**Computational Thinking and Programming – A.Y. 2023/2024**

Written examination – 14/03/2024

Given name: _____

Family name: _____

Matriculation number: _____

University e-mail: _____

Enrolment a. year: [ ] 2023/2024 [ ] 2022/2023 [ ] 2021/2022 [ ] 2020/2021 [ ] other

Is it your first try? Yes | No

The examination is organised in three different sections:

- Section 1: basic questions [max. score: 16]. It contains four simple questions about the topics of the whole course. Each question requires a short answer. Each question answered correctly will give you 4 points (or less for partial answers).
- Section 2: understanding [max. score 8]. It contains an algorithm in Python, and you have to report the particular results of some of its executions according to specific input values.
- Section 3: development [max. score 8] It describes a particular computational problem to solve, and you are asked to write an algorithm in Python for addressing it.

You have 1 hour and 30 minutes for completing the examination. By the final deadline, you should deliver only the original text (i.e. this document) with the definitive answers to the various exercises that must to be written with a pen – pencils are not permitted. You can keep all the draft papers that you may use during the examination for your convenience – blank sheets will be provided to you on request.

**Section 1: basic questions**

1 – Being the variable *s* a string, which of the following Python codes are valid instructions?

- `s.add("c")`
- `s.add(1)`
- `s + "hello"`
- `s.strip()`
- `s.append("c")`
- `s + 6`

2 – Consider the following Python function:

```
def f(x, y):
    if x * x > y:
        return y - x
    else:
        return 0
```

What is the value returned by `f(4, 19)`?

3 – Write down a small function in Python that takes in input a positive integer and returns how many times it can be divided by 2 with 0 as remainer.

4 – Describe what are the two conditions that must hold to apply a greedy algorithmic technique.

**Section 2: understanding**

Consider the following functions written in Python:

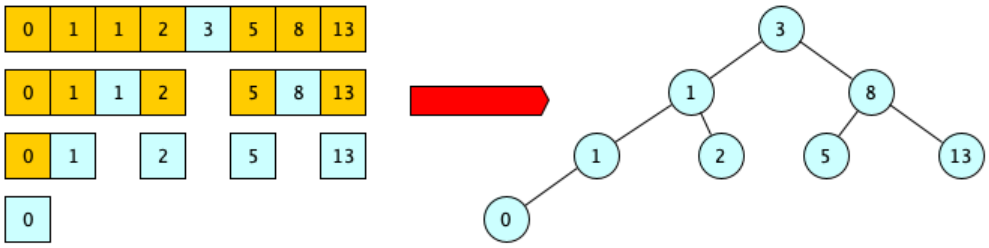
```
def sm(mat):
    new_m = []

    for d in mat:
        n_d = (int(d) + 1) % 10
        new_m.append(str(n_d))

    i = len(new_m) - 1
    if i > -1:
        return new_m[i] + sm("".join(new_m[:i]))
    else:
        return ""
```

Consider the variables `my_mat_number` containing the 10-digit strings of your matriculation number. What is the value returned by calling the function `sm` as shown as follows:

```
sm(my_mat_number)
```

**Section 3: development**

A **binary search tree** is a binary tree data structure where each node may have at most two children and the value of each node is greater than (or equal to) all the values in the respective node's left subtree and less than (or equal to) the ones in its right subtree. It can be built, recursively following an approach which recalls the binary search strategy, starting from a list of ordered items (e.g. a list of integers), where each item become a node of a tree.




As a reminder, the binary search strategy first checks if the middle item of a list is equal to item to search for and, in case this is not true, it continues to analyse the part of the list that either preceeds or follows the middle item if it is greater than or less than the item to search.

Write a *recursive* algorithm in Python – `def create_bst(ordered_list, parent)` – which takes in input an ordered list of integers `ordered_list` and a parent node `parent`, and returns the root node of the binary search tree created from the input list. In the first call, the function is run passing `None` as input of the second parameter `parent`, e.g.

```
create_bst([0, 1, 1, 2, 3, 5, 8, 13], None)
```

and returns the binary search tree (i.e. its root node) shown in the example above.