Python Lab Assignment – 2

1. Compare uppercase and lowercase of your name using set() and {} syntax.
2. Write first seven Fibinacci numbers using generator next function/ yield in python.
3. Write a code which yields all terms of the geometric progression a, aq, aq^2 , aq^3 , ... When the progression produces a term that is greater than 100,000, the generator stops (with a return statement). Compute total time and time within the loop.
4. Create a generator expression for first 10 cubes.
5. Write a program to compute square area of square class with self() to get square value in python.
6. Create book, ebook, journal classes to use inheritance with title, publisher, page, year of publishing details.
7. Show multiple inheritance in shape, 2-D shapes, 3-D shapes, square, rectangle, polygon, hexagon, cube, cone, cylinder etc. classes with their areas.
8. Search for palindrome and unique words in a text using class method and string method.
9. Check and set a person's age in person class using property decorator.
10. Write a operator overloading for “len” which shows string length for any given string and return only length of last three words if the string is in "Hello! I am 42 years old!" format.
11. Write a operator overloading for “len” which shows string length for any given string and return only length of repetitive words with the text if the text has some repetitive parts. Determine the most frequently occurring words using most_common.
12. Write a function that flattens a nested dictionary structure like one obtained from Twitter and Facebook APIs or from some JSON file.

```
nested = {
    'fullname': 'Alessandra',
    'age': 41,
    'phone-numbers': ['+447421234567', '+447423456789'],
    'residence': {
        'address': {
            'first-line': 'Alexandra Rd',
            'second-line': '',
            Testing, Profiling, and Dealing with Exceptions
            [ 230 ]
```

},

'zip': 'N8 0PP',

'city': 'London',

'country': 'UK',

},

}

13. Use parameterized or nose_parameterized to compute power of following values:

(2, 2, 4),

(2, 3, 8),

(1, 9, 1),

(0, 9, 0). Use pytest to check errors.

14. Use profile/cprofile to check pythogorian triples code in python. Think about time complexity of the code.

15. Write a program to sort in descending order by the sum of credits accumulated by students, so to have the best student at position 0. Write a function using map, to produce a decorated object, to sort, and then to undecorate. Each student has credits in three (possibly different) subjects. To decorate an object means to transform it, either adding extra data to it, or putting it into another object, in a way that allows to sort the original objects the way you want. After the sorting, one reverts the decorated objects to get the original ones from them. This is called to undecorate.

16. Write a python program to calculate the number of editing operations (substitution, deletion and insertion) in the output sequence with respect to a given reference input. Prepare the Minimum Edit Distance (MED) Table and print the backtrace to MED (Consider the root form of words while calculating the number of editing operations)

17. Write a single python program to do the following operations on a text file by writing different user defined functions.

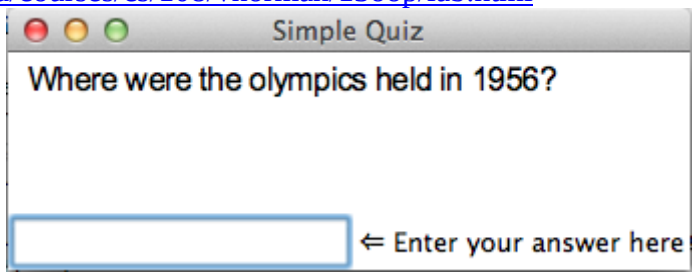
a. Remove all the special characters.
b. Remove all single characters.
c. Substitute multiple spaces with single space.
d. Convert all the words into Lowercase.
e. Convert the words into literal form from their contracted form (e.g., Couldn't → Could not).

18. Using Numpy create random vector of size 15 having only Integers in the range 0 -20. Write a program to find the most frequent item/value in the vector list.

19. Check http://yann.lecun.com/exdb/mnist/ web page. Execute the training-testing model of classifications and compare accuracy and other ROC measures for the classification solutons for any two algorithms using Python among - K-NN with non-linear deformation (IDM), K-NN, shape context

matching, Y. LeCun, L. Bottou and Y. Bengio: Reading Checks with graph transformer networks, 3-layer NN, 500+300 HU, softmax, cross entropy, weight decay, 6-layer NN 784-2500-2000-1500-1000-500-10 (on GPU) [elastic distortions], Convolutional net Boosted LeNet-4, [distortions], committee of 35 conv. net, 1-20-P-40-P-150-10 [elastic distortions].

20. Create a quiz GUI using inheratnce and polymorphism in Python to ask a sequence of questions of the user. You may follow controller.py, quiz.py or shortAnswer.py files or may write your own codes. The interface is shown here: https://cs.calvin.edu/courses/cs/108/vnorman/13oop/lab.html



The application displays problems in a text area (on the top) and messages (on the bottom right); the user types answers in a text box (on the bottom left) and presses enter.

Three classes work together to create the application:

- **Controller** - implements a GUI driver for a quiz.
- **Quiz** - implements a simple quiz with short answer problems.
    - Keeps track of current problem.
    - Randomizes the order of the problems.
    - Keeps track of whether or not all problems have been used.
- **ShortAnswer** - implements a simple problem class with a string question and string answer.
    - Provides interface for asking the question.
    - Provides interface for checking if a provided answer is correct.

Exercise 20.1

Create a new package for this lab called Exercise20 and copy the starting code files into this package: controller.py, quiz.py, shortAnswer.py (files attached).

Familiarize yourself with the quiz mechanism by doing the following:

- Run the controller a couple times;
- Run the unit tests for the ShortAnswer problem class;
- Add a new short-answer problem (of your choice) to the quiz.

Inheritance

Right now this quiz mechanism can only ask short answer problems, but this is a bit too limited for the purposes. You will add fill-in-the-blank problems, true-false problems, and maybe even multiple choice problems. As we start planning, we realize that we will be duplicating code if we write each kind of problem from scratch. Instead, let's start with a Problem class that will be the parent class of

all the different kinds of problems. This class will collect all of the attributes and methods that are shared between all problems.

So what is shared between all problems? Each problem has some text, but asks the question in a different way (e.g. short-answer just added a question mark, but a fill-in-the-blank problem should add both the question mark and an indication to "Fill in the blank."). Further, all problems have answers, but a true-false problem has a boolean answer instead of a string. Considering these properties, we proceed as follows:

Exercise 20.2

Refactor your code to include a Problem class and a ShortAnswer class that inherits from Problem.

The **new** Problem class:

- Includes a constructor that receives a string and stores it in an instance variable called self.text
- Includes an accessor for the question called get_text()
- Put this in a separate file.

The **updated** ShortAnswer class:

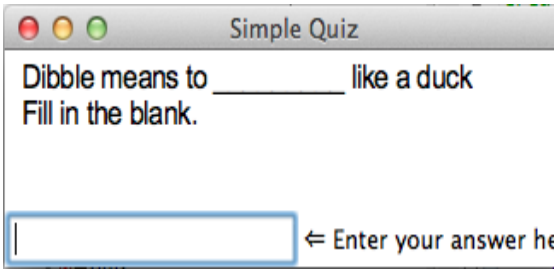
- Has an updated class declaration that indicates it is inheriting from Problem
- Has an updated constructor:
    - Calls the Problem constructor to initialize the text instance variable (**instead** of doing the assignment statement itself): Problem.__init__(self, q)
    - Does **not** remove or change the initialization of the answer instance variable
- Has an updated ask_question method that **replaces** the access of self.text with a call to the appropriate accessor in the Problem class and then appends the question mark: self.get_text() + '?'
- Removes the get_text method (since the Problem class is taking care of this for us).

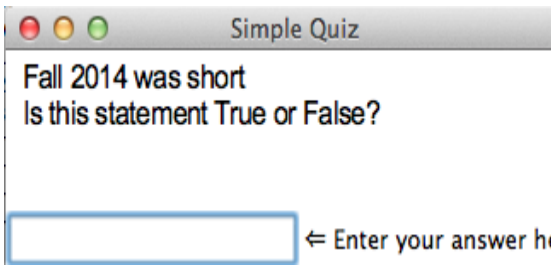
  If all has gone well, your controller should run just as it did before, and all of the ShortAnswer tests should still pass.

Python does not require that the question class definitions be placed in separate files, but it is common practice to separate more complex classes into separate files. These question classes are probably simple enough that they can be kept together, but if you choose this option, you should rename the file to “problem” to more accurately indicate what it contains.

Polymorphism

We are now ready to add more types of problems, such as true-false problems and fill-in-the-blank problems. Examples of how your application should present these questions are shown below:





The new problem classes are very similar to a ShortAnswer problem, but they each have differences:

- The FillInTheBlank class is **very** similar to the ShortAnswer class (so similar we should maybe not have a whole separate class, but this is for learning purposes, right?) with the following modification:
    - The ask_question method must append the string '\nFill in the blank.' to the end of the problem text instead of just a question mark.
- The TrueFalse class is also similar, but with a few more details to be worked out:
    - The answer is expected to be a boolean value, so the constructor should raise an exception if the calling program provides an answer that is not an instance of bool. You can check that answer is boolean using: isinstance(answer, bool).
    - The ask_question method must append the string '\nIs this statement true or false?' to the end of the problem text.
    - The get_answer method should return a string (to match the other problems): str(self.answer)
    - The check_answer method will receive a string, so we must compare a string version of the correct answer to the received answer.

Because all problems are using the same method names, the quiz will be able to create a list of Problems, and ask each problem to ask its question, check its answer, and tell us the correct answer.

Exercise 20.3

Do the following (if you haven't already):

- Add the FillInTheBlank and TrueFalse classes as described above.
- Add some unit tests to the test cases at the end of the file(s).
- Add at least one sample problem for each of the two new classes to the quiz.

Your quiz application should now operate as it did before, with randomly ordered questions of all three subtypes.

Checking In

Submit the final version of all of your files. We will grade this exercise according to the following criteria:

- **Correctness:**
  - 40% - exercise 13.2 - Add the required inheritance.
  - 40% - exercise 13.3 - Configure the polymorphic behavior.
  - 10% - New questions of each type are added
- **Understandability:**
  - 5% - Header Documentation - Document the basic purpose, authors and assignment number for each file.
  - 5% - Code Documentation - Separate the logical blocks of your program with useful comments and white space.
