Cs1010s-practical-apr17

Programming Methodology (National University of Singapore)

Studocu is not sponsored or endorsed by any college or university
Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

CS1010S — Programming Methodology
National University of Singapore

Practical Examination

15 April 2017

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.

GOOD LUCK!

This document is available free of charge on studocu

Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

Question 1 : 1337 speak [10 marks]

"Leet (or '1337'), also known as eleet or leetspeak, is an alternative alphabet for many languages that is used primarily on the Internet. It uses some characters to replace others in ways that play on the similarity of their glyphs via reflection or other resemblance. For example, leet spellings of the word leet include 1337 and l33t; eleet may be spelled 31337 or 3l33t."

- Source: Wikipedia

We can use Python to help us easily translate words to leet to impress0r our friends by using a dictionary of characters to replace. The keys of the dictionary will be a single character and the value will be the replacement characters.

A. The function `l33tify` takes as inputs a string and a dictionary, and outputs the string translated to leet according to the dictionary.

Examples:

```
>>> l33t_dict = {
    'a': '4',
    'b': '8',
    'c': '(',
    'e': '3',
    'g': '[',
    'i': '1',
    'o': '0',
    's': '5',
    't': '7',
    'x': '%',
    'z': '2',
    }

>>> l33tify("pheer my leet skills", l33t_dict)
'ph33r my l337 5k1ll5'

>>> l33tify("python is cool", l33t_dict)
'py7h0n 15 (00l'
```

Implement the function `l33tify`. You may assume that **all the characters of the input string are lowercase** and the keys of the dictionary are single characters.

[5 marks]

B. The `l33tify` function is not very leet because it always replaces with a fixed character. Since there are usually multiple possible replacements for a character, we can do better by having our function rotate through the possible replacements.

For example, the character '`l`' can be replaced by the following list of strings: `['|_', '|', '1']`. In this case, the first replacement of '`l`' will use `'|_'`, the second will use `'|'`, the third will use `'1'`. The next replacement will go back to the first string `'|_'`, and the cycle continues.

Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

The values of the leet dictionary will now be a list of strings, and the order of the replacement will follow the order of the list.

The function `advance_l33tify` takes as inputs a string and a dictionary, and outputs the string translated to leet according to scheme described above.

Examples:

```
>>> adv_l33t_dict = {
        'a': ['4', '/-\\', '/_\\', '@', '/\\'],
        'b': ['8', '|3', '13', '|}', '|:', '|8', '18', '6', '|B'],
        'c': ['<', '{', '[', '('],
        'd': ['|)', '|}', '|]'],
        'e': ['3'],
        'f': ['|=', 'ph', '|#', '|"'],
        'g': ['[', '-', '[+', '6'],
        'h': ['|-|', '[-]', '{-}', '|=|', '[=]', '{=}'],
        'i': ['1', '|'],
        'j': ['_|', '_/', '_7', '_)'],
        'k': ['|<', '1<'],
        'l': ['|_', '|', '1'],
        'm': ['|\\/|', '^^', '/\\/\\'],
        'n': ['|\\|', '/\\/', '/V', '][\\\\]['],
        'o': ['0', '()', '[]', '{}'],
        'p': ['|o', '|O', '|>', '|*', '|Âř', '|D', '/o'],
        'q': ['O_', '9', '(', ')', ''],
        'r': ['|2', '12', '.-', '|^'],
        's': ['5', '$', '§'],
        't': ['7', '+', '7`', "'|'"],
        'u': ['|_|', '\\_\\', '/_/', '\\_/', '(_)'],
        'v': ['\\/'],
        'w': ['\\/\\/', '(/\\)', '\\^/', '|/\\|'],
        'x': ['%', '*', '><', '}{', ')('],
        'y': ['`/', '¥'],
        'z': ['2', '7_', '>_']
    }

>>> print(advance_l33tify("Bow b4 me 4 I am root!!!", adv_l33t_dict))
80\/\/ |34 |\/|3 4 1 4^^ |2()[]7!!!

>>> print(advance_l33tify("Mississippi", adv_l33t_dict))
|\/|15$|§51|o|0|
```

Provide an implementation for the function `advance_l33tify`. Note that for this part, the input string can be a **mix of uppercase and lowercase characters**, while the dictionary keys will always be a single, lowercase character. You can also assume that the lists in the dictionary values will not be empty.

Tip: Be mindful when you are modifying a mutable input parameter. [5 marks]

This document is available free of charge on studocu

Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

Question 2 : 3G Coverage [10 marks]

Important note: You are provided with a data file `3g-coverage.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.

It is time to renew your mobile phone contract and you wonder which telco provides the best coverage. You decided to review the historical data that IMDA has collected over the past few years.

The first line of the data file is a header which describes each column of data. You may assume that all data files follow this same fixed order of columns and there are no duplicate data for a telco in a month.

A. You wish to find out the average coverage of all available telcos in any given year.

Implement the function `yearly_average` that takes as input a filename and a year (as strings). The function returns a dictionary which keys are the telcos obtained from the data file and the values are the average coverage (rounded to 4 decimal places) for the stated year. You can use the function `round(n, d)` to round n to d decimal places.

Note, the data for some telcos might not be available for certain years and there might not always be data for every month in the year. [5 marks]

Sample execution:

```
>>> yearly_average("3g-coverage.csv", "2015")
{'Singtel': 99.5308, 'M1': 99.5358, 'Starhub': 99.7608}

>>> yearly_average("3g-coverage.csv", "2013")
{'Singtel': 99.4078, 'M1': 98.6622, 'Starhub': 99.4544}
```

B. Sometimes the average does not give an accurate picture. You think it might be better to compare the telcos on a monthly basis. Essentially, you want to know which telco has the best coverage for each month, and how many times each telco win the "best telco of the month" in a given year.

Implement the function `best_telco` that takes as input a filename and a year (as strings). The function returns a dictionary which keys are the telcos obtained from the data file and the values are number of times the telco was the "best telco of the month" in the given year.

Note, for this part, you may assume that every month will have all the same telcos represented and that there is always one best telco. However, it may still be the case that data for some months are not available. You should also still assume that names of the telcos are not fixed. [5 marks]

Sample execution:

```
>>> best_telco("3g-coverage.csv", "2015")
{'Singtel': 0, 'M1': 1, 'Starhub': 11}

>>> best_telco("3g-coverage.csv", "2015")
{'Singtel': 5, 'M1': 0, 'Starhub': 4}  # only 9 months in 2013
```


Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

Question 3 : Villains [10 marks]

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

Villains are evil people out to do bad things in the world. They accomplish their evil doings using Gadgets which increases their reputation and evilness.

Figure 1: Gru, a villain with his Freeze Ray.

The class `Villain` models a villain and the class `Gadget` models a gadget.

`Gadget` is created with two inputs, its name (which is a string) and an awesomeness value (which is an integer). A `Gadget` can only be owned by at most one `Villain` at any time.

`Gadget` supports the following methods:

- `get_description()` returns the string `"<Gadget name> has level <Gadget awesomeness> awesomeness"`
- `owned_by()` returns the string "<Gadget name> belongs to <Owner name>" if it is currently owned by a villain. Otherwise, the string `"<Gadget name> is unowned"` is returned.

`Villain` is created with one input, its name, which is a string. Every villain starts out with an evilness of 0, has no proficiency in any gadgets and owns no gadgets. Each time a villain does an evil deed with a gadget, the gadget's awesomeness will decrease by 1 and the villain's proficiency in that gadget will increase by 1.

It is possible for different gadgets to have the same name. In which case they are considered separate gadgets and a villain's proficiency in them are independent. It is also possible for a villain to own more than one gadget of the same name.

`Villain` supports the following methods:

- `get_evilness()` returns the current evilness of the villain, which is an integer.

This document is available free of charge on studocu

Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

- `gadgets_owned()` returns a tuple of the names of the gadgets the villain currently possesses.
- `get_proficiency(gadget)` takes as input a `Gadget`. If the villain's proficiency for the gadget is more than 0, the string `"<Villain name>'s proficiency with <Gadget name> is <proficiency>"` is returned, where `<proficiency>` is the villain's proficiency value for that gadget. If the villain is not proficient with the gadget at all, the string `"<Villain name> is not proficient with <Gadget name>"` is returned.
- `do_evil(action, gadget)` takes as inputs an action (which is a string) and a `Gadget`. It returns a string and performs some actions based on the following conditions:
  - If the `gadget` is not in the villain's possession, the string `"<Villian name> does not have <Gadget name>"` is returned.
  - Otherwise, the following actions will occur: 1) The villain's evilness increases by the sum of the gadget's awesomeness and the villain's proficiency of the gadget; 2) The gadget's awesomeness will decrease by 1, but to not less than 0; and 3) the villain's proficiency of the gadget will increase by 1.

    The string `"<Villain name> <action> with <Gadget name>"` is returned.
- `steals(gadget)` takes as input a `Gadget` and returns a string while performing some actions based on the following conditions:
  - If the villain already owns the gadget, the string `"<Villain name> already has <Gadget name>"` is returned.
  - Otherwise, the villain steals the gadget and is now owns it. The gadget's awesomeness is also restored to its original value.

    If the gadget was previously owned by another villain, the other villain's evilness will be halved (rounded down to the nearest integer) and the string `"<Villain name> steals <Gadget name> from <Previous Owner name>"` is returned. Else the gadget had no previous owner and the string `"<Villain name> steals <Gadget name>"` is returned.
- `envy(other)` takes as input either a `Gadget` or `Villain` and returns a string based on the following conditions:
  - If `other` is a `Gadget` without an owner, the string `"<Villain name> envies <Gadget name>"` is returned.
  - If `other` is a `Gadget` currently owned by another villain, the string `"<Villain name> envies <Other Villain name>'s <Gadget name>"` is returned.
  - If `other` is a `Gadget` currently owned the villain, the string `"<Villain name> already has <Gadget name>"` is returned.
  - If `other` is the villain himself, the string `"<Villain name> cannot envy himself"` is returned.
  - If `other` is another `Villain` who has the same evilness as the villain, the string `"Nobody is envious"` is returned.

Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

- If `other` is another `Villain` with a different evilness, the string `<Villain A name> envies <Villain B name>"` is returned, where Villain A has a lower evilness than Villain B.

Provide an implementation for the classes `Villain` and `Gadget`.

For simplicity, you do not have to worry about data abstraction and can access the properties of both classes directly. Take careful note of the characters in the returned strings, especially spaces and punctuation.

Sample Execution:

```
>>> gru = Villain("Gru")
>>> vector = Villain("Vector")
>>> freeze_ray = Gadget("Freeze Ray", 5)
>>> lava_gun = Gadget("Lava Lamp Gun", 3)


>>> gru.get_evilness()
0

>>> gru.gadgets_owned()
()

>>> freeze_ray.get_description()
"Freeze Ray has level 5 awesomeness"

>>> freeze_ray.owned_by()
"Freeze Ray is unowned"

>>> gru.steals(freeze_ray)
"Gru steals Freeze Ray"

>>> gru.gadgets_owned()
('Freeze Ray',)

>>> freeze_ray.owned_by()
"Freeze Ray belongs to Gru"

>>> gru.get_proficiency(freeze_ray)
"Gru is not proficient with Freeze Ray"

>>> gru.do_evil("robs a bank", freeze_ray)
"Gru robs a bank with Freeze Ray"

>>> gru.get_evilness()
5  # Gru evilness increase by Freeze Ray's awesomeness

>>> gru.get_proficiency(freeze_ray)
"Gru's proficiency with Freeze Ray is 1"
```


This document is available free of charge on studocu

Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

```
>>> freeze_ray.get_description()
"Freeze Ray has level 4 awesomeness"  # Freeze Ray awesomeness
   decreases

>>> gru.do_evil("steals candy", freeze_ray)
"Gru steals candy with Freeze Ray"

>>> gru.get_proficiency(freeze_ray)
"Gru's proficiency with Freeze Ray is 2"

>>> gru.get_evilness()
10

>>> freeze_ray.get_description()
"Freeze Ray has level 3 awesomeness"

>>> gru.envy(freeze_ray)
"Gru already has Freeze Ray"

>>> vector.envy(freeze_ray)
"Vector envies Gru's Freeze Ray"

>>> gru.envy(vector)
"Vector envies Gru"

>>> vector.steals(freeze_ray)
"Vector steals Freeze Ray from Gru"

>>> gru.get_evilness()
5  # Gru got his evilness halved

>>> freeze_ray.get_description()
"Freeze Ray has level 5 awesomeness"  # awesomeness restored to
   original

>>> freeze_ray.owned_by()
"Freeze Ray belongs to Vector"

>>> gru.gadgets_owned()
()

>>> vector.do_evil("freezes Miami", freeze_ray)
"Vector freezes Miami with Freeze Ray"

>>> vector.get_evilness()
5
```


Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)

```
>>> gru.envy(vector)
"Nobody is envious"

>>> gru.envy(lava_gun)
"Gru envies Lava Lamp Gun"

>>> gru.do_evil("steals Freeze Ray", lava_gun)
"Gru does not have Lava Lamp Gun"

>>> gru.envy(lava_gun)
"Gru envies Lava Lamp Gun"

>>> gru.steals(lava_gun)
"Gru steals Lava Lamp Gun"

>>> gru.do_evil("steals the Queen's crown", lava_gun)
"Gru steals the Queen's crown with Lava Lamp Gun"

>>> gru.get_evilness()
8

>>> gru.envy(vector)
"Vector envies Gru"

>>> gru.steals(freeze_ray)
"
Gru steals Freeze Ray from Vector"

>>> vector.get_evilness()
2

>>> gru.get_evilness()
8

>>> gru.gadgets_owned()
('Lava Lamp Gun', 'Freeze Ray')

>>> gru.do_evil("freezes Vector", freeze_ray)
"Gru freezes Vector with Freeze Ray"

>>> gru.get_evilness()
15
```

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.

— E N D O F P A P E R —

This document is available free of charge on studocu

Downloaded by Daniel Lim (daniel.lim.ws.90@gmail.com)