Computational Thinking and Programming – A.Y. 2020/2021

Written examination – 20/09/2021

Given name: _____

Family name: _____

Matriculation number: _____

University e-mail: _____

Group name: _____

Is it your first try? Yes | No

The examination is organised in three different sections:

- Section 1: basic questions [max. score: 8]. 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 either 2 points (full answer) or 1 point (partial answer).
- Section 2: understanding [max. score 4]. 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 4] 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 – Which of items in the following lists are algorithmic techniques?

- queue
- greedy
- backtracking
- dictionary
- list

2 – Identify all the mistakes introduced in the following implementation in Python of the linear search algorithm, and propose the corrections to them.

```
def linear_search(input_list, value_to_search):
      for position, item in enumerate(input_list):
            if position == value_to_search:
                  return item
```

3 – Write down a small function in Python that takes in input two floating number (e.g. 19.2 and 3.9) are returns the sum of their integer parts (e.g. 22).

4 – Introduce the tripartite classification (i.e. macro-sets) of programming languages, explaining their main characteristics.

Section 2: understanding

Consider the following functions written in Python:

```
def n(f_name, mat):
    chars = []
    first = ""

    for sn in mat:
        if first == "":
            first = sn
        i = int(sn)
        cur = f_name[i % len(mat)]
        chars.append(cur)

    result = "".join(chars)
    if result != "":
        return result[0] + n(result, mat.replace(first, ""))
    else:
        return ""
```

Consider the variable `my_mat` containing the string of the numbers in your matriculation number (e.g. "`0000123456`"), and the variable `my_full_name` containing string of your full name (i.e. given name plus family name separated by a space), all in lowercase. What is the value returned by calling the function `n` as shown as follows:

```
n(my_full_name, my_mat)
```

Section 3: development

The **bubble sort** is a sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. In practice, the algorithm consider all the possible pairs of consecutive items starting from the beginning of the list and it swaps them in case the first one of the pair in consideration is greater than the second one. Once considered all the pairs of consecutive items in the list, the algorithm starts again from the first pair, and stop the process only when it has one whole iteration without swapping any pair, by returning the final result. For instance, supposing to have the list of integer `[4, 3, 6, 7, 5]`, the following are the iterations that the bubble sort performs:

Iteration 1	*Iteration 2*	*Iteration 3*
[4, 3, 6, 7, 5] → 4 > 3, swap	[3, 4, 6, 5, 7] → 3 < 4, do not swap	[3, 4, 5, 6, 7] → 3 < 4, do not swap
[3, 4, 6, 7, 5] → 4 < 6, do not swap	[3, 4, 6, 5, 7] → 4 < 6, do not swap	[3, 4, 5, 6, 7] → 4 < 5, do not swap
[3, 4, 6, 7, 5] → 6 < 7, do not swap	[3, 4, 6, 5, 7] → 6 > 5, swap	[3, 4, 5, 6, 7] → 5 < 6, do not swap
[3, 4, 6, 7, 5] → 7 > 5, swap	[3, 4, 5, 6, 7] → 6 < 7, do not swap	[3, 4, 5, 6, 7] → 6 < 7, do not swap
List at the end of the iteration: [3, 4, 6, 5, 7]	List at the end of the iteration: [3, 4, 5, 6, 7]	Final list (since no swaps happened): [3, 4, 5, 6, 7]

Write an algorithm in Python – `def bubble_sort(value_list)` – which takes in input a list of values of the same type (e.g. integer) in input, and returns the same list but ordered.