Computational Thinking and Programming – A.Y. 2022/2023

Written examination – 22/06/2023

Given name:	____________________
Family name:	____________________
Matriculation number:	____________________
University e-mail:	____________________

Enrolment a. year: [ ] 2022/2023 [ ] 2021/2022 [ ] 2020/2021 [ ] 2019/2020 [ ] other

Is it your first try? Yes | No

The examination is organised in three different sections:

- Section 1: basic questions [max. score: 16]. 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 4 points (or less for partial answers).
- Section 2: understanding [max. score 8]. 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 8] 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 – Being the variable *test* a string with alphabetic characters, which of the following Python codes are valid instructions?

- `return test + 5`
- `return test + "abc"`
- `return test, "abc"`
- `return int(test)`
- `return test.split(5)`
- `return test % len(test)`

2 – Consider the following Python function:

```
def f(s):
    res = len(s)
    for c in s:
        if c in "aeiou":
            res = res - 1
    return res
```

What is the value returned by `f("ciao")`?

3 – Write down a small function in Python that takes in input two integers and returns how many numbers to add to the smaller one to reach the greater one.

4 – Explain, by providing at least one example, while the *dynamic programming* approach is more efficient than the *divide and conquer* algorithmic technique.

Section 2: understanding

Consider the following functions written in Python:

```
from collections import deque


def first(mat_string, given_name):
    s = deque()

    for char in mat_string:
        n = int(char)
        if n > 0:
            s.append(n)

    return second(s, given_name)


def second(c, gn):
    if len(c) == 0:
        return 0
    else:
        i = c.pop()
        s = gn[(i - 1) % len(gn)]
        if s in "aeiou":
            return 2 + second(c, gn)
        else:
            return -1 + second(c, gn)
```

Consider the variable `my_given_name` containing string of your given name in lowercase and without spaces, and the variable `my_given_name` containing string of your matriculation number. What is the value returned by calling the function `first` as shown as follows:

```
first(my_mat_string, my_given_name)
```

Section 3: development

Trial division is one of the integer factorisation algorithms. The idea is to see if an integer *n* greater than 1, provided as input, can be divided by each number in turn from 2 to *n*. For example:

- for the integer *n = 12*, the list of factors dividing it is *2, 2, 3* (i.e. *12 = 2 * 2 * 3*);
- for the integer *n = 11*, the list of factors dividing it is 11 (i.e. 11 = 11, since 11 is prime andm thus, it can be divided by itself only);
- for the integer n = 15, the list of factors dividing it is *3, 5* (i.e. *15 = 3 * 5*).

The algorithm proceed by dividing the input number starting from the smallest possible number *f*, initially set to 2. If the division returns a reminder, it repeat the operation by incrementing *f* of one unit. Instead, if the division returns no reminder, *f* is added to the list of factors, and *n* will be assigned with the result of the division, before repeating the operation. For instance, considering *n = 18*, the initial *f = 2*, and the list of factors to return initially empty:

1. 18 / 2 = 9 (with no remainder) → list of factors: 2; n = 9
2. 9 / 2 = 4 (with remainder 1) → f = 3
3. 9 / 3 = 3 (with no remainder) → list of factors: 2, 3; n = 3
4. 3 / 3 = 1 (with no reminder) → list of factors: 2, 3, 3; n = 1

The algorithm stop when *f* is greater than *n*, and returns the list of factors.

Write an algorithm in Python – `def trial_div(n)` – which takes in input an integer `n` greater than 1, and returns the list with the factors dividing `n` according to the rules described above.