Student Database Challenge Problem

For this challenge problem, we will create a small database of user information. The python code we write will be able to add new data to the database, save it to a file, read it from a file, and do some basic analysis of the data. The objective of this problem is to get you comfortable with the following:

- using dictionaries to hold data
- read and write data with csv files
- iterate over data to calculate basic statistics

Once you know how to do this, you then know the commands necessary to read other csv data, do any manipulations (filtering, converting, joining) to the data you may need to do, and then write it back out to another csv file. While we can do advanced statistics in python, you may be more comfortable using a statistical program like SPSS or R.

We begin by describing the data we will be capturing. Often a database containing information about a user includes things like the user's name, gender, age, address, and more. A user will have an ID that is used to lookup the related information about the user. For example, if the ID is a number, like a student ID, then we might have the following information:

Student ID: 112233
Name: Willie Wilson
Department: CS
Gender: M
Age: 99

We can store this information in a dictionary, where the ID is the key and we use a tuple containing the rest of the information as the value in the dictionary. For example, try this:

```
studentDB = dict()
studentDB[112233] = ('Willie Wilson', 'CS', 'M', 99)
print(studentDB)
```

This should then print out this:

```
{112233: ('Willie Wilson', 'CS', 'M', 99)}
```

Now we create a function to prompt the user for this information and store it in a dictionary. Here are two suggestions for this function. First, have the dictionary (studentDB) passed in as an argument to the function. Second, to get the user input use a command like the following:

```
studentID = raw_input('What is the student ID? ')
```

Create the function in a new file. We will be adding to this as we go. An example of this function is on the next page, but try to do it without looking.

Does your function look something like this?

```
def recordStudent(studentDB):
    print("Creating new student")
    studentID = raw_input("What is the student ID? ")
    name = raw_input("Enter the student's name: ")
    gender = raw_input("Enter the student's gender: ")
    age = raw_input("Enter the student's age: ")
    studentDB[studentID] = (name,gender, age)
```

Now let's try it out. After the function, add in the following two lines and then run your script.

```
studentDB = dict()
recordStudent(studentDB)
print(studentDB)
```

Did your student information get printed?

We now have a dictionary of student information, but we need to save this information to file. There are many formats in which we can save the data, but we will save it as a csv file because it is a format that is easily read by many applications (and can even be opened in a text editor and easily read).

First, we need to import the csv package. Add the following line at the top of the file:

```
import csv
```

Now we will create a new function to save the information. It does not matter much where in the file we add this function, but let's add it to the end for now.

The function will need to open a csv file that is to be written to. The command to open a file named studentDB.csv with write permissions (meaning we want to write to the file) is the following:

```
open("studentDB.csv", "wb")
```

We will embed this command in a **with** statement. The **with** keyword provides a scope for when the file is open and will automatically close the file at the end of the block of code following the **with** statement. In the end, the **with** statement looks like this:

```
with open("studentDB.csv", "wb") as csv_file:
```

Now indent to write the next block of code that will be writing the data to the open file. The first command in this block will be creating an object that can write csv data. Creating a csv writer will result in code like this:

```
with open("studentDB.csv", "wb") as csv_file:
    writer = csv.writer(csv_file, delimiter=',')
```

We are not ready to write data from the dictionary to the file. You will need to create a **for** loop that gets the key and value from the dictionary and then for each iteration write another row to the csv file. Let's use the following command for writing an entry of the dictionary to the file:

```
writer.writerow([key] + list(value))
```

You now have all the commands you need to write this function. The function will save the student database to a csv file. It will open the file, create a csv writer, iterate over the items in the studentDB, and write each entry to the file. Give it a try, and then look at an example of this function on the next page.

```
def saveDB(studentDB):
    with open("studentDB.csv", "wb") as csv_file:
        writer = csv.writer(csv_file, delimiter=',')
        for key, value in studentDB.iteritems():
            writer.writerow([key] + list(value))
```

Great. Now let's test this out. We want to record a student into the database and then save it to a file. We will use the commands we used before to test our code to the end of the file and then add a call to the function we just wrote. The end of the file may now look something like this:

```
studentDB = dict()
recordStudent(studentDB)
print(studentDB)
saveDB(studentDB)
```

The file should now be saved. If you look at the directory where you program is saved, you should now also see another file called "studentDB.csv". Try opening this file in one of a few ways. You can open it with a basic text editor (like Notepad on Windows or TextEdit on Mac). Or you can open it in Excel, if you have that. Another quick way to look at a small text file like this is from you command line (Command Prompt on Windows or Terminal on Mac) and type the following on Windows:

```
type studentDB.csv
```

or on a Mac:

```
cat studentDB.csv
```

If we run our python program again now, we will overwrite our database file. Instead, let's start by loading the file into memory and then adding to the dictionary. Then when we write the dictionary back out to file - saving the old data and the new.

To load the database from the csv file, we will create another function. This will be similar to the one where we saved the database. The differences will be that we will open the file for reading instead of writing

```
open("studentDB.csv", "r")
```

and we will put entries into the dictionary from each row with a statement like

```
studentDB[studentID] = studentData
```

Try writing this new function, passing the studentDB in as an argument to the function. An example of the function is on the next page.

```
def loadDB(studentDB):
     with open("studentDB.csv", "r") as csv_file:
          reader = csv.reader(csv_file, delimiter=',')
          for row in reader:
               studentID = row[0]
               studentData = row[1:]
               studentDB[studentID] = studentData
```

How do we test this? Try the following code at the end of your file:

```
studentDB = dict()
loadDB(studentDB)
recordUser(studentDB)
print(studentDB)
saveDB(studentDB)
```

We are almost done with creating a database of student information. Perhaps we want to enter the information for more than one student before we save the database. We can do this by prompting the user to see if they want to add another student. If they answer 'yes' (or 'Y' or 'y') then we call `recordUser(studentDB)` again. If we create a loop where we keep asking the user if another student is to be added, we can add as many students as we want. Try modifying the code we have been using to test our program. Add a **while** loop to keep asking the user if another student is to be added and calling `recordUser(studentDB)` each time. An example of the code is on the next page.

```
studentDB = dict()
loadDB(studentDB)
another = raw_input("Enter another student? [Y/N]: ")
while (another == 'Y' or another == 'y'):
      recordUser(studentDB)
      another = raw_input("Enter another student? [Y/N]: ")
print(studentDB)
saveDB(studentDB)
```

We now can read and write the student database from and to a csv file and add new content to it. The last step in this challenge is to do some basic statistics. For example, what is the average age of the students? What portion of the students are under a certain age? How many students are in each department?

Try writing a new function to do one of these. On the next page you will find the entire program and a function for reporting what portion of the students are under a specified age.

```
import csv

def loadDB(studentDB):
	with open("studentDB.csv", "r") as csv_file:
		reader = csv.reader(csv_file, delimiter=',')
		for row in reader:
			studentID = row[0]
			studentData = row[1:]
			studentDB[studentID] = studentData

def recordStudent(studentDB):
    print("Creating new student")
    studentID = raw_input("What is the student ID? ")
    name = raw_input("Enter the student's name: ")
    gender = raw_input("Enter the student's gender: ")
    age = raw_input("Enter the student's age: ")
    studentDB[studentID] = (name,gender, age)

def saveDB(studentDB):
    with open("studentDB.csv", "wb") as csv_file:
        writer = csv.writer(csv_file, delimiter=',')
        for key, value in studentDB.iteritems():
            writer.writerow([key] + list(value))

def reportAgePortion(studentDB, age):
	count = 0.0
	for key, value in studentDB.iteritems():
		userAge = int(value[2])
		if (userAge < age):
			count = count + 1
	print("Portion of users under the age of " + str(age) + ": " + str(count /
len(studentDB)))


studentDB = dict()
loadDB(studentDB)

another = raw_input("Enter another student? [Y/N]: ")
while (another == 'Y' or another == 'y'):
	recordStudent(studentDB)
	another = raw_input("Enter another student? [Y/N]: ")
print(studentDB)

saveDB(studentDB)
reportAgePortion(studentDB, 30)
```