The Chinese University of Hong Kong
2020 – 2021 Term 2

CSCI1040 Hands-on Introduction to Python
Online Quiz

Date: March 17, 2021

Quiz period: 11:40am to 12:55pm

Instructions:

(1) There are four questions in this quiz and answer ALL of them.

(2) Each question carries equal marks (25%).

(3) Question 4 is associated with an input text file. Download it from Blackboard along with this paper.

(4) You may assume that all input values (entered by the user or read from a file) are valid, and input validation or exception handling is NOT required unless specified otherwise.

(5) You are NOT allowed to import other modules (standard or third-party) that are not specified in the questions to solve the problems.

(6) Use a separate Python source file to answer each question.

(7) Name your source files for questions 1 through 4 as q1.py, q2.py, q3.py and q4.py respectively.

(8) Submit all source files to Blackboard. No need to zip them.

(9) You may submit multiple times, but only the last attempt will be graded. So, your last attempt must attach ALL the script files even if you have submitted some of them in earlier attempt(s).

(10) Stop working and start submission at or no later than 12:55 noon. You are given an extra 5-minute grace period to finish the submission by 13:00.

(11) The submission will close sharply at 13:00. After that, no more submissions will be accepted.

(12) Important announcements on the quiz paper, if any, will be made on the general channel of Slack for this course. You may keep https://csci1040-2021.slack.com running in a web browser on the computer that you used to attempt this quiz.

Question 1 (25%) [q1.py]

Let *w* and *h* be two integers ≥ 1. Write a Python program to print a hollow rectangle pattern composed of *w* rows and *h* columns of asterisks. The values of *w* and *h* are entered by the user via console input. There is a single space following after every asterisk in the same row.

Hint: you may use the `split()` method of the `str` class to split the user input string into two parts, and convert them to integers *w* and *h*.

Sample runs:

Note: The underlined blue text represents the user input.

```
Enter width and height: 4 3↵
* * * *
*     *
* * * *

Enter width and height: 5 6↵
* * * * *
*       *
*       *
*       *
*       *
* * * * *

Enter width and height: 10 7↵
* * * * * * * * * *
*                 *
*                 *
*                 *
*                 *
*                 *
* * * * * * * * * *

Enter width and height: 3 1↵
* * *

Enter width and height: 1 2↵
*
*

Enter width and height: 1 1↵
*
```

Question 2 (25%) [q2.py]

Credit card numbers must follow certain patterns. A credit card number must have between 13 and 16 digits, and must pass a validity check illustrated as follows.

Consider the card number 4388576018402626.

1. Double every second digit from right to left, i.e., the digits underlined / highlighted in red above. If doubling of a digit results in a two-digit number, add up the two digits to get a single-digit number (which can also be obtained by subtracting 9 from the two-digit result):
   - $2 \times 2 = \mathbf{4}$
   - $2 \times 2 = \mathbf{4}$
   - $4 \times 2 = \mathbf{8}$
   - $1 \times 2 = \mathbf{2}$
   - $6 \times 2 = 12 \implies 1 + 2 = \mathbf{3}$ (or 12 - 9 = 3)
   - $5 \times 2 = 10 \implies 1 + 0 = \mathbf{1}$ (or 10 - 9 = 1)
   - $8 \times 2 = 16 \implies 1 + 6 = \mathbf{7}$ (or 16 - 9 = 7)
   - $4 \times 2 = \mathbf{8}$
2. Sum all single-digit numbers obtained from step 1:
   - 4 + 4 + 8 + 2 + 3 + 1 + 7 + 8 = 37
3. Sum all digits in the odd places from right to left in the card number:
   - 6 + 6 + 0 + 8 + 0 + 7 + 8 + 3 = 38
4. Sum the results from step 2 and step 3:
   - 37 + 38 = 75

If the result of step 4 is divisible by 10, the card number is valid; otherwise, it is invalid.

So, the above card number 4388576018402626 is invalid. On the other hand, the card number 4388576018410707 is valid.

Write a function called `validate()` to implement this validation scheme. It accepts a string representing the card number, and returns `True` if the number is valid, and `False` otherwise.

Sample output:

```
print(validate('4388576018402626')) # prints False
print(validate('5219761217451578')) # prints False

print(validate('4388576018410707')) # prints True
print(validate('5260791805687041')) # prints True
print(validate('30386383888630'))   # prints True

print(validate('79927398713')) # prints False (too short, < 13 digits)
print(validate('3529091773866439059')) # prints False (as > 16 digits)
```

Question 3 (25%) [q3.py]

The series S_n of a sequence of numbers is the sum of the sequence to a certain number (n) of terms. For example, for the sequence 2, 4, 6, 8, ... , the sum to 3 terms = S_3 = 2 + 4 + 6 = 12.

Create an abstract base class called `Series` which defines:

- an instance variable `self.n` that represents the number of terms in the series;
- a constructor that receives an integer parameter `n`, which is used to initialize `self.n`;
- an abstract instance method called `compute()`.

Create a class `GeometricSeries` which inherits from `Series` and defines:

- an instance variable `self.a` that represents the first term in the series;
- an instance variable `self.r` that represents the common ratio in the series;
- a constructor that receives parameters `a`, `r`, and `n`, initializes `self.a` and `self.r` using `a` and `r` respectively, and call the superclass's constructor to initialize `self.n`;
- an instance method `compute()`, which returns the sum to *n* terms as follows:

$$\sum_{k=0}^{n-1}(ar^k) = a + ar + ar^2 + \ldots + ar^{n-1}$$

Sample usage:

```
g1 = GeometricSeries(a=10, r=3, n=4)
g2 = GeometricSeries(a=1/2, r=1/2, n=5)
g3 = GeometricSeries(a=1/2, r=1/2, n=10)

print(g1.compute()) # prints 400
print(g2.compute()) # prints 0.96875
print(g3.compute()) # prints 0.9990234375
```

Note:

You may import anything useful from Python's `abc` module for making the abstract class.

Restriction:

You are required to use looping techniques to compute the sum; **don't** use the formula:

$$\sum_{k=0}^{n-1}(ar^k) = a\left(\frac{1-r^n}{1-r}\right)$$

Question 4 (25%) [q4.py]

Given an input data text file named "q4data.txt" (downloadable from Blackboard) which stores a few lines of text quotes.

Write a class `WordCounter` which defines:

- a constructor that accepts a string parameter for receiving the file name;
- an instance method called `frequency()`;
- a class variable called `trivial_words` that is assigned the following list:

```
['in', 'at', 'on', 'of', 'to', 'for', 'a', 'an', 'the', 'and']
```

The `frequency()` method counts the number of occurrences of each word that appears in the text file in a case-insensitive manner, and returns a dictionary with each word as key and its frequency of occurrences as value.

For example, the word "love" (ignore the case) appears four times in the file, so an item `"love": 4` (key written in all lowercase) should be added to the dictionary.

However, for those words (again, case-insensitive) that exist in the `trivial_words` list, they are not counted and/or are skipped in the dictionary being returned.

Some words are followed by a punctuation mark. They should be counted as if there were no such punctuation. And you are required to handle only four punctuation marks which are period, comma, semicolon and exclamation mark (`.` `,` `;` `!`).

Up to your design, you may open and read the file in the constructor or in the `frequency()` method. Exception handling on file I/O is NOT required.

Sample client code:

```
wc = WordCounter("q4data.txt")
# your code has to import the json module
print(json.dumps(wc.frequency(), indent=4, sort_keys=True))
```

Sample output:

```
{
    "all": 1,
    "do": 2,
    "enemies": 1,
    "front": 1,
    "hatred": 1,
    "have": 1,
    "kindness": 1,
    "lead": 1,
    "love": 4,
    "manage": 2,
    "neighbors": 1,
    "none": 1,
    "not": 2,
    "patience": 1,
```

```
    "people": 4,
    "practice": 1,
    "righteousness": 1,
    "so": 1,
    "things": 1,
    "who": 1,
    "you": 3,
    "your": 3
}
```


- The end of the paper -