CS1010X — Programming Methodology
National University of Singapore

Re-Practical Examination

16 June 2018

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 : Counting Triangles [10 marks]

You just got a new job as a Research Assistant at a data analytics firm. The firm is working on a new simulation for a classified agency. For your first job, you were assigned to assist in writing some simple libraries to support the simulator team.

A. [**Warm-Up**] Write a function `pythagoras` that takes in 3 points, which are pairs and return `True` if the 3 points make up a right angle triangle. [5 marks]

Note: Given 2 points (x_1, y_1) and (x_2, y_2), the distance between them is given by:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

in case you have forgotten your Math. The square root function in Python is `sqrt`. Also a triangle is a right angle triangle if the square of one side the the sum of squares of the 2 remaining sides.

Sample execution:

```
>>> pythagoras((0,0),(1,2),(1,1))
False
>>> pythagoras((0,0),(1,0),(0,1))
True
>>> pythagoras((0,0),(2,0),(0,2))
True
```

B. Write a function `count_triangles` that takes in a list of points and returns the number of right angled triangles that can be formed with the points in the list. You may assume that all the points are distinct. A correct implementation of `pythagoras` has been defined for you for this task. [5 marks]

Sample execution:

```
>>> count_triangles(((0,0),(1,0),(0,1)))
1
>>> count_triangles(((0,0),(1,1),(2,1)))
0
>>> count_triangles(((0,0),(1,0),(0,1),(-1,-2)))
# [[(0, 0), (1, 0), (0, 1)], [(1, 0), (0, 1), (-1, -2)]]
2
>>> count_triangles(((0,0),(1,0),(0,1),(1,1)))
4
```

Question 2 : Finding Happiness [10 marks]

Very little is needed to make a happy life; it is all within yourself, in your way of thinking.
— Marcus Aurelius Antoninus

For a very long time, wise men have been undertaking the task of defining happiness or finding ways to achieve it, yet, there is no real consensus over what makes people happy. Don't worry though, we are not asking you to find an answer to that! :) We would like, however, to analyze the data from the World Happiness report. The report shows how countries from around the world have scored for particular metrics during different years. The metrics that we are intrerested in are: GDP, Social, HLE (Healhy Life Expectancy), Freedom and Generosity. The data file that is provided for this question contains on each line the name of the country, the year, followed by the scores of that country for the mentioned metrics.

To read the data, you are provided with the `read_csv(filename)` function that takes in the name of the filename and produces a tuple of lists, each list containing data of a particular country for a particular year.

Sample execution:

```
>>> read_csv("happiness.csv")[1]
['Afghanistan', '2008', '7.16', '0.45', '49.20', '0.71', '0.18']
```

A. [**Organizing the Data**] Before we can analyze the data, we have to store it an appropriate data structure. Since we are going to work with data from multiple years, we want to be able to access the data from a particular year easily. We would also like to be able to access a particular country's data easily. To do so, we are going to use a dictionary of dictionaries of dictionaries! The first dictionary is indexed by year:

```
>>> data = parse_data("happiness.csv")
>>> data[2015]
{'Afghanistan': {'GDP': 7.46, 'Social': 0.52, 'HLE': 51.69, ...},
'Albania': {'GDP': 9.30, 'Social': 0.63, 'HLE': 68.69, ...}, ...}
```

The value is a dictionary indexed by countries, so we can extract a country's data like this:

```
>>> data[2015]['Indonesia']
{'GDP': 9.24, 'Social': 0.80, 'HLE': 60.28, 'Freedom': 0.77, 'Generosity': 0.45}
```

Lastly, we can also get a specific metric doing this:

```
>>> data[2013]['France']['GDP']
10.52
```

Note that the numbers were truncated to fit better in page.

Write a function `parse_data`, that takes in the name of the file containing all the data and creates the associated data structure such that we can answer queries like the ones described above. Note that some metric values may be missing occasionally – in that case, we will skip the metric. Trying to access a non-existent matric will result in a `KeyError`.

[4 marks]

Sample execution:

```
>>> report = parse_data("happiness.csv")
>>> report[2017]['India']['Social']
0.6067674756
>>> report[2013]['Singapore']['GDP']
11.2714776993
>>> report[2014]['Brazil']['Generosity']
-0.1286974549
>>> report[2008]['Austria']['Freedom']
0.8790692687
>>> report[2011]['Chile']['HLE']
68.6461639404
>>> report[2007]['Canada']['Social']
KeyError: 'Social'
```

B. [**Data Filtering**] We would like to use the Happiness Report to retrieve countries that fulfill certain conditions. Write a function `get_countries` that takes in the filename of the happiness report and a variable number of conditions represented as functions. The function returns a list containing the names of the countries that satisfy all conditions. If no conditions are specified, then the list should contain the name of all countries present in the report, and if no country satisfies all conditions, then the function should return the empty list. The condition is a predicate that takes in a year and a country and returns `True` or `False` based on whether the country satisfies the predicate for that year, e.g.:

```
def GDP_greater_than_10_in_2012(year, country):
  return year == 2012 and country['GDP'] > 10
```

[3 marks]

Sample execution:

```
>>> get_countries("happiness.csv", lambda y, c: y >= 2012 and c['GDP'] > 11,
lambda y, c: y == 2011 and c['Social'] > 0.9)
['Ireland', 'Singapore', 'Luxembourg']
>>> get_countries("happiness.csv", lambda y, c: y == 2012 and c['GDP'] > 11,
lambda y, c: y == 2011 and c['Social'] > 0.9)
['Luxembourg']
>>> get_countries("happiness.csv", lambda y, c: y > 2020 and c['GDP'] > 0)
[]
```

Note that there is no data for Singapore in 2012, and Ireland's GDP was less than 11 in 2012, which is why the second call only has Luxembourg as an answer.

It might also be instructive that each condition is an OR condition while the returned list of countries. For example, `lambda y, c: y >= 2012 and c['GDP'] > 11` will be true for a country if in *any* year after 2012, its GDP metric exceeds 11.

You can use the `get_all_countries` function which takes in the data representation (i.e., the map you created for the previous task), and returns a list containing the names of all the countries that appear in the data. Note the Coursemology contains the correct definition of `parse_data` from Part (A), so even if your answer to Part (A) is incorrect, you will have access to a correct version of `parse_data`.

C. [**Prediction!**] Lastly, we would like to use the data we have to predict how countries will do in the future using linear regression.

Linear Regression. Linear Regression is a way to model the relationship between two sets of variables, in our case, the year and the value of a particular metric. The result is a linear regression equation that can be used to make predictions about the data. The equation has the form `Y = a + bX`. The constants `a` and `b` can be computed according to the following equations:

$$a = \frac{\sum y \sum x^2 - \sum x \sum xy}{n\sum x^2 - (\sum x)^2}$$

$$b = \frac{n\sum xy - \sum x \sum y}{n\sum x^2 - (\sum x)^2}$$

where `x` and `y` refer to the values in the data set. In our case, `x` refers to the years, and `y` refers to the associated metric values.

To predict the GDP of Japan in 2050, we look at all the information we have about Japan so far, i.e., in 2005, Japan's GDP was 10.48, in 2007 10.51 and so on, and with this data we create a list of tuples [(2005, 10.48), (2007, 10.51),...]. This data is then used to compute the Linear Regression coefficients `a` and `b`. The predicted value for Japan's GDP in 2050 will be `a + b * 2050`.

Write a function `predict`, that takes in the filename, the name of the country, the name of the metric and the year the prediction should be made for, and returns the value of the prediction. If there is no data to compute the prediction, the function should return `None` [3 marks]
Sample execution:

```
>>> predict("happiness.csv", "Japan", "GDP", 2050)
10.786426327185783
>>> predict("happiness.csv", "Italy", "Social", 2030)
0.9245298996310285
>>> predict("happiness.csv", "Nigeria", "Generosity", 2019)
0.018709013077190306
```

Question 3 : The Summit [10 marks]

A highly anticipated summit between United States President Donald Trump and North Korean leader Kim Jong Un ended on Tuesday afternoon (Jun 12).

Both leaders emerged from a bilateral meeting and signed a joint statement. Among the key points in the document was North Korea's commitment to "work towards the complete denuclearisation of the Korean Peninsula."

President Trump said during the signing ceremony that they have "developed a special bond".

Earlier in the day, in a historic moment, the two leaders walked across the aisles towards each other at the Capella Hotel in Singapore's Sentosa island and shook hands as the world watched.

In a news conference later that day, Mr Trump said the two countries were "ready to write a new chapter" and called the day's talks "honest, direct and productive".

– Source: CNN

Figure 1: LTA Announcement.

While the Summit was a raving success, the event posed a big headache for the hotels in the Orchard area. You happened to be working in one of the hotels. Your boss learnt about your new-found programming abilities picked up from CS1010X and you have been tasked to write a program that will provide the tourists on the current traffic conditions as LTA provides updates on the road closures.

In this problem, you will implement a `TripPlanner` class. This class will store local road data and allow a tourist to query if it is possible to travel between 2 places. The constructor takes in the name of a data file that contains the road and place data. The class supports the following 9 methods:

1. `read` is implemented for you. This method reads the road and place data from a specified data file.

2. `add_junction(road1,road2,...,roadn)` accepts a bunch of roads (strings) that all meet at a common junction.

3. `add_place(place,road)` specifies that a place is situated on the specified road.

4. `is_connected(road1,road2)` returns `True` if `road1` is *directly* connected to `road2`, or `False` otherwise. `"No such road"` is returned if either of the specified roads does not exist.

5. `get_road(place)` returns the road that a place is situated on. `"No such place"` is returned if the specified place does not exist.

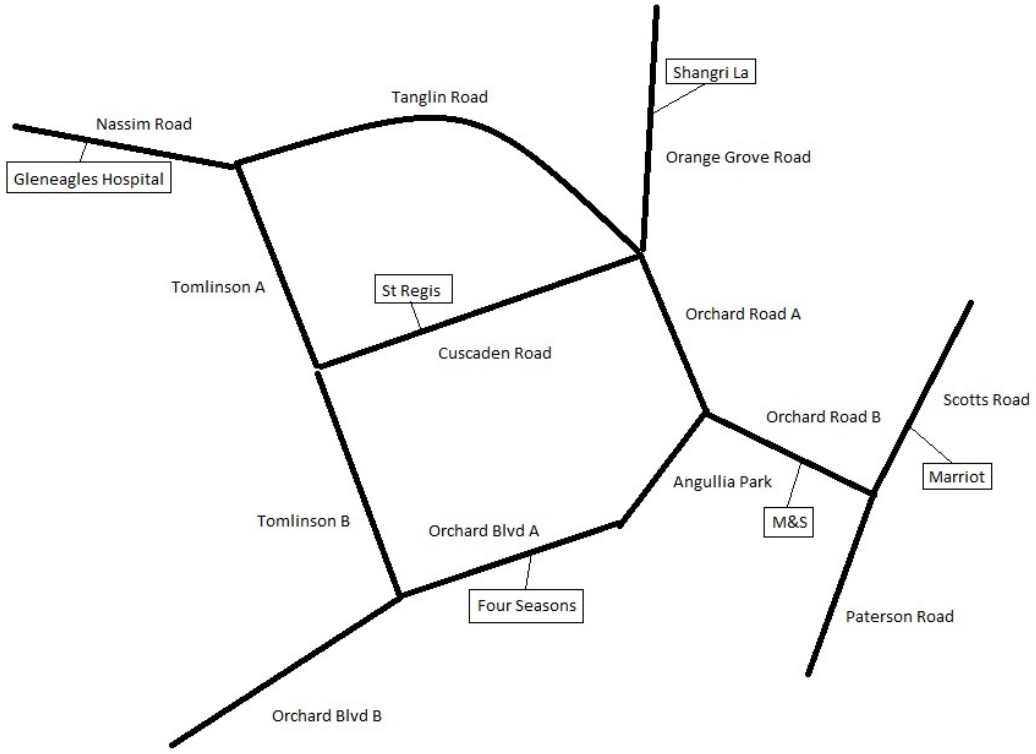
6. `block_road(road)` specifies that LTA has blocked the specified road. Returns `True` if the status of the road is updated correct; returns `False` if the road is already blocked. `"No such road"` is returned if the specified road does not exist.

7. `unblock_road(road)` specifies that LTA has announced that the specified road is no longer blocked. Returns `True` if the status of the road is updated correct; returns `False` if the road is not already blocked. `"No such road"` is returned if the specified road does not exist.

8. `is_blocked(road)` returns `True` if the specified road is blocked, or `False` otherwise.

9. `can_reach(place1, place2)` returns `True` if some set of roads leading from `place1` to `place2` are not blocked, or `False` otherwise. If either of `place1` or `place2` in invalid, return `"No such place"`.

Note: You can define your own helper methods if required. To make this more concrete, consider the simplified view of Orchard Road:



Sample execution:

```
>>> t = TripPlanner("orchard.csv")
>>> t.can_reach('Gleneagles Hospital', 'St Regis')
True
>>> t.can_reach('Four Seasons', 'St Regis')
True
>>> t.can_reach('M&S', 'St Regis')
True
>>> t.can_reach('Marriot', 'St Regis')
True
>>> t.can_reach('Shangri La', 'St Regis')
True

>>> t.is_connected('Tanglin Road', 'Orchard Road A')
True
>>> t.is_connected('Tanglin Road', 'Paterson Road')
False
>>> t.get_road('Shangri La')
"Orange Grove Road"

>>> t.block_road('Cuscaden Road')
True
>>> t.block_road('Cuscaden Road')
False
```

```
>>> t.can_reach('Gleneagles Hospital', 'St Regis')
False
>>> t.can_reach('Four Seasons', 'St Regis')
False
>>> t.can_reach('M&S', 'St Regis')
False
>>> t.can_reach('Marriot', 'St Regis')
False
>>> t.can_reach('Shangri La', 'St Regis')
False

>>> t.unblock_road('Cuscaden Road')
True
>>> t.unblock_road('Cuscaden Road')
False
>>> t.can_reach('Shangri La', 'St Regis')
True
>>> t.block_road('Orange Grove Road')
True
>>> t.can_reach('Shangri La', 'St Regis')
False
>>> t.can_reach('Gleneagles Hospital','Marriot')
True
```