Problem #1: How much should I study outside of class?

Issue:

Your fellow students liked the 2nd version of study hour’s application and want to expand it again by adding the features listed below.

Study Hours per Week per Class Grade

15 A

12 B

9 C

6 D

0 F

Project Specifications:

1. The menu driven program has the following options:
2. Determine Hours to Study
3. Determine Grade
4. Quit

**ALGORITHM** Display welcome screen

1. Display splash screen
2. Present Menu options – repeat until option ‘C’ is selected:
   1. Option 1- Determine Hours to Study
   2. Option 2 – Determine Grade
   3. Option 3 – Exit program
3. If Menu option A is entered:
   1. Check for *studyHours.txt* file, appends blank *studyHours.txt* file if not found, then closes file
   2. Open a text file named *StudyHours.txt*
   3. Set *StudyHours.txt* to read mode as **studyHoursFile**
   4. Open a text file named *HowManyHours.txt*
   5. Set *HowManyHours.txt* to append mode **howManyHoursFile**
   6. Read in first line of *StudyHours.txt* file as a string variable named **userName**
   7. Create while loop statement with the condition of the **userName** != ‘’
   8. The program uses a function called validateName(**userName**) to validate the **userName**
      1. This function uses a try/except statement to validate the passed argument of **userName**
      2. The try statement will test to makes sure a valid string of alphabetic letter characters is used
      3. The except statement will test for ValueErrors, in case the line read from the *studyHours.txt* is blank
   9. The user corrects any bad data read from file.
   10. The program uses a function called processNameCase(**userName**) to process the characters of **userName** to lowercase
       1. This function uses the lower() sting modification method to return a copy string of the passed argument of **userName** with all lower cased letters
   11. The program uses a function called findFirstLast(**userName, indexPosition**) to find index position of whitespace between names of the **userName** variable and saves it as a varaiable named **indexPosition**
       1. This function uses the find() search method on the passed argument of **userName**
       2. The substring used as an argument is the whitespace in the string or ‘ ‘
       3. Every name has a space between the first name and last name, the program will find this index position and save it to the variable name **indexPosition**
   12. The program uses a function called getFirstName(**userName, indexPosition**) to create a first name variable called **firstName**
       1. This function uses the string slicing expression with the passed arguments of **userName** and **indexPosition**
       2. The **userName** variable will be the string with the range of characters to be sliced
       3. The **indexPosition** will be used in this function as the index marking the end of the slice, the start will be index 0
       4. The expression will return a string containing a copy of the characters from the start index up to the end index
       5. That string will be saved to a variable named **firstName**
   13. The program uses a function called getLastName(**userName**, **indexPosition**) to create a last name variable called **lastName**
       1. This function uses the string slicing expression with the passed arguments of **userName** and **indexPosition**
       2. The **userName** variable will be the string with the range of characters to be sliced
       3. The **indexPosition** will be used in this function as the index marking the begining of the slice, the end will be marked blank so the program will go to the end of the string
       4. The expression will return a string containing a copy of the characters from the start index up to the end index
       5. That string will be saved to a variable named **lastName**
   14. The lstrip() method will be used to strip the leading white space character left from the username input on the **lastName** variable
   15. The capitalize() method will be used to return a copy of the string with the first character from the **firstName** variable
   16. The capitalize() method will be used to return a copy of the string with the first character from the **lastName** variable
   17. The string concatenation operation will be used on the **firstName** variable ant the **lastName** variable and save the new string to a variable named **fullName**
   18. Read in second line of *StudyHours.txt* file as a variable named **creditHours**
   19. The rstrip() method will be used to strip the trailing new line character left from the file input on the **creditHours** variable
   20. The program uses a function called validateCreditHours(**creditHours**) to validate the **creditHours**
       1. This function uses a try/except statement to validate the passed argument of **creditHours**
       2. The try statement will test for a digit that is divisible by 3 and be greater than 3 and less than 18, the lowest and highest number of credits a student can take
       3. The except statement will test for ValueErrors, in case the line read from the *studyHours.txt* is blank
   21. The user corrects any bad data read from file
   22. The program uses a function called getNumofClasses(**creditHours**) to get number of classes from passed argument of **creditHours**
       1. The function uses a constant value of 3 on the variable named **CREDITSPERCLASS**
       2. The calculation divides the value of **creditHours** by **CREDITSPERCLASS**
       3. The value returned by the calculation is saved to a variable called **classes**
   23. Read in third line of *StudyHours.txt* file as a variable named **grade**
   24. The rstrip() method will be used to strip the trailing new line character left from the file input on the **grade** variable
   25. The program uses a function called validateGrade(**grade**) to validate the **grade**
       1. This function uses a try/except statement to validate the passed argument of **grade**
       2. The try statement will test for a letter grade between A and F
       3. The except statement will test for ValueErrors, in case the line read from the *studyHours.txt* is blank
   26. The user corrects any bad data read from file
   27. The program uses a function called getStudyHours(**grade**) to preform a calculation to get study hours per class from the **grade** variable
       1. Using an elif statement the function passed through the **grade** argument variable each letter grade worth different study hour amounts
       2. A = 15
       3. B = 12
       4. C = 9
       5. D = 6
       6. Else = 0
       7. The value is saved to a variable named **studyHours**
   28. The program uses a function called getTotalStudyHoursPerWeek(**classes**, **studyHours**) to perform a calculation get total study hours per week with the passed arguments of **classes** and **studyHours**
       1. The calculation multiplies the values of classes and studyHours
       2. The value returned by the calculation is saved to a variable named **TotalStudyHoursPerWeek**
   29. The program displays the output
       1. Student Name from the variable **fullName**
       2. Credit Hours from the variable **creditHours**
       3. Study Hours Recommended from the variable **TotalStudyHoursPerWeek**
       4. Desired Grade from the variable **grade**.upper()
   30. The program uses a write method on the **howManyHoursFile**
       1. The values of the displayed output is the same order in one command
   31. The program reads in the next line from the *studyHours.txt* file
       1. Reads back another **userName** and returns to step letter g
       2. If no value to read it meets the condition of the while loop
   32. Close both txt files
   33. Open the **howManyHoursFile** in read mode
       1. Read in the first line of the file saved to a temp variable
       2. Create while loop with the condtion of their be a line to read
       3. Strip the line and save the string to a list named **fileList**
       4. Read the second line of the file saved to a temp variable
       5. Strip the line and save the value to a list named **creditList**
       6. Read the third line of the file saved to a temp variable
       7. Strip the line and save the value to a list named **studyList**
       8. read the fourth line of the file save to a temp variable
       9. strip the line and save the string to a list named **gradeList**
       10. read the next line and start the loop again
   34. close the **howManyHoursFile**
   35. open *SortedHours.txt* file in write mode
   36. using a for loop with the condition of items in **fileList**
       1. use sort method on **fileList, creditList, studyList, gradeList**
       2. write the item to the *SortedHours.txt*
       3. using an iteratition count write the index item of the next three lists
       4. close the *SortedHours.txt* file
   37. Return to step #2
4. If Menu option B is entered:
   1. Check for *Grades.txt* file, appends blank *Grades.txt* file if not found, then closes file
   2. Open a text file named *Grades.txt*
   3. Set *Grades.txt* to read mode as **gradesFile**
   4. Open a text file named *HowManyHours.txt*
   5. Set *HowManyHours.txt* to append mode **howManyHoursFile**
   6. Read in first line of *Grades.txt* file as a string variable named **studentName**
   7. Create while loop statement with the condition of the **studentName** != ‘’
   8. The program uses a function called validateName(**studentName**) to validate the **studentName**
      1. This function uses a try/except statement to validate the passed argument of **studentName**
      2. The try statement will test to makes sure a valid string of alphabetic letter characters is used
      3. The except statement will test for ValueErrors, in case the line read from the *grades.txt* is blank
   9. The user corrects any bad data read from file.
   10. The program uses a function called processNameCase(**studentName**) to process the characters of **studentName** to lowercase
       1. This function uses the lower() sting modification method to return a copy string of the passed argument of **studentName** with all lower cased letters
   11. The program uses a function called findFirstLast(**studentName, indexPositionB**) to find index position of whitespace between names of the **studentName** variable and saves it as a varaiable named **indexPositionB**
       1. This function uses the find() search method on the passed argument of **studentName**
       2. The substring used as an argument is the whitespace in the string or ‘ ‘
       3. Every name has a space between the first name and last name, the program will find this index position and save it to the variable name **indexPositionB**
   12. The program uses a function called getFirstName(**studentName, indexPosition**) to create a first name variable called **firstNameB**
       1. This function uses the string slicing expression with the passed arguments of **studentName** and **indexPositionB**
       2. The **studentName** variable will be the string with the range of characters to be sliced
       3. The **indexPositionB** will be used in this function as the index marking the end of the slice, the start will be index 0
       4. The expression will return a string containing a copy of the characters from the start index up to the end index
       5. That string will be saved to a variable named **firstNameB**
   13. The program uses a function called getLastName(**studentName**, **indexPosition**) to create a last name variable called **lastName**
       1. This function uses the string slicing expression with the passed arguments of **studentName** and **indexPositionB**
       2. The **studentName** variable will be the string with the range of characters to be sliced
       3. The **indexPositionB** will be used in this function as the index marking the beginning of the slice, the end will be index will be left empty
       4. The expression will return a string containing a copy of the characters from the start index up to the end index
       5. That string will be saved to a variable named **lastNameB**
   14. The lstrip() method will be used to strip the leading white space character left from the username input on the **lastNameB** variable
   15. The capitalize() method will be used to return a copy of the string with the first character from the **firstNameB** variable
   16. The capitalize() method will be used to return a copy of the string with the first character from the **lastNameB** variable
   17. The string concatenation operation will be used on the **firstNameB** variable ant the **lastNameB** variable and save the new string to a variable named **fullNameB**
   18. Read in second line of *Grades.txt* file as a variable named **creditHoursB**
   19. The rstrip() method will be used to strip the trailing new line character left from the file input on the **creditHoursB** variable
   20. The program uses a function called validateCreditHours(**creditHoursB**) to validate the **creditHoursB**
       1. This function uses a try/except statement to validate the passed argument of **creditHoursB**
       2. The try statement will test for a digit that is divisible by 3 and be greater than 3 and less than 18, the lowest and highest number of credits a student can take
       3. The except statement will test for ValueErrors, in case the line read from the *Grades.txt* is blank
   21. The user corrects any bad data read from file
   22. The program uses a function called getNumofClasses(**creditHoursB**) to get number of classes from passed argument of **creditHoursB**
       1. The function uses a constant value of 3 on the variable named **CREDITSPERCLASS**
       2. The calculation divides the value of **creditHoursB** by **CREDITSPERCLASS**
       3. The value returned by the calculation is saved to a variable called **classesB**
   23. Read in third line of **Grades***.txt* file as a variable named **studyHoursB**
   24. The rstrip() method will be used to strip the trailing new line character left from the file input on the **studyHoursB** variable
   25. The program uses a function called validateStudyHrs(**studyHoursB**, **creditHoursB**) to validate the **studyHoursB** variable
       1. This function uses a try/except statement to validate the passed argument of **studyHoursB**
       2. The try statement test for number between 0 and 125
       3. The except statement will test for ValueErrors, in case the line read from the *Grades.txt* is blank
   26. The user corrects any bad data read from file
   27. The program uses a function called getGrade(**classesB**, **studyHoursB**) to create the gradeB variable
       1. The function gets grade from passed arguments of number of classes and expected hours of study
       2. The calculation divides the value of **studyHoursB** by **classesB**
       3. The value returned is saved to the variable named gradeB
   28. The program displays the output
       1. Student Name from the variable **fullNameB**
       2. Credit Hours from the variable **creditHoursB**
       3. Study Hours Recommended from the variable **studyHoursB**
       4. Desired Grade from the variable **gradeB**.upper()
   29. The program uses a write method on the **howManyHoursFile**
       1. The values of the displayed output is the same order in one command
   30. The program reads in the next line from the *Grades.txt* file
       1. Reads back another **studentName** and returns to step letter g
       2. If no value to read it meets the condition of the while loop
   31. Close both txt files
   32. Open the **howManyHoursFile** in read mode
       1. Read in the first line of the file saved to a temp variable
       2. Create while loop with the condtion of their be a line to read
       3. Strip the line and save the string to a list named **fileListB**
       4. Read the second line of the file saved to a temp variable
       5. Strip the line and save the value to a list named **creditListB**
       6. Read the third line of the file saved to a temp variable
       7. Strip the line and save the value to a list named **studyListB**
       8. read the fourth line of the file save to a temp variable
       9. strip the line and save the string to a list named **gradeListB**
       10. read the next line and start the loop again
   33. close the **howManyHoursFile**
   34. open *SortedHours.txt* file in write mode
   35. using a for loop with the condition of items in **fileList**
       1. use sort method on **fileListB, creditListB, studyListB, gradeListB**
       2. write the item to the *SortedHours.txt*
       3. using an iteratition count write the index item of the next three lists
       4. close the *SortedHours.txt* file
   36. Return to step #2
5. If menu option C is entered:
   1. initialize variables
   2. Check for *HowManyHours.txt* file, appends blank *HowManyHours.txt* file if not found, then closes file
   3. Open *howmanyhours.txt* text file to read
   4. Read in the first line of the file to a variable named **readName**
   5. Use a while loop with the condition that the **readName** variable is not empty
      1. Accumulate **studentCount** variable
      2. Strip **readName** variable of ‘\n’
      3. Read the second line of the file to a variable named **addToCreditTotal**
      4. Strip **addToCreditTotal** variable of ‘\n’
      5. Accumulate value of **addToCreditTotal** to the **creditTotal** variable
      6. Read the third line of the file to a variable named **addToStudyTotal**
      7. Strip the **addToStudyTotal** variable of ‘\n’
      8. Accumulate value of **addToStudyTotal** to the **creditTotal** variable
      9. Read in the fourth line of the file to a variable named **gradeLine**
      10. Strip the **gradeLine** variable of ‘\n’
      11. Read the next line of the file **readName**
   6. Use try/except statement to perform calculations on the variable made in the last step
      1. The try statement will calculate:
         1. **averageCredits** = **creditTotal** / **studentCount**
         2. **averageStudy** = **totalStudyHours** / **studentCount**
         3. The program will then display the output of the calculations and farewell message
      2. The except statement will test for ZeroDivisionError
         1. The program will display an error message
         2. The program will display the output with zero students and a farewell message

**VARIABLES/CONSTANTS**

VariableName (data type)

VariableName Value (data type) functionName

beginSequence =’y’ str welcome()

GRADES = "Grades.txt" str main()

HOWMANYHOURS = "HowManyHours.txt" str main()

menuOption =’’ str getMenuOption ()

STUDYHOURS = ’studyHours.txt’ str main()

fileList =[] list main()

creditList =[] list main()

studyList =[] list main()

gradeList =[] list main()

count =0 int main()

userName =’’ str main()

indexPosition =0 int findFirstLast ()

firstName =’’ str getFirstName ()

lastName =’’ str getLastName ()

fullName =’’ str main()

creditHours =0 int main()

classes =0 int getNumofClasses()

grade =’’ str main()

studyHours =0 int getStudyHours()

TotalStudyHoursPerWeek =0 int getTotalStudyHoursPerWeek()

GRADES =’Grades.txt’ str main()

BcreditTotal =0 int accumulateTotal()

BstudentCount =0 int accumuateCount()

BtotalStudyHours =0 int accumulateTotal()

gradesFile =’Grade.txt’ file main()

studentName =’’ str main()

indexPositionB =0 int findFirstLast ()

firstNameB =’’ str getFirstName ()

lastNameB =’’ str getLastName()

fullNameB =’’ str main()

creditHoursB =0 int main()

classesB =0 int getNumofClasses()

studyHoursB =0 int main()

gradeB =0 int getGrade()

studentCount =0 int main()

creditTotal =0 int main()

totalStudy =0 int main()

averageCredits =0 int main()

averageStudyHours =0 int main()

**FORMULAS**

classes = int(creditHours) // CREDITSPERCLASS

TotalStudyHoursPerWeek= classes \* studyHours

countVariable +=1

totalVariable += int(accVariable)

averageCredits = creditTotal / studentCount

averageStudy = totalStudyHours / studentCount

**TEST DATA EXECUTION RESULTS (up to a 15 point deduction)**

#################################### TEST 1 ##################################

Hello Users!

---------------

Thank you for taking the time to use this program.

The program was made by Jeremy Bargy.

Last update April 2020

Instructions

------------

The program being used is designed to help students identify their study needs for this semester.

With this information, students can build and develop their own study habits to meet their academic goals.

Begin program?

Please enter Y for yes

y

Select A -- Determine Hours to Study.

Select B -- Determine Grade.

Select C -- End the Program.

a

File - studyHours.txt - found...

File - HowManyHours.txt - opened...

Student Name: Barry Sanders

Credit Hours: 12

Study Hours Recommended: 48

Desired Grade: B

Student Name: Brett Farve

Credit Hours: 6

Study Hours Recommended: 18

Desired Grade: C

Error: incorrect credit hours input

---------------------------------------

Please enter the credit hours you are currently enrolled in. i.e. Must be between 1 and 6 classes. Or as required, 3 - 18 credit hours.

18

Student Name: Russel Wilson

Credit Hours: 18

Study Hours Recommended: 90

Desired Grade: A

Student Name: David Carr

Credit Hours: 3

Study Hours Recommended: 6

Desired Grade: D

Error: incorrect grade input

------------------------------------

Please enter the grade you wish to earn:

f

Student Name: Calvin Johnson

Credit Hours: 9

Study Hours Recommended: 0

Desired Grade: F

File - studyHours.txt - closed...

File - HowManyHours.txt - closed...

File -SortedHours.txt- opened...

File -SortedHours.txt- closed...

Select A -- Determine Hours to Study.

Select B -- Determine Grade.

Select C -- End the Program.

b

File - Grades.txt - found...

File - HowManyHours.txt - opened...

Error: incorrect credit hours input

---------------------------------------

Please enter the credit hours you are currently enrolled in. i.e. Must be between 1 and 6 classes. Or as required, 3 - 18 credit hours.

12

Student Name: Dez Bryant

Credit Hours: 12

Study Hours Allocated: 36

Possible Grade Earned: C

Student Name: Amari Cooper

Credit Hours: 12

Study Hours Allocated: 48

Possible Grade Earned: B

Error: incorrect credit hours input

---------------------------------------

Please enter the credit hours you are currently enrolled in. i.e. Must be between 1 and 6 classes. Or as required, 3 - 18 credit hours.

18

Error: incorrect study hours input

---------------------------------------

Please enter in a positve number for your study hours that is more than your credit hours.

Do not excced 125. You still need time to rest! :

36

Student Name: Peyton Manning

Credit Hours: 18

Study Hours Allocated: 36

Possible Grade Earned: D

Student Name: Marvin Harrison

Credit Hours: 18

Study Hours Allocated: 90

Possible Grade Earned: A

Student Name: Derrick Henry

Credit Hours: 6

Study Hours Allocated: 24

Possible Grade Earned: B

File - studyHours.txt - closed...

File - HowManyHours.txt - closed...

File -SortedHours.txt- opened...

File -SortedHours.txt- closed...

Select A -- Determine Hours to Study.

Select B -- Determine Grade.

Select C -- End the Program.

c

File - HowManyHours.txt - found...

File - HowManyHours.txt - closed...

Preforming calculations...

Calculations completed...

Program Averages:

--------------------

Total Students: 10

Average Credits: 11.4

Average Study Hours: 39.6

Thanks for using the program!

Goodbye!

>>>