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

Written examination – 17/05/2021

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 the following sentences are true statements?

- COBOL is a low-level programming language
- In Python, a list is a mutable object
- Only undirected graph can be defined in Python
- The *recursive-step* is one of the main step of backtracking
- The first step of the Test-Driven Development concerns writing the new code

2 – Consider the following Python function:

```
def i(obj, x):
    if obj and x not in obj:
        return x
    else:
        return obj
```

What is the value returned by `i([], 6)`?

3 – Write down a small function in Python that takes in input two strings and returns a set containing the characters that are not contained in both the strings.

4 – List and describe the main characteristics that a computational problem should show 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 functions written in Python:

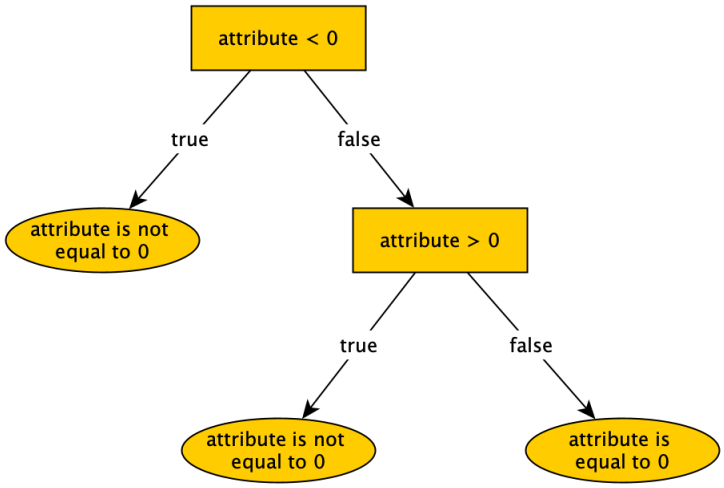
```
def f(m_string):
    t = 0
    cl = list()
    for c in m_string:
        cl.append(int(c))
        t = t + int(c)

    eo_value = t % 2 == 0
    cl = on(cl, eo_value)
    idx = t % len(m_string)
    return cl[idx], cl

def on(ln, flag):
    c_len = len(ln)
    if c_len > 1:
        if (flag and ln[0] > ln[c_len - 1]) or (not flag and ln[0] < ln[c_len - 1]):
            t = ln[0]
            ln[0] = ln[c_len - 1]
            ln[c_len - 1] = t
        m = c_len // 2
        return on(ln[0:m], flag) + on(ln[m:c_len], flag)
    else:
        return ln
```

Consider the variable `my_mat_string` containing the string of your matriculation number. What is the value returned by calling the function `f` as shown as follows:

```
f(my_mat_string)
```

Section 3: development

A **decision tree** is a flowchart-like structure in which each internal node represents a *test* on an attribute (e.g. whether a coin flip comes up heads or tails), each branch represents the outcome of the test, and each leaf node represents a class label (i.e. decision taken after computing all attributes). The paths from root to leaf represent classification rules. An example of decision tree is shown as follows:



The decision tree above allows one to check whether a given number (identified by the variable *attribute*) is equal to 0. For checking this, supposing to execute such a decision tree passing the number 3 as input, one has (a) to start from the root, (b) to execute the condition (i.e. $3 < 0$), (c) to follow the related branch (i.e. *false*), and (d) to repeat again the process if we arrived in an inner node or (e) to return the result if we arrived in a leaf node.

As shown in the figure, in a decision tree each internal node has always two children: the left one is reached when the condition the internal node specifies is *true*, while the right one is reached when the condition the internal node specified is *false*.

Write an algorithm in Python – `def exec_dt(decision, attribute)` – which takes in input a tree `decision` (which is represented by the root node of the tree defined as an object of the class `anytree.Node`) describing a decision tree, in which the name of each non-leaf node in the tree is a Python function – it means that we can see the node name as a function that can be executed by passing an input, e.g. considering the node `n`, we can call the function specified as its name passing the input between parenthesis, as usual: `n.name(56)`. Each Python function, to execute using `attribute` as input, returns either `True` or `False` if the condition defined by that function is satisfied or is not satisfied respectively. The algorithm return the leaf node reached by executing the decision tree with the value in `attribute`.