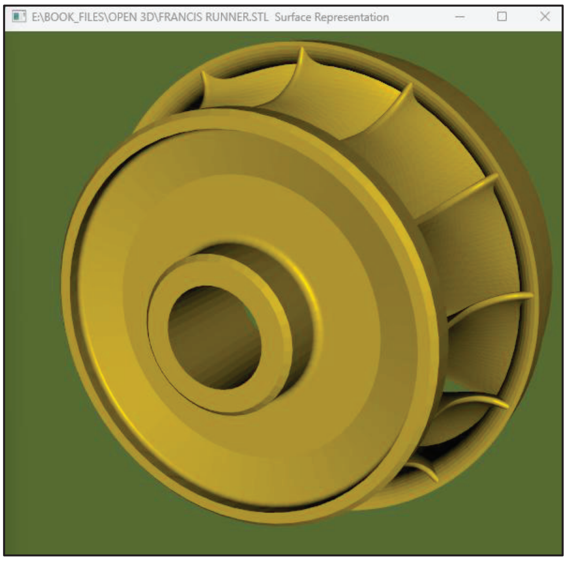

E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL Surface Representation

Figure 24.3 The **Surface** view of **Francis runner.STL** file

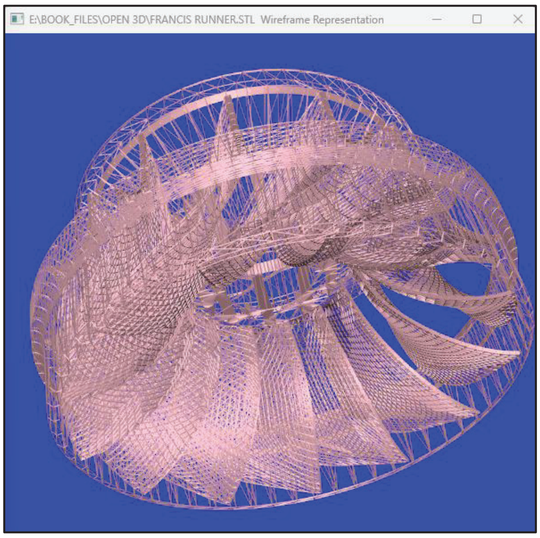

E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL Wireframe Representation

Figure 24.4 The **Wireframe** view of **Francis runner.STL** file

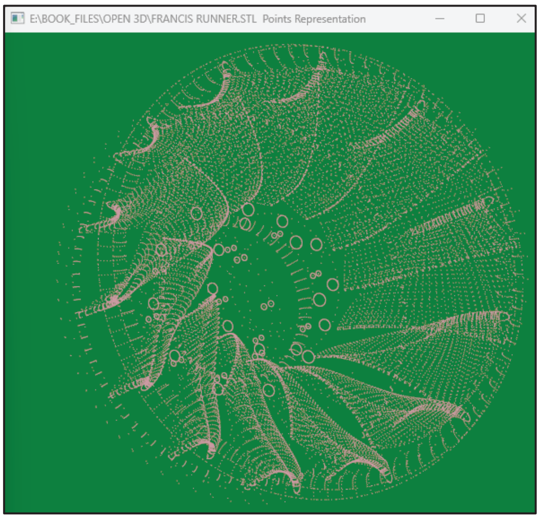

E:\BOOK_FILES\OPEN 3D\FRANCIS RUNNER.STL Points Representation

Figure 24.5 The **Points** view of **Francis runner.STL** file

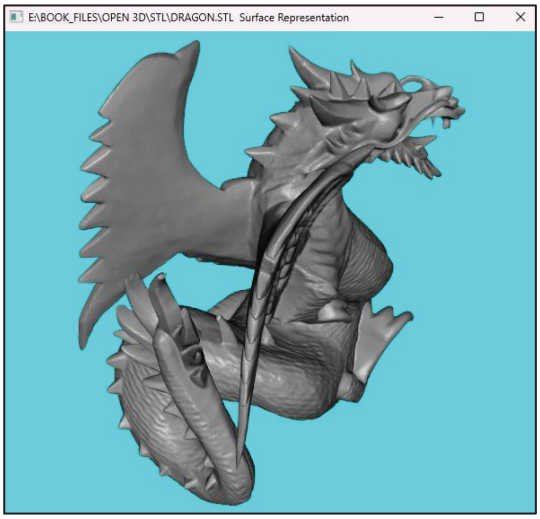


Figure 24.6 The **Surface** view of **Dragon.stl** file

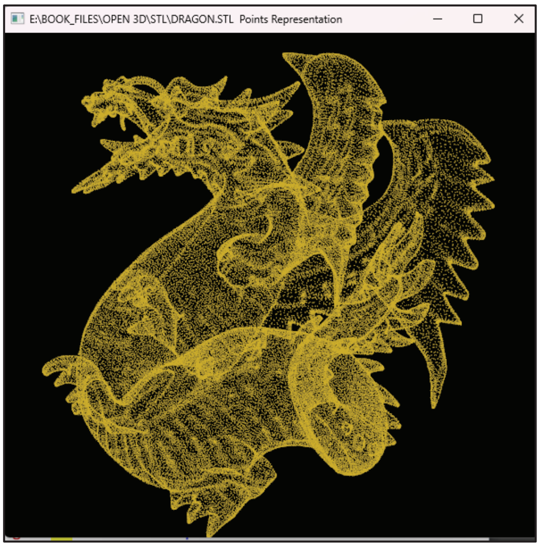
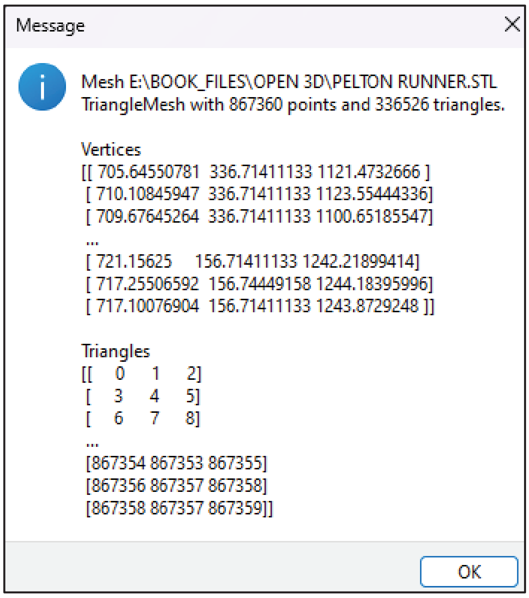


Figure 24.7 The **Points** view of **Dragon.stl** file



```
Mesh E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL
TriangleMesh with 867360 points and 336526 triangles.

Vertices
[[ 705.64550781  336.71411133 1121.4732666 ]
 [ 710.10845947  336.71411133 1123.55444336]
 [ 709.67645264  336.71411133 1100.65185547]
 ...
 [ 721.15625     156.71411133 1242.21899414]
 [ 717.25506592  156.74449158 1244.18395996]
 [ 717.10076904  156.71411133 1243.8729248 ]]

Triangles
[[     0      1      2]
 [     3      4      5]
 [     6      7      8]
 ...
 [867354 867353 867355]
 [867356 867357 867358]
 [867358 867357 867359]]
```

Figure 24.8 The **Info** for **Pelton runner.STL** file (867360 points & 336526 triangles)

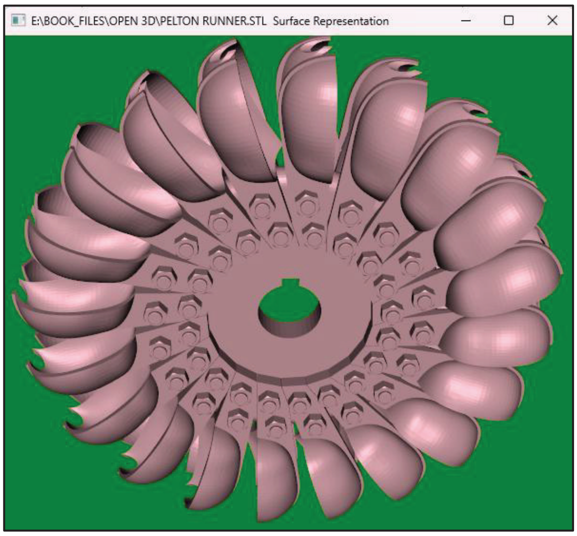

E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL Surface Representation

Figure 24.9 The **Surface** view of **Pelton runner.STL** file

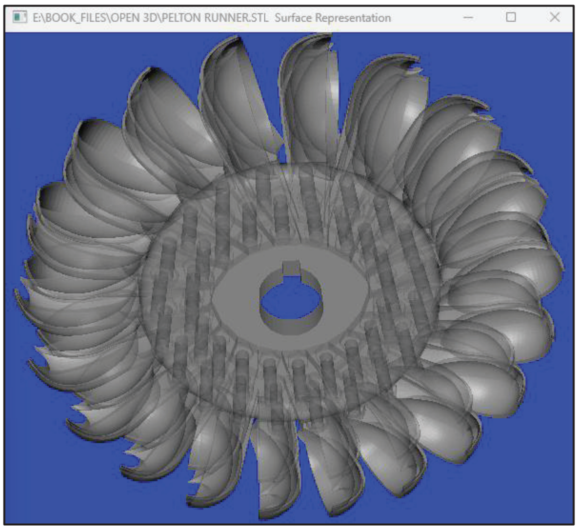

E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL Surface Representation

Figure 24.10 The **Surface** view of **Pelton runner.STL** file (Opacity = 94 %)

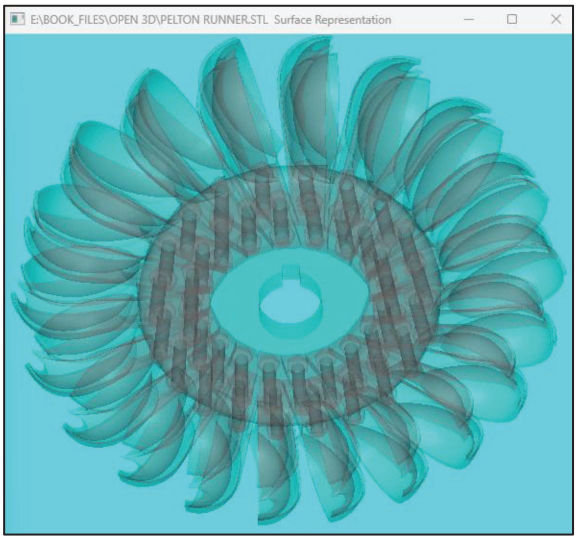

E:\BOOK_FILES\OPEN 3D\PELTON RUNNER.STL Surface Representation

Figure 24.11 The **Surface** view of **Pelton runner.STL** file (Opacity = 22 %)

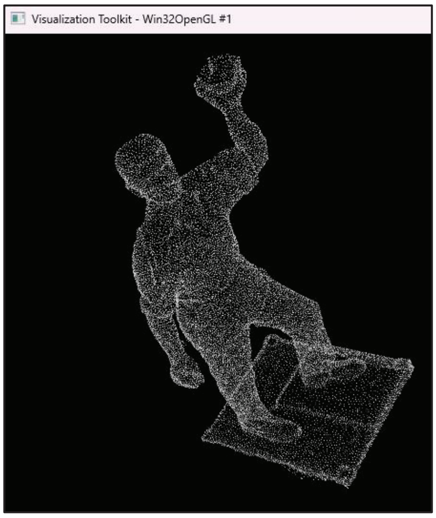


Figure 24.12 The **Point** view of **Worker.xyz** file

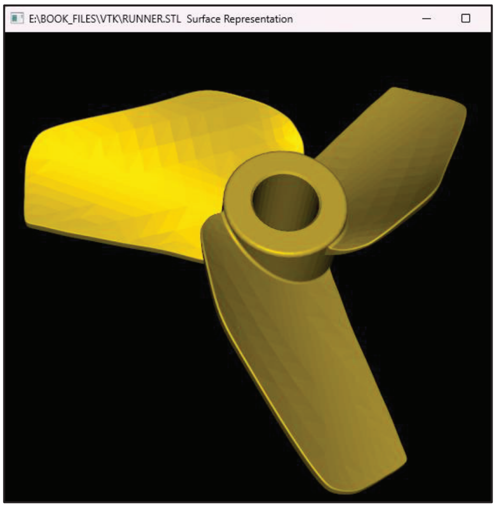


Figure 24.13 The **Surface** view of **Runner.STL** file

25. Executable Python

The chapter exemplifies the creation of the **Python** executable files, using the following libraries: **cx_Freeze** and **Inno Setup**. The **Inno Setup** library can be found at [**25.1**]; download **innosetup-6.3.3.exe** file and click on it to install.

To install **cx_Freeze** go to [**20.2**], write **cx-Freeze** in **Search** control and select **cx-Freeze 7.2.0** version to download **cx_Freeze-7.2.0-cp312-cp312-win_amd64.whl** file; then open command prompt, go to folder where the **whl** was saved and write the following command:

```
pip2.7 install cx_Freeze-7.2.0-cp312-cp312-win_amd64.whl
```

We will create executables file for **Magic_Square.pyw** script and for **Python 2.7** version. The following steps must be follow to create **Python** executable files:

❶ create a folder installation with any name;

❷ copy the **Magic_Square.pyw** and **python.ico** files in this folder;

❸ using any text editor create **cx_freeze_setup.py** file in the same folder, **Listing 25.1**;

❹ open command prompt write the following command:

```
C:\Python27\python.exe cx_freeze_setup.py build
```

❺ at the process end , the message ***The Magic_Square files are created in folder: 'build' !*** will inform that the **build\exe.win32-2.7** folder was created in the folder installation with files required to create executable file;

❻ using any text editor create **License.txt** file in the same folder, **Listing 25.2**;

❼ using any text editor create **Readme.txt** file in the same folder, **Listing 25.3**; the content of this file may be different from that proposed;

❽ create **InnoSetupFile.iss** file in the same folder, **Listing 25.4**; this file can be created with any text editor, but recommandation is to use **Inno Setup** wizard; open **InnoSetup**, **Figure 25.1** and the wizard start automatically, **Figure 25.2**; also, a new file can be created through **File** => **New** option, **Figure 25.3**, followed by click on **Next** button;

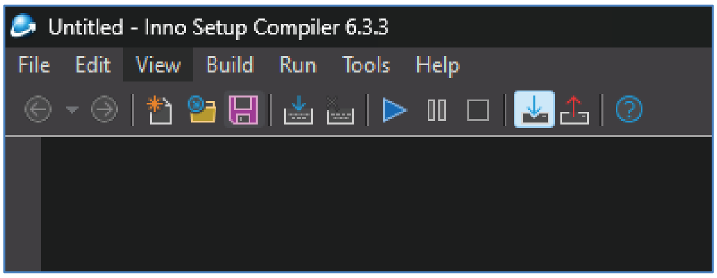
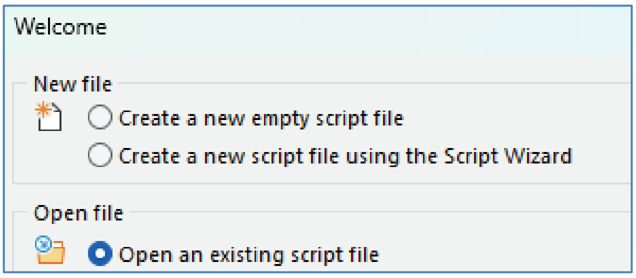


Figure 25.1 The **Inno Setup** interface


Welcome

New file

○ Create a new empty script file

○ Create a new script file using the Script Wizard

Open file

● Open an existing script file

Figure 25.2 Create script in **Inno Setup**

❾ write the informations from **Figure 25.4**: **Application name, version**, **publisher** & **website** and click **Next**;

❶⓿ write the informations from **Figure 25.5**: **Destination folder** & **Folder name** and click **Next**; activate or no the bottom options: **Allow user to change the application folder** & **The application doesn't need a folder**; click **Next**;

❶❶ write the informations from **Figure 25.6**: **Application main executable file**; here must be selected the previous file created by **cx_freeze** which is available in the following folder:

```
...\build\exe.win32-2.7\Magic_Square.exe
```

also, through **Add files** it is posible to add additional files like icons, databases or folders; click **Next**;

❶❷ write the informations from **Figure 25.7**; click **Next**;

❶❸ select the options from **Figure 25.8**; click **Next**;

❶❹ select the filess from **Figure 25.9**: **License.txt** & **Readme.txt**; click **Next**;

❶❺ select the options from **Figure 25.10**; click **Next**;

❶❻ select the options from **Figure 25.11**; click **Next**;

❶❼ select the options from **Figure 25.12**; click **Next**;

❶❽ write the informations from **Figure 25.13**; click **Next**;

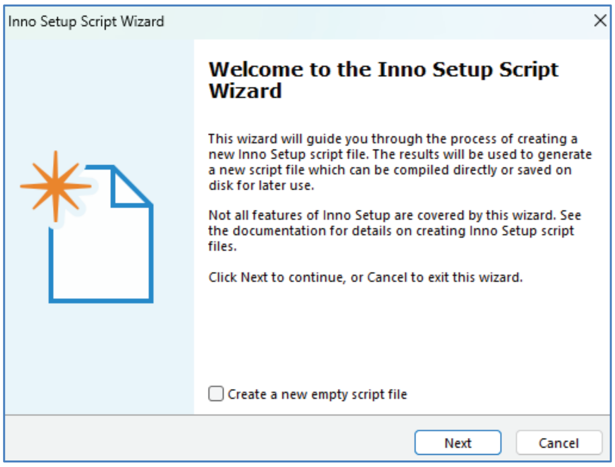
❶❾ select the options from **Figure 25.14**; click **Next**;

❷⓿ rewiew the message from **Figure 25.15**; click **Finish**;

❷❶ from menu select **Build** => **Compile** or **Ctrl+F9**;

❷❷ The installation file **Magic_Square_Setup.exe** will be created in the **Output** folder inside the installation folder;

❷❸ click on the **Magic_Square_Setup** file to start installation.



Welcome to the Inno Setup Script Wizard

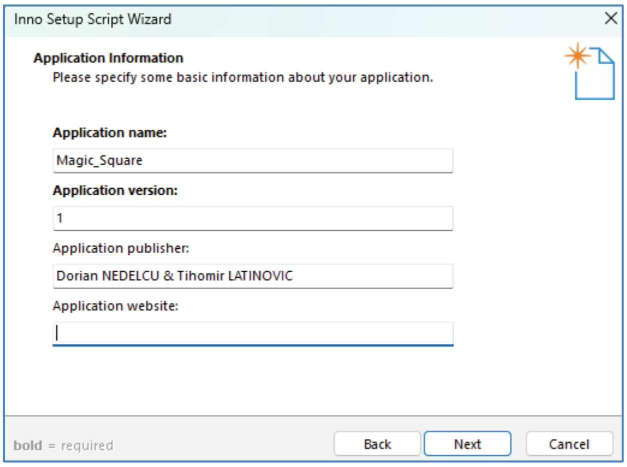
This wizard will guide you through the process of creating a new Inno Setup script file. The results will be used to generate a new script file which can be compiled directly or saved on disk for later use.

Not all features of Inno Setup are covered by this wizard. See the documentation for details on creating Inno Setup script files.

Click Next to continue, or Cancel to exit this wizard.

☐ Create a new empty script file

Figure 25.3 The **Wizard** first step



Application Information
Please specify some basic information about your application.

Application name: Magic_Square

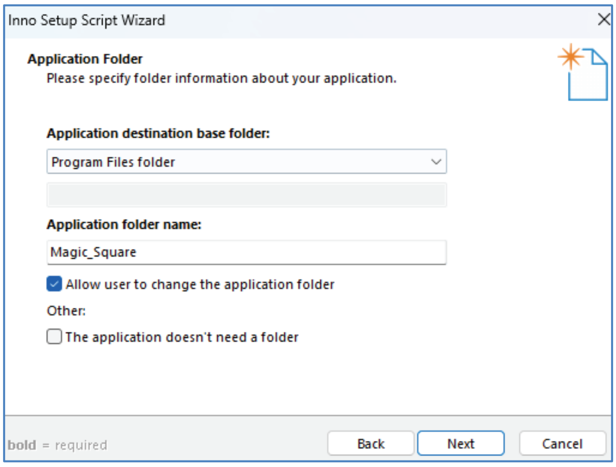
Application version: 1

Application publisher: Dorian NEDELCU & Tihomir LATINOVIC

Application website:

bold = required

Figure 25.4 The **Wizard** second step



Application Folder
Please specify folder information about your application.

Application destination base folder: Program Files folder

Application folder name: Magic_Square

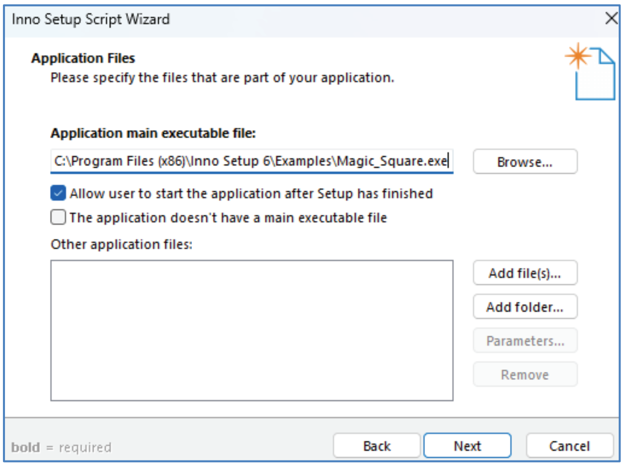
☑ Allow user to change the application folder

Other:

☐ The application doesn't need a folder

bold = required

Figure 25.5 The **Wizard** third step



Application Files
Please specify the files that are part of your application.

Application main executable file: C:\Program Files (x86)\Inno Setup 6\Examples\Magic_Square.exe

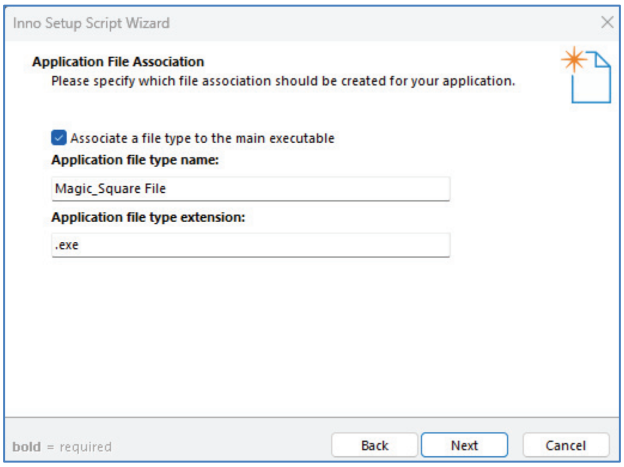
☑ Allow user to start the application after Setup has finished

☐ The application doesn't have a main executable file

Other application files:

bold = required

Figure 25.6 The **Wizard** four step



Application File Association
Please specify which file association should be created for your application.

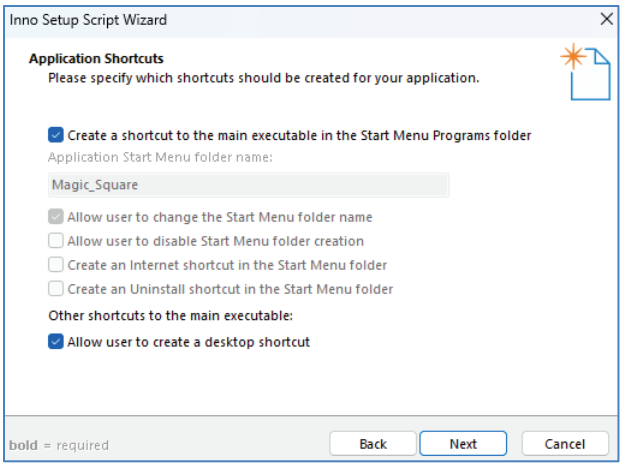
☑ Associate a file type to the main executable

Application file type name: Magic_Square File

Application file type extension: .exe

bold = required

Figure 25.7 The **Wizard** five step



Application Shortcuts
Please specify which shortcuts should be created for your application.

☑ Create a shortcut to the main executable in the Start Menu Programs folder

Application Start Menu folder name: Magic_Square

☑ Allow user to change the Start Menu folder name

☐ Allow user to disable Start Menu folder creation

☐ Create an Internet shortcut in the Start Menu folder

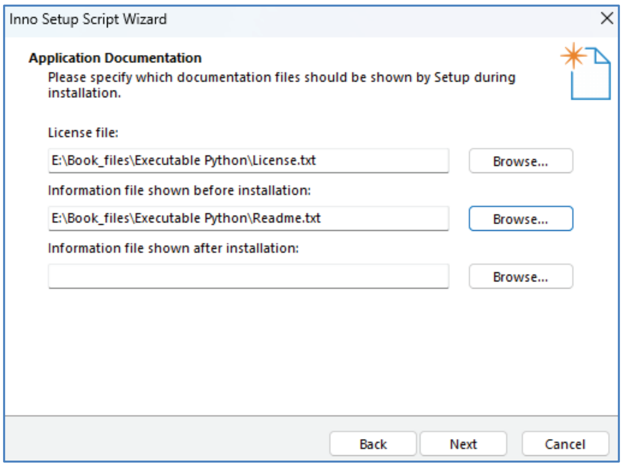
☐ Create an Uninstall shortcut in the Start Menu folder

Other shortcuts to the main executable:

☑ Allow user to create a desktop shortcut

bold = required

Figure 25.8 The **Wizard** six step



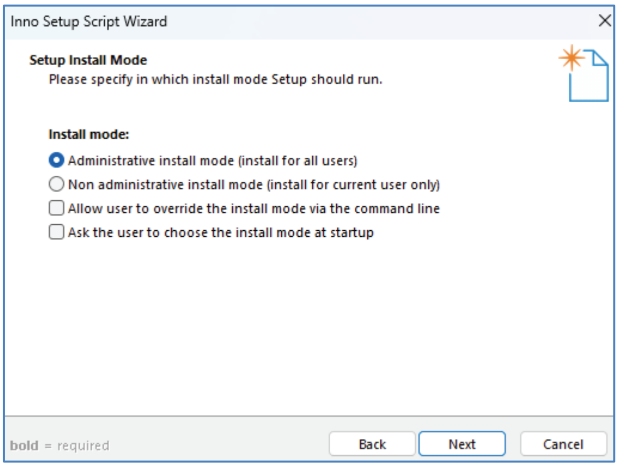
Application Documentation

Please specify which documentation files should be shown by Setup during installation.

License file:
E:\Book_files\Executable Python\License.txt

Information file shown before installation:
E:\Book_files\Executable Python\Readme.txt

Information file shown after installation:



Setup Install Mode

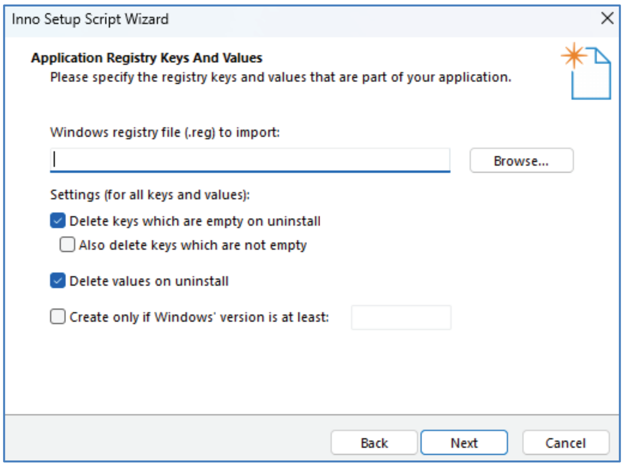
Please specify in which install mode Setup should run.

Install mode:

● Administrative install mode (install for all users)
○ Non administrative install mode (install for current user only)
□ Allow user to override the install mode via the command line
□ Ask the user to choose the install mode at startup

Figure 25.9 The Wizard seven step

Figure 25.10 The Wizard eight step



Application Registry Keys And Values

Please specify the registry keys and values that are part of your application.

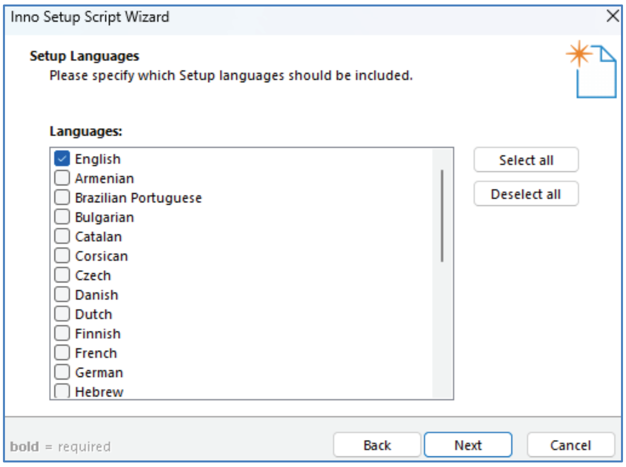
Windows registry file (.reg) to import:

Settings (for all keys and values):

☑ Delete keys which are empty on uninstall
□ Also delete keys which are not empty

☑ Delete values on uninstall

□ Create only if Windows' version is at least:



Setup Languages

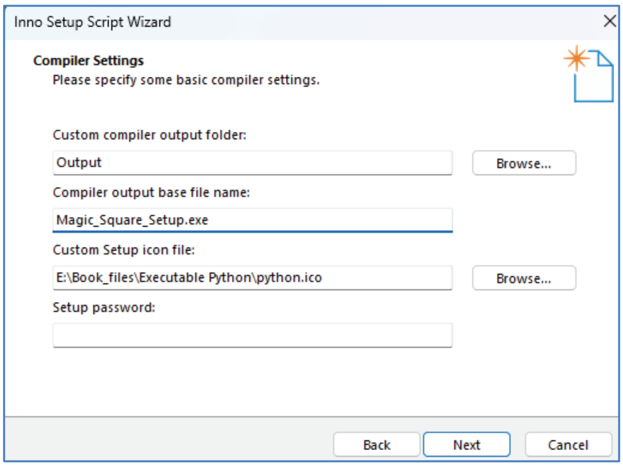
Please specify which Setup languages should be included.

Languages:

☑ English
□ Armenian
□ Brazilian Portuguese
□ Bulgarian
□ Catalan
□ Corsican
□ Czech
□ Danish
□ Dutch
□ Finnish
□ French
□ German
□ Hebrew

Figure 25.11 The Wizard nine step

Figure 25.12 The Wizard ten step



Compiler Settings

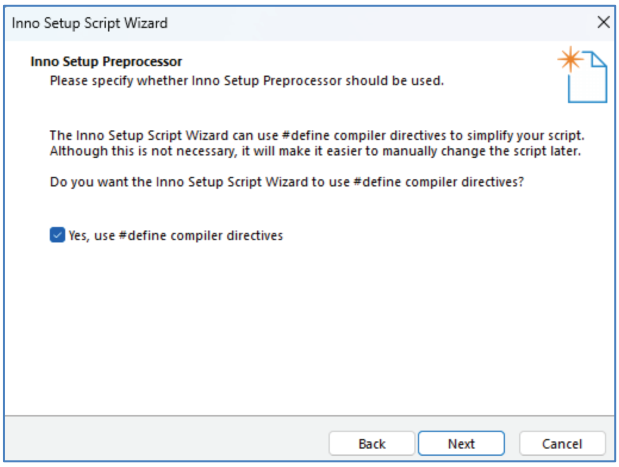
Please specify some basic compiler settings.

Custom compiler output folder:
Output

Compiler output base file name:
Magic_Square_Setup.exe

Custom Setup icon file:
E:\Book_files\Executable Python\python.ico

Setup password:



Inno Setup Preprocessor

Please specify whether Inno Setup Preprocessor should be used.

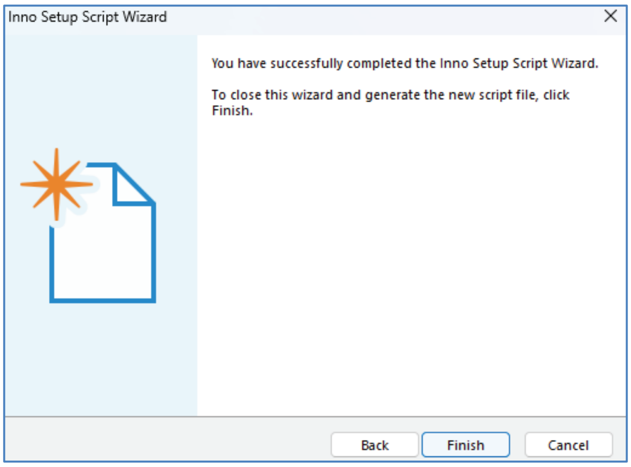
The Inno Setup Script Wizard can use #define compiler directives to simplify your script. Although this is not necessary, it will make it easier to manually change the script later.

Do you want the Inno Setup Script Wizard to use #define compiler directives?

☑ Yes, use #define compiler directives

Figure 25.13 The Wizard eleven step

Figure 25.14 The Wizard twelve step



You have successfully completed the Inno Setup Script Wizard.

To close this wizard and generate the new script file, click Finish.

Figure 25.15 The **Wizard** thirteen step

Listing 25.1 cx_freeze_setup.py – Executable Python – Python & wxPython version

```
from cx_Freeze import setup, Executable

exe = Executable(
    script="Magic_Square.pyw",
    base="Win32GUI",
    )

setup(
    name="Magic_Square",
    version="1.0",
    description="cx_Freeze script for Magic_Square application",
    executables=[exe]
)
print "The Magic_Square files are created in folder:  'build' !"
```

Listing 25.2 License.txt – Executable Python – Python & wxPython version

```
License CC0 1.0
You can copy, modify, distribute and perform the work, even for commercial purposes, all without asking permission.

THE APPLICATION DOES NOT SHARE DATA OVER THE INTERNET, THE CODE
RUNS ENTIRELY ON THE USER'S COMPUTER, DOES NOT CONTAIN ANY VIRUSES AND NOT
STORE DATA TO ANY EXTERNAL SERVER.
```

Listing 25.3 Readme.txt – Executable Python – Python & wxPython version

```
Magic_Square software ; Vers. 1.0 - @ 2024
@ 2024 - Dorian Nedelcu*  & Tihomir Latinovic**

*  Former Prof. Dorian Nedelcu
   Member of Bosnia and Herzegovina American Academy od Arts and Science (BHAAAS)
   Babes-Bolyai University of Cluj-Napoca, Romania
   Faculty of Engineering,
   Department of Engineering Sciences,
   Traian Vuia square, no.1-4, 320085, Resita.
   dorian.nedelcu@ubbcluj.ro
   ne_dor@yahoo.com
   ORCID ID: orcid.org/ 0000-0002-4927-1042

** Prof. Doctor Tihomir Latinovic
   Member of Bosnia and Herzegovina American Academy od Arts and Science (BHAAAS)
   Member of Western Balkan Alumni Association (WBAA)
   University of Vitez
   Middle-Bosnian canton, Bosnia and Herzegovina
   Faculty of Information Technology
   e-mail: tihomirlatinovic@live.com
   University web:  https://unvi.edu.ba/
   Personal web: https://tihomirlatinovic.wordpress.com/cv-english/
   ORCID ID: orcid.org/0000-0003-3682-6464
```

Listing 25.3 Readme.txt – Executable Python – Python & wxPython version

```

Magic_Square - A Python module to create a Magic Square.
A magic square is a matrix for which the sum of the elements on the rows,
columns and the two diagonals (main and secondary) are equal.

Magic_Square - License CC0 1.0
---------------------------------
You can copy, modify, distribute and perform the work, even for commercial purposes,
all without asking permission.

The software packages
--------------------
The Magic_Square software is created with the following free and Open Source resources:

= Python 2.7 - a high-level programming language. Python runs on Windows, Linux/Unix,
            Mac OS X, and has been ported to the Java and .NET virtual machines. Python is
            free to use, even for commercial products, because of its OSI-approved open source
            license. Python is an interpreted, interactive, object-oriented programming language.
= wxPython - a graphical user interface toolkit for the Python programming language.
            It allows Python programmers to create programs with a robust, highly functional
            graphical user interface, simply and easily.
= comtypes - a lightweight Python COM package, based on the ctypes FFI library.
            comtypes allows to define, call, and implement custom and dispatch-based
            COM interfaces in pure Python. It works on Windows and 64-bit Windows.

If the user allready have installed these modules on the computer, the application can
be run by launching the compiled file 'Magic_Square.pyc', whithout use of the installer
Magic_Square_Setup.exe. In this case, the file Magic_Square.pyc file and all application files,
except 'Magic_Square.exe' file, must exist in the same folder.

Application files
----------------
Magic_Square.exe   The application installation kit
Magic_Square.pyc   The application compiler file (can be used only if the
              software packages are installed on the computer).
Readme.txt       This file.
License.txt      The license file.


THE APPLICATION DOES NOT SHARE DATA OVER THE INTERNET, THE CODE
RUNS ENTIRELY ON THE USER'S COMPUTER, DOES NOT CONTAIN ANY VIRUSES AND NOT
STORE DATA TO ANY EXTERNAL SERVER.

Magic_Square Installation through the installation kit 'Magic_Square_Setup.exe'
---------------------------------------------------------------------------

The application was generated and tested through Python 2.7 version installed by
"Python(x,y)" package (https://python-xy.github.io/. ) on WINDOWS 10 operating
system, comtypes-0.6.2.

The cx_Freeze application was used to produces a folder containing an executable
file for the program, along with the shared libraries (DLLs or others files) needed
to run it. The installation kit is created through the Inno Setup application and
delivered as a single file "Magic_Square_Setup.exe".

1. The application will be installed via the "Magic_Square_Setup.exe" file.
   All application files will be installed by the application installation kit.
   The application files will be installed on 'Program Files (x86)' folder.
2. An icon will be placed on the desktop to open the application.
3. Launching the application is done by double clicking on the main file
   "Magic_Square.exe" from the install folder 'Program Files (x86)'  or on the
    desktop icon created by installer.
```

Listing 25.4 InnoSetupFile.iss – Executable Python – Python & wxPython version

```
; Script generated by the Inno Setup Script Wizard.
; SEE THE DOCUMENTATION FOR DETAILS ON CREATING INNO SETUP SCRIPT FILES!

#define MyAppName "Magic_Square"
#define MyAppVersion "1.0"
#define MyAppPublisher "Dorian NEDELCU & Tihomir LATINOVIC"
#define MyAppExeName "Magic_Square.exe"

[Setup]
; NOTE: The value of AppId uniquely identifies this application. Do not use the same AppId value in installers for other applications.
; (To generate a new GUID, click Tools | Generate GUID inside the IDE.)
AppId={{B5A2B9D9-6237-4041-B53C-824E07D24587}
AppName={#MyAppName}
AppVersion={#MyAppVersion}
;AppVerName={#MyAppName} {#MyAppVersion}
AppPublisher={#MyAppPublisher}
DefaultDirName={autopf}\{#MyAppName}
DisableDirPage=yes
DisableProgramGroupPage=yes
LicenseFile=E:\Book_files\Executable Python\License.txt
InfoBeforeFile=E:\Book_files\Executable Python\Readme.txt
; Uncomment the following line to run in non administrative install mode (install for current user only.)
;PrivilegesRequired=lowest
OutputBaseFilename=Magic_Square_Setup
SetupIconFile=E:\Book_files\Executable Python\python.ico
Compression=lzma
SolidCompression=yes
WizardStyle=modern

[Languages]
Name: "english"; MessagesFile: "compiler:Default.isl"

[Tasks]
Name: "desktopicon"; Description: "{cm:CreateDesktopIcon}"; GroupDescription: "{cm:AdditionalIcons}"; Flags: unchecked

[Files]
Source: "E:\Book_files\Executable Python\build\exe.win32-2.7\{#MyAppExeName}"; DestDir: "{app}"; Flags: ignoreversion
Source: "E:\Book_files\Executable Python\build\exe.win32-2.7\*"; DestDir: "{app}"; Flags: ignoreversion recursesubdirs
createallsubdirs
; NOTE: Don't use "Flags: ignoreversion" on any shared system files

[Icons]
Name: "{autoprograms}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"
Name: "{autodesktop}\{#MyAppName}"; Filename: "{app}\{#MyAppExeName}"; Tasks: desktopicon

[Run]
Filename: "{app}\{#MyAppExeName}"; Description: "{cm:LaunchProgram,{#StringChange(MyAppName, '&', '&&')}}"; Flags: nowait
postinstall skipifsilent
4
```

Scripts Index

No.	*Script* / **Folder**	Python 2.7	Python 3.11	Streamlit Python 3.12
1	*Integration by rectangles method /* **Integrate**	Integrate.pyw	X	Integrate.py
2	*Integration by Gauss method /* **Gauss**	Gauss.pyw	X	Gauss.py
3	*NACA hydrodynamic profile* **NACA profile**	NACA.pyw	X	NACA.py
4	*Recursive functions* **Recursive functions**	Generate_Sequence.pyw	X	Generate_Sequence.py Turtle.py
5	*Cylinder unfold /* **Unfold**	Cylinder.pyw	X	Cylinder.py
6	*Elbow unfold /* **Elbow**	Elbow.pyw	X	Elbow.py
7	*Flatten surface /* **Flatten surface**	Surfaces.pyw	X	Surfaces.py
8	*Rotating a 3D geometry /* **Rotate**	Rotate.pyw	X	Rotate.py
9	*Generate the magic square* **Magic_Square**	Magic_Square.pyw	X	Magic_Square.py
10	*Spline Interpolation* **Spline Interpolation**	Interpolation.pyw	X	Interpolation.py
11	*Curve vectorization* **Curve vectorization**	Vectorization.pyw	X	X
12	*SQLite Database /* **SQLite database**	DataBase.pyw	X	DataBase.py
13	*MySQL Database /* **MySQL Database**	MySQL DataBase.pyw Transfer.pyw	X	X
14	*Image Viewer – Matplotlib & OpenCV /* **Image Viewer**	Image_Viewer.pyw	X	Image_Viewer.py
15	*Image warp – OpenCV /* **Image Warp**	Image Warp.pyw	X	X
16	*Detect faces & eyes with Open CV /* **Detect faces & eyes**	X	X	Detect_faces_eyes.py
17	*PDF file operations /* **PDF files**	PDF operations.pyw	X	X
18	*WEB tools* **WEB Tools/Python 2.7 ; WEB Tools**	WEB Tools 2.7.pyw	WEB Tools 3.11.pyw	X
19	*Optical Character Recognition /* **OCR**	X	OCR.pyw	X
20	*Open 3D /* **Open 3D**	X	Open_3D_wx_Python.pyw	X
21	*VTK /* **VTK**	X	VTK.pyw	X
22	*Executable Python* **Executable Python/Output**	Magic_Square_Setup.exe	X	X

REFERENCES

[2.1] https://unze.ba/am/rg/2018/5958.pdf
[2.2] https://www.sk.rs/2006/02/skpr01.html
[2.3] https://www.automatika.rs/vesti/vesti-obrada-signala/unapredenje-sistema-za-face-recognition.html
[2.4] https://www.kpu.edu.rs/data/materijali/biometrijske%20identifications/Slika_Lica.pdf
[2.5] https://trizgyrus.com/howstuff/Howstuffworks How Facial RecognitionSystems Work.htm
[2.6] https://www.ucg.ac.me/skladiste/blog_7446/postojva_22478/fajlovi/Recognition%20face%20and%voice.pdf
[2.7] https://repozitorij.mathos.hr/islandora/object/mathos%3A93/datastream/PDF/view
[2.8] https://www.intechopen.com/books/ face-recognition-semisupervised-classification-subspace-projection-and-evaluation-methods/face-recognition-issues-methods-and-alternative-applications
[2.9] https://arxiv.org/pdf/1909.01815.pdf

[3.1] https://python-xy.github.io/
[3.2] https://sourceforge.net/projects/pywin32/files/pywin32/Build%20214/pywin32-214.win32-py2.7.exe/download
[3.3] https://sourceforge.net/projects/wxpython/files/wxPython/3.0.0.0/
[3.4] https://code.visualstudio.com/
[3.5] https://docs.streamlit.io/deploy/streamlit-community-cloud/deploy-your-app
[3.6] D. Nedelcu, T. Latinovic, L. Sikman, M. Todic, A. Majstorovic - *Using Streamlit and Basic4Android (B4A) to create the* same *application - Streamlit version*, The Book of Abstarcts, International Conference on Applied Sciences, ICAS 2024, Travnik, May-June 2024.
[3.7] https://pypi.org/

[10.1] E. Birtarescu, V. C. Campian, D. Nedelcu - *Strength calculations performed on the spiral casing of a Francis turbine operating in secondary control regime*, U.P.B. Sci. Bulletin, Series D, Vol. 83, Iss. 2, ISSN 1454-2358, 2021.
[10.2] D. Nedelcu, I. Padurean - *Computer aided design of an Draft Tube's Elbow for hydraulic turbine using Microstation,* Scientific Bulletin of the "Politehnica" University of Timisoara, Transactions on Mechanics, 2007.

[13.1] https://numpy.org/devdocs/reference/generated/numpy.interp.html#numpy.interp
[13.2] J. Kiusalaas - *Numerical methods in engineering with Python*, Cambridge University Press, New York, ISBN-13 978-0-521-19132-6, Second Edition, 2010.
[13.3] https://gist.github.com/komasaru/f46775fc77753fd48fdae106b3ee01a2
[13.4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.splprep.html
[13.5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html#r0cc18619484f-1
[13.6] https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.Akima1DInterpolator.html
[13.7] https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.PchipInterpolator.html

[14.1] D. Nedelcu, T. Latinovic, L.Sikman – *PyDigitizer – A Python module for digitizing 2D curves*, Journal of Physics: Conference Series Volume: 2540 Issue: 1, Published: July, 2023, https://iopscience.iop.org/article/10.1088/1742-6596/2540/1/012015/pdf
[14.2] PyDigitizer - A Python module for digitizing 2D curves, https://www.youtube.com/watch?v=WifxfTgQKcY
[14.3] Dorian Nedelcu, Tihomir Latinovic - *PyDigitizer – A Python module for digitizing 2D curves*, https://data.mendeley.com/datasets/jnwyr7jzrd/1
[14.4] D. Nedelcu, T. Latinovic - *PyChart – A Python module for analysis and visual view of 2D/3D Charts*, International Conference on Applied Sciences (ICAS 2020), IOP Publishing, 1781 (2021) 012044, doi:10.1088/1742-6596/1781/1/012044.
[14.5] D. Nedelcu, T. Latinovic - *PyChart_Python_Application*, Mendeley Data, V1, DOI: http://dx.doi.org/10.17632/35jrmwd4fw.1, Published: 28 June 2020.

[**15.1**] https://github.com/jpwhite3/northwind-SQLite3?tab=readme-ov-file
[**15.2**] https://www.sqliteexpert.com/download.html
[**15.3**] https://sqlitebrowser.org/dl/
[**15.4**] https://sqlitestudio.pl/
[**15.5**] https://docs.streamlit.io/get-started/fundamentals/main-concepts

[**19.1**] https://realpython.com/face-recognition-with-python/
[**19.2**] https://stackabuse.com/facial-detection-in-python-with-opencv/
[**19.3**] https://blog.devgenius.io/face-and-eyes-detection-using-python-b88c679d9a05
[**19.4**] https://medium.com/@makiex/face-and-eye-detection-with-opencv-1be56c40893e
[**19.5**] https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html#cascadeclassifier-detectmultiscale
[**19.6**] https://stackoverflow.com/questions/36218385/parameters-of-detectmultiscale-in-opencv-using-python
[**19.7**] https://www.pexels.com/

[**20.1**] https://www.reportlab.com/
[**20.2**] https://pypi.org/project/pypdf/

[**21.1**] https://www.python.org/downloads/windows/
[**21.2**] https://python-visualization.github.io/folium/latest/
[**21.3**] https://www.gps-coordinates.net/map/country/BA

[**22.1**] https://github.com/UB-Mannheim/tesseract/wiki
[22.2] https://tesseract-ocr.github.io/tessdoc/Data-Files-in-different-versions.html
[**22.3**] A.Rosebrock, A. Thanki, S. Paul, J. Haase - *OCR with OpenCV, Tesseract, and Python*, Intro to OCR - 1st Edition (version 1.0), © 2020 PyImageSearch

[**23.1**] https://www.open3d.org/
[**23.2**] Qian-Yi Zhou, Jaesik Park, Vladlen Koltun, *Open3D - A Modern Library for 3D Data Processing*, arXiv:1801.09847, 2018
[**23.3**] https://pypi.org/
[**23.4**] http://vechi.uem.ro/fileadmin/3_ACADEMICA/1_FACULTATI/1_FIM/cadre/NedelcuD/Aplicatii/_index.html

[**24.1**] https://vtk.org/

[**25.1**] https://jrsoftware.org/isdl.php

UNIVERSITAS CLAUDIOPOLITANA
ROMANIA
PRESA UNIVERSITARĂ

ISBN: 978-606-37-2500-5