What we did in lesson 2

1. Recap on basic Python + methods
   See readings from last week. We also pointed out that within each data type (such as a string or a list) some *methods* can be defined, so that if we create an object of that data type, we will have access to those methods. Methods are functions that use that object as an argument by default, and that may or may not require extra arguments as well. They are accessed by writing a dot (.) after the variable containing it. Two examples:

   The `split` *method of strings splits the string in correspondence of a given separator and returns a list:*

   ```
   my_string = 'whatSPACEdoSPACEyouSPACEwantSPACE?'
   my_list = my_string.split('SPACE')
   print(my_list)
   >>> ['what', 'do', 'you', 'want', '?']
   ```

   The `append` *method of lists adds items to the end of the list:*

   ```
   my_list = [2, 'duck', 23.9]
   my_list.append(1000)
   print(my_list)
   >>> [2, 'duck', 23.9, 1000]
   ```

2. Exercise: write simple programs WITHOUT online help
   Split into teams, we tried to write programs to accomplish specific tasks using only basic Python. The list of tasks is in the next page.

Homework

- Apart from the readings, please **finish the exercise we started in class**. Actually, it's much more important than the readings. It may be hard at first, but I promise you that it will become natural very soon. You are all brilliant people and I'm sure you'll have no problems if you practise. Unfortunately, there is literally no way around it: you *have* to practise.
- Contact me so that we can **agree on a personal project before the next lesson**. If you already have an idea, that's perfect! If you don't, don't worry, we'll come up with something together. Ideally, it should be something that's actually useful for your research.

Tasks for the exercise we did in class

Many of these tasks were taken, or inspired by, the ones collected here. They also have solutions, so it's a very good place to practise.

- [difficulty = 1] create a function that reverses a string of text;
- [1] create a function that calculates the statistics of a list of numbers (max, min, mean, std);
- [1] create a function that prints a list of numbers from 0 to a given number; if they are multiples of 3 also print "Okey", if they are multiples of 4 print "Dokey", if they are both print "OkeyDokey";
- [2] create a function that checks whether a given string is palindrome;
- [2] create a function that counts the number of words in a string (optionally, it could also write to screen a summary of how long they are);
- [2] create a function to find the number of times a substring appears in a string;
- [3] create a function that uses the Sieve of Eratostenes to find all prime numbers up to a given number.

I also selected these other tasks to be done by looking online for help, because they require syntax that we didn't see yet. If you find code online, really try to understand what each line does. If you use conversational AI tools, such as ChatGPT, remember that you can ask an explanation for each line of code. Also remember: *never* trust the AI, always check for yourself.

- [3] read the contents of a .csv file into a string, using only basic Python, no extra libraries;
- [4] read the contents of a .csv file into a list of lists (for rows and columns respectively), using only basic Python.

Readings before lesson 3

Tips: accessing the documentation of a function

There is a command to get the documentation of a function from the Python console or from a cell in Jupyter. Its just the name of the function with a question mark (?) before. An example: the command

```
?range
```

gives this output:

```
>>> Init signature: range(self, /, *args, **kwargs)
Docstring:
range(stop) -> range object
range(start, stop[, step]) -> range object

Return an object that produces a sequence of integers from start
(inclusive) to stop (exclusive) by step.  range(i, j) produces i, i+1,
i+2, ..., j-1. start defaults to...
```

When using Jupyter, you can also position your cursor between the brackets of a function you're writing and press `SHIFT+TAB` to see its documentation. It's very handy when you don't remember the order of the arguments that the function requires.

Libraries/modules/packages/however you want to call them

We introduced the concept of libraries during the first lesson, when we talked about all the stuff that Anaconda installed on your computer. Libraries are collections of code (mostly *just text*) that are written by the Python community to expand the "vocabulary" of the programming language, giving you access to new data types and functions. The most common ones are installed by distributions such as Anaconda automatically.
To import a library you just need a single command. If you wanted to import the `numpy` library, for example, you could write:

```
import numpy
```

From then on, if you want to access something that is defined inside that library, you have to write the name of the library, then a dot, then the name of the thing you want to use (a value, or a function, or something else). For example, within `numpy` there is the value of π. You can access it by writing:

```
numpy.pi
```

and you can do calculations with it:

```
numpy.pi + 1
>>> 4.141592653589793
```

Some libraries have long names, and it may be inconvenient to write them fully each time. For this reason, it's possible to define a shorter name for the library when importing it. For example, if you use the command:

```
import numpy as np
```

You can then rewrite the instruction above as:

```
np.pi + 1
>>> 4.141592653589793
```

and get the same result. The most widespread libraries have short names that are more or less standard, and `np` for `numpy` is one example. You can use whatever short name you want, but I strongly suggest sticking to standard because this greatly improves the readability of your code (which is very, *very* important).

`numpy` library

During the next lesson we'll make extensive use of `numpy`, which is the basic library (or "module") to work with data. The `numpy` library introduces the array data type. Arrays are containers for data of a single type (usually all numbers) organized in multi-dimensional tables. You can think of a one-dimensional array as a vector, and of a two-dimensional one as a matrix. But you can have as may dimensions as you want.
Items, or sets of items, can be retrieved by using their indices, in a manner that is similar to what could be done with lists. All sorts of operations between arrays can be performed in an efficient way.

As an introduction to `numpy` arrays I suggest reading *Numpy: the absolute basics for beginners* from the official library documentation. Since it's quite long and we don't actually need all of it, I'll link here to specific paragraphs that are important (I know they are a lot, but they're all quite short):

- What is an "array"?
- Array fundamentals
- Array attributes
- How to create a basic array
- Indexing and slicing

- Basic array operations
- Broadcasting
- Creating matrices
- Transposing and reshaping a matrix

But the numpy library is more than just arrays! If there is something to be done with numbers, numpy is usually a good place to look. It's impossible to memorize all the contents of the library; it's much better to search when in need. But just as an example, here is some stuff you can do with numpy:

- access constants like π or e;
- use mathematical functions such as exponentials and logarithms, or trigonometric functions;
- do Fourier transforms;
- create and manipulate (even fit) polynomials;
- do some random sampling;
- most of all, do **linear algebra**, from regressions and matrix-vector operations to eigenvalues and systems of equations.

Finally, for those of you who already have some experience in programming and are coming from the **Matlab** ecosystem, there is a specific documentation page to help you find equivalent (more or less) syntax.

Extra stuff: list comprehensions

A list comprehension is a powerful syntax in Python that allows you to define a list with a loop using a single line of code. You will probably find this syntax if you look for code online, because it's very popular. Here is an example of two programs that do the exact same thing, one with an explicit loop and one with a list comprehension:

Version 1: with an explicit loop

```
my_list = []
for i in range(3):
      my_list.append('number ' + str(i))
print(my_list)
>>> ['number 0', 'number 1', 'number 2']
```

Version 2: with a list comprehension

```
my_list = [‘number ‘ + str(i) for i in range(3)]
print(my_list)
>>> [‘number 0’, ‘number 1’, ‘number 2’]
```