Computational Thinking and Programming – A.Y. 2018/2019

Written examination – 20/09/2019

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 2 points.
- Section 2: understanding [max. score 4]. It contains an algorithm in Python, and you have explain what it does and 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 the following steps are defined by backtracking algorithms:

- solution exists
- recursive-step
- memorize
- combine
- leaf-lose

2 – Please consider the following function implementing the fibonacci dynamic programming algorithm:

```
def fib_dp(n, d=dict()):
    if n not in d:
        if n == 0 or n == 1:
            d[n] = 0
        else:
            d[n] = fib_dp(n+1, d) - fib_dp(n+2, d)
    return n
```

Identify the mistakes in the aforementioned code and correct it.

3 – Write down a small function in Python that takes in input a string and a number and returns *True* if the division of the number of characters in the input string by the input number does not return any remainder, otherwise it returns *False*.

4 – Explain what are the two main characteristics that a computational problem should show so as to be sure that the application of a greedy approach will bring to an optimal solution to the problem.

Section 2: understanding

Consider the following function written in Python:

```
def rs(gn, fn, m, lst=[]):
    if gn != "" and fn != "":
        if gn[0] < fn[0]:
            lst.append(int(m[0]))
        elif len(lst) > 1:
            v = lst[len(lst) - 1] * int(m[0])
            lst = lst[:len(lst) - 1] + [v]
        lst = rs(gn[1:], fn[1:], m[1:], lst)

    return lst
```

Consider the variable `my_gn` containing the **string** of your given name as written in the first page but in **lowercase** and **without** any space, the variable `my_fn` containing the **string** of your family name as written in the first page but in **lowercase** and **without** any space, and the variable `my_mn` the **string** of your matriculation number as written in the first page. What is the value returned by calling the function `rs` as shown as follows:

```
rs(my_gn, my_fn, my_mn)
```

Section 3: development

The function *multiple replace* allows one to replace either all or a fixed number of occurrences of a given character in a list with another character contained in an input list of characters. In particular, it takes in input four different values: the full string that should be modified, the character to replace in the string, a non-empty list of characters to use as replacement, and the number of occurrences of the character to replace that should be substituted with the related replacement (*None* means all).

Every time the function finds a character to replace c_i, it substitutes it with the character in the list of replacements positioned in the index equal to the number of occurrences of c that precedes c_i (c_1, c_2, …, c_{i-1}). In case such number of occurrences that precedes c_i is greater than the maximum index in the list of replacements, the function starts again from the beginning of the list of replacements.

For instance:

- *multiple_replace("mamma mia!", "m", ["n"], 3)* returns *"nanna mia!"*
- *multiple_replace("mamma mia!", "m", ["p", "l", "l"], 3)* returns *"palla mia!"*
- *multiple_replace("mamma mia!", "m", ["n", "s", "t"], None)* returns *"nasta nia!"*

Write a function in Python – `multiple_replace(s, c, r, o)` – that takes in input the string to modify (`s`), the character to replace (`c`), the list of replacements (`r`), and the maximum number of occurrences of `c` to replace (`o`, set to None if one wants to replace all the occurrences), and that returns the modified string.