Faculty Of Engineering and Technology

Electrical And Computer Engineering Department

Linux Lab

ENCS3130

Python Project Report

Students:

Amal Ziad_1192141

Zainab Shaabneh_1182820

Instructors:

Mohammad Jubran

Khader Mohammad

Sections: 1 & 2

Date:

9/1/2021

Abstract:

The aim of this project is to be more familiar with python programming by building a program that does some operations on college student's manager and gives many choices for both student and admin.

Contents

Abstract: 2
Over View: 4
Main Menu: 4
Admin's options: 5
- Adding a new student: 6
- Adding a new semester with courses and grades for a student: 7
- Updating data regarding a specific student: 8
- Getting student's statistics: 10
- Getting global statistics: 12
- Searching: 13

Student's options: 16
- Getting student's statistics: 17
- Getting global statistics: 19

Full Code: 20
Conclusion: 32

Over View:

We worked together on a program called “Spyder” that uses python programming and started developing a program that satisfies student’s record system.

Main Menu:

First the program asks type of user as shown below (fig.1.1):

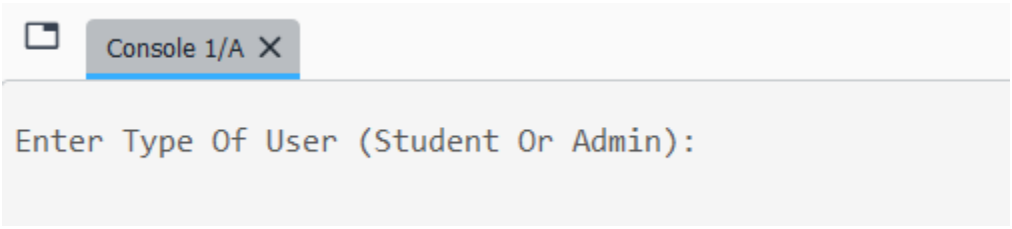

Fig.1.1

If you chooses admin then this is your menu (fig.1.2):

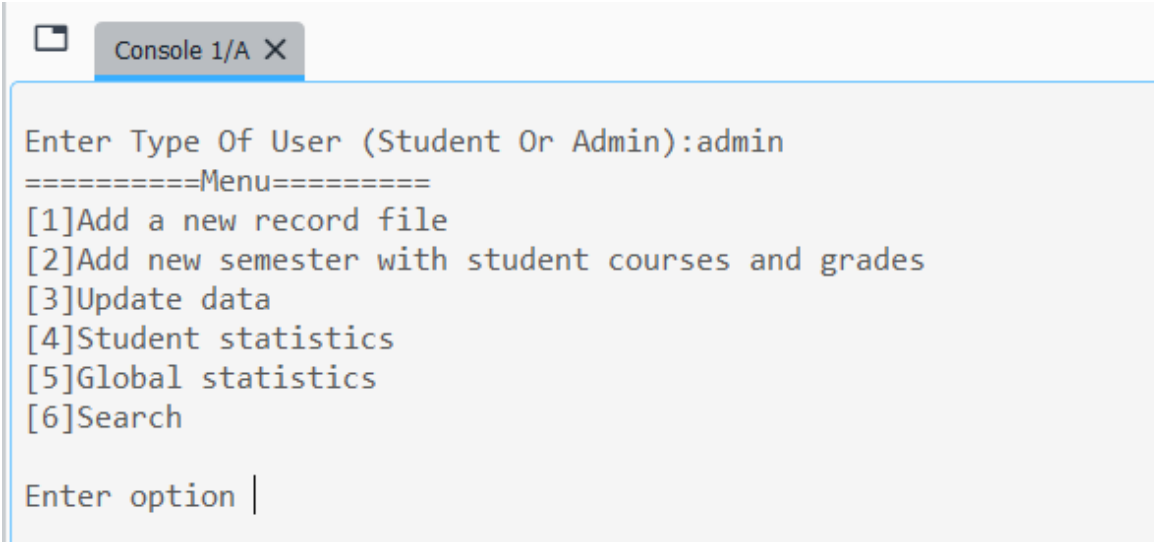

Fig.1.2

- ✓ Note: ignoring case for the input

And if you chooses student then this is your menu (fig.1.3):

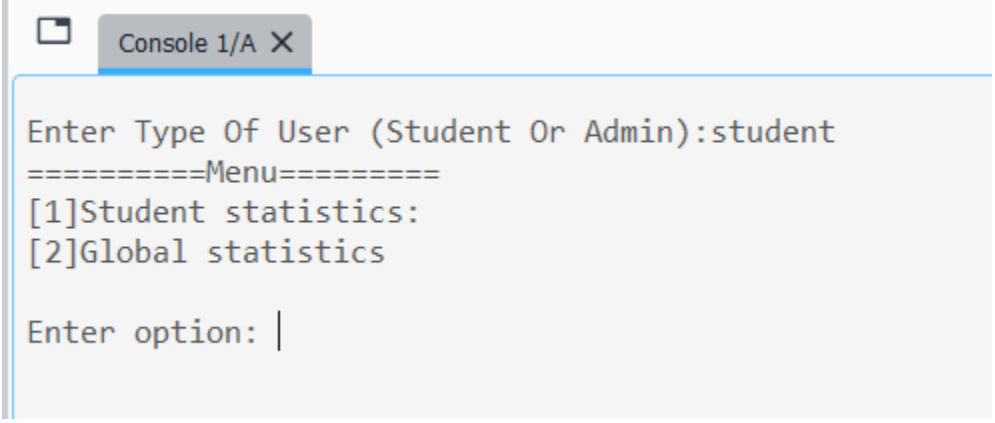

Fig.1.3

Admin's options:

After choosing admin's menu which have the following options (fig.2):

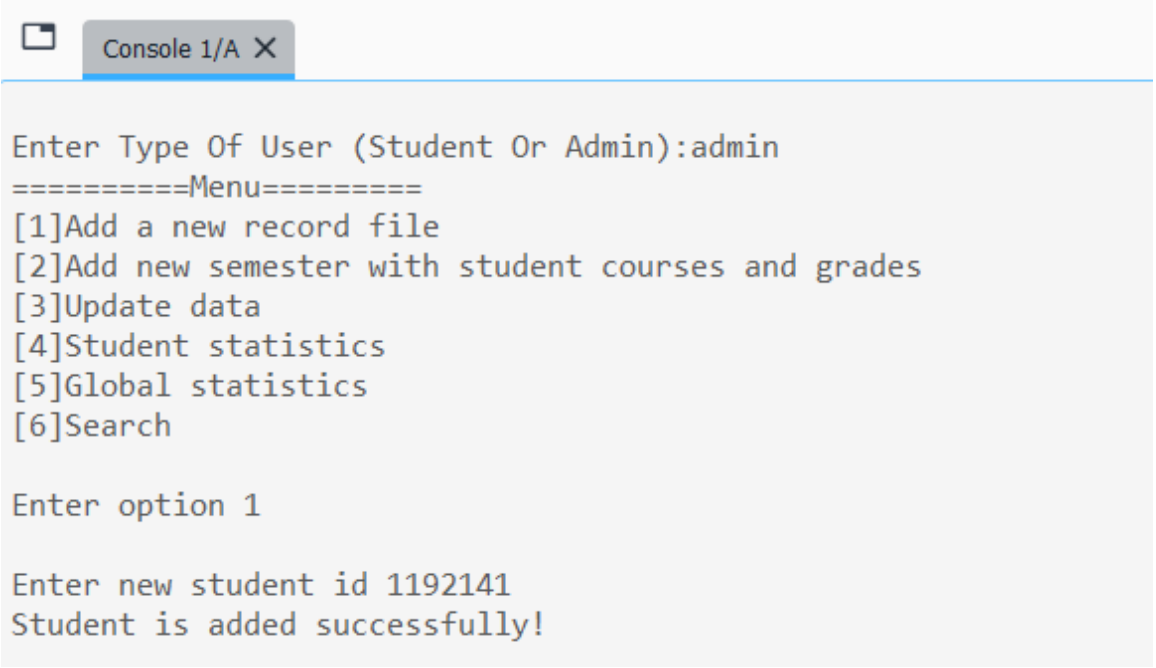

Fig.2

- Adding a new student:

The full code for Adding method in class admin is shown in fig.2.1.1.

```
1   import os
2   import glob
3   import os.path
4
5
6   class admin:
7       def __init__(self):#constructor
8           self
9
10      def Addfile(self):  # adding a new record method
11          st_id = (input("Enter new student id") + ".txt")
12          try:
13              save_path = 'C:/Users/Administrator/Desktop/records'
14              completeName = os.path.join(save_path,st_id)
15              f = open(completeName, 'w')
16              print("Student is added successfully!")
17              f.close()
18          except FileExistsError:
19              print("Error: student id is already exists")
```

Fig.2.1.1

Discussion:

`Addfile()` method, first takes argument of type `self` then asks user to enter the new student id then the program will check if file exists or not by `try` and `except`. If the file does not exist then adding a new file will works successfully.

Running example:

```
Enter Type Of User (Student Or Admin):admin
==========Menu==========
[1]Add a new record file
[2]Add new semester with student courses and grades
[3]Update data
[4]Student statistics
[5]Global statistics
[6]Search

Enter option 1

Enter new student id1234
Student is added successfully!
```

IPython Console History

LSP Python: ready custom (Python 3.7.9) Line 7, Col 36 UTF-8 CRLF RW

10°C בהיר ENG 7:08 PM 1/5/2022

Fig.2.1.2

Discussion:

As we see the file added successfully in fig.2.1.2 and the file showed up in the records file where all students' files are in it as shown in fig.2.1.3.

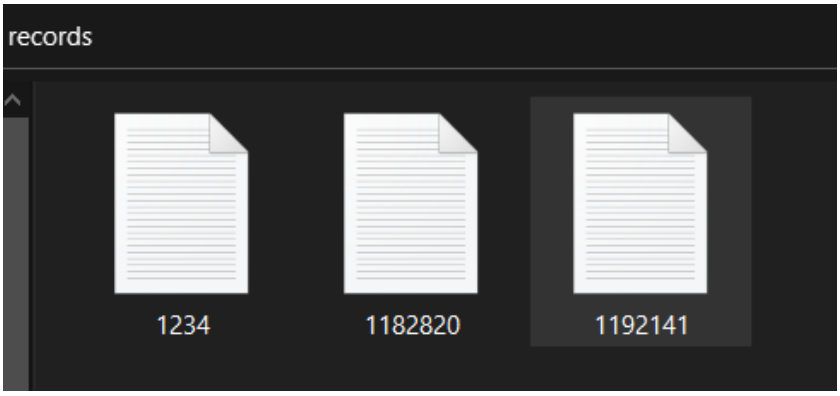

Fig.2.1.3

- Adding a new semester with courses and grades for a student:

The full code for Adding new Semester method in class admin is shown in fig.2.2.1.

```
21      def AddSemester(self):  # adding new semester method
22          st_id = (input("Enter student id") + ".txt")
23          save_path = 'C:/Users/Administrator/Desktop/records'
24          completeName = os.path.join(save_path, st_id)
25          with open(completeName, "a+") as f:
26              st_sem = input("Enter the new semester year")
27              for line in f:
28                  tokens = line.split(";")  # because ';' is seperating the data in the same line
29                  st_sem.strip(" ")  # for easier comparing
30                  tokens[0].strip(" ")  # also for easier comparing and tokens[0] is always for semester year
31                  if (st_sem == tokens[0]):
32                      print("Error: semester is already here")
33                      quit()  # quitting the program
34              status = True
35              new = []
36              while (status == True):  # it won't reach this point unless the semester is not exist in the file
37
38                  course = input(
39                          "enter course and grade and seperate them with ':' ,Note:'if you are done enter 'done''")
40                  if (course.lower() == 'done'):
41                      status = False
42                  else:
43                      course.upper()
44                      new.append(course)
45              text = st_sem + ";"
46              for i in range(0, len(new)):
47                  text = text + new[i] + ";"
48              text=text+"\n" #to write in a new line
49              f.write(text)
50              print("Addition completed successfully!")
51              f.close
```

Fig.2.2.1

Discussion:

When user chooses AddSemester option, the program first asks him for student id and opens the file for it, then asks for the new semester and checks if the semester is already in the file or not by getting each line and split “;” because I made it to separate each information from the other. If the semester is not exists then the program accept to add it and ask user for course name and its grade separated by “:” for easier work for me and enter done if the work is done then group inputs in text and append it to the wanted file.

Running example:

```
Enter Type Of User (Student Or Admin):admin
==========Menu=========
[1]Add a new record file
[2]Add new semester with student courses and grades
[3]Update data
[4]Student statistics
[5]Global statistics
[6]Search

Enter option 2

Enter student id1192141

Enter the new semester year2019/2020

enter course and grade and seperate them with ':' ,Note:'if you are done enter 'done''ENCS3310:90

enter course and grade and seperate them with ':' ,Note:'if you are done enter 'done''ENCS3320:88

enter course and grade and seperate them with ':' ,Note:'if you are done enter 'done''ENCS3330:86

enter course and grade and seperate them with ':' ,Note:'if you are done enter 'done''done
Addition completed successfully!
```

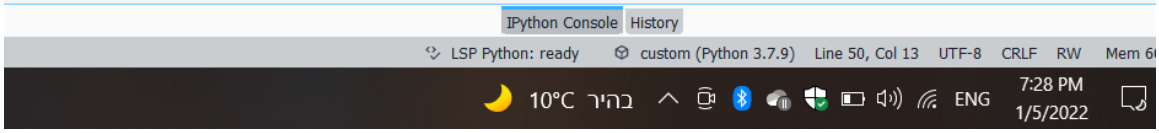

Fig.2.2.2

Discussion:

As shown in fig.2.2.2, the work is done correctly and the wanted result is shown in fig.2.2.3.

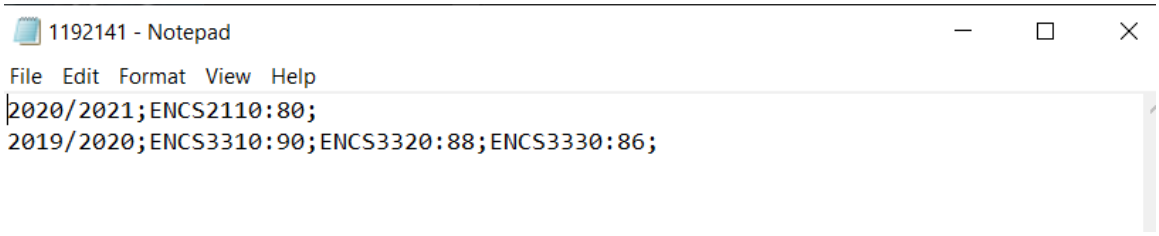

Fig.2.2.3

- Updating data regarding a specific student:

The full code for Updating method in class admin is shown in fig.2.3.1.

```
53      def Update(self):  # update data method
54          st_id = (input("Enter student id ") + ".txt")
55          save_path = 'C:/Users/Administrator/Desktop/records'
56          completeName = os.path.join(save_path, st_id)
57          with open(completeName, 'r+') as f:
58              st_old = input("Enter course name you want to update ")
59              for line in f:
60                  data = line
61                  tokens = data.split(";")
62                  for i in range(1, len(tokens)):  # because tokens[0] is for semester year (it is not important here)
63                      temp1 = st_old.strip(" ").upper()  # for easier comparing
64                      tokens2 = tokens[i].split(":")
65                      temp2 = tokens2[0].strip(" ").upper()  # also for easier comparing
66                      if (temp1 == temp2):
67                          new = input("Enter new grade: ")
68                          f.write(data.replace(tokens2[1],new))
69                          print("Update is done!")
70
71              f.close()
```

Fig.2.3.1

Discussion:

The program first as usual asks for student id and opens the file then asks user for course name then the program checks in each line if the course is exists, if yes then asks user for the new grade and save changes to the file. Note: if the course is not included in the file then the program won't do anything and won't ask user for the new grade.

Running example:

```
Enter Type Of User (Student Or Admin):admin
==========Menu=========
[1]Add a new record file
[2]Add new semester with student courses and grades
[3]Update data
[4]Student statistics
[5]Global statistics
[6]Search

Enter option 3

Enter student id1192141

Enter course name you want to updateENCS3310

Enter new grade:99
Update is done!
```

IPython Console History

LSP Python: ready custom (Python 3.7.9) Line 67, Col 55 UTF-8 CRLF RW M

10°C בהיר ENG 8:04 PM 1/5/2022

Fig.2.3.2

Discussion:

As we see the program worked successfully and the result is shown in fig.2.3.3.

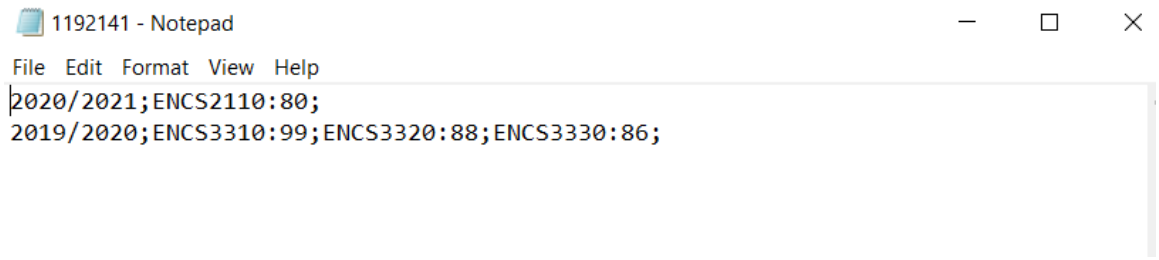

Fig.2.3.3

- Getting student's statistics:

The full code for getting student's statistics method in class admin is shown in fig.2.4.1 and fig.2.4.2.

```
73      def statistics(self):
74          st_id = (input("Enter student id") + ".txt")
75          save_path = 'C:/Users/Administrator/Desktop/records'
76          completeName = os.path.join(save_path, st_id)
77          courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3), "ENCS3130": int(1),
78                     "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3), "ENCS3340": int(3),
79                     "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1), "ENCS4210": int(2),
80                     "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3), "ENCS4330": int(3),
81                     "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1), "ENCS5150": int(1),
82                     "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1), "ENEE2304": int(3),
83                     "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3), "ENEE3309": int(3),
84                     "ENEE4113": int(1)
85                     }
86          allhours = int(69)
87          with open(completeName, 'r') as f:
88              hourstaken =int(0)
89              sum = 0
90              counter = 0
91              remaining = ["ENCS2110", "ENCS2340", "ENCS2380", "ENCS3130",
92                           "ENCS3310", "ENCS3320", "ENCS3330", "ENCS3340",
93                           "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",
94                           "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",
95                           "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",
96                           "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",
97                           "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",
98                           "ENEE4113"]
99              temp=["ENCS2110", "ENCS2340", "ENCS2380", "ENCS3130",
100                          "ENCS3310", "ENCS3320", "ENCS3330", "ENCS3340",
101                          "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",
102                          "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",
103                          "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",
104                          "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",
105                          "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",
106                          "ENEE4113"]
```

Fig.2.4.1

```
            for line in f:
                tokens = line.split(";")
                localsum = 0
                localcounter = 0
                for i in range(1, len(tokens)):
                    token2 = tokens[i].split(":")
                    token2[0]=str(token2[0])
                    for x in range(0,len(remaining)):
                        if(token2[0] == remaining[x]):
                            temp.remove(remaining[x])
                    num=courses.get(token2[0])
                    hourstaken = hourstaken + num # to get hours taken
                    localsum = localsum + float(token2[1])
                    localcounter = localcounter + 1
                    sum = sum + float(token2[1])
                    counter = counter + 1
                localaverage = localsum / localcounter
                print("Average for semester " + tokens[0] + " is " + localaverage)

            average = sum / counter
            print("Avargae all times is " + average)
            print("Finished hours=" + (allhours - hourstaken))
            print("Remaining courses are:" + temp)
```

Fig.2.4.2

Discussion:

The method first takes id number then opens the file on reading mode then we defined courses and their hours in a dictionary and remaining array for the same courses and temp. Then the program read each line and split it by “;” because this what separates data in files. After entering for loop to get the remaining courses and how many hours it takes and finally calculating the average for each semester and all time then get all remaining courses.

Running Example:

```
Enter Type Of User (Student Or Admin):admin
==========Menu==========
[1]Add a new record file
[2]Add new semester with student courses and grades
[3]Update data
[4]Student statistics
[5]Global statistics
[6]Search

Enter option 4

Enter student id1192141
Average for semester 2020/2021 is 80
Average for semester 2019/2020 is 91
Avargae all times is 88.25
Finished hours=10
Remaining courses are:["ENCS2340", "ENCS2380", "ENCS3130","ENCS3340",
                       "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",
                       "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",
                       "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",
                       "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",
                       "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",
                       "ENEE4113"]
```

Fig.2.4.3

Discussion:

As we see the program worked successfully and gave as the results from the file in fig.2.4.4.

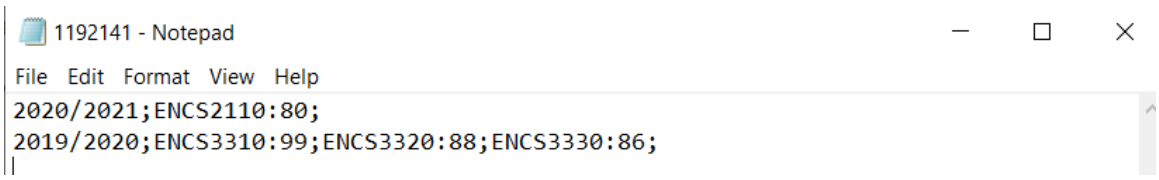

Fig.2.4.4

- Getting global statistics:

The full code for getting global statistics method in class admin is shown in fig.2.5.1 and fig.2.5.2.

```
131     def Global(self):
132         counter = 0  # to get students' number
133         sum = 0
134         courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3), "ENCS3130": int(1),
135                    "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3), "ENCS3340": int(3),
136                    "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1), "ENCS4210": int(2),
137                    "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3), "ENCS4330": int(3),
138                    "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1), "ENCS5150": int(1),
139                    "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1), "ENEE2304": int(3),
140                    "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3), "ENEE3309": int(3),
141                    "ENEE4113": int(1)
142                    }
143
144         semSum = {}  # summation of all hours in semesters
145         for filename in glob.glob('*.txt'):
146             counter = counter + 1
147             with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
148                 hourstaken = 0
149                 localsum = 0
150                 localcounter = 0  # to get average for a student in all times
151                 for line in f:
152                     tokens = line.split(";")
153
154                     for i in range(1, len(tokens)):  # because 0 is for semester year
155                         token2 = tokens[i].split(":")
156                         token2[0].strip(" ").upper()
157                         hourstaken = int(hourstaken + courses.get(token2[0]))  # to get hours taken
158                         localsum = localsum + float(token2[1])
159                         localcounter = localcounter + 1
160
161                     if (semSum.has_key(tokens[0])):
162                         temp = semSum.get(tokens[0]).split("#")#because in semSum={sem year: summation of hours#number of hours , ...}
163                         semSum[tokens[0]] = int(temp[0] + hourstaken) + "#" + int(
164                             temp[1] + 1)  # if semester year is in semSum then add taken hours to its value
165                     else:
166                         semSum[tokens[0]] = int(hourstaken) +"#"+ int(1)  # if not then add it and put the value of taken hours
167
```

Fig.2.5.1

```
167
168                     localaverage = localsum / localcounter
169                     sum = sum + localaverage
170                 average = sum / counter
171                 print("Overall students average=" + average)
172                 x = 0
173                 for x in range(0, len(semSum)):
174                     sem_list = list(semSum.keys())
175                     value_list = list(semSum.values())
176                     temp2 = value_list[x].split("#")
177                     avgHours = temp2[0] / temp2[1]
178                     print("Average hours for semester " + sem_list[x] + " is=" + avgHours)
179         return "Operation done successfully!"
180
```

Fig.2.5.2

Discussion:

This method opens all files in the records directory and then calculates overall average by taking sum of all students' averages and number of them. In semSum dictionary to calculate average hours per semester and that by taking summation of hours in that semester and number of students that took in that semester and this done by the following: `semSum={sem year: summation of hours#number of students , ...}` . Finally, print al this data.

- ## Searching:

The full code for searching methods are shown in fig.2.6.1, fig.2.6.2, fig,2.6.3, fig.2.6.4, and fig.2.6.5.

In admin options:

```
    elif (option == '6'):
        print("You can search for these:")
        print("[1]Based on average")
        print("[2]Based on taken hours")
        option2 = input("Enter option: ")
        if (option2 == '1'):
            print("[1]Averages greater than .....")
            print("[2]Averages less than ........")
            print("[3]Average equals ............")
            ad6 = admin()
            print(ad6.SearchAvg())
        elif (option2 == '2'):
            print("[1]Taken hours greater than ......")
            print("[2]Taken hours less than .........")
            print("[3]Taken hours equals ............")
            ad7 = admin()
            print(ad7.SearchH())
```

Fig.2.6.1

Discussion:

After the user chooses based on average or hours then choose one of the options above, the program calls SearchAvg or SearchH from admin class.

In admin class, SearchAvg method:

```
def SearchAvg(self):
    op= input("Enter option: ")
    if (op == '1'):
        avg = input("Enter average: ")
        result = []
        for filename in glob.glob('*.txt'):
            with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
                sum = 0
                counter = 0
                for line in f:
                    tokens = line.split(";")
                    for i in range(1, len(tokens)):  # because 0 is for semester year
                        token2 = tokens[i].split(":")
                        sum = sum + float(token2[1])
                        counter = counter + 1
                    average = sum / counter
                    if (average > avg):
                        result.append(f.name + ", average=" + average)
                        print("Students that have average greater than " + avg + " are:" + result)
    elif (op == '2'):
        avg = input("Enter average: ")
        result = []
        for filename in glob.glob('*.txt'):
            with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
                sum = 0
                counter = 0
                for line in f:
                    tokens = line.split(";")
                    for i in range(1, len(tokens)):  # because 0 is for semester year
                        token2 = tokens[i].split(":")
                        sum = sum + float(token2[1])
                        counter = counter + 1
                    average = sum / counter
                    if (average < avg):
                        result.append(f.name + ", average=" + average)
                print("Students that have average less than " + avg + " are:" + result)
```

Fig.2.6.2

```
        elif (op == '3'):
            avg = input("Enter average: ")
            result = []
            for filename in glob.glob('*.txt'):
                with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
                    sum = 0
                    counter = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):  # because 0 is for semester year
                            token2 = tokens[i].split(":")
                            sum = sum + float(token2[1])
                            counter = counter + 1
                        average = sum / counter
                        if (average == avg):
                            result.append(f.name + ", average=" + average)
            print("Students that have the average " + avg + " are:" + result)
        else:
            print("Error: wrong option")
```

Fig.2.6.3

Discussion:

In this method after the user chooses his option, the program opens all files in records directory and calculates each students' averages and check if it's less, equals, or more that the entered average and if yes then adds file name (which is student id) to result list and display it to the admin.

In SearchH method:

```
    def SearchH():
        op= input("Enter option: ")
        courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3), "ENCS3130": int(1),
                   "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3), "ENCS3340": int(3),
                   "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1), "ENCS4210": int(2),
                   "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3), "ENCS4330": int(3),
                   "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1), "ENCS5150": int(1),
                   "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1), "ENEE2304": int(3),
                   "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3), "ENEE3309": int(3),
                   "ENEE4113": int(1)
                   }

        if (op == '1'):
            hour = input("Enter hours: ")
            result = []
            for filename in glob.glob('*.txt'):
                with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
                    hourstaken = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):
                            tokens2 = tokens[i].split(":")
                            tokens2[0].strip(" ").upper()
                            hourstaken = int(hourstaken + courses.get(tokens2[0]))
                        if (hourstaken > hour):
                            result.append(f.name + ", taken hours=" + hourstaken)
                    print("Students that have taken hours more than " + hour + " are:" + result)
```

Fig.2.6.4

```
        elif (op == '2'):
            hour = input("Enter hours: ")
            result = []
            for filename in glob.glob('*.txt'):
                with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
                    hourstaken = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):
                            tokens2 = tokens[i].split(":")
                            tokens2[0].strip(" ").upper()
                            hourstaken = int(hourstaken + courses.get(tokens2[0]))
                        if (hourstaken < hour):
                            result.append(f.name + ", taken hours=" + hourstaken)
                    print("Students that have taken hours less than " + hour + " are:" + result)
        elif (op == '3'):
            hour = input("Enter hours: ")
            result = []
            for filename in glob.glob('*.txt'):
                with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
                    hourstaken = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):
                            tokens2 = tokens[i].split(":")
                            tokens2[0].strip(" ").upper()
                            hourstaken = int(hourstaken + courses.get(tokens2[0]))
                        if (hourstaken == hour):
                            result.append(f.name + ", taken hours=" + hourstaken)
                    print("Students that have taken hours equals " + hour + " are:" + result)
        else:
            print("Error: wrong option")
```

Fig.2.6.5

Discussion:

The as in searching depends on hours taken, after choosing option and entering a specific hours by user, the program calculates hours taken for each student and then checks if this student is less, equals, or more that the entered, if yes then add file name to result list and display it.

Student's options:

After choosing student's menu which have the following options (fig.3.1):

```
Enter Type Of User (Student Or Admin):student
==========Menu==========
[1]Student statistics:
[2]Global statistics

Enter option: 1
```

Fig.3.1

And the code for student's options (fig.3.2).

```
352  elif (user.lower() == "student"):
353      print("==========Menu=========")
354      print("[1]Student statistics:")
355      print("[2]Global statistics")
356      option = input("Enter option: ")
357      if (option == '1'):
358          st1 = student()
359          st1.statistics()
360      elif (option == '2'):
361          print("Here's the global statistics:")
362          st2 = student()
363          st2.Global
364      else:
365          print("Error: wrong option")
```

Fig.3.2

And the code for student class shown in fig.3.3.

```
298  class student(admin):
299
300      def __init__(self):
301          self
302
303
```

Fig.3.3

Discussion:

After the user chooses his option, the program creates object of type student then calls its method in student class.

- Getting student's statistics:

The full code for getting student's statistics method in class student that is inherited from class admin is shown in fig.3.1.1 and fig.3.1.1.

```
73      def statistics(self):
74          st_id = (input("Enter student id") + ".txt")
75          save_path = 'C:/Users/Administrator/Desktop/records'
76          completeName = os.path.join(save_path, st_id)
77          courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3), "ENCS3130": int(1),
78                     "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3), "ENCS3340": int(3),
79                     "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1), "ENCS4210": int(2),
80                     "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3), "ENCS4330": int(3),
81                     "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1), "ENCS5150": int(1),
82                     "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1), "ENEE2304": int(3),
83                     "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3), "ENEE3309": int(3),
84                     "ENEE4113": int(1)
85                     }
86          allhours = int(69)
87          with open(completeName, 'r') as f:
88              hourstaken =int(0)
89              sum = 0
90              counter = 0
91              remaining = ["ENCS2110", "ENCS2340", "ENCS2380", "ENCS3130",
92                           "ENCS3310", "ENCS3320", "ENCS3330", "ENCS3340",
93                           "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",
94                           "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",
95                           "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",
96                           "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",
97                           "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",
98                           "ENEE4113"]
99              temp=["ENCS2110", "ENCS2340", "ENCS2380", "ENCS3130",
100                          "ENCS3310", "ENCS3320", "ENCS3330", "ENCS3340",
101                          "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",
102                          "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",
103                          "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",
104                          "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",
105                          "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",
106                          "ENEE4113"]
```

Fig.3.1.1

```
107             for line in f:
108                 tokens = line.split(";")
109                 localsum = 0
110                 localcounter = 0
111                 for i in range(1, len(tokens)):
112                     token2 = tokens[i].split(":")
113                     token2[0]=str(token2[0])
114                     for x in range(0,len(remaining)):
115                         if(token2[0] == remaining[x]):
116                             temp.remove(remaining[x])
117                     num=courses.get(token2[0])
118                     hourstaken = hourstaken + num # to get hours taken
119                     localsum = localsum + float(token2[1])
120                     localcounter = localcounter + 1
121                     sum = sum + float(token2[1])
122                     counter = counter + 1
123                 localaverage = localsum / localcounter
124                 print("Average for semester " + tokens[0] + " is " + localaverage)
125
126             average = sum / counter
127             print("Avargae all times is " + average)
128             print("Finished hours=" + (allhours - hourstaken))
129             print("Remaining courses are:" + temp)
```

Fig.3.1.2

Discussion:

The method first takes id number then opens the file on reading mode then we defined courses and their hours in a dictionary and remaining array for the same courses and temp. Then the program read each line and split it by “;” because this what separates data

in files. After entering for loop to get the remaining courses and how many hours it takes and finally calculating the average for each semester and all time then get all remaining courses.

Running Example:

```
Enter Type Of User (Student Or Admin):student
==========Menu==========
[1]Student statistics:
[2]Global statistics

Enter option: 1

Enter student id1192141
Average for semester 2020/2021 is 80
Average for semester 2019/2020 is 91
Avargae all times is 88.25
Finished hours=10
Remaining courses are:["ENCS2340", "ENCS2380", "ENCS3130","ENCS3340",
                       "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",
                       "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",
                       "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",
                       "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",
                       "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",
                       "ENEE4113"]
```

Fig.3.1.3

Discussion:

As we see the program worked successfully and gave as the results from the file in fig.3.1.4.

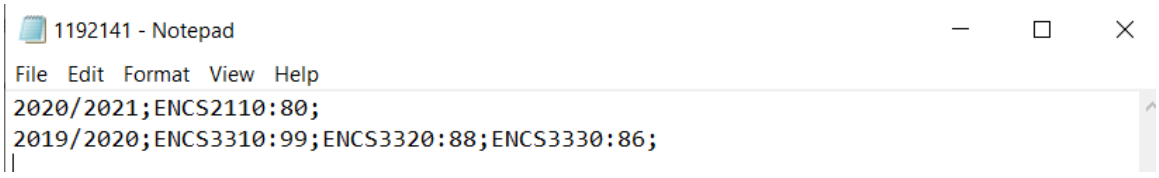

Fig.3.1.4

- Getting global statistics:

The full code for getting global statistics method in class student that is inherited i=from class admin is shown in fig.3.2.1 and fig.3.2.2.

```
131     def Global(self):
132         counter = 0  # to get students' number
133         sum = 0
134         courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3), "ENCS3130": int(1),
135                    "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3), "ENCS3340": int(3),
136                    "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1), "ENCS4210": int(2),
137                    "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3), "ENCS4330": int(3),
138                    "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1), "ENCS5150": int(1),
139                    "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1), "ENEE2304": int(3),
140                    "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3), "ENEE3309": int(3),
141                    "ENEE4113": int(1)
142                    }
143
144         semSum = {}  # summation of all hours in semesters
145         for filename in glob.glob('*.txt'):
146             counter = counter + 1
147             with open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'), filename), 'r') as f:
148                 hourstaken = 0
149                 localsum = 0
150                 localcounter = 0  # to get average for a student in all times
151                 for line in f:
152                     tokens = line.split(";")
153
154                     for i in range(1, len(tokens)):  # because 0 is for semester year
155                         token2 = tokens[i].split(":")
156                         token2[0].strip(" ").upper()
157                         hourstaken = int(hourstaken + courses.get(token2[0]))  # to get hours taken
158                         localsum = localsum + float(token2[1])
159                         localcounter = localcounter + 1
160
161                     if (semSum.has_key(tokens[0])):
162                         temp = semSum.get(tokens[0]).split("#")#because in semSum={sem year: summation of hours#number of hours , ...}
163                         semSum[tokens[0]] = int(temp[0] + hourstaken) + "#" + int(
164                             temp[1] + 1)  # if semester year is in semSum then add taken hours to its value
165                     else:
166                         semSum[tokens[0]] = int(hourstaken) +"#"+ int(1)  # if not then add it and put the value of taken hours
167
```

Fig.3.2.1

```
167
168                     localaverage = localsum / localcounter
169                     sum = sum + localaverage
170                 average = sum / counter
171                 print("Overall students average=" + average)
172                 x = 0
173                 for x in range(0, len(semSum)):
174                     sem_list = list(semSum.keys())
175                     value_list = list(semSum.values())
176                     temp2 = value_list[x].split("#")
177                     avgHours = temp2[0] / temp2[1]
178                     print("Average hours for semester " + sem_list[x] + " is=" + avgHours)
179         return "Operation done successfully!"
180
```

Fig.3.2.2

Discussion:

This method opens all files in the records directory and then calculates overall average by taking sum of all students' averages and number of them. In semSum dictionary to calculate average hours per semester and that by taking summation of hours in that semester and number of students that took in that semester and this done by the following: `semSum={sem year: summation of hours#number of students , ...}` . Finally, print al this data.

Full Code:

```
import os

import glob

import os.path

class admin:
```

```
    def __init__(self):#constructor
        self


    def Addfile(self):  # adding a new record method
        st_id = (input("Enter new student id") + ".txt")
        try:
            save_path = 'C:/Users/Administrator/Desktop/records'
            completeName = os.path.join(save_path,st_id)
            f = open(completeName, 'w')
            print("Student is added successfully!")
            f.close()
        except FileExistsError:
            print("Error: student id is already exists")
    def AddSemester(self):  # adding new semester method
        st_id = (input("Enter student id") + ".txt")
        save_path = 'C:/Users/Administrator/Desktop/records'
        completeName = os.path.join(save_path, st_id)
        with open(completeName, "a+") as f:
            st_sem = input("Enter the new semester year")
            for line in f:
                tokens = line.split(";")  # because ';' is seperating the data
in the same line
                st_sem.strip(" ")  # for easier comparing
                tokens[0].strip(" ")  # also for easier comparing and tokens[0]
is always for semester year
                if (st_sem == tokens[0]):
                    print("Error: semester is already here")
                    quit()  # quitting the program
            status = True
            new = []
            while (status == True):  # it won't reach this point unless the
semester is not exist in the file
                course = input(
                        "enter course and grade and seperate them with ':'
,Note:'if you are done enter 'done''")
                if (course.lower() == 'done'):
```

```
                    status = False
                else:
                    course.upper()
                    new.append(course)
            text = st_sem + ";"
            for i in range(0, len(new)):
                text = text + new[i] + ";"
            text=text+"\n" #to write in a new line
            f.write(text)
            print("Addition completed successfully!")
            f.close


    def Update(self):  # update data method
        st_id = (input("Enter student id ") + ".txt")
        save_path = 'C:/Users/Administrator/Desktop/records'
        completeName = os.path.join(save_path, st_id)
        with open(completeName, 'r+') as f:
            st_old = input("Enter course name you want to update ")
            for line in f:
                data = line
                tokens = data.split(";")
                for i in range(1, len(tokens)):  # because tokens[0] is for
semester year (it is not important here)
                    temp1 = st_old.strip(" ").upper()  # for easier comparing
                    tokens2 = tokens[i].split(":")
                    temp2 = tokens2[0].strip(" ").upper()  # also for easier
comparing
                    if (temp1 == temp2):
                        new = input("Enter new grade: ")
                        f.write(data.replace(tokens2[1],new))
                        print("Update is done!")
            f.close()
    def statistics(self):
        st_id = (input("Enter student id") + ".txt")
        save_path = 'C:/Users/Administrator/Desktop/records'
```

```
        completeName = os.path.join(save_path, st_id)

        courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3),
"ENCS3130": int(1),

                   "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3),
"ENCS3340": int(3),

                   "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1),
"ENCS4210": int(2),

                   "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3),
"ENCS4330": int(3),

                   "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1),
"ENCS5150": int(1),

                   "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1),
"ENEE2304": int(3),

                   "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3),
"ENEE3309": int(3),

                   "ENEE4113": int(1)

                   }

        allhours = int(69)

        with open(completeName, 'r') as f:

            hourstaken =int(0)

            sum = 0

            counter = 0

            remaining = ["ENCS2110", "ENCS2340", "ENCS2380", "ENCS3130",

                         "ENCS3310", "ENCS3320", "ENCS3330", "ENCS3340",

                         "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",

                         "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",

                         "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",

                         "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",

                         "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",

                         "ENEE4113"]

            temp=["ENCS2110", "ENCS2340", "ENCS2380", "ENCS3130",

                         "ENCS3310", "ENCS3320", "ENCS3330", "ENCS3340",

                         "ENCS3390", "ENCS4110", "ENCS4130", "ENCS4210",

                         "ENCS4300", "ENCS4310", "ENCS4320", "ENCS4330",

                         "ENCS4370", "ENCS4380", "ENCS5140", "ENCS5150",

                         "ENCS5200", "ENCS5300", "ENEE2103", "ENEE2304",

                         "ENEE2307", "ENEE2312", "ENEE2360", "ENEE3309",
```

```
                    "ENEE4113"]
            for line in f:
                tokens = line.split(";")
                localsum = 0
                localcounter = 0
                for i in range(1, len(tokens)):
                    token2 = tokens[i].split(":")
                    token2[0]=str(token2[0])
                    for x in range(0,len(remaining)):
                        if(token2[0] == remaining[x]):
                            temp.remove(remaining[x])
                    num=courses.get(token2[0])
                    hourstaken = hourstaken + num # to get hours taken
                    localsum = localsum + float(token2[1])
                    localcounter = localcounter + 1
                    sum = sum + float(token2[1])
                    counter = counter + 1
                localaverage = localsum / localcounter
                print("Average for semester " + tokens[0] + " is " +
localaverage)

            average = sum / counter
            print("Avargae all times is " + average)
            print("Finished hours=" + (allhours - hourstaken))
            print("Remaining courses are:" + temp)


    def Global(self):
        counter = 0  # to get students' number
        sum = 0
        courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3),
"ENCS3130": int(1),
                   "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3),
"ENCS3340": int(3),
                   "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1),
"ENCS4210": int(2),
```

```
                    "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3),
"ENCS4330": int(3),
                    "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1),
"ENCS5150": int(1),
                    "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1),
"ENEE2304": int(3),
                    "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3),
"ENEE3309": int(3),
                    "ENEE4113": int(1)
                    }


        semSum = {}  # summation of all hours in semesters
        for filename in glob.glob('*.txt'):
            counter = counter + 1
            with
open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'),
filename), 'r') as f:
                hourstaken = 0
                localsum = 0
                localcounter = 0  # to get average for a student in all times
                for line in f:
                    tokens = line.split(";")


                    for i in range(1, len(tokens)):  # because 0 is for
semester year
                        token2 = tokens[i].split(":")
                        token2[0].strip(" ").upper()
                        hourstaken = int(hourstaken + courses.get(token2[0]))
# to get hours taken
                        localsum = localsum + float(token2[1])
                        localcounter = localcounter + 1


                    if (semSum.has_key(tokens[0])):
                        temp = semSum.get(tokens[0]).split("#")#because in
semSum={sem year: summation of hours#number of hours , ...}
                        semSum[tokens[0]] = int(temp[0] + hourstaken) + "#" +
int(
                            temp[1] + 1)  # if semester year is in semSum then
add taken hours to its value
```

```
                    else:
                        semSum[tokens[0]] = int(hourstaken) +"#"+ int(1)  # if
not then add it and put the value of taken hours


                    localaverage = localsum / localcounter
                    sum = sum + localaverage
                average = sum / counter
                print("Overall students average=" + average)
                x = 0
                for x in range(0, len(semSum)):
                    sem_list = list(semSum.keys())
                    value_list = list(semSum.values())
                    temp2 = value_list[x].split("#")
                    avgHours = temp2[0] / temp2[1]
                    print("Average hours for semester " + sem_list[x] + " is="
+ avgHours)
        return "Operation done successfully!"


    def SearchAvg(self):
        op= input("Enter option: ")
        if (op == '1'):
            avg = input("Enter average: ")
            result = []
            for filename in glob.glob('*.txt'):
                with
open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'),
filename), 'r') as f:
                    sum = 0
                    counter = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):  # because 0 is for
semester year
                            token2 = tokens[i].split(":")
                            sum = sum + float(token2[1])
                            counter = counter + 1
```

```
                        average = sum / counter
                        if (average > avg):
                            result.append(f.name + ", average=" + average)
                            print("Students that have average greater than " +
avg + " are:" + result)
        elif (op == '2'):
            avg = input("Enter average: ")
            result = []
            for filename in glob.glob('*.txt'):
                with
open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'),
filename), 'r') as f:
                    sum = 0
                    counter = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):  # because 0 is for
semester year
                            token2 = tokens[i].split(":")
                            sum = sum + float(token2[1])
                            counter = counter + 1
                        average = sum / counter
                        if (average < avg):
                            result.append(f.name + ", average=" + average)
                    print("Students that have average less than " + avg + "
are:" + result)
        elif (op == '3'):
            avg = input("Enter average: ")
            result = []
            for filename in glob.glob('*.txt'):
                with
open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'),
filename), 'r') as f:
                    sum = 0
                    counter = 0
                    for line in f:
                        tokens = line.split(";")
```

```
                        for i in range(1, len(tokens)):  # because 0 is for
semester year
                            token2 = tokens[i].split(":")
                            sum = sum + float(token2[1])
                            counter = counter + 1
                        average = sum / counter
                        if (average == avg):
                            result.append(f.name + ", average=" + average)
            print("Students that have the average " + avg + " are:" + result)
        else:
            print("Error: wrong option")


    def SearchH():
        op= input("Enter option: ")
        courses = {"ENCS2110": int(1), "ENCS2340": int(3), "ENCS2380": int(3),
"ENCS3130": int(1),
                   "ENCS3310": int(3), "ENCS3320": int(3), "ENCS3330": int(3),
"ENCS3340": int(3),
                   "ENCS3390": int(3), "ENCS4110": int(1), "ENCS4130": int(1),
"ENCS4210": int(2),
                   "ENCS4300": int(3), "ENCS4310": int(3), "ENCS4320": int(3),
"ENCS4330": int(3),
                   "ENCS4370": int(3), "ENCS4380": int(3), "ENCS5140": int(1),
"ENCS5150": int(1),
                   "ENCS5200": int(2), "ENCS5300": int(3), "ENEE2103": int(1),
"ENEE2304": int(3),
                   "ENEE2307": int(3), "ENEE2312": int(3), "ENEE2360": int(3),
"ENEE3309": int(3),
                   "ENEE4113": int(1)
                   }


        if (op == '1'):
            hour = input("Enter hours: ")
            result = []
            for filename in glob.glob('*.txt'):
                with
open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'),
filename), 'r') as f:
                    hourstaken = 0
```

```
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):
                            tokens2 = tokens[i].split(":")
                            tokens2[0].strip(" ").upper()
                            hourstaken = int(hourstaken +
courses.get(tokens2[0]))
                        if (hourstaken > hour):
                            result.append(f.name + ", taken hours=" +
hourstaken)
                    print("Students that have taken hours more than " + hour +
" are:" + result)
        elif (op == '2'):
            hour = input("Enter hours: ")
            result = []
            for filename in glob.glob('*.txt'):
                with
open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'),
filename), 'r') as f:
                    hourstaken = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):
                            tokens2 = tokens[i].split(":")
                            tokens2[0].strip(" ").upper()
                            hourstaken = int(hourstaken +
courses.get(tokens2[0]))
                        if (hourstaken < hour):
                            result.append(f.name + ", taken hours=" +
hourstaken)
                    print("Students that have taken hours less than " + hour +
" are:" + result)
        elif (op == '3'):
            hour = input("Enter hours: ")
            result = []
            for filename in glob.glob('*.txt'):
                with
open(os.path.join(os.getcwd('C:/Users/Administrator/Desktop/records'),
filename), 'r') as f:
```

```
                    hourstaken = 0
                    for line in f:
                        tokens = line.split(";")
                        for i in range(1, len(tokens)):
                            tokens2 = tokens[i].split(":")
                            tokens2[0].strip(" ").upper()
                            hourstaken = int(hourstaken +
courses.get(tokens2[0]))
                        if (hourstaken == hour):
                            result.append(f.name + ", taken hours=" +
hourstaken)
                    print("Students that have taken hours equals " + hour + "
are:" + result)
        else:
            print("Error: wrong option")



class student(admin):


    def __init__(self):
        self




user = input("Enter Type Of User (Student Or Admin):")
if (user.lower() == "admin"):
    print("==========Menu=========")
    print("[1]Add a new record file")
    print("[2]Add new semester with student courses and grades")
    print("[3]Update data")
    print("[4]Student statistics")
    print("[5]Global statistics")
    print("[6]Search")
    option = input("Enter option ")
    status = False
    if (option == '1'):
```

```
        ad1 = admin()
        ad1.Addfile()
    elif (option == '2'):
        ad2 = admin()
        ad2.AddSemester()
        print(ad2.AddSemester())
    elif (option == '3'):
        ad3 = admin()
        ad3.Update()
    elif (option == '4'):
        ad4 = admin()
        ad4.statistics()
    elif (option == '5'):
        print("Here's the global statistics:")
        ad5 = admin()
        ad5.Global()
    elif (option == '6'):
        print("You can search for these:")
        print("[1]Based on average")
        print("[2]Based on taken hours")
        option2 = input("Enter option: ")
        if (option2 == '1'):
            print("[1]Averages greater than .....")
            print("[2]Averages less than ........")
            print("[3]Average equals ............")
            ad6 = admin()
            ad6.SearchAvg()
        elif (option2 == '2'):
            print("[1]Taken hours greater than ......")
            print("[2]Taken hours less than .........")
            print("[3]Taken hours equals ............")
            ad7 = admin()
            ad7.SearchH()
    else:
```

```
        print("Error: option is not in the menu")


elif (user.lower() == "student"):
    print("==========Menu========")
    print("[1]Student statistics:")
    print("[2]Global statistics")
    option = input("Enter option: ")
    if (option == '1'):
        st1 = student()
        st1.statistics()
    elif (option == '2'):
        print("Here's the global statistics:")
        st2 = student()
        st2.Global()
    else:
        print("Error: wrong option")
else:
    print("Error: invalid input")
```

Conclusion:

In conclusion, we have got better knowledge in python language and tried our best in this project. Finally, we hope our program gets your interests.