CS1010S — Programming Methodology
National University of Singapore

Practical Examination

14 November 2015

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 **four** questions, out of which you are expected to answer **three**. The time allowed for solving this quiz is **2 hours**.
3. The maximum score of this quiz is **30 marks** and will consist of the best three scores out of the answers for the four questions (if all four questions are attempted). Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. The total score for this quiz is capped at **30 marks**. Marks in excess of your practical exam grade (because you attempted more than 3 questions) will be added to the marks for the midterm exams, which is capped at **100 marks**.
5. 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.
6. While you are also provided with the template `practical-template.py` to work with, your answers should be submitted on Coursemology.org. Note that you can run the test cases on Coursemology.org 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.
7. 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 rename your file `practical-exam-<mat no>.py` where `<mat no>` is your matriculation number and upload the file to the Practical Exam folder in IVLE Workbin. You should only upload one file. If you upload multiple files, we will choose one at random for grading.
8. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology (or uploaded to IVLE, if there are problems) 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.
9. 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 Chars [10 marks]

A. Write a function `count_char` that take in as input, a character and a sentence, and outputs the number of occurrences of the character in the sentence. The function should be case insensitive, i.e., uppercase and lowercase characters should be counted as the same.

Hint: the string functions `lower()` and `upper()` can be used to convert the case of a string. [5 marks]

Examples:

```
>>> sentence = 'The quick brown fox jumps over the lazy dog.'
>>> count_char('e', sentence)
3

>>> count_char('t', sentence)
2

>>> count_char('7', sentence)
0
```

B. Write a function `char_count` that takes as input a non-negative integer n and a sentence s, and returns a list characters that occurs n times in s. The function should be case insensitive, i.e., uppercase and lowercase characters should be counted as the same, and the output should be lowercase letters sorted in ascending order. [5 marks]

Examples:

```
>>> sentence = 'The quick brown fox jumps over the lazy dog.'
>>> char_count(4, sentence)
['o']

>>> char_count(2, sentence)
['h', 'r', 't', 'u']

>>> char_count(5, sentence)
[]
```

Question 2 : Airline Industry [10 marks]

Important note: You are provided with a data file `airline_routes.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.

Your exorbitantly rich friend wants to start his own airline and is looking for your help to conduct a feasibility study. He has provided you with a data files which contains information about the current airline routes.

Each row in the CSV file describes the **airline name** that operate on the route, the **source airport** name, the **city** and the **country** of the source airport, the **destination airport** name, the **city** and the **country** of the destination airport. It also also contains information on the aircraft being used for that particular route, specifically the **aircraft manufacturer** and the **aircraft model/type**.

Note that the routes are directional, i.e., a flight listed from A to B does not mean that a flight from B to A exists. A separate flight from B to A will be listed if it exists. Further note that an airline could use multiple aircrafts to serve the same route, resulting in separate entries of the same route with different aircrafts.

A. Given a source city and country, and a destination city and country, your friend wants to know which airlines currently operate on that direction. He is also interested in the aircraft(s) the airline uses for this particular route.

You are to implement the function `get_route_info` which takes as inputs the source city, source country, destination city, destination country and the name of the CSV file. The output is a dictionary with the **names of the airlines as the keys** and the **values will be the list of aircraft model/type** which the respective airline uses for that route. [5 marks]

Sample execution:

```
>>> get_route_info("Singapore", "Singapore",
                   "Dubai", "United Arab Emirates",
                   "airline_routes.csv")
{'Ethiopian Airlines': ['777-300ER', 'A330-300'],
 'Qantas': ['777-300ER', 'A380-800'],
 'Singapore Airlines': ['777-300ER', 'A330-300'],
 'Egyptair': ['A330-300'],
 'Emirates': ['777-300ER', 'A380-800']}

>>> get_route_info("Singapore", "Singapore",
                   "Melbourne", "Australia", "airline_routes.csv")
{'Ethiopian Airlines': ['A380-800'],
 'Virgin Australia': ['A330-300', '777-300ER', 'A380-800', '777-300'],
 'Singapore Airlines': ['A330-300', '777-300ER', 'A380-800', '777-300'],
 'Qantas': ['A330-300'],
 'China Eastern Airlines': ['A330-300'],
 'Emirates': ['777-300ER'],
 'Jetstar Airways': ['A330-200']}
```

B. Your friend is also interested to know the aircrafts an airline is using. In particular, he is interested in finding out for each aircraft model the airline owns, what percentage of its routes are flown by the aircraft model.

You are to implement the function `get_airline_equipment_info` which takes as inputs the name of an airline and the name of the CSV file, and outputs a dictionary where the keys are the aircraft model and the values are the percentages of routes flown.

As the names of the airports are not unique, a unique route is identified by the source airport, city and country, and the destination airport, city and country. The opposite direction is counted as another unique route.

Note that because an airline can use several different aircraft models on one route, the percentages do not necessarily sum to 100. For example, if an airline has only one route but 3 different airplane models fly that route, the percentages will be 100 for all 3 models.

[5 marks]

Sample execution:

```
>>> get_airline_equipment_info("AirAsia", "airline_routes.csv")
{'A320-100/200': 100.0}
# AirAsia only has the A320-100/200

>>> get_airline_equipment_info("SilkAir", "airline_routes.csv")
{'A319': 58.06451612903226, 'A320-100/200': 82.79569892473118,
 'A330-300': 2.1505376344086025, '737-800': 23.655913978494624}
# SilkAir files 93 unique routes, of which
# the A319 flies 54 routes (54/93 = 58.06%),
# the A320-100/200 files 77 routes (77/93 = 82.8%),
# the A330-300 flies 2 routes (2/93 = 2.15%)
# and the 737-800 flies 22 routes (22/93 = 23.66%),
```

Question 3 : Despicable Me [10 marks]

– BACKGROUND STORY (Okay to skip) –

Gru, a supervillain, has his pride injured when an unknown supervillain steals the Great Pyramid of Giza, an action that is described by his colleague Dr. Nefario as "making all other villains look lame." Gru decides to do better, with the assistance of Dr. Nefario, by shrinking and stealing the Moon, an idea based on his childhood dream of being an astronaut, which was always disparaged by his mother Marlena. The plan is expensive and Gru seeks a loan from the Bank of Evil, where the president Mr. Perkins is impressed by the plan, but will only provide the money if Gru can obtain the necessary shrink ray first.

Gru and his Minions steal the shrink ray from a secret base in East Asia, but the up-and-coming supervillain, Vector, who was also responsible for the Pyramid theft, immediately steals it from Gru, as revenge for freezing his head earlier. Gru attempts to break into Vector's fortress to get the shrink ray back, but is defeated by numerous booby traps. However, he notices three orphan girls, Margo, Edith, and Agnes, who are able to easily walk into the base because they are selling cookies. Gru disguises himself as a dentist and adopts the girls from Miss Hattie's Home for Girls, planning on using them to infiltrate Vector's base so he can get the shrink ray back. However, Gru has difficulty nurturing them properly due to their rambunctiousness, their ballet classes, and his own ineptitude as a parent.

Eventually, Gru and the girls arrive at Vector's fortress and Gru manages to steal the shrink ray. The girls then suggest a day at a theme park; Gru agrees, believing he can abandon the girls there, but he is later told by an attendant that they must be accompanied by an adult. He is then dragged around the theme park for the day, eventually warming to the girls after they compliment him on blowing up a rigged carnival game.

Later, Gru contacts Mr. Perkins, stating that he finally has the shrink ray in his possession. Margo, Edith, and Agnes interrupt the meeting, and Perkins announces that he has lost confidence in Gru and will no longer fund his operations. As Gru tells the Minions he can no longer pay them for their services, the girls offer the contents of their piggy bank to fund the plan. The Minions then hand over their own savings, too. Gru, inspired, sacrifices parts of his lair to construct a spacecraft. Gru plans to steal the Moon when it is nearest the Earth, but this ends up being the same day as the girls' ballet recital. Gru becomes conflicted, and Dr. Nefario, seeing the recital as interfering with the plan, arranges for the girls to be returned to the orphanage. At the same time, Mr. Perkins informs Vector (who is revealed to be his son) of Gru's possession of the shrink ray and the adoption of the three girls, encouraging Vector to take action.

Gru successfully shrinks and steals the Moon, but is too late to attend the recital—finding a note from Vector, who has kidnapped the girls, and will exchange the Moon for them. After arriving at Vector's headquarters, Gru readily makes the trade, but Vector reneges on the deal, flying off with the girls and the Moon, much to Gru's anger. Meanwhile, Dr. Nefario discovers that the effects of the shrink ray are temporary; the bigger the object was originally, the faster it will regain its original size. As the Moon starts to expand in Vector's ship, Gru, Dr. Nefario, and the Minions pull off a daring mid-air rescue of the girls just as the Moon explodes out of Vector's ship and launches itself back into orbit, with Vector trapped on it.

Figure 1: Gru and his minions stealing the shrink ray from Vector's lab.

Sometime later, Gru has readopted the girls and treats them as his daughters, writing them a bedtime storybook framed around his own experience. The film ends with the girls performing their own ballet recital for Gru, Marlena, Dr. Nefario, and the Minions.

Source: Wikipedia

— START OF ACTUAL QUESTION —

Warning: Please read the entire question carefully and plan well before starting to write your code.

In this question, you will model and implement two classes: `Thing` and `ShrinkRay`. `Thing` models objects in the world, and has a name and some mass. `ShrinkRay` models a Shrink Ray that is capable of shrinking the mass of a `Thing`.

When a `Thing` is shrunk by a `ShrinkRay`, its mass will be reduced by dividing the mass of the `Thing` with the mass of the `ShrinkRay`. For example, a `Thing` which is 20 kg, when shrunk by a 4 kg `ShrinkRay` will become $20/4 = 5$ kg in mass. This shrinking effect is not permanent and will eventually wear off, returning the `Thing` to its original mass.

A `Thing` is created with two arguments, the name (which is a String) and the mass (which is a float) in kg. `Thing` supports the following **three** methods:

- `get_name()` which returns the name of the `Thing`.
- `get_mass()` which returns the current mass of the `Thing`.
- `wait_for(time)` which takes as input a time duration (in min) and causes that amount of time to "pass by" for the `Thing`. The return value will be a String according to the following conditions:
  - Since the effects of shrinking is only temporary, each `ShrinkRay` only shrinks the `Thing` for a given amount of time. Once this amount of time has "passed by" since being shrunk, the effects of shrinking will be reversed. A String `'<Thing name> has unshrunk to <mass> kg'` will be returned.

  - Otherwise, if there is no "unshrinking" to be done, the String `'Nothing happens after <time> mins'` will be returned.

A `ShrinkRay` is a subclass of `Thing`, and is created with three arguments: m, its mass (which is a float) in kg, the time duration t (in min) of the shrinking effect and n the number of items it can concurrently shrink. Because Shrink Rays work by temporary suspending the atoms of a thing in another dimension, there is a limit on how many things it and shrink at any one time. As a subclass of `Thing`, the name of the every `ShrinkRay` is `"Shrink Ray"`.

`ShrinkRay` supports the following two methods:

- `shrink(thing)` will cause the mass of `thing` to be temporary reduced by a proportion given by the mass of the `ShrinkRay` (See earlier example). The return value will be a String according to the following conditions:
  - If the input `thing` is itself, the String `'Shrink Ray cannot shrink itself'` will be returned.
  - If the shrink ray is currently shrinking n things, it cannot shrink any more thing. The String `'Maximum number of objects shrunk'` is returned.
  - Otherwise, the mass of `thing` will be reduced accordingly and the String `'<Thing name> has shrunk to <Thing mass> kg'` will be returned.
- `things_shrunk()` will return a tuple containing the **names** of the Things that are currently being shrunk by the ray (the order of the elements does not matter).

Sample execution:

```
>>> kevin = Thing("Kevin", 25)
>>> bob = Thing("Bob", 20)
>>> moon = Thing("Moon", 7 * 10**22)
>>> kevin.get_name()
'Kevin'

>>> kevin.get_mass()
25

>>> sr1 = ShrinkRay(4, 5, 2)
>>> sr1.shrink(kevin)
'Kevin has shrunk to 6.25 kg'

>>> kevin.wait_for(3)
'Nothing happens after 3 mins'

>>> sr1.shrink(bob)
'Bob has shrunk to 5.0 kg'

>>> sr1.shrink(moon)
'Maximum number of objects shrunk'

>>> sr1.things_shrunk()
```

```
('Kevin', 'Bob')

>>> kevin.wait_for(3)
'Kevin has unshrunk to 25 kg'

>>> sr1.shrink(moon)
'Moon has shrunk to 1.75e+22 kg'

>>> sr1.things_shrunk()
('Bob', 'Moon')

>>> bob.get_mass()
5.0

>>> bob.wait_for(5)
'Bob has unshrunk to 20 kg'
```

It turns out, that since Shrink Rays are also Things, they can be shrunk by other Shrink Rays as well! And not only that, when shrunk, their shrinking power is reduced too! Thankfully, as inter-dimensional devices, Shrink Rays have a minimum mass of 1 kg regardless of any shrinking effect placed on them. Also, to prevent any inter-dimensional rift from happening, we **will not** create a shrinking loop, e.g., sr1 shrinks sr2 and then sr2 shrink sr1.

Sample Execution (continued):

```
>>> sr1.shrink(kevin)
'Kevin has shrunk to 6.25 kg'

>>> sr2 = ShrinkRay(2, 2, 3)
>>> sr2.shrink(sr1)
'Shrink Ray has shrunk to 2.0 kg'

>>> kevin.get_mass()
12.5 # Kevin's mass changed to 25/2 = 12.5 because sr1 is now 2 kg

>>> sr1.shrink(sr1)
'Shrink Ray cannot shrink itself'

>>> sr3 = ShrinkRay(5, 2, 1)
>>> sr3.shrink(sr1)
'Shrink Ray has shrunk to 1 kg' # minimum mass is 1 kg

>>> kevin.get_mass()
25 # Kevin is now back to 25 kg since sr1 is now 1 kg

>>> sr1.wait_for(5)
'Shrink Ray has unshrunk to 4 kg' # sr2 and sr3 effect wears off
```

```
>>> kevin.get_mass()
6.25 # Kevin is now shrunk to 25/4 since sr1 is now 4 kg

>>> kevin.wait_for(5)
'Kevin has unshrunk to 25 kg'
```

One last thing, notice that Things can be shrunk several times by the same or different Shrink Rays!

Sample Execution (continued)

```
>>> sr1.shrink(kevin)
'Kevin has shrunk to 6.25 kg' # sr1 has a time of 5

>>> kevin.wait_for(4)
'Nothing happens after 4 mins'

>>> sr2.shrink(kevin)
'Kevin has shrunk to 3.125 kg' # sr2 has a time of 2

>>> kevin.wait_for(1)
'Kevin has unshrunk to 12.5 kg' # sr1 effect wears off

>>> kevin.wait_for(1)
'Kevin has unshrunk to 25 kg' # sr2 effect wears off
```

You are advised to solve this problem incrementally. Even if you cannot fulfill all the desired behaviours, you can still get partial credit for fulfilling the basic behaviours.

Question 4 : Burglary [10 marks]

A professional burglar is planning to break into houses along a street in a neighbourhood. Each house has a certain amount of money stashed and the only constraint stopping him from breaking into every one of them is that adjacent houses have a connected security system which will automatically contact the police if both adjacent houses are broken into.

To make matters worse, the street or this neighbourhood is circular, with the houses arranged in a circle such that the first house and last house are neighbours.

A list of non-negative integers represents the amount of money contained in each house. Write a function `rob()` which takes in the list as an input and return the the maximum amount of money that can be stolen without the police being alerted.

Sample Execution:

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

>>> rob([0, 1, 0, 1, 0, 0, 1, 0, 1, 0])
4

>>> rob(list(range(10)))
25

>>> street3 = [183, 219, 57, 193, 94, 233, 202, 154, 65, 240, 97,
               234, 100, 249, 186, 66, 90, 238, 168, 128, 177, 235,
               50, 81, 185, 165, 217, 207, 88, 80, 112, 78, 135,
               62, 228, 247, 211]
>>> rob(street3)
3365

>>> rob(list(range(100)))
2500
```

— E N D O F P A P E R —