In this lecture we are going to recall the basics of python and git and introduce a package manager for python (pdm) as well as GitHub as our platform of choice. The following exercises are focused on recalling the basics and introduce the new topics.

Prepare before class such that you can present your result: nothing - first lecture
We will be working on the following in class: 1, 2, 3, 4, 5, 6.

(1) Setup a git project on GitHub

In order to get a bit of `git` training done we work for all the exercises on GitHub.

(a) Create a new **private** project for the exercises on GitHub

(b) Give the instructor (kandolfp) access to the project (see `GitHubdocumentation` for help)

(c) Create a `pdm` project in your repository and commit the necessary files.

(d) Create an appropriate structure in your repository for the rest of the exercise sheet (maybe have a look at the exercises to have a better idea first), not everything should be in the main folder.

(e) Try to structure your work on the exercises with git, i.e.

- Don't commit things that do not belong together in one single commit. Each exercise can be considered as a separate thing. Subparts of an exercise might be independent as well.
- Use meaningful commit messages `conventionalcommit`
- Make sure that you do not commit something that does not work - produces an error. If you have difficulties with an exercise you can also commit your best effort in this case.

(f) Add a README.md that explains what you are doing, how to run the exercises and anything else that is necessary (quick guide to `pdm`), maybe note your name somewhere.

(g) Optional

- Work with issues, you can reference the issue in the commit message, GitHub documentation

(2) Difference between assignment, copy and deep copy

Have a look at the module `copy` and the functions `copy`, `deepcopy` as well as the standard assignment operator `=`.

Run the following code and explain its output:

```
import copy
list1 = [[1, 2, 3], [4, 5, 6], ['a', 'b', 'c']]
list2 = list1
list3 = copy.copy(list1)
list4 = copy.deepcopy(list1)

print('List # \tID\tEntries')
```

```
print('1\t', id(list1), '\t', list1)
print('2\t', id(list2), '\t', list2)
print('3\t', id(list3), '\t', list3)
print('4\t', id(list4), '\t', list4)

list2[2][2] = 9

print('List # \tID\tEntries')
print('1\t', id(list1), '\t', list1)
print('2\t', id(list2), '\t', list2)
print('3\t', id(list3), '\t', list3)
print('4\t', id(list4), '\t', list4)

list1.append([0, 8, 15])
print('1\t', id(list1), '\t', list1)
print('2\t', id(list2), '\t', list2)
print('3\t', id(list3), '\t', list3)
print('4\t', id(list4), '\t', list4)
```

Hint: the function `id` returns the *identity* of an object. It is guaranteed to be unique and constant for every object as long as it exists in memory.

We will discuss this in detail (if necessary) in class and points are given on how you approach this problem, not if you can fully explain all details. So provide your reasoning and your sources.

(3) Monte Carlo simulations with random numbers, functions, conditionals and loops: We are going to compute π with a Monte Carlo Method, see this Wiki article for a more detailed background[1].

A circle with radius r has an area of

$$A_{circle} = \pi r^2$$

and the square that encases it

$$A_{square} = 4r^2.$$

The ratio between the area of the circle and the area of the square is

$$\frac{A_{circle}}{A_{square}} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4}$$

and therefore we can define π as

$$\pi = 4\frac{A_{circle}}{A_{square}}.$$

The same is true if we just take the first quadrant, so $\frac{1}{4}$ of the square as well as the circle. This simplification will make the code more compact and faster.

The algorithm therefore becomes:

(a) For a given number N of uniformly scattered points in the quadrant determine if these points are in the circle (distance less than 1) or not. We call the number of points in the circle M.

(b) Estimate π by computing

$$\pi \approx 4\frac{M}{N}. \tag{1}$$

[1] `https://en.wikipedia.org/wiki/Monte_Carlo_method`

To write this in python follow these steps:

(a) Search for a module that allows to generate random values in python.

(b) Define a function `def in_unit_circle(N)` that computes and returns M from the function input N, which is a single positive integer. **Hint:** You can interpret two numbers between 0 and 1 as cartesian coordinates of a point and the squared sum of them will tell you the distance of this point from the origin.

(c) Define a function `def estimate_pi(N)` that computes and returns the estimation of π with (1).

(d) Search for a module that provides the exact value of π and write a function `def get_accuracy(N)` that returns the absolute difference between the above function and π.

(e) Test your function with different values of N. **Hint:** N needs to be quite large to have multiple digits of π correct.

(4) VISUALISATION OF A PROBABILITY DENSITY WITH FUNCTIONS, LOOPS, AND PLOTS: Write a program that visualises the probability density of the normal distribution:

(a) Write a function `def f(x, mu, sigma)` that expects $x, \mu, \sigma \in \mathbb{R}$ as input and returns the value

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

.

(b) Create the lists
$xs = [-5, -4.9, -4.8, \ldots, 4.8, 4.9, 5]$ and
$ys = [f(-5, \mu, \sigma), f(-4.9, \mu, \sigma), f(-4.8, \mu, \sigma), \ldots, f(4.8, \mu, \sigma), f(4.9, \mu, \sigma), f(5, \mu, \sigma)]$
for $\mu = 1, \sigma = 5$.

(c) Plot a graph that visualises xs, ys.

(d) Try different values for μ and σ.

(e) Try plotting multiple graphs with different μ and σ and a legend in a single plot.

Hint: Use the module `matplotlib`.
Try to find a solution without the module `numpy` as we will use this in the lecture to show the capabilities.

(5) SOME FUN WITH `numpy`

Create some suitable `numpy` arrays and compute the following values for them:

- different norms of vectors or matrices (two-, one-, infinity-, Frobenius-norm)
- median
- mean, also along an axis of a multidimensional array
- average, play around with weights
- variance
- variance by building it yourself with element wise operations and mean
- standard deviation
- compute the Cholesky decomposition of a square positive definite matrix. **Hint:** For a random vector a, with elements in $[0, 1[$ the following matrix is square, symmetric and positive definite $A = aa^T + I$

(6) Vectorisation

Recall exercise (3) from this exercise sheet. With the help of the `numpy` module write a second version where you use vectorization to solve the exercise. Go along the following steps:

- With the `numpy.random.random()` create a $2 \times N$ matrix of random numbers.
- Use element wise computation to get M. **Hint:** True is interpreted as 1 and False as 0 when you try to add boolean values.
- Test your two versions if they create similar accuracy for the same N.
- Have a look at the module `timeit` and see which of your two code versions is faster for which N.
- Plot your results, i.e. for different values of N the different execution time for the two versions. Can you think of something else useful to plot?

Try to solve these problems on your own. If you get stuck, maybe some of your fellow students can help you. Why not write down your question in the course forum in SAKAI. Please try not to write back entire solutions or exchange solutions in the forum, this is not the idea and will backfire in the long run (also we might decide to delete such posts).