Python Libraries

In practice, a library (also called sometimes a package) is **a collection of functions** written by someone that you can directly use without having to program them yourself.

There are literally hundreds if not thousands of them for many various tasks: from plotting to solving equations, analysing data, building Artificial Intelligence applications or even generating moving objects for videogames! This means that in many (if not even most!) cases you won't have to re-code everything yourself but you will find some library that can do that for you. As we said at the very beginning, Python coding is often about assembling these functions rather than re-writing them yourself from scratch!

In the next part of the lecture, we will have a quick look at the `Matplotlib` library, or better one of its subpackages `matplotlib.pyplot`. This is a library for plotting high-quality, publication-ready graphs. You might find it quite useful in writing reports, now and in the future, and you should really learn how to use it. Another perk? It is 100% free, unlike proprietary softwares!

How to use libraries

First thing to do when you want to use a package, you need to `import` it in your program. This can be done with the general command:

```
import name_of_library as alias
```

the `as alias` part is not necessary but it is pretty useful, as we will see later. For example, let us say we want to import the matplotlib library, we could simply write:

```
import matplotlib as mat
```

If you do not need to import the whole library but only a sub-package, you can do it instead via the command:

```
import name_of_library.name_of_package as alias
```

For example, because we will only see in this lecture the pyplot sub-package of matplotlib, we will actually import that part only, using the declaration:

```
import matplotlib.pyplot as plt
```

To understand what an alias does, it is important to see how can you use functions inside the library that you have imported. In this case, you must call the function with the full name and use a declaration like:

```
var = name_of_library.name_of_sub-package.name_of_function( inputs )
```

which can be quite cumbersome and also lead to a lot of mistakes, especially when the name of the library or the sub-package is very long! The alias solve this problem because it allows you to use a much shorter declaration, of the form:

```
var = alias.name_of_function( inputs )
```

Aliases tend to be short to save time and to be as clear as possible. Please note that you can decide to use whatever alias you want when importing, but some very common libraries tend to have **universally recognised aliases** in the Python community and it is good practice to stick to them. Among the library (or sub-packages) that we will use, the typically recognised aliases are `np` for the numpy library and `plt` for the matplotlib.pyplot subpackage.

For completeness, although good coding practice would actually ask you **not to use this type of declaration**, you can simply import some functions from a given library instead of all of the whole library. For example, the numpy library contains the function `sqrt` (that takes the square root of a number). If we only need that specific function, we could have simply imported it as:

```
from numpy import sqrt
```

in which case it could have been used directly with the short declaration that does not involve neither the full name of the library nor any alias. To be clear, the following code is perfectly valid and would use the sqrt function as defined in numpy:

```
from numpy import sqrt
a = sqrt( 2.0 )
```

The reason why importing functions in this way should be avoided and the full library name, or its alias, should be used, is that different libraries may contain functions with the same name, which can lead to confusion. In this case, for clarity the full declaration is better, although slightly longer.

Where do libraries come from?

At this point, you might wonder where does `import` look for these libraries? In practice, it looks for them in a hierarchical way among the libraries present in the Python distribution downloaded on your computer, and will complain if they are not found (in which case, you might have to download them from the internet).
The discussion is somehow a bit technical, but more details you can be found here (https://stackabuse.com/relative-vs-absolute-imports-in-python/#:~:text=The%20import%20statements%20do%20a,Python%20Standard%20Library%20for%20it.).

IMPORTANT NOTE: In some of the Jupyter Notebooks associated to this section, you will start to look at some useful functionalities of the numpy and matplotlib library. In practice, those notebooks will serve as *both exercises and notes* on those specific parts, in a kind of hybrid way that integrates the two together. Please let me know what you think about this somewhat different approach, any feedback is welcome!