Advanced Programming Techniques

Aurel Vlaicu University Arad

Practical Session #1

Python introduction:

- Make sure you have Python 3.0 installed:
- Windows: `https://docs.conda.io/projects/conda/en/latest/user-guide/install/windows.html`
- Linux: `https://docs.conda.io/projects/conda/en/stable/user-guide/install/linux.html`
- Quick intro: `https://www.w3schools.com/python/python_intro.asp`
- Basic commands will be introduced in the first lab; Coding syntax is quite straightforward
- Install `Jupyter Notebook`: Google it! Straightforward through Anaconda/Miniconda or with `pip`. Advantage: facilitates teaching/learning, quick coding/running code, useful for projects: combine text cells with code cells.

What to do if you don't know a command/method? **Search the documentation! Use a search engine!**

Exercise 1 (**Basic Python commands).** In this exercise we look at the most basic things we can do in Python:

0. Comments: start with `#`.
1. Print a message: `print("Hello world!")`
2. Basic arithmetic operations
3. Arrays, lists, etc `https://www.w3schools.com/python/python_arrays.asp`
4. `if, else` statements; indentation!
5. Loops `for, while`
6. Function definition `def`.

Exercise 2. Define and test basic functions

1. Define a function which adds two numbers: two inputs, one output
2. Define a function which concatenates two arrays: output: [first input, second input]
3. Code the Insertion-Sort function taught in the course.

Exercise 3. Define and test basic recursive functions

1. Implement the Factorial function recursively.
2. Implement Fibonacci's sequence: $F_0 = 1, F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. Use both iterative and recursive approaches. Use a global counter (keyword `global` to keep track of the number of executions of the recursive version).
3. Can you implement Fibonacci recursively with linear complexity?
4. Implement the Merge-Sort function.
5. Implement the Insertion-Sort function recursively.

Exercise 4. String manipulation, find matching substrings

1. Strings are defined using single or double quotes quotes, for example: `'some string'`. Define a few strings of your choice and print them.

2. Compute the length of a string with `'len'`. Iterate and print all characters printed in a string: `s[i]`, where the index ranges from 0 to the length of the string.

3. Test the concatenation operator: `+`.

4. Test if the comparison operators `<,>,==` output the expected boolean variables corresponding to the lexicographic order. Search the documentation to be sure.

5. Download a DNA sequence data set from Kaggle: `https://www.kaggle.com/datasets/nageshsingh/dna-sequence-dataset/data?select=dog.txt`.

   Consider the problem defined in the course: *Find if the DNA sequence contains subsequences of length $M \ll N$ (for example $M = 4$).*

   - Load the DNA string from the text file. Create a new substring which is **significantly shorter** (for example 1000 characters), so that testing algorithms on it will be fast.
   - Construct a function which builds a list containing all substrings of size M from the DNA string.
   - Sort the list using the algorithm of your choice.
   - Answer the question if the string contains duplicate substrings of length M.