CS1010X — Programming Methodology
National University of Singapore

Re-Practical Examination

15 June 2019

Time allowed: 2 hours

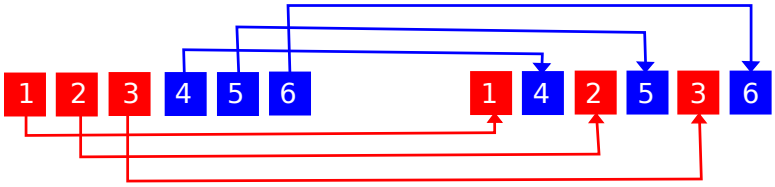
Instructions (please read carefully):

1. This is **an open-book exam**.
2. This practical exam consists of **three** questions. The time allowed for solving this quiz is **2 hours**.
3. The maximum score of this quiz is **30 marks** and you will need to answer all three questions to achieve the maximum score. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question. If you are taking this quiz as a re-exam, your maximum score will be capped at **20 marks**.
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. Your answers should be submitted on Coursemology.org. You will be allowed to run some public tests cases to verify that your submission is correct. Note that you can run the test cases on Coursemology.org up to a **maximum of 5 times** 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 on Coursemology.org. Do however ensure that you submit your answers correctly by running the test cases at least once.
6. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology 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. You have been warned.
7. 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.

GOOD LUCK!

Question 1 : Shuffling Mania [10 marks]

A. You managed to secure an internship with MBS to develop a new online casino program. You have been assigned to write a function `shuffle` that takes a list and returns a new list with the elements interleaved k number of times. An illustration of how this interleaving operation works is shown below. How this works should also be clear from the sample execution. [5 marks]



Sample execution:

```
>>> shuffle([1, 2, 3, 4, 5, 6, 7, 8], 1)
[1, 5, 2, 6, 3, 7, 4, 8]

>>> shuffle([1, 2, 3, 4, 5, 6, 7, 8], 2)
[1, 3, 5, 7, 2, 4, 6, 8]

>>> shuffle([1, 2, 3, 4, 5, 6, 7, 8], 3)
[1, 2, 3, 4, 5, 6, 7, 8]

>>> shuffle([1, 2, 3, 4, 5, 6, 7], 1)
[1, 5, 2, 6, 3, 7, 4]

>>> shuffle([1, 2, 3, 4, 5, 6, 7], 2)
[1, 3, 5, 7, 2, 4, 6]
```

B. After writing the `shuffle` function, you realised that something is wrong. The list goes back to the original configuration after some number of operations. Write a function `backto_original` that takes in a list and returns the number of shuffle operation $n \geq 1$, which will cause the list to return to its original configuration. [5 marks]

Sample execution:

```
>>> back_to_original([1, 2, 3, 4, 5, 6, 7, 8])
3

>>> back_to_original([1, 2, 3, 4, 5])
4

>>> back_to_original([1, 1, 1, 1])
1
```

Question 2 : Following POTUS on Twitter [10 marks]

Donald Trump, the current president of the United States of America is famous for using social media to express his strong opinions on anything from politics to sports. The prolific tweeter and commander in chief is the 13th most followed person on Twitter. In this question, you will be using your recently acquired skills in data processing using python to analyze his tweets.

The data for each of his tweets is provided in a comma-separated file `donald-tweets.csv`, where each line contains the following information:

- Date (yy-mm-dd)
- Time (24 hour format)
- Tweet text
- Hashtags (separated with ";")
- Likes
- Retweets

Note that you cannot assume that your code will only be run on the `donald-tweets.csv` file provided. Your code will also be tested on private data sets with the same format.

A. [Warm up] Write a functions `get_dates_for_hashtag` that takes two arguments, the filename and a hashtag, and returns a list of unique dates on which a certain Hashtag was used. If the hashtag passed to your function never appears in the tweets, the function should return an empty list. [3 marks]

Sample execution:

```
>>> get_dates_for_hashtag("donald-tweets.csv", "ObamacareFail")
['16-10-31', '16-10-29', '16-10-25']

>>> get_dates_for_hashtag("donald-tweets.csv", "ElectionDay")
['16-11-08', '16-08-09']

>>> get_dates_for_hashtag("donald-tweets.csv", "China")
[] #return empty list
```

B. [Most active hour] Donald Trump is busy man. But where does he find the time to write the number of tweets he does? For this question, you must write a function `activehour` that takes three arguments, the filename, a `startdate` and an `enddate`. The function must return a list of the hours of the day (0-23) where he made the highest number of tweets between `startdate` and `end_date` (both inclusive). For ease of notation, if most of the tweets happened between 1PM and 2PM, the function should return the integer 13 (signifying the time interval between 13:00:00 and 13:59:59 hours). If there are no tweets in the specified time period, you should return an empty list. You can assume that `startdate` will not be later than `enddate`.[4 marks]

You can use the following library function to convert a string into a datetime object:

```
>>> from datetime import datetime
>>> d = datetime.strptime("16-11-11 15:26:37", "%y-%m-%d %X")
```

`datetime` objects can be compared to each other using $<$, $==$ and $>$. You can obtain the hour in a `datetime` by accessing the `hour` property, i.e. `<datetime>.hour`.

Sample execution:

```
>>> active_hour("donald-tweets.csv", "16-11-06", "16-11-08")
[21]

>>> active_hour("donald-tweets.csv", "16-08-23", "16-11-06")
[1]

>>> active_hour("donald-tweets.csv", "16-11-06", "16-11-06")
[0, 23] #Hour 0 and Hour 23 both had equally high number of tweets
```

C. [Top *k* tweets] Write a function `top_k` that given *k*, a `startdate` and `enddate`, retrieves the "Tweet text" for the *k* most popular tweets within this time period (ranked in descending order by "Likes", and `startdate` and `enddates` are inclusive). If the number of tweets in the given time period is smaller than *k*, return all the tweets. The list of "Tweet text"s returned should be in the descending order of the likes they received. If there are no tweets in the given time period, return an empty list. [3 marks]

Sample execution:

```
>>> top_k("donald-tweets.csv", 1, "16-11-08", "16-11-08")
['TODAY WE MAKE AMERICA GREAT AGAIN!']


>>> top_k("donald-tweets.csv", 3, "15-10-13", "16-10-11")
['The media is spending more time doing a forensic analysis of Melanias speec
than the FBI spent on Hillarys emails.', 'Such a great honor to be the Republi
Nominee for President of the United States. I will work hard and
never let you down! AMERICA FIRST!', 'Here is my statement.
https://t.co/WAZiGoQqMQ']

>>>top_k("donald-tweets.csv", 1, "16-11-18", "16-11-20")
[]
```

Question 3 : Six Degrees of Separation in College [10 marks]

Once you come to university, it would be helpful to form study groups and get to know people. Most of the friends you make will likely be people who will be taking the same courses. In this problem, you will try to model the problem of getting to know people through friends taking the same classes.

In this problem, you will implement a `Student` class. This class represents a student in your School. The constructor takes in a `name` of the student (string). We assume in this question that 2 students who take a common course are friends. You are to decide on the representation of the object after reading carefully the following 7 methods to be supported and the sample execution:

1. `add_courses(*courses)` is to add the given `courses` (one or more) to the student. The courses will be specified as strings.

2. `drop_courses(*courses)` is to drop the given `courses` (one or more) taken by the student. You can safely ignore any `courses` in the list if the student is not already taking it.

3. `get_courses()` will return a list of the courses that the student is currently taking.

4. `is_coursemate(student)` will return `True` if the specified student is taking a same course; otherwise, return `False`.

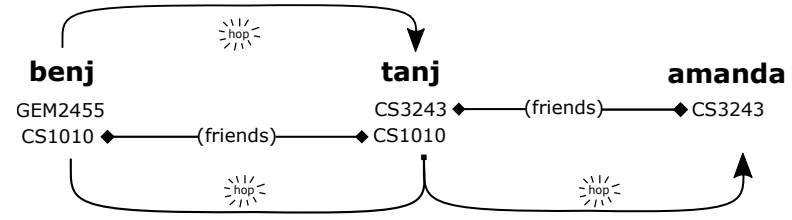
5. `common_friends(student)` will return a list of the names of the students who are common friends between a student and the specified student. Common friends are students who take at least one course in common with each of them. Do note that Students <u>cannot be their own friends</u>.

   For example, if only two students `benj` and `tanj` take CS1010X as their only course, they will have no common friends.

6. `common_courses(student)` will return a list of the common courses that the student is taking together with the specified student. If there are none, an empty list is returned.

7. `six_degrees(student)` will return the minimum "distance" between a student and the specified student. How exactly this distance is calculated should be clear from the graphic below. If a chain of common friends cannot be found, return `None`.

You are free to define your own helper functions and classes as you see fit.




```
benj.six_degrees(tanj) = 1
benj.six_degrees(amanda) = 2
```

Class Variables. In class, we have only discussed instance variables for OOP. Basically, when you access a variable with `self.<something>`, that variable exists for a given instance of a class. It is possible to define variables for a class like:

```
class A:
    my_list = []

    def __init__(self):
        ...
```

Then, we can access the class variable with `A.my_list`. Class variables might be helpful for this question.

Sample execution:

```
>>> benj = Student("Ben Junior")
>>> benj.add_courses("CS1010", "CS1014", "CS2010", "CS2060", "CS2016", "GEM24
>>> tanj = Student("Tan Junior")
>>> tanj.add_courses("CS1010", "CS1014", "CS2010", "CS2060", "CS3243")
>>> amanda = Student("Amanda See")
>>> amanda.add_courses("GEM1010", "CS1014",  "CS1231", "CS2017", "GEM2455")
>>> ad = Student("Ad Lee")
>>> ad.add_courses("CS1010", "CS1000", "CS2010", "CS2040", "CS1207")
>>> ayush = Student("Ayush")
>>> ayush.add_courses("MA1016", "MA1014", "MA2050", "MA2016")

>>> benj.is_coursemate(tanj)
True
>>> benj.is_coursemate(amanda)
True
>>> benj.is_coursemate(ad)
True
>>> benj.is_coursemate(ayush)
False

>>> benj.common_courses(tanj)
['CS1010', 'CS1014', 'CS2010', 'CS2060']
>>> benj.common_courses(amanda) # Found interesting girl in class!
['CS1014', 'GEM2455']
>>> benj.common_courses(ad)
['CS1010', 'CS2010']
>>> print(benj.common_courses(ayush)
[]

>>> amanda.drop_courses("CS1014", "GEM2455")
>>> amanda.add_courses("CS3243")
>>> benj.is_coursemate(amanda) # She disappears from our class :'(
False
```

```
>>> benj.common_courses(amanda)
[]
>>> amanda.get_courses() # What is she taking now?
['GEM1010', 'CS1231', 'CS2017', 'CS3243'])

>>> benj.six_degrees(amanda) # How can we get to know her?
2
>>> benj.common_friends(amanda)
['Tan Junior']
>>> amanda.common_courses(tanj)
['CS3243']

>>> tanj.drop_courses("CS3243")
>>> benj.six_degrees(amanda)  # No more friends
None
>>> benj.common_friends(amanda)
[]

>>> amanda.add_courses("MA2050")
>>> ayush.add_courses("CS1000")
>>> benj.six_degrees(amanda)  # How do we find her now?
3
```