Exercise Sheet #11

Deadline: 01.07.2024, 12:00h

Bonus Points There is a total of 40 points on this sheet. As for every other sheet 20 points count towards your exercise grade, but unlike the other sheets you have the chance to obtain 20 bonus points.

Introduction to Python Python is (after JavaScript) the second most popular programming language in the world and is used widely in academia and in particular machine learning. We therefore want to give you the opportunity to familiarize yourself with Python in this sheet. There are a lot of simple exercises designed to cover various aspects of the language. There are explanations on this sheet but to solve them, the lecture will most likely not suffice, and you are encouraged to use external sources.

Install Python Python is, in contrast to `C++` an interpreted language. To run programs you need to install the Python interpreter. Tutorials how to do that for your OS are everywhere on the internet, e.g. here.

Problem 1 (*Hello World*)

The first step when dealing with a new programming language is usually to test the installation by running a *Hello World!*-program. In Python this can be done in just one line – Python does not have a `main` function!

```
print("Hello World!")
```

Execute the program by running `$ python <file_name>.py` from the console.[1]

[1] If that does not work, replace `python` with `python3`. If there are still problems, check the name of your Python executable. Hint for googeling: The location of the executable must be in the `PATH` variable.

Problem 2 (*Functions and Control Flow: Prime Tester*) (5 points)

In this problem, we write a basic function using `if`-statements and loops. Consider the following code snippet and familiarize yourself with the syntax.

```python
# function_example.py

# Define a function
def my_function(my_parameter):
    if my_parameter < 0:
        print("I hate your negativity !")
    else:
        for i in range(10):
            print(my_parameter, "for the", i, "th time")
    return my_parameter + 1
# Run the function
my_function(-1)
print(my_function(2))
```

A couple notes on that snippet:

- Python is, unlike `C++`, dynamically typed, i.e. you do not have to specify the type of a variable (or parameter) or the return type of a function.
- Again notice that Python does not have a `main()` function. By running the script using `$ python function_example.py`, all the code written above is executed.
- The `range()` function generates an iterable object with all integers strictly smaller than the passed number, starting from zero.

Try running the snippet and play around with it.
Then write a function `is_prime(num)` that takes in an integer `num` and tests if `num` is prime or not. The result of the test should be printed to the console. Test your function with a couple meaningful examples.
Hint: You might need the modulo operator `%`.

Problem 3 (*Data Structures*) (10 points)

The main data structures in Python are lists (resizable arrays), tuples (fixed size container), dictionary (hash table) and sets (unordered container with no duplicates and fast lookup time). Research how to use these data structures. Then solve the following task.
You are given a list of dictionaries representing employees in a company. Each dictionary contains the following information:

```
employees = [
    {'name': 'John', 'age': 28, 'department': 'Sales'},
    {'name': 'Emily', 'age': 34, 'department': 'Marketing'},
    {'name': 'Michael', 'age': 42, 'department': 'Finance'},
    {'name': 'Jessica', 'age': 31, 'department': 'Sales'},
    {'name': 'David', 'age': 29, 'department': 'Finance'},
    {'name': 'Sarah', 'age': 36, 'department': 'Marketing'},
    {'name': 'Daniel', 'age': 32, 'department': 'Sales'}
]
```

1. Create a list called `sales_employees` and populate it with the names of employees who belong to the 'Sales' department.

2. Create a tuple called `finance_ages` and populate it with the ages of employees who belong to the 'Finance' department.

3. Create a dictionary called `marketing_employees` and populate it with the names and ages of employees who belong to the 'Marketing' department. The keys should be the names, and the values should be the ages.

4. Create a set called `unique_departments` and populate it with the unique department names found in the list of employees.

5. Create a function called `find_employee` that takes a name as an argument and returns the department of the employee with that name. If the employee is not found, return "Employee not found."

6. Write a program that calls the `find_employee` function and prints the department of the employee named 'Michael'.

7. Write a program that iterates over the `employees` list and prints the name and age of each employee in the following format: "Name: [name], Age: [age]".

Problem 4 (*PIP and Virtual Environments*)

One of the most powerful things about Python is that there is a library (or package) for almost everything. In the upcoming problems we will use some popular libraries, however, they first must be installed.
The most common and easiest way to install libraries is using the *PIP package manager*. Usually PIP is installed together with Python. To test if you already have PIP installed, run `$ pip --version` from the command line.[2]
To install a package, you can simply run `$ pip install <package>`.
To avoid conflicts between libraries installed for different projects, it is a common practice to create a *virtual environment* for every project. At the core, this is just a copy of the Python interpreter and the PIP executable. Newly installed packages are placed directly in the directory of the project and are not visible to other projects.
Create a virtual environment and install a package using PIP. Follow these steps:

- To create a virtual environment, navigate to a new project's directory and run `$ python3 -m venv ./venv` from the command line.
- After creating the environment, you still have to activate it using

```
$ source venv/bin/activate          # MacOS or Linux
PS C:\> venv\Scripts\Activate.ps1   # PowerShell on Windows
```

  You should now see a `(venv)` in front of your prompt.
- Run `$ pip --version` to check that everything worked. To 'solve' this problem it suffices to hand in the output of that command. It should tell you that the instance of PIP that is used is from your project's directory.
- Now install a package! Run `$ pip install numpy`. This should install the *NumPy* package that we will use later.

[2] If problems arise try `pip3` instead or check the name of the executable. Also make sure that you have PIP installed.

Problem 5 (*NumPy*) (10 points)

A very important library in Python is called *NumPy*. It provides an array class and a huge variety of methods to operate on these arrays. The reason to use NumPy is that the array operations are all performed in `C` instead of Python, which is much faster in most cases. NumPy can therefore be understood as a Python 'wrapper' around fast `C` arrays. When using NumPy it is important to limit yourself as much as possible to the functions that NumPy provides instead of e.g. using a Python `for`-loop to iterate a NumPy array, since this would again be slow Python code.
From Problem 4 you should have NumPy installed. To use it, you first have to import it. At the top of your `.py` file, write

```
import numpy as np
```

Anything from the NumPy library can now be used via `np.<function>()`. Using the documentation, solve the following tasks only using NumPy functions.

1. Create a 3×3 NumPy array called `matrix` with random integer values between 1 and 10.
2. Find the maximum value in the matrix.
3. Find the minimum value in each row of the matrix.
4. Calculate the sum of all the elements in the matrix.
5. Calculate the mean of each column in the matrix.
6. Transpose the matrix.
7. Create a 2×2 NumPy array called `submatrix` that contains the top-left corner elements of the original matrix.
8. Generate a new 3×3 NumPy array called `new_matrix` with random integer values between 1 and 10. Perform element-wise multiplication between the `matrix` and the `new_matrix`.
9. Reshape the resulting array from step 8 into a 1-dimensional array.
10. Calculate the dot product between the reshaped array and a randomly generated 1-dimensional array of the same length.

Problem 6 (*Classes and Matplotlib: Euler Solver*) (10 points)

Just like `C++`, Python is an object-oriented programming language. In this problem we want to learn how to write a class and work with objects.

1. Research how to write classes in Python.

2. Now implement a class for an Euler integrator. For a differential equation $\dot{x}(t) = f(t, x(t))$, the Euler method to advance a differential equation with step size h looks like this:

$$x_{n+1} = x_n + hf(t_n, x_n).$$

   The class is constructed with three arguments: The function f given as a Python function, an initial value x_0 and a step size h. Store the calculated points x_n and integration times in lists as member variables. The skeleton structure of the class is given in the snippet below.

```
class EulerSolver():
    # Constructor
    def __init__(self, f, x0, h):
        # Write code here

    # Step function
    def step(self):
        # Write code here
```

   Class constructors in Python are implemented by defining a method `__init__()`. The `self` keyword is used to reference the instance of the class and must be passed to every class method as the first argument. Your task is to implement the constructor and the `step()` method that advances the solution of the equation for one step of size h.

3. Test your class by integrating the logistic equation $\dot{x}(t) = x(t)(1 - x(t))$ with initial value $x_0 = 0.001$ and step size $h = 0.001$. Integrate the equation until a target time $t_f = 20$.

4. Using PIP, install *Matplotlib* which is a plotting library for Python.

5. Plot the solution obtained in part (c) using Matplotlib. Research how to do that in the documentation.

Problem 7 (*Jupyter Notebooks*) (5 points)

A nice feature in Python is the ability to run code in so called *Jupyter notebooks*. Notebooks are comprised of cells that can be run individually and also provide Markdown cells (formatted text or even LaTeXequations) right next to code cells.
Begin by installing JupyterLab using `$ pip install jupyterlab`. Then, from the command line, run `$ jupyter lab`. This should automatically open a browser window that greets you with a nice user interface.

- The interface consists of a file browser on the left, the main work area in the center, and various tabs and panels for different tools and extensions.
- To create a new Jupyter notebook, click the "+ Notebook" button in the file browser or go to "File" > "New" > "Notebook" from the menu bar.
- The notebook interface allows you to write and execute code in individual cells. You can choose the cell type as "Code" or "Markdown" using the dropdown menu in the toolbar. Code cells are used to write and run code, while Markdown cells are used for text and documentation.
- To run the code in a cell, you can either click the "Run" button in the toolbar or use the keyboard shortcut "Shift + Enter". The output will be displayed directly below the cell.
- You can add new cells by clicking the "+" button in the toolbar or by using the keyboard shortcuts "B" to add a cell below the current cell or "A" to add a cell above the current cell.

Solve this problem by handing in a notebook with content of your choice. It should feature Markdown and code cells. Create a heading and a math formula in Markdown. Write some code cells doing simple calculations. You can also try plotting some data using Matplotlib.