CS1010X — Programming Methodology
National University of Singapore

Practical Examination

9 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.
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 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. You are also provided with the template `practical-template.py` to work with. If Coursemology.org fails, you will be required to rename your file `practical-exam-<mat no>.py` where `<mat no>` is your matriculation number and submit that file.
7. 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.
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.

GOOD LUCK!

Question 1 : Digit Product [10 marks]

A. [**Warm-Up**] Write a function `digit_product` that takes in a non-negative integer n and returns the product of all the digits of the integer. [5 marks]

Sample execution:

```
>>> digit_product(1111)
1
>>> digit_product(123)
6
>>> digit_product(123041)
0
```

B. Write a function `max_digit_product(n,k)` that returns the maximum possible product of some k **consecutive** digits ($k > 0$) in the a non-negative integer n. [5 marks]

Sample execution:

```
>>> max_digit_product(11123,1)
3
>>> max_digit_product(112311,2)
6
>>> max_digit_product(1111111,5)
1
>>> max_digit_product(189113451,2)
72
```

Question 2 : Show me the Money! [10 marks]

The Graduate Employment Survey (GES) is conducted by NTU and NUS annually to survey the employment conditions of graduates about six months after their final examinations. The Ministry of Education (MOE) publishes the results of key employment indicators of the survey to provide prospective students with timely and comparable data to assist them in making informed course decisions.

The data was provided as comma-separated file `employment.csv`, where each entry contains the following information:

- year
- university: NUS or NTU
- school
- degree: The degree offered
- variable (two):
  - employment_rate_overall: Overall employment (percentage)
  - basic_monthly_median: Median of basic monthly salary
- value: The corresponding value

Some example entries are provided below:

```
2013,Nanyang Technological University,Business,Accountancy
(3-yr direct Honours Programme),employment_rate_overall,97.1

2015,Nanyang Technological University,Engineering,
Aerospace Engineering,basic_monthly_median,3650
```

A. [**Data Parsing**] It is difficult to work with the data in the form provided in the csv file, so your first job is to read the data to convert it into a form that is easier to work with. In particular, we want to convert the data into a list for each course/year (brackets denote the datatype) as follows:

```
[year(int), university(str), school(str), degree(str),
employment_rate_overall(float/string), basic_monthly_median(float/string)]
```

One example entry would look like:

```
[2013,Nanyang Technological University,Business,Accountancy
(3-yr direct Honours Programme),97.1,2700]
```

Due to some mistakes made during data recording, there is some missing data, i.e. `employment_rate_overal` or `basic_monthly_median` is missing for some of the years/-course. In such instances, `"NA"` should be used in the field with the missing data. All entries should consists of **exactly 6 elements**.

Write a function `parse_data`, that takes in the name of the csv file containing the data and returns a `list` of records, where each record' is a **list** containing the data in the structure described above. Note that this will involve combining multiple lines in the csv file into each record. A `read_csv` function is provided to you in the template file. [4 marks]

Sample execution:

```
>>> parse_data('employment.csv')
[[2013, 'Nanyang Technological University', 'Arts and Social Sciences',\
 'Art, Design & Media', 81.6, 2500.0],
 [2013, 'Nanyang Technological University', 'Arts and Social Sciences', \
'Chinese', 85.1, 2800.0],
 [2013, 'Nanyang Technological University', 'Arts and Social Sciences',\
 'Communication Studies', 89.4, 2930.0]
  ... (too many entries to show all) ...
 [2014, 'National University of Singapore', 'Medicine', 'Bachelor of \
  Medicine and Bachelor of Surgery', 'NA', 4500.0]]
>>> len(parse_data('employment.csv'))
179
>>> count_NA_employment(parse_data('employment.csv'))
1
>>> count_NA_salary(parse_data('employment.csv'))
1
```

Note that the order of your entries might not be the same as the above. We also created two helper functions `count_NA_employment` and `count_NA_salary` to help you check that you have the correct number of `"NA"`'s in your processed data.

B. [**How Employed Are We?**] Write a function `compute_employment_rate` that takes in 5 arguments:

- `filename`: The file name of the data file (i.e. `"'employment.csv"'`)
- `university`: university
- `degree`: course of study
- `start_year`: start year
- `end_year`: end year (inclusive)

and returns the average employment rate for the specified degree between the years specified. If data is not available for a given year, we will ignore that year. If data for the specified course is not available at all, `"'NA"` is returned. Note that even if you cannot do Part(A) above correctly, it is still possible to solve this question independently. You do not have to redefine `parse_data` in Coursemology for this question. There is a correct version that you can just call and use. [3 marks]

Sample execution:

```
>>>compute_employment_rate("employment.csv",'National University of Singapore', \
        "Bachelor of Medicine and Bachelor of Surgery",2000,2018)
100.0
>>>compute_employment_rate("employment.csv",'Nanyang Technological University', \
        "Bachelor of Medicine and Bachelor of Surgery",2000,2018)
"NA"
>>>compute_employment_rate("employment.csv",'National University of Singapore', \
        "Bachelor of Medicine and Bachelor of Surgery",2014,2014)
"NA"
>>>compute_employment_rate("employment.csv",'National University of Singapore', \
        "Bachelor of Computing (Computer Science)",2014,2018)
93.8
```

C. [**Top *K* Promising degrees**] To provides insights to future students into choosing majors, we would like to find some promising degrees based on our data. We say a degree from a specific university is promising between a start year and an end year if the median salary of it is always increasing from the start year to the end year (inclusive) year-on-year. By increasing, we mean `>`, not `>=`.

Write a function `top_k_degree` that takes in 4 arguments:

- `data`: The file name of the data file (`employment.csv`)
- `start_year`: start year
- `end_year`: end year (inclusive)
- `k`: An integer

and returns a list of top k degrees (ranked by the average median salary during the period). If there are degrees satisfying the conditions and having the same average salary as the k-th degree, they should also be included. Degrees with missing data(i.e. `"NA"`) in the specified period should be ignored. [3 marks]

Sample execution:

```
>>> top_k_degree("employment.csv",2014,2015,3)
[['Business and Computing', 'Nanyang Technological University'],\
 ['Bachelor of Engineering (Computer Engineering)', 'National University of Singapore'],
 ['Bachelor of Business Administration (Hons)', 'National University of Singapore']])
>>> top_k_degree("employment.csv",2014,2018,3)
[]
```

Question 3 : Social Network Security [10 marks]

You have a new job at a cannot-be-named social network company. Things have been difficult after the recent Cambridge Analytica scandal. You have been asked to prove yourself by implementing a simple social network that enforces privacy settings correctly.

In this problem, you will implement a `User` class. This class represents a user in your social network. The constructor takes in a name (string) and the class supports the following 7 methods:

1. `request(user)` sends a request to friend another user. Returns `True` if the request is successful; returns `False` if a friend request to the specified user is already pending. Note that if the specified user already has a pending friend request to the requesting user, a friendship bond is immediately created.

2. `accept(user)` accepts the friend request from the specified user and returns `True`. Returns `False` if there is no such pending request waiting to be accepted.

3. `is_friend(user)` returns `True` if the specified user is currently a friend.

4. `unfriend(user)` will terminate the friendship with the specified user and return `True`. Returns `False` if the specified user is not currently a friend.

5. `post(message, [privacy_setting]` posts a message on one's wall with the specified privacy setting (see below). The `privacy_setting` is optional. If not specified, then the setting for the last message posted will be used. By default, privacy for posts is set to `"public"` if `privacy_setting` is never specified.

6. `read_posts(user)` returns a list of the posts from user that are currently visible.

7. `update_privacy(message, privacy_setting)` updates the privacy setting for the specified message with the new privacy setting. If there are multiple of the same message, the latest is changed first. If the latest is already at the right setting, the we will change the setting for the previous one. If no posts matching the input message is found, a warning `"Message not found"` should be returned.

Note: You can define your own helper methods if required.

Your social network supports 4 privacy settings:

1. `"public"` - readable by anyone.

2. `"FOF"` - readable by friends and friends of friends (2 hops away only).

3. `"friends"` - readable by friends.

4. `"private"` - readable by self only.

Any attempts to set privacy to something different from these four settings should return a warning message `"Bad privacy setting"`.

This question is a bit complicated, so you are advised to read through the sample execution below to make sure that you understand the requirements clearly. Ask if you have any questions.

Sample execution:

```
>>> ben = User("Ben")
>>> oana = User("Oana")
>>> chenhao = User("Chenhao")
>>> clement = User("Clement")

>>> ben.is_friend(oana)
False
>>> ben.is_friend(chenhao)
False
>>> ben.accept(oana) # No request pending
False

>>> oana.request(ben)
True
>>> oana.request(chenhao)
True
>>> oana.request(chenhao)
False
>>> oana.is_friend(ben)
False
>>> oana.is_friend(chenhao)
False

>>> ben.accept(oana)
True
>>> chenhao.request(oana)
True
>>> oana.is_friend(ben)
True
>>> oana.is_friend(chenhao)
True

>>> ben.post("CS1010X is fun")
>>> ben.post("No tutorials next week","FOF")
>>> ben.post("Did you remember to order pizza?","friends")
>>> ben.post("Exam grading will be done on Tuesday.")
>>> ben.post("Finals will be very difficult","private")

>>> ben.read_posts(ben)
['CS1010X is fun', 'No tutorials next week',
'Did you remember to order pizza?', 'Exam grading will be done on Tuesday.',
 'Finals will be very difficult']
>>> oana.read_posts(ben)
['CS1010X is fun', 'No tutorials next week',
'Did you remember to order pizza?', 'Exam grading will be done on Tuesday.']
```

```
>>> chenhao.read_posts(ben)
['CS1010X is fun', 'No tutorials next week']
>>> clement.read_posts(ben)
['CS1010X is fun']

>>> ben.post("Finals will be very difficult")  # Repeated private message.
>>> ben.update_privacy("Finals will be very difficult","public")
>>> ben.read_posts(ben)
['CS1010X is fun', 'No tutorials next week', 'Did you remember to order pizza?',
'Exam grading will be done on Tuesday.', 'Finals will be very difficult',
'Finals will be very difficult']
>>> oana.read_posts(ben)
['CS1010X is fun', 'No tutorials next week', 'Did you remember to order pizza?',
'Exam grading will be done on Tuesday.', 'Finals will be very difficult']
>>> chenhao.read_posts(ben)
['CS1010X is fun', 'No tutorials next week', 'Finals will be very difficult']
>>> clement.read_posts(ben)
['CS1010X is fun', 'Finals will be very difficult']

>>> ben.update_privacy("Finals will be very difficult","friends")
>>> ben.read_posts(ben)
['CS1010X is fun', 'No tutorials next week', 'Did you remember to order pizza?',
'Exam grading will be done on Tuesday.', 'Finals will be very difficult',
'Finals will be very difficult']
>>> oana.read_posts(ben)
['CS1010X is fun', 'No tutorials next week', 'Did you remember to order pizza?',
 'Exam grading will be done on Tuesday.', 'Finals will be very difficult']
>>> chenhao.read_posts(ben)
['CS1010X is fun', 'No tutorials next week']
>>> clement.read_posts(ben)
['CS1010X is fun']

>>> ben.update_privacy("Finals will be very difficult","friends")
>>> oana.read_posts(ben)
['CS1010X is fun', 'No tutorials next week', 'Did you remember to order pizza?',
'Exam grading will be done on Tuesday.', 'Finals will be very difficult',
'Finals will be very difficult']
>>> chenhao.read_posts(ben)
['CS1010X is fun', 'No tutorials next week']
>>> clement.read_posts(ben)
['CS1010X is fun']

>>> oana.unfriend(chenhao)
True
>>> oana.unfriend(chenhao)
False
>>> oana.is_friend(chenhao)
```

```
False

>>> oana.read_posts(ben)
['CS1010X is fun', 'No tutorials next week', 'Did you remember to order pizza?',
'Exam grading will be done on Tuesday.', 'Finals will be very difficult',
'Finals will be very difficult']
>>> chenhao.read_posts(ben)
['CS1010X is fun']
>>> clement.read_posts(ben)
['CS1010X is fun']
```