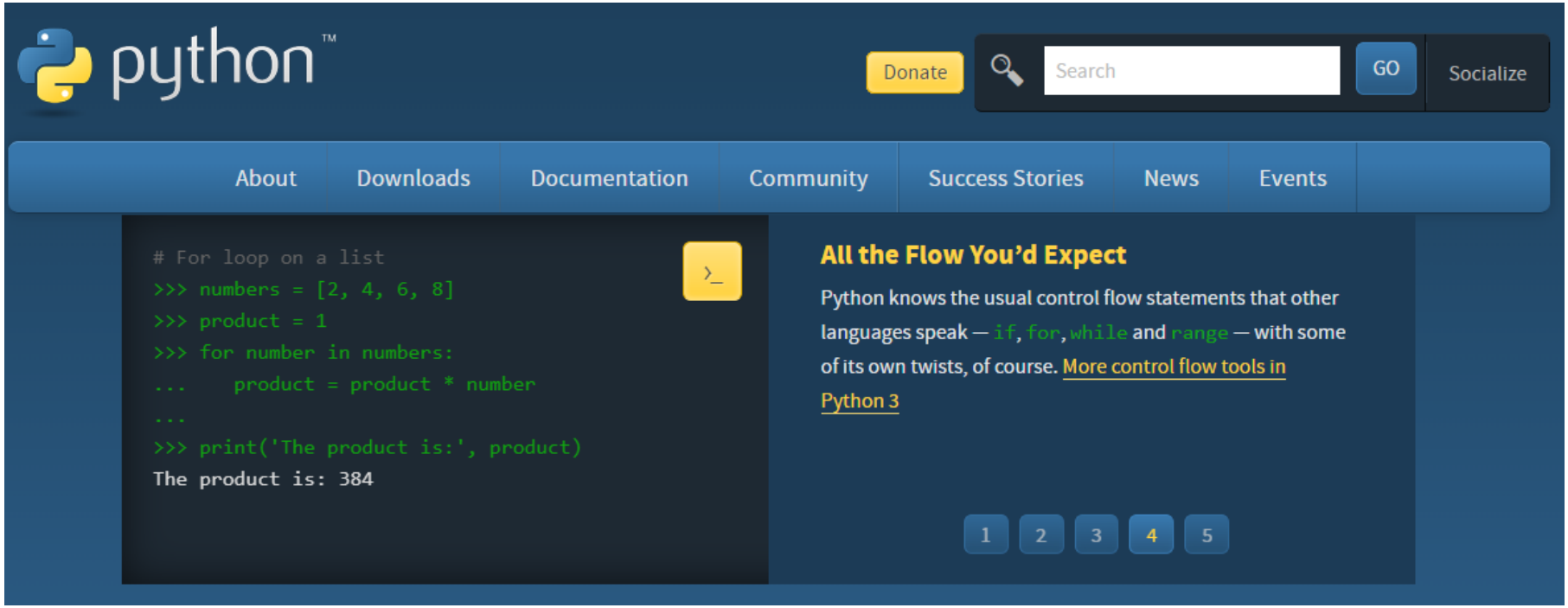
python™

https://www.python.org/



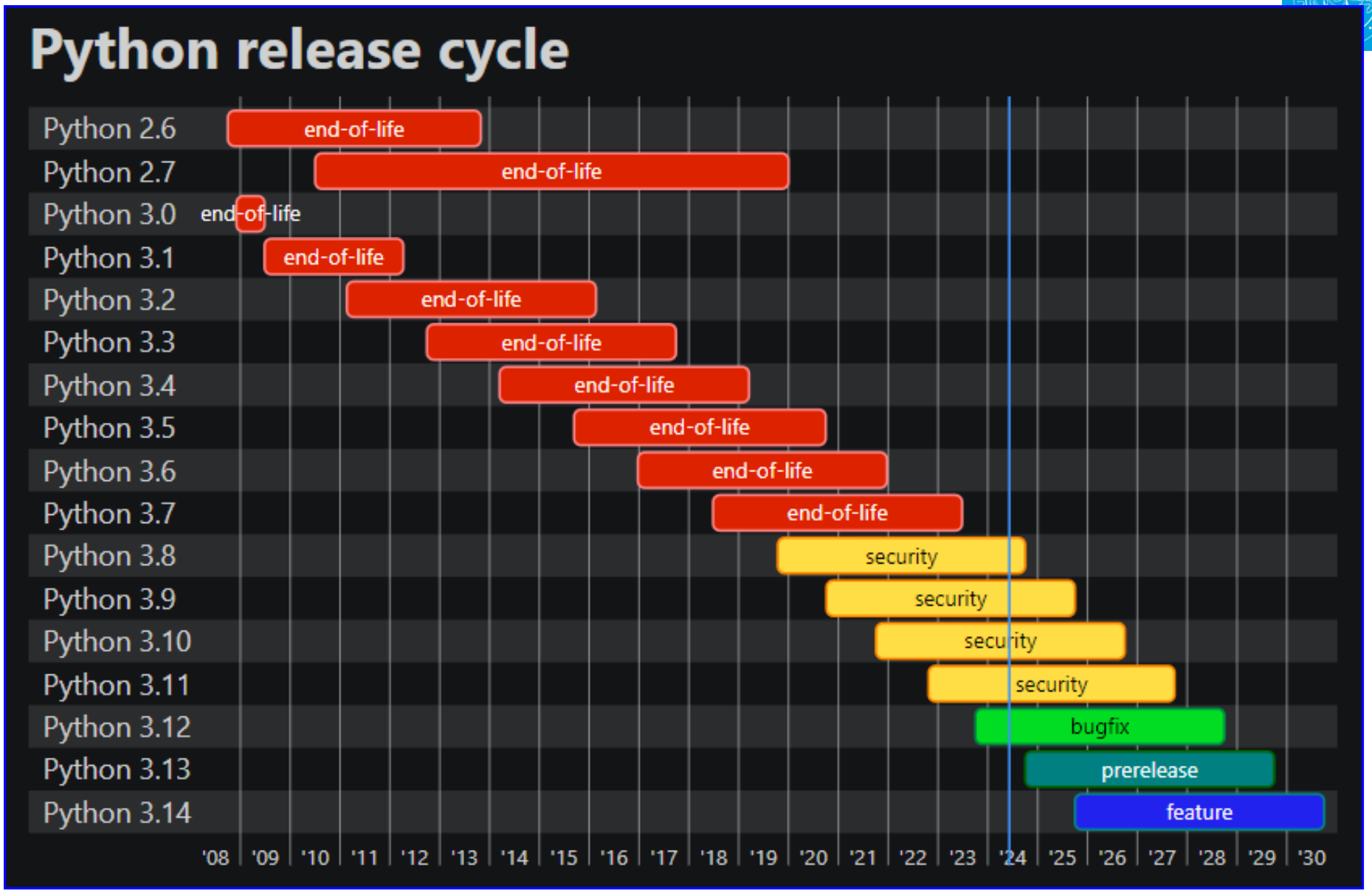
https://dorian-nedelcu.streamlit.app/

TIOBE Index for June 2024

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors.*

Jun 2024	Jun 2023	Change	Programming Language	Ratings	Change
1	1		Python	15.39%	+2.93%
2	3	↑	C++	10.03%	-1.33%
3	2	↓	C	9.23%	-3.14%
4	4		Java	8.40%	-2.88%
5	5		C#	6.65%	-0.06%
6	7	↑	JavaScript	3.32%	+0.51%
7	14	⇈	Go	1.93%	+0.93%
8	9	↑	SQL	1.75%	+0.28%
9	6	↓	Visual Basic	1.66%	-1.67%
10	15	⇈	Fortran	1.53%	+0.53%
11	11		Delphi/Object Pascal	1.52%	+0.27%
12	19	⇈	Swift	1.27%	+0.33%
13	10	↓	Assembly language	1.26%	-0.03%
14	12	↓	MATLAB	1.26%	+0.14%
15	8	⇊	PHP	1.22%	-0.52%

* *https://www.tiobe.com/tiobe-index/*



https://devguide.python.org/versions/

History of Python

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language developed by Guido van Rossum during 1985- 1990 at the National Research Institute for Mathematics and Computer Science in the Netherlands.

- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, Unix shell, and other scripting languages.

- Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

- Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

https://khu.ac.ir/documents/10545364/20813987/Python-Tutorial.pdf

Overview

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

- Python is available on a wide variety of platforms including Windows, Linux and Mac OS X.

https://khu.ac.ir/documents/10545364/20813987/Python-Tutorial.pdf

Python Features

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.
- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

https://khu.ac.ir/documents/10545364/20813987/Python-Tutorial.pdf

IDLE	IDLE (Integrated Development and Learning Environment) is a dedicated platform or software to develop Python applications	https://docs.python.org/3/library/idle.html
PyCharm 2024.1.3	The Python IDE for data science and web development	https://www.jetbrains.com/pycharm/
Visual Studio Code	VS Code an excellent Python editor	https://code.visualstudio.com/
eclipse	Python Programming in the Eclipse IDE	https://eclipseide.org/
spyder	Spyder is a free and open source scientific environment written in Python, for Python	https://www.spyder-ide.org/
PyScripter 4.3.4.0 Jim McKeeth	PyScripter is an Integrated Development Environment (IDE) specifically for Python	https://pyscripter.en.uptodown.com/windows
WING PYTHON IDE THE INTELLIGENT DEVELOPMENT ENVIRONMENT FOR PYTHON	Wing Python IDE was designed from the ground up for Python, for a more productive development experience.	https://wingware.com/
Th	Development environment for Python coding	https://thonny.en.softonic.com/
Atom	Atom is a Github-developed open-source code editor that may be used for Python programming	https://atom.en.softonic.com/download
Sublime Text	Source code editor which support many programming and markup languages. However, its Python support is considered the best	https://www.sublimetext.com/
Ulipad python	UliPad is a flexible editor, based on wxPython	https://code.google.com/archive/p/ulipad/ https://code.google.com/archive/p/ulipad/downloads

https://www.geeksforgeeks.org/top-python-ide/

Python Applications Editors

	Python 2	Python 3
Release date	2000	2008
Syntax	More complex and difficult to interpret	Readable and easily understandable
Performance	Slower performance due to design flaws	Improved performance of the code's runtime compared to Python 2
print function	print "Welcome to Datacamp"	print ("Welcome to Datacamp")
Integer division	The result is an integer value. Decimals are always truncated	The result is always a float value
Unicode support	Uses by default ASCII characters. To store Unicode values, you need to define them using "u" prefix	The default storing of strings is Unicode
Backward compatibility	Relatively easy to port Python2 to Python 3.	Python 3 is not backwardly compatible with python 2
Libraries	Many older libraries for Python 2 are not forward-compatible	Most of the new libraries for Python 3 cannot be used in python 2

https://www.datacamp.com/blog/python-2-vs-3-everything-you-need-to-know

Python GUI Frameworks - Using a GUI framework, you can create the graphical version of an application.

Tkinter - Tkinter comes preinstalled with Python, so there's no need to install anything else to create a GUI application. **Tkinter** is very easy to learn and use. Each **Tkinter** widget includes a different level of customization. Include widgets for things like frames, buttons, check buttons, labels, file dialogs, and canvas. **Tkinter** is an open-source library and offers a syntax that is very much in line with the simplicity of Python code.

wxPython - This Python GUI library simplifies the process of creating native-looking UIs without adding extra overhead to the application. **wxPython** can create applications for Linux, macOS, UNIX, and Windows. **wxPython** includes a large number of widgets, all of which look very good across every supported platform without the need to customize them. https://wxpython.org/index.html https://zetcode.com/wxpython/

PyQT5 - PyQT5 is probably one of the most popular Python GUI frameworks on the market. Built around the **PyQT** package, this framework makes it easier to create all types of applications for just about any platform. **PyQT5** supports Android, iOS, Linux, macOS, and Windows. What makes **PyQT5** stand out is its use of QtGUI and QtDesigner, which provide for the drag-and-drop means of implementing visual elements for your GUI applications. Even better, if your Python developers prefer to program those elements via code, they have that option as well. **PyQT5** must be installed separately from Python. https://pypi.org/project/PyQt5/

Kivy - Kivy is a framework designed for creating more modern-looking interfaces with Python. **Kivy** is an OpenGL accelerated framework that supports Android, iOS, Linux, macOS, and Windows. With over 20 widgets in its toolkit, **Kivy** is a fairly flexible option for creating the most intuitive user interfaces. **Kivy** was written in Python and Cython and can even build multi-touch applications that help to implement a natural user interface (NUI), which makes it easier for the user to easily learn the different types of interactions required for an application. https://kivy.org/

PySimpleGUI - PySimpleGUI is simple enough to use that even Python beginners can quickly create GUI applications. In fact, **PySimpleGUI** might be the easiest Python GUI framework on the market. So, if you have a number of new Python developers, this might be the perfect framework to get them started. One thing to keep in mind with **PySimpleGUI** is that it relies on other frameworks, specifically Qt, Tkinter, wxPython, and Remi. Because of this, developers can choose the GUI framework they want to use and will have immediate access to all of the elements that are included with their choice. This also makes **PySimpleGUI** fairly flexible, as it's not limited to a single GUI framework.

https://www.bairesdev.com/blog/best-python-gui-libraries/

Databases & SQL - Every large enterprise system uses a database for storing data. A *database* is a file that is organized for storing data. The primary data structures in a database are: *tables*, *rows*, and *columns*.

SQLite - **SQLite** is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. **SQLite** is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day. **SQLite** source code is in the public-domain and is free to everyone to use for any purpose. https://sqlite.org/

MySQL - **MySQL** is one of the most widely used and well known open-source RDBMS connectors. It employs a server/client architecture consisting of a multi-threaded SQL server. This allows **MySQL** to perform well because it easily utilizes multiple CPUs. MySQL was originally written in C/ C++ and then expanded to support various platforms. The key features of MySQL are scalability, security and replication. To use MySQL, you need to install its connector. In the command line, you can do that by running: `python -m pip install mysql-connector-python`

PostgreSQL - **PostgresSQL** is another open-source RDBMS connector that focuses on extensibility and uses a client/server database structure. In **PostgresSQL**, we call the communications managing the database files and operations "the Postgres process," which is where the library gets its name. To communicate with a **PostgresSQL** database, you need to install a driver that enables Python to do that. One commonly used driver is psycopg2. You can install it by running the following command-line instruction: `pip install psycopg2`

MongoDB - **MongoDB** is a well-known database data store among modern developers. It's an open-source document-oriented data storage system. We commonly use PyMongo to enable interaction between one or more MongoDB instances through Python code. MongoEngine is a Python Object Relational Mapping written for MongoDB on top of PyMongo. To use MongoDB, you need to install an engine and the actual **MongoDB** libraries.

`pip install pymongo==3.4.0` `pip install mongodb`

https://builtin.com/data-science/python-database

Plotting & Charts

Matplotlib - **Matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations in Python. **Matplotlib** makes easy things easy and hard things possible. https://matplotlib.org/

- Create publication quality plots.
- Make interactive figures that can zoom, pan, update.
- Customize visual style and layout.
- Export to many file formats.
- Embed in Graphical User Interfaces packages.
- Use a rich array of third-party packages built on Matplotlib.

Plotly - **Plotly's** Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heatmaps, subplots, multiple-axes, polar charts, and bubble charts. **Plotly** is free and open source . https://plotly.com/python/getting-started/

Bokeh - **Bokeh** is another web-based interactive visualization library that also has dashboarding Products built on top of it. It is a direct competitor to **Plotly** and is, frankly, incredibly similar in use, it has some stylistic differences. Either way, **Bokeh** is an extremely popular Python visualization package that Python users may be very comfortable using. https://bokeh.org/

Altair - **Vega-Altair** is a declarative visualization library for Python. Its simple, friendly and consistent API, built on top of the powerful Vega-Lite grammar, empowers you to spend less time writing code and more time exploring your data. https://altair-viz.github.io/

Scientific library

NumPy - **NumPy**, is one of the most broadly-used open-source Python libraries and is mainly used for scientific computation. Its built-in mathematical functions enable lightning-speed computation and can support multidimensional data and large matrices. It is also used in linear algebra. **NumPy Array** is often used preferentially over lists as it uses less memory and is more convenient and efficient. https://numpy.org/

Pandas - **Pandas** is an open-source library commonly used in data science. It is primarily used for data analysis, data manipulation, and data cleaning. **Pandas** allow for simple data modeling and data analysis operations without needing to write a lot of code. As stated on their website, pandas is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool. https://pandas.pydata.org/

SciPy - Fundamental algorithms for scientific computing in Python. **SciPy** provides algorithms for optimization, integration, interpolation, eigenvalue problems, algebraic equations, differential equations, statistics and many other classes of problems. Extends **NumPy** providing additional tools for array computing and provides specialized data structures, such as sparse matrices and k-dimensional trees. https://scipy.org/

OpenCV - **OpenCV** is the world's biggest computer vision library. It's open source, contains over 2500 algorithms and is operated by the non-profit Open Source Vision Foundation. https://opencv.org/

Python Libraries for Machine Learning

Scikit-learn - **Scikit-learn** is a very popular machine learning library that is built on NumPy and SciPy. It supports most of the classic supervised and unsupervised learning algorithms, and it can also be used for data mining, modeling, and analysis. **Scikit-learn**'s simple design offers a user-friendly library for those new to machine learning.

TensorFlow - **TensorFlow's** open-source Python library specializes in what's called differentiable programming, meaning it can automatically compute a function's derivatives within high-level language. Both machine learning and deep learning models are easily developed and evaluated with **TensorFlow's** flexible architecture and framework. **TensorFlow** can be used to visualize machine learning models on both desktop and mobile.

Theano - **Theano** is a Python library that focuses on numerical computation and is specifically made for machine learning. It is able to optimize and evaluate mathematical models and matrix calculations that use multi-dimensional arrays to create ML models. **Theano** is almost exclusively used by machine learning and deep learning developers or programmers.

Keras - **Keras** is a Python library that is designed specifically for developing neural networks for ML models. It can run on top of **Theano** and **TensorFlow** to train neural networks. **Keras** is flexible, portable, user-friendly, and easily integrated with multiple functions.

PyTorch - **PyTorch** is an open-source machine learning Python library based on the C programming language framework, **Torch**. It is mainly used in ML applications that involve natural language processing or computer vision. **PyTorch** is known for being exceptionally fast at executing large, dense data sets and graphs.

https://www.coursera.org/articles/python-machine-learning-library?campaignid=20882109092&adgroupid=&device=c&keyword=&matchtype=&network=x&devicemodel=&adposition=&creativeid=&hide_mobile_promo&gad_source=1

WEB frameworks

Django - **Django** is a free, open-source Python framework that enables rapid development of complicated code and applications by programmers. Python web developers can use it to create high-quality web apps. **Django** is widely used to construct APIs and web applications and is one of the top Python frameworks. Approximately 12,000 projects are reported to have been created in it. Popularity of this Python framework is due to its extensive library collection, reduced coding requirements, and reusability of components.

CherryPy - **CherryPy** is a lightweight, quick, and stable Python web development framework. It is open-source and can run on any Python-compatible framework. The **CherryPy** web framework enables the use of any data access and templating technology. It can perform every function of a web framework, including sessions, file uploads, static content, cookies, etc. **CherryPy** also enables developers to create web applications like they would any other object-oriented Python application. This reduces the time required to produce minor source code.

TurboGears - TurboGears is a Python framework for data-driven, full-stack web applications. It addresses common flaws in web and mobile app frameworks, enabling developers to design web apps with minimal configuration.

Web2Py - **Web2Py** includes a debugger, a code editor, and a deployment instrument for testing and maintaining web applications. It is a framework that supports multiple platforms, including Windows, Unix/Linux, Mac, Google App Engine, and others. Using a web server, a SQL database, and an online interface, the framework simplifies Python application development. It lets clients create, modify, deploy, and administer online applications through web browsers.

Flask - Inspired by the Sinatra Ruby framework, **Flask** is a Python framework available under the BSD license. The Werkzeug WSGI toolkit and Jinja2 template are utilized by **Flask**.

Tornado - **Tornado** is a Python web framework and unconventional library framework. It employs a non-blocking I/O framework. When properly configured, it can handle more than 10,000 simultaneous connections. This makes it an exceptional instrument for developing apps with a large number of concurrent users.

Streamlit - A faster way to build and share data apps. Streamlit turns data scripts into shareable web apps in minutes. All in pure Python. No front-end experience required.

https://www.browserstack.com/guide/top-python-web-development-frameworks

Python Distribution Kit

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.
https://python-xy.github.io/

Anaconda - **Anaconda** is an open-source distribution of the Python and **R programming** languages that is used for data science, machine learning, and artificial intelligence applications. It includes over 250 popular data science packages and management tools for simplifying package installation and deployment.
https://www.anaconda.com/

WinPython - **WinPython** is a free open-source **portable** distribution of the Python programming language for Windows 10/11(***) and scientific and educational usage. **WinPython** is a portable application, so the user should not expect any integration into Windows explorer during installation.
https://winpython.github.io/#overview

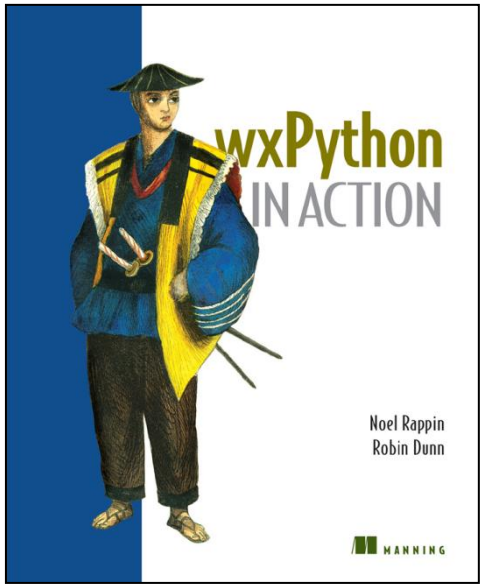
IronPython - **IronPython** is an open-source implementation of the Python programming language which is tightly integrated with .NET. **IronPython** can use .NET and Python libraries, and other .NET languages can use Python code just as easily.
https://ironpython.net/

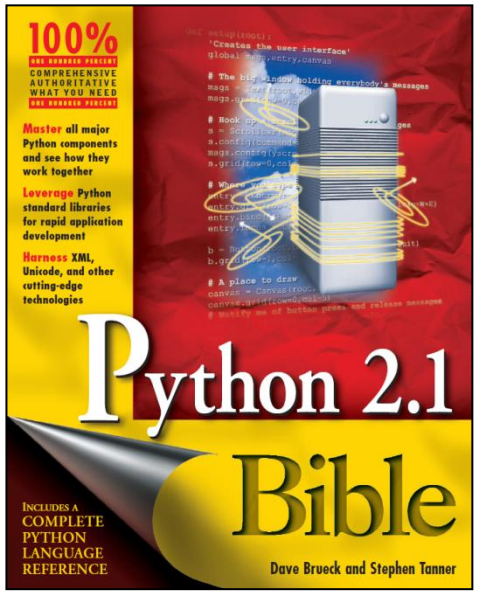
Portable Python - **Portable** **Python** is a minimalistic Python distribution for Microsoft Windows that does not require elevated privileges during installation. One can simply unpack distribution into any folder (local, external, network) and start programming in Python.
https://portablepython.com/download/

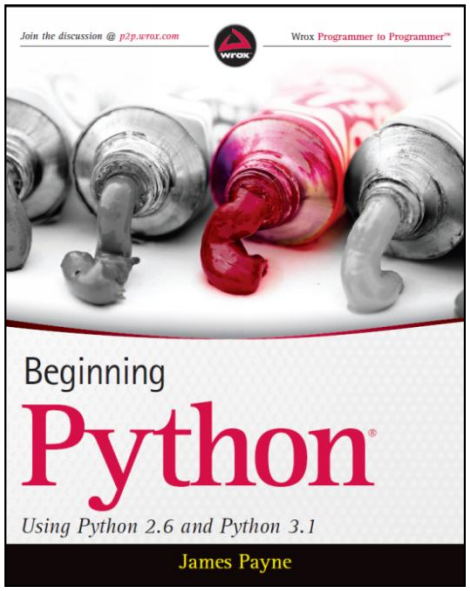
onecompiler – On line Python compiler. https://onecompiler.com/python

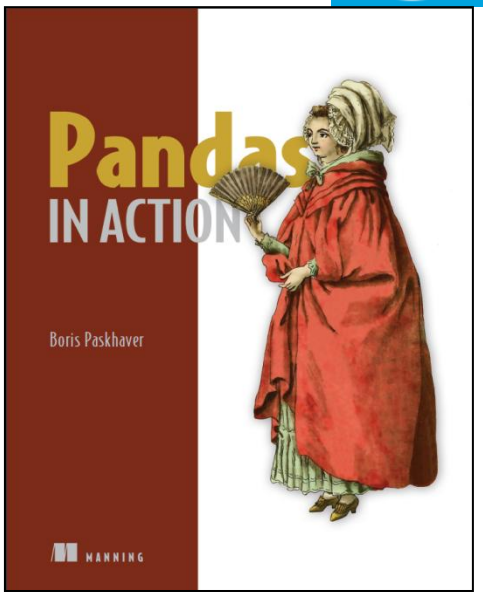
Programiz - On line Python compiler.
https://www.programiz.com/python-programming/online-compiler/

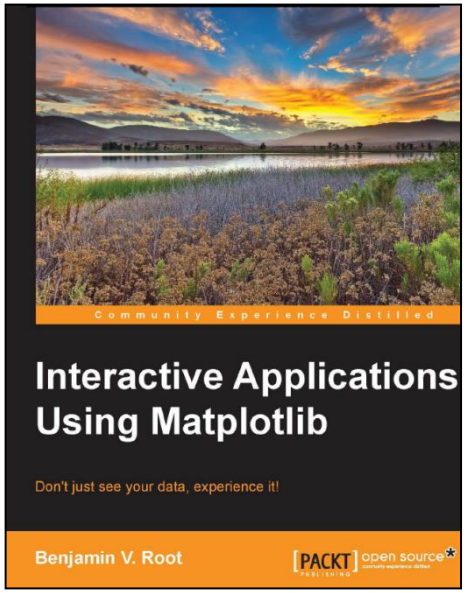
Books

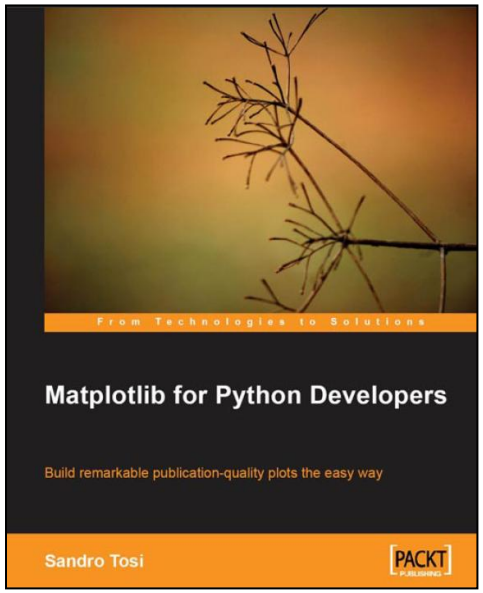

wxPython
IN ACTION
Noel Rappin
Robin Dunn
MANNING

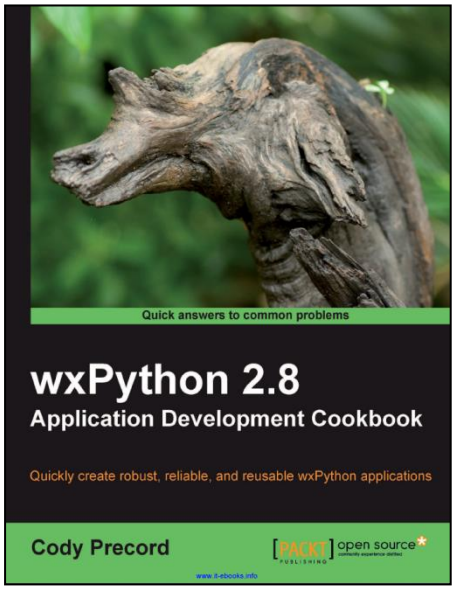

100%
COMPREHENSIVE
AUTHORITATIVE
WHAT YOU NEED
Master all major Python components and see how they work together
Leverage Python standard libraries for rapid application development
Harness XML, Unicode, and other cutting-edge technologies
Python 2.1
Bible
INCLUDES A COMPLETE PYTHON LANGUAGE REFERENCE
Dave Brueck and Stephen Tanner

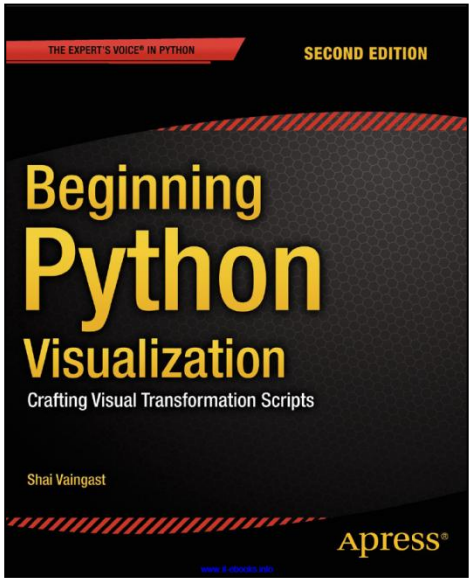

Join the discussion @ p2p.wrox.com
Wrox Programmer to Programmer
Beginning
Python
Using Python 2.6 and Python 3.1
James Payne


Pandas
IN ACTION
Boris Paskhaver
MANNING

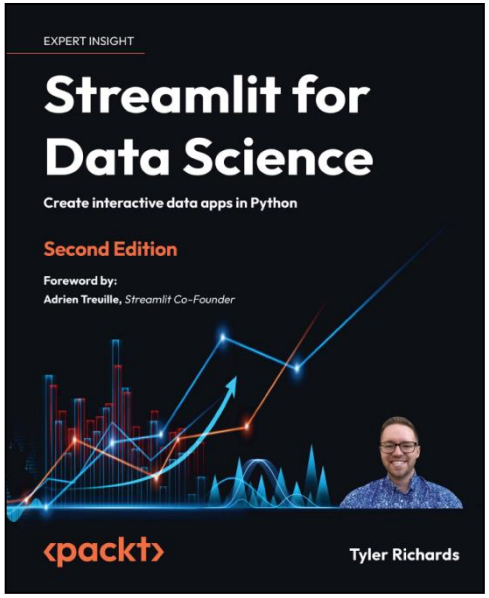

Community Experience Distilled
Interactive Applications Using Matplotlib
Don't just see your data, experience it!
Benjamin V. Root
PACKT open source

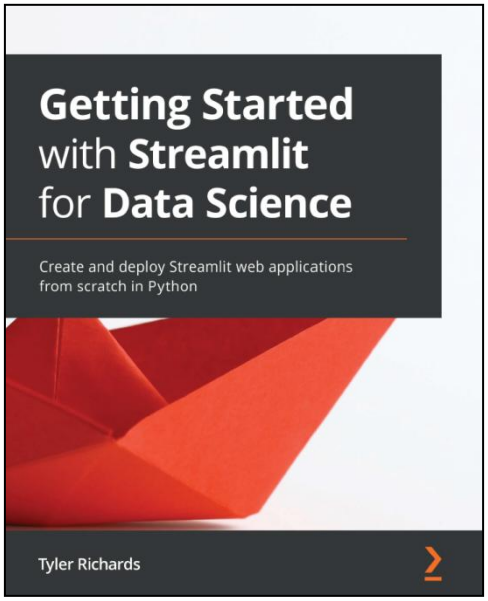

From Technologies to Solutions
Matplotlib for Python Developers
Build remarkable publication-quality plots the easy way
Sandro Tosi
PACKT

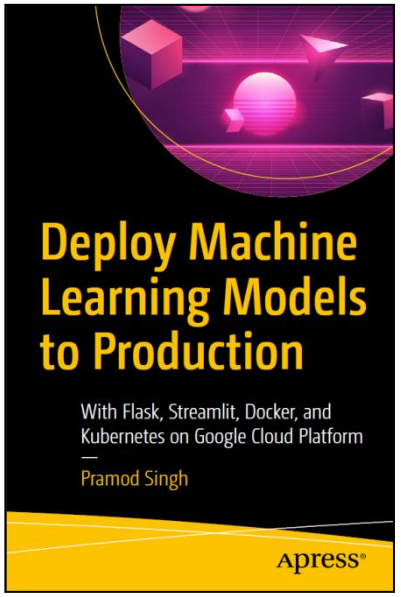

Quick answers to common problems
wxPython 2.8
Application Development Cookbook
Quickly create robust, reliable, and reusable wxPython applications
Cody Precord
PACKT open source

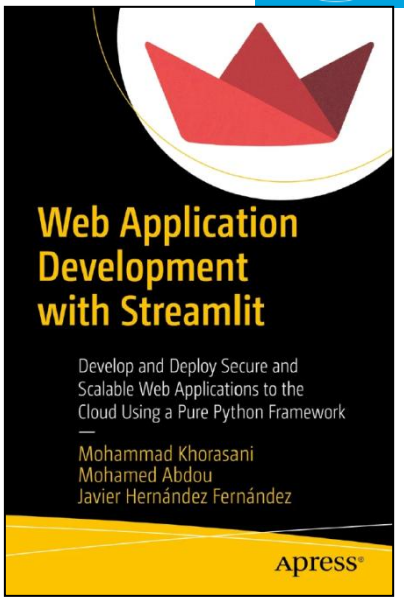

THE EXPERT'S VOICE® IN PYTHON
SECOND EDITION
Beginning
Python
Visualization
Crafting Visual Transformation Scripts
Shai Vaingast
Apress®

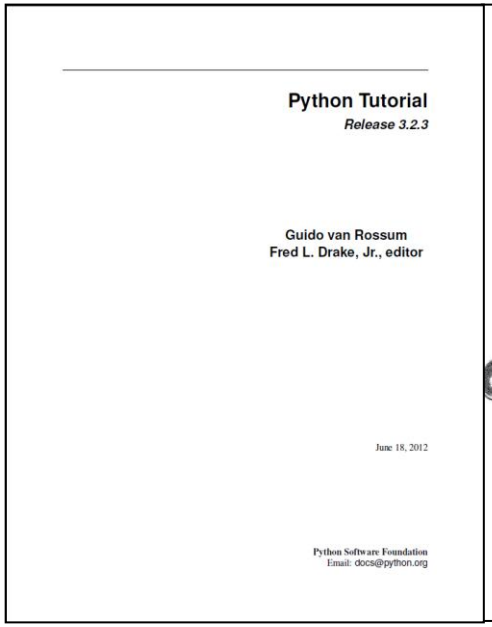
Books

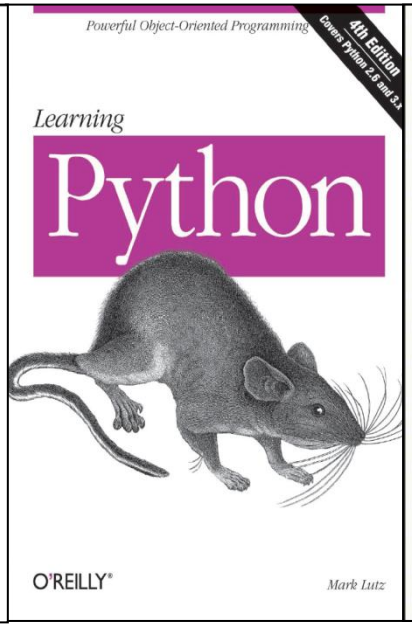

EXPERT INSIGHT
Streamlit for Data Science
Create interactive data apps in Python
Second Edition
Foreword by:
Adrien Treuille, Streamlit Co-Founder
<packt>
Tyler Richards

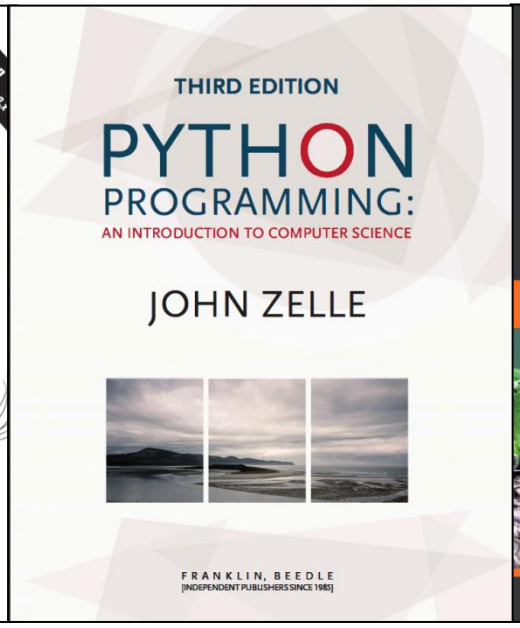

Getting Started with Streamlit for Data Science
Create and deploy Streamlit web applications from scratch in Python
Tyler Richards

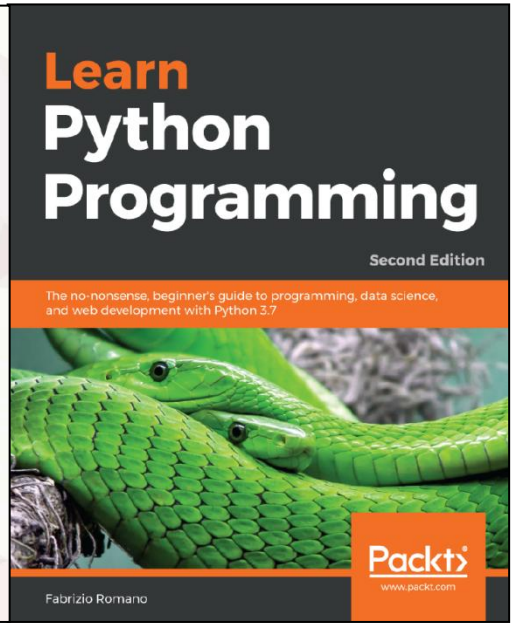

Deploy Machine Learning Models to Production
With Flask, Streamlit, Docker, and Kubernetes on Google Cloud Platform
Pramod Singh
Apress


Web Application Development with Streamlit
Develop and Deploy Secure and Scalable Web Applications to the Cloud Using a Pure Python Framework
Mohammad Khorasani
Mohamed Abdou
Javier Hernández Fernández
Apress


Python Tutorial
Release 3.2.3
Guido van Rossum
Fred L. Drake, Jr., editor
June 18, 2012
Python Software Foundation
Email: docs@python.org


Powerful Object-Oriented Programming
4th Edition
Covers Python 2.5 and 3.x
Learning
Python
O'REILLY
Mark Lutz

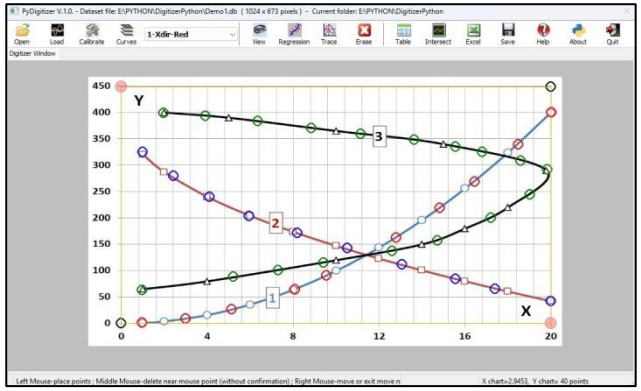

THIRD EDITION
PYTHON PROGRAMMING:
AN INTRODUCTION TO COMPUTER SCIENCE
JOHN ZELLE
FRANKLIN, BEEDLE
[INDEPENDENT PUBLISHERS SINCE 1985]


Learn Python Programming
Second Edition
The no-nonsense, beginner's guide to programming, data science, and web development with Python 3.7
Packt
www.packt.com
Fabrizio Romano

Tutorials & References

- https://www.w3schools.com/python/default.asp
- https://www.w3schools.com/python/numpy/default.asp
- https://www.w3schools.com/python/pandas/default.asp
- https://www.w3schools.com/python/scipy/index.php
- https://www.w3schools.com/python/matplotlib_intro.asp
- https://www.tutorialandexample.com/python-tutorial
- https://zetcode.com/lang/python/
- https://zetcode.com/python/sqlite/
- https://zetcode.com/python/python-pandas/
- https://zetcode.com/wxpython/
- https://www.tutorialspoint.com/python/index.htm

Python & wxPython Projects



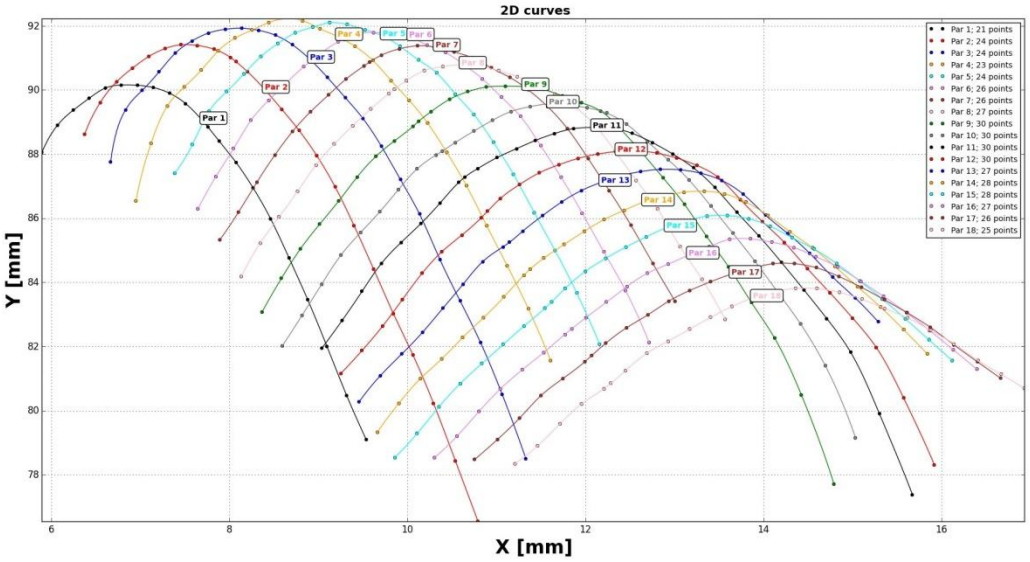
A Python module for digitizing 2D curves

The digitizing process consists of extracting a curve numerical coordinates from an explicit image. The paper describes the PyDigitizer module for digitizing 2D curves from images, created with the help of free and Open Source resources, using Python as a programming language and wx.Python as a graphical user interface toolkit.

Contributors: Dorian Nedelcu & Tihomir Latinovic

https://data.mendeley.com/datasets/jnwyr7jzrd/1

https://www.youtube.com/watch?v=WifxfTgQKcY



Contributors: Dorian Nedelcu & Tihomir Latinovic

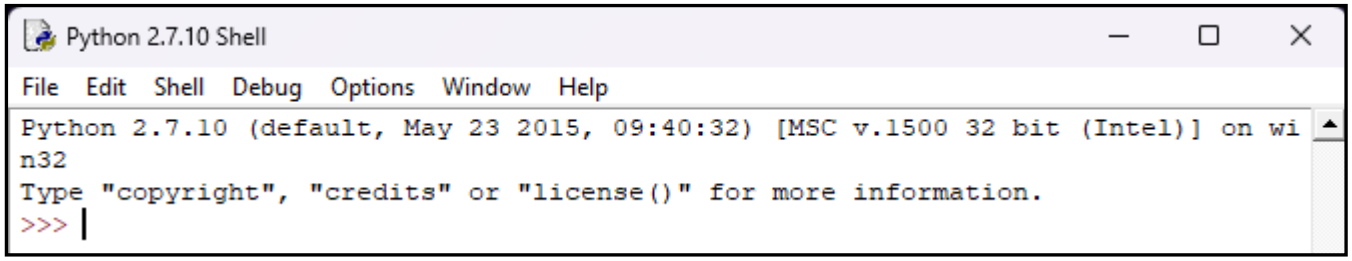
https://data.mendeley.com/datasets/35jrmwd4fw/1

PyChart

The PyChart module (aimed at analysis and visual view of 2D/3D Charts) was created with the help of free and Open Source resources, using Python as a programming language and wx.Python as a graphical user interface toolkit. The chart data is imported from a Excel/ CSV file with a template structure and is drawn in the PyChart module as XY or XYZ curves similar with Excel scatter with smooth lines and markers style. The module is provided with zooming instruments (fit, pan, zoom in, zoom out), cubic spline curves interpolation, chart intersection with constant X, Y or Z values, visual follow of the 2D chart points to view coordinates, export of data in Windows Clipboard, Excel or Microsoft Word format and saving the chart as a image file.

Execution Modes

To write and run (execute) a Python program, we need to have a Python interpreter installed on our computer or we can use any online Python interpreter. The interpreter is also called *Python shell*. A sample screen of Python interpreter is shown in the next figure. Here, the symbol **>>>** is called Python prompt, which indicates that the interpreter is ready to receive instructions. We can type commands or statements on this prompt for execution.



There are two ways to run a program using the Python interpreter:

(A) Interactive Mode

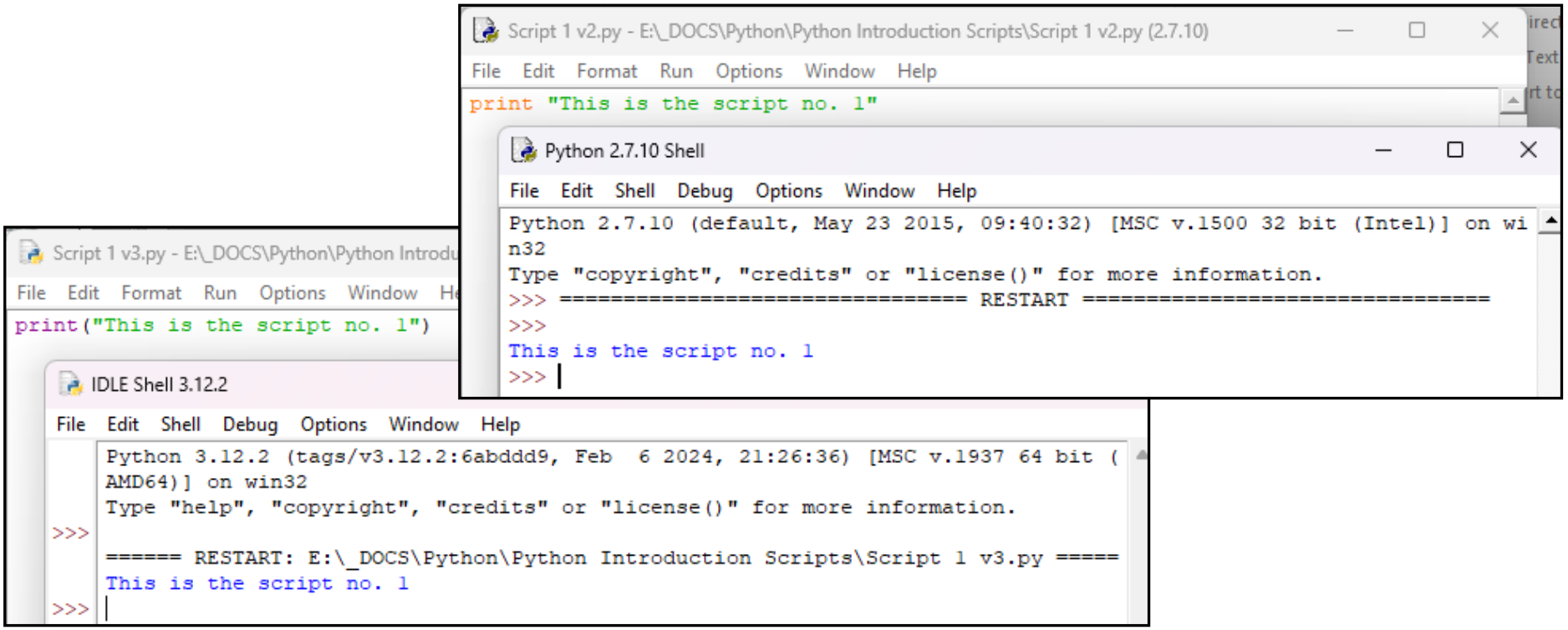
In the interactive mode, we can type a Python statement on the **>>>** prompt directly. As soon as we press enter, the interpreter executes the statement and displays the result(s). Working in the interactive mode is convenient for testing a single line code for instant execution. But in the interactive mode, we cannot save the statements for future use and we have to retype the statements to run them again.

https://ncert.nic.in/textbook/pdf/keip103.pdf

(B) Script Mode

In the script mode, we can write a Python program in a file, save it and then use the interpreter to execute the program from the file. Such program files have a **.py** or **.pyw** extension and they are also known as scripts. Usually, beginners learn Python in interactive mode, but for programs having more than a few lines, we should always save the code in files for future use. Python scripts can be created using any editor. Python has a *built-in editor* called IDLE which can be used to create programs. After opening the IDLE, we can click **File>New File** to create a new file, then write our program on that file and save it with a desired name. By default, the Python scripts are saved in the Python installation folder. To execute a Python program in script mode:

- Open the program using an editor, for example IDLE.
- In IDLE, go to [**Run**]->[**Run Module**] or **press F5** taste to execute the **xxxxxxx.py** script.



https://ncert.nic.in/textbook/pdf/keip103.pdf

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module, or other object. An identifier starts with a letter A to Z or a to z, or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9). Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python. Here are naming conventions for Python identifiers:

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Python Keywords

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

And	exec	Not
Assert	finally	or
Break	for	pass
Class	from	print
Continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced. The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example:

```
if True:
        print "True"
else:
        print "False"
```

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example:

```
total = item_one + \
item_two + \
item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example:

```
days = ['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string. The triple quotes are used to span the string across multiple lines. For example, all the following are legal:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

```
# First comment
print "Hello, Python!"; # second comment
```

You can type a comment on the same line after a statement or expression: **name = "Madisetti" # This is again comment**

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon:

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Assigning Values to Variables

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables. The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable. For example:

```
counter = 100 # An integer assignment
miles = 1000.0 # A floating point
name = "John" # A string
print counter
print miles
print name
```

Here, 100, 1000.0, and "John" are the values assigned to *counter*, *miles,* and *name* variables respectively. This produces the following result:

```
100
1000.0
John
```

Standard Data Types

The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters. Python has various standard data types that are used to define the operations possible on them and the storage method for each of them. Python has five standard data types:

Numbers String List Tuple Dictionary

Python supports four different numerical types:

int (signed integers) **long (long integers***, they can also be represented in octal and hexadecimal*)

float (floating point real values) **complex (***complex numbers*)

Python Strings

Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

```
str = 'Hello World!‘
print str + "TEST" # Prints concatenated string    => Hello World!TEST
```

Python Lists

Lists are the most versatile of Python's compound data types. A list contains items separated by commas and enclosed within square brackets ([]). To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type. The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

list = ['abcd', 786 , 2.23, 'john', 70.2] ; tinylist = [123, 'john']
print list *# Prints complete list* => **['abcd', 786, 2.23, 'john', 70.200000000000003]**
print list + tinylist *# Prints concatenated lists* => **['abcd', 786, 2.23, 'john', 70.200000000000003, 123, 'john']**

Python Tuples

A tuple is another sequence data type that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses. The main differences between lists and tuples are: Lists are enclosed in brackets (**[ ]**) and their elements and size can be changed, while tuples are enclosed in parentheses (**( )**) and cannot be updated. Tuples can be thought of as **readonly** lists.

tuple = ('abcd', 786 , 2.23, 'john', 70.2)

Python Dictionary

Python's dictionaries are kind of hash table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]). For example:

```
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept': 'sales'}
```

https://www.tutorialspoint.com/python/index.htm

Data Type Conversion

Sometimes, you may need to perform conversions between the built-in types. To convert between types, you simply use the type name as a function.

Types of Operators

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 30
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -10
* Multiplication	Multiplies values on either side of the operator	a * b = 200
/ Division	Divides left hand operand by right hand operand	b / a = 2
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 0
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 = 4 and 9.0//2.0 = 4.0

Function	Description
int(x [,base])	Converts x to an integer. base specifies the base if x is a string.
long(x [,base])	Converts x to a long integer. base specifies the base if x is a string.
float(x)	Converts x to a floating-point number.
complex(real [,imag])	Creates a complex number.
str(x)	Converts object x to a string representation.
repr(x)	Converts object x to an expression string.
eval(str)	Evaluates a string and returns an object.
tuple(s)	Converts s to a tuple.
list(s)	Converts s to a list.
set(s)	Converts s to a set.
dict(d)	Creates a dictionary. d must be a sequence of (key,value) tuples.
frozenset(s)	Converts s to a frozen set.
chr(x)	Converts an integer to a character.
unichr(x)	Converts an integer to a Unicode character.
ord(x)	Converts a single character to its integer value.
hex(x)	Converts an integer to a hexadecimal string.
oct(x)	Converts an integer to an octal string.

https://www.tutorialspoint.com/python/index.htm

Python Comparison Operators

These operators compare the values on either sides of them and decide the relation among them. They are also called Relational operators. Assume variable a holds 10 and variable b holds 20, then:

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Logical Operators

There are following logical operators supported by Python language. Assume variable **a** holds 10 and variable **b** holds 20 then:

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not (a and b) is false.

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below:

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

https://www.tutorialspoint.com/python/index.htm

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

Operator	Description
**	Exponentiation (raise to the power)
~ + -	Ccomplement, unary plus and minus (method names for the last two are +@ and -@)
* / % //	Multiply, divide, modulo and floor division
+ -	Addition and subtraction
>> <<	Right and left bitwise shift
&	Bitwise 'AND'
^ |	Bitwise exclusive `OR' and regular `OR'
<= < > >=	Comparison operators
<> == !=	Equality operators
= %= /= //= -= += *= **=	Assignment operators
is is not	Identity operators
in not in	Membership operators
not or and	Logical operators

Operator precedence affects how an expression is evaluated. For example:

x = 7 + 3 * 2; here, x is assigned **13**, not **20**

because operator * has higher precedence than +, so it first multiplies 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

```
a = 20 ; b = 10 ; c = 15 ; d = 5 ; e = 0
e = (a + b) * c / d #( 30 * 15 ) / 5
print "Value of (a + b) * c / d is ", e
e = ((a + b) * c) / d # (30 * 15 ) / 5
print "Value of ((a + b) * c) / d is ", e
e = (a + b) * (c / d); # (30) * (15/5)
print "Value of (a + b) * (c / d) is ", e
e = a + (b * c) / d; # 20 + (150/5)
print "Value of a + (b * c) / d is ", e
```

When you execute the above program, it produces the following result:

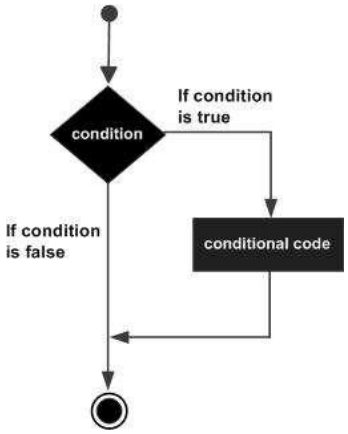
```
Value of (a + b) * c / d is 90
Value of ((a + b) * c) / d is 90
Value of (a + b) * (c / d) is 90
Value of a + (b * c) / d is 50
```

Python – Decision Making

If *Statement*

Syntax

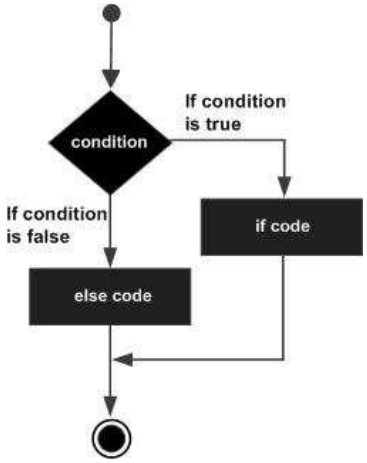
```
if expression:
    statement(s)
```



If...else *Statement*

Syntax

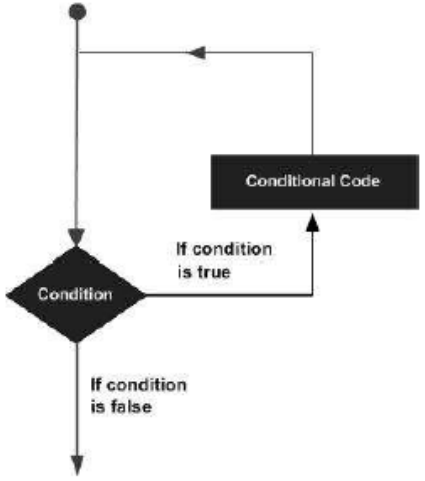
```
if expression:
    statement(s)
else:
    statement(s)
```



The *elif* Statement

Syntax

```
if expression1:
    statement(s)
elif expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
```



Python – Loops

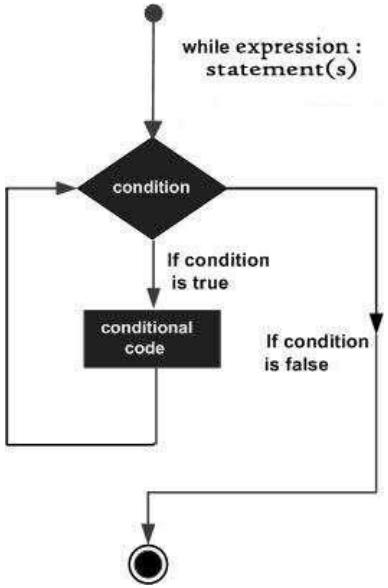
Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
for loop	Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
nested loops	You can use one or more loop inside any another while, for or do..while loop.

https://www.tutorialspoint.com/python/index.htm

While Loop

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true. When the condition becomes false, program control passes to the line immediately following the loop. In Python, all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code. Python uses indentation as its method of grouping statements. The syntax of a **while** loop in Python programming language is:

while expression:
statement(s)



The Infinite Loop

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop. An infinite loop might be useful in client/server programming where the server needs to run continuously so that client programs can communicate with it as and when required.

Using else Statement with Loops

Python supports to have an **else** statement associated with a loop statement.

- If the **else** statement is used with a **for** loop, the **else** statement is executed when the loop has exhausted iterating the list.
- If the **else** statement is used with a **while** loop, the **else** statement is executed when the condition becomes false.

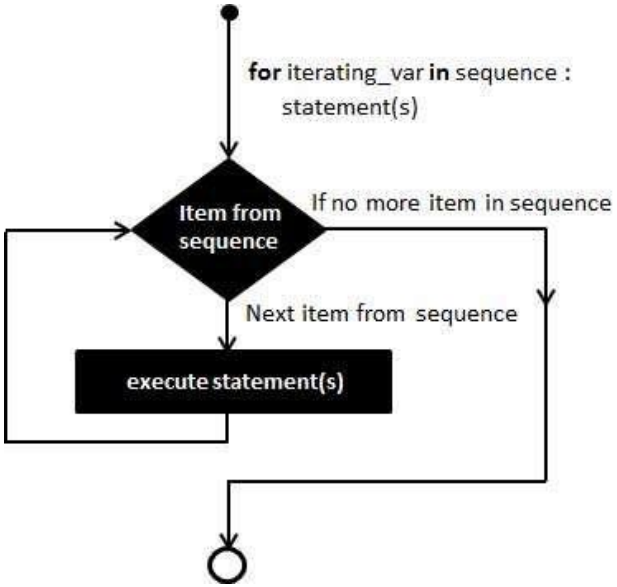
```
count = 0
while count < 5:
    print count, " is less than 5"
    count = count + 1
else:
    print count, " is not less than 5"
```

For Loop

It has the ability to iterate over the items of any sequence, such as a list or a string. Syntax:

for iterating_var in sequence:
statements(s)

If a sequence contains an expression list, it is evaluated first. Then, the first item in the sequence is assigned to the iterating variable *iterating_var*. Next, the statements block is executed. Each item in the list is assigned to *iterating_var*, and the statement(s) block is executed until the entire sequence is exhausted.



Nested Loops

Python programming language allows to use one loop inside another loop. Following section shows few examples to illustrate the concept. Syntax:

for iterating_var in sequence:
for iterating_var in sequence:
statements(s)
statements(s)

The syntax for a **nested while loop** statement in Python programming language is as follows:

while expression:
while expression:
statement(s)
statement(s)

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

https://www.tutorialspoint.com/python/index.htm

Loop Control Statements

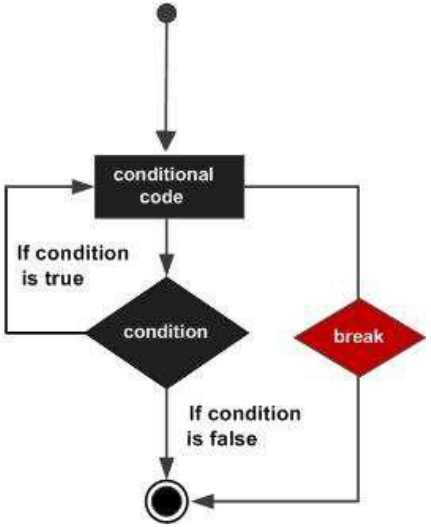
Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed. Python supports the following control statements.

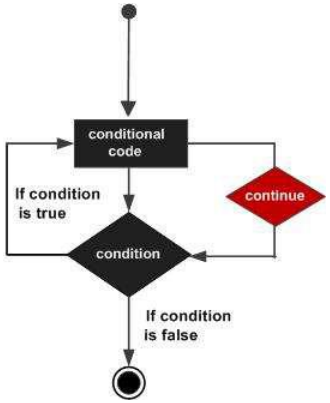
Control Statement	Description
break statement	Terminates the loop statement and transfers execution to the statement immediately following the loop.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
pass statement	The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Break Statement

It terminates the current loop and resumes execution at the next statement, just like the traditional break statement in C. The most common use for break is when some external condition is triggered requiring a hasty exit from a loop. The **break** statement can be used in both *while* and *for* loops. If you are using nested loops, the break statement stops the execution of the innermost loop and start executing the next line of code after the block. The syntax for a **break** statement in Python is as follows:

break





Continue Statement

It returns the control to the beginning of the while loop. The **continue** statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The **continue** statement can be used in both *while* and *for* loops. Syntax:

continue

Pass Statement

It is used when a statement is required syntactically but you do not want any command or code to execute. The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example): Syntax:

pass

https://www.tutorialspoint.com/python/index.htm

Mathematical Functions

Function	Returns (description)
abs(x)	The absolute value of x: the (positive) distance between x and zero.
ceil(x)	The ceiling of x: the smallest integer not less than x
cmp(x, y)	-1 if x < y, 0 if x == y, or 1 if x > y
exp(x)	The exponential of x: e^x
fabs(x)	The absolute value of x.
floor(x)	The floor of x: the largest integer not greater than x
log(x)	The natural logarithm of x, for x> 0
log10(x)	The base-10 logarithm of x for x> 0.
max(x1, x2,...)	The largest of its arguments: the value closest to positive infinity
min(x1, x2,...)	The smallest of its arguments: the value closest to negative infinity
modf(x)	The fractional and integer parts of x in a two-item tuple. Both parts have the same sign as x. The integer part is returned as a float.
pow(x, y)	The value of x**y.
round(x [,n])	x rounded to n digits from the decimal point. Python rounds away from zero as a tie-breaker: round(0.5) is 1.0 and round(-0.5) is -1.0.
sqrt(x)	The square root of x for x > 0

Trigonometric Functions

Function	Description
acos(x)	Return the arc cosine of x, in radians.
asin(x)	Return the arc sine of x, in radians.
atan(x)	Return the arc tangent of x, in radians.
atan2(y, x)	Return atan(y / x), in radians.
cos(x)	Return the cosine of x radians.
hypot(x, y)	Return the Euclidean norm, sqrt(x*x + y*y).
sin(x)	Return the sine of x radians.
tan(x)	Return the tangent of x radians.
degrees(x)	Converts angle x from radians to degrees.
radians(x)	Converts angle x from degrees to radians.

Mathematical Constants

Constants	Description
pi	The mathematical constant pi.
e	The mathematical constant e.

https://www.tutorialspoint.com/python/index.htm

Python – User Defined Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Python gives you many built-in functions such as print() and but you can also create your own functions. These functions are called *user-defined functions.* By default, parameters have a positional behavior and you need to inform them in the same order that they were defined.

Defining a Function

You can define functions to provide the required functionality. Here are rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses (( )).
- Any input parameters/arguments should be placed within these parentheses.
- The first statement of a function can be an optional statement – the documentation string of the function or docstring.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Syntax

```
def functionname( parameters ):
    "function_docstring"
    function_suite
    return [expression]
```

Passing by Reference Versus Passing by Value

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function. For example:

Calling a Function

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

```
# Function definition is here
def printme( str ):
    "This prints a passed string into this function"
    print str;
    return;
```

Now you can call **printme** function:

```
printme("I'm first call to user defined function!");
printme("Again second call to the same function");
```

When the above code is executed, it produces the following result:

```
I'm first call to user defined function!
Again second call to the same function
```

Python – Classes and Objects

Creating Classes

The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows:

```
class ClassName:
    'Optional class documentation string'
    class_suite
```

The *class_suite* consists of all the component statements defining class members, data attributes and functions.

Example:

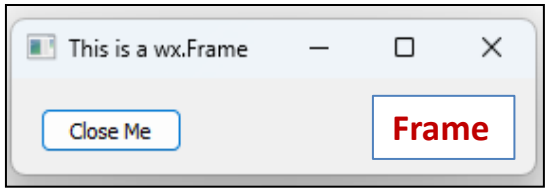
```
class Employee:
   'Common base class for all employees'
   empCount = 0
def __init__(self, name, salary):
   self.name = name
   self.salary = salary
   Employee.empCount += 1
def displayCount(self):
   print "Total Employee %d" % Employee.empCount
def displayEmployee(self):
   print "Name : ", self.name, ", Salary: ", self.salary
```

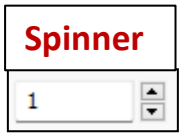
Creating Instance Objects & Accessing Attributes

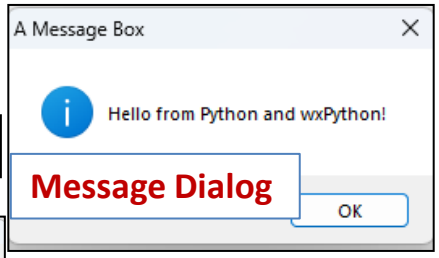
```
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

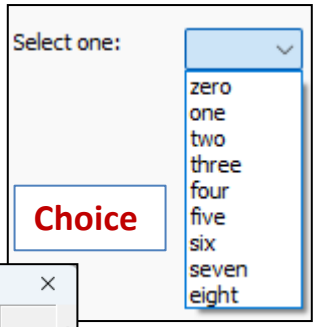
- The variable **empCount** is a class variable whose value is shared among all instances of a this class. This can be accessed as **Employee.empCount** from inside the class or outside the class.
- The first method **__init__()** is a special method, which is called class constructor or initialization method that Python calls when you create a new instance of this class.
- You declare other class methods like normal functions with the exception that the first argument to each method is **self**. Python adds the self argument to the list for you; you do not need to include it when you call the methods.

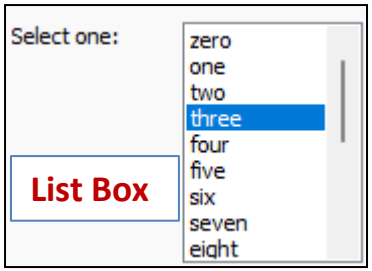
wxPython - easiest, most powerful ways of building a real graphical user interface (GUI) program.

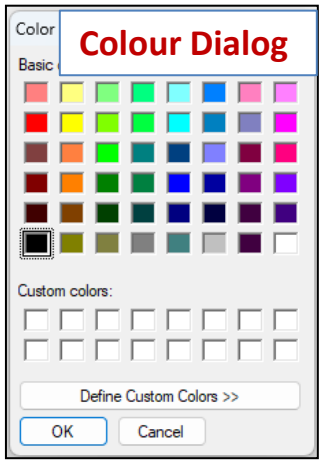


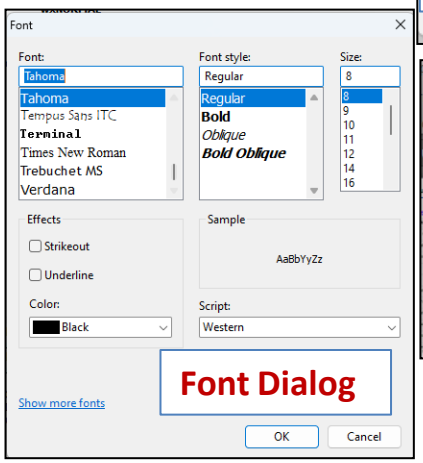


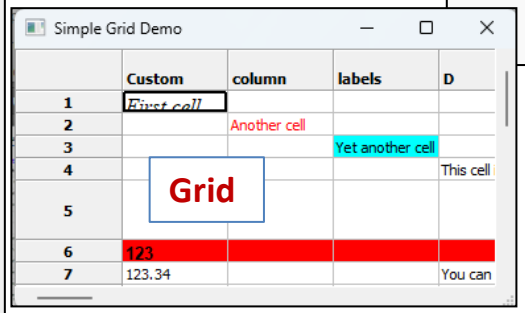


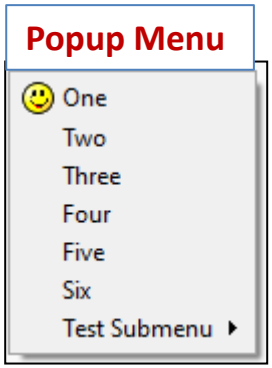


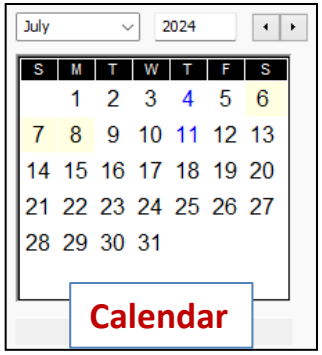


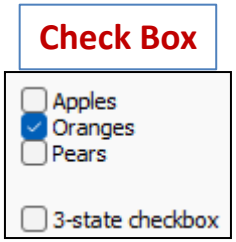


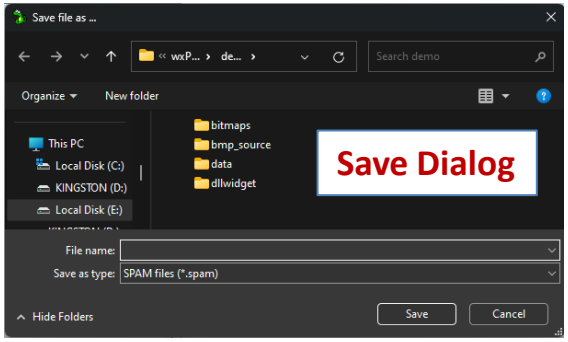


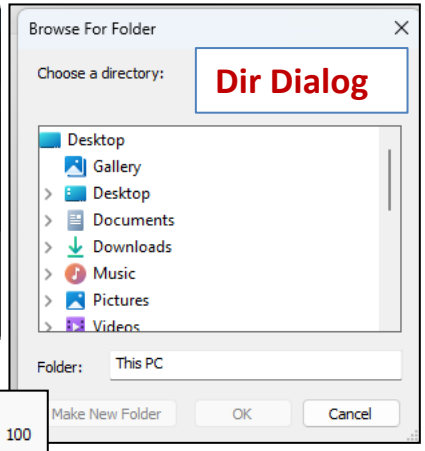


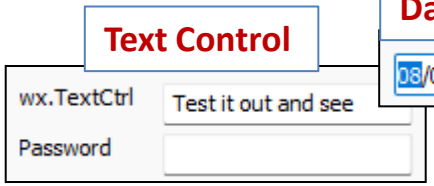


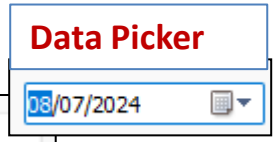


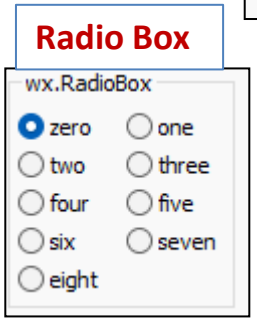


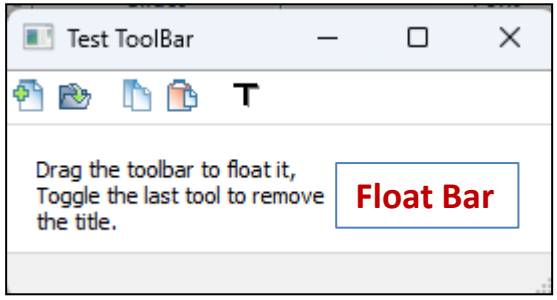












Streamlit - is a web application framework that helps you build and develop Python-based web applications that can be used to share analytics results, build complex interactive experiences and illustrate new machine learning models.

Tyler Richards - Getting Started with Streamlit for Data Science, Published by Packt Publishing Ltd., ISBN 978-1-80056-550-0, 2021

Display text

```
st.text('Fixed width text')
st.markdown('_Markdown_')
st.title('My title')
st.header('My header')
st.subheader('My sub')
```

Display media

```
st.image('./header.png')
st.audio(data)
st.video(data)
st.video(data, subtitles=
                    "./subs.vtt")
st.logo("logo.jpg")
```

Columns

```
col1, col2 = st.columns(2)
col1.write('Column 1')
col2.write('Column 2')
# Three columns with different widths
col1, col2, col3 = st.columns([3,1,1])
# col1 is wider
with col1:    # Using 'with' notation
    st.write('This is column 1')
```

Display charts

```
st.area_chart(df)
st.bar_chart(df)
st.bar_chart(df, horizontal=True)
st.line_chart(df)
st.map(df)
st.scatter_chart(df)

st.altair_chart(chart)
st.bokeh_chart(fig)
st.graphviz_chart(fig)
st.plotly_chart(fig)
st.pydeck_chart(chart)
st.pyplot(fig)
st.vega_lite_chart(df, spec)
```

Display interactive widgets

```
st.button('Hit me')
st.data_editor('Edit data', data)
st.checkbox('Check me out')
st.radio('Pick one:', ['nose','ear'])
st.selectbox('Select', [1,2,3])
st.multiselect('Multiselect', [1,2,3])
st.slider('Slide me', min_value=0, max_value=10)
st.select_slider('Slide to select', options=[1,'2'])
st.text_input('Enter some text')
st.number_input('Enter a number')
st.text_area('Area for textual entry')
st.date_input('Date input')
st.time_input('Time entry')
st.file_uploader('File uploader')
st.download_button('On the dl', data)
st.camera_input("Take a picture")
st.color_picker('Pick a color')
```

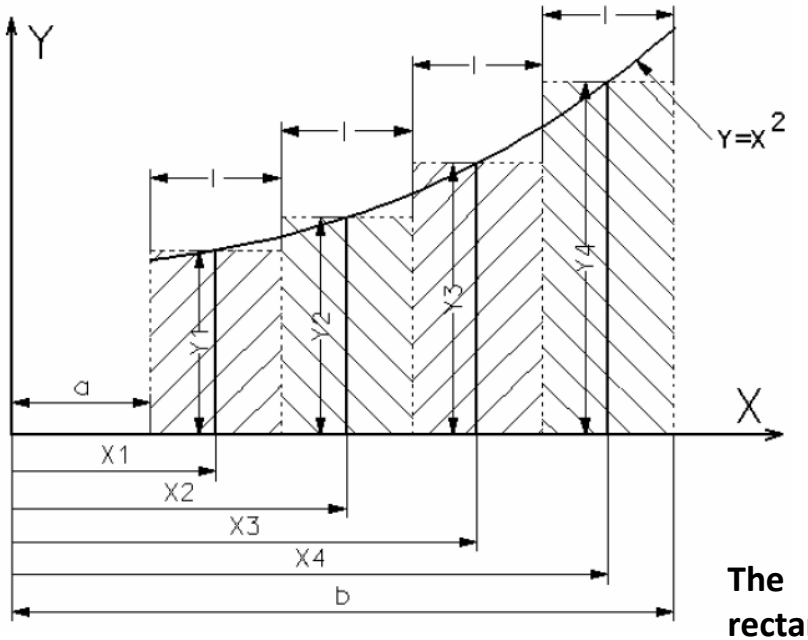
Group multiple widgets:

```
with st.form(key="my_form"):
    username = st.text_input("Username")
    password = st.text_input("Password")
    st.form_submit_button("Login")
```

Display status

```
st.warning("Warning message")
st.info("Info message")
st.success("Success message")
st.exception(e)
```

Example
Integration by rectangles method



Consider the parabolic function: $Y=X^2$. The area under the curve $Y=X^2$, 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.

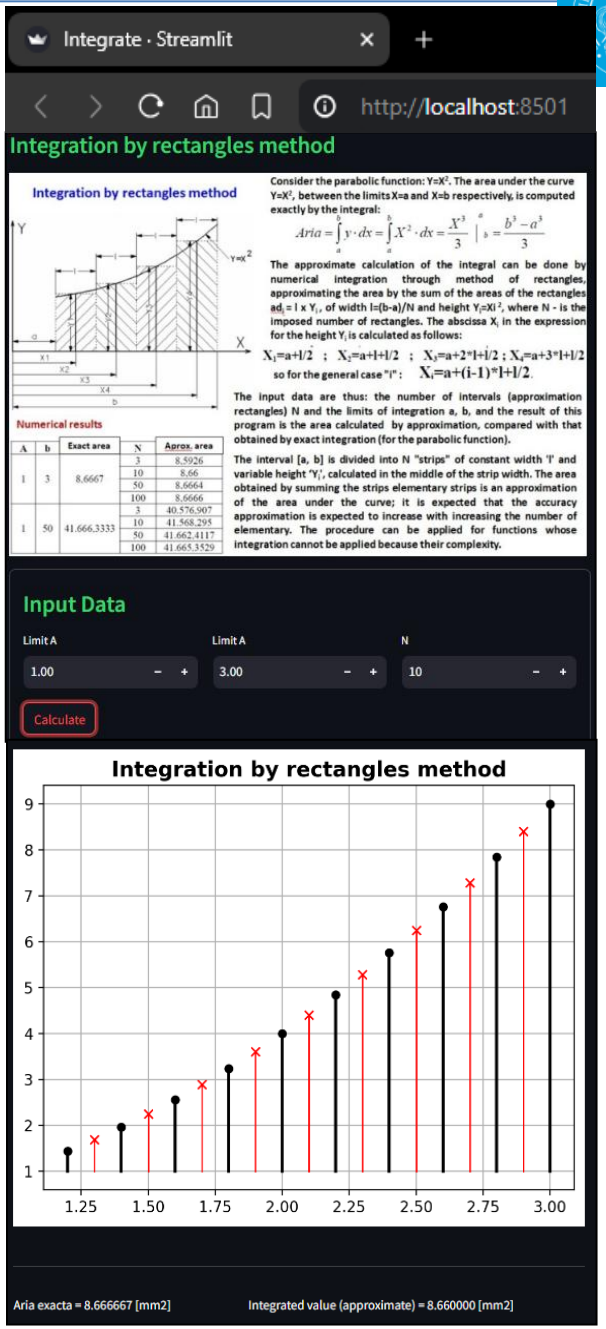
Numerical results

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

Dorian NEDELCU, COMPUTER PROGRAMMING IN TURBOPASCAL, Publishing: Eurostampa, Timişoara, ISBN: 973-687-013-8, 2002

```
import streamlit as st
from PIL import Image
import matplotlib.pyplot as plt
from pathlib import Path
current_dir = Path(__file__).parent if "__file__" in locals() else Path.cwd()
crt_dir_img= str(current_dir).rstrip("pages")+"/"
st.subheader(":green[Integration by rectangles method]")
st.empty().image(Image.open(str(current_dir)+"/"+"Integrate.jpg"), width=700, caption='')
with st.form('Input_Data'):
        st.subheader(':green[Input Data]')
        col1, col2, col3 = st.columns(3)
        a = col1.number_input('Limit A ', value=1.0)
        b = col2.number_input('Limit A ', value=3.0)
        N = col3.number_input('N ', value=10)
        submit_button = st.form_submit_button('Calculate')
if submit_button:
        l=(b-a)/N  ; aria=0  ;  X=[]  ;  Y=[]
        for i in range(1,N+1):
                xi = a+(i-1)*l+l/2  ;  X.append(xi)
                yi = xi * xi  ;  Y.append(yi)
                aria = aria + l * yi
  # Chart drawing
  Place_Chart = st.empty()
  fig=plt.figure()
  plt.grid(True)
  plt.title("Integration by rectangles method", fontsize=14, fontweight='bold',color='Black')
  for i in range(1, N):
          plt.plot([X[i], X[i]],[Y[i], Y[i]], 'x', color='Red', linewidth=2)
          plt.plot([X[i], X[i]],[a*a, X[i]*X[i]], '-', color='Red',linewidth=1)
          x1=X[i]-l/2  ; y1= x1 * x1
          plt.plot(x1,y1, '.', color="Black", markersize=10)
          plt.plot((x1,x1),(a*a,x1*x1), '-', color="Black",linestyle="solid",linewidth=2.0)
  plt.plot(b,b*b, '.', color="Black", markersize=10)
  plt.plot((b,b),(a*a,b*b), '-', color="Black",linestyle="solid",linewidth=2.0)
  Place_Chart.write(fig)
  # Results
  st.divider()
  col1, col2 = st.columns([0.4,0.6])
  Exact_val = (b**3-a**3)/3
  col1.write("Aria exacta = "+'%0.6f' % Exact_val+" [mm2]")
  col2.write("Integrated value (approximate) = "+'%0.6f' % aria +" [mm2]")
```

Integration by rectangles method – Streamlit 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

# The tools that we will use are imported from the Matplotlib library
import matplotlib as mpl
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
```

Integration by rectangles method – Python 2.7 & wxPython version 1/2

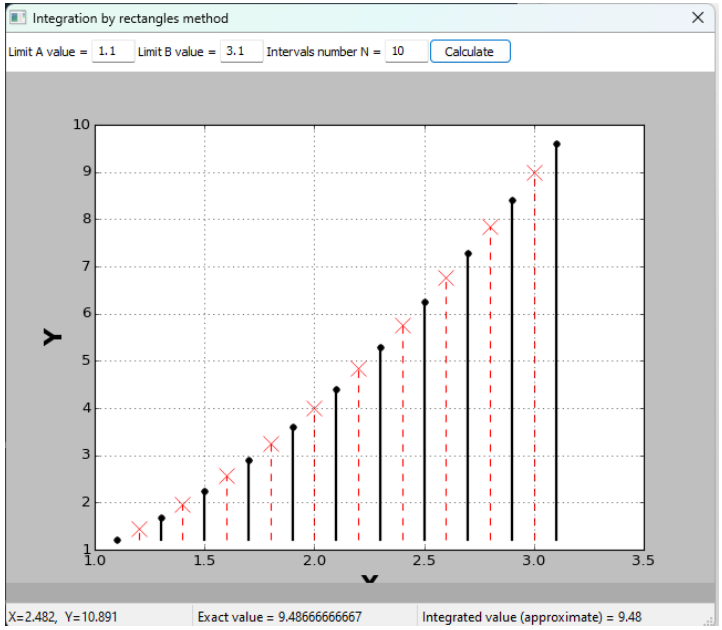
```
class Plot_Window(wx.Frame):
  def __init__(self, parent, title):
      wx.Frame.__init__(self, parent, title=title, size=(650,550),
          style=wx.DEFAULT_FRAME_STYLE ^ (wx.RESIZE_BORDER | wx.MINIMIZE_BOX |
              wx.MAXIMIZE_BOX )) # Initialize the frame
    self.panel = wx.Panel(self)  # 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 StatusBa
    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))
    st1 = wx.StaticText(self.toolbar, -1, " Limit A value = ")
    self.txt_Lim_A = wx.TextCtrl(self.toolbar, value="1.1", size=(40,-1))
    st2 = wx.StaticText(self.toolbar, -1, " Limit B value = ")
    self.txt_Lim_B = wx.TextCtrl(self.toolbar, value="3.1", 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(st1)
    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 MouseMotion(self, event):
    x,y = event.xdata, event.ydata
    sir=""
    a1,b1=self.axes.transData.inverted().transform([event.x, event.y])
    sir="X="+'%0.3f' % a1+",  Y="+'%0.3f' % b1
    self.statusbar.SetStatusText(sir,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()
        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)
              self.axes.plot((xplot,xplot),(funct(a),funct(xplot)), '-', color="Red",linestyle="dashed",linewidth=1.0)
              x1=xplot-l/2  ; y1= funct(x1)
              self.axes.plot(x1,y1, '.', color="Black", markersize=10)
              self.axes.plot((x1,x1),(funct(a),funct(x1)), '-', color="Black",linestyle="solid",linewidth=2.0)
        self.axes.plot(b,funct(b), '.', color="Black", markersize=10)
        self.axes.plot((b,b),(funct(a),funct(b)), '-', color="Black",linestyle="solid",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 our 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()
```

Integration by rectangles method – Python 2.7 & wxPython Version 2/2



Internet presentation

https://www.facebook.com/watch/?mibextid=qi2Omg&v=478849444746531&rdid=HCsDGsFFIDTg3wCo