Exam in Introduction to programming

2022-01-04

Instructions. Answer the questions below. They are roughly ordered by difficulty, but independent. Some questions are broken down into subquestions, *e.g*, question 4 might consist of subquestions 4-1, 4-2, and 4-3; the subquestions are supposed to be treated in that order.

Provide your answers by editing the files inside the `answers-xxxx` directory. You are *not supposed to create any new files*, nor change the names of the provided files in that directory.

Before handing in, replace the `xxxx` in the `answers-xxxx` directory with your ITU student ID, *e. g.*, `answers-jegp`. Hand in the renamed `answers-xxxx` directory as a single zip file called `answers-xxxx.zip`, where xxxx is again your student ID.

Some questions require extra input data. Such data is provided in a directory of the same name as the question. For instance, data for question four would be in the directory called `4`. In the questions below, the file name is given relative to the program you are supposed to write, so the data file `names.txt` for question four would be referred to as `../4/names.txt`, assuming you are standing in the `answers-xxxx` directory.

Any free text, such as program comments or explanations, can be written in Danish or English, but English is preferred.

Points. At most 100 points can be earned in this exam.

Question 1 (7 Points)

What do the following expressions evaluate to?

```
4 + 2 - 3
str(4) + '2' * 3
23 - 1 * 2
3 ** 5
not (False or True)
not False or True
3 * [3]
```

Question 2 (7 Points)

What is printed by the following 7 print statements?

```
a = {'Copenhagen': 'Denmark', 'Helsinki': 'Finland', 'Oslo': 'Norway'}
s = "Isn't there someone missing?"
print(len(a))
print(a['Oslo'])
print(s[6:11])
print(a['Copenhagen'].upper())
print(a['Copenhagen'][5])
```

```
print(a['Helsinki'][::-1])
print(f"{s[:12]}{a['Oslo']} {s[20:]}")
```

Question 3 (8 Points)

What is the purpose of the following code?

```
a = True

for x in L:
    for y in L:
        if x != y:
            a = False
```

Pick one (and exactly one) of the following:

a. Check if `x` or `y` are present in `L`.

b. Check if all values in `L` are the same.

c. Count the number of `a` in `L`.

d. Check if the length of `L` is at least 2.

e. Check if `L` is sorted.

Question 4 (12 Points)

Consider the following code that describes a function that is supposed to work on a list `L` of integers:

```
def f(L):
    s = 0
    for x in L:
        if x > 0:
            s = s + x
    return s
```

4-1 (2 Points)

Provide a list `L` such that `f(L)` returns 2.

4-2 (10 Points)

Rewrite this code such that the `for` loop is replaced with a `while` loop.

Provide *two* assert statements that check whether the function produces the correct output.

Question 5 (13 Points)

You want to manage some kind of database to store people and their favorite place. A Python dictionary `favorite_places` seems to be a good choice to store such a database.

Provide the solution to these three subtasks in a single file `5.py`.

5-1 (3 Points)

Assume that you start with the code

```
favorite_places = {}
```

Add the following three entries to this dictionary:

- Sarah's favorite place is Spain.
- Peter's favorite place is Italy.
- Thore's favorite place is Spain.

Both the key and the value are supposed to be strings.

You are not allowed to change the first line, i.e., your program has to start with `favorite_places = {}`.

5-2 (4 Points)

Define a function `print_places(favorite_places)` that takes as argument a dictionary as described above. For each person X and her favorite place Y found in the dictionary, the function shall print

```
X's favorite place is Y.
```

In other words, you are supposed to iterate through the items of the dictionary.

Don't worry about capitalization.

5-3 (6 Points)

Define a function `count_favorites(favorite_places, place)` that takes two arguments: a dictionary as above, storing people with their favorite places, and the name of a place. It returns an integer that counts the number of times the given `place` occurs as the favorite place in `favorite_places`.

For example, `count_favorites(favorite_places, "Spain")` shall return 2 with the dictionary described in 5-1. **Add two assert statements that check if `count_favorites` behaves correctly.**

Question 6 (10 Points)

Every year, Martin has the same problem: On the one hand, his introduction to programming class has exercises in a number of different rooms; on the other hand, he gets a certain number of TAs that he can hire. Sometimes, there are more rooms than TAs, leaving rooms unstaffed!

He wants to automate the process of reminding the administration that the number of assigned TAs does not suffice to run the exercise sessions. Moreover, if he gets enough TAs, he wants to automatically assign them to rooms.

6-1 (4 Points)

Write a function `check_ta_count` that takes two arguments: the number of `rooms` assigned to the exercise session, and the number of `tas` that he can hire. If `tas` is at least as large as `rooms`, then your program should print

> Thank you very much for such a generous support.

Otherwise, your program should print

> Unfortunately, I need more TAs to run the exercise sessions.

For example, running `check_ta_count(11, 11)` should print the "Thank you" message, and `check_ta_count(6, 2)` should print the "Unfortunately, …" message.

Note: You do not have to care about user input, only the function definition is important.

6-2 (6 Points)

Write a function `assign_tas` that takes two arguments: the number of `rooms` and the number of `tas`. The function prints a suggestion how to assign the TAs to the rooms so that no room is left empty. For example, `assign_tas(3, 4)` could print:

```
TA 1 goes to room 1.
TA 2 goes to room 2.
TA 3 goes to room 3.
TA 4 goes to room 3.
```

You can assume that there are enough TAs so that every room can be staffed with at least one. Each TA has to be assigned to exactly one room. **You do not have to provide a solution that balances the number of TAs in each room.**

Question 7 (16 Points)

Write a class `VaccineRecord` for keeping track of the vaccine treatment of a single patient. In particular, for keeping track of the number of vaccines administered and to answer whether it is time for a booster shot. It is time for a booster shot, if at least 7 months have passed since the latest vaccination, or no vaccination has yet been administered. Vaccines can be added in any order, in case someone forgot to add an earlier vaccine.

Provide the solution to these two subtasks in a single file `7.py`.

7-1 (10 Points)

Implement the class and the following methods:

- a constructor, that takes the name of a person and the name of the vaccine as argument.
- an `add_vaccination(month, year)` method, that does what it says. Note that we only keep track of the month and the year.
- a `vaccination_count()` method, that returns how many vaccines have been administered.

- a `print()` method, that prints what the current state of vaccination is (details depend on how you have chosen to model the class). At least, it should include the name of the person, the name of the vaccine, the vaccination count, and the date of the last vaccination.

Make sure to add documentation to the class and all methods.

7-2 (6 Points)

Implement the `is_time_for_booster(month, year)` method. Add documentation to this method as well.

Here is an example of the intended behaviour. In particular, it shows the intended behavior for `is_time_for_booster`.

```
>>> peter = VaccineRecord("Peter", "Pfizer")
>>> katrine = VaccineRecord("Katrine", "Moderna")
>>> peter.vaccination_count()
0
>>> peter.add_vaccination(7, 2021)
>>> peter.vaccination_count()
1
>>> peter.is_time_for_booster(12, 2021)
False
>>> peter.is_time_for_booster(1, 2022)
False
>>> peter.is_time_for_booster(2, 2022)
True
>>> peter.is_time_for_booster(3, 2022)
True
>>> peter.add_vaccination(4, 2021)
>>> peter.vaccination_count()
2
>>> peter.is_time_for_booster(12, 2021)
False
>>> katrine.vaccination_count()
0
>>> katrine.add_vaccination(5, 2021)
>>> katrine.is_time_for_booster(12, 2021)
True
>>> katrine.vaccination_count()
1
```

By the lines starting with `>>>` we refer to lines in the Python program, and we show the printed output right underneath.

Question 8 (15 Points)

You are part of the investigation team that looked into what happened leading up to the eradication of mink farming in Denmark. Your task is to write a program for filtering through the text messages of high ranking officials of the Danish government.

The input is given in two files:

- `../8/messages.txt`, containing the body of each message, with one line per message.
- `../8/meta.txt`, containing information about who sent each message when. Each line starts with three numbers; year month day. The rest of the line is the author of the message.

The two files contain the same number of lines, and the `i`th line in the `meta.txt` file describes the `i`th line in the `messages.txt`.

As an example, the message on line 13 in `../8/messages.txt`

> Det er meget, meget skidt det her. Ægte tillidsbrud.

was written by Barbara on the 9. November, 2020.

OBS: The following task asks you to use object-oriented programming. You can deviate from that and not use classes, but you will not be able to earn full points.

Solve the following four tasks in a single file `8.py`.

8-1 (3 Points)

Write a class `DataBase` with a constructor, that takes two file names as arguments. The first is the filename for the file containing the messages, and the second is for the file containing the meta data. One way of calling this would be

```
db = DataBase('../8/messages.txt', '../8/meta.txt')
```

It is up to you how the data is represented internally; whether you need helper classes and helper functions is up to you as well.

8-2 (4 Points)

Add a method `show_by_author`, which takes a single string as argument. It must then print all the messages, where the author matches the given argument.

Example: `db.show_by_author("Henrik")` should print two messages, the one on line 2 and the one on line 5 in `../8/messages.txt`.

8-3 (4 Points)

Add a method `show_date_range`, which takes six integer arguments; `from_year`, `from_month`, `from_day`, `to_year`, `to_month`, `to_day`. It must print all the messages in the given range of dates.

8-4 (4 Points)

Add a method `show_by_match`, which takes a single string as argument. It must print all messages, where the message contains the argument.

For example, `db.show_by_match("mink")` should print the message

> Lige om lidt - vi skal lige have godkendt præciseret brev til minkavlerne

(and nothing else.)

Question 9 (12 Points)

Anna and Betty are best friends, but they each have a lot of other friends, some shared and others not. They want to know how many friends they have, combined; that is, the number of people that are friends with at least one of them. Anna proposes the following piece of code to settle this question:

```
def has_friend(friends, friend):
    for f in friends:
        if f == friend:
            return True
    return False

def people_they_know(friends_anna, friends_betty):
    common = []
    for friend_anna in friends_anna:
        if not has_friend(common, friend_anna):
            common.append(friend_anna)
        for friend_betty in friends_betty:
            if not has_friend(common, friend_betty):
                common.append(friend_betty)
    return len(common)
```

As an example, `people_they_know(["Cecilie", "Katrine", "Rasmus"], ["Rasmus", "Adam"])` returns 4.

9-1 (2 Points)

What is the worst case running time complexity of `has_friend`:

a. constant

b. logarithmic in the length of the list

c. linear in the length of the list

d. quadratic in the size of the list

9-2 (2 Points)

Run `people_they_know` on some larger lists. For simplicity, let `friends_anna` and `friends_betty` have the same number of elements and let us assume that `friends_anna` and `friends_betty` have no elements in common and that each list contains only distinct elements. What is the length of the largest lists that the program can handle in a few seconds?

a. 50

b. 500

c. 5000

d. 50000

9-3 (4 Points)

If Anna and Betty each have at most `N` friends, what is the worst case running time of Anna's implementation of `people_they_know`?

a. logarithmic in `N`

b. linear in `N`

c. quadratic in `N`

d. higher than all above

9-4 (2 Points)

Write an improved implementation of `people_they_know`. You don't have to reuse any of the code, but the method name and parameters must be the same.

9-5 (2 Points)

If Anna and Betty each have at most `N` friends, what is the worst case running time of your implementation of `people_they_know`?

a. constant

b. logarithmic in N

c. linear in N

d. quadratic in N