CS1010X — Programming Methodology
National University of Singapore

Re-Practical Examination

10 June 2017

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 : Periodic Strings [10 marks]

A. [**Warm Up**] Write a function `factors` that returns a list of the factors for a positive integer $n > 1$. The factors should be ordered in ascending order. [5 marks]

Sample execution:

```
>>> factors(2)
[1,2]

>>> factors(13)
[1,13]

>>> factors(18)
[1, 2, 3, 6, 9, 18]
```

B. Write a function `repetitions` that given a string, returns the maximum number of repeated substrings within the string. How this works should be clear from the following examples. [5 marks]

Sample execution:

```
>>> repetitions("aaaaa")
5

>>> repetitions("abababab")
3

>>> repetitions("abababc")
1

>>> repetitions("abadbabcabadbabc")
2
```

Question 2 : IMDB Data Mining [10 marks]

In this problem, we will work with movie data. The data is provided in the accompanying files `IMDBsmall.csv` and `IMDB.csv`, where each entry contains the following information:

1. Director Name
2. Genre: It contains a list of genres separated by |
3. Actor Name
4. Movie Name
5. IMDB Rating
6. Release Year

A. [**Data Cleaning**] Data is often rife with problems and discrepancies. Before an analyst can dive into the data, the first task is to "clean" or sanitize the data. Basic idea is to delete "bad" data. Write the function `cleandata` that sanitizes the data by removing entries that do not conform to the following specifications:

- The data should be complete. None of the fields, i.e. (i) director, (ii) genre, (iii) actor, (iv) movie, (v) rating and (iv) release year should be blank.
- IMDB rates a movie on the scale of 0-10 (inclusive). As a part of data cleaning, we want to filter out all entries for which movie rating is out of these bounds.
- The dataset comprises of movies released between years 1916 and 2016 (both inclusive). The data for the movies which are released outside this timespan is not reliable. These entries should also be removed.

`cleandata` should return a list of **cleaned entries** and the order of the entries should be the same as that in the original dataset file. [3 marks]

Sample execution:

```
>>> answer1 = [['Christopher Nolan', 'Adventure|Drama|Sci-Fi', 'Matthew McCor
              'Interstellar', '8.6', '2014'], ['Alan Taylor', \
              'Action|Adventure|Sci-Fi', 'J.K. Simmons', 'Terminator Genisys'
              '8.6', '2015'], ['Francis Lawrence', 'Adventure|Sci-Fi', \
              'Jennifer Lawrence', 'The Hunger Games: Mockingjay - Part 2', \
              '8.6', '2015'], ['Duncan Jones', 'Action|Adventure|Fantasy', \
              'Dominic Cooper', 'Warcraft', '8.6', '2016'], ['James Bobin', \
              'Adventure|Family|Fantasy', 'Johnny Depp',\
              'Alice Through the Looking Glass', '8.6', '2016']]
>>> clean_data("IMDB_small.csv")==answer1
True
```

```
>>> answer2 = [['A. Raven Cruz', 'Action|Adventure|Comedy|Fantasy|Sci-Fi',\
              'Scott Levy', 'The Helix... Loaded', '4.9', '2005'], ['Aaron Han
              'Drama|Horror|Mystery|Sci-Fi|Thriller', 'Jordi Vilasuso',\
              'Circle', '5.3', '2015']]
>>> clean_data("IMDB.csv")[:2]==answer2
True
```

Note that for the remaining parts of this question, we will work with cleaned data.

B. [**Best movies to watch**] Ben Bitdiddle is bored and wants to watch some movies. To help him out, write a function `best_movies` which takes dataset file name, genre name and k as its arguments and returns a list of the best k movies in the chosen genre (based on movie rating). Ties are broken with the name of the movie in alphabetical order. [3 marks]

Sample execution:

```
>>> best_movies("IMDB_small.csv", "Comedy",5)
[]

>>> best_movies("IMDB_small.csv", "Action",3)
['Terminator Genisys', 'Warcraft']

>>> best_movies("IMDB.csv", "Comedy",5)
['Up', 'Monsters vs. Aliens', 'Evan Almighty', 'Suicide Squad', 'Wild Wild We

>>> best_movies("IMDB.csv", "Action",3)
['Jupiter Ascending', 'The Legend of Tarzan', 'The Dark Knight']
```

Hint: You can use `in` to check that a substring is contained within a string.

C. [**Best Actor**] Ben and his friend Alyssa are having an argument over who is the best actor(ess). They decide to settle this argument using IMDB data. Write a function `best_actor` that takes a dataset file name, a start year and an end year (both inclusive) and returns the best actor for the specified period. The best actor is defined as the actor who has the best average film rating in the given period. In case of ties, return a list of all the actor(ess) who are tied for best. If no movies are found in the specified period, then there's obviously no best actor and `best_actor` should return `None`. [4 marks]

Sample execution:

```
>>> best_actor("IMDB_small.csv", 1916, 1940)
None

>>> best_actor("IMDB_small.csv", 1940, 2016)
['Matthew McConaughey', 'J.K. Simmons', 'Jennifer Lawrence', \
'Dominic Cooper', 'Johnny Depp']

>>> best_actor("IMDB.csv", 1916, 1940)
"Mel Blanc"

>>> best_actor("IMDB.csv", 1941, 1990)
"Ronny Cox"

>>> best_actor("IMDB.csv", 1991, 2016)
"Steve Bastoni"
```

Question 3 : Go Schooling, Go! [10 marks]

Joseph Schooling made history in the 2016 Olympic Games by winning the gold medal in the 100 m butterfly at the 2016 Olympics, attaining Singapore's first-ever Olympic medal in swimming, as well as its first-ever Olympic gold medal in any sport. In this problem, you will relive those moments of glory with him. How cool is that?

An Olympic swimming pool is typically 50 m long and competitions are held in terms of swimming laps up and down the pool. For example, the 100 m butterfly involves 2 laps up and down the pool. We will model each swimmer using 2 parameters, swimming speed s (in m/s) and turning time t (in s). We assume that all the swimmers will start at the same time. The time taken to swim 100 m in a 50 m pool is therefore $2 \times \frac{50}{s} + t$. The time taken to swim 200 m in a 50 m pool will be $4 \times \frac{50}{s} + 3 \times t$, because a swimmer will need to swim 4 laps and turn 3 times.

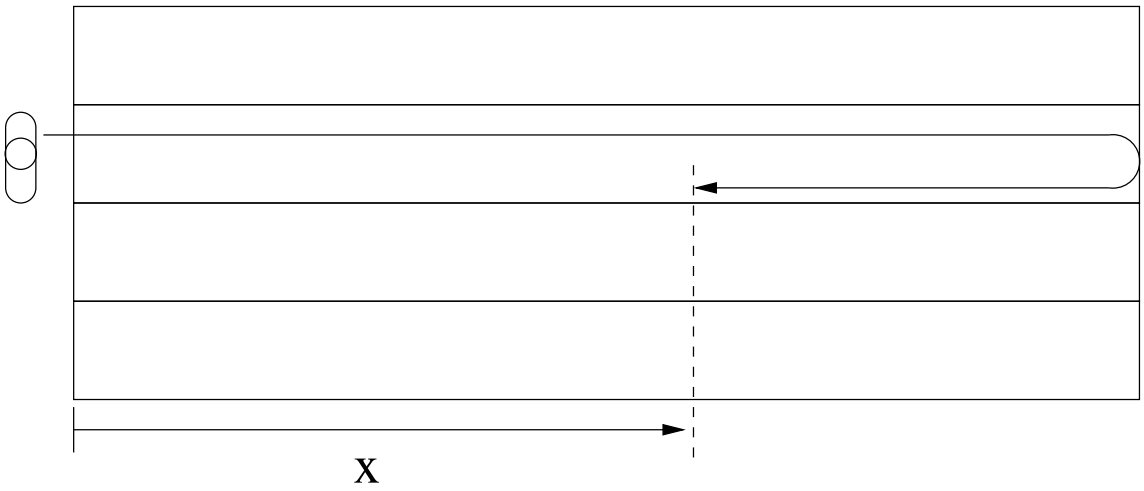

Figure 1: Model of a swimming pool.

In this problem, you will implement a class `SwimmingCompetition`. By default, the pool length is 50 m and there are 8 lanes. Also, we expect that that most times the race will be 100 m. However, in creating a new `SwimmingCompetition`, we can add up to 2 optional arguments to specify first the laps and then the number of lanes. In addition, `SwimmingCompetition` supports the following methods:

- `addswimmer(name, speed, turning)` adds a new swimmer with the specified name, speed and turning time. If there are no more empty lanes, returns `"No more lanes!"`.
- `set_length(length)` allows us to set the length of the pool, if we do not like the default of 50 m.
- `get_position(swimmer, t)` returns the position of the specified swimmer in terms of the distance x from the starting point after t s. For a 50 m race, `get_position` will return 50 after a swimmer finishes a race and henceforth; similarly, for a 100 m race, `get_position` will eventually return 0, since the race finishes when the swimmer returns to the starting point. Returns `"No such swimmer"` if the specified swimmer is not in the pool.

- `finishtime(swimmer)` return the time when the specified swimmer finishes the race. Returns `"No such swimmer"` if the specified swimmer is not in the pool.
- `winner()` will return the winner of the swimming competition. However, `winner(*k)` can take an optional argument to specify a list of the top *k* finishers for the race ordered in ascending order, i.e. fastest one first.

You are welcome to define additional helper methods if you think they are helpful.

Sample execution (newlines added for readability):

```
>>> s1 = SwimmingCompetition()
>>> s1.add_swimmer("Joseph Schooling", 1.96, 0.39)
>>> s1.add_swimmer("Laszlo Cseh",1.955, 0.37)
>>> s1.add_swimmer("Tom Shields",1.953, 0.38)
>>> s1.add_swimmer("Michael Phelps",1.95, 0.42)
>>> s1.add_swimmer("Mehdy Metella",1.945, 0.42)

>>> s1.finish_time("Joseph Schooling")
51.41
>>> s1.finish_time("Laszlo Cseh")
51.52
>>> s1.finish_time("Tom Shields")
51.58
>>> s1.finish_time("Michael Phelps")
51.70
>>> s1.finish_time("Mehdy Metella")
51.83

>>> s1.winner()
"Joseph Schooling"
>>> s1.winner(2)
['Joseph Schooling', 'Laszlo Cseh']
>>> s1.winner(5)
['Joseph Schooling', 'Laszlo Cseh', 'Tom Shields', 'Michael Phelps', 'Mehdy M
>>> s1.winner(6)
['Joseph Schooling', 'Laszlo Cseh', 'Tom Shields', 'Michael Phelps', 'Mehdy M

>>> s1.get_position("Joseph Schooling",0)
0.0
>>> s1.get_position("Joseph Schooling",10)
19.6
>>> s1.get_position("Joseph Schooling",25.5)
49.98
>>> s1.get_position("Joseph Schooling",25.7)
50
```

```
>>> s1.get_position("Joseph Schooling",25.8)
50
>>> s1.get_position("Joseph Schooling",26)
49.80
>>> s1.get_position("Joseph Schooling",27)
47.84
>>> s1.get_position("Joseph Schooling",50)
2.76
>>> s1.get_position("Joseph Schooling",60)
0

>>> s2 = SwimmingCompetition(1,3)
>>> s2.add_swimmer("Joseph Schooling", 1.96, 0.39)
>>> s2.add_swimmer("Laszlo Cseh",1.955, 0.37)
>>> s2.add_swimmer("Tom Shields",1.953, 0.38)
>>> s2.add_swimmer("Michael Phelps",1.95, 0.42)
"No more lanes!"
>>> s2.add_swimmer("Mehdy Metella",1.945, 0.42)
"No more lanes!"

>>> s2.winner()
"Joseph Schooling"
>>> s2.winner(2)
['Joseph Schooling', 'Laszlo Cseh']
>>> s2.winner(4)
['Joseph Schooling', 'Laszlo Cseh', 'Tom Shields']

>>> s2.get_position("Trump",0)
"No such swimmer"
```