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

Written examination – 15/07/2019

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 sets of languages are used to define programming languages:

- recursively enumerable languages
- machine language
- high-level programming languages
- low-level programming languages
- regular languages

2 – Please consider the following function implementing the line wrap greedy algorithm:

```
def line_wrap(text, line_width):
    result = []
    space_left = line_width
    line = []
    for word in text.split(" "):
        word_len = len(word)
    if word_len + 1 > space_left:
        result.append(" ".join(line))
        line = [word_len]
        space_left = line_width – word_len
    else:
        line.append(word)
        space_left = space_left - word_len + 1
    return "\n".join(result)
```

Identify the mistakes in the aforementioned code and correct it.

3 – Write down a small function in Python that takes in input a function and two integers and returns *True* if the result of the execution of the input function using the first integer as its input is equal to the second integer, otherwise it returns *False*.

4 – Describe what is the main difference between the divide and conquer algorithmic technique and the dynamic programming technique, and describe which are the pros and cons of the latter one.

Section 2: understanding

Consider the following function written in Python:

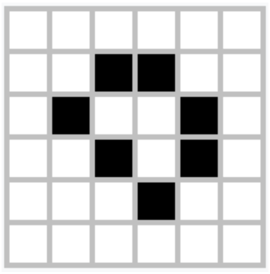
```
def r_char(idx, c_str, c_set):
    cur_c = c_str[idx % len(c_str)]
    if cur_c in c_set:
        c_set.remove(cur_c)


def char_n(gn, fn, mn):
    result = 0

    gn_set = set(gn.replace(" ", ""))
    fn_set = set(fn.replace(" ", ""))

    rn = 0
    for n in mn:
        rn += int(n)

    r_char(rn, gn, gn_set)
    r_char(rn, fn, fn_set)

    n_set = gn_set.union(fn_set)
    for idx in range(len(n_set)):
        if str(idx) in mn:
            result = result + idx
        else:
            result = result - idx
        print(result, idx)

    return result
```

Consider the variable `my_gn` containing the **string** of your given name as written in the first page but in **lowercase**, the variable `my_fn` containing the **string** of your family name as written in the first page but in **lowercase**, 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 `char_n` as shown as follows:

```
char_n(my_gn, my_fn, my_mn)
```

Section 3: development

The *Game of Life* is a game proposed by John Horton Conway in 1970. The universe in which the game is set is an infinite, two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, *alive* or *dead*. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent, as shown as follows (where the black cells is alive, the white ones are dead):



At each step in time, the following transitions occur, according to the following rules:

1. any live cell with fewer than two live neighbours dies in the next step, as if by underpopulation;
2. any live cell with two or three live neighbours lives on to the next step;
3. any live cell with more than three live neighbours dies in the next step, as if by overpopulation;
4. any dead cell with three live neighbours becomes a live cell in the next step, as if by reproduction.

Write a function in Python – `def alive_in_next_step(is_alive, neigh_alive)` – which takes in input a boolean defining the status of a particular cell (if *True* the cell is alive, otherwise it is dead) and a set that contains its neighbour cells that are currently alive, and returns *True* if the cell depicted by the input status will be alive in the next step, otherwise it returns *False*.