Homework 4/Final Project

Due: Monday, May 8, at 11:59 PM. Submit your completed Python source files through the "Homework 4" link on Blackboard (multiple submissions are permitted; we will only grade the last/latest submission). Submissions that do not execute due to syntax or other errors will receive a 0.

For this assignment, you will write two small programs. Follow the instructions below to complete each program. Starter code and sample data files are available from the "Resources" tab on Piazza.

Part 1: QuizMaster (75 points)

QuizMaster is a simple program for administering quizzes consisting of true/false and multiple-choice questions. The questions are drawn from a (plain text) data file; each quiz consists of a randomly-selected subset of these questions.

1. Start by defining a Python class to represent a single true/false question. The `TrueFalse` class contains the following methods:

    1. An `__init__()` method that takes three user-specified parameters (in addition to `self`): a string containing the question text, a string identifying the correct answer (the single letter 'T' or 'F', not the whole word), and an integer representing the question's point value. This method assigns each of these three parameters to an appropriately-named instance variable.

    2. A `__repr__()` method that returns a string containing the question's point value, the phrase "True or false:", and the question's text. For example, this method might return a string of the form:

        (25 points) True or false: Albany is the capital of New York.

    3. A method named `points()` that does not take any arguments (other than `self`). This method returns the question's point value.

    4. A method named `score()` that (in addition to `self`) takes a single string argument representing a player's answer. If the player's answer matches the question's correct answer (use `lower()` in your comparison to account for differing capitalization), return the question's point value. Otherwise, return -1 to indicate that the player's answer was incorrect.

2. Define a second Python class to represent a single multiple-choice question. The `MultipleChoice` class is very similar to the `TrueFalse` class you just defined, and contains the following methods:

    1. An `__init__()` method that takes four user-specified parameters (in addition to `self`): a string containing the question text, a list of strings corresponding to possible answer choices, a string identifying the correct answer (a single letter between 'A' and 'Z'), and

an integer representing the question's point value. This method assigns each of these three parameters to an appropriately-named instance variable.

2. A `__repr__()` method that assembles and returns a string representing the current question. This string should begin with the question's point value and the question text. Use a loop to add the elements of the answer list to this string, appropriately labeled (lettered?), using newlines to display them on separate lines. When you are done, return the completed string. For example, the resulting string might display as:

   (15 points) Which of the following was NOT one of the Beatles?

   A) John Lennon
   B) George Harrison
   C) Mick Jagger
   D) Ringo Starr
   E) Paul McCartney

3. A method named `points()` that does not take any arguments (other than `self`). This method returns the question's point value.

4. A method named `score()` that (in addition to `self`) takes a single string argument representing a player's answer. If the player's answer matches the question's correct answer (use `lower()` in your comparison to account for differing capitalization), return the question's point value. Otherwise, return -1 to indicate that the player's answer was incorrect.

3. Complete the `loadQuestions()` function, which takes a single string parameter representing the name of a data file and stores its contents (as question objects) into a global list variable. This function should do the following:

   1. Open the specified data file and store its entire contents, line by line, in order, into a new list. Due to the way we need to process this file, this is easier than simply using a for loop to open and process the file in one shot as we have done previously. Don't forget to close the file when you are done with it!

   2. Set up a `while` loop to process the contents of our list of file data. While you have not yet reached the end of the list:

      1. Read the current line of the list and split it based on spaces. Each question record in the data file begins with a line of the form:

         question-type point-value correct-answer

         where question-type can be either "TF" or "MC".

      2. Copy the following line of the list (current line + 1) into a new variable. This string contains the question text.

3. If the question type is "TF", create a new `TrueFalse` variable using the question text, answer, and point value that you just read. Append this question to the global list of questions, and update the `while` loop's index to point to the first line of the next question.

4. If the question type is "MC" instead, read the next line of the list (current line + 2) and split it based on semicolons (';'). Multiple-choice questions take up three lines in the data file: the basic question properties, the question text, and the semicolon-separated list of answer choices. Create a new `MultipleChoice` variable using the question text, list of answer choices, correct answer, and point value. Append this question to the global list of questions, and update the `while` loop's index to point to the first line of the next question.

3. When the loop ends, print a message reporting the total number of questions that were loaded.

4. Complete the `administerQuiz()` function, which takes a single integer parameter representing the length of the quiz to give.

    1. If the requested quiz length is greater than the number of questions available, print an error message and set the quiz size to the total number of questions available (the length of your list of questions).

    2. Use `random.sample()` to randomly select the desired number of questions from the master list of questions. Then initialize the player's score and the total number of points possible to 0.

    3. For each question in the quiz, retrieve it from the list you created in the previous step. Then:

        1. Print the question number and the player's current score.

        2. Add the current question's point value to the total number of points available.

        3. Print the current question and collect the user's response.

        4. If the user answered the question correctly, print "Correct!" and update the player's score appropriately. Otherwise, print "Incorrect!".

    4. At the end of the quiz (when the loop ends), print the player's final score. Be sure to include the total number of points possible, just for comparison. For example:

        Your final score is 25 out of 100 points.

Sample Program Output

(user input is in **bold**; program output is in *italics*)

Enter the name of the quiz file to load: **question-set-1.txt**
10 question(s) loaded.
How many questions long is the quiz? **4**

Question 1 (Current score: 0 points)

(10 points) Which of the following is NOT related to genetic algorithms?

A) genome inversion
B) selection
C) point mutations
D) crossovers

Your answer: **b**
Incorrect!

Question 2 (Current score: 0 points)

(10 points) Python slicing notation uses the ___ operator

A) < >
B) ( )
C) { }
D) [ ]

Your answer: **d**
Correct!

Question 3 (Current score: 10 points)

(10 points) True or false: The largest digit in a number base N is always N-1

Your answer: **t**
Correct!

Question 4 (Current score: 20 points)

(10 points) True or false: Iteration is generally more efficient than recursion.

Your answer: **f**
Incorrect!

Your final score is 20 out of 40 points.

Part 2: Path Adjuster (25 points)

A Unix file path consists of a series of directory names, starting with a slash and separated by forward slash characters. For example, `/usr/local/bin/` has three directories (the final directory may or may not be followed by a slash).

Complete the `changePath()` function in the "paths.py" file, which takes two string arguments, representing a current path and a new "updated" path (a string of directory names) with which to update the current path, according to the following rules:

1. If the updated path begins with a forward slash, it is an *absolute path*, and should completely replace the current path.

2. A valid directory name (letters or digits, followed by a slash) should just be appended to the current path, with a single forward slash separating the two. For example, "/var/lib" plus "postfix/" would produce "/var/lib/postfix/".

3. A directory "name" of exactly "../" (two consecutive periods and a slash) means "go up one level to the preceding directory", so the last/latest directory name should be removed from the current path. The sole exception to this is if the current path is only a single slash (the *root directory*), in which case you should ignore the ".." and proceed to the next directory in the updated path. ***Be careful of removing the final (topmost) directory!***

   "/usr/local/bin/" plus "../share/man/" would produce "/usr/local/share/man/" (the ".." eliminates "bin/"). Likewise, "/var/tmp/" plus "../../.." would simply produce "/" (the first two instances of ".." remove "tmp/" and "var/", but you can't go back before the starting slash).

   You may find the string method `rfind()` helpful here; it returns the index of the final (rightmost) occurrence of a particular substring.

4. For simplicity, your code should make sure that the current path always begins with a single forward slash; if it doesn't have one, be sure to add it!

5. Forward slashes will always appear one at a time (i.e., you should never see "//" in a path).

`changePath()` returns a string representing the new file path.

Sample Test Cases

Starting Path	Adjustment	Final Path
/usr/share/tmp	foo	/usr/share/tmp/foo
/var/man/db	/etc/tmp	/etc/tmp
/this/is/a/long/path	../../different/path	/this/is/a/different/path
/foo/bar	../../baz	/baz

Grading Breakdown

This assignment is worth a total of 100 points. QuizMaster is graded on the basis of program functionality, as described below:

Point Value	Grading Criterion
15	Program works correctly with any data file (not just the sample file)
15	Program correctly displays true/false questions
15	Program correctly displays multiple-choice questions
15	Program gives appropriate feedback for player answers and updates the player's score
10	Program prints the player's final score
5	Program correctly handles "error conditions" like too few questions

Path Adjuster will be graded on the basis of the number of test cases that it solves correctly (5 test cases, worth 5 points each).