CHAPTER 2

Introduction to the Python World

The Python language, and the world around it, is made by interpreters, tools, editors, libraries, notebooks, and so on. This Python world has expanded greatly in recent years, enriching and taking forms that developers who approach it for the first time can sometimes find complicated and somewhat misleading. Thus, if you are approaching Python for the first time, you might feel lost among so many choices, especially about where to start.

This chapter gives you an overview of the entire Python world. You'll first gain an introduction to the Python language and its unique characteristics. You'll learn where to start, what an interpreter is, and how to begin writing your first lines of code in Python before being presented with some new and more advanced forms of interactive writing with respect to shells, such as IPython and the IPython Notebook.

Python—The Programming Language

The Python programming language was created by Guido Von Rossum in 1991 and started with a previous language called ABC. This language can be characterized by a series of adjectives:

- Interpreted
- Portable
- Object-oriented
- Interactive
- Interfaced
- Open source
- Easy to understand and use

Python is an *interpreted* programming language, that is, it's pseudo-compiled. Once you write the code, you need an *interpreter* to run it. An interpreter is a program that is installed on each machine; it interprets and runs the source code. Unlike with languages such as C, C++, and Java, there is no compile time with Python.

Python is a highly *portable* programming language. The decision to use an interpreter as an interface for reading and running code has a key advantage: portability. In fact, you can install an interpreter on any platform (Linux, Windows, and Mac) and the Python code will not change. Because of this, Python is often used with many small-form devices, such as the Raspberry Pi and other microcontrollers.

Python is an *object-oriented* programming language. In fact, it allows you to specify classes of objects and implement their inheritance. But unlike C++ and Java, there are no constructors or destructors. Python also allows you to implement specific constructs in your code to manage exceptions. However, the structure of the language is so flexible that it allows you to program with alternative approaches with respect to the object-oriented one. For example, you can use functional or vectorial approaches.

Python is an *interactive* programming language. Thanks to the fact that Python uses an interpreter to be executed, this language can take on very different aspects depending on the context in which it is used. In fact, you can write long lines of code, similar to what you might do in languages like C++ or Java, and then launch the program, or you can enter the command line at once and execute a command, immediately getting the results. Then, depending on the results, you can decide what command to run next. This highly interactive way to execute code makes the Python computing environment similar to MATLAB. This feature of Python is one reason it's popular with the scientific community.

Python is a programming language that can be *interfaced*. In fact, this programming language can be interfaced with code written in other programming languages such as C/C++ and FORTRAN. Even this was a winning choice. In fact, thanks to this aspect, Python can compensate for what is perhaps its only weak point, the speed of execution. The nature of Python, as a highly dynamic programming language, can sometimes lead to execution of programs up to 100 times slower than the corresponding static programs compiled with other languages. The solution to this kind of performance problem is to interface Python to the compiled code of other languages by using it as if it were its own.

Python is an *open-source* programming language. CPython, which is the reference implementation of the Python language, is completely free and open source. Additionally every module or library in the network is open source and their code is available online. Every month, an extensive developer community includes improvements to make this language and all its libraries even richer and more efficient. CPython is managed by the nonprofit Python Software Foundation, which was created in 2001 and has given itself the task of promoting, protecting, and advancing the Python programming language.

Finally, Python is a simple language to use and learn. This aspect is perhaps the most important, because it is the most direct aspect that a developer, even a novice, faces. The high intuitiveness and ease of reading of Python code often leads to "sympathy" for this programming language, and consequently most newcomers to programming choose to use it. However, its simplicity does not mean narrowness, since Python is a language that is spreading in every field of computing. Furthermore, Python is doing all of this very simply, in comparison to existing programming languages such as C++, Java, and FORTRAN, which by their nature are very complex.

The Interpreter and the Execution Phases of the Code

Unlike programming languages such as Java or C, whose code must be compiled before being executed, Python is a language that allows direct execution of instructions. In fact, it is possible to execute code written in Python it two ways. You can execute entire programs (`.py` files) by running the `python` command followed by the file name, or you can open a session through a special command console, characterized by a `>>>` prompt (running the `python` command with no arguments). In this console, you can enter one instruction at a time, obtaining the result immediately by executing it directly.

In both cases, you have the immediate execution of the inserted code, without having to go through explicit compilation or other operations.

This direct execution operation can be schematized in four phases:

- Lexing or tokenization
- Parsing
- Compiling
- Interpreting

Lexing, or *tokenization*, is the initial phase in which the Python (human-readable) code is converted into a sequence of logical entities, the so-called lexical tokens (see Figure 2-1).

Parsing is the next stage in which the syntax and grammar of the lexical tokens are checked by a parser, which produces an abstract syntax tree (AST) as a result.

Compiling is the phase in which the compiler reads the AST and, based on this information, generates the Python bytecode (`.pyc` or `.pyo` files), which contains very basic execution instructions. Although this is a compilation phase, the generated bytecode is still platform-independent, which is very similar to what happens in the Java language.

The last phase is *interpreting*, in which the generated bytecode is executed by a *Python virtual machine (PVM)*.

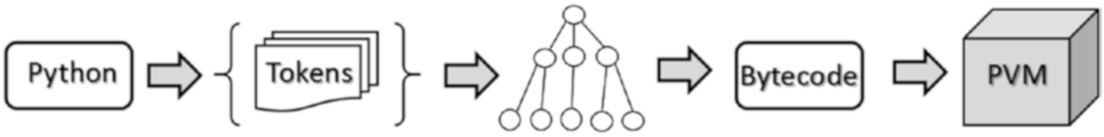


Figure 2-1. *The steps performed by the Python interpreter*

You can find good documentation on this process at `www.ics.uci.edu/~pattis/ICS-31/lectures/tokens.pdf`.

All these phases are performed by the interpreter, which in the case of Python is a fundamental component. When referring to the Python interpreter, this usually means the `/urs/bin/python` binary. In reality, there are currently several versions of this Python interpreter, each of which is profoundly different in its nature and specifications.

CPython

The standard Python interpreter is CPython, and it was written in C. This made it possible to use C-based libraries over Python. CPython is available on a variety of platforms, including ARM, iOS, and RISC. Despite this, CPython has been optimized on portability and other specifications, but not on speed.

Cython

The strongly intrinsic nature of C in the CPython interpreter has been taken further with the Cython project. This project is based on creating a compiler that translates Python code into C. This code is then executed within a Cython environment at runtime. This type of compilation system makes it possible to introduce C semantics into the Python code to make it even more efficient. This system has led to the merging of two worlds of programming language with the birth of *Cython,* which can be considered a new programming language. You can find documentation about it online. I advise you to visit `cython.readthedocs.io/en/latest/`.

Pyston

Pyston (`www.pyston.org/`) is a fork of the CPython interpreter that implements performance optimization. This project arises precisely from the need to obtain an interpreter that can replace CPython over time to remedy its poor performance in terms of execution speed. Recent results seem to confirm these predictions, reporting a 30 percent improvement in performance in the case of large, real-world applications. Unfortunately, due to the lack of compatible binary packages, Pyston packages have to be rebuilt during the download phase.

Jython

In parallel to Cython, there is a version built and compiled in Java, called *Jython*. It was created by Jim Hugunin in 1997 (www.jython.org/). Jython is an implementation of the Python programming language in Java; it is further characterized by using Java classes instead of Python modules to implement extensions and packages of Python.

IronPython

Even the .NET framework offers the possibility of being able to execute Python code inside it. For this purpose, you can use the IronPython interpreter (https://ironpython.net/). This interpreter allows .NET developers to develop Python programs on the Visual Studio platform, integrating perfectly with the other development tools of the .NET platform.

Initially built by Jim Hugunin in 2006 with the release of version 1.0, the project was later supported by a small team at Microsoft until version 2.7 in 2010. Since then, numerous other versions have been released up to the current 3.4, all ported forward by a group of volunteers on Microsoft's CodePlex repository.

PyPy

The PyPy interpreter is a JIT (just-in-time) compiler, and it converts the Python code directly to machine code at runtime. This choice was made to speed up the execution of Python. However, this choice has led to the use of a smaller subset of Python commands, defined as *RPython*. For more information on this, consult the official website at www.pypy.org/.

RustPython

As the name suggests, RustPython (rustpython.github.io/) is a Python interpreter written in Rust. This programming language is quite new but it is gaining popularity. RustPython is an interpreter like CPython but can also be used as a JIT compiler. It also allows you to run Python code embedded in Rust programs and compile the code into WebAssembly, so you can run Python code directly from web browsers.

Installing Python

In order to develop programs in Python, you have to install it on your operating system. Linux distributions and macOS X machines should have a preinstalled version of Python. If not, or if you want to replace that version with another, you can easily install it. The process for installing Python differs from operating system to operating system. However, it is a rather simple operation.

On Debian-Ubuntu Linux systems, the first thing to do is to check whether Python is already installed on your system and what version is currently in use.

Open a terminal (by pressing ALT+CTRL+T) and enter the following command:

```
python3 --version
```

If you get the version number as output, then Python is already present on the Ubuntu system. If you get an error message, Python hasn't been installed yet.

In this last case

```
sudo apt install python3
```

If, on the other hand, the current version is old, you can update it with the latest version of your Linux distribution by entering the following command:

```
sudo apt --only-upgrade install python3
```

Finally, if instead you want to install a specific version on your system, you have to explicitly indicate it in the following way:

```
sudo apt install python3.10
```

On Red Hat and CentOS Linux systems working with `rpm` packages, run this command instead:

```
yum install python3
```

If you are running Windows or macOS X, you can go to the official Python site (`www.python.org`) and download the version you prefer. The packages in this case are installed automatically.

However, today there are distributions that provide a number of tools that make the management and installation of Python, all libraries, and associated applications easier. I strongly recommend you choose one of the distributions available online.

Python Distributions

Due to the success of the Python programming language, many Python tools have been developed to meet various functionalities over the years. There are so many that it's virtually impossible to manage all of them manually.

In this regard, many Python distributions efficiently manage hundreds of Python packages. In fact, instead of individually downloading the interpreter, which includes only the standard libraries, and then needing to individually install all the additional libraries, it is much easier to install a Python distribution.

At the heart of these distributions are the *package managers,* which are nothing more than applications that automatically manage, install, upgrade, configure, and remove Python packages that are part of the distribution.

Their functionality is very useful, since the user simply makes a request regarding a particular package (which could be an installation for example). Then the package manager, usually via the Internet, performs the operation by analyzing the necessary version, alongside all dependencies with any other packages, and downloads them if they are not present.

Anaconda

Anaconda is a free distribution of Python packages distributed by Continuum Analytics (`www.anaconda.com`). This distribution supports Linux, Windows, and macOS X operating systems. Anaconda, in addition to providing the latest packages released in the Python world, comes bundled with most of the tools you need to set up a Python development environment.

Indeed, when you install the Anaconda distribution on your system, you can use many tools and applications described in this chapter, without worrying about having to install and manage them separately. The basic distribution includes *Spyder*, an IDE used to develop complex Python programs, *Jupyter Notebook*, a wonderful tool for working interactively with Python in a graphical and orderly way, and *Anaconda Navigator*, a graphical panel for managing packages and virtual environments.

The management of the entire Anaconda distribution is performed by an application called *conda*. This is the package manager and the environment manager of the Anaconda distribution and it handles all of the packages and their versions.

```
conda install <package name>
```

One of the most interesting aspects of this distribution is the ability to manage multiple development environments, each with its own version of Python. With Anaconda, you can work simultaneously and independently with different Python versions at the same time, by creating several virtual environments. You can create, for instance, an environment based on Python 3.11 even if the current Python version is still 3.10 in your system. To do this, you write the following command via the console:

```
conda create -n py311 python=3.11 anaconda
```

This will generate a new Anaconda virtual environment with all the packages related to the Python 3.11 version. This installation will not affect the Python version installed on your system and won't generate any conflicts. When you no longer need the new virtual environment, you can simply uninstall it, leaving the Python system installed on your operating system completely unchanged. Once it's installed, you can activate the new environment by entering the following command:

```
source activate py311
```

On Windows, use this command instead:

```
activate py311
C:\Users\Fabio>activate py311
 (py311) C:\Users\Fabio>
```

You can create as many versions of Python as you want; you need only to change the parameter passed with the `python` option in the `conda create` command. When you want to return to work with the original Python version, use the following command:

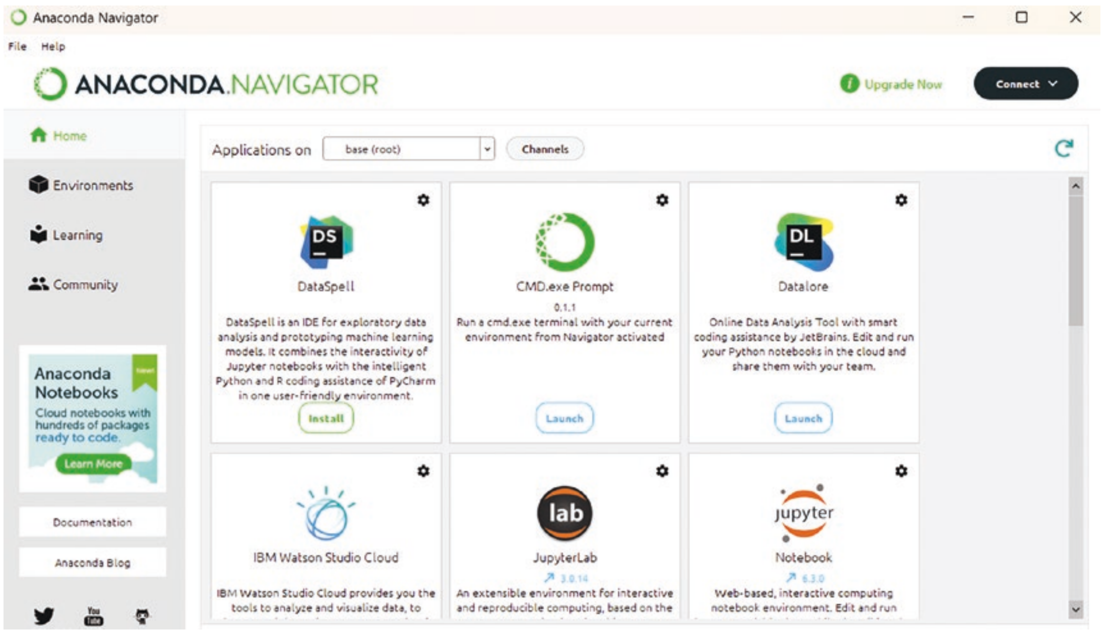
```
source deactivate
```

On Windows, use this command:

```
(py311) C:\Users\Fabio>deactivate
Deactivating environment "py311"...
C:\Users\Fabio>
```

Anaconda Navigator

Although at the base of the Anaconda distribution there is the `conda` command for the management of packages and virtual environments, working through the command console is not always practical and efficient. As you will see in the following chapters of the book, Anaconda provides a graphical tool called Anaconda Navigator, which allows you to manage the virtual environments and related packages in a graphical and very simplified way (see Figure 2-2).



DataSpell is an IDE for exploratory data analysis and prototyping machine learning models. It combines the interactivity of Jupyter notebooks with the intelligent Python and R coding assistance of PyCharm in one user-friendly environment.

Run a cmd.exe terminal with your current environment from Navigator activated

Online Data Analysis Tool with smart coding assistance by JetBrains. Edit and run your Python notebooks in the cloud and share them with your team.

IBM Watson Studio Cloud provides you the tools to analyze and visualize data, to

An extensible environment for interactive and reproducible computing, based on the

Web-based, interactive computing notebook environment. Edit and run

Figure 2-2. *Home panel of Anaconda Navigator*

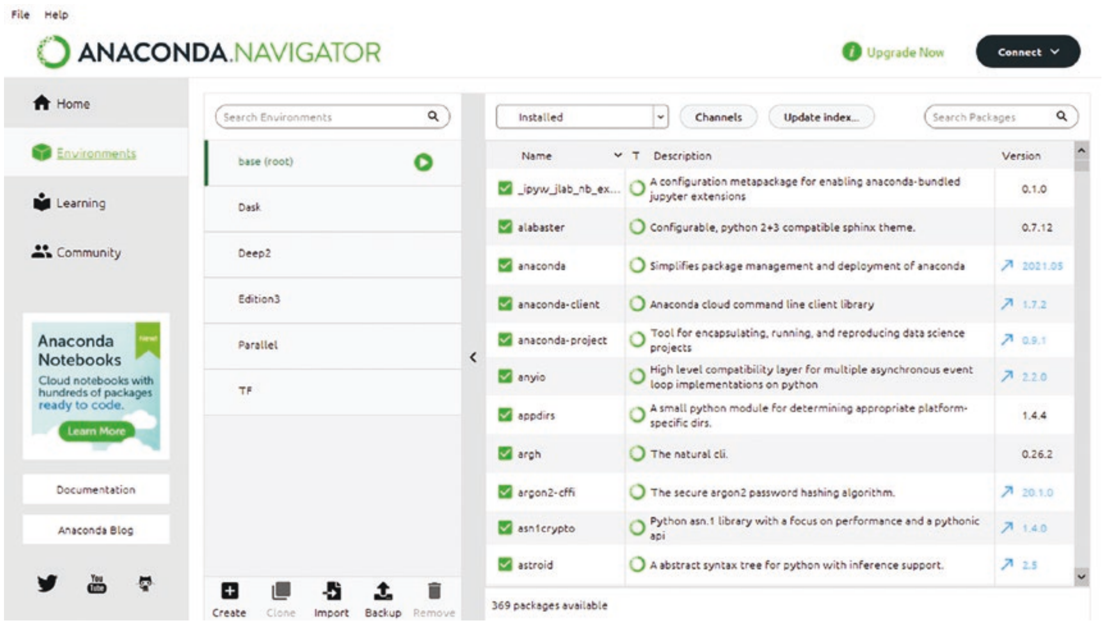
Anaconda Navigator is mainly composed of four panels:

- Home
- Environments
- Learning
- Community

Each of them is selectable through the list of buttons clearly visible on the left.

The Home panel presents all the Python (and also R) development applications installed (or available) for a given virtual environment. By default, Anaconda Navigator will show the base operating system environment, referred as base(root) in the top-center drop-down menu (see Figure 2-2).

The second panel, called Environments, shows all the virtual environments created in the distribution (see Figure 2-3). From there, it is possible to select the virtual environment to activate by clicking it directly. It will display all the packages installed (or available) on that virtual environment, with the relative versions.



Name	T	Description	Version
✓ _ipyw_jlab_nb_ex...		A configuration metapackage for enabling anaconda-bundled jupyter extensions	0.1.0
✓ alabaster		Configurable, python 2+3 compatible sphinx theme.	0.7.12
✓ anaconda		Simplifies package management and deployment of anaconda	↗ 2021.05
✓ anaconda-client		Anaconda cloud command line client library	↗ 1.7.2
✓ anaconda-project		Tool for encapsulating, running, and reproducing data science projects	↗ 0.9.1
✓ anyio		High level compatibility layer for multiple asynchronous event loop implementations on python	↗ 2.2.0
✓ appdirs		A small python module for determining appropriate platform-specific dirs.	1.4.4
✓ argh		The natural cli.	0.26.2
✓ argon2-cffi		The secure argon2 password hashing algorithm.	↗ 20.1.0
✓ asn1crypto		Python asn.1 library with a focus on performance and a pythonic api	↗ 1.4.0
✓ astroid		A abstract syntax tree for python with inference support.	↗ 2.5

369 packages available

Figure 2-3. *Environments panel on Anaconda Navigator*

Also from the Environments panel it is possible to create new virtual environments, selecting the basic Python version. Similarly, the same virtual environments can be deleted, cloned, backed up, or imported using the menu shown in Figure 2-4.

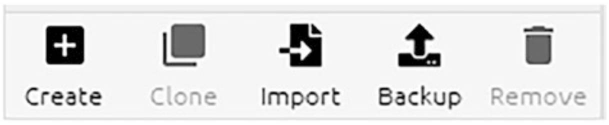


Figure 2-4. *Button menu for managing virtual environments in Anaconda Navigator*

But that is not all. Anaconda Navigator is not only a useful application for managing Python applications, virtual environments, and packages. In the third panel, called Learning (see Figure 2-5), it provides links to the main sites of many useful Python libraries (including those covered in this book). By clicking one of these links, you can access a lot of documentation. This is always useful to have on hand if you program in Python on a daily basis.

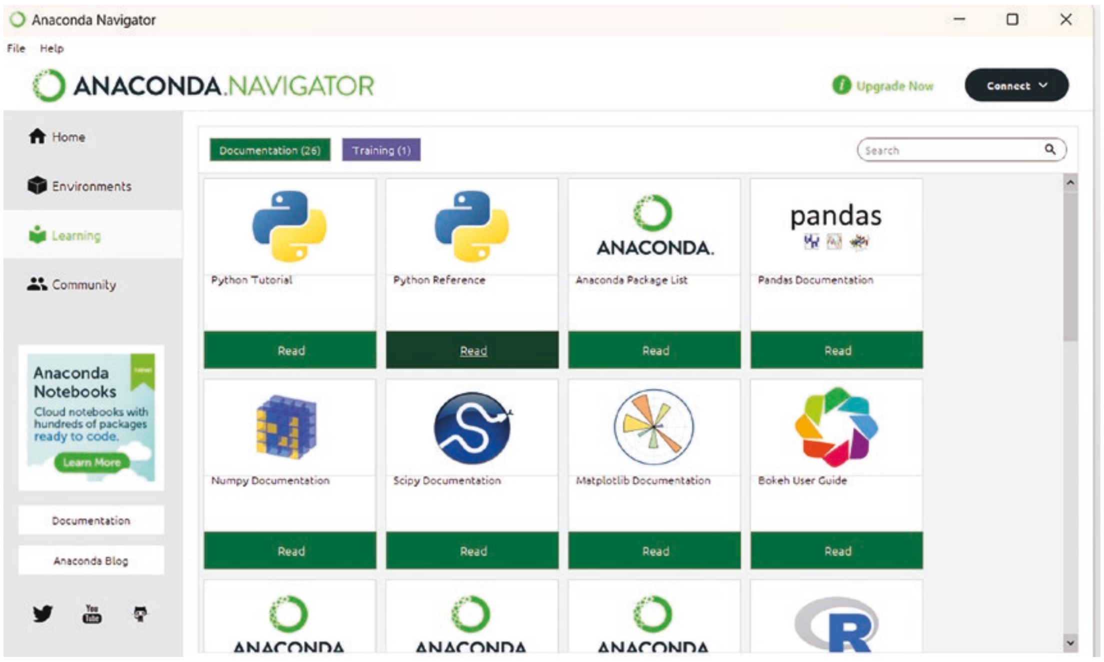


Figure 2-5. *Learning panel of Anaconda Navigator*

An identical panel to this is the next one, called Community. There are links here too, but this time to forums from the main Python development and Data Analytics communities.

The Anaconda platform, with its multiple applications and Anaconda Navigator, allows developers to take advantage of this simple and organized work environment and be well prepared for the development of Python code. It is no coincidence that this platform has become almost a standard for those belonging to the sector.

Using Python

Python is rich, but simple and very flexible. It allows you to expand your development activities in many areas of work (data analysis, scientific, graphic interfaces, etc.). Precisely for this reason, Python can be used in many different contexts, often according to the taste and ability of the developer. This section presents the various approaches to using Python in the course of the book. According to the various topics discussed in different chapters, these different approaches will be used specifically, as they are more suited to the task at hand.

Python Shell

The easiest way to approach the Python world is to open a session in the Python shell, which is a terminal running a command line. In fact, you can enter one command at a time and test its operation immediately. This mode makes clear the nature of the interpreter that underlies Python. In fact, the interpreter can read one command at a time, keeping the status of the variables specified in the previous lines, a behavior similar to that of MATLAB and other calculation software.

This approach is helpful when approaching Python the first time. You can test commands one at a time without having to write, edit, and run an entire program, which could be composed of many lines of code.

This mode is also good for testing and debugging Python code one line at a time, or simply to make calculations. To start a session on the terminal, simply type this on the command line:

```
C:\Users\nelli>python
Python 3.10 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:21) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The Python shell is now active and the interpreter is ready to receive commands in Python. Start by entering the simplest of commands, but a classic for getting started with programming.

```
>>> print("Hello World!")
Hello World!
```

If you have the Anaconda platform available on your system, you can open a Python shell related to a specific virtual environment you want to work on. In this case, from Anaconda Navigator, in the Home panel, activate the virtual environment from the drop-down menu and click the Launch button of the CMD.exe Prompt application, as shown in Figure 2-6.

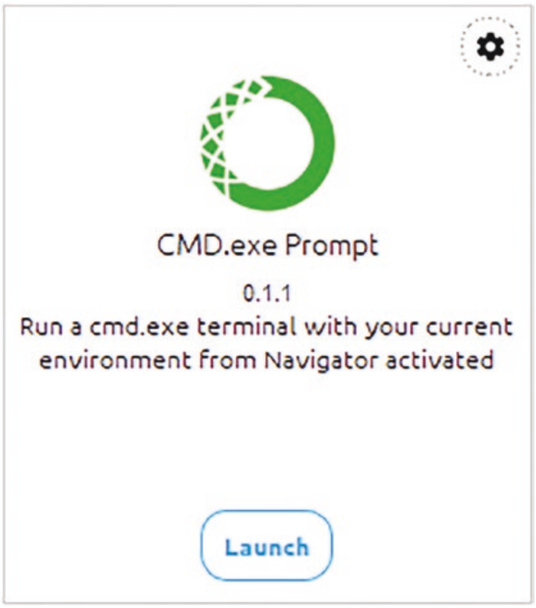

CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch

Figure 2-6. *CMD.exe Prompt application in Anaconda Navigator*

A command console will open with the name of the active virtual environment prefixed in brackets in the prompt. From there, you can run the `python` command to activate the Python shell.

```
(Edition3) C:\Users\nelli>python
Python 3.11.0 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:21) [MSC v.1916 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Run an Entire Program

The best way to become familiar with Python is to write an entire program and then run it from the terminal. First write a program using a simple text editor. For example, you can use the code shown in Listing 2-1 and save it as `MyFirstProgram.py`.

Listing 2-1. MyFirstProgram.py

```
myname = input("What is your name?\n")
print("Hi %s, I'm glad to say: Hello world!" %myname)
```

Now you've written your first program in Python, and you can run it directly from the command line by calling the `python` command and then the name of the file containing the program code.

```
python MyFirstProgram.py
```

From the output, the program will ask for your name. Once you enter it, it will say hello.

```
What is your name?
Fabio Nelli
Hi Fabio Nelli, I'm glad to say: Hello world!
```

Implement the Code Using an IDE

A more comprehensive approach than the previous ones is to use an IDE (an *Integrated Development Environment*). These editors provide a work environment on which to develop your Python code. They are rich in tools that make developers' lives easier, especially when debugging. In the following sections, you see in detail which IDEs are currently available.

Interact with Python

The last approach to using Python, and in my opinion, perhaps the most innovative, is the interactive one. In fact, in addition to the three previous approaches, this approach provides you with the opportunity to interact directly with the Python code.

In this regard, the Python world has been greatly enriched with the introduction of *IPython*. IPython is a very powerful tool, designed specifically to meet the needs of interacting between the Python interpreter and the developer, which under this approach takes the role of analyst, engineer, or researcher. IPython and its features are explained in more detail in a later section.

Writing Python Code

In the previous section, you saw how to write a simple program in which the string "Hello World" was printed. Now in this section, you get a brief overview of the basics of the Python language.

This section is not intended to teach you to program in Python, or to illustrate syntax rules of the programming language, but just to give you a quick overview of some basic principles of Python necessary to continue with the topics covered in this book.

If you already know the Python language, you can safely skip this introductory section. Instead, if you are not familiar with programming and you find it difficult to understand the topics, I highly recommend that you visit online documentation, tutorials, and courses of various kinds.

Make Calculations

You have already seen that the print() function is useful for printing almost anything. Python, in addition to being a printing tool, is a great calculator. Start a session on the Python shell and begin to perform these mathematical operations:

```
>>> 1 + 2
3
>>> (1.045 * 3)/4
0.78375
>>> 4 ** 2
16
>>> ((4 + 5j) * (2 + 3j))
(-7+22j)
>>> 4 < (2*3)
True
```

Python can calculate many types of data, including complex numbers and conditions with Boolean values. As you can see from these calculations, the Python interpreter directly returns the result of the calculations without the need to use the print() function. The same thing applies to values contained in variables. It's enough to call the variable to see its contents.

```
>>> a = 12 * 3.4
>>> a
40.8
```

Import New Libraries and Functions

You saw that Python is characterized by the ability to extend its functionality by importing numerous packages and modules. To import a module in its entirety, you have to use the import command.

```
>>> import math
```

In this way, all the functions contained in the math package are available in your Python session so you can call them directly. Thus, you have extended the standard set of functions available when you start a Python session. These functions are called with the following expression.

```
library_name.function_name()
```

For example, you can now calculate the sine of the value contained in the variable a.

```
>>> math.sin(a)
```

As you can see, the function is called along with the name of the library. Sometimes you might find the following expression for declaring an import.

```
>>> from math import *
```

Even if this works properly, it is to be avoided for good practice. In fact, writing an import in this way involves the importation of all functions without necessarily defining the library to which they belong.

```
>>> sin(a)
0.040693257349864856
```

This form of import can lead to very large errors, especially if the imported libraries are numerous. In fact, it is not unlikely that different libraries have functions with the same name, and importing all of these would result in an override of all functions with the same name that were previously imported. Therefore, the behavior of the program could generate numerous errors or worse, abnormal behavior.

Actually, this way to import is generally used for only a limited number of functions, that is, functions that are strictly necessary for the functioning of the program, thus avoiding the importation of an entire library when it is completely unnecessary.

```
 >>> from math import sin
```

Data Structure

You saw in the previous examples how to use simple variables containing a single value. Python provides a number of extremely useful data structures. These data structures can contain lots of data simultaneously and sometimes even data of different types. The various data structures provided are defined differently depending on how their data are structured internally.

- List
- Set
- Strings
- Tuples
- Dictionary
- Deque
- Heap

This is only a small part of all the data structures that can be made with Python. Among all these data structures, the most commonly used are *dictionaries* and *lists*.

The type *dictionary*, defined also as *dicts*, is a data structure in which each particular value is associated with a particular label, called a *key*. The data collected in a dictionary have no internal order but are only definitions of key/value pairs.

```
>>> dict = {'name':'William', 'age':25, 'city':'London'}
```

If you want to access a specific value within the dictionary, you have to indicate the name of the associated key.

```
>>> dict["name"]
'William'
```

If you want to iterate the pairs of values in a dictionary, you have to use the `for-in` construct. This is possible through the use of the `items()` function.

```
>>> for key, value in dict.items():
...    print(key,value)
...
name William
age 25
city London
```

The type *list* is a data structure that contains a number of objects in a precise order to form a sequence to which elements can be added and removed. Each item is marked with a number corresponding to the order of the sequence, called the *index*.

```
>>> list = [1,2,3,4]
>>> list
[1, 2, 3, 4]
```

If you want to access the individual elements, it is sufficient to specify the index in square brackets (the first item in the list has `0` as its index), while if you take out a portion of the list (or a sequence), it is sufficient to specify the range with the indices `i` and `j` corresponding to the extremes of the portion.

```
>>> list[2]
3
>>> list[1:3]
[2, 3]
```

If you are using negative indices instead, this means you are considering the last item in the list and gradually moving to the first.

```
>>> list[-1]
4
```

In order to do a scan of the elements of a list, you can use the `for-in` construct.

```
>>> items = [1,2,3,4,5]
>>> for item in items:
...        print(item + 1)
...
2
3
4
5
6
```

Functional Programming

The `for-in` loop shown in the previous example is very similar to loops found in other programming languages. However, if you want to be a "Python" developer, you have to avoid using explicit loops. Python offers alternative approaches, specifying programming techniques such as *functional programming* (expression-oriented programming).

The tools that Python provides to develop functional programming comprise a series of functions:

- `map(function, list)`
- `filter(function, list)`
- `reduce(function, list)`
- `lambda`
- `list comprehension`

The `for` loop that you just saw has a specific purpose, which is to apply an operation on each item and then somehow gather the result. This can be done by the `map()` function.

```
>>> items = [1,2,3,4,5]
>>> def inc(x): return x+1
...
>>> list(map(inc,items))
[2, 3, 4, 5, 6]
```

In the previous example, it first defines the function that performs the operation on every single element, and then it passes it as the first argument to `map()`. Python allows you to define the function directly within the first argument using `lambda` as a function. This greatly reduces the code and compacts the previous construct into a single line of code.

```
>>> list(map((lambda x: x+1),items))
[2, 3, 4, 5, 6]
```

Two other functions working in a similar way are `filter()` and `reduce()`. The `filter()` function extracts the elements of the list for which the function returns `True`. The `reduce()` function instead considers all the elements of the list to produce a single result. To use `reduce()`, you must import the `functools` module.

```
>>> list(filter((lambda x: x < 4), items))
[1, 2, 3]
>>> from functools import reduce
>>> reduce((lambda x,y: x/y), items)
0.008333333333333333
```

Both of these functions implement other types by using the `for` loop. They replace these cycles and their functionality, which can be alternatively expressed with simple functions. That is what constitutes *functional programming.*

The final concept of functional programming is *list comprehension*. This concept is used to build lists in a very natural and simple way, referring to them in a manner similar to how mathematicians describe datasets. The values in the sequence are defined through a particular function or operation.

```
>>> S = [x**2 for x in range(5)]
>>> S
[0, 1, 4, 9, 16]
```

Indentation

A peculiarity for those coming from other programming languages is the role that *indentation* plays. Whereas you used to manage the indentation for purely aesthetic reasons, making the code somewhat more readable, in Python indentation assumes an integral role in the implementation of the code, by dividing it into logical blocks. In fact, while in Java, C, and C++, each line of code is separated from the next by a semicolon (`;`), in Python you should not specify any symbol that separates them, included the braces to indicate a logical block.

These roles in Python are handled through indentation; that is, depending on the starting point of the code line, the interpreter determines whether it belongs to a logical block or not.

```
>>> a = 4
>>> if a > 3:
...    if a < 5:
...        print("I'm four")
... else:
...    print("I'm a little number")
...
I'm four
>>> if a > 3:
...    if a < 5:
...       print("I'm four")
...    else:
...       print("I'm a big number")
...
I'm four
```

In this example you can see that, depending on how the `else` command is indented, the conditions assume two different meanings (specified by me in the strings themselves).

IPython

IPython is a further development of Python that includes a number of tools:

- The IPython shell, which is a powerful interactive shell resulting in a greatly enhanced Python terminal.
- A QtConsole, which is a hybrid between a shell and a GUI, allowing you to display graphics inside the console instead of in separate windows.
- An IPython Notebook, called Jupyter Notebook, which is a web interface that allows you to mix text, executable code, graphics, and formulas in a single representation.

IPython Shell

This shell apparently resembles a Python session run from a command line, but actually, it provides many other features that make this shell much more powerful and versatile than the classic one. To launch this shell, just type `ipython` on the command line.

```
> ipython
Python 3.11.0 | packaged by Anaconda, Inc. | (main, Mar  1 2023, 18:18:21) [MSC v.1916 64
bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.12.0 -- An enhanced Interactive Python. Type '?' for help.
In [1]:
```

As you can see, a particular prompt appears with the value `In [1]`. This means that it is the first line of input. Indeed, IPython offers a system of numbered prompts (indexed) with input and output caching.

```
In [1]: print("Hello World!")
Hello World!
In [2]: 3/2
Out[2]: 1.5
In [3]: 5.0/2
Out[3]: 2.5
In [4]:
```

The same thing applies to values in output that are indicated with the values `Out[1]`, `Out [2]`, and so on. IPython saves all inputs that you enter by storing them as variables. In fact, all the inputs entered were included as fields in a list called `In`.

```
In [4]: In
Out[4]: [", 'print "Hello World!"', '3/2', '5.0/2', 'In']
```

The indices of the list elements are the values that appear in each prompt. Thus, to access a single line of input, you can simply specify that value.

```
In [5]: In[3]
Out[5]: '5.0/2'
```

For output, you can apply the same concept.

```
In [6]: Out
Out[6]:
{2: 1.5,
 3: 2.5,
 4: ['', 'print("Hello World!")', '3/2', '5.0/2', 'In', 'In[3]', 'Out'], 5: u'5.0/2'}
```

The Jupyter Project

IPython has grown enormously in recent times, and with the release of IPython 3.0, everything is moving toward a new project called Jupyter (https://jupyter.org)—see Figure 2-7.

Figure 2-7. *The Jupyter project logo*

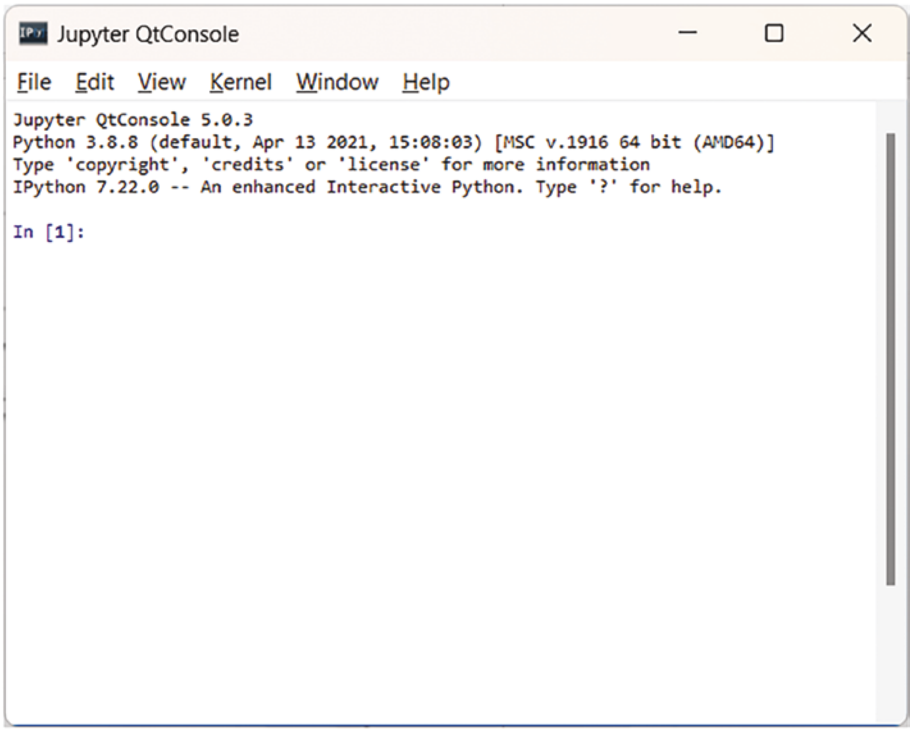
IPython will continue to exist as a Python shell and as a kernel of Jupyter, but the Notebook and the other language-agnostic components belonging to the IPython project will move to form the new Jupyter project.

Jupyter QtConsole

In order to launch this application from the command line, you must enter the following command:

```
jupyter qtconsole
```

The application consists of a GUI that has all the functionality present in the IPython shell. See Figure 2-8.



```
Jupyter QtConsole 5.0.3
Python 3.8.8 (default, Apr 13 2021, 15:08:03) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.22.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Figure 2-8. *The IPython QtConsole*

Jupyter Notebook

Jupyter Notebook is the latest evolution of this interactive environment (see Figure 2-9). In fact, with Jupyter Notebook, you can merge executable code, text, formulas, images, and animations into a single web document. This is useful for many purposes, such as presentations, tutorials, debugging, and so forth.

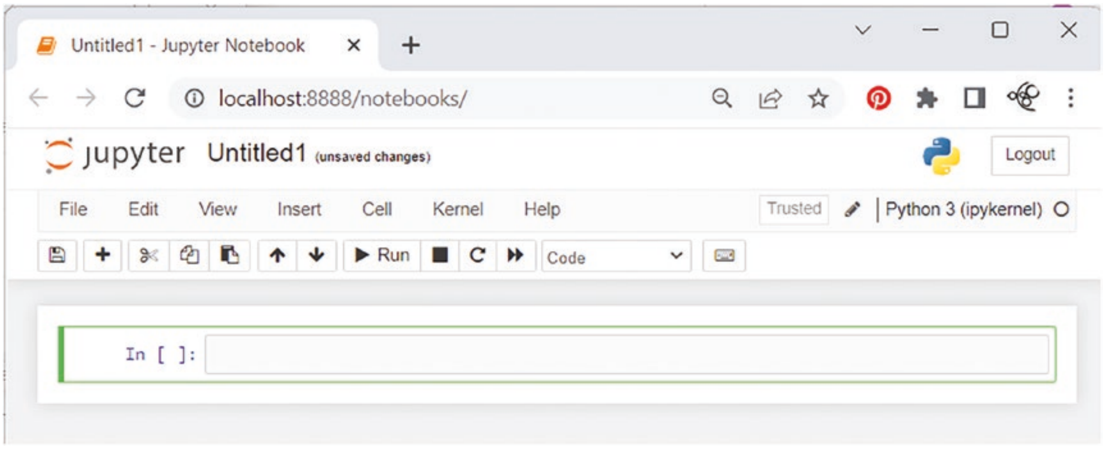
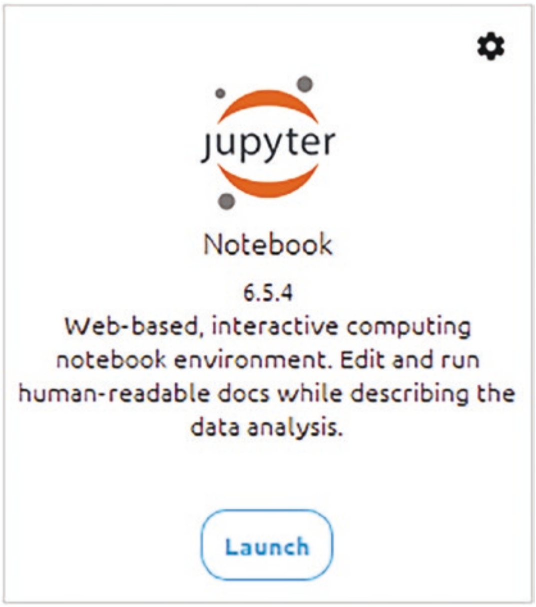

Untitled1 - Jupyter Notebook
localhost:8888/notebooks/
jupyter Untitled1 (unsaved changes)
Logout
File Edit View Insert Cell Kernel Help
Trusted
Python 3 (ipykernel)
Run
Code
In [ ]:

Figure 2-9. *The web page showing the Jupyter Notebook*

To start Jupyter Notebook on your web browser, run the following command from the console:

```
jupyter notebook
```

If instead you are working with Anaconda Navigator, in the Home panel, click the Launch button of the Jupyter Notebook application to start it (see Figure 2-10).



Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

Figure 2-10. *Jupyter Notebook application in Anaconda Navigator*

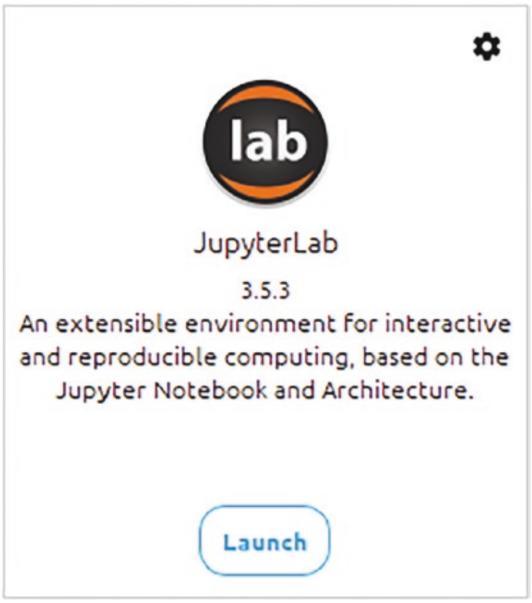
Jupyter Lab

Another application that brings together the characteristics of all the applications seen so far is Jupyter Lab. It runs on browsers like Jupyter Notebook, but it's a real development environment, where you can manage files, data, and code in the form of files, sessions, notebooks, and so on.

To start Jupyter Lab on your web browser, run the following command from the console:

```
jupyter lab
```

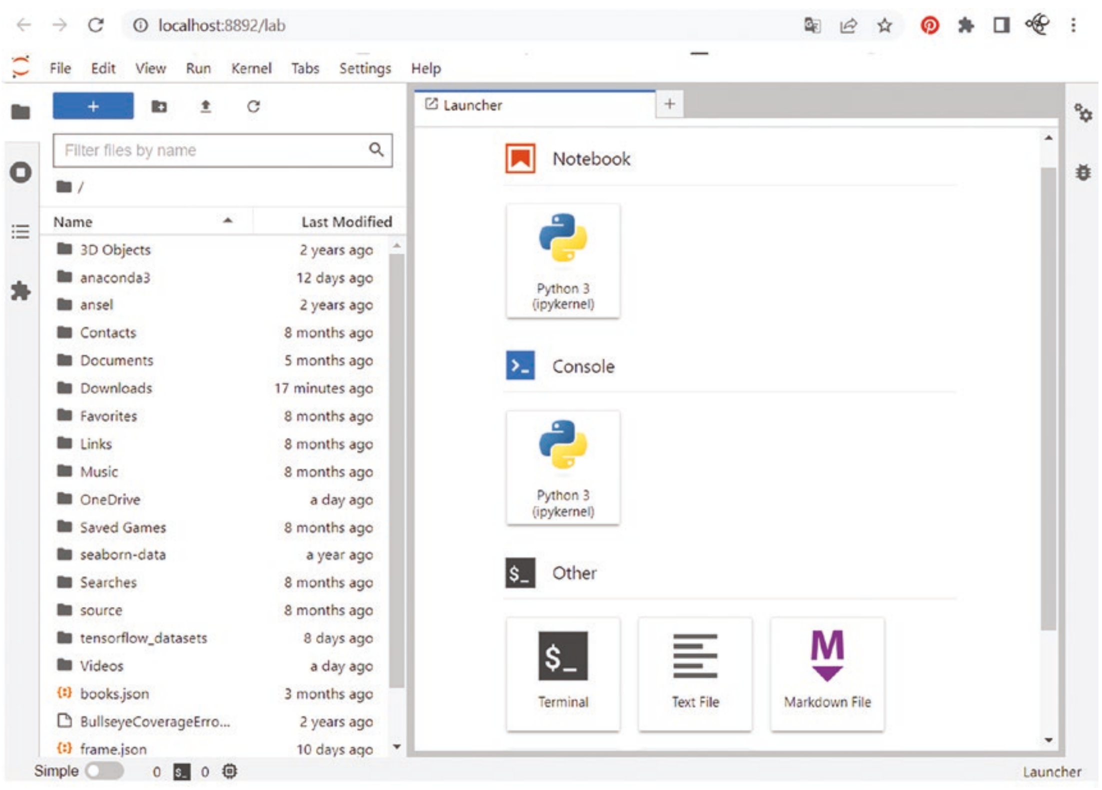
This application is also present on Anaconda Navigator together with the others, and to start it from one of the virtual environments, simply click the Launch button of the corresponding icon shown in Figure 2-11.



An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.

Figure 2-11. *Jupyter Lab icon in Anaconda Navigator*

Starting the application will open the default browser (if it's not already open) and load the `https://localhost:8892/lab` page, which corresponds to Jupyter Lab, as shown in Figure 2-12.



Name	Last Modified
3D Objects	2 years ago
anaconda3	12 days ago
ansel	2 years ago
Contacts	8 months ago
Documents	5 months ago
Downloads	17 minutes ago
Favorites	8 months ago
Links	8 months ago
Music	8 months ago
OneDrive	a day ago
Saved Games	8 months ago
seaborn-data	a year ago
Searches	8 months ago
source	8 months ago
tensorflow_datasets	8 days ago
Videos	a day ago
books.json	3 months ago
BullseyeCoverageErro...	2 years ago
frame.json	10 days ago

Figure 2-12. *Jupyter Lab application*

PyPI—The Python Package Index

The Python Package Index (PyPI) is a software repository that contains all the software needed for programming in Python—for example, all Python packages belonging to other Python libraries. The content repository is managed directly by the developers of individual packages that deal with updating the repository with the latest versions of their released libraries. For a list of the packages contained in the repository, go to the official page of PyPI at https://pypi.python.org/pypi.

As far as the administration of these packages, you can use the pip application, which is the package manager of PyPI.

By launching it from the command line, you can manage all the packages and individually decide if a package should be installed, upgraded, or removed. Pip will check if the package is already installed, or if it needs to be updated, to control dependencies, and to assess whether other packages are necessary. Furthermore, it manages the downloading and installation processes.

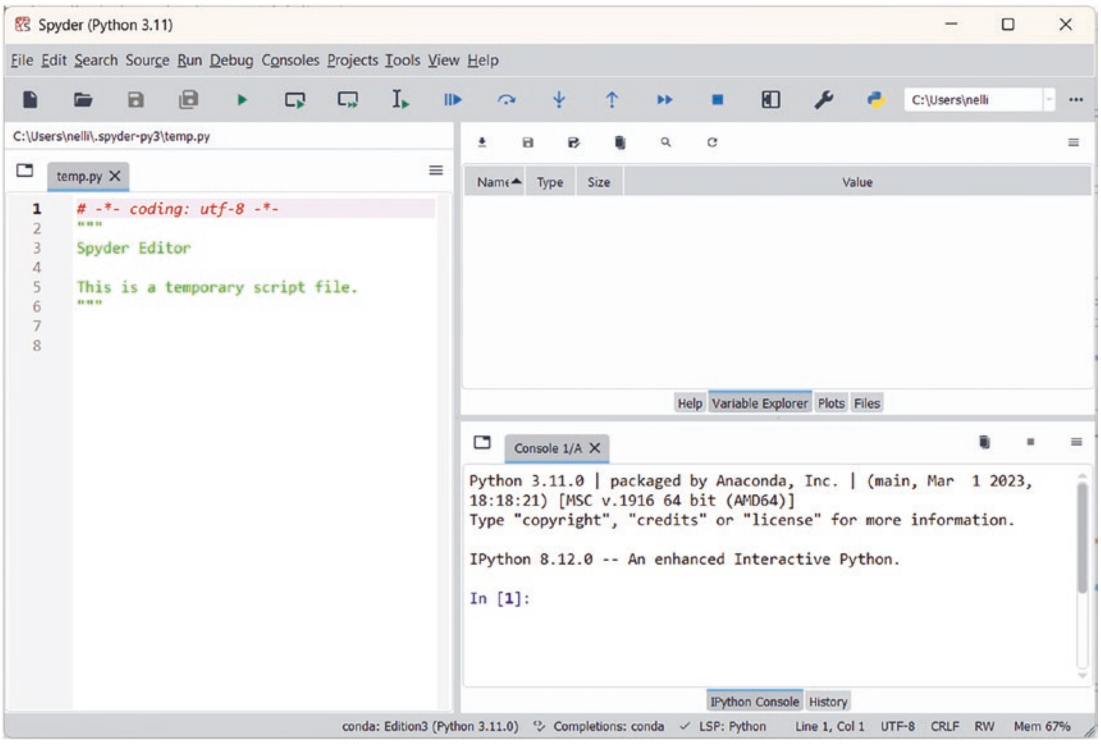
```
$ pip install <<package_name>>
$ pip search <<package_name>>
$ pip show <<package_name>>
$ pip unistall <<package_name>>
```

The IDEs for Python

Although most Python developers are used to implementing their code directly from the shell (Python or IPython), some IDEs (Interactive Development Environments) are also available. In fact, in addition to a text editor, these graphics editors also provide a series of tools that are very useful during the drafting of the code. For example, the auto-completion of code, viewing the documentation associated with the commands, debugging, and breakpoints are only some of the tools that this kind of application can provide.

Spyder

Spyder (Scientific Python Development Environment) is an IDE that has similar features to the IDE of MATLAB (see Figure 2-13). The text editor is enriched with syntax highlighting and code analysis tools. Also, you can integrate ready-to-use widgets in your graphic applications.


Spyder (Python 3.11)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\nelli
C:\Users\nelli\.spyder-py3\temp.py
temp.py
Name Type Size Value
Help Variable Explorer Plots Files
Console 1/A
IPython Console History
conda: Edition3 (Python 3.11.0) Completions: conda LSP: Python Line 1, Col 1 UTF-8 CRLF RW Mem 67%

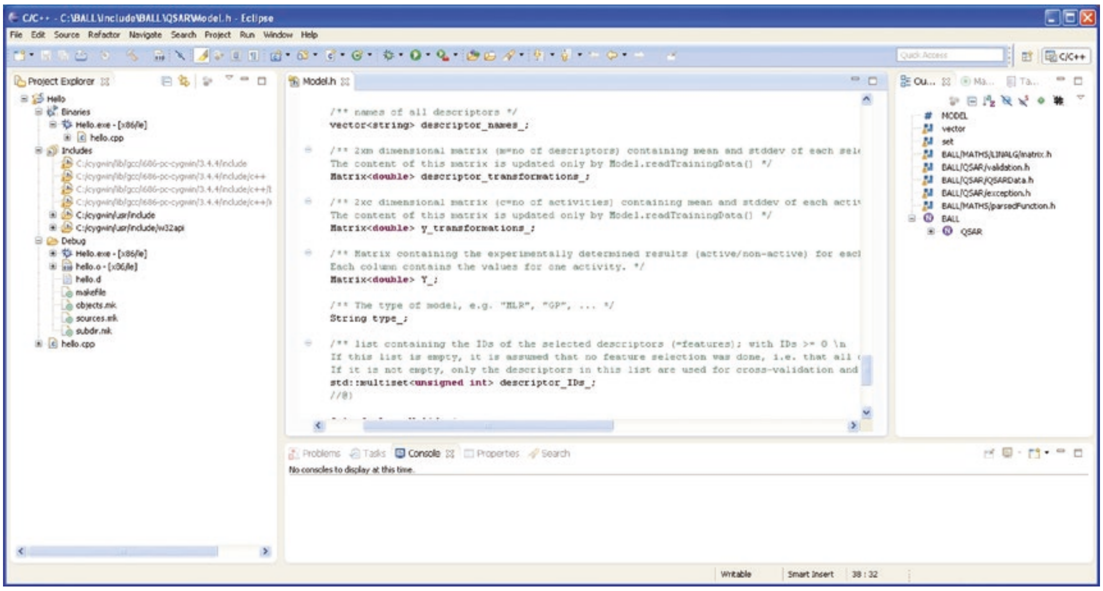
```
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""
```

```
Python 3.11.0 | packaged by Anaconda, Inc. | (main, Mar  1 2023,
18:18:21) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.12.0 -- An enhanced Interactive Python.

In [1]:
```

Figure 2-13. *The Spyder IDE*

Eclipse (pyDev)

Those of you who have developed in other programming languages certainly know Eclipse, a universal IDE developed entirely in Java (therefore requiring Java installation on your PC) that provides a development environment for many programming languages (see Figure 2-14). There is also an Eclipse version for developing in Python, thanks to the installation of an additional plugin called *pyDev*.


C/C++ - C:\BALL\include\BALL\QSAR\Model.h - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
Model.h
C/C++
Problems Tasks Console Properties Search
Writable Smart Insert 38 : 32

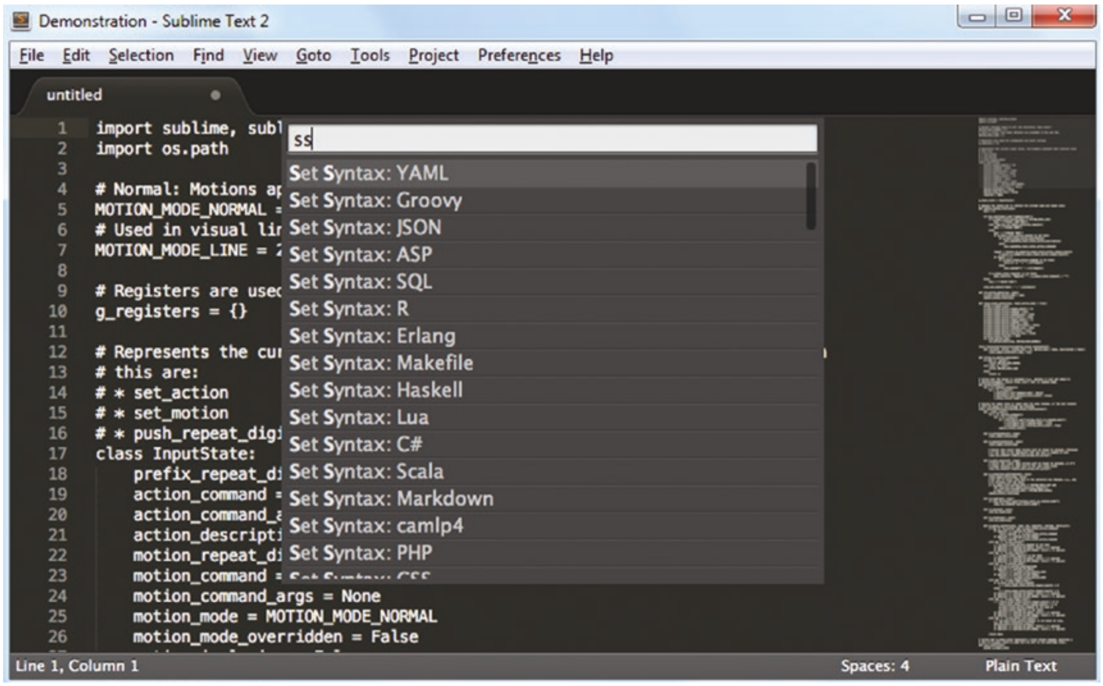
```
/** names of all descriptors */
vector<string> descriptor_names_;

/** 2xm dimensional matrix (m=no of descriptors) containing mean and stddev of each sel
The content of this matrix is updated only by Model.readTrainingData() */
Matrix<double> descriptor_transformations_;

/** 2xc dimensional matrix (c=no of activities) containing mean and stddev of each acti
The content of this matrix is updated only by Model.readTrainingData() */
Matrix<double> y_transformations_;

/** Matrix containing the experimentally determined results (active/non-active) for eac
Each column contains the values for one activity. */
Matrix<double> Y_;

/** The type of model, e.g. "MLR", "GP", ... */
String type_;

/** list containing the IDs of the selected descriptors (=features); with IDs >= 0 \n
If this list is empty, it is assumed that no feature selection was done, i.e. that all
If it is not empty, only the descriptors in this list are used for cross-validation and
std::multiset<unsigned int> descriptor_IDs_;
//@)
```

No consoles to display at this time.

Figure 2-14. The Eclipse IDE

Sublime

This text editor is one of the preferred environments for Python programmers (see Figure 2-15). In fact, there are several plugins available for this application that make Python implementation easy and enjoyable.

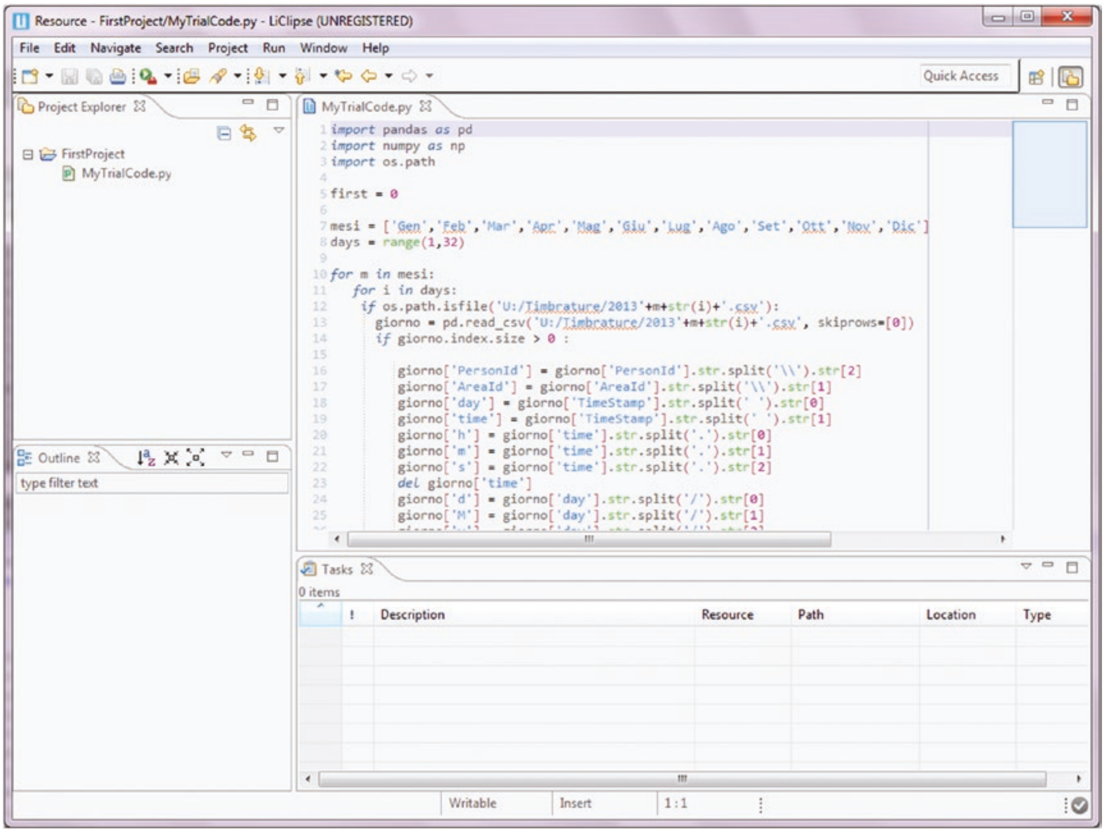


```
import sublime, subl
import os.path

# Normal: Motions ap
MOTION_MODE_NORMAL =
# Used in visual lir
MOTION_MODE_LINE =

# Registers are usec
g_registers = {}

# Represents the cur
# this are:
# * set_action
# * set_motion
# * push_repeat_digi
class InputState:
    prefix_repeat_di
    action_command =
    action_command_a
    action_descripti
    motion_repeat_di
    motion_command =
    motion_command_args = None
    motion_mode = MOTION_MODE_NORMAL
    motion_mode_overridden = False
```

Figure 2-15. *The Sublime IDE*

Liclipse

Liclipse, similarly to Spyder, is a development environment specifically designed for the Python language (see Figure 2-16). It is very similar to the Eclipse IDE but it is fully adapted for a specific use in Python, without needing to install plugins like PyDev. So its installation and settings are much simpler than Eclipse.

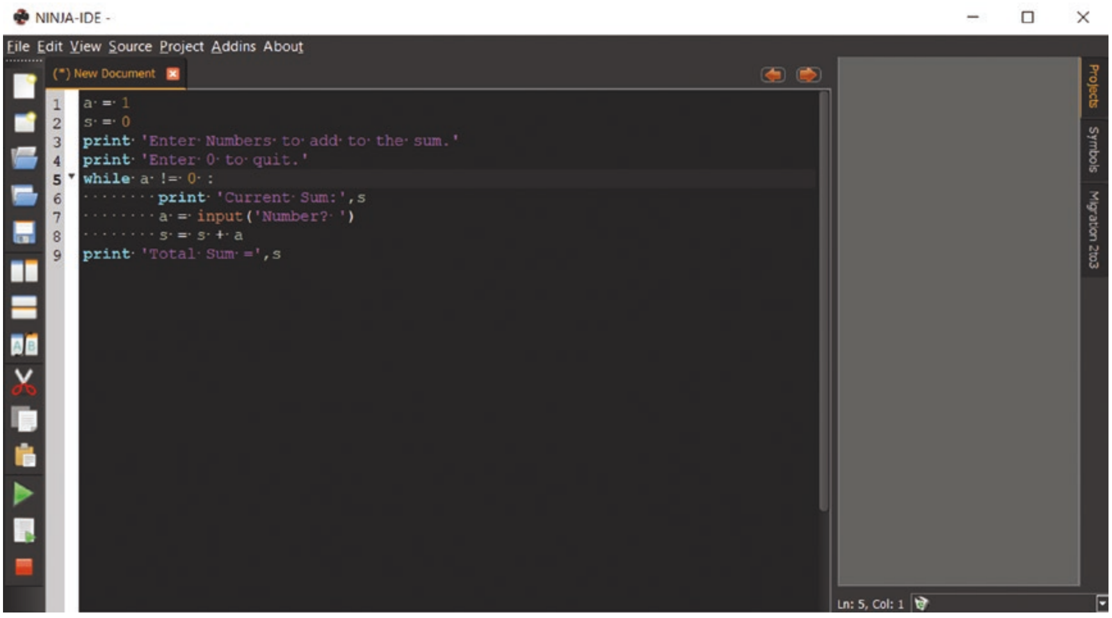

Resource - FirstProject/MyTrialCode.py - LiClipse (UNREGISTERED)
File Edit Navigate Search Project Run Window Help
Quick Access
Project Explorer
FirstProject
MyTrialCode.py
Outline
type filter text
MyTrialCode.py
Tasks
0 items
! Description Resource Path Location Type
Writable Insert 1:1

```
import pandas as pd
import numpy as np
import os.path

first = 0

mesi = ['Gen','Feb','Mar','Apr','Mag','Giu','Lug','Ago','Set','Ott','Nov','Dic']
days = range(1,32)

for m in mesi:
   for i in days:
    if os.path.isfile('U:/Timbrature/2013'+m+str(i)+'.csv'):
      giorno = pd.read_csv('U:/Timbrature/2013'+m+str(i)+'.csv', skiprows=[0])
      if giorno.index.size > 0 :

         giorno['PersonId'] = giorno['PersonId'].str.split('\\').str[2]
         giorno['AreaId'] = giorno['AreaId'].str.split('\\').str[1]
         giorno['day'] = giorno['TimeStamp'].str.split(' ').str[0]
         giorno['time'] = giorno['TimeStamp'].str.split(' ').str[1]
         giorno['h'] = giorno['time'].str.split('.').str[0]
         giorno['m'] = giorno['time'].str.split('.').str[1]
         giorno['s'] = giorno['time'].str.split('.').str[2]
         del giorno['time']
         giorno['d'] = giorno['day'].str.split('/').str[0]
         giorno['M'] = giorno['day'].str.split('/').str[1]
```

Figure 2-16. *The Liclipse IDE*

NinjaIDE

NinjaIDE (NinjaIDE is “Not Just Another IDE”), which characterized by a name that is a recursive acronym, is a specialized IDE for the Python language (see Figure 2-17). It’s a very recent application on which the efforts of many developers are focused. Already very promising, it is likely that in the coming years, this IDE will be a source of many surprises.

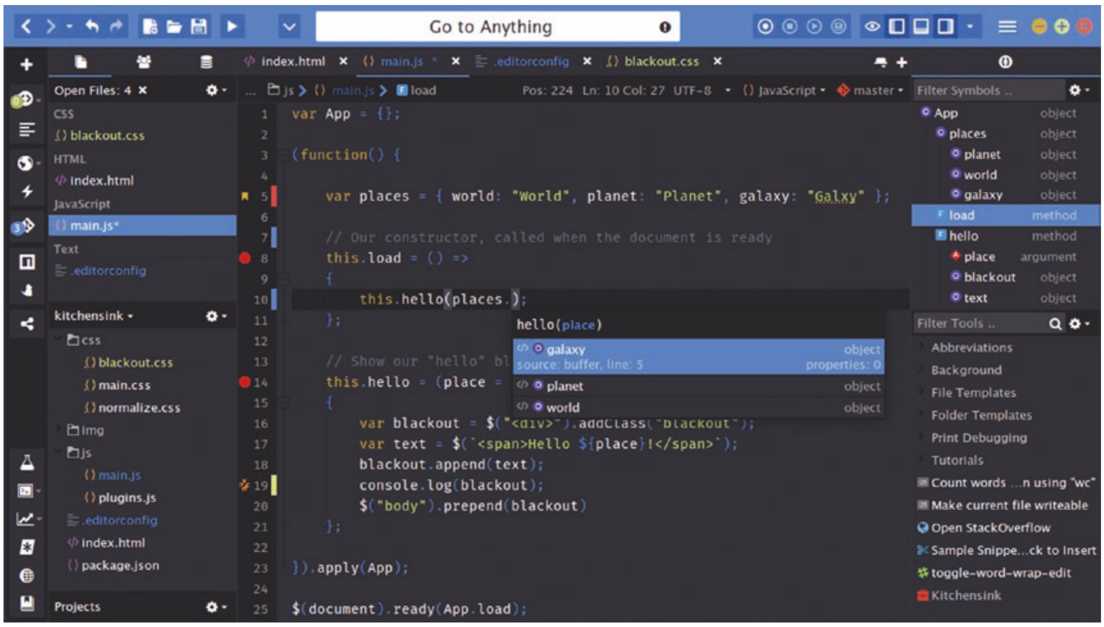


```
a = 1
s = 0
print 'Enter Numbers to add to the sum.'
print 'Enter 0 to quit.'
while a != 0 :
        print 'Current Sum:',s
        a = input('Number? ')
        s = s + a
print 'Total Sum =',s
```

Figure 2-17. *The Ninja IDE*

Komodo IDE

Komodo is a very powerful IDE full of tools that make it a complete and professional development environment (see Figure 2-18). Paid software and written in C++, the Komodo development environment is adaptable to many programming languages, including Python.



```
var App = {};

(function() {

    var places = { world: "World", planet: "Planet", galaxy: "Galxy" };

    // Our constructor, called when the document is ready
    this.load = () =>
    {
        this.hello(places.);
    };

    // Show our "hello" bl
    this.hello = (place =
    {
        var blackout = $("<div>").addClass("blackout");
        var text = $(`<span>Hello ${place}!</span>`);
        blackout.append(text);
        console.log(blackout);
        $("body").prepend(blackout)
    };

}).apply(App);

$(document).ready(App.load);
```

Figure 2-18. *The Komodo IDE*

SciPy

SciPy (pronounced "sigh pie") is a set of open-source Python libraries specialized for scientific computing. Many of these libraries are the protagonists of many chapters of the book, given that their knowledge is critical to data analysis. Together they constitute a set of tools for calculating and displaying data. It has little to envy from other specialized environments for calculation and data analysis (such as R or MATLAB). Among the libraries that are part of the SciPy group, there are three in particular that are discussed in the following chapters:

- NumPy
- matplotlib
- Pandas

NumPy

This library, whose name means *numerical Python,* constitutes the core of many other Python libraries that have originated from it. Indeed, NumPy is the foundation library for scientific computing in Python since it provides data structures and high-performing functions that the basic package of the Python cannot provide. In fact, as you will see later in the book, NumPy defines a specific data structure that is an *N*-dimensional array defined as *ndarray*.

Knowledge of this library is essential in terms of numerical calculations since its correct use can greatly influence the performance of your computations. Throughout the book, this library is almost omnipresent because of its unique characteristics, so an entire chapter is devoted to it (Chapter 3).

This package provides some features added to the standard Python:

- *Ndarray*: A multidimensional array much faster and more efficient than those provided by the basic package of Python.
- *Element-wise computation*: A set of functions for performing this type of calculation with arrays and mathematical operations between arrays.
- *Reading-writing datasets*: A set of tools for reading and writing data stored in the hard disk.
- *Integration with other languages such as C, C++, and FORTRAN*: A set of tools to integrate code developed with these programming languages.

Pandas

This package provides complex data structures and functions specifically designed to make the work on them easy, fast, and effective. This package is the core of data analysis in Python. Therefore, the study and application of this package is the main goal on which you will work throughout the book (especially in Chapters 4, 5, and 6). Knowledge of its every detail, especially when it is applied to data analysis, is a fundamental objective of this book.

The fundamental concept of this package is the *DataFrame*, a two-dimensional tabular data structure with row and column labels.

Pandas applies the high-performance properties of the NumPy library to the manipulation of data in spreadsheets or in relational databases (SQL databases). In fact, by using sophisticated indexing, it will be easy to carry out many operations on this kind of data structure, such as reshaping, slicing, aggregations, and the selection of subsets.

matplotlib

This package is the Python library that is currently the most popular for producing plots and other data visualizations in 2D. Because data analysis requires visualization tools, this library best suits this purpose. In Chapter 7, you learn about this rich library in detail so you will know how to represent the results of your analysis in the best way.

Conclusions

During the course of this chapter, all the fundamental aspects characterizing the Python world have been illustrated. The basic concepts of the Python programming language were introduced, with brief examples explaining its innovative aspects and how it stands out compared to other programming languages. In addition, different ways of using Python at various levels were presented. First you saw how to use a simple command-line interpreter, then a set of simple graphical user interfaces were shown until you got to complex development environments, known as IDEs, such as Spyder, Liclipse, and NinjaIDE.

Even the highly innovative project Jupyter (IPython) was presented, showing you how you can develop Python code interactively, in particular with the Jupyter Notebook.

Moreover, the modular nature of Python was highlighted with the ability to expand the basic set of standard functions provided by Python's external libraries. In this regard, the PyPI online repository was shown along with other Python distributions such as Anaconda and Enthought Canopy.

In the next chapter, you deal with the first library that is the basis of all numerical calculations in Python: NumPy. You learn about the ndarray, a data structure that is the basis of the more complex data structures used in data analysis in the following chapters.