**PROJECT Object Oriented Programming with Classes 50 points**

**Objective** To write a program that utilizes a Student class and various student objects.

***PROJECT DESCRIPTION***

For this exercise you will include logic to process grades of various students from a file.

You will include a class (starter code below) that will create various student objects (a list of objects for code efficiency) that will be used to process scores from a given file. Each student score is read in from a file and cross-checked against an answer key in another file. The outcome of the class will show the processed scores for each student object by displaying each score result for each student by name.

The code module below really syncs us with all our knowledge thus far in the course, utilizing list processing, dictionaries, classes, loops, ifs and file processing. Also introduced is an array or list of class objects! This is again to be efficient whereby each object can easily be indexed thru and be used to call class methods to operate differently on the object’s data.

You will start by creating two files shown below. Make sure to include the data exactly as shown below for each respective file. Save each file to the same folder as your source program you’ll create right after this.

Create a file called **answers.txt** and add the following context EXACTLY as shown below.

**[ Contents of file: answers.txt ]**

a

b

c

d

a

b

c

d

a

a

This will represent the answer key to a particular quiz let’s say.

Next create a file called **data.txt** and add the following context EXACTLY as shown below.

**[ Contents of file: data.txt ]**

tom Schulz,a,b,c,d,a,b,c,d,a,a

Alice brown,a,b,c,d,a,b,c,d,a,b

Betty sanchez,b,b,c,d,a,b,c,d,a,a

Sammy Student,c,b,d,d,a,b,c,d,a,a

You will see how these files will be read in with the starter code that follows. The files actually will be read into lists, then compared by each students responses with the corresponding answer key responses. Study these files a bit in your head to see how things are presented and mapped out.

For example you’ll notice the **data.txt** file has the student name to begin with, followed by their particular answers to the quiz.

So the first record for instance:

tom Schulz,a,b,c,d,a,b,c,d,a,a

will have each of his answers (that follow his name), 10 in all, compared with each answer key item in the **answers.txt** file. Each record will be checked for a score and the students name and score will ultimately be put into a dictionary and displayed as sorted to the user.

Now create a .py file, lets call it **student.py** and enter the following starter code:

**class Student:**

**Scores = {}**

**# initializing the constructor method**

**def \_\_init\_\_(self, name, grade):**

**self.name = name**

**self.grade = grade**

**def getScores(self):**

**answer\_key = []**

**# read into answer\_key list, the answer key from file**

**answer\_key = [line.strip() for line in open("answers.txt", 'r')]**

**student\_answers = []**

**# read into student\_answers list, student answers from file**

**student\_answers = [line.strip().split(',')**

**for line in open("data.txt", 'r')]**

**total\_score = 100**

**# place additional code statements here for the above function**

**#---start your loop processing logic here---#**

**#---end your loop processing logic here---#**

**#---continue the class definition#**

**Student.Scores[self.getName()] = total\_score**

**def getName(self):**

**return self.name**

**@staticmethod**

**def sortDict():**

**return sorted(Student.Scores.items())**

**#---end the class definition#**

**student\_objs = [**

**Student('Sammy Student', 65),**

**Student('Betty sanchez', 45),**

**Student('Alice brown', 100),**

**Student('tom Schulz', 50),**

**]**

**for index in range(len(student\_objs)):**

**student\_objs[index].getScores()**

**sortList = Student.sortDict()**

**for k, v in sortList:**

**print(k, "has score:", v)**

Finish the processing logic marked within the commented lines

#---start your loop processing logic here---#

#---end your loop processing logic here---#

to correctly grade each student by creating a loop(s) to correctly score students using the lists created above ( namely the answer\_key list and the student\_answers list ) by matching each of the student answers to each answer key item.

Each incorrect answer deducts 10 points from the 'total\_score' variable shown

( initialized ) just above the block comments

# place additional code statements here for the above function

Study over the code a bit. Notice a few things, for instance the static method versus

various instance methods, indicated by the first word ‘self’, its parameter. A static

method is unique to the class as you can call the method without a class object just like

a static variable as you also see in the class, namely the Scores dictionary variable. Hope

you will see this *usage* as you walk thru the program code. *Note*- if you run your program at this point you should see all scores as 100.

Okay now just finish the commented section as stated within the comments in the

getScores() method. The code is pretty straight forward to follow but now needs some

logic you will provide to properly score each student within the method definition.

One thing you’ll notice about the code is the **student\_objs** list, which is actually a list of Student objects created for you, listing each member from the data.txt file (which happens to be in the order the objects are created by for simplicity). Each list object contains a student name and their last quiz grade. You will add your own name as a student and a grade for yourself (hopefully a good one!) in a bit. By including a list of objects it becomes easy to index thru each student in a loop and call the getScores() instance method to evaluate each students score, one object at a time.

Suggested pseudocode to perform loop logic commented in the getScores() method

definition is as follows:

**for loop each student from the student\_answers list**

**#check the name on the file with the object’s name**

**if student\_answers[outerIndex][…]==self.getName():**

**for loop thru your answer\_key list**

**if answer\_key[innerIndex]!=student\_answers[outerIndex][…]:**

**decrement appropriately your total\_score variable**

Notice how the pseudocode utilizes the lists, which stored their respected values from the two given files. Again think through WHAT is stored in each list as you create your loops and how data should be processed. Think of setting the right loop ranges and how to point to appropriate indices to match each list item appropriately to see if an answer is correct. That will get you through things!!!

**Figure 1-*sample* output of the four student records AFTER code modification**

**>>>**

**Alice Brown has score: 90**

**Betty Sanchez has score: 90**

**Sammy Student has score: 80**

**Tom Schulz has score: 100**

**>>>**

***Steps To Complete This Project***

**STEP 1**  **Create your files and complete the program code as stated.**

After getting things working with your code your output should be the same above as shown in Figure 1.

**STEP 2**  **Add a Record and Create a New Object**

After a successful run add in one more record to the file as well as a new student

object.

Return to your source and add another student object in, using your name and whatever grade you want to add as parameters to the constructor.

Also go back to your **data.txt** file and add your name and 10 quiz responses you desire in a similar manner as the other four records.

Also make sure to display each student with an appropriate title when displaying your results, meaning each letter in their first and last name should be capitalized if they are not.

\*\*\*Grads

Include some statistics for display after the results are printed.

Take the initial grade each student started with (as shown in each instance for each student) and then compare that grade to their new quiz grade. Display the average grade for each student, and what their **grade range** was

(**range= High score-Low score**).

Likewise display as a summary, an overall average for all student grades as well as the average range of scores overall. It’s up to you if you would like to add an additional method(s) to your class for this to make things more efficient and organized!

**STEP 3**  **Submit Your Program Code and Your Run Time Output**

Rerun your code with the newly added record scored and snapshot that for credit.

When completed, submit your program source code as well as the program output(s) and a snapshot of **data.txt** after your new record has been added in.