



الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي
جامعة قسنطينة 1 – الإخوة منتوري
كلية العلوم الدقيقة

PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
Constantine 1 University – Frères Mentouri
Faculty of Exact Sciences



*Extrait du Procès-verbal N° 90 du 08 Février 2024.
du Conseil Scientifique de la Faculté*

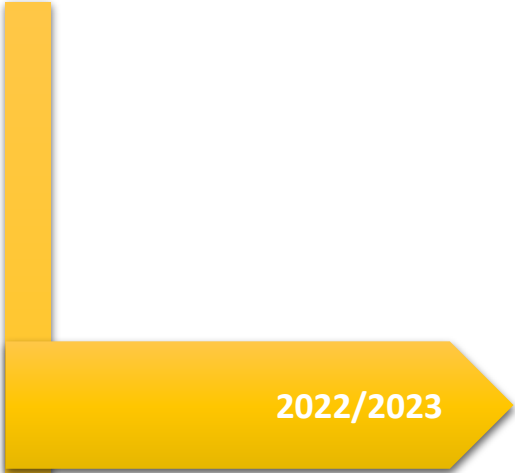
Suite aux rapports positifs du Professeur DALAH Mohamed et Docteur LAID Chahrazed chargés d'expertiser le polycopié de Mme **LABDAOUI Ahlam**, Maitres de conférences classe « A » intitulé : «*Python course*» destiné aux étudiants de 1ère année Doctorat option : Statistique appliquée, le Conseil scientifique émet un avis favorable pour sa publication.

Fait à Constantine le **20 MARS 2024**



Le Président du Conseil Scientifique

رئيس المجلس العلمي
لكلية العلوم الدقيقة
أ/ بودين بوبكر



2022/2023

Python course

For 1st year PhD students



Done by:

LABDAOUI AHLAM

0

Summaries

General introduction.....	5
---------------------------	---

Chapter 1: Introduction to Python

- What is Python?	6
- History of Python.....	6
- What can you do with Python?	7
- Python features.....	7
- Adapted from Python.....	8
- Different versions of Python.....	9
- Implementing Python.....	10
- Python environments.....	10
- Python distributions.....	14
- List of Python packages for data science.....	14
- Language tags.....	16

Chapter 2: Python-Anaconda-Jupyter installation

- Installing Python under Windows	17
• How to install Python on Windows ?.....	17
• How to install packages ?.....	21
- Installing Anaconda and Jupyter Notebook.....	32

Chapter 3: Basics

- Module and import.....	33
- Comments.....	35
- Display " print() function "	35
- Import CSV file.....	36

Chapter 4: Variables and functions

- Variables.....	37
○ Definition	37
○ Variable types.....	37
○ Naming.....	38
○ Operation.....	38
○ Maximum and minimum	40
- Function.....	4
○ Principle and generality.....	41
○ Definition.....	42
○ Passing arguments.....	42
○ Result forwarding.....	43

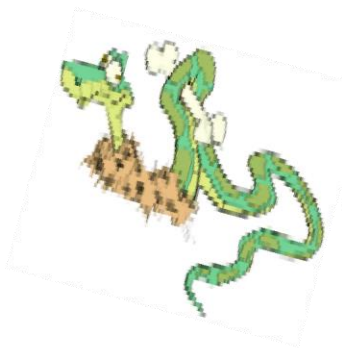
Chapter 5: Lists and tuples

- List.....	44
○ Definition	44
○ Use	44
○ List operation.....	45
○ Slices.....	45
○ len()function	46
○ The range() and list()functions	46
○ Minimum, maximum and sum of a list.....	47

- Tuples.....	48
○ Introduction.....	48
○ How do you write a function that returns a tuple of values?	49
○ How to iterate over the elements of a tuple ?.....	49
- How do I create a tuple containing a single element?...	49

Chapter 6: Loops and Testing

- Loops and comparison	50
✱ Loop for.....	50
✱ Comparison	52
✱ While loop.....	53
- Test.....	54
* Definition	54
* Multi-case test.....	55
* Multiple tests	55
* Instructions break and continue	56
General conclusion	57



Python identity card :

- ★ **Designed by :** Guido Van Rossum
- ★ **Date of first version:** 20/2/1991
- ★ **Version under development:** 3.10.0a5(3/2/2021)
- ★ **Latest version:** 3.9.1 (7/12/2020)
- ★ **Paradigms:** object, imperative and functional.
- ★ **Parent company:** Python Software Foundation.
- ★ **Influenced by :** ABC, C, Eiffel, ICON, Modula-3, Java, Perl, Small-talk, Tcl.
- ★ **Influenced by:** Ruby, Groovy, Boo, Julia

WHY PYTHON?

Python is an incredibly efficient language: your programs will do more in fewer lines of code than many other languages. Python's syntax will also help you write "clean" code. Your code will be easy to read, easy to debug, and easier to extend and develop than other languages.

Python is a great language to learn, so let us get started...

GENERAL INTRODUCTION

Programming is essentially getting your computer to do things. It is not the most technical definition, but it is an accurate one. Programming is part science, part art and part adventure. By learning Python, you will be able to create a program. Whether it is a simple game, a small utility or a business product with a full graphical user interface. It will be all yours, something you made, and it will do exactly what you said it would. The clear and relatively intuitive syntax of this language makes it an ideal candidate as part of an introduction to programming. So, if you've never programmed in any language, if you're only vaguely aware of what it means, Python is, it seems to me, a good choice to start your learning.

This course will set you apart on your Python programming journey. In it, you will learn:

- ★ *What is Python and what is so great about this language?*
- ★ *How to install python on your computer*
- ★ *What are the Python environments, distributions and packages?*
- ★ *What syntaxes are there and how do I use them?*

ENJOY
ENJOY

YOUR READING!
YOUR READING!

Chapter 1: Introduction to Python

I. What is Python?

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

- **Python is interpreted:** the interpreter processes Python at runtime. You do not need to compile your program before running it. This is similar to PERL and PHP.
- **Python is interactive:** You can actually sit down at a python prompt and interact directly with the interpreter to write your programs.
- **Python is object-oriented:** Python supports the style or technique of object-oriented programming, which encapsulates code in objects.

Python is an excellent language for novice programmers and supports the development of a wide range of applications, from simple word processors to WWW browsers and games.

II. Python story:

Guido van Rossum, in the Netherlands, created the Python programming language, in 1989. The name Python comes from a tribute to the TV series "Monty Python's Flying Circus", of which G. van Rossum is a fan. The first public version of the language was published in 1991. The latest version of Python is version 3. Specifically, version 3.9. Python version 2 is now obsolete. If possible, avoid using it.

The Python Software Foundation is the association that organizes Python development and leads the community of developers and users.

III. *What can you do with python?*

- ↗ Web programming [Django, Pyramid, Flask...]
- ↗ GUI development [Gtk, Qt, Tcl/Tk, WxWidgets...]
- ↗ Scientific computing [numpy, scipy, sage...]
- ↗ Graphical representation [gnuplot, matplotlib, VTK...]
- ↗ Databases [MySQL, PostgreSQL, Oracle....]
- ↗ Software development [Buildbot, Trac, Roundup...] ↗ Education
[Rashberry pi, Turtle, Robotic...]

IV. *Python features:* Python features include :

- ★ **Easy to learn:** Python has few key words, a simple structure and clearly defined syntax. This allows students to pick up the language quickly.
- ★ **Easy to read:** Python code is more clearly defined and visible to the eye.
- ★ **Easy to maintain:** Python's source code is easy to maintain.
- ★ **A large standard library:** Most of the Python library is highly portable and cross-platform, compatible with UNIX, Windows and Macintosh.
- ★ **Interactive mode:** Python supports an interactive mode for interactive testing and debugging of code snippets.
- ★ **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- ★ **Extensible:** You can add low-level modules to the Python interpreter. These modules allow programmers to add 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 numerous system calls, libraries and Windows systems, such as Windows MFC, Macintosh and Unix System X Windows.
- ★ **Scalable:** Python offers better structure and support for large programs than Shell scripts.

Apart from the features mentioned above, Python has a great list of good features, some of which are listed below:

- It supports functional and structured programming methods, as well as OOP.
- It can be used as a scripting language or compiled into bite-code for building large-scale applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It handles automatic waste collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA and Java.

V. Adoption of Python:

Many companies and organizations mention on their official websites that they use Python:

- Google (Guido van Rossum worked for this company between 2005 and 2012);
- Industrial Light & Magic;
- NASA;
- And CCP Games, creators of the video game EVE Online.

Python is also the command language for a large number of free software programs:

- ☞ Blender, 3D modeling and video editing software.
- ☞ Inkscape, vector-drawing software.
- ☞ Libre Office and Apache Open Office, the two development arms of an office suite derived from Star Office.
- ☞ Portage, the package manager for the Gentoo operating system.
- ☞ ParaView, digital data visualization software.
- ☞ Kodi, a multimedia player.
- ☞ QGIS mapping software.
- ☞ Weblate, a translation tool.
- ☞ Gedit, a text editor (plugins are written in Python).
- ☞ Sage Math, formal calculation software.

Moreover, commercial:

- ◆ Wing IDE, an integrated development environment specialized in Python, and written in Python.
- ◆ Corel Paint Shop Pro, image processing and graphics editing software.
- ◆ Capella, music notation software.
- ◆ Arc GIS mapping software.

VI. *Different versions of Python :*

When the Python Software Foundation was set up in 2001, and in the years since, the Python language has gone through a series of versions that have come to be known as Python 2.x (2.3, 2.5, 2.6...).

When a programming language is updated, developers are careful not to remove or modify old features too much. The idea is that a program that works with a certain version will always work with the latest version. However, the Python Software Foundation, observing a good number of obsolete features, implemented several times... decided to clean up the whole project. A program that runs perfectly under Python 2.x will therefore need to be updated a little to run again under Python 3. It is more readable and more enjoyable than Python 2; it is better constructed and avoids the inconsistencies associated with Python 2.

For example:

- ✓ Print has become a function.
- ✓ Text management is simplified.
- ✓ A number of iterations no longer return lists (which was not very logical).
- ✓ The comparison operators have been cleaned up (so that python is stricter in its comparisons).
- ✓ There is now only one type of integer.
- ✓ Basic python libraries have been renamed or reorganized to make naming more consistent.

Therefore, there is no longer any reason for you to prefer python 2. There is only one thing left for you to do: master python 3 even better. That is why we would like to advise you to download and install the latest version of Python later.

VII. *Implementing Python :*



In addition to the reference version, called CPython (because it is written in C), there are other systems implementing the Python language:

- ✧ Stackless Python, a version of CPython that does not use the call stack of the C language;
- ✧ Jython, a Python interpreter for Java virtual machines. It has access to the libraries provided with the Java ;
- ✧ IronPython, an experimental interpreter/compiler for the .Net / Mono ;
- ✧ Brython, a Python 3 implementation for web browsers
- ✧ MicroPython, a lightweight variant for microcontrollers ;
- ✧ PyPy is a Python interpreter written in a subset of Python compliant to C or LLVM ;
- ✧ an (experimental) compiler for Parrot, the Perl 6 virtual machine; Shed Skin, a compiler for a subset of Python producing C++ code;
- ✧ Unladen Swallow (en), an LLVM-based optimized version of CPython, now discontinued (the last version was released in October 2009);
- ✧ RustPython, Rust implementation project.
- ✧ PySpark is a programming interface for using Spark from the Python language to perform distributed calculations.

These other versions do not necessarily benefit from the entire library of functions written in C for the reference version, nor from the latest language evolutions.

VIII. *Python environments :*

↳ *What is an integrated development environment?*

Commonly known as IDE (Integrated Development Environment).

Development software enables the developer to write, execute and debug code with relative ease. While the ability to write, execute and debug source code is the most fundamental functionality of an IDE, it is not the only one. It is safe to say that all IDEs perform the fundamental tasks in the same way; however, most modern IDEs come with a plethora of other features specifically to facilitate the workflow for a particular type of development pipeline.

In this chapter, we will concentrate on development environments for Python, i.e. those that support or are dedicated to Python programming.

IDEs are generally developed by a community of people (open source) or by a commercial entity.

Each IDE has its own strengths and weaknesses. Some IDEs, such as Jupyter or Spyder, are open source and have been developed for the scientific and artificial research community. These IDEs have additional features that make it quick and easy to prototype machine learning models and scientific simulations. However, they are not well equipped to support the end-to-end application development process.

👉 ***The 10 best development environments for Python***

1. **PyCharm** is a cross-platform integrated development environment specifically developed for Python by Czech company [JetBrains](#) . It is one of the best development environments for Python. There are two main versions of the software available for download - Professional Edition and Community Edition. The Professional Edition has additional features for development, which are not available in the Community Edition and must be purchased. The Community Edition is published under the Apache license and is a free-to-use open source IDE that is identical to the Professional Edition in many respects; however, it lacks the additional features.
2. **Spyder** is an open-source Python scientific development environment supplied with Anaconda. Spyder has several features that are developed to aid scientific and data-driven development and is therefore an ideal IDE for data scientists. It is written in Python itself with the PyQt5 library and therefore offers additional features mentioned below.
3. **Jupyter Notebook** is one of the most widely used editors in the data science industry. It makes the most of the fact that Python is an interpreted language, meaning that lines of Python code can be executed one line at a time, and the whole thing doesn't need to be compiled together like C / C ++. This makes the editor ideal for writing and prototyping machine learning models. Since there's a significant amount of pre-processing done at the outset, and after that there's a repeated process of hyper-parameter tuning and model prototyping, the ability to

run one cell (a group of lines) together at a time gives data scientists the ability to tune their models easily.

4. **Atom** first started out as a lightweight, cross-platform, open source code editor based on Node.js developed by GitHub. It is popularly known as "the lendable text editor for the 21st century" by its developers. Atom is based on Electron, which is a framework that enables a cross-platform desktop application, using Chromium and Node.js and is written in CoffeeScript and Less.
5. **Canopy is** an IDE developed and maintained by Enthought specifically with scientists and engineers in mind. It contains integrated tools for iterative data analysis, visualization and Python application development. It offers two specialized versions of Canopy: Canopy Enterprise and Canopy Geoscience.
6. **Microsoft Visual Studio IDE is one of** the most popular IDEs in the development industry. It was originally designed for C/C++ development. However, with the growing popularity and adoption of Python in the industry, Microsoft decided to add support for Python development via an open-source extension called Python Tools. This brought together the best of both worlds in an integrated environment. Visual Studio's superior development-centric features are the best. With all the features bundled together, it is almost neck and neck with PyCharm.
7. **Sublime Text:** Similar to Atom, Sublime Text is more of a code editor than an IDE. However, because of its support for various packages, it contains enough functionality to be considered for a complete end-to-end development environment. Its language support is not limited to one or two programming languages. In turn, it supports almost every language used in the industry. It has syntax highlighting and automatic code completion for almost every language, making it quite versatile and flexible. Sublime text has a free trial and advertises as paid. It is a cross platform editor, supporting a single license key on all platforms.

8. **Eclipse** is one of the best open source IDE suites available for Java development. It supports numerous extensions. One such open-source extension is PyDev, which transforms Eclipse into a powerful Python IDE.
9. **Wing is one of** the best cross-platform development environments for Python, with the necessary features and decent development support. It is free for personal use, but there is a charge for the pro version, which is intended for commercial use. The pro version comes with a 30-day trial so developers can try it out. It even has a specialized version called Wing 101, which is aimed at beginners and is a toned-down version that makes it easier for beginners to get started.
10. **Rodeo** is a cross-platform open source IDE developed by yhat. It is designed and developed for data science, and features many of the tools needed for data analysis and visualization.

In this chapter, we have looked at 10 of the best development environments for Python. To be honest, there is no clear-cut answer to which IDE is best, as most of them are specifically designed for a given group of developers or scientists. Therefore, we will choose the most preferred IDE for each type of use case.

IX. Python distributions:

Different distributions are available, sometimes including many packages dedicated to a given domain:

- **Active-Python:** available as a free version (not suitable for "production use") or a commercial version.
- **Python(x, y):** Python distribution for scientists based on Qt and Eclipse.
 - Obsolete, replaced by Win-Python
- **Enthought Canopy:** distribution for scientific use, available in free (Canopy Express) or commercial versions.
- **Anaconda:** distribution for scientific use, available in free or commercial versions.
- **Intel Distribution for Python:** Anaconda-based distribution, featuring Intel's MKL library to accelerate numerical computations in libraries such

as NumPy and SciPy. It is available free on its own, or integrated with Intel Parallel Studio, which requires a paying license.

- **Pyzo**: "Python to the people", designed to be easy to use.
- **Win-Python**: distribution for scientific use with Spyder, QT, etc....

These are not different implementations of the Python language: they are based on CPython, but come with a number of pre-installed libraries.

X. List of Python packages for data science:

Python has gained real value thanks to the fact that it is both a general-purpose language and has Python packages specifically adapted to data science.

Whether for data science, statistics, data analysis or data visualization, Python offers an interesting alternative to R.

Here we present a list of Python packages that are extremely useful for data scientists wishing to use Python.

- ☞ **Jupyter Notebook**: A more intuitive interface. An important foundation for your Python projects. It lets you work interactively, especially on graphics, directly in a browser.
- ☞ **NumPy**: Matrix manipulation with Python. This is one of the most important Python packages for matrix manipulation. It lets you manage arrays, import and export them and work with them. Most of the code used to code the package is C, which ensures good performance.
- ☞ **SciPy**: Reference for scientific computing. This project aims to create a complete Python environment for scientific computing. It is based on NumPy for table management. It includes functions for linear algebra, statistics, image processing, etc. It includes one of the most widely recognized Python packages for visualization: matplotlib, which we will talk about below.
- ☞ **Pandas**: Python data analysis library. Pandas brings to Python the ability to manipulate large volumes of structured data in a simple and intuitive way. We are getting closer to R with the notion of DataFrame.
- ☞ **Statsmodels**: Classic statistics with Python. This package includes more classical statistical models such as the linear model, categorical data analysis...

- ☞ **Scikit-learn:** Machine learning tool. This is one of the most widely used Python packages for machine learning. It includes numerous methods for classification, cross-validation, clustering and model selection.
- ☞ **Matplotlib:** The most complete, but also the most complex. This Python package is extremely powerful and complete for data visualization, with numerous customization options.
- ☞ **Bokeh:** Beautiful graphics for the Web. This is one of many Python packages for data visualization. It focuses on the creation of interactive web graphics and web data visualization applications.
- ☞ **Seaborn:** improved matplotlib. This Python package is based on matplotlib, but adds a few new graphics and, above all, simplifies the creation of high-quality graphics.
- ☞ **Keras:** This is the simplest Python package for deep learning.

XI. Language keywords:

The reserved keywords of the Python language are provided in the **keyword.kwlist** of the **keyword** module. **Python 2.7.5** keywords are: **and** , **as** , **assert** , **break** , **class** , **continue** , **def** , **del** , **elif** , **else** , **except** , **exec** , **finally** , **for** , **from** , **global** , **if** , **import** , **in** , **is** , **lambda** , **not** , **or** , **pass** , **print** , **raise** , **return** , **try** , **while** , **with** , **yield** .

As of **Python 3.0**, **print** and **exec** are no longer language keywords, but functions in the **builtins** module. **True**, **False**, **None** and **nonlocal** are added to the keywords. The first three were already present in previous versions, but are no longer modifiable (previously, the assignment **True** = 1 was possible). **nonlocal** was introduced by **PEP 3104**, and allows, in a function defined inside another function, to modify a variable of a higher level of scope. Previously, only variables local to the function and global (module level) could be modified. However, it was possible, and still is without the **nonlocal** keyword, to modify an object assigned to a variable of a higher scope level, e.g. a list with the **append** method - this is obviously impossible for an immutable object.

Chapter 2 : Python-Anaconda-Jupyter installation

I. Installing Python under Windows:

1. How to install Python on Windows?

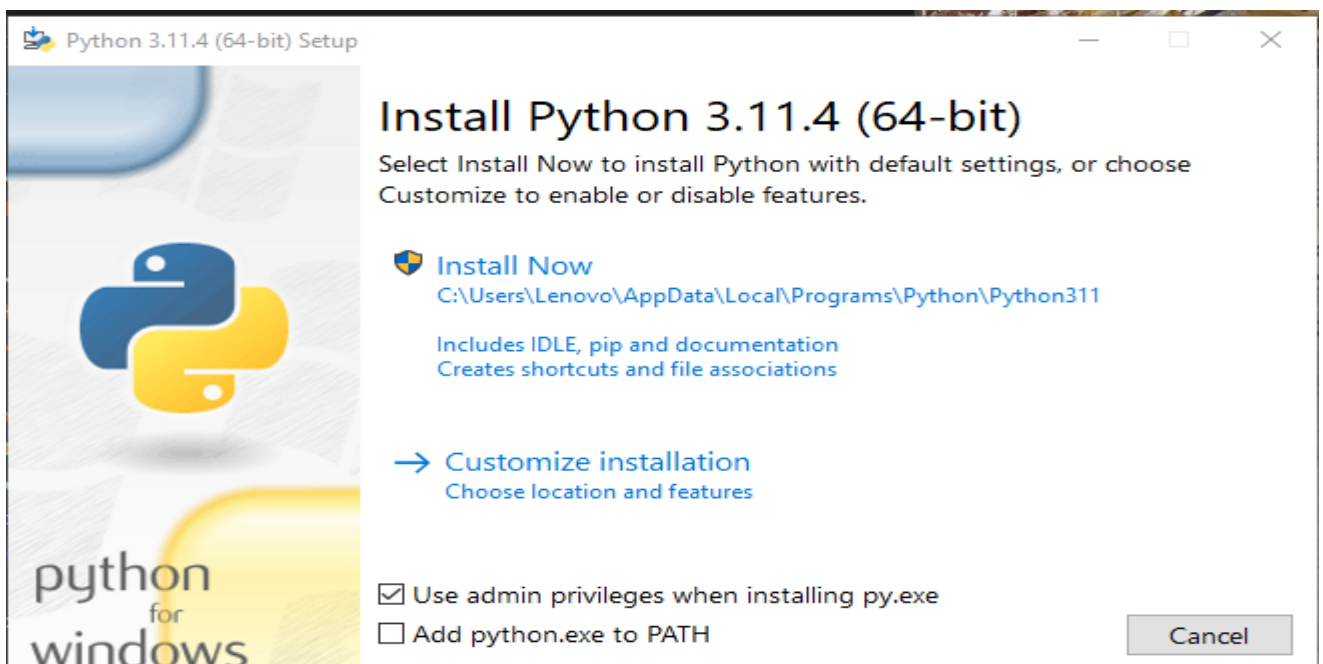
Step1:

To download and install Python, visit the official Python website <http://www.python.org/downloads/> and choose your version. We have chosen Python version 3.11.4



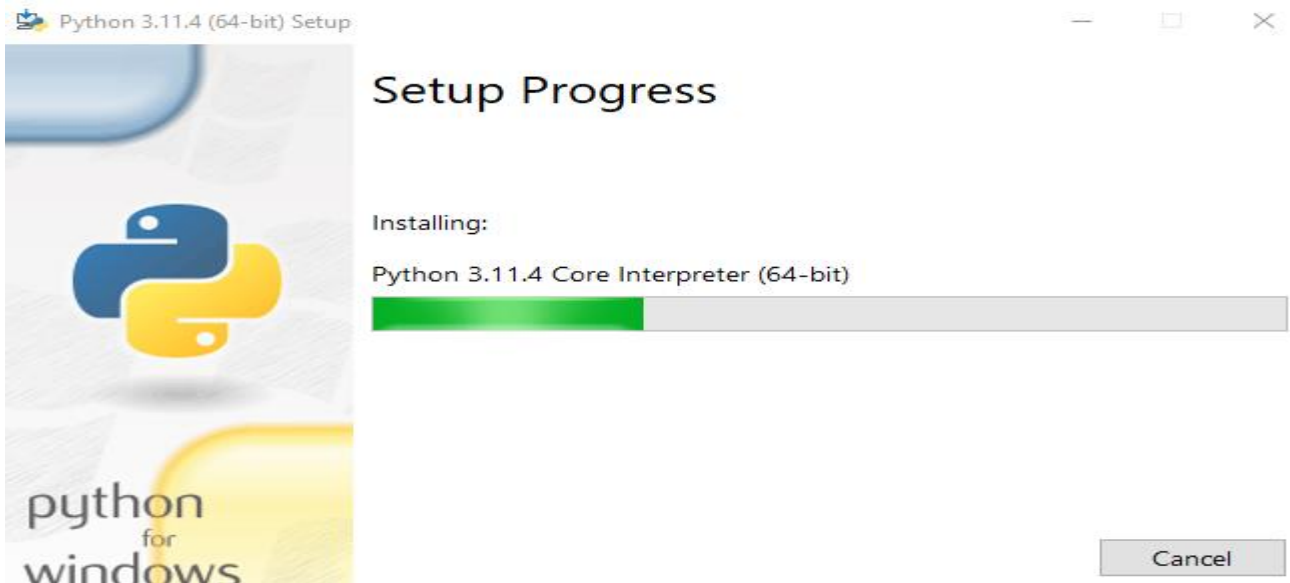
Step2:

Once the download is complete, run the exe file to install Python. Now click on "Install Now".



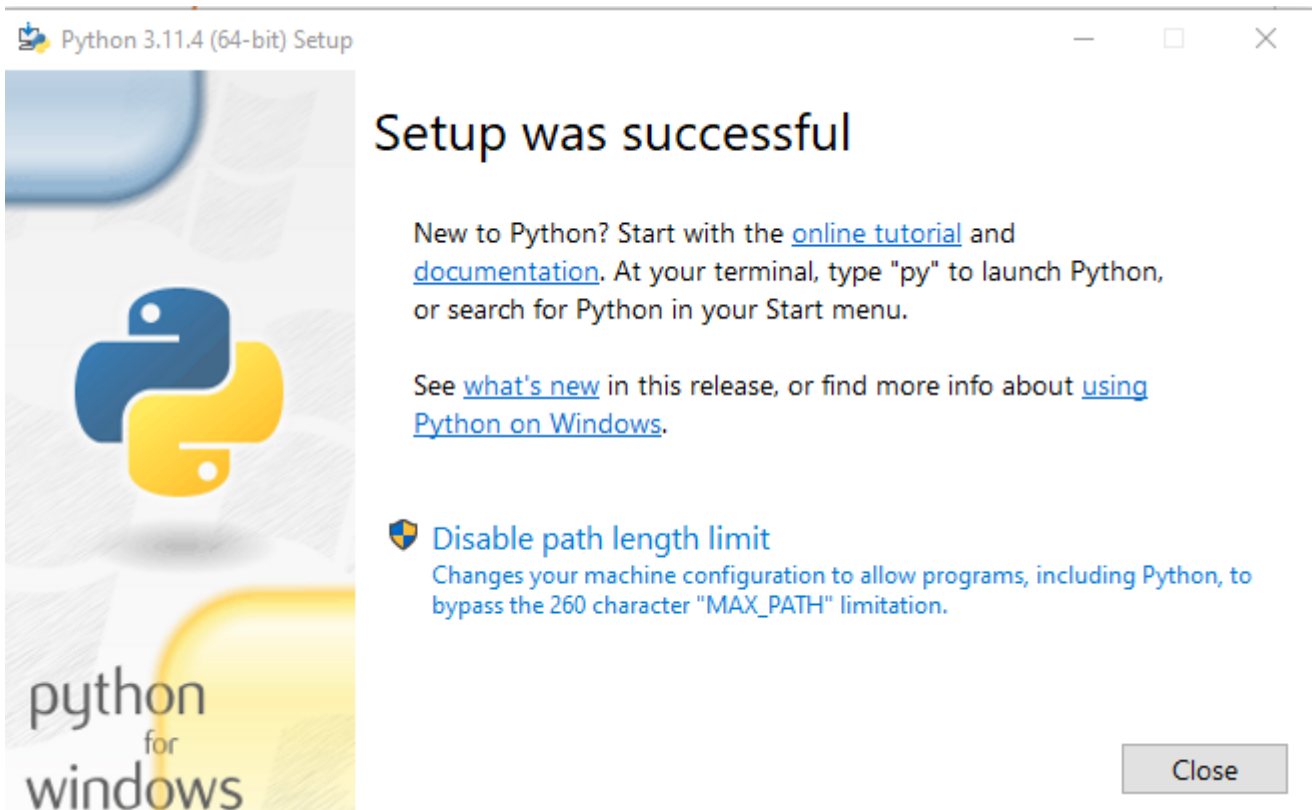
Step3:

You can see the Python installation at this point.

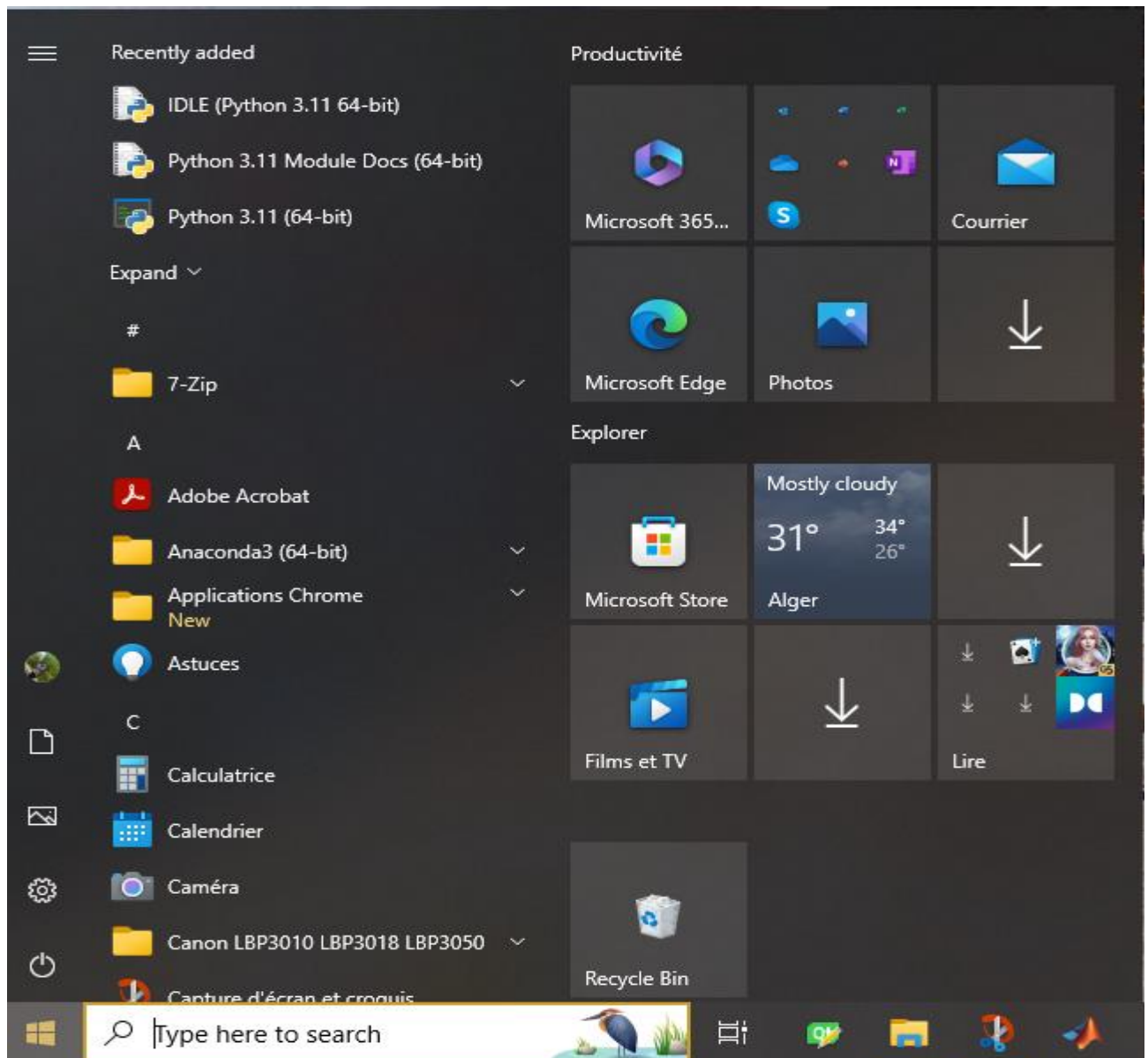


Step4:

Once installation is complete, you will see a window indicating that configuration has been successful. Now click on "Close".



Once installation is complete, go to Start > All Programs. Python should appear in this list (see next figure). We will soon see how to launch it, so do not be too impatient...



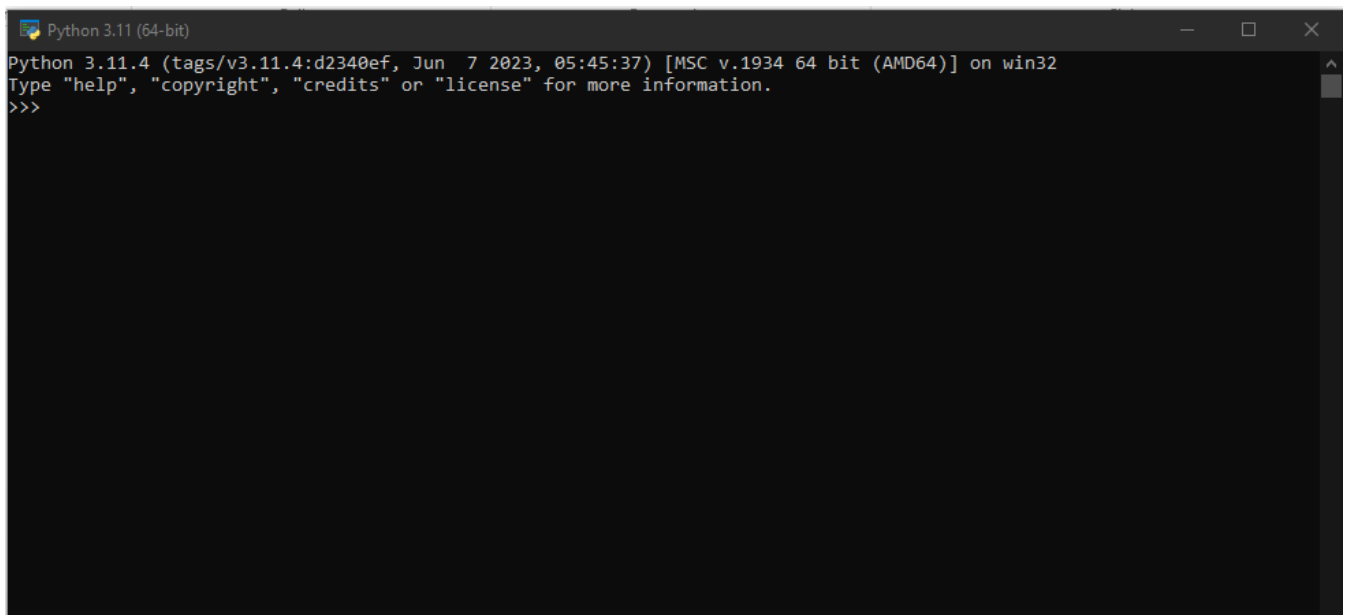
→ -Launching Python: *Phew! Now that is done!*

Well, in theory, we will start using Python in the next chapter, but to reward you for your exemplary installation, here are the different ways to access the Python command line.

There are several ways of accessing the Python command line, the most obvious being via the Start > All Programs > Python 3.11 > Python (Command Line) menus. If all goes well, you should get a beautiful console (see next figure). The information displayed in yours may not be the same, but do not worry. You can also use the Windows command line; because of the shortcuts, I generally prefer this method, but it is a matter of taste.

Go to the Start menu, and then click on Run. In the window that appears, simply type "python" and the Python command line should reappear. You can also go directly to run by typing the shortcut Windows + R.

To close the Python command interpreter, type "exit ()" and press Enter.

A screenshot of a Windows command prompt window titled "Python 3.11 (64-bit)". The window has a dark background and a light-colored text area. The text inside the window reads: "Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32", followed by "Type 'help', 'copyright', 'credits' or 'license' for more information." and a prompt ">>>". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

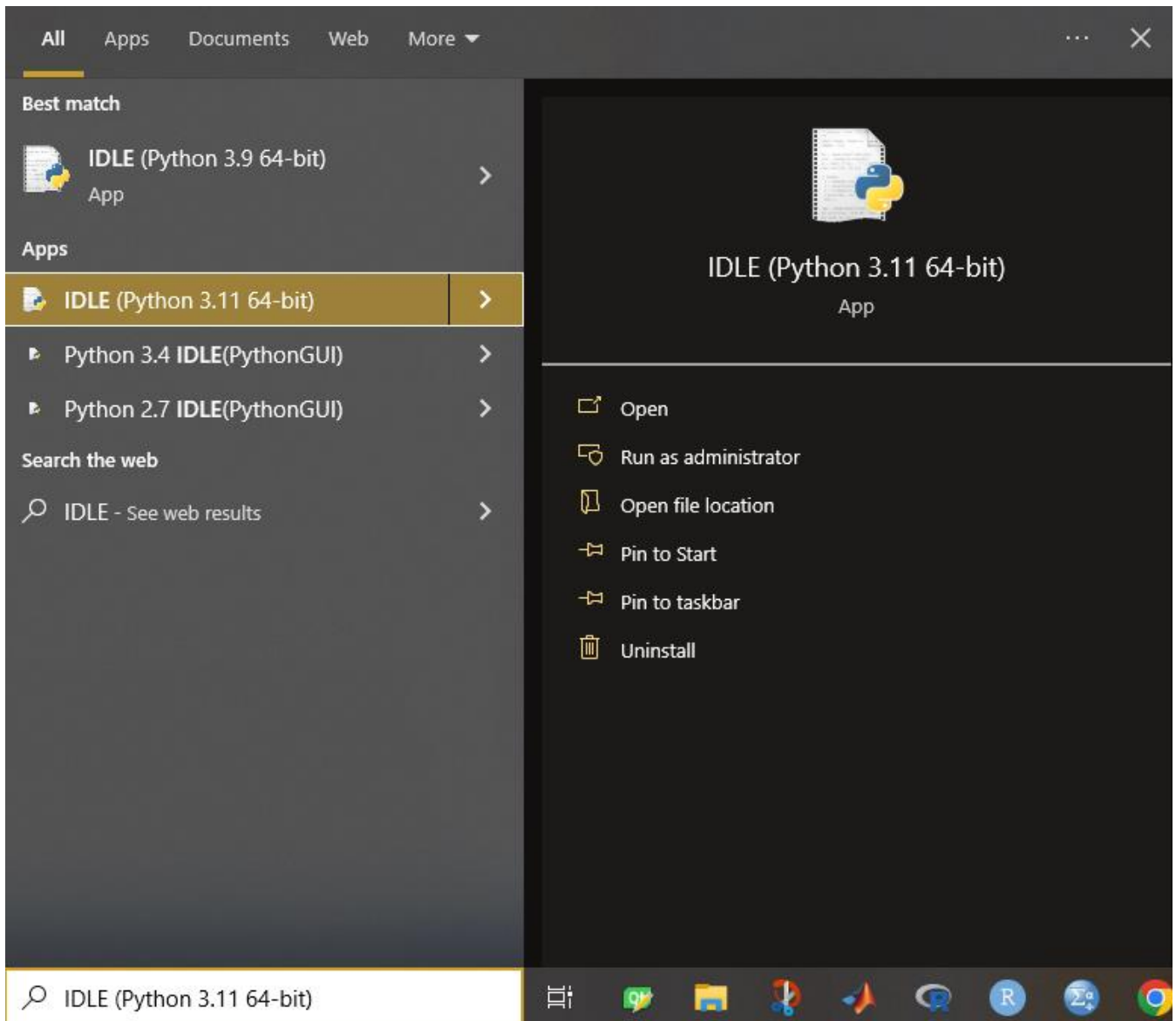
```
Python 3.11 (64-bit)
Python 3.11.4 (tags/v3.11.4:d2340ef, Jun 7 2023, 05:45:37) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

2. How do I install packages?

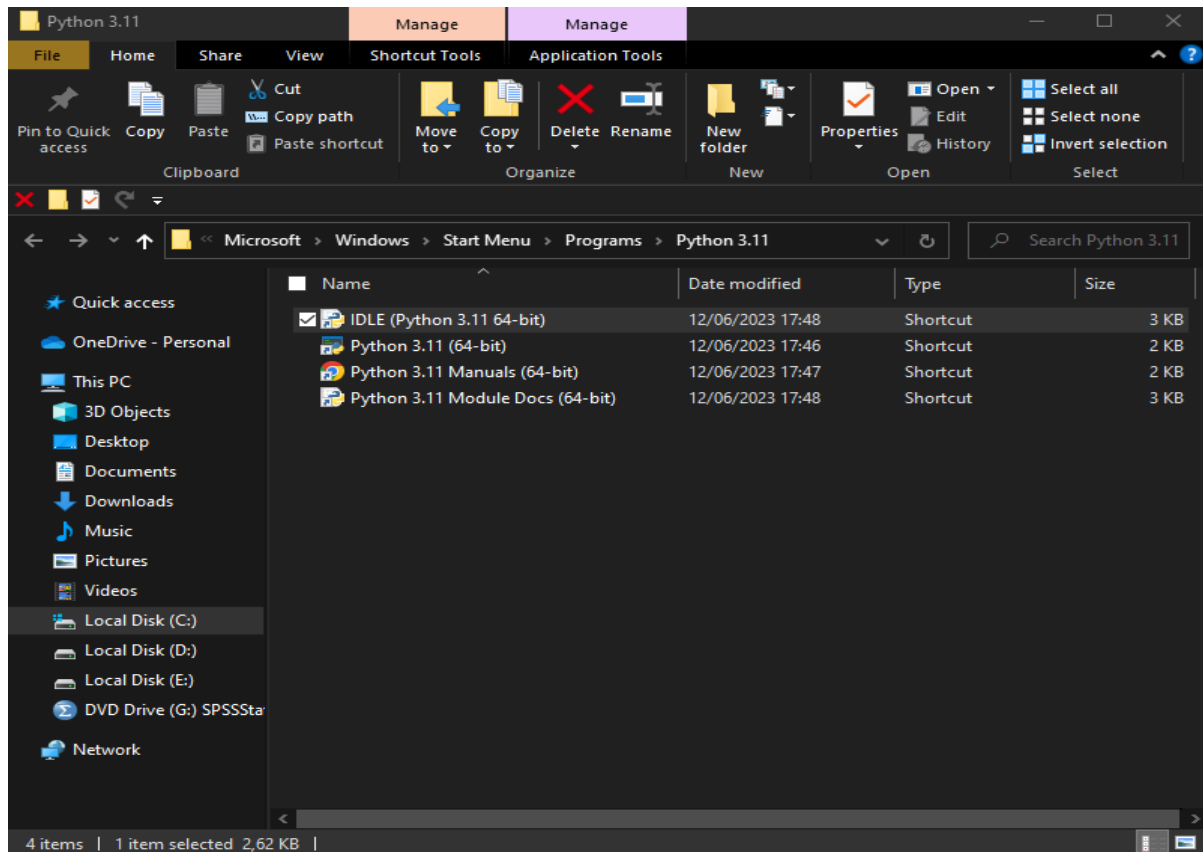
In this part of the chapter, we will explain package installation with an example: **numpy** package installation with **pip** (**pip** is a package management system used to install and manage libraries written in Python).

For more details, follow these steps:

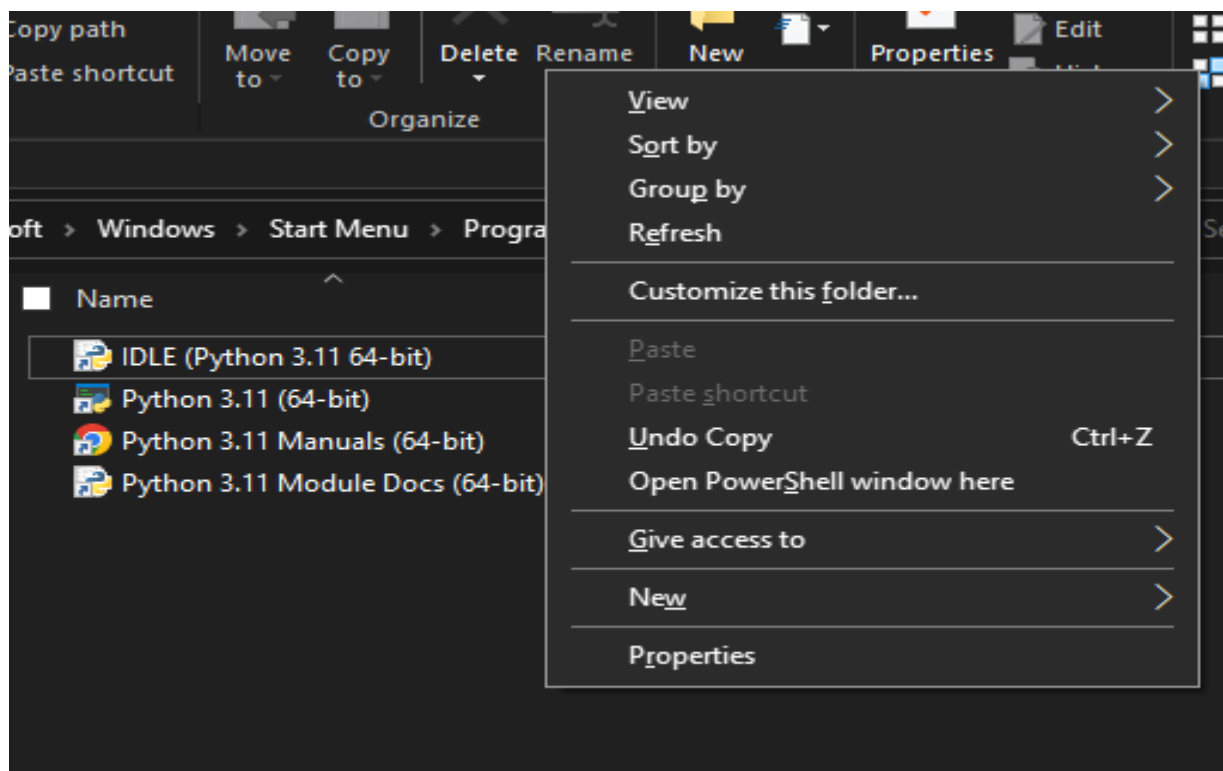
- 1- Type the word "**IDLE**" in the toolbar (bottom) in the search box, and you'll get :



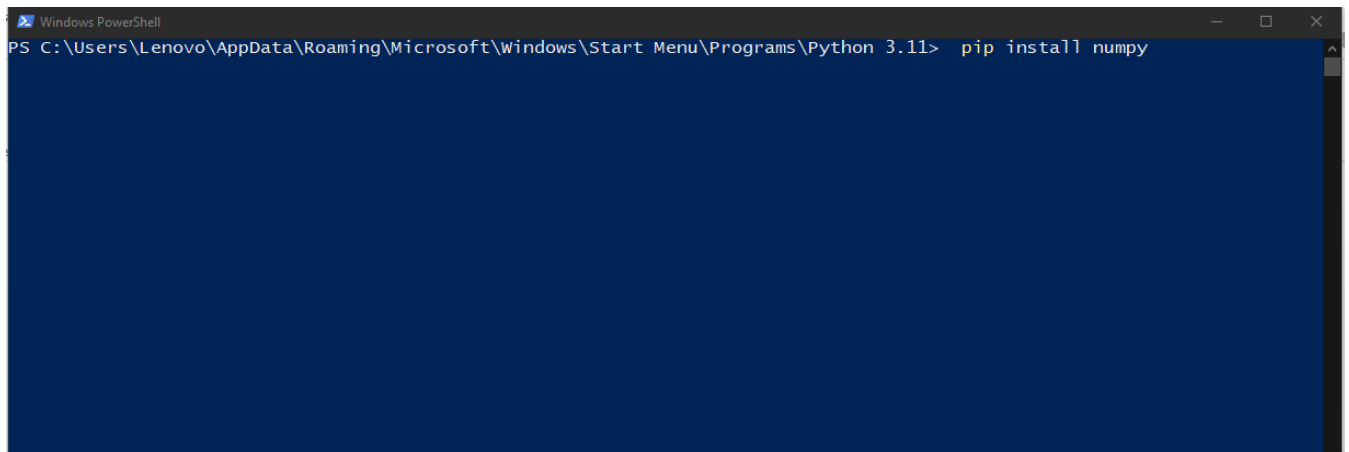
- 2- Click on : "**Open file location**"



3- Click on : **Shift (keyboard) + right button**, you will get this list:

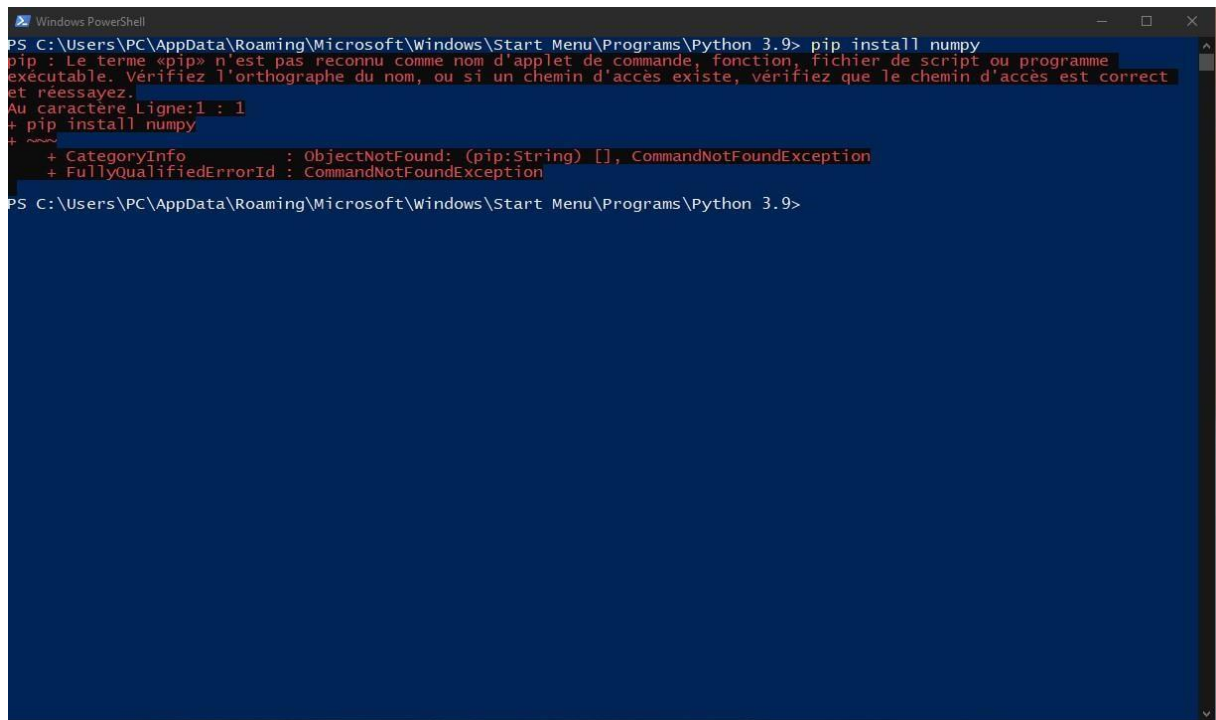


4- Select "**open PowerShell window here**", and write "**pip install numpy**", you will get this :



```
Windows PowerShell
PS C:\Users\Lenovo\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.11> pip install numpy
```

5- If you get this error

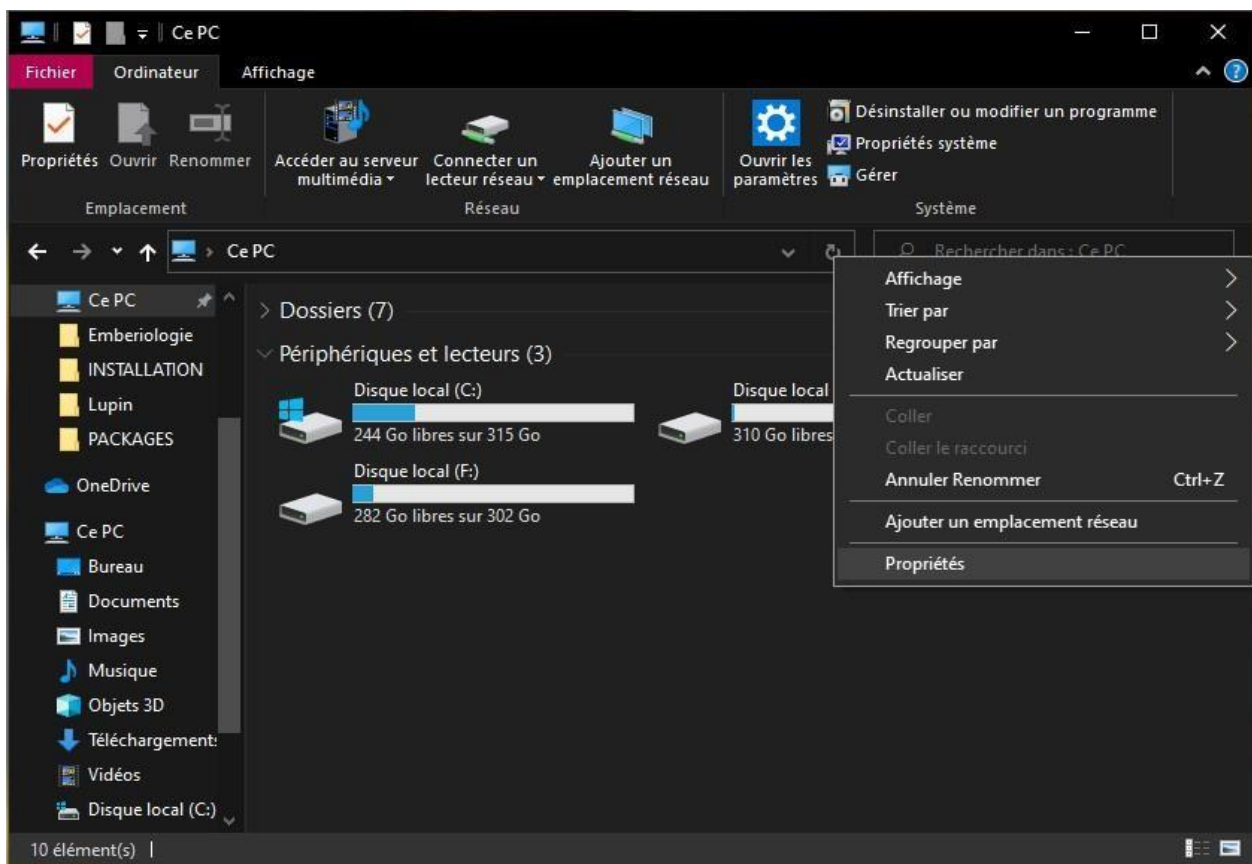


```
Windows PowerShell
PS C:\Users\PC\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.9> pip install numpy
pip : Le terme «pip» n'est pas reconnu comme nom d'applet de commande, fonction, fichier de script ou programme
exécutable. Vérifiez l'orthographe du nom, ou si un chemin d'accès existe, vérifiez que le chemin d'accès est correct
et réessayez.
Au caractère Ligne:1 : 1
+ ~~~~~
+ pip install numpy
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (pip:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\PC\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.9>
```

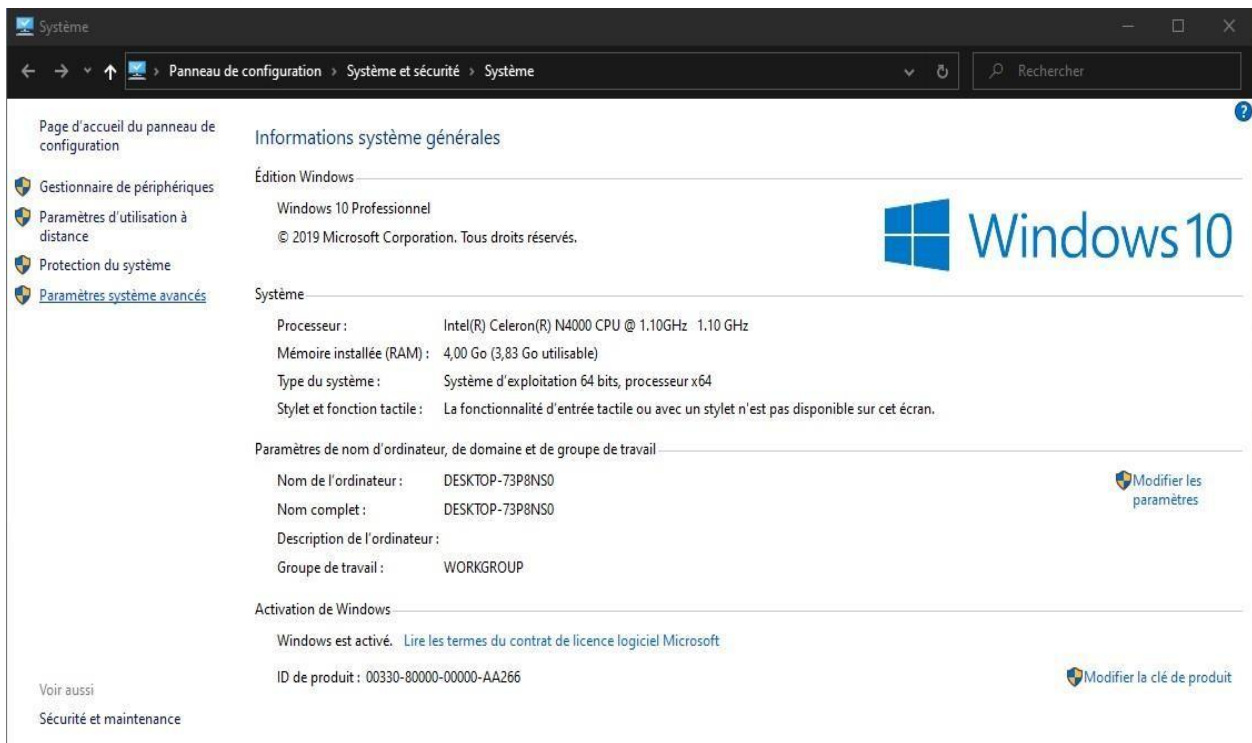
Follow the steps below.

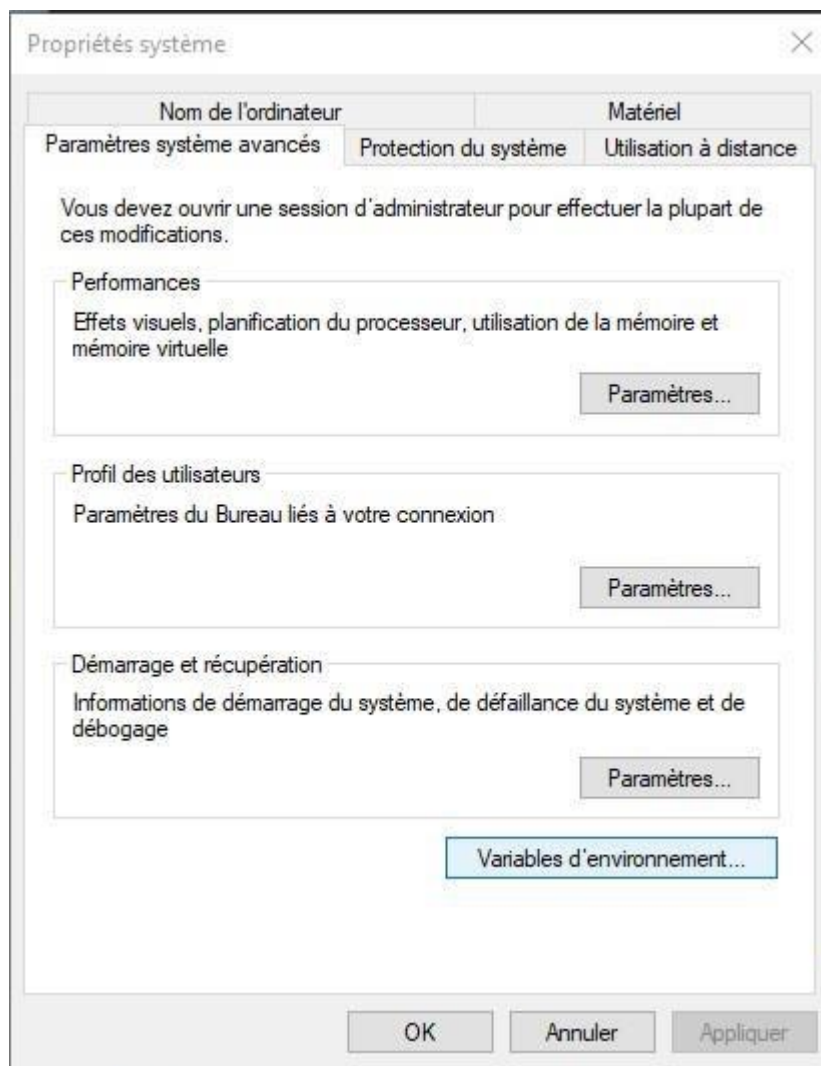
1. Go to "**This PC**", right-click and select "**Properties**".



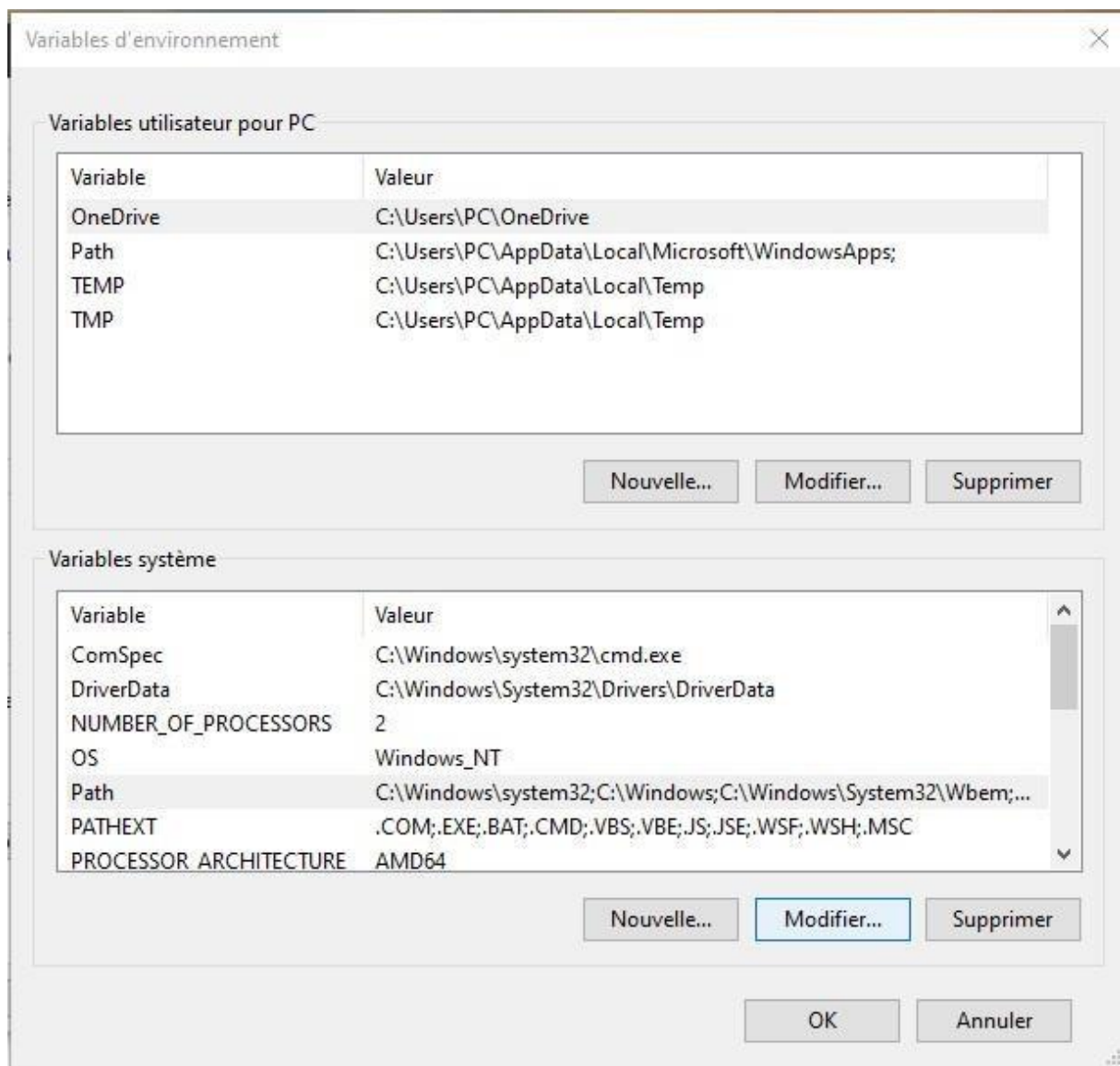
2. Select "**Advanced system settings**"

3. You'll get this, selected "**Environment variables**":

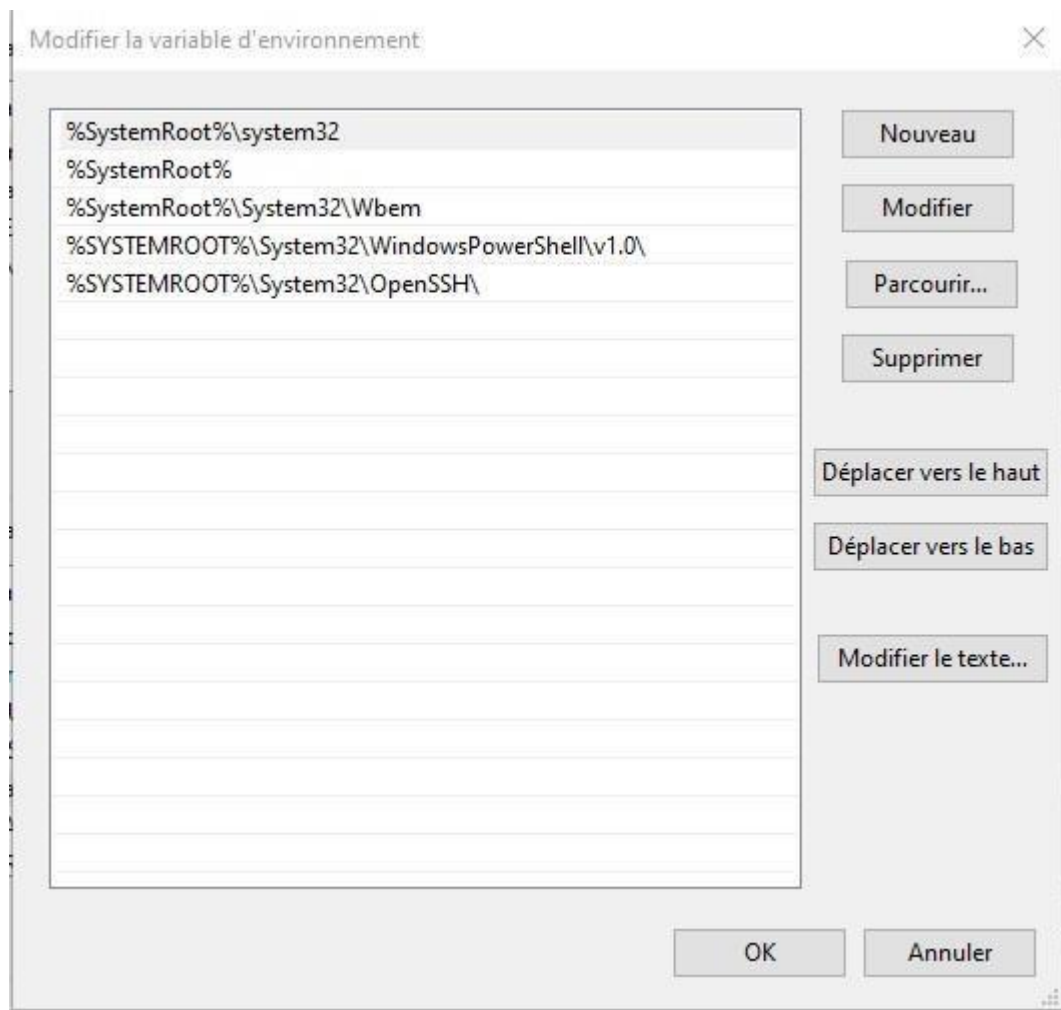




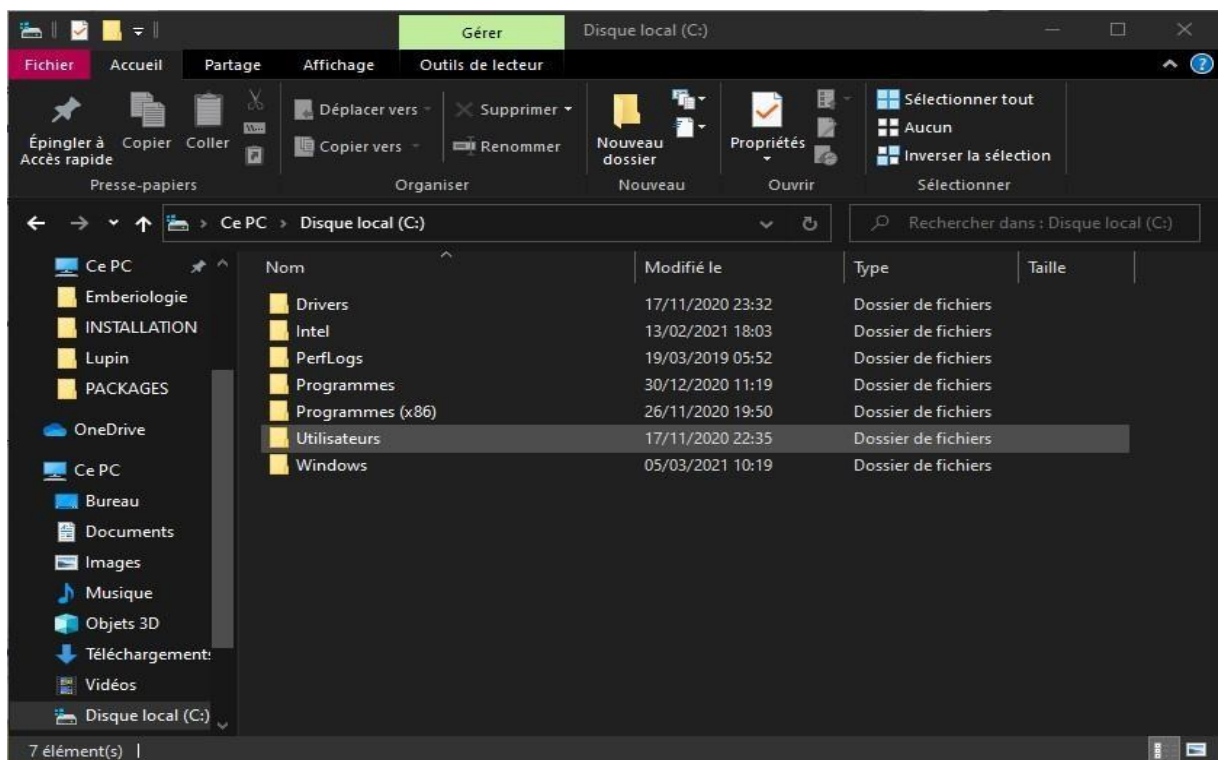
4. You will get this, select "**Path**", then "**Modify**":



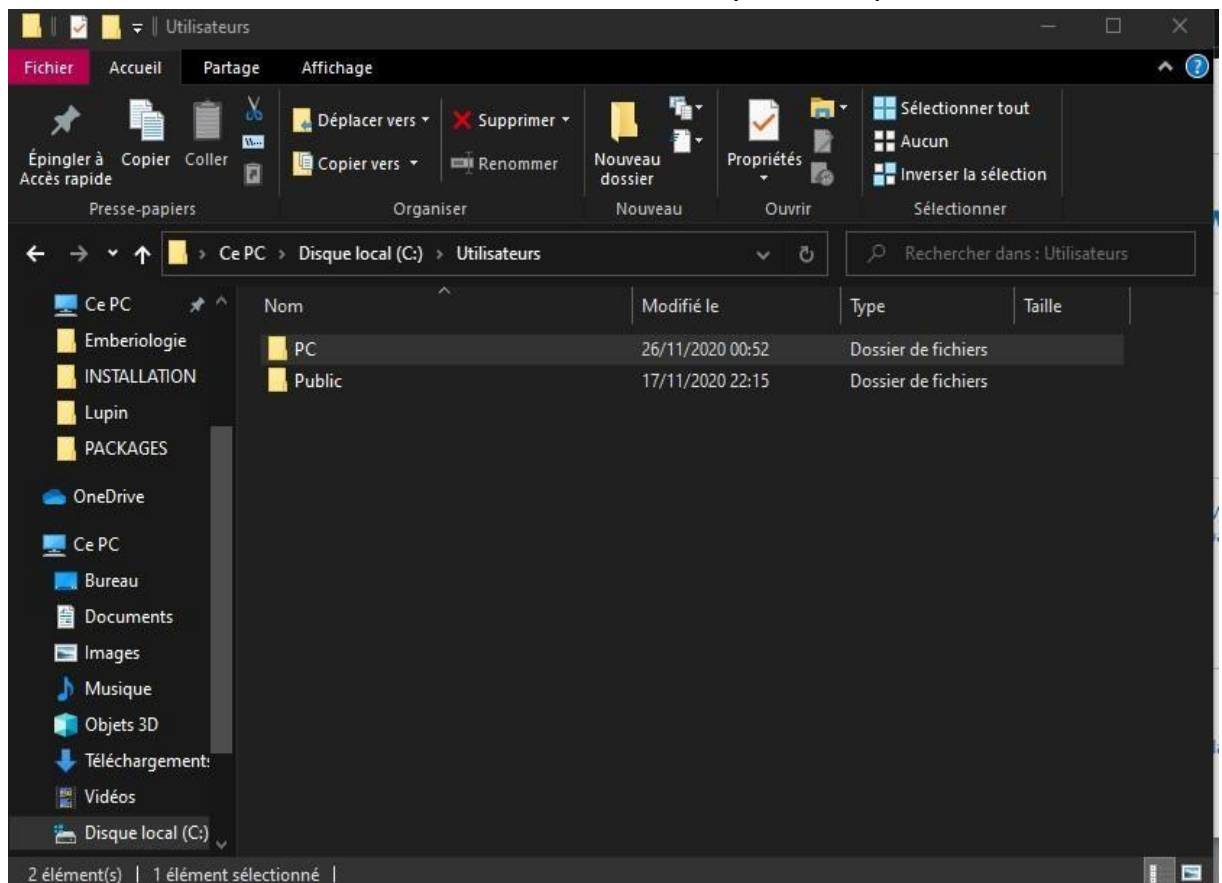
5. You will get this window, so leave it open and follow the rest of the steps.



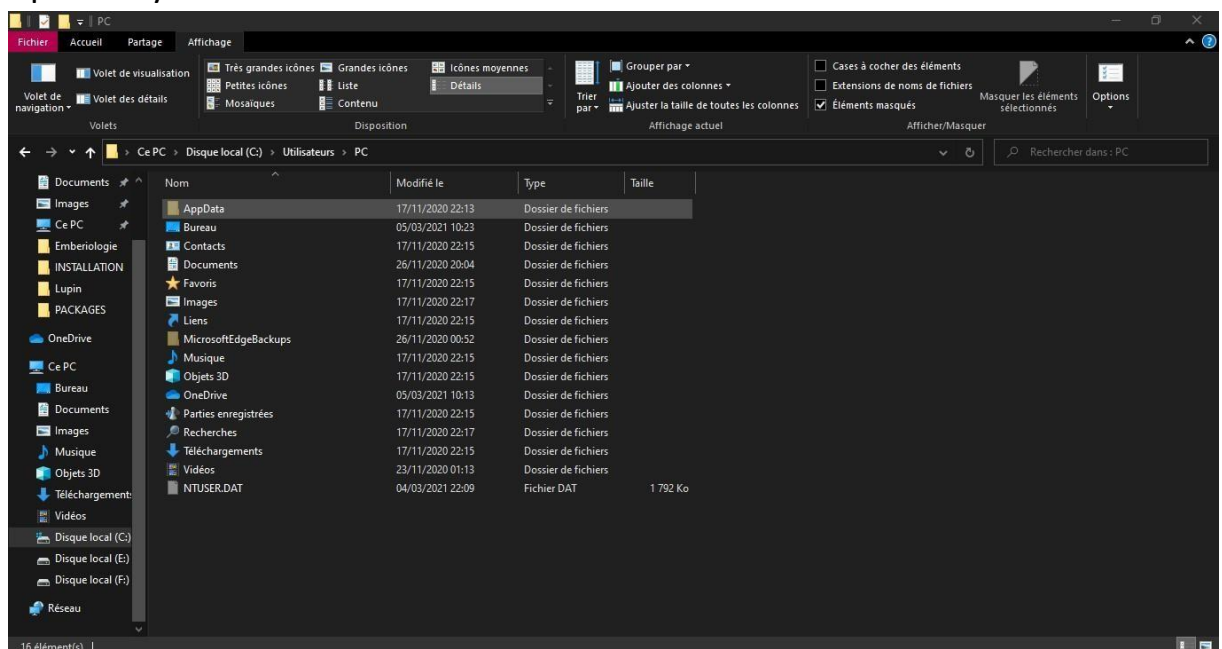
6. Open File Explorer, go to "**This PC**" and select the **local disk (C :)**, then select "**Users**".



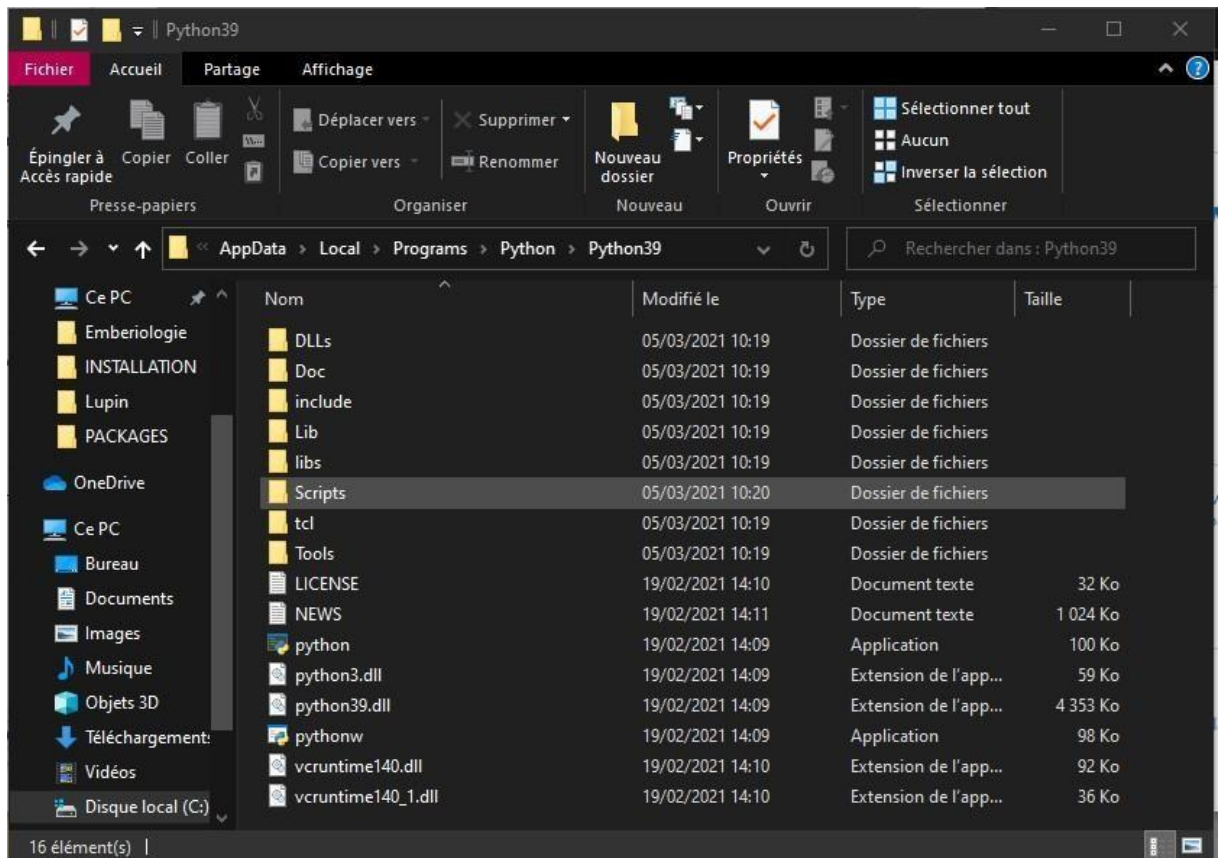
7. Select the "**PC**" folder or the folder named after your computer



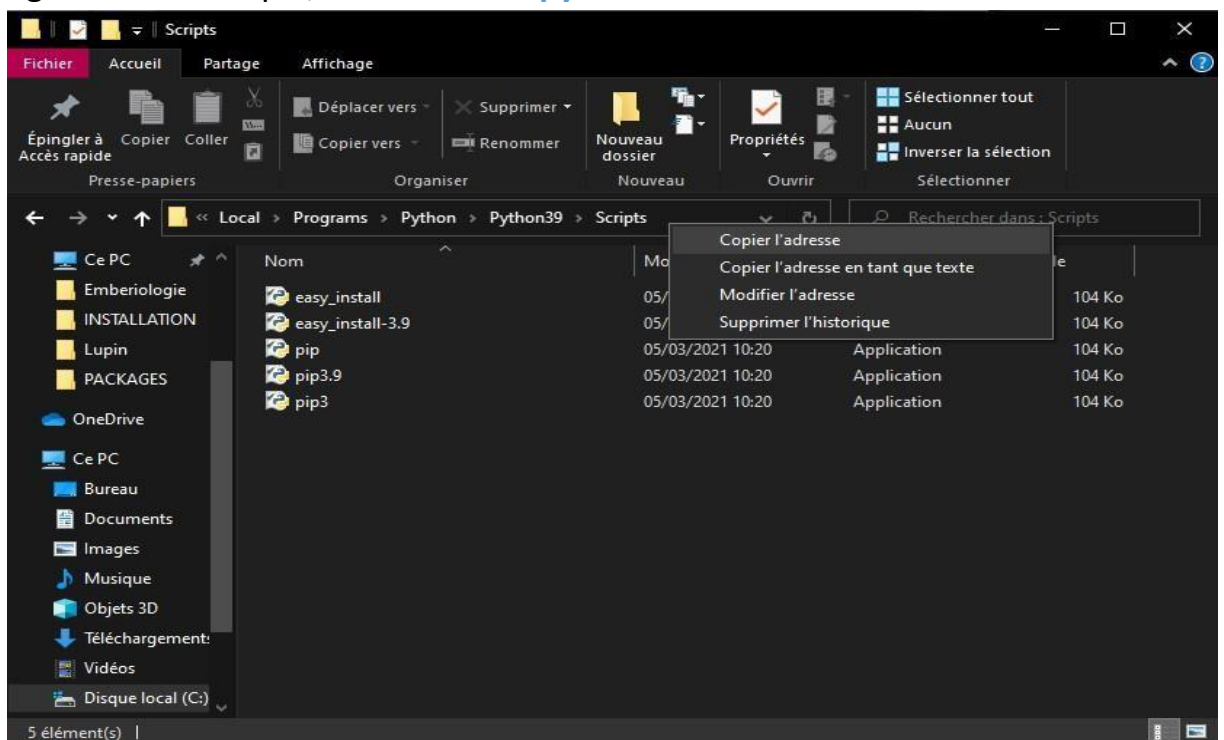
8. Select "**App Data**", if you have not found this folder go to "**View**" at the top and layer "**hidden items**".



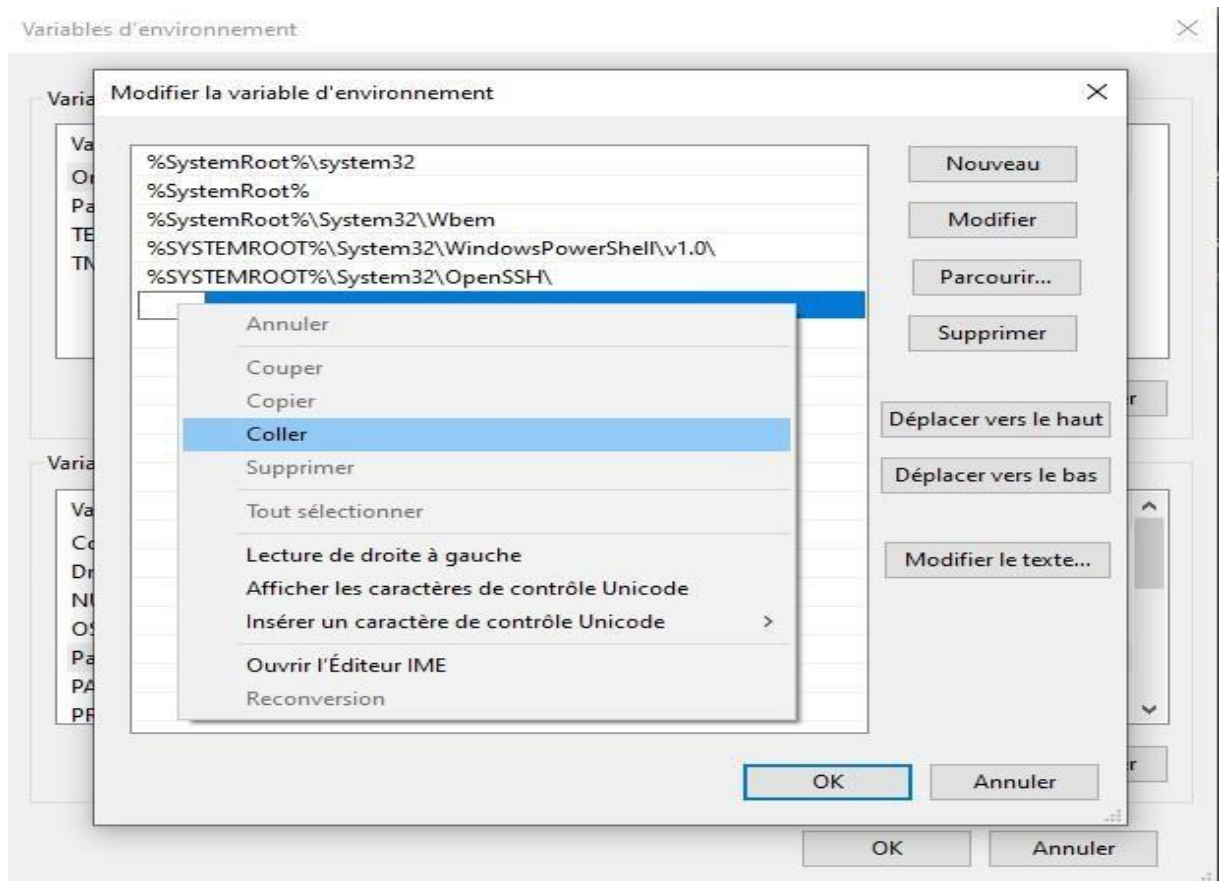
9. Select "**Local**", then "**programs**", then "**python**", then "**python39**" (this is the version of Python you have installed), then "**scripts**".



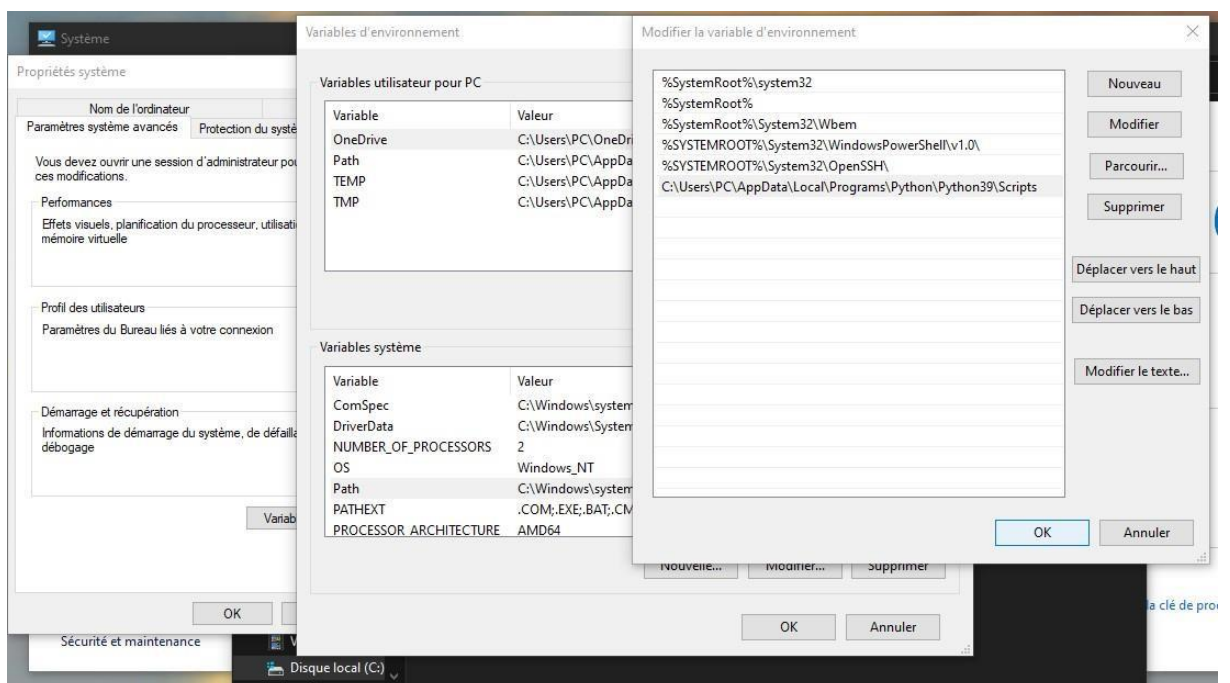
10. Right-click on scripts, and select "copy addresses".



11. Return to "Edit environment variables", select "New", then right-click and choose "Paste".



12. Click "OK" on all windows (there are 3 windows)



13. Go back to the first step and install **numpy** again

```
Windows PowerShell
PS C:\Users\PC\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.9> pip install numpy
Collecting numpy
  Downloading numpy-1.20.1-cp39-cp39-win_amd64.whl (13.7 MB)
    | 3.4 MB 501 kB/s eta 0:00:21
```

14. After installing **numpy** write **python -m pip install --upgrade pip**

```
Windows PowerShell
PS C:\Users\PC\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.9> pip install numpy
Collecting numpy
  Downloading numpy-1.20.1-cp39-cp39-win_amd64.whl (13.7 MB)
    | 13.7 MB 547 kB/s
Installing collected packages: numpy
Successfully installed numpy-1.20.1
WARNING: You are using pip version 20.2.3; however, version 21.0.1 is available.
You should consider upgrading via the 'c:\users\pc\appdata\local\programs\python\python39\python.exe -m pip install --up
grade pip' command.
PS C:\Users\PC\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Python 3.9> python -m pip install --upgrade pip
```

15. You can now use this package.

II. Installing Anaconda and Jupyter Notebook:

To install Anaconda, see this site:

<https://www.stat4decision.com/fr/installer-anaconda-data-python/> You

can also launch Jupyter via Anaconda Navigator. For more details on how to use Jupyter Notebook, visit this site:

https://python.sdv.univ-paris-diderot.fr/18_jupyter/

Chapter 3: Basics

I. Module and import :

1. Definition:

Modules are Python programs containing functions that will be reused frequently (also known as libraries). They are "toolboxes" that will come in very handy. Python developers have created numerous modules that perform a phenomenal number of tasks. For this reason, it is always a good idea to check whether some of the code you want to write already exists in the form of a module. Most of these modules are already installed in standard versions of Python.

2. Module import

: Example:

Let us take a closer look at this example:

- Line 1, the import instruction gives access to all **random** module functions.
- Then, on line 2, we use the **randint (0, 10)** function in the **random** module. This function returns a randomly drawn integer between 0 and 10.

```
Entrée [2]: import random
            random . randint (0, 10)

Out[2]: 1
```

3. To get help on imported modules: To get help on a module, simply use the **help ()** command, which returns something like :

```
Entrée [4]: import random
            help (random)
```

Help on module random:

NAME

random - Random variable generators.

MODULE REFERENCE

<https://docs.python.org/3.8/library/random>

The following documentation is automatically generated from the Python source files. It may be incomplete, incorrect or include features that are considered implementation detail and may vary between Python implementations. When in doubt, consult the module reference at the location listed above.

DESCRIPTION

integers

uniform within range

4. Some common modules

There are a number of modules you are likely to use if you are programming in Python. Here is a non-exhaustive list. For the complete list, see the modules 4 page on the Python website:

- **math:** basic mathematical functions and constants (sin, cos, exp...).
- **sys:** interaction with the Python interpreter, passing arguments
- **os:** dialog with the operating system
- **random:** generation of random numbers.
- **time:** access to computer time and time management functions.
- **urllib:** retrieve data from the Internet using Python.
- **Tkinter:** python interface with Tk. Creating graphical objects
- **re:** regular expression management

Finally, note that there are many other external modules that are not installed as standard in Python, but are widely used in bioinformatics and data analysis. These include NumPy (manipulating vectors and matrices, linear algebra), Biopython (searching biological databases, manipulating sequences or biomolecule structures), matplotlib (graphical representations: curves, point clouds, bar charts...), pandas (data analysis). These modules will be introduced in Chapter 17, some modules of interest in bioinformatics.

II. Comments :

In a script, anything following the **# character** is ignored by Python until the end of the line and is considered a comment. Comments should explain your code in human language.

Here is an example:

```
Entrée [5]: # Votre premier commentaire en Python .  
            print (' Hello world !')
```

Hello world !

///. Display "print () function

The **print ()** function displays the argument passed between brackets:

```
Entrée [6]: print (' Hello world !')
```

Hello world !

Another way of seeing this is to use two print () functions in succession. In the following code, the ";" character is used to separate several Python instructions on the same line:

```
Entrée [8]: print (" Hello ") ; print (" Jed ")
```

Hello
Jed

It is also possible to display the contents of several variables (of any type) separated by commas:

```
Entrée [10]: x = 4  
             nom = " Jed "  
             print (nom , "a", x, " ans ")
```

Jed a 4 ans

II. Importing a .CSV file:

↳ The 'pwd' command displays the current working directory. The .csv file and your script must be in the same folder.

Entrée [2]:

```
pwd
```

Out[2]: 'C:\\Users\\Lenovo\\Documents\\tp_mémoire\\new'

Entrée [3]:

```
import pandas as pd
df = pd.read_csv("acm.csv", sep=";")
print(df)
```

```
   DE Age W-dT EXP      METH S-VAG      COUR \
0  Dr-SP  A  Anb  Non          RS    Pos    Poly
1   Eco A+  Alg  Oui          PF    Pos    Poly
2  Info  A  Tizi Non    D-APP, YT    Neg    Vid
3  Soci  A  Lgh  Non          PF    Neg    Vid
4   ST   A  Sba  Non          PF    Neg    Poly
..  ...  ..  ...  ...      ...    ...    ...
299 Psy   A  Cnst  Oui    RS, D-APP  Pos    Poly
300 ENS   A  Cnst  Non    PF, D-APP  Pos    Poly
301 Psy  A+  Cnst  Oui          RS    Pos    Poly
302 SNV   A  Anb  Non          RS    Neg    Poly
303 S-hum A  Cnst  Oui  PF, RS, D-APP  Neg  Poly, CD

      DIFF DM  %DM Chrg-dT RSLT  OPN \
0  Diff-In, Diff-Cr, Mq-cnt, Abs-Com, Inc-evl -  NaN  M-Ch  Sim  Bn-P
1                    Diff-In, Abs-Com, Inc-evl -  NaN  M-Ch  Pir  Bn-P
2                    Abs-Com, Inc-evl -  NaN  P-Ch  Sim  Bn-P
3                    Diff-In, Mq-cnt, Abs-Com -  NaN  M-Ch  Pir  Bn-P
4                    Inc-evl -  NaN  S-Ch  Sim  Bn-P
```

Chapter 4: Variables and functions

A. Variables

1) Definition:

A variable is an area of computer memory in which a value is stored. To the programmer, this variable is defined by a name, but to the computer, it is actually an address, i.e. a particular area of memory.

In Python, the declaration of a variable and its initialization (i.e. the first value to be stored in it) take place at the same time. To see for yourself, test the following instructions after launching the interpreter:

```
Entrée [17]: x=5  
x  
Out[17]: 5
```

2) Variable types :

The type of a variable corresponds to its nature. The three main types we will need to start with are **integers** (integer or int), **decimal numbers** (we will call them floats) and **strings** (string or str). Of course, there are many other types (Booleans, complex numbers, etc.).

In the previous example, we stored an integer (int) in variable x, but it is perfectly possible to store floats, strings or many other types of variable that we will see later:

```
Entrée [19]: y = 3.14  
y  
Out[19]: 3.14  
Entrée [20]: a = " bonjour "  
a  
Out[20]: ' bonjour '
```

3) Naming :

Variable names in Python can be made up of lowercase letters (**a to z**), uppercase letters (**A to Z**), numbers (**0 to 9**) or the underscore character (**_**). You cannot use spaces in variable names.

In addition, a variable name must not start with a number, and starting it with the **_** character is not recommended.

In addition, you must absolutely avoid using a word "reserved" by Python as a variable name (e.g. `print`, `range`, `for`, `from`, etc.).

Finally, Python is case-sensitive, which means that the variables `TestT`, `test` or `TEST` are different.

4) Operations :

↳ Operations on numeric types:

The four basic arithmetic operations are performed in a simple way on numeric types (integers and floats):

```
Entrée [21]: x=20  
x+2
```

```
Out[21]: 22
```

```
Entrée [22]: x-5
```

```
Out[22]: 15
```

```
Entrée [23]: x*3
```

```
Out[23]: 60
```

```
Entrée [24]: y=5  
x+y
```

```
Out[24]: 25
```

```
Entrée [25]: (x/2)-y
```

```
Out[25]: 5.0
```

Note, however, that if you mix integer and float types, the result is returned as a float (as this type is more general). Parentheses can also be used to manage priorities.

The `/` operator performs a division. Unlike the `+`, `-` and `*` operators, this one always returns a float:

```
Entrée [26]: 3/4
Out[26]: 0.75

Entrée [27]: 2.5/2
Out[27]: 1.25
```

The power operator uses the ** symbols:

```
Entrée [28]: 2**3
Out[28]: 8
```

To obtain the quotient and remainder of an integer division, use the // and modulo % symbols respectively:

```
Entrée [29]: 8//3
Out[29]: 2

Entrée [30]: 8%3
Out[30]: 2
```

The symbols +, -, *, /, **, // and % are called operators, as they perform operations on variables. Finally, there are "combined" operators, which perform an operation and an assignment in a single step. The += operator performs an addition and then assigns the result to the same variable. This operation is called "incrimination". The -=, *= and /= operators behave in a similar way for subtraction, multiplication and division.

```
Entrée [32]: n=0
             n=n+1
             n
Out[32]: 1

Entrée [33]: n+=5
             n
Out[33]: 6

Entrée [34]: n*=2
             n
Out[34]: 12
```

👉 String operations:

For character strings, two operations are possible: addition and multiplication. The addition operator + concatenates (assembles) two character

strings. The multiplication operator `*` between an integer and a string duplicates (repeats) a string several times.

```
Entrée [35]: chaine="salut"
             chaine
Out[35]: 'salut'

Entrée [36]: chaine+'python'
Out[36]: 'salutpython'

Entrée [37]: chaine*2
Out[37]: 'salutsalut'
```

5) Maximum and minimum:

Python offers the `min ()` and `max ()` functions, which return the minimum and maximum of several integers and/or floats, respectively:

```
Entrée [39]: pi=4.13
             e=2.71
             max(e,pi)
Out[39]: 4.13
```

✓ Note on dividing two integers:

Note that in Python 3, dividing two integers returns a float by default:

```
Entrée [40]: m=3/4
             m
Out[40]: 0.75

Entrée [41]: type(m)
Out[41]: float
```

B. Function

1. Principle and generality:

In programming, functions are very useful for performing the same operation several times within a program.

They also make code clearer and more readable by breaking it down into logical blocks.

You are already familiar with some Python functions. For example, **`math.cos (angle)`** in the `math` module returns the cosine of the angle variable, expressed in radians.

□ To which you pass none, one or more variable(s) in brackets. These variables are called arguments. They can be any type of Python object.

□ Who performs an action.

□ And returns a Python object or nothing at all.

For example, if you call the **len ()** function as follows:

```
Entrée [42]: len ([0 , 1, 2])  
Out[42]: 3
```

Here is what happens:

- call **len ()** and pass it a list as an argument (in this case the list [0, 1, 2]);
- the function calculates the length of this list;
- It returns an integer equal to this length.

2. Definition

To define a function, Python uses the **def** keyword. If you want the function to return something, use the **return** keyword. For example:

```
Entrée [43]: def carre (x):  
              return x **2  
              print ( carre (2))  
4
```

Note that the syntax of **def** uses a **colon**, so a block of instructions is expected. **Indentation** of this block of instructions (called the function body) is **mandatory**.

3. Passing arguments :

The number of arguments that can be passed to a function is variable. The number of arguments is therefore left to the initiative of the programmer developing a new function.

A special feature of Python functions is that you do not have to specify the type of the arguments you pass to them, as long as the operations you perform with these arguments are valid. Python is in fact known as a "dynamically typed" language, meaning that it recognizes the type of variables for you at runtime.

For example:

```
Entrée [44]: def fois (x, y):  
              return x*y  
              fois (2, 3)
```

```
Out[44]: 6
```

```
Entrée [45]: fois (3.1415 , 5.23)
```

```
Out[45]: 16.430045000000003
```

```
Entrée [46]: fois ('to ', 2)
```

```
Out[46]: 'to to '
```

```
Entrée [47]: fois ([1 ,3] , 2)
```

```
Out[47]: [1, 3, 1, 3]
```

4. Forwarding results

A huge advantage in Python is that functions are able to return multiple objects at once, as in this code snippet:

```
Entrée [48]: def carre_cube (x):  
              return x**2 , x **3  
              carre_cube (2)
```

```
Out[48]: (4, 8)
```

In reality, Python returns a single object, but it can be sequential, i.e. it can contain other objects. In our example, Python returns an object of type tuple, a type we will see in the next chapter (roughly speaking, it is a kind of list with different properties). Our function could just as easily return a list:

```
Entrée [49]: def carre_cube2 (x):  
              return [x**2 , x **3]  
              carre_cube2 (3)  
Out[49]: [9, 27]
```

Returning a tuple or list of two or more elements is very useful in conjunction with multiple assignment, for example:

```
Entrée [50]: z1 , z2 = carre_cube2 (3)  
            z1  
Out[50]: 9  
Entrée [51]: z2  
Out[51]: 27
```

This allows you to retrieve several values returned by a function and assign them to different variables on the fly.

***Note:** A tuple is defined in the same way as a list, except that the set of elements is enclosed in brackets rather than square brackets.

Chapter 5: Lists and Tuples

I. List

1. Definition:

A list is a data structure containing a series of values. Python allows the construction of lists containing values of different types (e.g. integer and string), giving them great flexibility. A list is declared by a series of values (do not forget single or double quotation marks if strings are involved) separated by commas, and the whole enclosed in square brackets. Here are a few examples:

```
Entrée [1]: tailles = [5, 2.5 , 1.75 , 0.15]
            tailles
```

```
Out[1]: [5, 2.5, 1.75, 0.15]
```

```
Entrée [2]: mixte = ['girafe ', 5, 'souris ', 0.15]
            mixte
```

```
Out[2]: ['girafe ', 5, 'souris ', 0.15]
```

2. Use

One of the great advantages of a list is that you can call up its elements by their position. This number is called the index of the list. Bear in mind that the indices of a list of n elements start at 0 and end at $n-1$.

Consequently, if we call the index 4 element of our list, Python will return an error message:

```
Entrée [10]: mixte = ['girafe ', 5, 'souris ', 0.15]
```

```
Entrée [11]: mixte[0]
```

```
Out[11]: 'girafe '
```

```
Entrée [12]: mixte[3]
```

```
Out[12]: 0.15
```

```
Entrée [13]: mixte[4]
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-13-1c07c49f3746> in <module>
----> 1 mixte[4]
```

```
IndexError: list index out of range
```

3. List operation

Like strings, lists support the + operator for concatenation, as well as the * Operator for duplication. The + operator is very useful for concatenating two lists.

```
Entrée [14]: ani1 = ['girafe ', 'tigre ']  
             ani2 = ['singe ', 'souris ']  
             ani1 + ani2
```

```
Out[14]: ['girafe ', 'tigre ', 'singe ', 'souris ']
```

```
Entrée [15]: ani1 * 3
```

```
Out[15]: ['girafe ', 'tigre ', 'girafe ', 'tigre ', 'girafe ', 'tigre ']
```

You can also use the **.append ()** method to add a single element to the end of a list.

In the following example, we will create an empty list, then add two elements to it, one after the other, first with concatenation, then with the **.append()** method:

```
Entrée [16]: a=[]  
             a = a + [15]  
             a = a + [-5]  
             a. append (13)  
             a. append (6)  
             a
```

```
Out[16]: [15, -5, 13, 6]
```

4. Brackets

Another advantage of lists is that you can select part of a list by using an index built on the [m : n+1] model to retrieve all elements, from the eighth to the nth (from element m included to element n+1 excluded). Note that when no index is specified to the left or right of the colon symbol, Python defaults to all elements from the beginning or all elements to the end respectively.

```
Entrée [17]: animaux = ['girafe ', 'tigre ', 'singé ', 'souris ']  
animaux [0:2]
```

```
Out[17]: ['girafe ', 'tigre ']
```

```
Entrée [18]: animaux [1:]
```

```
Out[18]: ['tigre ', 'singé ', 'souris ']
```

```
Entrée [19]: animaux [:]
```

```
Out[19]: ['girafe ', 'tigre ', 'singé ', 'souris ']
```

You can also specify the pitch by adding an extra colon symbol and indicating the pitch with an integer.

```
Entrée [20]: x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
x [1:6:3]
```

```
Out[20]: [1, 4]
```

```
Entrée [21]: x [::2]
```

```
Out[21]: [0, 2, 4, 6, 8]
```

Finally, we realize that accessing the contents of a list is based on the **list** model **[start : end : step]**.

5. len () function

The **len ()** statement lets you find out the length of a list, i.e. the number of elements it contains.

```
Entrée [22]: len(x)
```

```
Out[22]: 10
```

6. The range () and list () functions

The **range ()** statement is a special function in Python that generates integers within an interval. When used in combination with the **list ()** function, it produces a list of integers. For example :

```
Entrée [23]: list ( range (10))
```

```
Out[23]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In the example above, the **range ()** function took one argument, but it can also take two or three arguments.

The **range ()** instruction is based on the **range** model (**[start,] end [, step]**) The arguments in square brackets are optional. To obtain a list of integers, it must be used systematically with the **list ()** function. The default step value is 1.

```
Entrée [24]: list ( range (15 , 20))
```

```
Out[24]: [15, 16, 17, 18, 19]
```

```
Entrée [25]: list ( range (2, -2, -1))
```

```
Out[25]: [2, 1, 0, -1]
```

7. Minimum, maximum and sum of a list

The **min ()**, **max ()** and **sum ()** functions respectively return the minimum, maximum and sum of a list passed as an argument.

```
Entrée [26]: liste = list ( range (10))  
liste
```

```
Out[26]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
Entrée [27]: sum ( liste )
```

```
Out[27]: 45
```

```
Entrée [28]: min ( liste )
```

```
Out[28]: 0
```

```
Entrée [29]: max ( liste )
```

```
Out[29]: 9
```

II. Tuples

a) Introduction :

From the basic types (int, float, etc.), it is possible to create new ones. These are called constructed types. An example of a constructed type is the tuple. It can be used to create an ordered collection of several elements. In mathematics, this is known as a puplet. For example, a quadruplet is made up of 4 elements.

Tuples are similar to lists, but you cannot modify them once they have been created. A tuple is said to be non-mutable. This is defined with parentheses. Sometimes, tuples are not surrounded by parentheses, even though they are still tuples. For example, the following notation can be used:

```
Entrée [30]: a = (3, 4, 7)
              type(a)
Out[30]: tuple

Entrée [32]: b, c = 5, 6
              b
Out[32]: 5

Entrée [33]: c
Out[33]: 6
```

This syntax with several variables on the left of the = sign can also be used with a single variable on the right if it contains a tuple.

```
Entrée [34]: a = (3, 4)
              u, v = a
              u
Out[34]: 3

Entrée [35]: v
Out[35]: 4
```

b) How do you write a function that returns a tuple of values?

It is very easy to create a function that returns a tuple. Simply specify the tuple after return.

```
Entrée [36]: def test():  
              return 3, 4
```

```
Entrée [37]: a = test()  
              a
```

```
Out[37]: (3, 4)
```

```
Entrée [38]: b, c = test()  
              b
```

```
Out[38]: 3
```

```
Entrée [39]: c
```

```
Out[39]: 4
```

c) How do you iterate over the elements of a tuple?

Like a list, a tuple can be browsed using a **for** loop.

```
Entrée [40]: for i in a:  
              print(i)
```

```
3  
4
```

d) How do you create a tuple containing a single element?

If you only use parentheses, you will not get the desired result. To create a tuple containing a single element, you need to use a specific syntax containing a comma.

```
Entrée [41]: a = (3)  
              type(a)
```

```
Out[41]: int
```

```
Entrée [42]: b = (3,)  
              type(b)
```

```
Out[42]: tuple
```

Chapter 6: *Loops and Testing*

I. *Loops and comparison:*

1. Loops for:

a. Principle:

```
Entrée [52]: animaux = ['girafe ', 'tigre ', 'singe ', 'souris ']  
for animal in animaux :  
    print ( animal )
```

```
girafe  
tigre  
singe  
souris
```

➤ Let's take a closer look at what happened in this example:

The animal variable is called the **iteration** variable, and takes on the various values of the animal list at each iteration of the loop. It is created by Python the first time the for line is executed (if it already existed, its contents would be overwritten). Once the loop is complete, this animal iteration variable will not be destroyed, and will therefore contain the last value of the animal list (in this case, the mouse string).

Note the types of variables used here: animals is a **list** being iterated over, and animal is a **string**, as each element in the list is a string. We will see later that the iteration variable can be of any type, depending on the list being iterated over. In Python, a loop always iterates over a sequential object (i.e. an object made up of other objects) such as a list.

Please note the colon ":" at the end of the line starting with for. This means that the for loop is waiting for a block of instructions, in this case all the instructions that Python will repeat on each iteration of the loop. This block of instructions is called the loop body. How do we tell Python where this block begins and ends? This is indicated solely by indentation, i.e. shifting the line(s) of the instruction block to the right.

In the following example, the loop body contains two instructions: **print (animal)** and **print (animal*2)**, because they are indented with respect to the line starting with **for**. **print ("It's finished")** is not part of the loop body, because it is at the same level as **for** :

```
Entrée [53]: for animal in animaux :  
              print ( animal )  
              print ( animal *2)  
            print ("C'est fini ")  
  
girafe  
girafe girafe  
tigre  
tigre tigre  
singe  
singe singe  
souris  
souris souris  
C'est fini
```

b. Range () function:

Python's **range ()** function is a convenient way of looping through a list of integers automatically:

```
Entrée [56]: for i in range(5):  
              print (i)  
  
0  
1  
2  
3  
4
```

c. Naming the iteration variable

In the previous example, we chose the name **i** for the iteration variable. This is customary in computer science and generally indicates that the variable is an integer (the name **i** probably comes from the word index). We advise you to follow this convention to avoid confusion: if you iterate on indices, you can call the iteration variable **i** (for example, in **for i in range(4) :**). If, on the other hand, you're iterating over a list containing character strings, give the iteration variable an explicit name. For example: **for name in ['Joe', 'Bill', 'John'] :**

2. Comparison :

Before we move on to another type of loop (while loops), we will start with comparisons. Python can perform a whole series of comparisons between the contents of two variables, such as:

Syntaxe Python	Signification
<code>==</code>	égal à
<code>!=</code>	différent de
<code>></code>	supérieur à
<code>>=</code>	supérieur ou égal à
<code><</code>	inférieur à
<code><=</code>	inférieur ou égal à

Look at the following examples using integers. Python returns True if the comparison is true and False if it is false. True and False are Booleans (a new type of variable).

```
Entrée [57]: x = 5
              x > 10
Out[57]: False

Entrée [58]: x < 10
Out[58]: True
```

In the case of character strings, a priori only the `==` and `!=` tests make sense. In fact, the operators `<`, `>`, `<=` and `>=` can also be used. In this case, alphabetical order is taken into account, for example:

```
Entrée [59]: "a" < "b"
Out[59]: True
```

- Be careful not to confuse the `=` **assignment** operator, which assigns a value to a variable, with the `==` **comparison** operator, which compares the values of two variables.

3. While loop:

Another alternative to the for statement commonly used in computing is the while loop. The principle is simple. A series of instructions is executed as long as a condition is true. For example:

```
Entrée [60]: i = 1
              while i <= 4:
                  print (i)
                  i = i + 1
              1
              2
              3
              4
```

Note that it is once again necessary to indent the block of instructions corresponding to the body of the loop. A while loop generally requires three elements to function correctly:

- ✓ Initialize the iteration variable before the loop.
- ✓ Test the iteration variable associated with the while statement.
- ✓ Update the iteration variable in the loop body.

Pay close attention to the tests and increments you use, as an error often leads to "infinite loops" that never stop. However, you can always stop the execution of a Python script by pressing **Ctrl-C**.

The while loop combined with the **input ()** function can be very useful when you want to ask the user for a numerical value. The **input ()** function takes a message as argument (in the form of a character string), asks the user to enter a value and returns this in the form of a character string. This must then be converted to an integer (using the **int()** function).

```
Entrée [*]: i = 0
              while i < 10:
                  reponse = input (" Entrez un entier supérieur à 10 : ")
                  i = int ( reponse )
```

Entrez un entier supérieur à 10 : 5

Entrez un entier supérieur à 10 :

```
Entrée [62]: i = 0
while i < 10:
    reponse = input (" Entrez un entier supérieur à 10 : ")
    i = int ( reponse )
```

```
Entrez un entier supérieur à 10 : 5
Entrez un entier supérieur à 10 : 12
```

```
Entrée [63]: i
```

```
Out[63]: 12
```

II. Test:

1) Definition:

Tests are an essential part of any computer language, if you want to give it any complexity at all, as they enable the computer to make decisions. For this purpose, Python uses the if statement as well as the comparison we discussed earlier. Here are two examples:

```
Entrée [64]: x = 2
if x == 2:
    print (" Le test est vrai !")

Le test est vrai !
```

```
Entrée [65]: x = " souris "
if x == " tigre ":
    print (" Le test est vrai !")
```

There are several points to be made about these two examples:

- ★ In the first example, since the test is true, the print statement ("The test is true!") is executed. In the second example, the test is false and nothing is displayed.
- ★ Instruction blocks in tests must be indented, as in for and while loops. Indentation indicates the range of instructions to be executed if the test is true.
- ★ As with the for and while loops, the line containing the if statement ends with a colon ":".

2) Multi-case tests

Sometimes it's useful to test whether a condition is true or false in the same if statement. Instead of using two if statements, you can use the **if** and **else** statements. You can use a series of tests in the same if statement, for example, to test several values of the same variable.

```
Entrée [66]: x = 2
              if x == 3:
                  print (" Le test est vrai !")
              else :
                  print (" Le test est faux !")

              Le test est faux !
```

3) Multiple tests

Multiple tests allow you to test several conditions at the same time, using Boolean operators. The two most commonly used operators are **OR** and **AND**. Here is a quick reminder of how the OR operator works:

Condition 1	Opérateur	Condition 2	Résultat
Vrai	OU	Vrai	Vrai
Vrai	OU	Faux	Vrai
Faux	OU	Vrai	Vrai
Faux	OU	Faux	Faux

and the **AND** operator:

Condition 1	Opérateur	Condition 2	Résultat
Vrai	ET	Vrai	Vrai
Vrai	ET	Faux	Faux
Faux	ET	Vrai	Faux
Faux	ET	Faux	Faux

In Python, we use the reserved word **and** for the AND operator and the reserved word **or** for the OR operator. Please note that the operators **and** and

or are case-sensitive, and are written in lower case in Python. Here is an example:

```
Entrée [67]: x = 2
              y = 2
              if x == 2 and y == 2:
                  print (" le test est vrai ")
              le test est vrai
```

Note that the same result would be obtained using two nested if statements:

```
Entrée [68]: x = 2
              y = 2
              if x == 2:
                  if y == 2:
                      print (" le test est vrai ")
              le test est vrai
```

Finally, we can use the logical negation operator not, which inverts the result of a condition:

```
Entrée [69]: not True
Out[69]: False
```

4) Break and continue instructions

These two instructions modify the behavior of a loop (for or while) with a test. The **break** instruction **stops the loop**. The **continue** instruction **jumps to the next iteration, without executing the rest of the loop instruction block**.

```
Entrée [70]: for i in range (5):
              if i > 2:
                  break
              print (i)
              0
              1
              2
```

```
Entrée [72]: for i in range (5):
              if i == 2:
                  continue
              print (i)
              0
              1
              3
              4
```

GENERAL CONCLUSION

Python is a language designed to be easily understood. For this reason, and for the structure of the language in general, which encourages developers to code more cleanly and adopt good practices, it is one of the most recommended languages for people wishing to learn programming.

Python may be simple to learn, but it is no less powerful and versatile: this language can be used to perform complex tasks and to carry out a wide variety of different projects, from the creation of Web applications to various programs. It takes much more than a few hours to master all the subtleties of the Python language. Just as every language has its own vocabulary, syntax and grammar, so too does Python. With the basics covered in this course, it's possible to handle a wide range of projects, both simple and not-so-simple, and the fact that Python is free and open source, and that there's a huge community, means that solutions can be found.

References

- Guido van Rossum, The Python Language Reference, 2018.
- Prolixe, Apprenez à programmer en Python, 2013.
- Patrick Fuchs et Pierre Poulain, Cours de Python, 2020.
- Michael Dawson, Python programming for the absolute beginner
- Gérard Swinnen, Apprendre à programmer avec Python, 2005.
- W. McKinney, Data Structures for Statistical Computing in Python
Proceedings of the 9th Python in Science Conference, 2010.
- Adam Stewart, Python Programming, Python Programming for
Beginners, 2016.
- Alexandre Casamayou-Boucau, Pascal Chauvin, Guillaume
Connan, Programmation en Python pour les mathématiques, 2016.
- T. Halwanter, An Introduction to Statistics with Python, Springer
2016.
- A. C. Müller & S. Guido, Introduction to Machine Learning with
Python, O'Reilly 2017.
- S. Bays, Informatique avec Python, Ellipses 2017.
- <https://www.anaconda.com/download/>
- <https://www.codecademy.com/>