CS1010S Programming Methodology
National University of Singapore

Practical Examination

13 Nov 2021

Time allowed: 1 hour 30 minutes

Instructions (please read carefully):

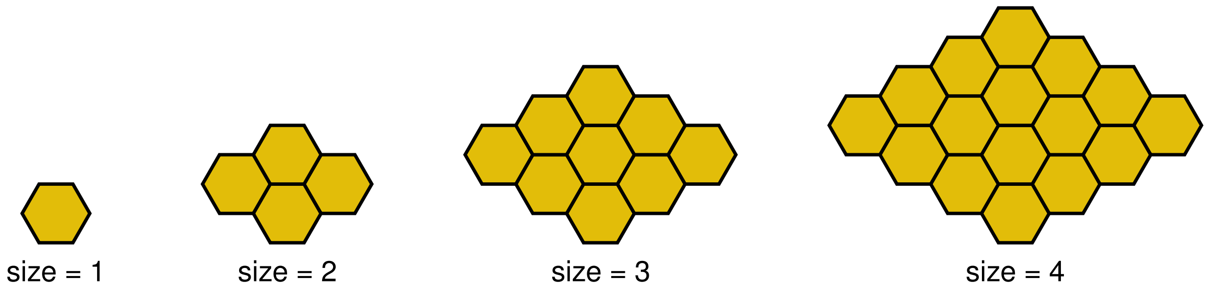
1. This is **an open-book exam**. You are allowed to bring in any course or reference materials in printed form. No electronic media or storage devices are allowed.
2. This practical exam consists of **three** questions. The time allowed for solving this test is **1 hour 30 minutes**.
3. The maximum 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 : Honeycomb Attack! [5 marks]

Dalgona candy, also known as honeycomb candy, is seeing a revival in popularity recently thanks to Squid Game! Jumping on the trend, you also want to create your own honeycombs. Before you proceed, the frontman Gerald wants you to prove your worthiness by telling him how many walls there are in a honeycomb, given some non-negative *size* where *size* ≥ 0.

Kindly demonstrate your iterative/recursive knowledge to prove yourself worthy, otherwise there may be dire consequences...



Implement the function `num_walls` that takes as input `size` and returns the number of walls in the honeycomb of the input `size`.

Sample Execution:

```
>>> num_walls(0)
0
>>> num_walls(1)
6
>>> num_walls(2)
19
>>> num_walls(3)
38
>>> num_walls(4)
63
>>> num_walls(5)
94
>>> num_walls(6)
131
>>> num_walls(7)
174
>>> num_walls(8)
223
>>> num_walls(9)
278
>>> num_walls(10)
339
```

```
>>> num_walls(11)
406
>>> num_walls(12)
479
>>> num_walls(13)
558
>>> num_walls(14)
643
>>> num_walls(15)
734
>>> num_walls(16)
831
>>> num_walls(17)
934
>>> num_walls(18)
1043
>>> num_walls(19)
1158
>>> num_walls(20)
1279
>>> num_walls(50)
7699
```

Question 2 : Cost-more-lor-gee [10 marks]

```
Subject: Background (OK to skip)

Dear student,

Why is the email service for my learning management system,
Cost-more-lor-gee, so expensive? Please check the email logs and
figure out where the emails are being sent to and report back.

Warmest Regards,
Ben
```

Important note: You are provided with a data file `email_logs.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.

Each row in an email log represents an email sent, with the first row being the header. The log contains the following columns in the specific fixed order and format:

event	resource	user_id	date (YYYY-MM-DD)	time (hh:mm:ss)

A. Emails are sent to subscribers who have subscribed to an event and resource. By counting the number of unique user IDs for each event and resource, we can obtain the number of subscribers.

Implement the function `num_subscribers` which takes as required inputs the **data filename**, the **event**, and the **resource**. It returns the total number of **unique** user IDs who have received at least one email of the given event and resource. [5 marks]

Sample Execution:

```
>>> fn = "email_logs.csv"
>>> num_subscribers(fn, "Comment", "Final Review: Question 5")
4
>>> num_subscribers(fn, "Announcement", "Desparado Remedial for PE")
754
>>> num_subscribers(fn, "Forum Reply", "L11")
182
>>> num_subscribers(fn, "Forum Reply", "M14")
36
>>> num_subscribers(fn, "Forum Reply", "M15")
11
```

B. Prof Ben wants to know the students who have received the highest daily average of emails within a certain date period. The daily average is simply the total number over the number of days.

Implement the function `top_k_avg_emails` 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—user ID (`int`), daily average emails (`float`). The list should arranged in descending order, based on the daily average emails received between the start date and end date (both inclusive), rounded off to 2 decimal places.

A helper function `num_days` is provided to you. It takes in two dates in ISO-format and returns the number of days from the start to the end.

As usual, use **standard competition ranking**. This means that all other user IDs tied with k-th ranked user ID 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. [5 marks]

Sample execution:

```
>>> top_k_avg_emails(filename, "2021-11-05", "2021-11-05", 2)
[(199368, 51.0), (903753, 51.0)]
>>> top_k_avg_emails(filename, "2021-11-06", "2021-11-06", 5)
[(527072, 45.0), (611569, 41.0), (983325, 39.0), (301261, 39.0),
   (199945, 38.0)]
>>> top_k_avg_emails(filename, "2021-11-05", "2021-11-06", 1)
[(527072, 44.0)]
>>> top_k_avg_emails(filename, "2021-11-05", "2021-11-07", 2)
[(527072, 41.67), (199368, 40.33)]
>>> top_k_avg_emails(filename, "2021-11-05", "2021-11-11", 3)
[(527072, 21.86), (301261, 21.29), (983325, 20.71)]
```

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(9.655, 2)` rounds off `9.655` to 2 decimal place to return `9.66`.

Question 3 : Sotong Game [5 marks]

A mysterious invitation to join the game is sent to students at dire risk of failing. Students from all faculties across NUS are locked into a secret location where they play a series of games in order to win extra S/U credits. Every game is a traditional children's game such as Zero-point and Pepsi-Cola 1-2-3. The consequence of losing is death.

During each game, players are given a choice to vote to stop the game and go home. If a majority of the players in the game votes, the game is stopped and all players who voted will be sent home without receiving any S/U credits. Those who did not vote will continue to play other games. Players who have left the game, either by winning or losing will not count towards the vote.

For this question, we will be modelling `Player` and `Game`. Both classes are initialised with a name.

`Player` supports the following methods:

- `join` takes a `Game` as input, and returns a string based on the following conditions:
  - `'<player name> is dead'` if the player is dead.
  - `'<player name> has gone home'` if the player has voted and got sent home.
  - `'<player name> is currently playing <some game name>'` if the player is currently playing in a game.
  - Otherwise, the player joins the game and the string `'<player name> is now playing <game name>'` is returned.
- `leave` takes in a reason (which is a string) as input, and returns a string based on the following conditions:
  - `'<player name> is not playing any game'`, if the player is not currently playing any game (or maybe even dead).
  - If the reason given is `'win'`, the player leaves the game, and the string `'<player name> has won <game name>'` is returned.
  - If the reason given is `'lose'`, the player leaves the game and becomes dead, and the string `'Bang! <player name> has lost <game name>'` is returned.
- `vote` takes no inputs.
  - `'<player name> is not playing any game'` is returned, if the player is not currently playing any game (or maybe even dead).
  - `'<player name> votes to stop'` is returned. The player's vote is registered, and if majority (more than half) of the current players in the game has voted, every player will leave the game. Players who have voted will be sent home and not allowed to join any more games. Players cannot rescind their vote.

    Note that a majority vote could also result from players having left the game.

Provide an implementation for the classes `Player` and `Game`. You may add other additional methods and properties to the classes as needed.

Sample Execution:

```
>>> darren = Player('Darren')
>>> linus = Player('Linus')
>>> matthew = Player('Matthew')
>>> nigel = Player('Nigel')
>>> russell = Player('Russell')
>>> sean = Player('Sean')
>>> terry = Player('Terry')

>>> zero_point = Game('Zero Point')
>>> pepsi_cola = Game('Pepsi Cola 1-2-3')

>>> darren.join(zero_point)
'Darren is now playing Zero Point'

>>> darren.join(pepsi_cola)
'Darren is currently playing Zero Point'

>>> linus.join(zero_point)
'Linus is now playing Zero Point'

>>> matthew.join(zero_point)
'Matthew is now playing Zero Point'

>>> nigel.join(zero_point)
'Nigel is now playing Zero Point'

>>> russell.join(zero_point)  # 5 players in the game
'Russell is now playing Zero Point'

>>> darren.vote()   # 1 of 5 players voted
'Darren votes to stop'

>>> linus.vote()    # 2 of 5 players voted
'Linus votes to stop'

>>> linus.vote()    # repeated votes do not count
'Linus votes to stop'

>>> sean.vote()
'Sean is not playing any game'

>>> sean.leave('win')
'Sean is not playing any game'

>>> matthew.leave('lose')    # 4 players left in game
'Bang! Matthew has lost Zero Point'
```

```
>>> russell.leave('win')     # 3 players left
'Russell has won Zero Point'
# Game ends as 2 of 3 players voted

>>> darren.join(pepsi_cola)
'Darren has gone home'

>>> linus.join(pepsi_cola)
'Linus has gone home'

>>> matthew.join(pepsi_cola)
'Matthew is dead'

>>> nigel.join(pepsi_cola)
'Nigel is now playing Pepsi Cola 1-2-3'

>>> russell.join(pepsi_cola)
'Russell is now playing Pepsi Cola 1-2-3'

>>> nigel.vote()  # 1 of 2 players voted
'Nigel votes to stop'

>>> sean.join(pepsi_cola)
'Sean is now playing Pepsi Cola 1-2-3'

>>> terry.join(pepsi_cola)
'Terry is now playing Pepsi Cola 1-2-3'

>>> sean.vote()   # 2 of 4 players voted
'Sean votes to stop'

>>> terry.vote()  # 3 of 4 players voted
'Terry votes to stop'
# Game ends with majority vote

>>> russell.vote()  # dude the game already ended
'Russell is not playing any game'

>>> jgp = Game('Ji Gu Pa')
>>> sean.join(jgp)
'Sean has gone home'

>>> terry.join(jgp)
'Terry has gone home'

>>> russell.join(jgp)
'Russell is now playing Ji Gu Pa'
```

```
>>> russell.leave('win')
'Russell has won Ji Gu Pa'
```

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 —