MAD 2020/2021 Optional Exercises 1

Bulat Ibragimov, Kim Steenstrup Pedersen

November 16, 2020

This is a set of optional exercises to help you with the topic of this week. Your solution to these exercises are not to be handed in. You may discuss solutions with your TA and your fellow students.

Specifically, in this set of exercises we focus on Python. If you already know a topic covered in a question, then skip ahead.

1 Python

Exercise 1 (**Python programs).** Python programs consists of files called scripts, which is written as a text file usually with the file extension `.py` (this is just a naming convention and not a requirement). Python prefers that these files contain text encoded in UTF-8 format (most Python aware editors does this automatically). For more information see `https://docs.python.org/3/tutorial/modules.html`

Create your first Python program by creating a text file called `myhelloworld.py` in your favorit editor (see Absalon / Course material for our recommendations). Add the following line of code to the file and save it:

```
print("Hello world!")
```

Now you can run the program by calling the Python interpreter. In a terminal window this is done by typing:

```
python myhelloworld.py
```

Click the run button in your editor / IDE, or double clicking on the file (depending on your OS, IDE and Python installation).

You should get the following output:

```
Hello world!
```

Exercise 2 (**Python flow control).** Python has the common flow control statements as any other imperative programming language which includes conditional branching with `if-else` statements and loops with `while` and `for` statements. Read about these at
`https://docs.python.org/3/tutorial/introduction.html#first-steps-towards-programming` and
`https://docs.python.org/3/tutorial/controlflow.html`.

Construct a Python program that declares the following list and assigns it to a variable called `L`

```
L = [1, 2, 3.5, 4, "Hello"]
```

Notice how the Python builtin list data type can store elements of mixed data type (here integer and floating point numbers as well as strings). Read about lists here `https://docs.python.org/3/tutorial/introduction.html#lists` and for more advanced usage see
`https://docs.python.org/3/tutorial/datastructures.html#more-on-lists`.

In your program, write a `while`-loop that iterates over the list and prints each element in the terminal. Next, also add a `for`-loop that does the same. Your program should print out the list twice.

Exercise 3 (**Python functions - Basics).** It is possible to define functions and procedures in Python (a function returns some data when called, a procedure does not). Read about functions here `https://docs.python.org/3/tutorial/controlflow.html#defining-functions` and here
`https://docs.python.org/3/tutorial/controlflow.html#more-on-defining-functions`.

Write a procedure (by defining a python function with side-effects) that takes 3 parameters x, y, and z and prints the largest odd integer among them. If non of them are odd (or of other data types than integer), the procedure should print a message about this.

Exercise 4 (**Python functions - Binomial coefficients).** The amount of subsets with exactly k elements taken from a set with n different elements is called the bionomial coefficient $\binom{n}{k}$ (read as: "binomial coefficient n over k" or simply "n over k") which can be computed by

$$\binom{n}{k} = \begin{cases} 0 & \text{hvis } k < 0 \text{ eller } k > n \\ 1 & \text{hvis } k = 0 \text{ eller } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{for } 0 < k < n \end{cases} \tag{1}$$

Define a function `binomcoeff(n,k)` that use the relations defined in (1) to compute and return the binomial coefficient n over k.

Exercise 5 (**Python functions - call by value or call by reference).** Python handles parameters of functions differently depending on whether the parameter is a basic builtin types (such as numbers and boolean) or more advanced type (such as lists).

Type in this program, what do you expect it to do?

```
def foo(a):
   a = 4

a = 3
foo(a)
print(a)
```

Run it and see if it does as expected.

Next type in this program, what do you expect it to do?

```
def bar(M):
   M.append("world!")

L = [1, 2, 3.5, 4, "Hello"]
bar(L)
print(L)
```

Run it and see if it does as expected.

Notice, how in the first case the function `foo` handles its parameter in a call by value fashion, which means that any changes made to the parameter is local to the function. However, in `bar` Python handles the parameter in a call by reference fashion, which means that any changes made to the parameter will also be seen outside the function. For advanced data types such as lists and objects (instances of classes) Python apply the call by reference principle when handling parameters in functions.

This exercise also touches upon the question of variable scope in Python. Read more about this at `https://docs.python.org/3/tutorial/classes.html#python-scopes-and-namespaces`.

2 Numpy

Before you start - then read
`https://absalon.ku.dk/courses/42639/pages/primer-slash-cheat-sheet-on-a-numpy-arrays?module_item_id=1088820`. Consider reading the parts of the numpy user guide that you find relevant - `https://numpy.org/doc/stable/numpy-user.pdf`.

In the following exercises we will take you through some of the key aspects of numpy. We will assume that you have imported numpy with the statement `import numpy as np` which allows us to refer to the `numpy` namespace with the shorthand `np`.

Exercise 6 (**The shape of a numpy array can be manipulated).** Create a `numpy.ndarray` with `A = np.arange(10)`. What is the shape of this array? Change the shape of `A` to a 2×5 matrix without copying into a new variable (Hint: you can use either `np.reshape` or change the shape property of `A`, see `https://numpy.org/doc/stable/user/quickstart.html#shape-manipulation`). Next change `A` to a 5×2 matrix.

Exercise 7 (**Data types in numpy arrays).** Create a `numpy.ndarray A` with 4 elements of data type `int`. Now assign each element the value 3.5 without using a loop control structure (Hint: You can use indexing into the numpy array to formulate this in one line, see `https://numpy.org/doc/stable/user/quickstart.html#indexing-slicing-and-iterating`). What is now stored in the elements of `A`?

Exercise 8 (Indexing elements in arrays). Create a `numpy.ndarray` with `A = np.arange(10)`. Used the indexing feature of numpy to extract every second element from `A` and store it in a variable called `B`. Now set the elements of `B` to the value -1 (Hint: Again you can use indexing to select every second element). What happens with the elements in `A` and can you explain why?

Exercise 9 (Using a `numpy.ndarray` as a matrix). What is the difference between these computations?

```
A = np.arange(4)
B = np.asmatrix(A)
C = A.T * A
D = B.T * B
E = A.T @ A
F = np.dot(A, A)
```

Why are C, D, E and F different?

Exercise 10 (Constructing arrays with specific content). You can construct arrays from Python lists using the function `np.array` and provide a list as argument. You can construct arrays of all zeros or ones using either the functions `np.zeros` or `np.ones`. A square identity matrix can be created with the function `np.eye`. See the documentation of these functions for details, `https://numpy.org/doc/stable/index.html`.

Write a function that returns a `numpy.ndarray` with 1 on the anti-diagonal elements. The following is an example of such a 3×3 anti-diagonal matrix

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} .$$

Can you write this function without the use of loop statements and by using the indexing mechanism of `numpy.ndarray`.

Exercise 11 (Linear algebra using numpy). Consider this system of linear equations

$$2x_1 + x_2 + 5x_3 + x_4 = 9, \tag{2a}$$

$$x_1 + x_2 - 3x_3 - x_4 = -5, \tag{2b}$$

$$3x_1 + 6x_2 - 2x_3 + x_4 = 8, \tag{2c}$$

$$2x_1 + 2x_2 + 2x_3 - 3x_4 = 3 \ . \tag{2d}$$

Use the function `numpy.linalg.inv` to solve for the variables x_i. See the manual for details, `https://numpy.org/doc/stable/reference/generated/numpy.linalg.inv.html`

Exercise 12 (Linear algebra using numpy, part II). If $\mathbf{A} \in \mathbb{R}^{n \times m}$ the the following expression is referred to as the pseudo inverse of $\mathbf{A}$

$$\mathbf{A}^{+} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T$$

Write a program that constructs a random matrix `A` using `numpy.random.random` and computes the pseudo inverse of `A` using the expresssion above (for this exercise, do not use the function `numpy.linalg.pinv`).

Exercise 13 (Linear algebra using numpy, part III). Define a random but symmetric `numpy.ndarray`, $\mathbf{X}$, using `numpy.random.random`. Compute its eigenvalues $\mathbf{e}$ and eigenvectors $\mathbf{V}$ using `numpy.linalg.eig`. Verify that $\mathbf{V}$ is an orthogonal matrix and that $\mathbf{V} * \text{diag}(\mathbf{e}) * \mathbf{V}^T$ is close to X by using the function `np.dot`.

Exercise 14 (Linear algebra using numpy, part IV). Compute the 5'th power of $\mathbf{X}$ from 13 using the `**` power operator and compare with the results you get from this expression $\mathbf{V} * \text{diag}(\mathbf{e})^5 * \mathbf{V}^T$. Do you get the same result?

3 Matplotlib

The package Matplotlib provides functionality for making nice plots of data and it works on both Python lists and numpy arrays. We will use this package a lot in this course. See the full documentation at `https://matplotlib.org/3.3.2/contents.html` and tutorials at `https://matplotlib.org/3.3.2/tutorials/index.html`

Exercise 15 (A simple 2D plot). Run the following code and see what it does:

```
from matplotlib import pyplot as plt
import numpy as np

x = np.arange(0,8.0, 0.1)
plt.plot(x, np.sin(x))
# Always add axis labels and title to your figures
plt.xlabel('x')
plt.ylabel('Sin(x)')
plt.title('Sinusoid')

plt.show()

print("The end!")
```

When does the last print statement get executed?

Exercise 16 (Multiple plots). Run the following code and see what it does:

```
from matplotlib import pyplot as plt
import numpy as np

x = np.arange(0,8.0, 0.1)
plt.figure()
plt.plot(x, np.sin(x))
# Always add axis labels and title to your figures
plt.xlabel('x')
plt.ylabel('Sin(x)')
plt.title('Sinusoid')

plt.figure() # New figure
x = np.random.random((100,))
t = np.exp(-x)
plt.subplot(1,2,1)
plt.plot(x,t, 'r*')
plt.title('Plot 1')
plt.xlabel('x')
plt.ylabel('t')

plt.subplot(1,2,2)
plt.hist(t)
plt.title('Plot 2')
plt.xlabel('t')
plt.ylabel('Hist(t)')

plt.show()
```

Exercise 17 (Saving a plot). Run the following code and see what it does:

```
from matplotlib import pyplot as plt
import numpy as np

x = np.arange(0,8.0, 0.1)
fig = plt.figure(figsize=(16,9))
plt.plot(x, np.sin(x))
# Always add axis labels and title to your figures
plt.xlabel('x')
plt.ylabel('Sin(x)')
plt.title('Sinusoid')

fig.savefig('myfigure.png')
```