CS1010S— Programming Methodology
National University of Singapore

Practical Examination

17 Apr 2021

Time allowed: 1 hour 30 minutes

Instructions (please read carefully):

1. This is **an open-book exam**. You are allowed to bring in any course or reference materials in printed form. No electronic media or storage devices are allowed.
2. This practical exam consists of **three** questions. The time allowed for solving this test is **1 hour 30 minutes**.
3. The maximum score of this test is **20 marks**. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. You are advised to attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
5. While you are also provided with the template `practical-template.py` to work with, your answers should be submitted on Coursemology. Note that you can **only run the test cases on Coursemology for a limited number of tries** because they are only for checking that your code is submitted correctly. You are expected to test your own code for correctness using IDLE and not depend only on the provided test cases. Do ensure that you submit your answers correctly by running the test cases at least once.
6. In case there are problems with Coursemology.org and we are not able to upload the answers to Coursemology.org, you will be required to name your file `<mat no>.py` where `<mat no>` is your matriculation number and leave the file on the Desktop. If your file is not named correctly, we will choose any Python file found at random.
7. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology and correctly left in the desktop folder at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Practical Exam, or the parts that are not uploaded correctly.
8. Please note that while sample executions are given, it is **not sufficient to write programs that simply satisfy the given examples**. Your programs will be tested on other inputs and they should exhibit the required behaviours as specified by the problems to get full credit. There is no need for you to submit test cases.
9. While you may use any built-in function provided by Python, you may not import functions from any packages, unless otherwise stated and allowed by the question.

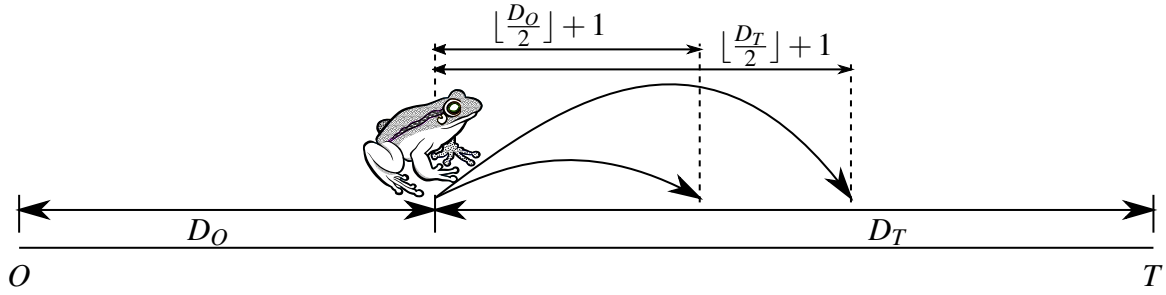
GOOD LUCK!

Question 1 : Jump! [5 marks]

A frog has to jump point to point in a straight line to reach a target T, which is an integer distance away. At each point, the frog can only choose between two distances to jump:

- With D_O being the distance to the origin, the frog may jump forward $\lfloor \frac{D_O}{2} \rfloor + 1$ units.
- With D_T being the distance to the destination, the frog may jump forward $\lfloor \frac{D_T}{2} \rfloor + 1$ units.

The frog must exactly land on the destination, i.e. it cannot overshoot.



Implement the function `jump` that takes as input the integer distance to the target T, and returns the minimum number of jumps necessary for the frog reach the target.

For example, to reach a target of distance 9, the frog requires at least 3 jumps: first covering 5 units ($\lfloor \frac{9}{2} \rfloor + 1$), then covering 3 units ($\lfloor \frac{5}{2} \rfloor + 1$)) and finally 1 unit ($\lfloor \frac{1}{2} \rfloor + 1$).

To reach a target of distance 10, the frog only needs 2 jumps: first covering 6 units ($\lfloor \frac{10}{2} \rfloor + 1$), then covering 4 units ($\lfloor \frac{6}{2} \rfloor + 1$)).

Sample Execution:

```
>>> jump(1)
1
>>> jump(2)
1
>>> jump(3)
2
>>> jump(4)
2
>>> jump(5)
2
>>> jump(6)
2
>>> jump(7)
2
>>> jump(8)
2
>>> jump(9)
3
>>> jump(10)
2
```

```
>>> jump(11)
3
>>> jump(12)
3
>>> jump(13)
3
>>> jump(14)
3
>>> jump(15)
3
>>> jump(16)
3
>>> jump(17)
4
>>> jump(18)
3
>>> jump(19)
4
>>> jump(20)
4
```

Question 2 : Registrar Office Woes [10 marks]

Important note: You are provided with a data file `modreg_timetable.csv` for this question for testing, but your code should work correctly for *any* data file with the same column format and *will* be tested on other data files. There is no guarantee that rows will be ordered in any way in the data files.

Once again, you find yourself back to the Registrar's office. After helping them schedule classes so excellently previously, your boss has another, more challenging task for you.

You are given a data file of students' schedules over a semester. Each row represents a scheduled class for a particular student. A scheduled class will always start and end on the same date. The data contains the following columns in specific order and format:

student_id	module_code	class_code	date (YYYY-MM-DD)	start (hh:mm)	end (hh:mm)

A. Your boss wishes to know the number of students enrolled into certain modules, to ensure the modules are not under-subscribed or over-subscribed.

Implement the function `get_enrollment` which takes as required inputs the **data filename**, and the **module code**, as well as the **class code** as an optional input. It returns the total number of unique student IDs who are currently enrolled in the given module code and class code. If the class code is not supplied, then return the total number of unique students enrolled in the given module code. [5 marks]

Sample Execution:

```
>>> filename = "modreg_timetable_trimmed.csv"
>>> get_enrollment(filename, "GEQ1000")
81
>>> get_enrollment(filename, "ACC1701", "LV2") # optional class code
10
>>> get_enrollment(filename, "CFG1002", "L02") # optional class code
59
>>> get_enrollment(filename, "CS1010S")
261
>>> get_enrollment(filename, "CS1010S", "R01") # optional class code
30
>>> get_enrollment(filename, "CS1010S", "T01") # optional class code
5
```

B. While planning for the students' timetables, you are also interested in how busy the busiest students are during the semester. To determine how busy a particular student is, obtain the total the number of hours of that student's scheduled classes for a given range of dates.

Implement the function `top_k_busy` which takes as input the **data filename**, a **start date**, an **end date**, and a positive integer **k**. Return the list of top-k tuple pairs—student ID (`int`), total hours (`float`). The list should arranged in descending order, based on the total number of hours of classes scheduled from the start date to the end date (both inclusive), rounded off to 1 decimal place. [5 marks]

As usual, use **standard competition ranking**. This means that all other students tied with k-th ranked student should also be included (*e.g. 1-2-2-4*). Ties are given the same ranking, and a gap is left in the ranking numbers. This gap is one less than the number of tied items.

Sample execution:

```
>>> top_k_busy(filename, "2021-01-01", "2021-04-30", 1) # top-1
[(239, 446.5)]
>>> top_k_busy(filename, "2021-01-01", "2021-04-30", 3) # top-3
[(239, 446.5), (152, 396.0), (22, 383.0)]
>>> top_k_busy(filename, "2021-02-01", "2021-02-28", 3) # feb only
[(239, 103.5), (57, 93.0), (152, 93.0)] # no fixed ordering for ties
   [(239, 103.5), (152, 93.0), (57, 93.0)] is also acceptable
```

Hint: Given `s = "21:31"`, the code `s.split(":")` will return a list `["21", "31"]`.

Hint: The dates are in ISO-format, which means string ordering respects chronological ordering: `"2021-01-01" < "2021-02-28"`, 1st Jan 2021 is earlier than 28th Feb 2021.

Hint: The function `round(9.654, 1)` rounds off `9.654` to 1 decimal place to return `9.7`.

Question 3 : Attack on Titan [5 marks]

Titans are a race of giant, man-eating humanoids that serve as the catalyst for the events in the Attack on Titan series. Titans do not possess a complete, functioning digestive tract; they merely have a stomach-like cavity that eventually fills up with what they swallow.

The Scout Regiment is a branch of the military that is most actively involved in direct Titan combat. Due to their constant expeditions and encounters with Titans, Scout Regiment members are the most skilled at killing Titans.

For this question, we will be modelling Titans and scouts. Titans come in different heights and scouts have different skill levels.

A scout can only kill a Titan whose height lower than his/her skill level. Otherwise he/she gets eaten by the Titan. Likewise, a Titan can only eat a scout if its height is equal or higher than the scout's skill level. Otherwise the Titan gets killed.

Contrary to real life, when a scout gets eaten, he/she is not killed. Instead, the scout is trapped in the stomach-like cavity of the Titan. When the Titan is killed, its belly is sliced open and all the previously eaten scouts are freed.

The class `Titan` models a Titan. It is initialised by a number, which is its height. It has only one method, which is `eat`. `eat` takes as input a scout, and returns a string based on the following order of conditions:

- `'Titan is already dead!'`, if the Titan has already been killed by a scout.
- `'<scout name> is currently eaten!'`, if the scout is currently eaten by a titan.
- If the Titan's height is equal or greater than the scout's skill level, then the Titan eats the scout, and the string `'A <Titan height>m Titan eats <scout name>!'` is returned.
- Otherwise, the Titan is killed by the scout, and all the scouts in its belly are freed. The string `'<scout name> kills a <Titan height>m Titan!'` is returned, joined with the strings `' <eaten scout name> has been freed!'` for each scout freed. The eaten scouts are freed in the reverse order of when they were eaten by the Titan.

The class `Scout` models a scout. It is initialised with a name, and a skill level. It has only one method, which is `attack`. `attack` takes as input a Titan, and returns a string based on the following order of conditions:

- `'<scout name> is currently eaten!'`, if the scout is currently eaten by a titan.
- `'Titan is already dead!'`, if the Titan has already been killed by a scout.
- If the Titan's height is equal or greater than the scout's skill level, then the Titan eats the scout, and the string `'A <Titan height>m Titan eats <scout name>!'` is returned.
- Otherwise, the Titan is killed by the scout, and all the scouts in its belly are freed. The string `'<scout name> kills a <Titan height>m Titan!'` is returned, joined with the strings `' <eaten scout name> has been freed!'` for each scout freed. The eaten scouts are freed in the reverse order of when they were eaten by the Titan.

Provide an implementation for the classes `Titan` and `Scout`.

Sample Execution:

```
>>> t3 = Titan(3)
>>> t5 = Titan(5)
>>> t10 = Titan(10)

>>> eren = Scout("Eren", 5)  # suicidal maniac
>>> armin = Scout("Armin", 7)
>>> mikasa = Scout("Mikasa", 999)  # much imba <3

>>> eren.attack(t3)
'Eren kills a 3m Titan!'

>>> t3.eat(eren)
'Titan is already dead!'

>>> eren.attack(t3)
'Titan is already dead!'

>>> eren.attack(t5)
'A 5m Titan eats Eren!'

>>> eren.attack(t10)
'Eren is currently eaten!'

>>> t5.eat(eren)
'Eren is currently eaten!'

>>> t5.eat(armin)
'Armin kills a 5m Titan! Eren has been freed!'

>>> eren.attack(t10)
'A 10m Titan eats Eren!'

>>> armin.attack(t10)
'A 10m Titan eats Armin!'

>>> mikasa.attack(t10)
'Mikasa kills a 10m Titan! Armin has been freed! Eren has been
   freed!'
```

You are advised to solve this problem incrementally. Even if you cannot fulfil all the desired behaviours, you can still get partial credit for fulfilling the basic behaviours.

— E N D O F P A P E R —