# **Object-Oriented Programming**

# Python is an object-oriented programming language—this means that in Python, everything is an object! In this chapter, you will see what this means and how you can use it in a practical way. Start by looking at a few objects from everyday life, such as pens, books, smartphones, computers, etc.

# Very different objects!

# Objects can have many different shapes and characteristics, but you can classify different versions of the same object into a category or group. That's why it's easy to recognize a chair in a store, for example, although its appearance (shape, color, etc.) can vary greatly from one model to another.

# It is by observing the common points between different objects that you are able, mentally, to classify the objects in the same group or category!

# For example, there are different types of books, but they all have a title, an author, a back cover, etc. All books share different attributes that let you classify them in a well-identified category: books.

# **Classes: Object Models**

# In programming, this concept of a group or category of objects is called a class. A class can be considered as the construction diagram for an object that will define the characteristics of all objects of this type and their features. From this class, you will be able to create different models of an object.

# Let's take a concrete example with a Car class. The plan of a car can be defined by:

# its characteristics, called attributes: it must have four wheels, a color, a shape, an engine power, etc.

# its functionalities, called methods: it can drive, brake, etc.

# So, from this plan, you can create different car models:

# An ordinary family car, green, medium power (110 hp)

# A sports car, red, relatively powerful (180 hp)

# A small blue city car, not very powerful (90 hp)

# etc.

# And no matter what the car model, they are all capable of driving or braking, but not with the same performance!

# In summary, a class is the outline of an object, defining its attributes and methods. From the same class, you can therefore create several objects of the same type, but with different attributes—these are called class instances.

# **Focus on Methods**

# As we said before, in Python, everything is an object. This means that, without knowing it, since the beginning of this course, you have been manipulating objects! Consider the following lines of code to illustrate this:

# var1 = 14

# var2 = 1031

# Here, you have declared two variables named var1 and var2 containing the values 14 and 1,031. In reality, you have created two instances of the int class, two objects each with a single attribute: its value. The same is true for floats or strings: every time you create a variable of one of these types, you are actually creating objects in Python with the value you assign to them as an attribute.

# Up until now, we have talked about attributes, so now it is time to see what methods are. A class method is a function that is only available for the instances of this class. If, for example, we consider the Car class presented above having a drive() method, and a Plane class having a fly() method, you will agree quite logically that a plane can’t drive, and a car can’t fly. The same goes for our various objects!

# The use of a method is always done via the variableName.method() notation. For example, strings have a method called lower() which will transform all the text contained in an object into lower case. Here's how to use it:

# The string method: lower

# In the same way as with functions, class methods can take parameters.

# **String Methods**

# During the various data analyses that you will have to perform, you will inevitably be confronted with textual variables at some point. You have already seen how to change your string to lowercase, but you may also need to replace some specific words, format the text in a certain way, etc.

# Python has implemented many methods to allow us to do all this. Here are the most common ones:

# upper() : returns the whole text in upper case.

# capitalize() : returns the whole text in lowercase with the first letter capitalized.

# replace(old, new) : this method takes two arguments: old and new, both of which are strings. The method returns the original string with all occurrences of old replaced with new.

# find(string) returns either the index of the first occurrence of the string passed in the argument, or -1 if it does not find it.

# Here are some examples of how these methods are used:

# **String Methods**

# As you can see here, especially with the lines concerning the a variable, the methods seen above do not modify the initial object! They only return the result of the method applied to the object. You will regularly have to reassign this result to the initial variable, when you want to modify it directly.

# Try it for Yourself

# Manipulate objects yourself in the following exercise.

# You can find the solution here.

# **Let's Recap**

# A class is a construction plan for an object.

# A variable is an instance of a class, or an object.

# An object is defined by its attributes.

# All instances of a class have access to the same methods via the. (dot) notation.

# A method, like a function, generally does not modify the initial object.

# In the next part, we will see in more detail how to organize your code via different structures and complex objects.

# **1.1.** [**Learn Python Basics for Data Analysis**](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis)

# **Course introduction**

To create **computer programs** or build **algorithms for data analysis**, you need to master a **programming language**.

A program tells a computer what to do, using data and statements that you give it! So where do you start to write your own programs in Python?

This course will teach you the basics of the **Python programming language**, which is necessary to work in the field of **data analysis**.

You will learn how to **use and write the basic functions** of Python. You will manipulate the **most common objects** of this language and implement **programming logic**. Your program will do exactly what you tell it to do!

Are you ready to learn about Python for data analysis?

**By the end of this course, you will be able to:**

* set up your Python work environment.
* use the fundamental functions and objects in Python.
* organize objects with collections.
* manage program flow.
* use specialized Python libraries for data analysis.

**Prerequisites:**

None!

**Tools needed:**

* A computer
* Anaconda with Jupyter Notebook
* Or Google Colaboratory
* [#](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis#/id/r-7978662)

**Continue this course**

**Unfollow**

### **Turn it into a career**

Take one of our 100% online diploma training programs, and transform your know-how into professional skills.

* Up to 100% of your training program funded
* Flexible start date
* Career-focused projects
* Individual mentoring

**[Enroll now](https://openclassrooms.com/en/application/welcome?appsource=generic)**

#### **Discover training programs**

1/2

[Data**Data analyst**Analyze data and model phenomena with realistic business cases. Start your career in Data Science now!](https://openclassrooms.com/en/paths/898)

[OpenClassrooms CertificationFull time: 12 months](https://openclassrooms.com/en/paths/898)

[Data**AI Engineer**Design and integrate Deep Learning models inside an AI product](https://openclassrooms.com/en/paths/906)

[OpenClassrooms CertificationFull time: 12 months](https://openclassrooms.com/en/paths/906)

## **Table of contents**

### **Part #1 - Set up Your Python Work Environment**

#### [1. Install Python and Anaconda](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7978748-install-python-and-anaconda)

#### [2. Take Your First Steps with Jupyter Notebook](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7978803-take-your-first-steps-with-jupyter-notebook)

### **Part #2 - Handle Fundamental Functions and Objects**

#### [1. Declare a Variable](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7978962-declare-a-variable)

#### [2. Handle Variable Types](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7979063-handle-variable-types)

#### [3. Write Your Own Functions](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7979178-write-your-own-functions)

#### [4. Object-Oriented Programming](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7979247-object-oriented-programming)

* + [Quiz: Handle Fundamental Functions and Objects](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/exercises/4679)

### **Part #3 - Organize Objects and Manage Program Flow Within a Project**

#### [1. Create Collections to Store Your Objects](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7979496-create-collections-to-store-your-objects)

#### [2. Control Your Code With Conditional Structures](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7979677-control-your-code-with-conditional-structures)

#### [3. Choose the Appropriate Loop](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7979810-choose-the-appropriate-loop)

* + [Quiz: Structure Your Code Within a Project](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/exercises/4680)

### **Part #4 - Use Specialized Python Libraries**

#### [1. Familiarize Yourself With Python Modules and Libraries](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7979957-familiarize-yourself-with-python-modules-and-libraries)

#### [2. Manipulate Random Numbers With the Random Module](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7980005-manipulate-random-numbers-with-the-random-module)

* + **[Quiz: Use Specialized Python Libraries](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/exercises/4681)**

## ***1.2. Install Python and Anaconda***

*Hello and welcome! My name is Llewelyn Fernandes. I'm a computer scientist with a focus on teaching about creative computing and data. It’s my pleasure to lead you through this course! Want to know more about what I do? Follow me @tcreatelearn on Instagram or Twitter.*

### ***Get the Most out of This Course***

*In this course, you will see all the basics of the Python language needed before you can start working with and analyzing data.*

*This course will include several coding activities that will let you test the various concepts learned. For this you will need to set up your environment as detailed below.*

### ***Discover Python***

*Python is a general-purpose programming language, meaning that you can code just about anything in it: websites and applications, mobile apps, personal scripts, desktop applications, data analysis, and even video games!*

*For example:*

* ***Instagram*** *is coded in Python.*
* *It is one of the main languages used by* ***Google*** *engineers.*
* ***Netflix*** *uses Python to build its recommendation algorithms.*
* *The* ***Dropbox*** *desktop application is developed in Python.*
* ***Reddit*** *is coded in Python.*

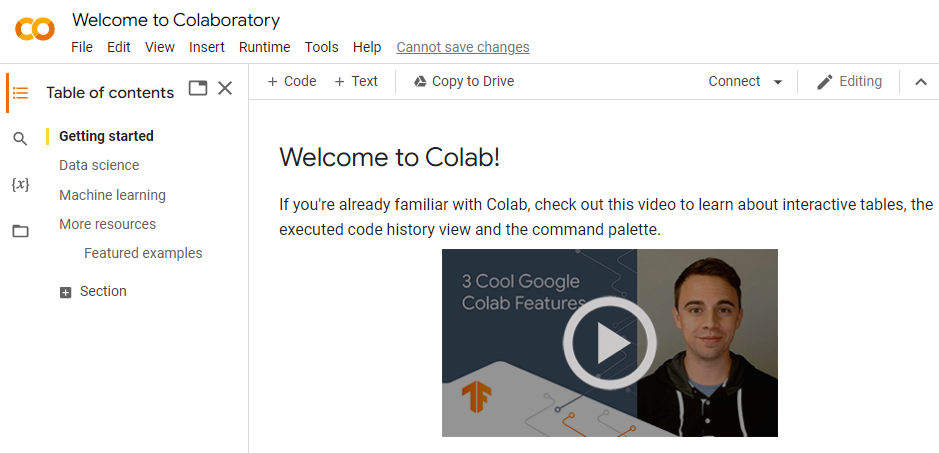
*When it comes to data, Python is the most widely used language, especially for data processing and manipulation, data visualization, and artificial intelligence algorithm development.*

*There are various distributions of Python that you can use to get going. Below, we introduce you to two options: Google Colab and Anaconda. Google Colab is the easiest option to start with as it does not require you to install anything on your computer.*

### ***Use Python in the Cloud via Google Colaboratory (Colab)***

*If you don't want the trouble of installing software on your computer, you can use Python in the Cloud, via* [*Google Colaboratory*](https://s3.eu-west-1.amazonaws.com/course.oc-static.com/courses/2304731/Google%2BColab%2BQuick%2BStart%2BGuide.pdf)*. This is a product from Google Research allowing anyone to write and execute Python code within a Jupyter Notebook, through a web browser (Chrome, Safari, Firefox, etc.). This is what we will use throughout this course to let you practice!*

*No installation is required, you can access it from any machine (as long as it has an internet connection) because all files are stored and run on Google's servers. You need to have a Google account, so set one up if you don’t have one. Then you just have to go* [*to the main Colab page*](https://colab.research.google.com/?utm_source=scs-index)*, create a new notebook and... TADAAAA!!!*

**

*You have launched your first Jupyter Notebook and are ready to continue learning more about this tool in the next chapter.*

### ***Install Python via Anaconda***

*As an alternative to Colab you can install Anaconda, to give you total control of your development environment on your own computer. Anaconda is a scientific distribution of Python, meaning that when you install Anaconda, you will install Python, Jupyter Notebook (which we will present in more detail in the next chapter), and dozens of scientific packages, some of which are essential for data analysis!*

*To start, download the Anaconda distribution corresponding to your operating system from here:* [*https://www.anaconda.com/distribution/*](https://www.anaconda.com/distribution/)

#### ***Install Anaconda on Windows or Mac***

* *Download the Windows or macOS installer and double-click to start the installation.*
* *Answer the different questions (the default options are enough!). Once the installation is complete, you can check that it went well by launching the Jupyter Notebook application.*

#### ***Install Anaconda on Linux***

* *Download the installer for Linux.*
* *Enter the following statement in the terminal, replacing ANACONDAINSTALLER with the name of the file downloaded:*

*bash ~/Downloads/ANACONDAINSTALLER.sh*

* *The installer will print: “In order to continue the installation process, please review the license agreement.” Click to see the license agreement, stroll to the bottom and click yes to accept.*
* *Answer "yes" if you see the following question:*

*Do you wish the installer to prepend the Anaconda3 install location to PATH in your /home/ec2-user/.bashrc ? [yes|no]*

* *Answer “yes” if you see the following question:*

*Do you want the installer to initialize Anaconda3 by running conda init? [yes|no]*

* *The previous commands added the path (the folder) to the .bashrc file where Anaconda is located. So, you will be able to launch Anaconda directly by just typing "anaconda" in your console! This operation will take effect when your computer restarts, or as soon as you run this command:*

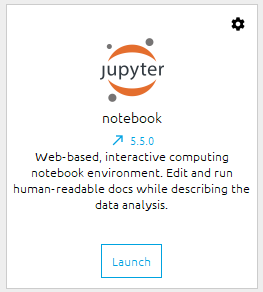
*source .bashrc*

* *Check that everything went well by running the Jupyter program (see below).*

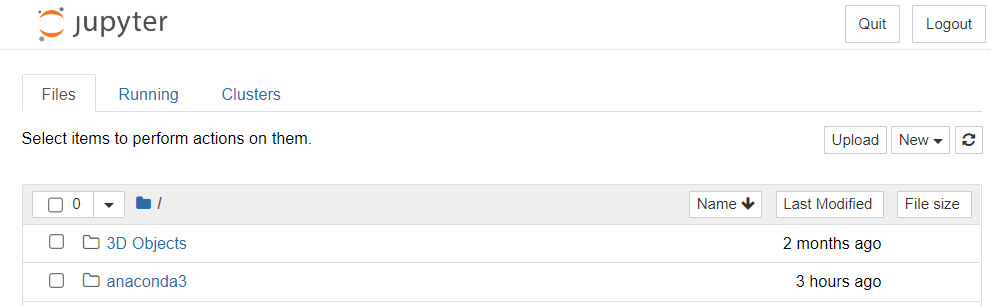
#### ***Launch Jupyter***

* *On macOS, launch Anaconda Navigator via Launchpad.*
* *On Linux, open a console and run: anaconda-navigator.*
* *On Windows, launch Anaconda Navigator from your Start menu.*

*Once this is done, you will have several applications at your disposal. You are interested in the one called Jupyter Notebook:*

**

*Click on the Launch button and a new tab will appear in your browser session:*

**

*Now that you have Anaconda and Python installed, you'll be introduced to one of the most widely used programming tools in data analysis. Easy to use, practical, and powerful, you will soon get to grips with Jupyter Notebook.*

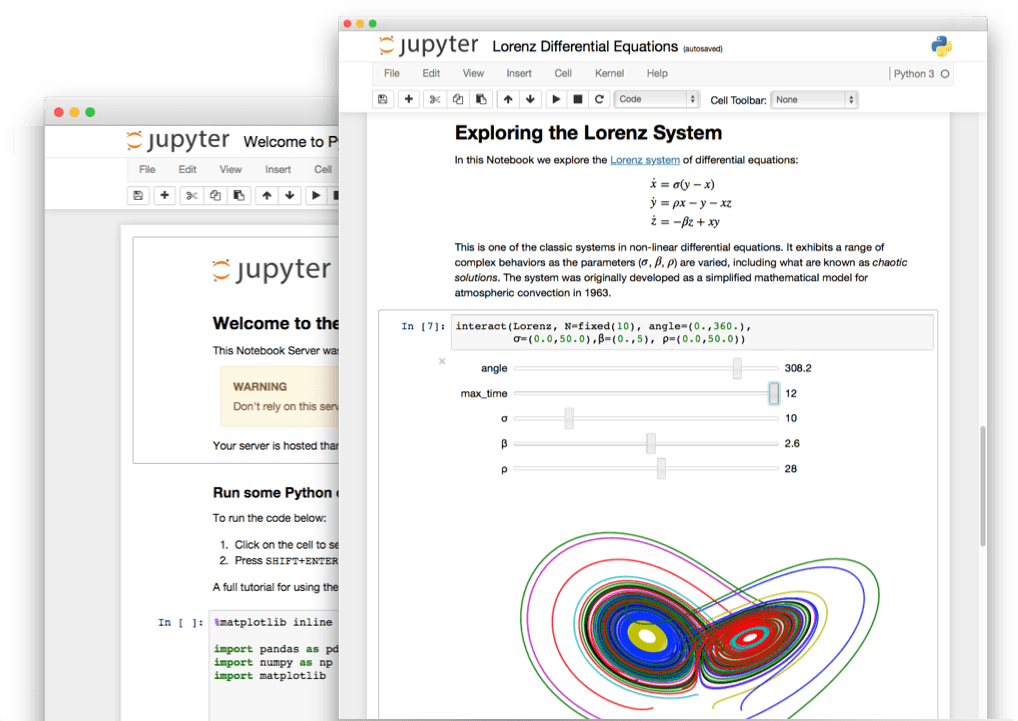
## ***1.3. Take Your First Steps with Jupyter Notebook***

*Jupyter Notebook is a web application that lets you store lines of Python code, their output (graphs, tables, etc.) and formatted text. This can be compared to a web page containing Python code! A bit like this course, really.*

*There are of course other ways and applications to work with than Python, but we are going to explain why Python is particularly well suited for data analysis and why we recommend it!*

### ***Discover Jupyter Notebook***

*Jupyter Notebook is a powerful tool that allows Python language users to create and share interactive documents containing dynamic, runnable code, content displays, documentation text, and equations. The term "notebook" is related to the intrinsic nature of the tool, which lets you write small pieces of runnable code (called "cells"), document them to explain what they do, and print the resulting output data. All this is stored in a document that can be shared with other users.*

**

*So it is particularly useful for prototyping algorithms or testing pieces of code to analyze the results and possibly add them to your main project. With a notebook, you don't need to organize your code into functions, with a main, etc. You just need to write your code in the code cells and run it!*

***Jupyter Notebook for Data Analysis***

*Jupyter Notebook is a very popular application for everyone who works in data analysis. It lets a team create and share reports in document/code format, facilitating productivity and collaboration.*

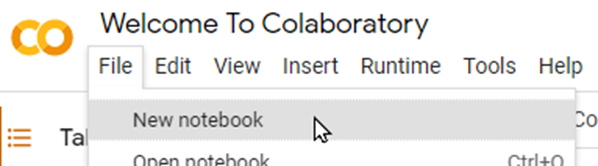
*For example, Netflix data scientists work with their data via Jupyter Notebook—their movie recommendation algorithm is coded from Jupyter! You can check out this article on* [*Medium*](https://medium.com/netflix-techblog/notebook-innovation-591ee3221233) *that talks about it in detail.*

### ***Take Your First Steps with Jupyter***

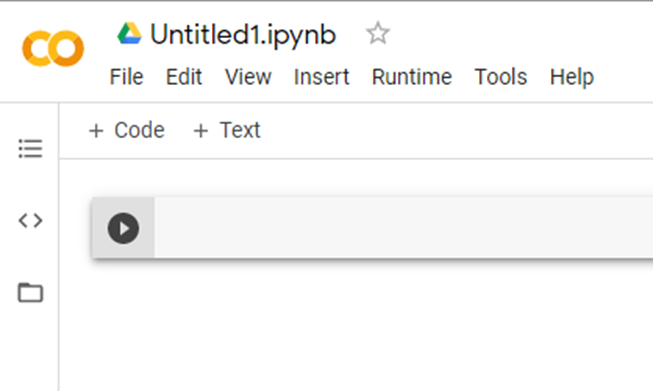
*Now that you've launched Jupyter (either locally or via Google Colab) and we've learned a little more about its usefulness, it's time to run our first Python code.*

*Throughout this course, you will find links to Google Colab so you can practice along at the same time. You can either use them to perform the different exercises in Colab, or download the Jupyter notebook as a .ipynb file and use your own local Jupyter to test the code shown to you.*

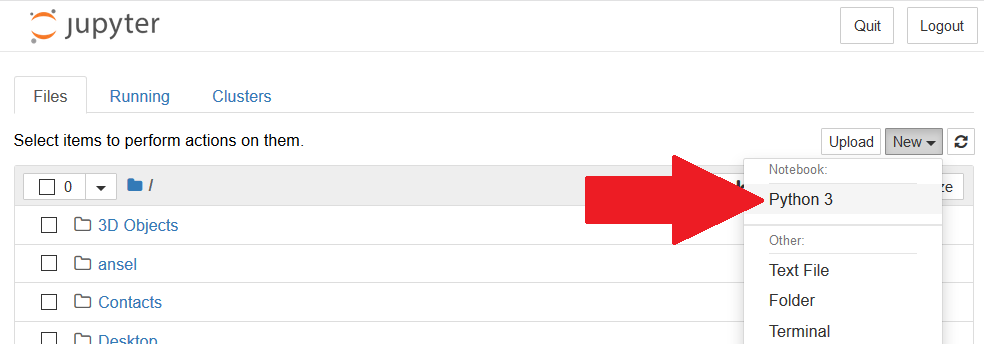
*From Colab create a new notebook by selecting “New notebook” from the File menu:*

**

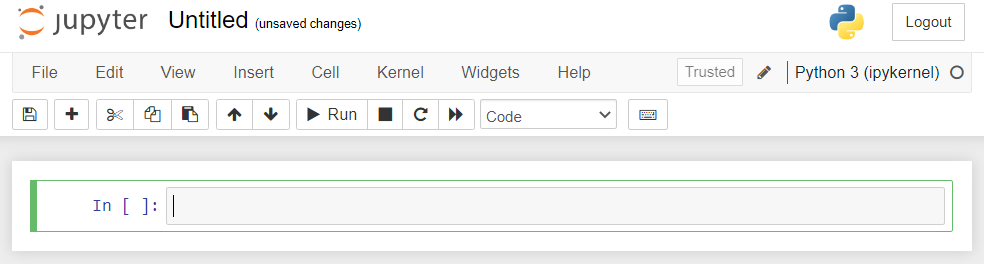
*You should see the notebook appear:*

**

*If you are using Anaconda, create a new notebook by clicking on New then Python 3 :*

**

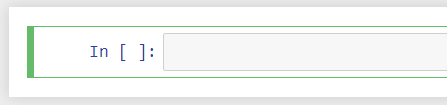
*This will open a new tab:*

**

*Whether you are using Colab or Anaconda, you will have a new code cell. In Colab, it looks like this:*

**

*In Anaconda, it looks like this:*

**

*Try writing your first line of code in this cell. Saying “Hello,” for example, would be a good start.*

*print('Hello, world!')*

*and then click on Run.*

* in Colab*

* in Anaconda*

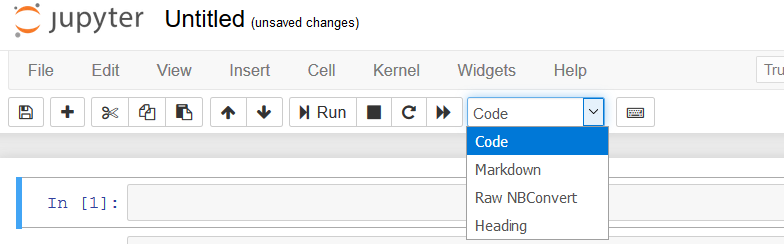
*"Hello, world!" should appear below your cell—this is the result of running it. Congratulations, you just ran your very first Python code!*

*Did you know that printing “Hello, world!” is usually the first thing you learn when learning any programming language?*

### ***Document Your Notebook***

*Before you really dive into the programming, you should know that there are different types of cells with Jupyter, the main ones being* ***code and markdown*** *(also called* ***text****).*

*In Anaconda, when you click on a cell, you can change the type simply by selecting it from the list, as in the example below:*

**

*Let's look at the two main types in more detail:*

* ***Code****—the standard code cell. This one is reserved for writing and running Python code! You can run your code by clicking on the Run button, as seen previously.*
* ***Markdown****—a text cell that is mainly used for notebook documentation, to write comments, titles, equations, etc. This type lets you structure your text using HTML tags or Markdown syntax.*

*In Colab, you can select the cell type by creating a new cell:*

**

*Whether it's markdown or code, just write your text/code and run it via the Run button to see the result.*

*For example, we could create a title. To do this, simply add a # in front of your text. A single # will correspond to a level one title, two ## to a level two title, etc.*

*You can also choose to make your text bold or italic. To do this, simply put the following symbols before and after the desired text:*

* *Use \*\* to make it bold.*
* *Use just \* to italicize it.*

*For example, the following Markdown cell:*

*# A level 1 title*

*Here is some text to introduce a markdown*

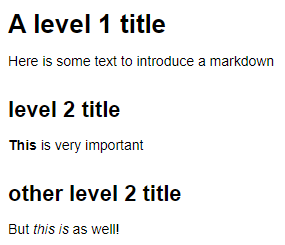
*## level 2 title*

*\*\*This\*\* is very important*

*## other level 2 title*

*But \*this is\* as well!*

*will give the following result:*

**

*If you want to go further into the features of Markdown cells, I advise you to follow* [*this link*](https://medium.com/analytics-vidhya/the-ultimate-markdown-guide-for-jupyter-notebook-d5e5abf728fd)*.*

*Now that your Python environment is set up, you are ready to get down to business!*

## ***2.1. Declare a Variable***

*You know how to print text on the screen. Great! This is a start, but you will be able to do much more after you have seen what* ***variables*** *are in programming.*

### ***Understand What a Variable is***

*Variables are one of the concepts found in all programming languages. You might as well say that without a variable, you can't program, and that's not an exaggeration.*

*As mentioned in the video, think of a variable as a kind of box containing a* ***value****. This box is itself stored on a shelf among many others, in a gigantic warehouse. The location of each box is very precisely recorded, just as your computer records the exact location of your variable in its memory.*

*Boxes stored in a warehouse*

*A* ***value*** *is what you will store in a variable. To return to the warehouse analogy, there are several boxes for storing different values. For example, if you work in a bank, you might want to store information about a customer in different boxes, such as their checking account balance and their savings account balance. We will also need to perform different operations on these boxes like emptying them, adding money, transferring the contents from one to another, etc. Variables will let you do this!*

*To get to the contents of each box, you will need to label them. The process is similar in programming: each variable is given a* ***name****.*

**

*Labeled jars containing your savings*

*In the same way as with box labeling, the name of a variable must always represent its contents. Here are some general recommendations for choosing names for your variables:*

* ***Use clear variable names****It sounds tedious to do, but it is really beneficial for you and the people you will share your code with—it makes it easier to read and maintain your code. For example, savingsAccount and checkingAccount are much more explicit names than account1 and account2 .*
* ***Use explicit variable names****Avoid abbreviations and acronyms, if possible, even if an abbreviation may seem obvious. For example, annualIncome is better than annualInc .*
* ***Follow a typographic convention****One of the most common typographic conventions is called* ***camel case*** *(also known as camel caps). It involves writing variable names containing several words without spaces or punctuation—the first word is written in lowercase, then each word is written with the first letter in upper case, as shown above.*

### ***Create a Variable***

*Before you can use it, you have to create your variable! You just have to associate a value with a name to create a variable; Python takes care of creating the right size ‘box’. Convenient, isn't it?*

*There are several types of variables in Python, much like in the physical world: text, numeric values, arrays, etc. Numeric variables are declared by associating a name with a* ***numeric value****. For example:*

*checkingAccount = 500*

*savingsAccount = 1000*

*You have declared two variables here, checkingAccount and savingsAccount , by storing the values 500 and 1,000, respectively.*

*You have only used integers here! But it is quite possible to store decimal numbers.*

*If you store a new value in an existing variable, the type of the variable will change according to the new value.*

### ***Understand Operations Between Variables***

*As the name suggests, a variable can vary, or rather* ***the value of a variable can change****. You can do this through different* ***operations****. Considering the two variables previously declared in the last example, you could:*

* *add some money to your savings account.*
* *withdraw some from your checking account.*
* *calculate how long it would take you to reach $5,000 if you save $500 every month.*
* *calculate how much would be in your checking account if you added $30 every day for a week.*
* *calculate how much would be in your checking account if you spent $10 every day.*
* *etc.*

*In short, these are concrete problems that can be solved with Python. Each operation will use arithmetic operators:*

* *+ : addition*
* *- : subtraction*
* *\* : multiplication*
* */ : division*

*The rules of arithmetic apply in Python, in particular* [*the order of operations*](https://en.wikipedia.org/wiki/Order_of_operations)*, but as in ordinary mathematics, you can use brackets to rearrange the order of the calculations. See how to do this in Python:*

*# add 100 to our savings (Yeah!)*

*savingsAccount = savingsAccount + 100*

*# remove 50 from our checkingaccount (Sniff)*

*checkingAccount = checkingAccount - 50*

*# calculate the number of days to save to reach 5000*

*numberDaysSave = (5000 - checkingAccount) / 500*

*# update our checkingaccount (again) after the daily gains/losses*

*checkingAccount = checkingAccount + (30 - 10) \* 7*

*That makes a nice piece of code, doesn't it? If you pay attention, you will notice that there are different colors in different places. This is because your code is made up of* ***comments*** *and* ***expressions****:*

* *Lines starting with a # are* ***comments****. They are used to document your code and help others understand it.*
* *The other lines of code (which can run operations, declarations, etc.) are* ***expressions****. They tell the computer what to do.*

*Here, each of the expressions* ***assigns*** *a value to a variable. The assignment operator is = .*

1. *To the right of the assignment operator is the expression that will create/calculate a* ***value****.*
2. *To the left of the assignment operator, you write* ***the name of the variable*** *to which you will assign the corresponding value.*

*To summarize, to assign a value to a variable, you write an expression. This expression is built with the name of the variable, followed by the assignment operator = and finally the value to associate.*

*There are also other arithmetic operators that are a little more complex, but nevertheless useful, such as:*

* *% modulo : returns the remainder of the euclidean division*
* *\*\* power : raises a number to a certain power*
* *// integer division: calculates the integer division (rounded down)*

*print(SavingsAccount % 500) # -> 100*

*# 1100 = 500 \* 2 + 100, so 1100 % 500 = remainder = 100*

*print(9 \*\* 3) # -> 729, 9\*9\*9 = 729*

*print(SavingsAccount // 500) # -> 2*

*# 1100 = 500 \* 2 + 100, so 1100 // 500 = integer division result = 2\**

#### ***Try it for Yourself***

*Try to declare a variable from two other variables* [*in the following exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C1_ex1.ipynb) *yourself.*

**

# *Learn Python Basics for Data Analysis*

## *Declare a Variable*

*Run the following case that declares the variables:*

* *savingsAccount corresponding to the amount in the savings account*
* *rate the interest rate for the year in question*

*savingsAccount = 10000*

*rate = 0.75*

*Create a variable interest, which is the amount in the savings account multiplied by the interest rate (which itself must be divided by 100, because rates are expressed as a percentage):*

*interest = savingsAccount \* (rate / 100)*

*interest*

*75.0*

*print("Your account balance was", savingsAccount, "€\nYou earned", interest, "€ this year")*

*Your account balance was 10000 €*

*You earned 75.0 € this year*

*You can find the* [*solution right here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C1_sol1.ipynb)*.*

***Write Shorter Code with Simplified Operators***

*When you want to change a variable by changing the initial value via a basic operator, you can use a shorter version. In other words, you can use* ***simplified operators****! For example, rather than using the expression savingsAccount = savingsAccount + 100 to add $100 to your variable, you can use a kind of combined arithmetic/assignment operator += :*

*# long version*

*savingsAccount = savingsAccount + 100*

*# equivalent short version*

*savingsAccount += 100*

*Naturally, there is a version for each arithmetic operator seen previously:*

* *-= for subtraction*
* *\*= for multiplication*
* */= for division*
* *%= for the remainder of the integer division*
* *etc.*

#### ***Try it for Yourself***

*Try to use these simplified operators by yourself* [*in the following exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C1_ex2.ipynb)*.*

*You can find* [*the corrected version right here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C1_sol2.ipynb)*.*

**

*keyboard\_arrow\_down*

# *Learn Python Basics for Data Analysis*

## *Declare a Variable*

1. *Run the following code to print the value of our savings*
2. *Modify the line savings = savings + 500 below the # TO DO comment, using the simplified operator +=*
3. *Run the code again to see the result of the calculation*

*account = 200*

*savings = 3000*

*# TO DO: modify the following line with the simplified operator +=*

*savings = savings + 500*

*print("Your savings are: ", savings)*

*Your savings are: 3500*

*account = 200*

*savings = 3000*

*savings += 500*

*print("Your savings are: ", savings)*

*Your savings are: 3500*

### ***Go Beyond Arithmetic***

*In the example above, all variables are used to store numerical amounts of money (integer or decimal).*

*Are there other types of variables other than numeric ones?*

*Of course! In fact, it is possible to store any type of data in a variable.*

*So far, you have seen that a variable is made up of a name and a value that will automatically define its type. Yet, to return to your warehouse analogy, one can imagine that storing money, storing a book, or storing a car require different size boxes, or even containers. In Python, the storage space of a variable automatically adapts to its contents, almost by magic!*

*At this point, it is safe to assume that having only the ‘amount’ value stored in each of your customer's accounts will probably not be enough. We will most likely also need their name, their interest rate expressed as a percentage, their address, etc., with a new variable for each of these pieces of information. So we would need to store text in our variables.*

*You could declare these three variables as follows:*

*firstName = "Benjamin"*

*interestRate = 1.5*

*address = "15 19 Bloomsbury Way, Holborn, London, WC1A 2TH"*

*Note that you need to use double* ***quotes*** *(") to define strings—textual variables—in Python, otherwise you will get an error! You can however use single (') or double* ***quotes*** *to declare text variables as seen below:*

*address = '15 19 Bloomsbury Way, Holborn, London, WC1A 2TH'*

*# equivalent to*

*address = "15 19 Bloomsbury Way, Holborn, London, WC1A 2TH"*

*You will go into more detail about the different types of variables in Python in the next chapters, but for now you have seen three different types of variables:*

* *Integer variables* ***(int)***
* *Decimal variables* ***(float)***
* *Character strings* ***(string)***

### ***Let's Recap***

*In this chapter, you have learned the basics about variables:*

* *A* ***variable*** *is made up of two parts: its* ***name*** *and its* ***value****.*
* *Assigning a value to a variable is called an* ***assignment****.*
* *The* ***value*** *of a variable can be* ***changed****.*
* *The* ***type*** *of a variable depends on its* ***value****.*
* *The names of your variables should be* ***clear****,* ***explicit****, and should follow a* ***typographic convention****.*

*In the next chapter, you will learn more about variable types!*

## ***2.2. Handle Variable Types***

### ***Understand Why We Need Variable Types***

*You looked at different types of variables in the previous chapter, but there is much more to know about the subject!*

*The types encountered so far are called* ***primitive types****. They exist in Python—a bit like atoms. These are the simplest types of variables; they are the foundation of all computer operations and programs. In the same way that atoms can be combined to make more complex molecules, you can combine primitive types to create much more complex variable types, as you will see in the next parts of this course. For now, you will explore* ***numeric types*** *and* ***strings*** *in a little more depth. Let's go!*

### ***Numeric Variables***

*Numeric variables can be broken down into two distinct types:*

* ***Integers****, which correspond to the set of positive or negative integers (1, 2, 0, 123, -3, etc.)*
* ***Decimals****, which, in addition to integers, include all decimal numbers (2.50, 5.99, -1.20, etc.)*

*Start with the one you are already familiar with:* ***integers****. Integers are declared like any other variable, by associating a value to a variable name.*

*account = 10*

*Here you have the value 10 associated with the variable account. 10 being an integer, accountis automatically an integer variable (int).*

*For decimal numbers , python uses the float type. You can define it in the same way as integers, by simply adding the decimal point explicitly:*

*length = 1876.79*

*width = 870.0*

*As long as the associated value is a decimal number, Python will automatically consider the variable as beingfloat. This is true even if the digit after the decimal point is a 0, as is the case above with the width variable.*

#### ***Mix Several Numeric Variables***

*It is important to keep in mind how the* ***different numeric types can be mixed together*** *and what the potential consequences are. If you mix different types, the most complex will be the one kept for the final result. For example, an integer value can be stored as a float, as seen above with the width variable, but the opposite is not possible if there are numbers after the decimal point! The float is therefore the most complex type: if you mix an int with a float , the result will always be a float , whatever operation is performed or whatever the result is.*

*a = 7.5*

*b = 3*

*c = a/b*

*print(c)*

*# this will print 2.5, which is a float*

*If the result of an operation between two integers is supposed to be a decimal number, Python will automatically convert it to a* ***float****. Moreover, division (even if the result is supposed to be an integer) will necessarily return a* ***float*** *as well:*

*a = 10*

*b = 5*

*c = a/b*

*print(c)*

*# it's a float*

*However, you can force the conversion of a variable into a well-defined type. This is called* ***typecasting****, because by doing so you are changing (****casting****) the type of a variable. To do this, you will need the corresponding functions:*

* *int() : for integers*
* *float() : for decimals*

*a = 14.0*

*# a is a float*

*a = int(a)*

*print(a)*

*# a is now an integer: it prints 14 and not 14.0*

***Try it for Yourself***

*Try to transform the type of our variable* [*in the following exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C2_ex1.ipynb)*.*

*You can find* [*the solution right here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C2_sol1.ipynb)*.*

### ***Character Strings***

*You will now explore a little more about* ***character strings****, which let you store text in your variables. First, a bit of semantics: we call them character strings, because Python does not consider these variables as text, as such, but as a set of characters put together. This is how you can define character strings in Python (you can use either single or double quotes):*

*city = 'New York'*

*film = 'taxi driver'*

*emptyString = ''*

*city = 'New York'*

*film = 'taxi driver'*

*emptyString = ''*

*Character strings are string types in Python.*

*Assembling several strings together is one of the most common operations you will have to perform when using strings—his operation is called* ***concatenation****. See how to do this in Python:*

*favoriteCityOne = "San Francisco"*

*favoriteCityTwo = "New York"*

*favorites = favoriteCityOne + favoriteCityTwo*

*print(favorites) # => "San FranciscoNew York"*

*Note that there is no space between the two. Make your code more readable by concatenating your variables with other strings:*

*favoriteCityOne = "San Francisco"*

*favoriteCityTwo = "New York"*

*favorites = "My favorite cities are " + favoriteCityOne + " and "+ favoriteCityTwo*

*print(favorites) # -> "My favorite cities are San Francisco and New York"*

*It's much better this way, isn't it? However, you cannot concatenate other types of variables with* ***strings****, such as numeric variables---this would return an error. To remedy this, you will need to cast your numeric variable to a* ***string****, via the str() function:*

*city = "Sydney"*

*numberTrips = 5*

*history = "I've been to " + city + " " + str(numberTrips) + " times "*

*print(history) # => "I've been to Sydney 5 times"*

*You have seen that the operator + can have different purposes depending on the types of variables you work with:*

* *With* ***numeric types****, it is used to* ***add****.*
* *With* ***strings****, it is used to* ***concatenate****.*

### ***Let's Recap***

*In this chapter, you have encountered three primitive types of variables, essential to all programs/analyses:*

* *Integers* ***(int)***
* *Decimals* ***(float)***
* *Character strings* ***(string)***

*You have also seen how to use these different types:*

* *You can perform* ***numeric operations*** *on numeric variables of different types.*
* *You can* ***cast*** *variables to force the transformation of the type of a variable into another specific type, according to your needs.*
* *Strings can be assembled together: this is called* ***concatenation****.*

*In the next chapter you will see how to write and use functions. Let's go!*

## ***2.3. Write Your Own Functions***

*Since the beginning of this course, you have used different functions, such as the print() function of the different* ***cast*** *functions like int() or str() . We will now take the time to define what a function is, what it is used for and how you can create your own functions—you are going to find out everything there is to know about functions!*

### ***Discover Functions***

*During your data analysis, you will regularly have to use groups of statements several times for a very specific purpose. One of the fundamental principles for any computer programmer is to* ***get maximum results for minimum effort*** *(there is even a saying that a good programmer is a lazy programmer!). It is thanks to this somewhat "lazy" but very effective principle that the idea of functions came about. Functions* ***can group several statements in a block which will be called using a name****.*

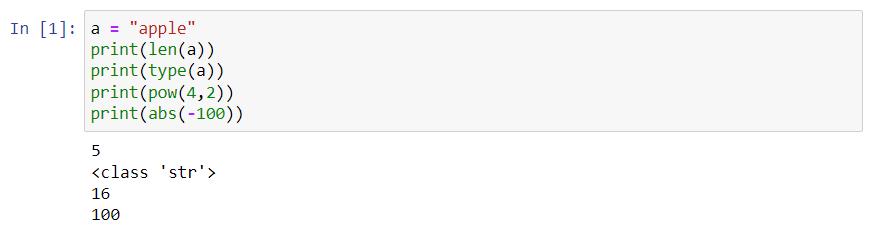
*Functions are not specific to Python; they are present in all computer languages. They can:*

* *reuse a portion of code already written just by stating the function name—so you don't have to rewrite the whole portion of code each time.*
* *simplify code and make it more readable!*

*There are many pre-existing functions in Python! In addition to those already seen, there are, for example:*

* *len() : a function that returns the length of an item. Do you remember strings? Using this function on a string tells you how many characters the string contains.*
* *type() : lets you print the type of a variable.*
* *pow(a, b) : lets you calculate a to the power of b. It is equivalent to writing a\*\*b.*
* *abs() : returns the absolute value of a number.*

*Here are some examples to illustrate the use of these functions:*

*Basic Functions in Python*

### ***Write Your Own Functions***

*Now suppose you are asked to develop a program involving geometry. You have to use a lot of triangles for which you have the length of the three sides, and you want to print their perimeter. You could, of course, do this for each triangle by hand, but remember: you want maximum results for the least amount of effort. So, you are going to create a function that will print the perimeter of a triangle according to the length of its sides!*

*A function is defined via the keyword def followed by the name of the function. All the statements associated with this function will then be written after the colon.*

*def functionName():*

*# statements*

*# that can go*

*# on several*

*# lines*

*Note that all the statements associated with your function are offset in the code: this is called* ***indentation****. Python is a block structured language: each group or block of program must be indented. For example, the set of indented statements following the : will only be available in functionName . Indentation starts the block, and "de-indentation" ends it (the first non-indented line will therefore no longer be included in functionName ).*

***Indentation*** *is a fundamental principle in Python, which you will encounter many times in the future. Generally, it translates into four spaces or a tab.*

*Now define a function that would answer your problem:*

*def printPerimeter():*

*dimension1 = 6*

*dimension2 = 4*

*dimension3 = 3*

*perimeter = dimension1 + dimension2 + dimension3*

*print(perimeter)*

*printPerimeter() # => 13*

*This function is correct, but not entirely useful: not all of your triangles will have the exact same dimensions.*

#### ***Define the Parameters***

*To overcome this limitation, you must make your function accept external numbers. You can do this by defining* ***parameters****.*

*In Python, parameters, just like the name of the function, are defined when the function is written. This is how it would look with the above example:*

*def printPerimeter(dimension1, dimension2, dimension3):*

*perimeter = dimension1 + dimension2 + dimension3*

*print(perimeter)*

*Parameters are variables defined during the declaration of the function, specified inside brackets. Now you can use your function with any existing triangle:*

*printPerimeter(10, 11, 4) # => 25*

*printPerimeter(2, 2, 3.5) # => 7.5*

*Each value is assigned to a parameter, in the order in which they were defined. For example, in the first test:*

* *the variable dimension1 will have a value of 10.*
* *the variable dimension2 will have a value of 11.*
* *the variable dimension3 will have a value of 4.*

*The function will then perform all the operations specified in the body of the function (i.e., the indented lines) with these values.*

*Parameters are variables declared in a function. The values that are passed as parameters are called* ***arguments****.*

*So, that's great, you've added some features to your function!*

*Now, what can I do with the result?*

*Often, when you use a function in a code, you expect an answer that you can reuse to move forward in the code. This answer can be provided via the* ***value returned*** *by a function.*

#### ***Define a Return Value***

*To define a return value, you must explicitly use the* ***return*** *keyword at the end of your function.*

*You could change your printPerimeter function into calculatePerimeter which will return the perimeter of a triangle, according to the length of its three sides, so that it can be reused afterwards:*

*def calculatePerimeter(dimension1, dimension2, dimension3):*

*perimeter = dimension1 + dimension2 + dimension3*

*return perimeter*

*Once you have defined your function, you can use it as many times as necessary:*

*perimeter1 = calculatePerimeter(6, 4, 3)*

*perimeter2 = calculatePerimeter(10, 3, 11)*

*print("The perimeter of my first triangle is", perimeter1, "and that of my second is", perimeter2)*

*And if you analyze these lines, you will realize that every time you use the print function, you send the items to be printed as parameters.*

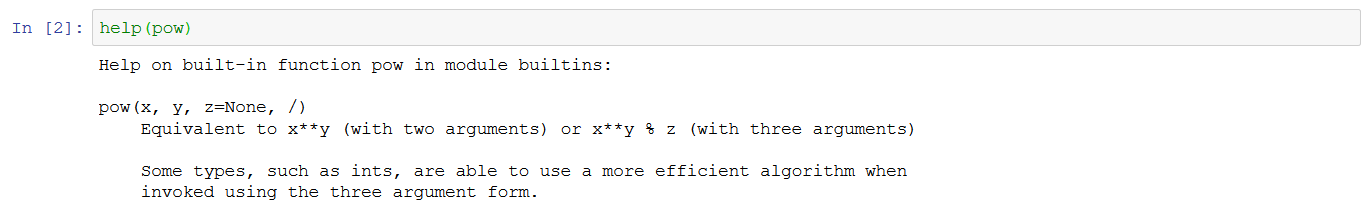
*Note that you can pass several arguments to the print function and the result will be the concatenation of all arguments. By doing so, you don't need to* ***cast*** *the numeric variables to* ***strings****, as the print function will do that for you!*

### ***Use the Help Function if You Forget***

*In your experience in data analysis, you will often remember the name of a function, but not necessarily what it does or its arguments, etc. Don't panic! The help function is there for that! If you run help(functionName) , this will print the documentation of this function, summarizing:*

* *its purpose.*
* *recommendations for use.*
* *the list and description of the parameters.*
* *sometimes even examples.*

*Here is an example with the power function (pow) seen above:*

*Help Documentation for the Pow Function*

*Phew, you've learned a lot in very little time. What progress!*

### ***Let's Recap***

*In this chapter, you have seen that:*

* *functions can have* ***parameters*** *and* ***return values****.*
* *a* ***return value*** *is the result of running the function. The return value is returned to the code that called the function, to be used as needed.*
* *the* ***parameters*** *are the data necessary for a function to be run and generate a result.*
* *parameters are variables defined by a* ***name****. Parameters are* ***specified in the function declaration****.*
* *when using a function, you pass it different values as parameters. These values are called* ***arguments****.*
* *you can use the* ***help*** *function to print the documentation of a given function.*

*In the next chapter, you will discuss the concept of object-oriented programming and what it means for you in Python.*

## ***2.4. Object-Oriented Programming***

*Python is an object-oriented programming language—this means that in Python, everything is an object! In this chapter, you will see what this means and how you can use it in a practical way. Start by looking at a few objects from everyday life, such as pens, books, smartphones, computers, etc.*

*Very different objects!*

*Objects can have many different shapes and characteristics, but you can classify different versions of the same object into a category or group. That's why it's easy to recognize a chair in a store, for example, although its appearance (shape, color, etc.) can vary greatly from one model to another.*

*It is by observing the common points between different objects that you are able, mentally, to classify the objects in the same group or category!*

*For example, there are different types of books, but they all have a title, an author, a back cover, etc. All books share different attributes that let you classify them in a well-identified category: books.*

### ***Classes: Object Models***

*In programming, this concept of a group or category of objects is called* ***a class****. A class can be considered as the construction diagram for an object that will define the characteristics of all objects of this type and their features. From this class, you will be able to create different models of an object.*

*Let's take a concrete example with a* ***Car*** *class. The plan of a car can be defined by:*

* *its characteristics, called* ***attributes****: it must have four wheels, a color, a shape, an engine power, etc.*
* *its functionalities, called* ***methods****: it can drive, brake, etc.*

*So, from this plan, you can create different car models:*

* *An ordinary family car, green, medium power (110 hp)*
* *A sports car, red, relatively powerful (180 hp)*
* *A small blue city car, not very powerful (90 hp)*
* *etc.*

*And no matter what the car model, they are all capable of driving or braking, but not with the same performance!*

*In summary,* ***a class*** *is the outline of an object, defining its* ***attributes*** *and* ***methods****. From the same class, you can therefore create several objects of the same type, but with different attributes—these are called* ***class instances****.*

### ***Focus on Methods***

*As we said before, in Python, everything is an object. This means that, without knowing it, since the beginning of this course, you have been manipulating objects! Consider the following lines of code to illustrate this:*

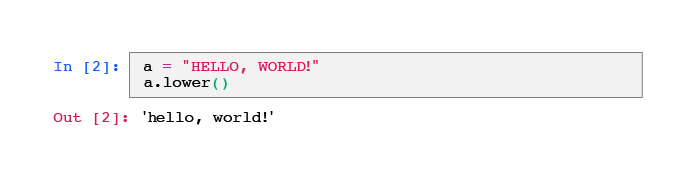
*var1 = 14*

*var2 = 1031*

*Here, you have declared two variables named var1 and var2 containing the values 14 and 1,031. In reality, you have created two instances of the* ***int*** *class, two objects each with* ***a single attribute****: its* ***value****. The same is true for* ***floats*** *or* ***strings****: every time you create a variable of one of these types, you are actually creating objects in Python with the value you assign to them as an attribute.*

*Up until now, we have talked about attributes, so now it is time to see what methods are. A class method is a function that is only available for the instances of this class. If, for example, we consider the Car class presented above having a drive() method, and a Plane class having a fly() method, you will agree quite logically that a plane can’t drive, and a car can’t fly. The same goes for our various objects!*

*The use of a method is always done via the variableName.method() notation. For example,* ***strings*** *have a method called lower() which will transform all the text contained in an object into lower case. Here's how to use it:*

*The string method: lower*

*In the same way as with functions, class methods can take parameters.*

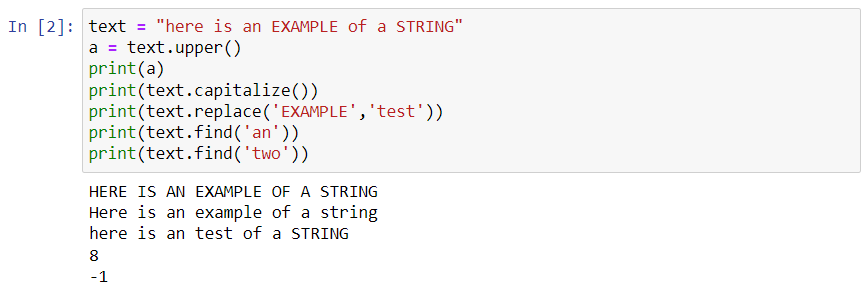
### ***String Methods***

*During the various data analyses that you will have to perform, you will inevitably be confronted with textual variables at some point. You have already seen how to change your string to lowercase, but you may also need to replace some specific words, format the text in a certain way, etc.*

*Python has implemented many methods to allow us to do all this. Here are the most common ones:*

* *upper() : returns the whole text in upper case.*
* *capitalize() : returns the whole text in lowercase with the first letter capitalized.*
* *replace(old, new) : this method takes two arguments: old and new, both of which are strings. The method returns the original string with all occurrences of old replaced with new.*
* *find(string) returns either the index of the first occurrence of the string passed in the argument, or -1 if it does not find it.*

*Here are some examples of how these methods are used:*

*String Methods*

*As you can see here, especially with the lines concerning the a variable, the methods seen above do not modify the initial object! They only return the result of the method applied to the object. You will regularly have to reassign this result to the initial variable, when you want to modify it directly.*

#### ***Try it for Yourself***

*Manipulate objects yourself* [*in the following exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C4_ex.ipynb)*.*

*You can find* [*the solution here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P2C4_sol.ipynb)*.*

### ***Let's Recap***

* *A* ***class*** *is a construction plan for an object.*
* *A variable is an* ***instance of a class****, or* ***an object****.*
* *An object is defined by its* ***attributes****.*
* *All instances of a class have access to the same* ***methods*** *via the. (dot) notation.*
* *A method, like a function, generally does not modify the initial object.*

*In the next part, we will see in more detail how to organize your code via different structures and complex objects.*

# ***Quiz***

# ***2.5. Handle Fundamental Functions and Objects***

### ***Evaluated skills***

* *Handle fundamental functions and objects*
* ***Question 1  
  What components define a variable in Python programming?***
  + *Its name and value*
* *The “official” names for the components are name and value. The type is automatically defined by the value!*
* ***Question 2  
  Which of the following statements about variable names is correct?****Careful, there are several correct answers.*
  + *The name of a variable can start with an underscore.*
  + *Keywords cannot be used as variable names.*
* *Variable names can start with underscores, but not with numbers. They cannot under any circumstances contain special characters (@, #, $, etc.).*
* *Since keywords are reserved in Python, it is impossible to use them for anything other than what they were created for (including naming a variable with one).*
* ***Question 3  
  Which of the items in the list below is not a primitive type/object in Python?***
  + *decimal*

*int and float are numeric types. string corresponds to strings of characters. Decimal numbers are represented by float in Python, the decimal type does not exist.*

* ***Question 4  
  What values do the x and y variables have when the following code is executed?***

*x = 10*

*x += 12*

*y = int(x/4)*

*x = x + y*

* + *x = 27, y = 5*

*Here is the step-by-step procedure:*

1. *x = 10*
2. *x = x + 12 = 10 + 12 = 22*
3. *y = int(x/4) = int(22/4) = int(5.5) = 5: the integer transformation deletes the digits after the decimal point.*
4. *x = 22 + 5 = 27*

*So x = 27 and y = 5.*

* ***Question 5  
  Which of the following options concatenates two strings?***
  + *The + operator*

*Only the+ operator can concatenate several strings together. For example, a = 'a' + 'concatenate' + 'string' will store 'concatenated string' in a .*

* ***Question 6  
  Which of the following statements is true?***
  + *A function is a block of code that is written to perform a specific task.*
* *The keyword for declaring new types of objects is class, different from the one for creating a function (def). A function does not necessarily make our program run faster, although it can in some cases!*
* ***Question 7  
  Which of the following must be present when creating a function?****Careful, there are several correct answers.*
  + *The name of the function (followed by brackets)*
  + *The keyword def*
* *To define a function, you must specify the function creation keyword (def), followed by the name of the function, followed by brackets. However, it is not necessary for it to have parameters or to return a value.*
* ***Question 8  
  Which of the following statements are true?****Careful, there are several correct answers.*
  + *Methods are a kind of function only available for a given object.*
  + *You can use a method via the . operator.*

*Methods are available/possible for any type/object. Methods can return an object or a value or change the original object.*

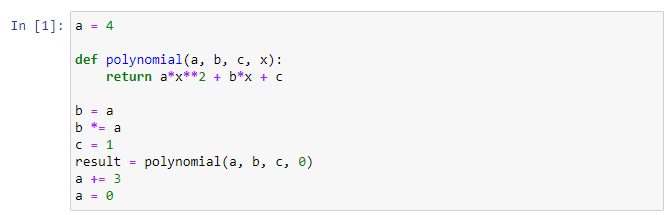
* ***Question 9  
  You have the following a variable:***
* *a = "This is A TEST"*
* ***Which code among those proposed can have the following output?  
  This is a test***
  + *a.capitalize()*

*upper: returns the string in upper case.*

*lower: returns the string in lower case.*

*capitalize: all in lowercase, except the first letter.*

*firstCapitalize does not exist!*

* ***Question 10  
  The following code is used:  
    
  What are the values of result, a, b and c once the code is run?***
  + *result = 1, a = 0, b = 16, c = 1*

*Sequence of operations:*

1. *a = 4*
2. *b = a = 4*
3. ***b*** *\*= a equivalent to b = b \* a = 4 \* 4 =* ***16***
4. ***c = 1***
5. ***result =*** *polynomial(4, 16, 1, 0) = 4 \* (0\*0) + 16 \* 0 + 1 = 0 + 0 + 1* ***= 1***
6. *a + = 3 equivalent to a = a + 3 = 4 + 3 = 7*
7. ***a = 0***

*So we have:*

*result = 1, a = 0, b = 16, c = 1*

## ***3.1. Create Collections to Store Your Objects***

*Imagine again that you work in a bank. What you saw in part two was fine for treating each customer individually, but in reality, there's more than just one customer to analyze.*

*If you want to analyze several customers at once, you can imagine that you need a variable for each customer. For the names, this could look like this:*

*customerName1 = 'Marion Weaver'*

*customerName2 = 'Alberto Mendoza'*

*customerName3 = 'Katharine Tyler'*

*customerName4 = 'Isaac Steele'*

*# etc.*

*If you have 10 customers to analyze, wouldn't it be easier to store them all in* ***a single variable that would contain all the information****?*

*You're in luck! Python offers a structure, a* ***class*** *capable of storing multiple pieces of information as a kind of array. This structure is called a* ***list****. Let's see how to use it.*

### ***Declare a List to Store Your Items***

*Lists are objects that can contain a collection of objects of any type. We can have a list containing several integers (1, 2, 50, 2,000 or more, it doesn't matter), a list containing floats, a list containing strings, or even a list mixing objects of different types.*

*Lists are* ***ordered objects****, i.e., each item of the list is associated with a number corresponding to its order in the list. This number is called an* ***index*** *and it starts at 0 (not 1!). The first item is therefore associated with index 0, the second with index 1, etc.*

*Declaring a list is quite similar to the declaration of any variable seen so far: via a* ***name*** *to which we associate* ***a list of items to be stored*** *in this name.*

*For example, here is the list containing the names of four customers:*

*customerName = ['Marion Weaver', 'Alberto Mendoza', 'Katharine Tyler', 'Isaac Steele']*

*Now that your list is created, you can perform two basic operations:*

* ***Access*** *a value at a given index*
* ***Change*** *the value at a given index*

*In both cases, the code consists of the name of the variable followed by [, the value of the index and ].*

*For example, if you made a mistake on the name of the first customer and you want to correct their name:*

*# assign the value 'Marianne Weaver' to the first name in our list*

*# it is index 0, because indices start at 0 in python!*

*customerName[0] = 'Marianne Weaver'*

*To print it, you can write the following line:*

*print(customerName[0])*

*Python also lets you use negative indices to access or modify an item. The index -1 corresponds to the last item of the list, -2 to the second last, and so on. You can also access an index range by using the : operator. For example, 1:3 will let you access items two to four.*

*# print the last item*

*print(customerName[-1])*

*# access the second item to the 3rd*

*print(customerName[1:3])*

*# access all items from the beginning to the second*

*print(customerName[:2])*

*Here you have manipulated lists of strings, but you can do the same thing with the amount in each individual's account:*

*amountAccount = [10000, 150, 300, 1800.74]*

*You will probably have noticed, but in our last list, the first three items are integers, while the last one is a decimal. As we said before, you can store several different objects in the same list.*

*For example, the following list is completely valid:*

*strangeList = [4, 10.2, 'Marion Weaver', ['another list', 1]]*

*# print the 4th item of the list*

*print(strangeList[3])*

#### ***Try it for Yourself***

*Work with lists yourself* [*in the following exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C1_ex1.ipynb)*.*

*You can find* [*the solution here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C1_sol1.ipynb)*.*

*What should we do now if a new customer is added to our analysis? When I try to do customerName[4] = '...', it returns an error!*

*Don't panic! You can't access an index that doesn't already exist in a list in Python... however, lists have many methods that let you remedy this.*

### ***List Methods***

*Now consider that you want to list animals in order of cuteness (from cutest to "least" cute). We can easily start with a list of four animals: fox, koala, owl, and otter. These days, while browsing the internet, you will inevitably come across this image:*

**

*This one deserves a place at the top, right?*

*And now you want to add the cat to the first position!*

*The good news for our little cat is that the lists are fully editable, whether it's the number of items, their order, etc. Thanks to the different list methods, we can:*

* ***search for*** *a specific item in the list.*
* ***add*** *a new item at the end.*
* ***insert*** *a new item at a specific index.*
* ***delete*** *an item from the list.*

#### ***Add Items to a List***

*You can create an empty list in Python and then add the items one by one via the* ***append*** *method:*

*list = []*

*list.append(7)*

*list.append(5)*

*print(list) # => [7, 5]*

1. *The first statement creates an empty list, very creatively named list.*
2. *You then add the integer 7 to the end of the list. So Python will add it to index 0.*
3. *Finally, you add the integer 5, which will be stored at the next index, which is index 1.*

*You will notice that you did not need to write list = list.append(...). Unlike the string methods seen previously, this one does modify the original object.*

*Here are some other methods that are essential to know about lists:*

* *insert to insert a new item at a specific position. For example, list.insert(1, 12) will insert the integer 12 at index 1, moving the old item 1 to index 2 and so on.*
* *extend : similar to append, but with another list. This allows you to concatenate several lists together.*
* *remove : searches for the given item in the list and deletes the first occurrence. For example, if you want to delete 5 from your list, you can use : list.remove(5) .*
* *index : this method lets you find the index of the first occurrence of an item to be searched for in our list;*
* *Keyword del to delete an item according to its index.*

*Now let's try out some of these methods:*

*list = []*

*list.append(7) # -> [7]*

*list.append(5) # -> [7, 5]*

*list.insert(1,12) # [7, 12, 5]*

*list[0] = 4 # -> [4, 12, 5]*

*list.remove(12) # [4, 5]*

*list.index(5) # prints 1*

*list.extend([1, 2, 3]) # [4, 5, 1, 2, 3]*

*del list[3] # [4, 5, 1, 3]*

*Let's break down these few lines:*

* *The first three lines correspond to what was seen before.*
* *You then add 12 to index 1. The value that was in position 1 is moved to position 2.*
* *You then replace the value at index 0 with 4.*
* *With the .remove() method, you remove the integer 12 from our list.*
* *You then ask for the index of the first item 5 in our list (here in second position, so return 1).*
* *You add the list [1, 2, 3] after our initial list.*
* *And, finally, you delete the item located at position 4 in our list.*

*In the end, this leaves you with the final list: [4, 5, 1, 3] .*

#### ***Keep Control of Your List***

*The len() function lets you retrieve the size of your list:*

*list = [1, 2, 3]*

*len(list) # will print 3*

*The len function is used a lot, especially when you need to scan the different items in a list with a* ***loop****, as you will see in the next chapter!*

#### ***Try it for Yourself***

*Take a more in-depth approach to working with lists* [*in the following exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C1_ex2.ipynb)*.*

*The solution is* [*right here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C1_sol2.ipynb)*.*

### ***Use Dictionaries***

*Now, let's come back to our problem with the bank's customer names and associated accounts. With the above method, you would need two lists - a list of customer names and a list of account balances. Each time a new person is added to our data, their name and bank account balance would be added to the corresponding lists.*

*Dictionaries are another type of object, similar to lists, but which will let you do this with a single variable! Indeed, a* ***dictionary*** *is a list of items organized via* ***a system of keys****. With a real dictionary, you look up a word to access its definition. In programming, this word corresponds to the key and the definition to the value associated with it. This is called a* ***key-value*** *pair. So, we could have:*

| ***Marion Weaver*** | ***Alberto Mendoza*** | ***Katharine Tyler*** | ***Isaac Steele*** |
| --- | --- | --- | --- |
| *10000* | *150* | *300* | *1800.74* |

*Here, the first row represents the keys and the second row represents the corresponding values.*

***Each key*** *in a dictionary must be* ***unique****. Strings are generally used to define keys, but this is not a requirement, per se.*

#### ***Declare a Dictionary***

*Lists and dictionaries are declared in a similar way, except that a dictionary uses* ***curly brackets*** *instead of square brackets, and key-value pairs must be declared:*

*accounts = {'Marion Weaver': 10000, 'Alberto Mendoza': 150, 'Katharine Tyler': 300, 'Isaac Steele': 1800.74}*

*print(accounts['Alberto Mendoza']) # -> 150*

*The last line will print the value associated with the key "Alberto Mendoza" which is 150.*

#### ***Manipulate the Items of a Dictionary***

*Here are the operations frequently carried out with dictionaries:*

* ***Access*** *the value of an item*
* ***Add*** *a new item (a new key-value pair)*
* ***Delete*** *an item via its key*

*A value can be accessed or modified using the same notation as with lists. With dictionaries, unlike lists, this notation even lets you add items.*

*Let's see this in action in the following example:*

*accounts['Marion Weaver'] -= 2000 # I subtract 2000 from David's account*

*accounts['Kristian Roach'] = 1000 # I add a new individual in my dictionary*

*print(accounts['Kristian Roach']) # I print the value of Kristian's account*

*Finally, you can delete an item via the pop() method by specifying the key of the item you want to delete.*

*accounts.pop('Alberto Mendoza') # deletes Alberto Mendoza from our dictionary*

*Finally, in the same way as with lists, you can use the len() function to see how your dictionary grows in size:*

*len(accounts) # -> 4*

### ***Understand Immutable Tuples***

*The last type of collection we will look at are tuples. These are very similar to lists:*

* *They are* ***ordered objects****, so we can access the different items stored in a tuple from their index.*
* *You can store any kind of object in a tuple.*

*The main difference is that once a* ***tuple has been declared, it cannot be modified.*** *It is then said that it is* ***immutable****.*

*We can't modify a tuple? So what's the point of it?!*

*They might not seem to provide much benefit at first sight, but they can be used:*

* *when you want to make sure that data is not modified within a program.*
* *to return several values from a function. Indeed, we didn't address this point when we talked about functions, but it is possible to return several values... with a tuple!*
* *to declare several variables in one line.*

#### ***Declare a Tuple***

*Tuples are declared in a very similar way to lists, except* ***parentheses*** *are used instead of square brackets:*

*my\_tuple = (1, 2, 3, 'a', 'b')*

#### ***Manipulate Tuples***

*As explained above, tuples are ordered objects, so we can use indices to select the items of a tuple:*

*print(my\_tuple[1]) # -> 2*

*print(my\_tuple[4]) # -> 'b'*

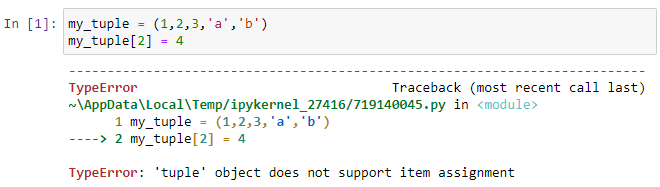
*You can also declare several variables at the same time from a tuple:*

*a, b = (1, 'apple')*

*print(a) # -> 1*

*print(b) # -> 'apple'*

*However, you will get an error if you try to modify your tuple in any way, as you can see below.*

*Attempt to modify a tuple—failed!*

### ***Let's Recap***

*In this chapter, you have learned all the basics of using Python’s different built-in "collection types":*

* ***Lists****: an* ***ordered****,* ***editable*** *collection where each item is associated with an* ***index***
* ***Dictionaries:*** *an* ***unordered****,* ***editable*** *collection where each item is associated with a* ***key***
* ***Tuples****: an* ***ordered****,* ***non-mutable*** *collection , where each item is associated with an* ***index***
* *The most common actions performed with lists and dictionaries are:*
  + ***Access*** *an item*
  + ***Add*** *an item*
  + ***Delete*** *an item*
  + ***Modify*** *an item*
  + ***Count*** *the number of items stored*
* *We can perform these different actions via* ***methods****.*

*The type to use depends on the task at hand. As you progress in your career, you will be able to better identify the structure that is best suited to your situation!*

*We will now see how to organize our code via conditional structures.*

## ***3.2. Control Your Code With Conditional Structures***

*As you work on more and more complex projects, writing a set of lines that run in sequence will not be enough! This is where conditional structures come into play.*

*In our very first chapters, you saw how to say "Hello, world." Wouldn't it be better to modify this program slightly to be a little more specific and say hello to a particular person?*

### ***Print Information if Available***

*When you start your program, you do not necessarily know the name of the user in advance. How about a program that can:*

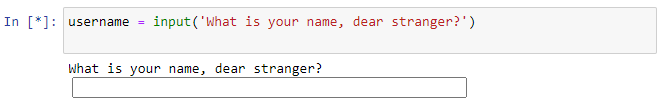
* *say hello to a particular user,* ***if*** *you know their name...*
* ***if not****, continue to say hello to everyone?*

*Here is your first* ***condition****, which will let you build your first conditional structure.*

*How can you get a person's name in the first place?*

*Do you remember functions? Yes, you worked it out, you can do this via a function: the input function. This will ask the notebook user (in other words... you!) to enter a string which will then be stored in a variable.*

*Let's look at an example:*

*Input function with Jupyter!*

*You have a space to answer the question and your answer will be stored as a string in the username variable.*

*Now design the code that will let you say hello to your user:*

1. *Ask the user their name and store their answer in a variable: name .*
2. *Check that the name variable definitely contains a value (in case the user doesn't give an answer). The len function will help you to perform this task!*
3. ***If*** *this is the case, say Hello to your user with their name.*
4. ***Otherwise****, keep saying Hello to the world.*

*Here is the corresponding code:*

*name = input( 'What is your name, dear stranger?')*

*if len(name) > 0:*

*print("Hello", name, "!")*

*else:*

*print("Hello, world!")*

*It works well! Everything indented below the* ***if*** *is executed if* ***the condition is true****,* ***otherwise*** *the program runs everything indented below the* ***else****.*

*As you can see in the example above, the block construction using the: operator and* ***indentation*** *are essential and omnipresent concepts in Python! Be very careful about the organization of your code at this level, to avoid some lines being run when they should not be, and vice versa.*

*Let's take a closer look at how the* ***if*** *structure works in practice.*

### ***Use Booleans: The No-Half-Measure Type***

*In Python, to validate a condition, you use a special type (or object :) ) called* ***boolean****. A Boolean variable can only contain two values:* ***True*** *or* ***False****. It's actually a pretty simple thing, but oh so useful!*

*In Python, the boolean can take the values* ***True*** *and* ***False****. Now see how to declare booleans in Python:*

*thisCourseIsGreat = True*

*itsAuthorIsVeryHumble = False*

*Easy, right?*

*To get back to your* ***if*** *conditional structure, I think you'll understand, but it's absolutely necessary that* ***what follows the if*** *keyword* ***results in a boolean****. This can be done via:*

* *a True or False value. For example, if True : .*
* *a boolean variable. For example, if myVariable: where myVariable is a boolean.*
* *an* ***expression*** *that results in a boolean value, as in the example above.*

*For example:*

*weather = "The weather is great!"*

*weather.startswith("The weather") # -> True*

*startswith is a method of the* ***string*** *class, which returns True when the string starts exactly with the string passed as a parameter; False , if not. For example, you could use this expression to perform an action if a sentence begins with a particular word.*

*To produce Booleans, you can also use* ***comparison operators****.*

#### ***Comparison Operators***

*As the name suggests, comparison operators are used to compare two values. There are six main ones:*

* *== equal to (two values are exactly the same)*
* *!= different from*
* *< less than*
* *<= less than or equal to*
* *> greater than*
* *>= greater than or equal to*

*Here are some examples with numeric variables:*

*2 == 2 # -> True*

*2 == 3 # -> False*

*4 != 4 # -> False*

*4!= 5 # -> True*

*1 < 2 # -> True*

*1 < 1 # -> False*

*1 <= 1 # -> True*

*3 > 4 # -> False*

*5 > 4 # -> True*

*5 >= 4 # -> True*

*The result of these operations can be assigned to a variable:*

*age=15*

*if age>=21:*

*# Do something if age is greater than or equal to 21*

*Sometimes, you will need more elaborate conditions, where the condition will be the result of combining several expressions. This is where* ***logical operators*** *come in.*

#### ***Logical Operators***

*These operators will let you mix several Boolean values: specific Boolean values or expression results. There are three of them:*

* *and : the* ***AND*** *operator.  
  The final result is true only when all expressions/values are true. For example: the result of expression1 and expression2 will be True only if expression1 is true* ***AND*** *expression2 is also true.*
* *or : the* ***OR*** *operator.  
  The final result is true when at least one of the expressions/values is true. For example: the result of expression1 or expression2 will be at True if expression1 is true* ***OR*** *expression2 is true.*
* *not : the* ***NOT*** *operator.  
  This simply reverses the result of the given expression. For example, the result of not(expression) is true when expression is false.*

*Here are some examples with the results shown as comments:*

*True and True # True*

*True and False # False*

*False and False # False*

*True or False # True*

*True or True # True*

*False or False # False*

*not(True) # False*

*not(False) # True*

*You can also mix more than two expressions/values:*

*True and True and True # True*

*True and True and False # False*

*True or False or False # True*

*False or False or False # False*

*As with numeric operations, logical operators respect the priorities of operations: the not operator is applied first, then the and operator, then the or operator. For example:*

*False or True and True # True*

*not(False) and True or False # True*

*You can also use parentheses to change the order:*

*(True and False) or True # True*

*not(True and False or not(True)) # True*

*The general form of a conditional* ***if*** *structure is if condition: where the* ***condition*** *can be either a boolean, or a variable of boolean type, or the result of an expression leading to a boolean result.*

#### ***The in Operator***

*Another useful logical operator in Python is the in operator. This returns True when a value is found in a sequence (a string or a list); False , if not.*

*For example:*

*myList = [4, 2, 3, 2, 10]*

*myStringList = ["a", "b", "c", "d"]*

*myString = "The weather is really good today!"*

*4 in myList # True*

*0 in myList # False*

*0 in myStringList # False*

*"c" in myStringList # True*

*"e" in myStringList # False*

*"weather" in myString # True*

*"really" in myString # True*

*"rain?" in myString # False*

*In your "Hello, world!" example, you have defined* ***only one alternative****. What if you have* ***more than one alternative****?*

### ***Manage a Chain of Conditions***

*To grant a loan, a bank relies (among other things) on the status of its users' accounts. For example, a naive decision rule might be:*

* *If the customer has more than $10,000 in their account, they are automatically approved for their loan.*
* *If they have between $100 and $10,000, we need to manually approve their application.*
* *Otherwise, the request is denied.*

*We could use two nested* ***if*** *statements, but Python can link several conditions thanks to the keyword elif (contraction of else and if). Here is the general form:*

*if condition1:*

*# instructions*

*elif condition2:*

*# instructions*

*else:*

*# instructions*

*Here is the code corresponding to the example presented above:*

*account = input("What is your account balance?")*

*account = int(account) # transform the answer into an integer*

*if account >= 10000:*

*print("Loan granted!")*

*elif account >= 100 and account < 10000:*

*print("Loan in process of validation: under study")*

*else:*

*print("Loan refused")*

#### ***Try it for Yourself***

*Use conditional structures* [*in the following exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C2_ex.ipynb)*.*

*You can find* [*the solution right here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C2_sol.ipynb)*.*

### ***Let's Recap***

* *Conditions let you execute a block of code when a* ***Boolean****,* ***variable****, or* ***expression*** *is true (True).*
* *Expressions use* ***Boolean arithmetic****, including* ***logical operators*** *and* ***comparison operators****.*
* *You can apply several conditions with* ***if/elif/else*** *chains.*

*In the next chapter, you will see another way to control the code via* ***loops****.*

## ***3.3. Choose the Appropriate Loop***

*Imagine that you have a block of code that you want to repeat several times in succession. You can of course store it in a function and call this function as many times as necessary. This would work, but it would be a bit like killing a fly with a bazooka: it works, but it is not necessarily the most optimal solution. Especially since, generally, we do not necessarily know in advance how many times we will need to repeat the said block of code.*

*Loops* ***solve this problem****! In programming, a loop is a structure that lets you repeat one or more statements, without having to rewrite them each time. There are two types of loops (****for*** *and* ***while****), which will be explained shortly.*

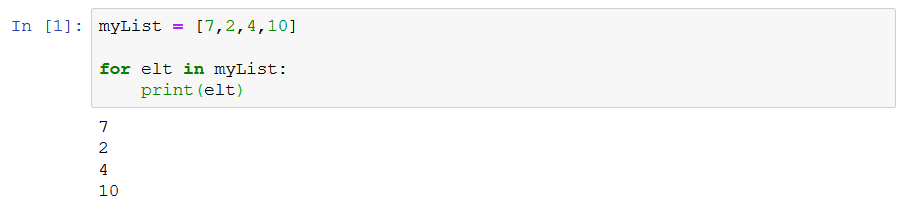
### ***Loop a Set Number of Times With the for Loop***

***For*** *loops are used when you know in advance how many times an action will be repeated.*

*However, unlike most other languages, in Python a* ***for*** *loop will necessarily* ***iterate through a collection*** *(list, dictionary, string, etc.).*

#### ***The for Loop on a Collection***

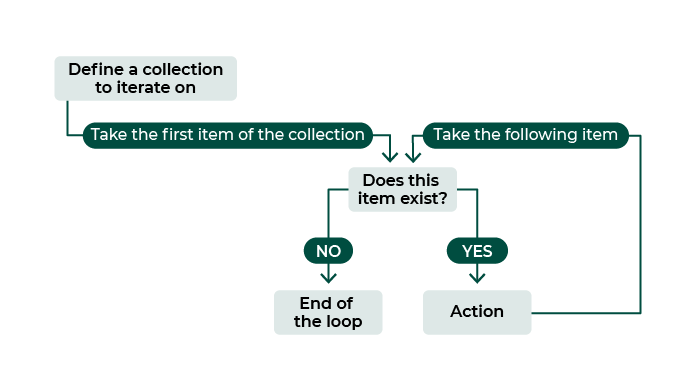
#### *The conventional use of loops in Python is to directly use the different values of a collection. Here is an example with a list:*

*Scan a list via a loop*

*The printed result corresponds to each item in the list taken one by one. Let's take a detailed look at what has been achieved in the above code:*

1. *You have created a list: myList, containing four items: 7, 2, 4, and 10.*
2. *The loop will store the first value of the list (in this case 7) in the elt variable.*
3. *Then, the whole block of code associated with the* ***for*** *loop (defined via indentation... again!) is run with elt holding the first value. Here, this block just involves printing elt.*
4. *Once this is done, elt will take the second value of the list (in this case 2) and the block of code is re-run.*
5. *The loop will continue until all the values in your list have been stored in the elt variable and the statements for the loop have been executed for each of them.*

*Here is a diagram to understand the Python logic behind the loop:*

*Understand the logic of the loop!*

*You can also iterate via a* ***string****! Remember that* ***strings*** *are also known as "character strings" because they represent a collection of characters, similar to a list.*

*myString = "Items"*

*for elt in myString:*

*print(elt)*

*In this case, elt will successively take each character of your string.*

***The for Loop via an Iterative Integer Value***

*Quite often, you will find that you simply need to loop over a range of integer values, e.g. 0, 1, 2, 3…. This is the conventional loop you see most often in languages like Javascript or C++. Because Python wants to loop over a collection, you need to create a collection containing your range of integer values.*

*To do this, you will use the range(start, stop, step) function, which will generate a collection of numbers according to three parameters:*

* *start : the first number of the sequence.*
* *stop corresponds to the last number of the sequence, non-inclusive. The function will generate numbers from start to stop-1.*
* *step : the step between each generated number.*

*Not all parameters are necessary. For example:*

*for i in range(0, 5, 1):*

*for i in range(0, 5, 1):*

*print(i) # -> print from 0 to 4 by steps of 1 (end - 1)*

*for i in range(0, 5):*

*print(i) # -> prints from 0 to 4 also (default step is 1)*

*for i in range(5):*

*print(i) # -> prints from 0 to 4 also (default start is 0)*

*for i in range(0, 5, 2):*

*print(i) # -> print 0, 2 then 4*

*The iterative variable can take any name. When iterating over an integer, we usually use values like i , j , or k. Otherwise, it is better to use explicit names, as seen above ( elt being the abbreviation for item).*

*The* ***for loop*** *is perfectly suited when you have to perform an action a certain number of times known in advance or an action for each item of a collection. For all other cases, we can make a* ***conditional loop****: a loop that does not iterate through a collection, but according to a condition.*

### ***Loop According to a Condition with the while Loop:***

*The conditional loop is the* ***while*** *loop in Python.*

*As its name implies, the* ***while*** *loop will run* ***as long as a condition is met****. It is a kind of combination of a* ***for loop*** *and an* ***if structure****. The number of repetitions is not defined in advance, but via a condition to be fulfilled, as with an* ***if****. This is called a* ***conditional loop****.*

*The syntax is as follows:*

*while expressionLogic:*

*# block to execute*

*It can be interpreted as: as long as my logical expression is true, run the statement block.*

*Here's how it works:*

1. *The program checks that expressionLogic is equal to True.*
2. *If this is the case, the indented statements following the : are run. Once this is done, we return to step one.*
3. *Otherwise, the program exits the loop without running the statements.*

*Try the example below:*

*numberTrees = 0*

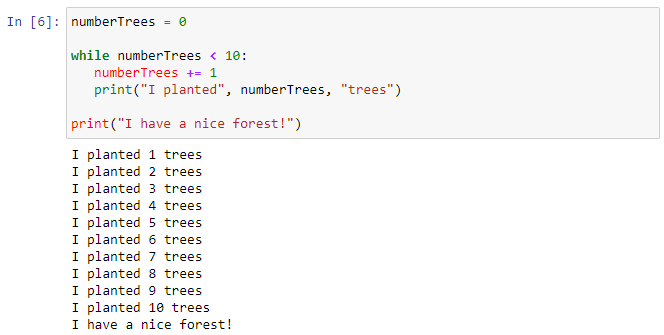
*while numberTrees < 10:*

*numberTrees += 1*

*print("I planted", numberTrees, "trees")*

*print("I have a nice forest!")*

*This will produce the following result:*

**

*Plant a forest!*

*With each iteration, the numberTrees is incremented by one. When the variable reaches the value 10, the expression numberTrees < 10 is no longer true! At this point, the loop ends and continues running the rest of the program in order. In this specific case, it prints: "I have a cool forest!"*

*❗️ It is essential to keep in mind that a misused* ***while*** *loop can crash your program! ☠️ While the condition is still true, the program remains stuck in a loop. If the condition never becomes false, the program will never exit the loop. We are in what is called, in programming, an* ***infinite loop****.*

*Here is an example not to be reproduced at home (this one was made by a professional...):*

*theSunIsShining = True*

*while theSunIsShining:*

*print("Stay awake... forever!")*

*# the theSunIsShining never changes, so the condition is always true*

*# so we never reach this line*

*print("Time to go to sleep!")*

*This is a common mistake and unfortunately it can happen very easily. So* ***be careful****!*

#### ***Try it for Yourself***

*Run a few loops yourself* [*in the next exercise*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C3_ex.ipynb)*.*

*You can find* [*the solution here*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P3C3_sol.ipynb)*.*

### ***Skip Some Statements Within Your Loop***

*Regardless of the type of loop, there will be situations where you will want to skip some of the iterations within your loop, or even terminate the loop prematurely.*

*For example, you want to repeat something 10 times, but skip (at least partially) when the value is 2 or 5. In Python, to force the start of the next loop iteration, use the keyword continue:*

*for i in range(10):*

*# statements executed at each iteration*

*print(i)*

*if (i == 2) or (i == 5):*

*print("Special case")*

*continue*

*# statements not executed if i == 2 or 5*

*print("i != 2 & i != 5")*

*You can also decide to interrupt the loop, for example when looking for a particular item in a list. For this, you will use the break keyword.*

*basket = ["apple", "orange", "banana"]*

*for fruit in basket:*

*if fruit == "orange":*

*print("I have an orange!")*

*break*

*Once the fruit has been found in your basket, you finish the loop.*

### ***Let's Recap***

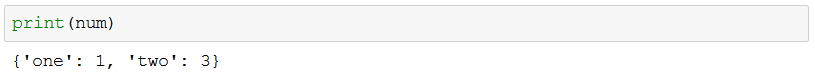
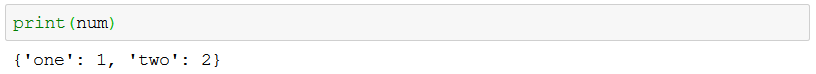
*In this chapter, you have discovered two types of loops:*

* *The loop for repeating an action a certain number of times, or according to a sequence: the* ***for loop****.*
* *The loop that allows you to repeat an action as long as a condition is true: the* ***while loop****.*
* *There is a common mistake that you must not make with the while loop: the* ***infinite loop****!*
* *You can choose to skip certain loop iterations via the* ***continue*** *keyword.*
* *The cycles of the loop can be interrupted via the* ***break*** *command.*

# ***Quiz***

# ***3.4. Structure Your Code Within a Project***

### ***Evaluated skills***

* *Structure your code within a project*
* ***Question 1  
  How can you change the following num variable:  
    
  Into:  
  ***
  + *num['two'] = 2*

*The num variable is a dictionary. To obtain the second result, you must change the value associated with the key “two” to 2, num['two'] = 2.*

* ***Question 2  
  Which of the following four statements about lists in Python is true?***
  + *A list has no conceptual size limit!*

*The items of a list can be of different types, including list type! Moreover, two lists are equivalent only if they contain the exact same values!*

*The right answer was that a list has no conceptual size limit: theoretically, you could add as many items as you want!*

***Question 3  
Which of the following statements is false about collections?***

* + *A dictionary is not mutable.*

*As a reminder:*

* *A* ***list*** *is an* ***ordered****,* ***editable*** *array, where each item is associated with an* ***index****.*
* *A dictionary is an* ***unordered****,* ***editable*** *array where each item is associated with a* ***key****.*
* *A* ***tuple*** *is an* ***ordered****,* ***non-mutable*** *array, where each item is associated with an* ***index****.*

*A dictionary is therefore modifiable: it is mutable.*

* ***Question 4  
  The following codes concern different actions that are possible with lists. Select the blocks containing correct actions from those below:****Careful, there are several correct answers.*

*colors = []*

*colors.append('Red')*

*colors.append('Green')*

*colors.append('Yellow')*

*print(colors)*

*frequency = ['sunday', 'monday', 'tuesday','wednesday']*

*frequency.remove('monday')*

***To add an item to a list, we use the .append method.***

***.insert just lets you insert an item in a fixed index (but you have to specify the index... for example: colors.insert(0, 'Red') would be correct!).***

***In addition, the code list[index] = ... is only used to modify an existing item.***

***.remove deletes the selected item from the list.***

***Question 5  
How do you mark the end of a block of statements in Python?***

* + *A line that is less indented than the previous one.*

*In Python, only indentation can mark out the different blocks of code. So, the end of a block of code is marked by a line less indented than the previous one.*

* ***Question 6  
  What is the output of the following code?***

*if 'bar' in {'foo': 1, 'bar': 2, 'baz': 3}:*

*print(1)*

*print(2)*

*if 'a' in 'qux':*

*print(3)*

*print(4)*

* + **

***bar is in the dictionary, so we enter the first conditional block:***

1. ***print 1***
2. ***print 2***

***a is not in the string qux, so we do not enter the second conditional block. So it does not print 3.***

***4 will always be printed, as it is outside the loop.***

***The final print is therefore:***

******

* ***Question 7  
  What will happen with the following code?***

*d = {'a': 0, 'b': 1, 'c': 0}*

*if d['a'] > 0:*

*print('yeah!')*

*elif d['b'] > 0:*

*print('yeah!')*

*elif d['c'] > 0:*

*print('ok')*

*elif d['d'] > 0:*

*print('ok')*

*else:*

*print('not ok')*

* + *It will print yeah! .*

*There is a potential error in the code, because the key* ***d*** *does not exist. Nevertheless, because the program follows the conditions in order:*

1. *it tests if the value associated with the key* ***a*** *is greater than 0: it is not the case, so it goes to the next condition.*
2. *it tests if the value associated with the key* ***b*** *is greater than 0: it is the case, so it runs the corresponding statements (print “yeah!”) and leaves the conditional structure completely.*

*The program therefore never reaches the line where the error is located.*

* ***Question 8  
  The while loop is primarily used when:***
  + *the number of iterations is known.*
  + *you want to repeat a block of code several times.*
  + *the number of iterations is not necessarily known.*
  + *we want to perform an infinite loop.*
* ***Question 9  
  What does the following code print?***

*a = ['foo', 'bar', 'baz', 'qux', 'corge']*

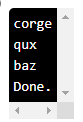
*while a:*

*if len(a) < 3:*

*break*

*print(a.pop())*

*print('Done.')*

* + **

*This code is a bit unusual:*

* *The while a: will execute the loop as long as a is not empty.*
* *Then, we add a condition to stop the loop as soon as the length of the list a is less than three.*
* *In all cases, at each loop, the last item of the list is removed (through a.pop()) and printed.*

*So, if we create the loop:*

1. *len(a) = 5, we pop and print corge*
2. *len(a) = 4, we pop and print qux*
3. *len(a) = 3, we pop and print baz*
4. *len(a) = 2, we exit the loop, and print Done.*

* ***Question 10  
  How can a loop be interrupted prematurely?***
  + *Via the keyword break*

*The : operator is used to define the beginning of a block of code.*

*The key word continue just lets you skip a loop.*

*However, stop is not a Python keyword!*

## ***4.1. Familiarize Yourself With Python Modules and Libraries***

*Now, suppose you need to calculate the square root of a number in one of your notebooks. There is no native square root function in Python. You could of course write it yourself, but hey, there's probably been a bunch of people who have asked themselves the same question. And guess what? One of them has already written the function and saved it in a module!*

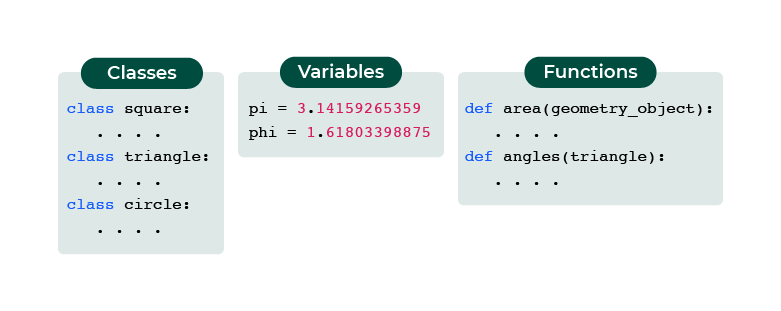
### ***A Module in Python***

*A* ***module*** *is a Python file containing a set of predefined and operational* ***functions****,* ***classes****, and* ***variables****, which you can use as you wish in your code!*

*For example, if you are working on a problem involving geometry, you might need:*

* ***classes:***
  + *Square—defined by the length of its side*
  + *Triangle—defined by the length of its three sides*
  + *Circle—defined by its radius*
  + *Etc.*
* ***variables:***
  + *Pi: constant necessary for calculating the area of a circle, equal to 3.1415...*
  + *Phi: constant that represents the golden ratio, equal to 1.6180...*
* ***functions:***
  + *Area: takes as parameter a geometrical object (square, triangle, etc.) and calculates its area*
  + *Angles: takes a triangle as a parameter, and calculates its internal angles*
  + *Etc.*

*You can of course define all these things in your notebook, but that would only make it more cumbersome. The best is to store all this in an external Python file, which you will then import into your notebook: it's a module!*

*Your geometry module*

*Here is a simplified example of a geometry module:*

*'''*

*Module geometry.py*

*'''*

*# variables*

*pi = 3.14159265359*

*phi = 1.6180*

*# function that calculates the area*

*def area(obj):*

*if type(obj) == square:*

*return obj.a\*\*2*

*# definitions of some classes*

*class square(object):*

*def \_\_init\_\_(self,a):*

*self.a = a*

*class triangle(object):*

*def \_\_init\_\_(self,a,b,c):*

*self.a = a*

*self.b = b*

*self.c = c*

*To* ***import a module****, you will need the import keyword. Here is an example with our geometry module:*

*import geometry*

*After doing this, you can use the different items defined in your module:*

*squa = geometry.square(4)*

*tri = geometry.triangle(3, 6, 5)*

*print(geometry.pi) # -> 3.14159265359*

*geometry.area(squa) # -> 16*

*All items included in the geometry module can be used via the moduleName. notation, i.e., moduleName.function() or moduleName.variable. So, in the above example, we can use geometry.area() or geometry.pi. If you don't want to rewrite geometry every time, you have two other options:*

* *Either give an* ***alias*** *to the name of your module, so you only have to write the alias:*
* *import geometry as geo # we can now access geo.area() or geo.pi*
* *Or, import specific functions that you can then use as native Python functions/variables (without the . notation):*

*from geometry import pi*

*print(pi) # -> 3.14159265359*

*A particular case of this last method is to import in one line all the objects contained in a module via the \* notation. However, this is not the recommended method, in order to avoid, for example, conflicts between several modules that might have identical function names.*

*from geometry import \**

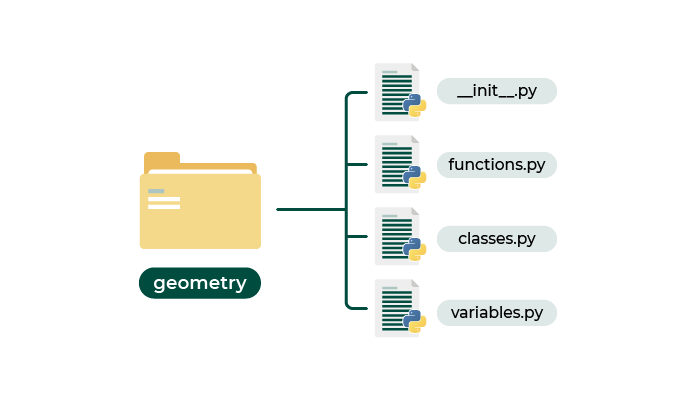
### ***When a Module is Not Enough: Packages***

*A* ***package*** *(sometimes called a* ***library****) is a collection, a* ***set of Python modules****. As you have seen above, a module is a Python file. A package is simply a folder containing several Python files (.py) and an additional file named \_\_init\_\_.py. This differentiates a package from an ordinary folder containing only Python codes.*

*For example, you could have stored your geometry module in three different files instead of just one:*

* *One for classes: classes.py*
* *One for variables: variables.py*
* *One for functions: functions.py*

*In this case, we would have the following file:*

**

*Organization of geometry package*

*You will need to use the . operator to access the module after importing the package:*

*import geometry # import all the geometry package*

*print(geometry.variables.pi) # -> 3.1415...*

*squa = geometry.classes.square(4)*

*geometry.functions.area(squa) # -> 16*

*Or, you can also import only one module from the package:*

*import geometry.variables as var # import only what is defined in variables.py*

*print(var.pi) # -> 3.1415...*

### ***Packages in Data Analysis***

*Packages are ubiquitous in data analysis with Python. Indeed, many packages have been created specifically to address the issues that this subject involves. As you progress, you will be required to:*

* ***manipulate your data*** *to facilitate analysis****.***
* *make* ***various relevant graphs*** *representing the behavior of your data.*
* *use* ***statistical methods****.*
* *run* ***machine learning algorithms*** *of varying complexity.*
* *Etc.*

*And to achieve all this, you will need to master the various objects and functions from the corresponding packages.*

*If you want to know more about the packages that let you do these different tasks, you can take the OpenClassrooms course* [***Use Python Libraries for Data Science***](https://openclassrooms.com/en/courses/1730411-use-python-libraries-for-data-science)*, which goes into more detail about the packages that are most used in data analysis.*

*To come back to your initial problem (having a square root function), there is for example the* ***numpy*** *package which offers the necessary function—and many other things!*

*import numpy as np*

*np.sqrt(16) # -> 4.0*

*In order to solidify the concept of packages, we will look at a concrete example of the use of therandom package in the next chapter.*

### ***Let's Recap***

*In this chapter, together we have seen the basics of using modules and packages:*

* *A module is a* ***file*** *containing Python code (.py extension) that can define* ***functions****,* ***classes****, and/or* ***variables****.*
* *You can* ***import*** *any Python module via the import keyword.*
* *To use a function class or a variable within a module, you must use the . operator.*
* *A* ***package*** *is a set of several Python modules.*
* *There are many packages specifically created for data analysis.*

*Now that you know what a module is in Python, follow me to the next chapter to discover how the* ***random*** *module works.*

## ***4.2. Manipulate Random Numbers With the Random Module***

*The ability to generate random numbers is extremely useful for all sorts of programming tasks, from a simple simulation of a dice roll to selecting data for data analysis activities. In Python, the* ***random*** *module contains several functions for generating random numbers or sequences of numbers.*

*If you are interested in the subject, the different functions of the random module use a very powerful and popular pseudo-random number generator, called the* [***Mersenne Twister***](https://en.wikipedia.org/wiki/Mersenne_Twister)*.*

### ***Generate Random Numbers***

*First, import your random module. The name of the package in Python is...* ***random****:*

*import random*

*The basic function for generating random numbers is called...* ***random()*** *as well (how original!). It will generate a* ***random float between 0 and 1 (not including 0 or 1)****. Let's try a simple example by displaying three random numbers:*

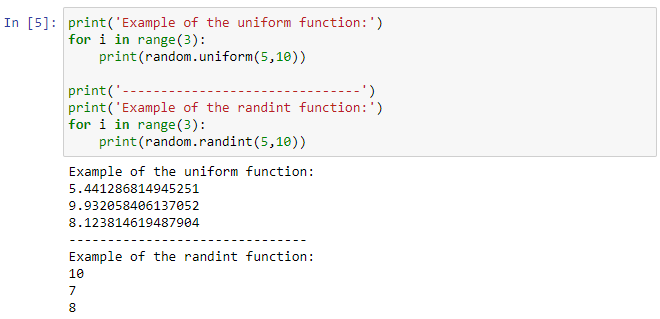
**

*Of course, if you run the same code at home, you will get different results (that’s the nature of randomness)!*

*But only having a number between 0 and 1 is a bit limited... isn't it?*

*Absolutely! But the people who created the random package fortunately thought of everything. There are other functions that let you generate a random number in a given range:*

* *uniform(a, b) : will generate a random float between a and b .*
* *randint(a, b) : as its name suggests, this one is similar to uniform except that the random number generated is an integer this time!*

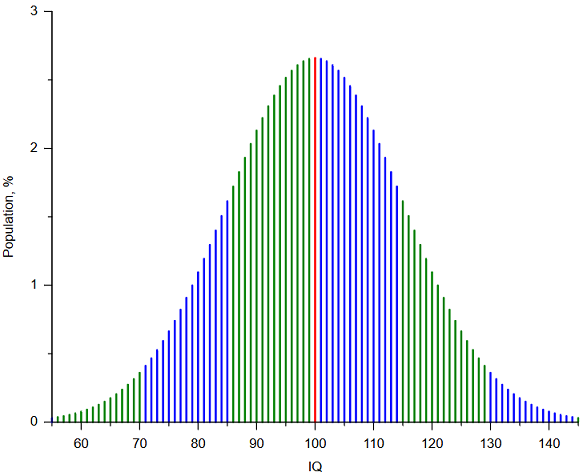
**

*You can use one or the other according to your needs!*

#### ***Generate a Random Number According to a Given Distribution***

*The random module can also generate a random number according to a distribution. One of the best known is the Gaussian (or normal) distribution. If you don't know it already, let me introduce you to it!*

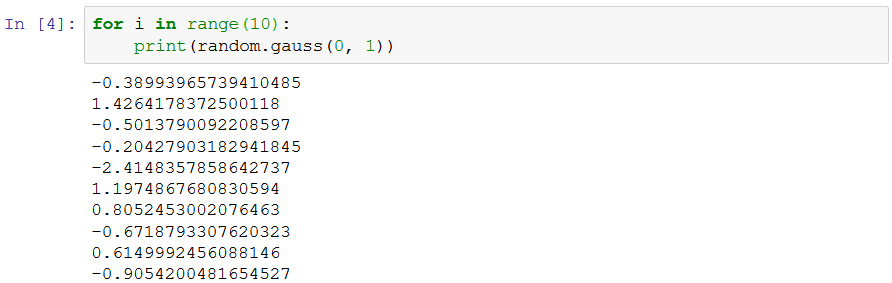
*The* ***normal law*** *is one of the most suitable probability laws to model natural phenomena resulting from several random events. These are all phenomena where the majority of individuals are around an average, with decreasing proportions below and above this average. Here is a very telling example, with the distribution of the population by IQ:*

**

*Proportion of the population by IQ—Alessio Damato, Mikhail Ryazanov*

*The random module lets you generate random numbers according to this law: i.e., you are much more likely to have values close to the average (with the example above, between 85 and 115) than extreme values (close to 70 or 130). The corresponding function is called gauss(mean, standard\_deviation) .*

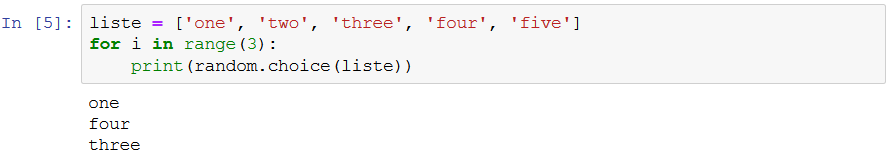
*Here is an example with a distribution centered at 0 and with a standard deviation of 1 (which is a “conventional” normal distribution):*

**

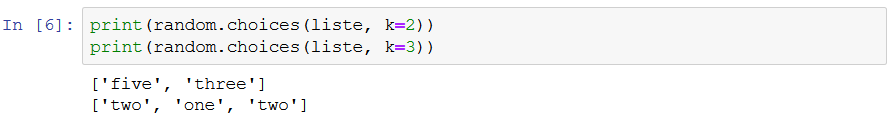
*We can see here, with 10 values, that the majority of the values are close to 0.*

### ***Choose Randomly From a List: Subsampling***

*As you already know, to select an item in a list, you have to select it via its index. If you want to select an item randomly from a list, a somewhat naive solution might be to draw the index randomly, and then use the random index to select the item. The* ***random*** *module goes a step further by offering a function that lets you make the selection directly from the list: the choice function.*

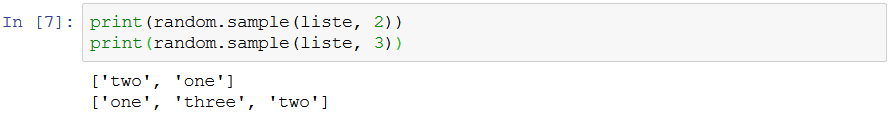
**

*The evolution of this is the choices function, now making it possible to select a sample from the initial list,* ***with replacement****:*

**

*Note how, in the second line, we get “two” returned twice, because the first “two” was effectively put back into the list once it was initially drawn.*

*This is called* ***subsampling****. The corresponding function, for a sample* ***without replacement****, is sample :*

**

*In data analysis, this concept of subsampling is essential, as it can select a sample from an initial population. In statistics, a sample is a set of individuals representative of a population. The use of a subsample is generally a solution to a practical constraint (lack of time, space, financial cost, etc.) that does not allow an exhaustive study of the entire population.*

### ***Further Reading***

*The random module offers more functions than those presented in this course, even though you have seen the ones that are most commonly used in practice. If you want to go further, you can consult* [*the official documentation of the random module*](https://docs.python.org/3/library/random.html)*, which lists all the possibilities offered by it.*

*You should also know that the* ***numpy*** *package, which we briefly mentioned in the previous chapter, also includes the random module. There, you will find all the functions seen above. All functions are accessible via the line (for example):*

*import numpy.random as random*

### ***Let’s Recap***

*You have seen the main features available through the random module. You can now:*

* *generate a random number,* ***integer****, or* ***decimalin a given range****.*
* *generate a random number* ***according to a given distribution****.*
* *randomly select* ***one or more items from a list****, with or without replacement.*
* *[#](https://openclassrooms.com/en/courses/2304731-learn-python-basics-for-data-analysis/7980005-manipulate-random-numbers-with-the-random-module#/id/r-7979990)*

# ***Quiz***

# ***4.3.Use Specialized Python Libraries***

### ***Evaluated skills***

* *Use specialized Python libraries*
* ***Question 1  
  Check the numbers that can be obtained if the following code is run:***

*import random*

*n = random.uniform(10,19)*

*print(n)*

* *Careful, there are several correct answers.*
  + *10.0*
  + *14.4*
  + *10.5*
  + *19.0*
* *Here, the uniform function will generate random decimal numbers, between 10 and 19 (including 10 and 19). All answers are correct except ‘6’ and ‘23.’*
* ***Question 2  
  What should the ... be replaced with, so that the following program prints one of the names in the list at random?***

*import random*

*basket = ["Apple", "Pear", "Banana", "Pineapple", "Orange"]*

*result = ...*

*print(result)*

* *Careful, there are several correct answers.*
  + *basket[random.randint(0, 4)]*
  + *random.choice(basket)*
* *random.randint(0, 4) will select a random number between 0 and 4. This random number will serve as an index to select from the basket.*
* *The choice function lets you select an item randomly from a list. This will return an error. The random function cannot take a list as a parameter.*
* *This function lets you mix a list.*
* ***Question 3  
  Consider the following experiment:  
  A normally-balanced die (with six faces) is thrown 10,000 times. Among these 10,000 throws, 1,000 are taken at random.  
  We carry out this experiment five times, and we write down m, the average number of times we get the number six out of the 10,000 throws, and n, the average number of times we get the number four in the subsample.  
  What would be the values of m and n at the end of this experiment?***
  + *m and n will be close to 1/6.*

*This piece of code, which performs a simulation of the experiment, should help you answer that question:*

*import random*

*m = 0*

*n = 0*

*for i in range(5):*

*listOf = []*

*j = 0*

*while j < 10000:*

*listOf.append(random.randint(1, 6))*

*j += 1*

*subSampleOf = random.sample(listOf, 1000)*

*m += listOf.count(6) / 10000*

*n += subSampleOf.count(4) / 1000*

*print("m =", m / 5, "and n =", n / 5)*

* ***Question 4  
  Let's consider two games of chance:   
  The next two questions will be based on these two sets---I strongly advise you to organize your code in several functions, to simplify organization and reading.***
  + *First, a game that we will call* ***A, which is a simple coin flip*** *with a biased coin (tails with a probability of p=0.49). The player bets one dollar and flips the coin: if they get tails, they win one dollar and recoup their bet, otherwise they lose their bet.*
  + *Then, a game that we will call* ***B, which is a game with two biased coins****. The first coin gives tails with probability p1 = 0.09 and the second coin gives tails with probability p2 = 0.74. The player can only bet one dollar at a time! However, at each flip, we look at the player's total amount of money to determine which coin to flip: if the amount is a multiple of three, we flip the first coin, otherwise we flip the second coin. As in game A, the player recoups their bet plus an extra dollar if the chosen coin lands on tails, otherwise they lose their bet.*
* ***A game is considered won when a player finishes with more money than they started with after playing a large number of rounds (e.g., several hundred). Implement these two games of chance with Python, using the libraries seen earlier, considering that the player starts with a capital of $1,000.   
  Which of the following statements is true?***
  + *Both games are losers.*
* *You will find a possible implementation of these two games* [*via the following link*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P4C3_1.ipynb)*. You can run the code yourself, to see that no matter how many times you try, both games lose!*
* ***Question 5  
  We will now mix the two games presented in the previous question! Effectively, at each turn, we now flip a coin which is balanced! If you have tails, you play game A, otherwise you play game B.  
  It is assumed that the player has $1,000 as starting capital.  
  After playing 1,000,000 games, what is the status of the game, from the player's point of view?***
  + *The game is won.*
* *You will find a possible implementation of this game* [*via the following link*](https://colab.research.google.com/github/OpenClassrooms-Student-Center/python-basics-data-analysis/blob/main/P4C3_2.ipynb)*. You can run the code yourself, to see that no matter how many times you try, the mixed game consisting of games A and B wins! This is called Parrondo's paradox!*
* ***Question 6  
  How can we generate N lottery tickets, i.e., N numbers between 0 and 999, which are unique?  
  You can use the range(p) function to produce a sequence of integers from 0 to p-1.***

*# range(p) produces a sequence of integers from 0 to p-1*

*population = range(1000)*

*# of tickets desired*

*N = 10*

* + **

***Only the sample() function allows sampling without replacement. This means that each sample of the population is taken in a unique way.***

***To be sure of this, just read the documentation of the function with***

***?random.sample***

***which states that***

***Signature: random.sample(population, k)***

***Docstring:***

***Chooses k unique random elements from a population sequence or set.***

***The other functions used (choices(), choice(), randint()) all sample “with replacement.” This means that a sample is put back into the population after sampling.***

***Question 7  
We try to generate a random float with the random function.  
What is the best way to import the right module and call the right function for this?***

*from numpy import random as rnd*

*x = rnd.random()*

*import numpy as np*

*x = np.random.random()*

*import random*

*x = random.random()*

* + *All versions are equivalent.*

*As the random module is accessible either directly by import random or as a sub-module of the numpy module, the three versions of the code are strictly the same.*

* ***Question 8  
  Which statement is false?***
* *A package consists of one or more python files and an \_\_init\_\_.py file.*
* *You can use a function or a class of a module using the dot (.) operator*
* *Packages and modules are imported in the same way:*
  + *import package\_name*
  + *import module\_name*
* *A module is a package consisting of a single python file.*

*A module is a Python file containing a set of predefined and operational functions, classes and variables, but a package is a set of several modules, so the last answer is wrong!*