Assignment 0

This assignment is not about numerical methods per se. It contains instead some simple PYTHON exercises to refresh your coding skills. Don't be scared by the number of exercises, as most (if not all) tasks can be solved in very few minutes. Also, remember that it is always a good practice to comment your code. This will help readability from other users as well as your future self!

Exercise 1 (Figuring out the precision)

The goal of this exercise is to check whether you are running your codes on a 32 or a 64 byte shell. To do this, type the following commands in PYTHON:

```
1 import sys
2 print(sys.maxsize > 2**32)
```

If the printed result is `True`, then you are on a 64 bit system (double precision), otherwise you are on a 32 bit system (single precision).

Now, let's try to understand what we have just done. First, with `import sys`, we import the PYTHON module giving access to system-specific parameters and functions. From this, we get the quantity `maxsize`, which the maximum value that an index in PYTHON can get (not to be confused with the maximum integer). This amounts to $2^{31} - 1$ on 32 bit systems and to $2^{63} - 1$ on 64 bit systems.

Exercise 2 (Basic commands and conditional statements)

Write programs that perform the following tasks.

(a) Compute the circumference of a circle with radius $r = 1$ and print its value on screen.

Hint: To get the value of π, you can load the module `math` and call `math.pi`. Note that also the module `numpy`, which we will see later, has its own definition of π.

(b) Write a function that, given a number in input, determines if this is even and prints the corresponding message to the user. As output, the function should return `True` is the input is even. Test your function with once 5 and once 10.

(c) Swap the values of two variables $a = 5$ and $b = 10$ (that is, the end values should be $a = 10$ and $b = 5$).

Exercise 3 (Loops)

In this exercise, we refresh the use of `for` and `while` loops. Write the code to perform the following tasks.

(a) Write a program that prints on screen all integers between 1000 and 2000 (both included) which are divisible by 5 and 7.

(b) A pair of prime numbers (x, y) is called a pair of *twin primes* if they differ by 2, that is, if they are consecutive primes. An example is the couple $(17, 19)$. Write a function that, given a positive integer n in input, prints on screen the first n couples of twin primes. Usually the pair $(2, 3)$ is not considered a pair of twin primes. Test your function with $n = 10$.

To check if a number is prime, you can use the following function (source: Geekflare):

```
def is_prime(n):
    import math
    for i in range(2,int(math.sqrt(n))+1):
        if (n%i) == 0:
            return False
    return True
```

Exercise 4 (Lambda functions)

For simple expressions, instead of using standard functions one can use anonymous functions. In PYTHON, these correspond to lambda functions. To refresh your knowledge on these, perform the following tasks.

(a) Create a lambda function that multiplies by 2 a given number passed in as an argument, and test your function on the input 3 and on the input 10.

(b) Initiate a list `[1,5,9]`, and write a program to square every number in the list using a lambda function. Print the result on screen.
Hint: You can use the PYTHON function `map` to apply the lambda function to each element in a iterable (for example in a list).

Exercise 5 (Numpy module)

An extremely useful PYTHON module for numerical algorithms is the `numpy` module, to handle arrays and perform operations on them. The goal of this exercise is to practice a bit with it.

(a) Create a `numpy` array of length 5 filled with zeros, and another `numpy` array, still of length 5, filled with ones.

(b) Write a program that initiates a `numpy` array with values `[1,2,3,4]` and then prints `True` if none of the array entries is zero. Repeat this task for the array `[0,1,2,3,4]`.

(c) Write a program that initiates a `numpy` array with values `[1,2,3,4]` and then prints `True` if at least one of the array entries is non-zero. Repeat this task for the arrays `[0,1,2,3,4]` and `[0,0,0]`.

(d) Create and print on screen a `numpy` array containing all integers between 1 and 10 (both included).

(e) Create two `numpy` arrays, with values `[1, 4, 5]` and `[6, 9, 5]` respectively. Multiply them elementwise and print the result on screen.

(f) Write a program to create a 4×4 matrix and fill it with a checkerboard pattern (alternating zeros and ones).

(g) Initiate a `numpy` array of length 5 with uniformly distributed random values between 0 and 1 (*Hint*: use `numpy.random`), and calculate round, floor and ceiling elementwise, printing on screen the results.

(h) This exercise is simple but **very important** to understand basics of memory management. Create a `numpy` array of zeros, with 5 entries, and call it `x`. Set a second `numpy` array `y=x`, and a third array `z=np.copy(x)`. Print all three arrays on screen. Now modify the last entry of `x` and set it to 10, and then print again `x`, `y` and `z` on screen: what do you observe? How do you explain this? Make sure you understand this, because mistakes in copies can introduce bugs in many algorithm implementations!