Computational Thinking and Programming – A.Y. 2017/2018

Written examination – 10/05/2018

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 – Briefly describe what is the machine language and what are the high-level programming languages, and highlight their the main differences.

2 - Consider the following data structures:

```
q = deque()  # queue
q.append(1)
q.append(2)
q.append(3)

s = deque()  # stack
s.append(4)
s.append(5)
s.append(6)
```

Write what `q.leftpop()` and `s.pop()` return respectively.

3 - The following list items describe the various passages of the dynamic programming approach, but they are listed in a wrong order. Sort these steps in the right order by assigning the a numbering position to all of them.

- [divide] split the input material into two or more balanced parts, each depicting a sub-problem of the original one
- [base case: solution exists] return the solution calculated previously to the problem if this is the case; otherwise
- [conquer] run the same algorithm recursively for every balanced part obtained in the previous step
- [base case: address directly] address the problem directly on the input material if it is actually depicting an easy-to-solve problem; otherwise
- [memorize] store the solution to the problem so as to reuse if needed by other recursive calls
- [combine] reconstruct the final solution of the problem by means of the partial solutions obtained from running the algorithms on the smaller parts of the input material

4 - Define what is a *recursive algorithm* and write down a simple recursive function in Python.

Section 2: understanding

Consider the following functions written in Python:

```
def resolve(email, group_name):
    l = list()

    for idx in reversed(range(len(email))):
        l.append(email[idx])

    d = dict()
    for c in group_name:
        add(d, c)

    r = ""
    for i in l:
        if i not in d or not remove(d, i):
            r = r + i

    return r


def add(d, i):
    if i not in d:
        d[i] = 0
    d[i] = d[i] + 1


def remove(d, i):
    if i in d and d[i] > 0:
        d[i] = d[i] - 1
        return True

    return False
```

Consider the variable `em` and `gp` containing the string of your email and the name of your group respectively. What is the value returned by calling the function `resolve` as shown as follows:

```
resolve(em, gp)
```

Section 3: development

In information retrieval, the *term frequency–inverse document frequency* (or *tf-idf*) is a numerical statistic that is intended to reflect how important a word is to a document in a corpus. It is based on two functions:

- the *term frequency*, `tf(t, d)`, which counts the number of times a term *t* occurs in document *d*;
- the *inverse document frequency*, `idf(t, d_list)`, which measures whether a term *t* is common or rare across all the documents in a list *d_list*, calculated as the logarithm of the division between the total number of documents in the list and the number of documents that contains the term *t*.

Thus, the *tf-idf* of a term *t* in a document *d* included in a collection of document *d_list* is simply the multiplication between its term frequency and its inverse document frequency.

Write an algorithm in Python – `tfidf(t, d, d_list)` – which takes a string *t* representing a term, a string *d* representing a document, and a list of strings *d_list* representing a collection of documents which includes also *d*, and that returns the *tf-idf* of the input term according to the document in that document list. As a simplification, all the input strings are composed only by lowercase English alphabetic characters with no punctuation. The logarithm function is available in Python within the module *math* (`import math`) and has the following signature: `math.log(n)` – e.g. `math.log(3)` calculates the logarithm of the number 3.