CS1010S — Programming Methodology
National University of Singapore

Practical Examination

16 Nov 2019

Time allowed: 2 hours

Instructions (please read carefully):

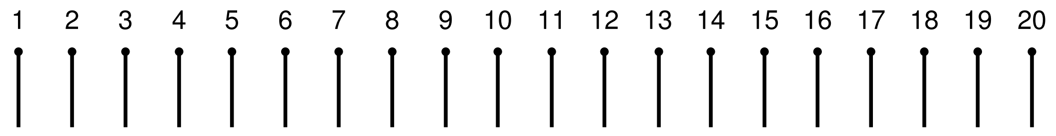
1. This is **an open-book exam**. You are allowed to bring in any course or reference materials in printed form. No electronic media or storage devices are allowed.
2. This practical exam consists of **three** questions. The time allowed for solving this test is **2 hours**.
3. The maximum score of this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

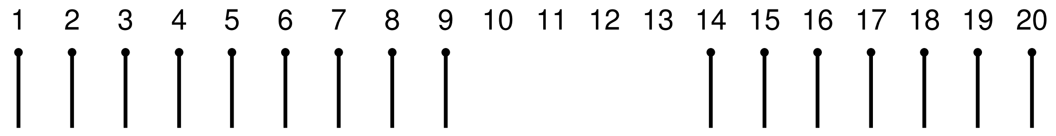
Question 1 : A Game of Matchsticks [10 marks]

A game of matchsticks is played by first arranging matchsticks in a sequence. Players then take turns to remove any number of **consecutive** matchsticks from the game. When a matchstick is removed, it creates a gap, splitting the matchsticks into separate piles.

For example, suppose the game starts with 20 matchsticks, labelled one to twenty as shown:

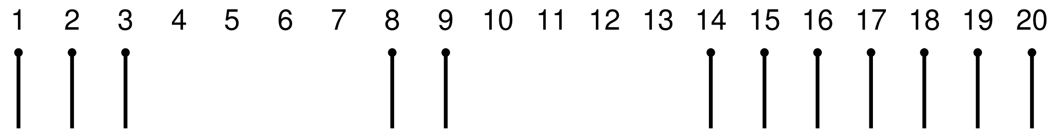


If a player removes matchsticks 10 through 13, he splits the matchsticks into two piles:



Thus, the next player can only remove matchsticks from either of the piles, but not both. I.e., he can remove any consecutive matchsticks numbered 1–9, or 14–20, but not across both piles.

If the next player removes matchsticks 4 through 7, the game is further split into 3 piles:



The game state is modelled in Python as a list of piles, with each pile being a list indicating the index of the starting and ending matchstick. Note that the numbering of the matchsticks do not change thus the **order of the piles in the list does not matter**.

For example, the above three game states are represented as:

```
[[1, 20]]
[[1, 9], [14, 20]]
[[1, 3], [8, 9], [14,20]]
```

A. A move is valid only when matchsticks are removed from one pile. Take for example that last game state mentioned above. Removing matchsticks 2 through 8 is **invalid** as it removes from two piles. Removing matchsticks 4 through 7 is also **invalid** because no matchsticks were removed.

Removing 2 through 6 is **valid** because although index 4 through 6 are empty, matchsticks 2 and 3 are removed and they are both from the first pile.

Implement a function `valid_move(game, start, end)` that takes in a game state in the above-mentioned representation, and the starting and ending indexes of matchstick to remove. You may assume that `end` will not be smaller than `start`. [5 marks]

Sample execution:

```
>>> game = [[1, 3], [8, 9], [14,20]]
>>> valid_move(game, 2, 8)
```

```
False

>>> valid_move(game, 4, 7)
False

>>> valid_move(game, 2, 6)
True
```

B. Implement the function `make_move(game, start, end)` that takes in a game state, and the starting and ending indexes of matchstick to remove. The function **updates and returns** the game state with the specified matchsticks removed. You may assume that the move will be valid. [5 marks]

Sample execution:

```
>>> game = [[1, 20]]
>>> make_move(game, 10, 13)
[[1, 9], [14, 20]]

>>> make_move(game, 4, 7)
[[1, 3], [8, 9], [14, 20]]

>>> make_move(game, 15, 17)
[[1, 3], [8, 9], [14, 14], [18, 20]]
```

Question 2 : Airbnb listings [10 marks]

Important note: You are provided with a data file `listings.csv` for this question for testing, but your code should work correctly for *any* data file with the same format and it *will* be tested with other data files.

You are given a data file of Airbnb room listings in Singapore for a particular month. Each row in the data file contains details for a unique room listing. The data contains the following columns in specific order:

ID	Name	Region	Neighbourhood	Latitude	Longitude	Price	Review Count

A. When looking for locations to stay at, it is always useful to be able to stay nearby your favourite locations.

Implement the function `k_nearest_listings` which takes as input the name of the data file, and the float values for the latitude and longitude respectively, and the integer value limiting the number of results. It returns a list containing the IDs of the k-closest listings to the given latitude and longitude, based on the data. The listings should be ordered from the closest-to-farthest.

All distances measured using latitude and longitude should be rounded-off to 5 decimal places. If there is a tie in the k-th closest listing, listings which tie with the k-th closest listing should also be included.

Note that a function `get_distance` is provided to help calculate the correct distance between two coordinates. The function takes in the four inputs—the latitude and longitude of the first coordinate and the latitude and longitude of the second coordinate—and returns the distance rounded-off to 5 decimal places. [5 marks]

Sample execution:

```
>>> k_nearest_listings("listings.csv", 1.38837, 103.67087, 1)
[13756029]

>>> k_nearest_listings("listings.csv", 1.42724, 103.84648, 3)
[9980935, 9105592, 15189554]

>>> k_nearest_listings("listings.csv", 1.29114, 103.87363, 5)
[35199403, 1178486, 17540932, 35219943, 8623041]
```

B. The price of each listing is usually affected strongly by location. Given a particular region, you are to calculate the average price of listings for each neighbourhood that exists in that region within the data. To make the price estimate more reliable, you should only use listings which have at least 1 review.

Implement the function `neighbourhood_price_per_region` which takes as input the name of the data file, and the string value for the region. It returns a dictionary where the keys are the neighbourhoods in that region and the values are the average price of listings for that neighbourhood that **have at least 1 review**.

Note that the average price should be rounded off to the nearest cent (2 decimal places). [5 marks]

Sample execution:

```
>>> neighbourhood_price_per_region("listings.csv", "East Region")
{'Tampines': 97.02, 'Bedok': 124.14, 'Pasir Ris': 86.35}

>>> neighbourhood_price_per_region("listings.csv", "North-East
   Region")
{'Serangoon': 88.4, 'Hougang': 89.16, 'Punggol': 75.0, 'Ang Mo Kio':
   80.96, 'Sengkang': 57.02}

>>> neighbourhood_price_per_region("listings.csv", "North Region")
{'Woodlands': 90.02, 'Sembawang': 89.58, 'Central Water Catchment':
   110.36, 'Yishun': 96.68, 'Mandai': 56.67, 'Sungei Kadut': 49.0}
```

Question 3 : Matryoshka Dolls [10 marks]

Matryoshka dolls are the set of wooden dolls of decreasing size that can be placed one inside another.

Matryoshka dolls. Source: Wikipedia

The class `Doll` models an individual doll. `Doll` is created with one input, its name which is a `str`. To simplify things, we ignore the physical size of the dolls and allow any doll to encase any other doll. While each doll can only **directly** contain one other doll, it can however, by nesting dolls, indirectly contain more.

When a doll (mother) encases another doll (daughter), the mother doll is being added to the same *series* with the daughter doll. Repeatedly encasing the mother doll in other dolls will also add them into the same series.

When a mother doll releases its daughter doll, it continues to remain in the same series. Only when the mother doll encases a doll from another series, the mother gets removed from her old daughter series and joins her new daughter's series.

The above behaviour of `Doll` is supported by the following methods:

- `encase(other)` takes as input one `other` doll, and attempts to place `other` inside this doll. For this to be successful, this doll must currently not contain any doll, and both dolls must not be encased by another. The method returns a string based on the following conditions:
  - `'<doll name> already contains <daughter doll name>'`, if this doll currently contains a doll (its daughter doll).
  - `'<doll name> is currently encased in <mother doll name>'`, if this doll is currently encased inside a doll (its mother doll).
  - `'<other name> is currently encased in <mother doll name>'`, if the other doll is currently inside a doll (its mother doll).
  - Otherwise, `'<doll name> encases <other name>'` is returned and the other doll is placed inside this doll.
- `release()` takes no input and attempts to release the doll which is encased in this doll. It returns a string based on the following conditions:

  - `'<doll name> is currently encased in <mother doll name>'`, if this doll is still encased within its mother doll.
  - `'<doll name> does not contain any dolls'`, if this doll currently contains no doll.
  - Otherwise, `'<doll name> releases <daughter name>'` is returned and the daughter doll contained in this doll is released.

- `get_name()` returns the name of the doll.
- `get_mother()` returns the direct doll that **contains** this doll. If this doll has no mother, return `None`.
- `get_daughter()` returns the direct doll that is **contained** by this doll. If this doll has no daughter, return `None`.
- `deeply_contains(other)` returns `True` if the specified doll `other` is deeply encased within this doll. Returns `False`, otherwise.
- `num_encased()` returns the number of dolls in the same series as this doll, which are currently encased inside another doll.
- `series()` returns a tuple containing the names of all the dolls in the same series as this doll. The order of the names do not matter.

Provide an implementation for the class `Doll`.

Sample Execution:

```
>>> alice.get_name()
'Alice'

>>> alice.get_mother()
None
>>> alice.get_daughter()
None
>>> alice.series()
('Alice',)

>>> betty.encase(alice)
'Betty encases Alice'

>>> betty.get_daughter() is alice
True
>>> alice.get_mother() is betty
True

>>> betty.encase(clara)
'Betty already contains Alice'

>>> alice.encase(clara)
```

```
'Alice is currently encased in Betty'

>>> clara.encase(alice)
'Alice is currently encased in Betty'

>>> clara.encase(betty)
'Clara encases Betty'

>>> alice.series()
('Alice', 'Betty', 'Clara')
>>> betty.series()
('Alice', 'Betty', 'Clara')
>>> clara.series()
('Alice', 'Betty', 'Clara')

>>> alice.num_encased()
2
>>> betty.num_encased()
2
>>> clara.num_encased()
2

>>> clara.deeply_contains(alice)
True
>>> alice.deeply_contains(clara)
False

>>> alice.release()
'Alice is currently encased in Betty'
>>> betty.release()
'Betty is currently encased in Clara'
>>> clara.release()
'Clara releases Betty'
>>> betty.release()
'Betty releases Alice'
>>> alice.release()
'Alice does not contain any dolls'

>>> alice.series() == betty.series() == clara.series()
True

>>> alice.encase(clara)
'Alice encases Clara'

>>> alice.num_encased()
1
>>> betty.num_encased()
1
```

```
>>> clara.num_encased()
1

>>> betty.encase(doris)
'Betty encases Doris'

>>> alice.series()
('Alice', 'Clara')
>>> betty.series()
('Doris', 'Betty')
>>> clara.series()
('Alice', 'Clara')
>>> doris.series()
('Doris', 'Betty')
```

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 —