



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
CAMPUS DI FORLÌ

Introductory Guide to Python

Master's Degree in Mechanical Engineering for Sustainability

didatticaforli.ingstudenti@unibo.it

Table of Contents

1. Introduction	3
2. Installation Guide.....	4
2.1. Anaconda Distribution.....	4
2.2. Python and pip (Advanced Users)	6
2.3. Introduction to Python Environments	6
2.4. Python versions	7
2.5. Managing Python Environments	7
2.6. Setting up an IDE	8
2.6.1. JupyterLab	9
2.6.2. Visual Studio Code	11
2.6.3. PyCharm	11
3. Coding in Python.....	11
3.1. Creating your first script.....	11
3.2. Help and Documentation	13
3.3. Modules.....	14
3.4. Functions, Classes, Objects	15
3.5. Python Scripts and Notebooks	15
3.6. Useful Modules	16
4. Code Examples	17
4.1. Coding Fundamentals	19
4.2. Data Importing	19
4.3. Data Processing and Applications	19
4.4. Data Export and Visualization	20
4.5. Good to Know	20
5. Where to go from here... ..	20
Additional Resources	20

1. Introduction

Python is a high-level, interpreted programming language that has garnered significant popularity in the software development community since its inception by Guido van Rossum in 1991. Python emphasizes code readability and simplicity. Van Rossum designed Python to be an easy-to-read language that encourages program modularity and code reuse. Its higher simplicity than other programming language makes it easy to start programming. Python is a versatile language used in various fields and for a wide range of applications, from data science and artificial intelligence to scientific computing and automation of procedures. Python's popularity has grown significantly over the years, becoming one of the most widely used programming languages in the world. Several factors contribute to its widespread adoption, like compatibility with different operating systems, clear and straightforward syntax, and extensive community and support, making it an easy-to-learn programming language with a rich ecosystem and a vast range of [applications](#).

This introductory guide aims to provide users the foundational knowledge needed to approach coding using python, guiding them from installing the required software to describing the fundamental syntax and required steps to setup a working Python installation. Finally, guidelines on how to continue learning are presented to render users autonomous in their learning journey. If you're unsure if Python is the right tool for you, you can check the [General Python FAQ](#)¹ webpage.

¹ <https://docs.python.org/3/faq/general.html>

2. Installation Guide

Installing Python is not straightforward, and the steps to do so can be different according to the user's preferences and familiarity with the language. Assuming advanced users need little to no guide in installing the software, this guide will focus on novice and intermediate users.

Downloading Python from the official website is not enough to make it ready to be used: when in use, Python is called from different environments that are created by the user, each with its own interpreter and libraries. These environments need to be setup and managed by users. While this can be done using the OS terminal, it is not suggested for novices. In this section we will see how to install a Python manager which allows users to control their Python installation using a GUI. Table 1 shows the suggested installation method for users according to their expertise with Python and/or the command prompt.

Novice Users	Intermediate Users	Advanced Users
Anaconda Distribution ²		Python Interpreter ³

Table 1: Suggested installation method for different users.

2.1. Anaconda Distribution

The suggested approach for novices is to install Anaconda. The Anaconda Distribution is a popular, open-source distribution of the Python and R programming languages for scientific computing, data science, and machine learning. It simplifies package management and deployment by providing a platform that includes a wide range of pre-installed packages and tools. This means that installing Anaconda gives the user access to tools to manage their Python installation and an integrated development environment (IDE) through a GUI. Managing Python with Anaconda is strongly suggested for new users and those who are not familiar with pip and terminal commands. The Anaconda individual distribution can be downloaded from the [Download Page](#). Anaconda is based on the conda package to install and manage Python environments, and it can be considered as the GUI to access conda functionalities in a user-friendly way. The Anaconda distribution is available for Windows, Mac, or Linux OS.

² <https://www.anaconda.com/download/success>

³ <https://www.python.org/downloads/>

Anaconda Installers



Figure 1: Anaconda Distribution Installers from the [download page](#).

To install the Anaconda Distribution, it is recommended to follow the on-screen instructions of the installer. If in doubt, follow the [Official Documentation](#)⁴. Once the installation is complete, Anaconda can be launched through the Anaconda Navigator executable. The Navigator allows users to have full control of their Python installation and virtual environments. The [Getting Started Documentation](#)⁵ on how to work with the Navigator is available online.

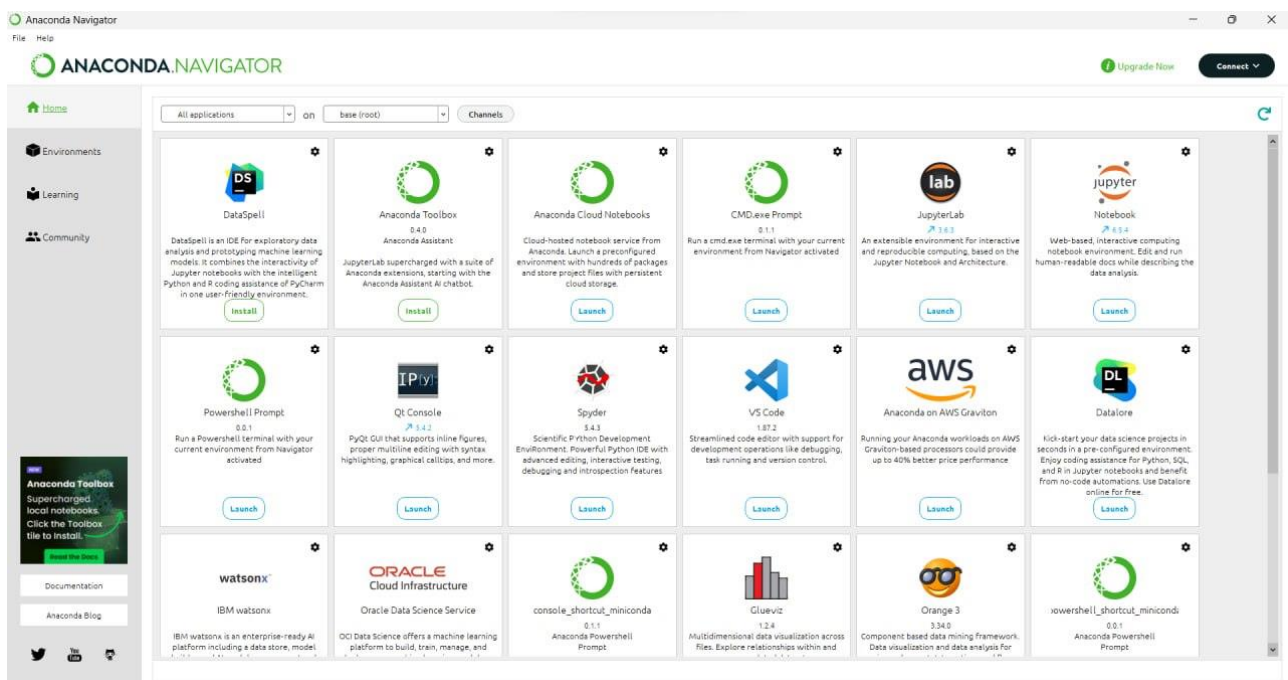


Figure 2: Anaconda Navigator User Interface.

⁴ <https://docs.anaconda.com/anaconda/install/>

⁵ <https://docs.anaconda.com/navigator/getting-started/>

2.2. Python and pip (Advanced Users)

This section is for users who want more flexibility and are confident in using the terminal. Instead of installing Anaconda, it is possible to simply download Python from the [official website](#)⁶. This installation is much quicker than downloading and installing Anaconda, but comes with no user interface. When installing Python this way, users must use the OS terminal to interact with it and manage their installation. In particular, managing the Python installation requires working with [pip](#)⁷ and terminal commands. As reference, if using Python and pip without anaconda, there's plenty of online tutorials on how to manage [virtual environments](#)⁸ and [packages](#)^{9 10}.

2.3. Introduction to Python Environments

While understanding Python Environments is not required for installing Python, knowing what they are is crucial to discern between pros and cons of different installation solutions. An environment consists in the combination of an interpreter that is called when running code, and the third-party libraries and packages installed in the environment. The virtual environments can be activated (only one at a time) and used when coding. This means that the environments are separated and can not communicate. Different versions of libraries can be installed in one or the other, ensuring legacy code can still work correctly while new projects can be based on the newest version of libraries.

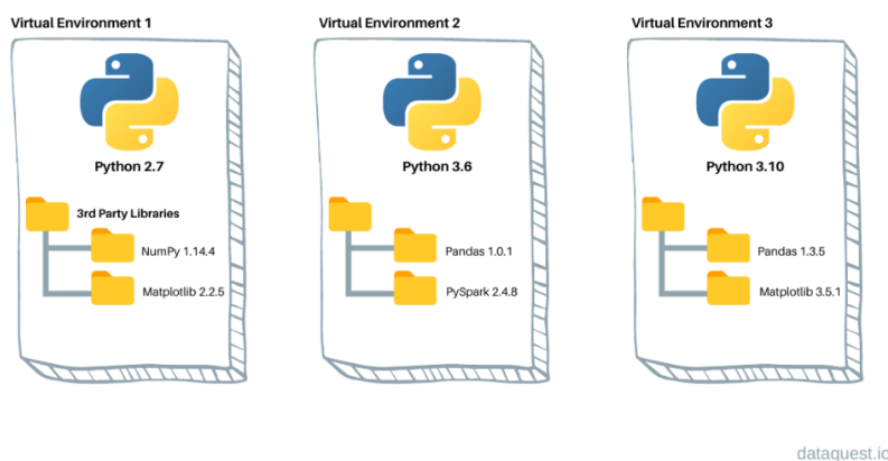


Figure 3: Virtual environments example, with each having a different Python version and different libraries¹¹.

⁶ <https://www.python.org/downloads/>

⁷ <https://packaging.python.org/en/latest/tutorials/installing-packages/>

⁸ <https://gist.github.com/ryanbehdad/858b47b54be441a684efb7ae6ca98a75>

⁹ <https://pip.pypa.io/en/stable/cli/>

¹⁰ <https://pgui.com/pip-cheat-sheet>

¹¹ <https://www.dataquest.io/blog/a-complete-guide-to-python-virtual-environments/>

2.4. Python versions

A quick note on Python versions: different Python versions (e.g. 2.6, 3.6, 3.12, etc...) might require a different version of libraries or packages. This can affect how scripts run and their stability. Therefore, it is important to always be aware of what Python version each environment is running on. Most importantly, there are coding syntax changes between Python 2 and Python 3. It is suggested, unless otherwise required, to run scripts on the latest stable version of Python 3. The Python version can be chosen when initializing an environment and can be different between environments.

Python release cycle

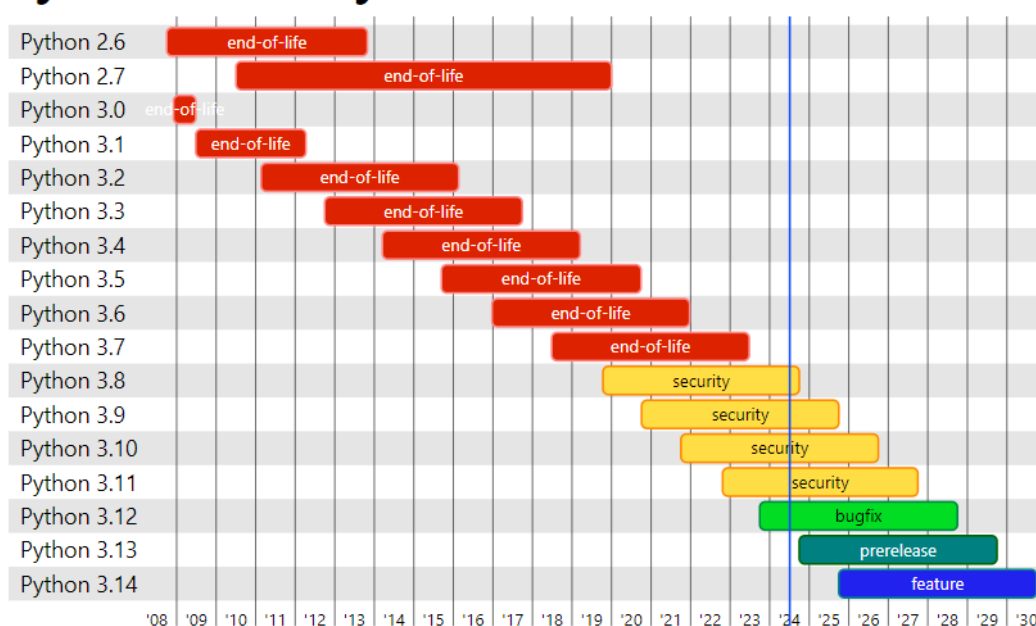


Figure 4: Python versions and release cycle summary.

2.5. Managing Python Environments

As previously introduced, Python environments are closed ecosystems of libraries. This implies that a script executed in Environment A will likely exhibit different behavior when run in Environment B, as environments are isolated and different libraries may be installed in each. Consequently, tracking and managing these environments is crucial to ensure the smooth operation of programs. Within the Navigator, users can access conda environments through the Environments page. The base environment serves as the default Python environment for every conda installation. It can be kept up-to-date and cloned to facilitate the rapid creation of new environments. Additionally, new environments can be created with the necessary Python version and specified target folder. A comprehensive guide on how to manage environments is

provided in the [Anaconda Documentation](#)¹². Advanced users can use commands in the terminal to manage their Anaconda installation ([Advanced environment management](#)¹³). A guide on how to manage packages is also available in the online [Documentation](#)¹⁴.

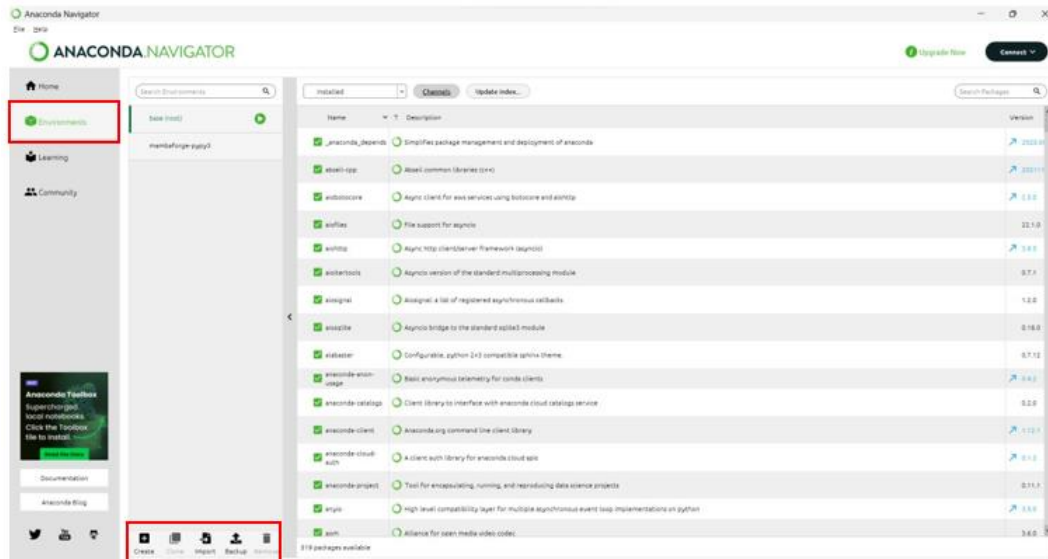


Figure 5: Environments tab of the Anaconda Navigator. Environment management commands are highlighted in red.

2.6. Setting up an IDE

Programming in Python is as simple as launching the Anaconda Prompt and typing “python.” However, this approach is sub-optimal, as advanced scripting benefits from visual aids and color differentiation to assist the user in the development process.

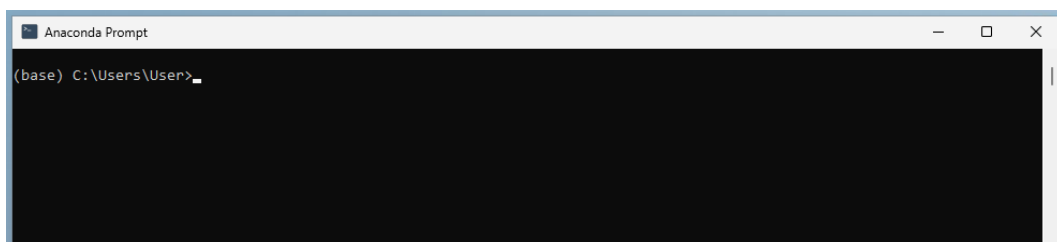


Figure 6: The simplest way to call Python is the Anaconda Prompt, a command-based interface that can be used to manage the Python installation and run Python commands.

¹² <https://docs.anaconda.com/navigator/tutorials/manage-environments/>

¹³ <https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>

¹⁴ <https://docs.anaconda.com/navigator/tutorials/manage-packages/>

Once the Python interpreter is installed, any text editor can function as an Integrated Development Environment (IDE), such as Sublime Text, Notepad++, etc., with an appropriate color palette. The drawback of this method is the necessity to copy and paste the code into the prompt to execute the scripts. The optimal solution is to utilize IDEs that can run scripts using kernels—processes that operate in the background to convey information to the Python interpreter. While these can be independently installed and linked to Python, Anaconda simplifies the installation process.

Since environments do not communicate with each other, an IDE must be installed within each environment where it will be used. Anaconda currently supports several IDEs, including VS Code, JupyterLab, Notebook, and Spyder. The Anaconda documentation offers tutorials on how to set up the various supported IDEs ([IDE Tutorials](#))¹⁵.

2.6.1. JupyterLab

For this guide, JupyterLab will be used due to its simplicity and resemblance to other coding environments, such as MATLAB. JupyterLab can be launched from the tiles in the Anaconda Navigator.

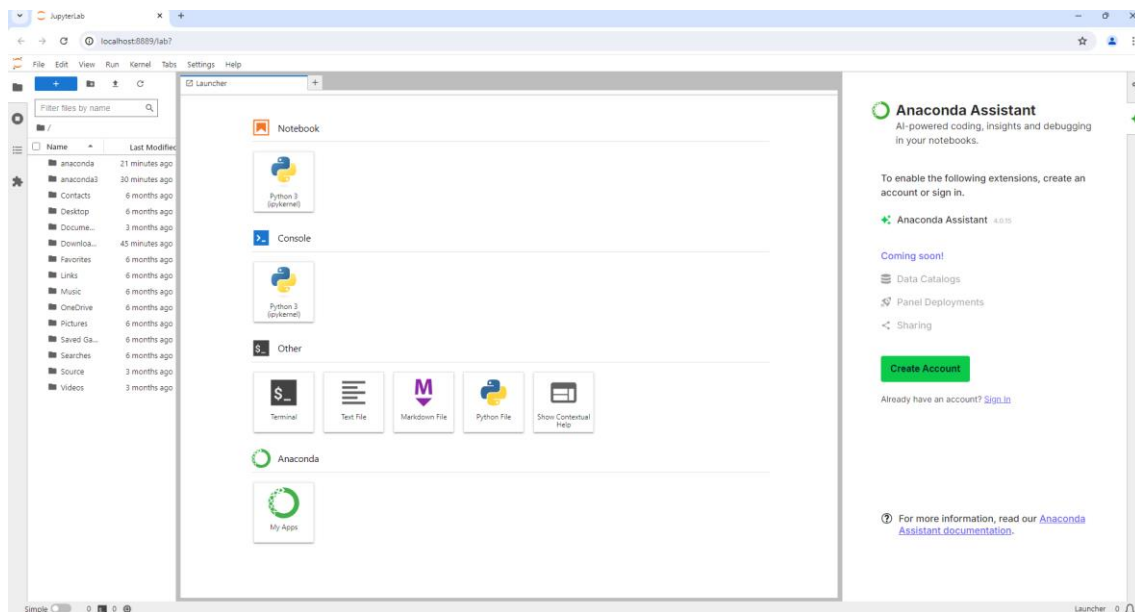


Figure 7: The JupyterLab interface. This IDE provides an intuitive yet effective browser-based solution for Python coding.

JupyterLab is a browser-based Integrated Development Environment (IDE), which operates through a web browser such as Google Chrome. Despite this, it provides a visually intuitive

¹⁵ <https://docs.anaconda.com/anaconda/getting-started/#ide-tutorials>

coding experience, featuring interactive cells and real-time output display. JupyterLab allows for the creation of Notebooks, which are files with the *.ipynb extension that contain interactive code cells. Within these files, results and outputs are displayed directly in the coding window, similar to MATLAB live scripts. Alternatively, simpler Python files can be created. These files maximize code simplicity and can be read by basic text editors, though they may sacrifice some clarity. Ultimately, the choice of format in which to code programs rests with the user.

JupyterLab is the perfect IDE for new users to create computational notebooks. A computational notebook is a shareable document that combines computer code, plain language descriptions, data, rich visualizations like 3D models, charts, graphs and figures, and interactive controls. A notebook, along with an editor like JupyterLab, provides a fast interactive environment for prototyping and explaining code, exploring and visualizing data, and sharing ideas with others¹⁶. JupyterLab can be installed and managed through the Anaconda package. A guide and overview of JupyterLab can be found on its official website ([Get Started](https://jupyterlab.readthedocs.io/en/latest/getting_started/overview.html)¹⁷, [User Guide](https://jupyterlab.readthedocs.io/en/stable/user/index.html)¹⁸).

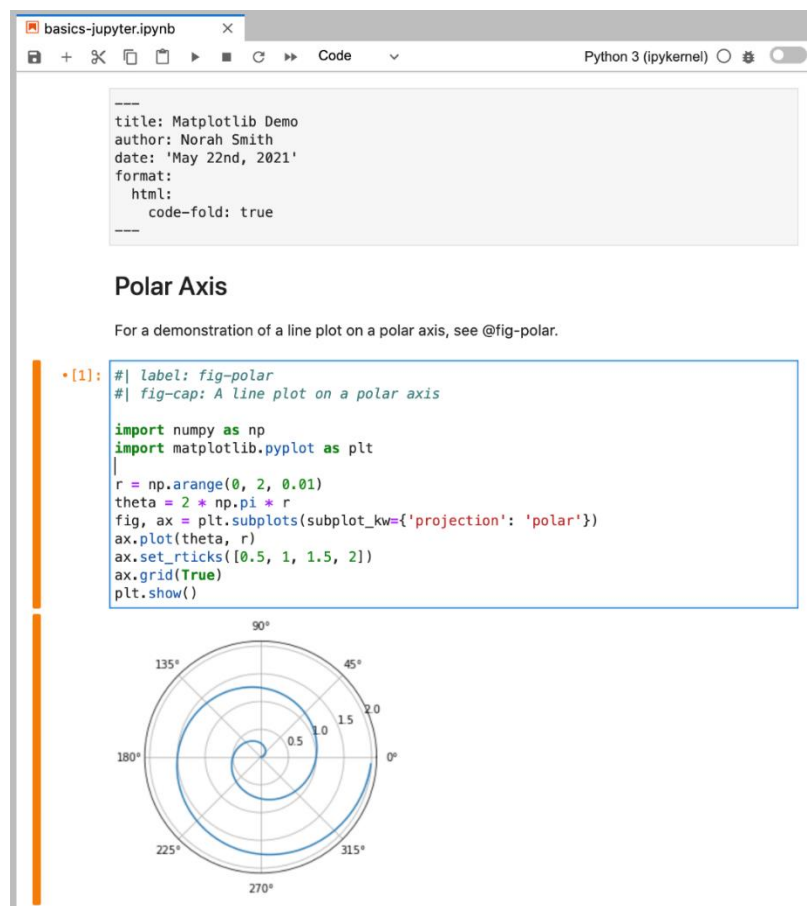


Figure 8: Jupyter Notebook example in JupyterLab.

¹⁶ <https://jupyterlab.readthedocs.io/en/latest/>

¹⁷ https://jupyterlab.readthedocs.io/en/latest/getting_started/overview.html

¹⁸ <https://jupyterlab.readthedocs.io/en/stable/user/index.html>

2.6.2. Visual Studio Code

VS Code is one of the most commonly used Integrated Development Environments (IDEs) in the industry. It offers a highly flexible programming experience due to its extensive library of extensions. However, its setup process can be less intuitive compared to JupyterLab. The Anaconda documentation provides a comprehensive tutorial on how to set up VS Code ([Visual Studio Code Tutorial](https://docs.anaconda.com/working-with-conda/ide-tutorials/vscode/)¹⁹). Additionally, there are official instructions on setting up [Python in VS Code](https://code.visualstudio.com/docs/languages/python)²⁰.

2.6.3. PyCharm

PyCharm is an IDE that integrates with Anaconda and supports managing virtual environments with conda. Similarly to VS Code, PyCharm provides a complete programming experience with tools and instruments to support coding. To setup PyCharm with the Anaconda Distribution, follow the [official documentation](https://docs.anaconda.com/working-with-conda/ide-tutorials/pycharm/)²¹.

3. Coding in Python

3.1. Creating your first script

Here is a quick recap on what should have been done up to this point:

- Install Anaconda Navigator.
- Create a new virtual environment.
- JupyterLab (or an equivalent IDE) has been installed on the new environment.

To start JupyterLab, select its tile in the Anaconda Navigator or type `jupyter-lab` in the OS terminal. The Launcher window should appear (Fig. 9) to select the kind of script to create. From here, we can create a notebook for an interactive programming experience.

¹⁹ <https://docs.anaconda.com/working-with-conda/ide-tutorials/vscode/>

²⁰ <https://code.visualstudio.com/docs/languages/python>

²¹ <https://docs.anaconda.com/working-with-conda/ide-tutorials/pycharm/>

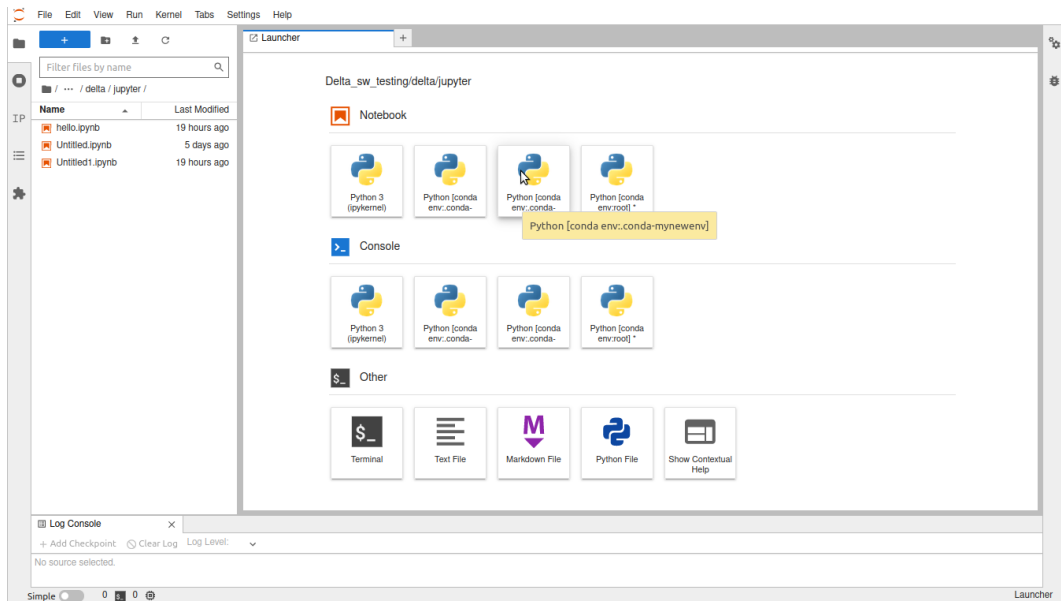


Figure 9: JupyterLab start screen.

One of the most important concepts of JupyterLab is the kernel, which is the process that runs Python and interacts with the interpreter. Changing (or restarting) the kernel inside of JupyterLab allows the user to change the environment while coding. JupyterLab offers documentation on [Managing Kernels and Terminals](#)²².

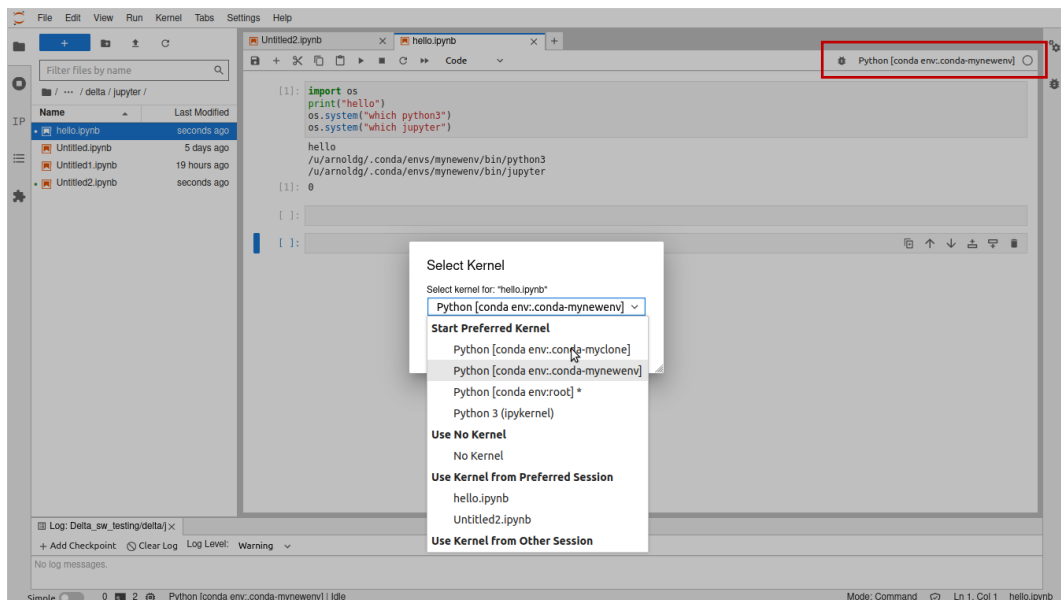


Figure 10: Kernel selections in JupyterLab.

²² <https://jupyterlab.readthedocs.io/en/stable/user/running.html>

The concept of working with Notebooks consists in creating and running cells, instead of the whole code every time. This means that by separating the importing of modules, functions, and data from the processing, can make working easier and without repeated instructions, as well as the code clearer.

Creating a script is straightforward. Choosing the Notebook option, a *.ipynb file is created and can be edited in the editor environment. A simple python script *.py can be created and edited in JupyterLab, but not run directly. These files can, on the other hand, be designed as modules to be called from an interactive script.

A new Notebook starts with an empty Code cell. This selection can be changed in the Notebook toolbar (Code/Markdown/Raw). Markdown and Raw cells contain text and are not compiled when running the script.

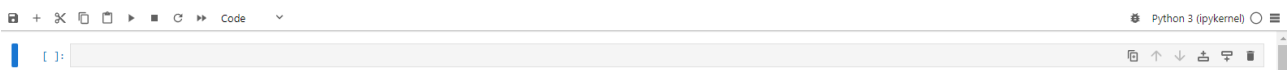


Figure 11: New Notebook interface. The empty cell is automatically set to Code and can be run with Shift+Enter.

We can try simple commands to check if everything is working correctly.

```
list = [1,2,3]

var1 = list[0]
var2 = list[2]

sum = var1 + var2

print(sum)
```

To run the code you can click on the play icon in the toolbar (Run this cell and advance) or use Shift+Enter. A nice cheatsheet for working with JupyterLab is available at this [page](https://www.edureka.co/blog/wp-content/uploads/2018/10/Jupyter_Notebook_CheatSheet_Edureka.pdf)²³. If the Notebook has printed “4”, everything is set up for more advanced coding.

3.2. Help and Documentation

The Python documentation is available online. This is valid for the basic Python modules and third-party modules. In addition, Python includes a built-in help system that you can access directly from the Python interpreter. To call the Help System, it is possible to use the command:

```
help()
```

²³ https://www.edureka.co/blog/wp-content/uploads/2018/10/Jupyter_Notebook_CheatSheet_Edureka.pdf

Help can also be called on a specific function or module:

```
help('modules') # Lists all available modules
help(math)      # Provides help on the math module
help(math.sqrt) # Provides help on the sqrt function in the math module
```

Or on keywords:

```
help('for')
help('if')
```

The `dir()` function can be used to list the attributes and methods of an object. This can be useful to see what functions are available in a module.

```
import math
dir(math)
```

Most modules, functions, classes, and methods also have a `__doc__` attribute that contains their documentation string:

```
print(math.__doc__)
print(math.sqrt.__doc__)
```

3.3. Modules

In Python, modules are essentially files that contain Python code. They can define functions, classes, and variables, as well as include runnable code. Modules allow you to logically organize your Python code into manageable sections, making it easier to maintain and reuse. Modules can be divided into:

- **Built-in Modules**: Python comes with a large standard library of modules that you can use without installing anything extra. Examples include `math`, `os`, `sys`, and `datetime`.
- **Third-Party Modules**: These are modules developed by the Python community. You can install them using package managers like `pip`. Examples include `requests`, `numpy`, `pandas`, and `matplotlib`.
- **Custom Modules**: These are modules you create yourself to organize your code. For instance, if you have a file named `mymodule.py`, it can be imported and used in other Python scripts.

Modules can be imported in different ways. The most common one is using aliases to simplify calling their content. In this example we are calling the `numpy` (numerical python) module with the alias `np`. This means that every time we want to access a class or function of `numpy`, we will have to use the dot notation to specify the module the function belongs to.

```
import numpy as np
print(np.array([1, 2, 3])) # Output: [1 2 3]
```

This way, there is no confusion what modules or functions are being called. Another approach is importing only specific functions from modules, like in the following example. This approach takes away the complexity of the dot notation.

```
from mymodule import myFunction
myFunction()
```

3.4. Functions, Classes, Objects

As the Python lexicon can be confused to new users. We will briefly introduce the most common keywords found when reading coding tutorials and books. First of all, functions are blocks of reusable code that perform specific tasks. These take input parameters and return values. Differently, classes are prototypes for creating objects. Classes encapsulate data (attributes) and behaviours (methods). Attributes are the variables that belong to a class, and methods are the functions that belong to a class. Finally, objects are instances of classes, they have with their own attributes and can use the methods defined by the class. Here is a trivial example of a class, and its instancing in an object.

```
class Dog:
    """A simple class representing a dog."""

    def __init__(self, name, age):
        """Initialize the attributes of the dog."""
        self.name = name
        self.age = age

    def bark(self):
        """A method that makes the dog bark."""
        print(f"{self.name} says Woof!")

# Creating an instance of the class
my_dog = Dog("Buddy", 3)

# Accessing attributes and methods
print(my_dog.name) # Output: Buddy
print(my_dog.age) # Output: 3
my_dog.bark() # Output: Buddy says Woof!
```

In this case, and in general, the `__init__` method is a special method called a constructor. It is automatically called when a new instance of the class is created. It initializes the object's attributes. And must always be defined when creating a class.

3.5. Python Scripts and Notebooks

In general, Python scripts are used to perform analyses where the output needs to be collected in the end. While Notebooks provide a more interactive programming experience, as well as a

simpler way of sharing results between users. Scripts and Notebooks are generally supported by all IDEs. Jupyter Notebooks can be tested in the web version of [JupyterLite](https://jupyter.org/try-jupyter/lab/)²⁴, and the documentation of [JupyterLab](https://jupyterlab.readthedocs.io/en/latest/)²⁵ is online.

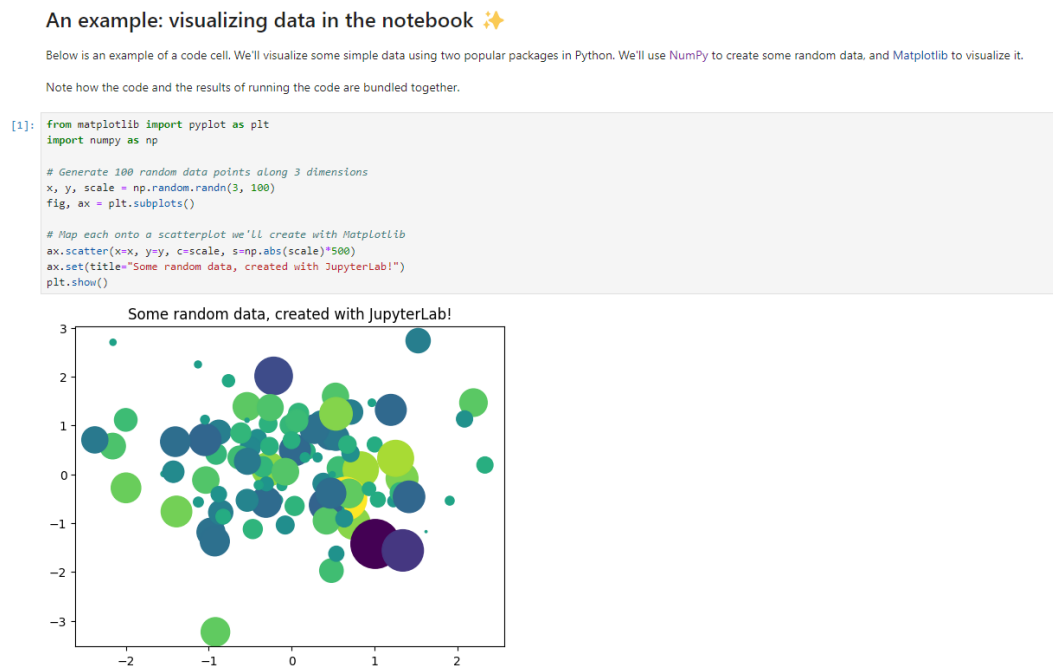


Figure 12: Example of interactive Notebook from the official JupyterLite website.

3.6. Useful Modules

Performing calculations in Python, including array and scientific calculations, involves using specialized libraries designed to handle complex mathematical operations efficiently. The following points outline the primary modules used for such calculations in Python:

1. **NumPy**: A fundamental library for numerical computations in Python, NumPy provides support for large, multi-dimensional arrays and matrices. It includes a wide range of mathematical functions to perform operations on these arrays.
2. **SciPy**: Built on top of NumPy, SciPy is a library used for scientific and technical computing. It includes modules for optimization, integration, interpolation, eigenvalue problems, algebraic equations, and other advanced mathematical operations.
3. **Pandas**: Primarily known for data manipulation and analysis, Pandas also supports a range of mathematical operations. It provides data structures like DataFrames, which facilitate complex calculations on structured data.
4. **SymPy**: A library for symbolic mathematics, SymPy allows for algebraic manipulations, calculus, equation solving, and other symbolic computations. It is useful for problems that require exact solutions rather than numerical approximations.

²⁴ <https://jupyter.org/try-jupyter/lab/>

²⁵ <https://jupyterlab.readthedocs.io/en/latest/>

5. **Matplotlib:** Although primarily a plotting library, Matplotlib also supports basic mathematical operations and can be used in conjunction with other libraries to visualize results of calculations.
6. **Scikit-learn:** A machine learning library built on NumPy, SciPy, and Matplotlib, Scikit-learn provides tools for data mining and data analysis, including a wide range of algorithms for classification, regression, clustering, and dimensionality reduction.
7. **TensorFlow and PyTorch:** Libraries for deep learning and complex neural network calculations. They provide tools for automatic differentiation and GPU acceleration, enabling efficient computation of large-scale models.

By leveraging these libraries, Python can handle a wide range of numerical and scientific calculations, making it a powerful tool for both simple and complex mathematical tasks.

4. Code Examples

The following links give access to a series of Python Jupyter Notebooks created to provide further context and examples on how to use Python. These can be opened using JupyterLab or any program able to read Jupyter Notebooks. For an easy access, the links are provided directly to the folder containing the Notebook. While other IDEs can open directly the files, IDEs based on Anaconda (which has been chosen for its easy maintainability) cannot be called directly, but need to be opened through Anaconda. This greatly facilitates the selection and management of environments, but also requires users to manually open the IDE from the Navigator. Here is a step-by-step guide on how to open the code examples using Anaconda. Users who have set up their own python and environments might be able to simply click on the Notebooks in Explorer to open them in their selected IDE.

If Anaconda/JupyterLab is installed (Default):

- 1) Click on the links below, this opens the folder where the Notebook is in Explorer.
- 2) Open the Anaconda Navigator.
- 3) Activate the environment you want to work on (see Fig. 13).
- 4) Launch JupyterLab from the Navigator.
- 5) In JupyterLab, navigate to the required folder in one of the following ways (see Fig. 14):
 - a. In the JupyterLab File Browser, navigate through the folders to the folder containing the Notebook.
 - b. Under the File tab, click on “Open from Path...” and paste the path of the folder opened at step 1.
- 6) In the JupyterLab File Browser, double click on the Jupyter Notebook to open it.

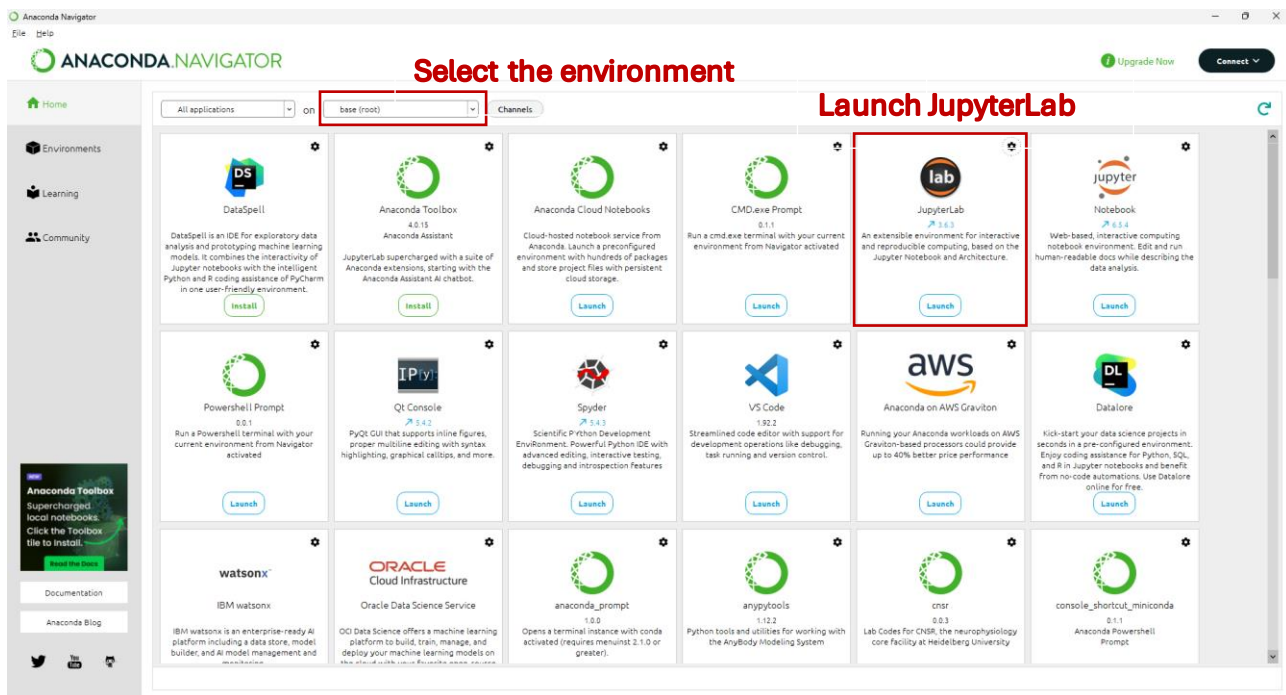


Figure 13: Anaconda Navigator interface to launch JupyterLab within a selected environment.

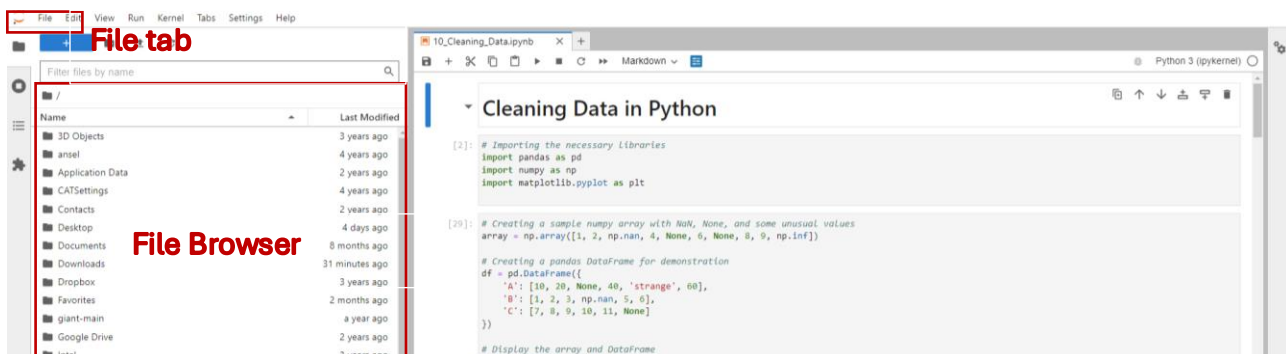


Figure 14: JupyterLab interface with the File tab and File Browser to open a chosen folder or Notebook.

If a custom installation has been performed, and assuming the environment, interpreter, and modules can be selected within the user's IDE (Custom):

- 1) Click on the links below, this opens the folder where the Notebook is in Explorer.
- 2) Select "<filename>.ipynb"
- 3) Select "Open with..." and select the IDE to be used.

4.1. Coding Fundamentals

[Python Basics](#)

[Arithmetic Operations in Python](#)

[Python Data Types](#)

[Indexing in Python](#)

[Python Arrays and Matrices](#)

[Introduction to Lambda Functions in Python](#)

[Introduction to Loops and Control Flow in Python](#)

[Introduction to Classes and Functions in Python](#)

4.2. Data Importing

[Introduction to Importing Data in Python](#)

[Importing and Reading Images in Python](#)

[Cleaning Data in Python](#)

4.3. Data Processing and Applications

[Performing Linear Least-Square Fitting](#)

[Performing Nonlinear Least-Square Fitting](#)

[Fitting and Statistical Distributions](#)

[Data Interpolation in Python](#)

[Numerical Integration in Python](#)

[Introduction to Symbolic Coding in Python](#)

[Fast-Fourier Transform in Python](#)

[Finding Roots of Functions in Python](#)

[Solving ODEs in Python](#)

[Units of Measurement in Python](#)

4.4. Data Export and Visualization

[2D Plotting in Python](#)

[3D Plotting in Python](#)

4.5. Good to Know

[Using the Documentation in Python](#)

[Parallel Computing in Python](#)

[Custom Modules in Python](#)

[Managing the Search Path in Python](#)

5. Where to go from here...

This document provides instructions on how to first approach learning Python, from understanding the concepts behind the language to a library of code examples to show the language capabilities.

Novice users can rely on guided tutorials to understand better the core concepts. [Microsoft](#)²⁶ provides free and comprehensive tutorials to learn Python. [Datacamp](#)²⁷ also provides interactive courses (free to a limited extent). Additionally, following along roadmaps (e.g. [Python Roadmap](#)²⁸) can be helpful to learn the fundamentals in the proper order.

Intermediate and autonomous users can rely on [suggested books](#)²⁹ to learn more about using Python in their applications. If you are already familiar with basic syntax, spending time on the user guide of scientific libraries (e.g. [SciPy](#)³⁰, [NumPy](#)³¹, [Control](#)³²).

Additional Resources

Examples Library:

²⁶ <https://learn.microsoft.com/en-us/training/paths/beginner-python/>

²⁷ <https://www.datacamp.com/>

²⁸ <https://roadmap.sh/python>

²⁹ <https://wiki.python.org/moin/IntroductoryBooks>

³⁰ <https://docs.scipy.org/doc/scipy-1.14.0/tutorial/index.html#user-guide>

³¹ <https://numpy.org/doc/stable/user/>

³² <https://python-control.readthedocs.io/en/0.10.0/>

[Python Code Snippets](#)

[Python Snippets](#)

[30 Days of Python](#)

Documentation:

[Scipy Documentation](#)

[NumPy Documentation](#)

[Matplotlib Documentation](#)

[Pandas Documentation](#)

[SymPy Documentation](#)

Interactive Courses:

[Datacamp](#)

[learnpython.org](#)

[Python for Beginners \(Microsoft\)](#)

Guides/Courses:

[Python Beginners Guide \(Python\)](#)

[Introductory Books \(Python\)](#)

[The Python Tutorial \(Python\)](#)

[Introduction To Computer Science And Programming In Python \(MIT OpenCourseWare\)](#)

[Get started with Python using Windows \(Microsoft\)](#)

[Anaconda Learning](#)

Roadmaps:

[Python Roadmap](#)

[PyFlo](#)