The Knowledge Hub Universities | partnered with Coventry University

school of computing

CW1

PROGRAMMING AND ALGORITHMS 1

Toqa Mahmoud
CU1900305

INTRODUCTION

In this coursework, I extended the grading system of CW1 to accept multiple modules using object oriented programming (OOP). The purpose of the program is to provide users with the ability to enter modules' data and students' data in each semester. It gives the user options to calculate and display the average score of the entire class for a specific assessment per module in a semester, the average score for the module in a semester, and the total score and academic performance for each student per module in a semester. It also provides the user with options to sort the output based on their selection. In addition, it can find and display the maximum/minimum score for a specific assessment per module in a semester, the maximum/minimum score for the module in a semester.

The program was precisely designed by optimal use of data types and data structures, proper use of control statements, efficient modular design using classes, methods, functions, correct use of specialized algorithms, and coherent flow sequence. This report aims to provide detailed analysis of the resulting grading system. It shows how the grading system was extended from CW1 on python.

DECISIONS AND ASSUMPTIONS

In this grading system program, I precisely designed and implemented a lot of decisions and assumptions to meet the user's best needs. To start the program, the user first enters whether they want the same modules or different modules in both semesters. After the user's selection, he/she enters the modules data and the student's data in each module. The program starts this way to make sure that it has data stored in it to implement the required options by displaying a menu and giving the user the option to choose whatever he/she wishes to.

1. One of the unique features I implemented was that the program supports semesters. In other words, in most universities/schools, they have two semesters per year. Likely, the program gives the user the full requirements for a grading system. He/she are asked to enter the module data in two semesters.
    a. The user has an option to enter the same module data in both semesters.
    b. The user has another option to enter different module data in both semesters.
    c. The user has to enter all students in at least 3 modules (not specifically the same), the students are entered by the user in each module. In other words, if a student is attending 3 modules, the user enters their data in each module. This gives the user the flexibility to choose any module for any student.
2. To make the code efficient the following limitations were executed
    a. I decided to make the modules limited to six modules per semester, adding up to twelve modules per year (if different modules were entered).
    b. I decided to make the assessments limited to 10 assessments per module in a semester, adding up to 20 assessments per year.
        i. The weight of all assessments per module in a semester have to add up to 100.
    c. I decided to make the students limited to 1,000 students per module in a semester.
        i. The grade for each student per assessment cannot be less than 0 or more than 100.
        ii. All students' names are changed to (.capitalize()) which capitalizes the students' names to display it professionally.
3. To make the code have a coherent flow sequence, the menu for all the possible options for the user is showed every time after the user executes a selection, he/she can end the program by selecting the option that ends it.

   a. After choosing any available option, the user is asked to choose the semester, module code, and the assessment number (if required), he/she wishes to for executing the option he/she chose from the menu.

4. In options such as finding the minimum/maximum or sorting, the user gets the option to choose what exactly they want.
   a. For example, in finding the minimum/maximum, the user gets to choose whether they want the minimum or maximum and after that they also get the option to choose whether they want to find the data of the student that got this particular grade.
   b. Another example, in sorting, the user also has the option whether to sort by first name, last name, or total score.
5. Most of the outputs are displayed using the tabulate feature which makes it aesthetically pleasing for the user.
6. An option is given to the user whether they want to save the output in a file.
7. Validation is accurately implemented in all inputs to ensure that the code runs smoothly.

The code is designed in a systemized way by conduction classes/OOP. Working with OOP did not only give me the flexibility to structure my program clearly, but also eased its modification.

PYTHON DATA STRUCTURES

Table 1 data structures

Class	Integers	Floats	Lists	Dictionaries
file	module_no	weight_list = [float]	cw	module_obj
student	no_students	grades	weights	obj
module	no_cw	CWs_list = [float]	marks	

	module_num		performance	
	cw_num		degree	
	sum		students	
	student_num		weight_list	
	count		students_lst	
	id_target		semester1	
	avg		semester2	
	semester		modules	
	assessment		main_studentslst_sem1	
	maxPerAssessment		main_studentslst_sem2	
	minPerAssessment		stu_table	
	c		stu	
	idx		lesser	
	userInput		greater	
	same_module			
	selection			

EXTRA FEATURES

1. Having two semesters in a year.
    a. Giving the user an option to enter the same module data in both semesters.

Figure 1 extra feature 1

```
def same_module_data():
    module_num = INTvalidation("Enter number of modules: ")
    while module_num not in range(0, 7):
        print("\nYou can enter up to 6 modules ONLY.")
        module_num = INTvalidation("Enter number of modules: ")
    for j in range(module_num):
        module_name = input("\nEnter module {} name: ".format(j + 1)).upper()
        module_code = input("Enter module {} code: ".format(j + 1)).upper()
        for sem in range(2):
            print("\n----------SEMESTER {} in MODULE {}----------\n".format(
            cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            while cw_num not in range(0, 11):
                print("You can enter up to 10 courseworks only.")
                cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            sum = 0
            while sum != 100:
                sum = 0
                weight_list = []
                for k in range(cw_num):
                    weight_list.append(FLOATvalidation("Enter the weight of CW {}: ".format(k + 1)))
                    sum += weight_list[k]
                if sum != 100:
                    print("Invalid, sum of weights should be equal to 100.")
            print("\nYou can enter up to 1,000 students.")
            student_num = INTvalidation("Enter number of students you want to enter: ")
            while student_num not in range(0, 1001):
                print("You can enter up to 1,000 students ONLY.")
                student_num = INTvalidation("Enter number of students you want to enter: ")
            students_lst = student_data(student_num, cw_num, weight_list, module_num, sem)
```

b. Giving the user an option to enter different module data in both semesters.

Figure 2 extra feature 2

```
def diff_module_data():
    for sem in range(2):
        print("\n-------------------------------SEMESTER {}-------------------------------\n".format(sem + 1))
        print("You can enter up to 6 modules.")
        module_num = INTvalidation("\nEnter number of modules: ")
        while module_num not in range(0, 7):
            print("You can enter up to 6 modules ONLY.")
            module_num = INTvalidation("\nEnter number of modules: ")
        for j in range(module_num):
            module_name = input("\nEnter module {} name: ".format(j + 1)).upper()
            module_code = input("Enter module {} code: ".format(j + 1)).upper()
            cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            while cw_num not in range(0, 11):
                print("You can enter up to 10 courseworks only.")
                cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            print(" ")
            sum = 0
            while sum != 100:
                sum = 0
                weight_list = []
                for k in range(cw_num):
                    weight_list.append(FLOATvalidation("Enter the weight of cw: " + str(k + 1) + " "))
                    sum += weight_list[k]
                if sum != 100:
                    print("Invalid, sum of weights should be equal to 100.")
            print("\nYou can enter up to 1,000 students.")
            student_num = INTvalidation("Enter number of students you want to enter: ")
            while student_num not in range(0, 1001):
                print("You can enter up to 1,000 students ONLY.")
                student_num = INTvalidation("Enter number of students you want to enter: ")
            # call student fn to enter student info
            students_lst = student_data(student_num, cw_num, weight_list, module_num, sem)
```

2. All options in the menu can be implemented per semester/module

Figure 3 extra feature 3

```
for module in semester1:
    if module_code == module.get_code():
        for student in module.get_students():
```

```
for module in semester2:
    if module_code == module.get_code():
        for student in module.get_students():
```

3. Display students data in a table format by using tabulate.

Figure 4 extra feature 4

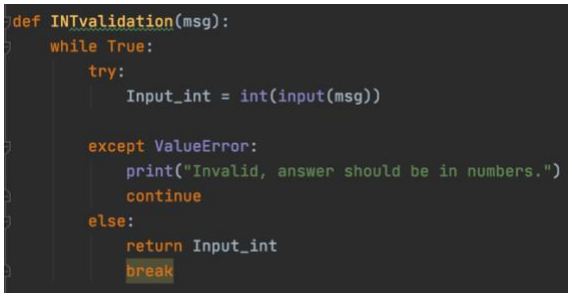
```
print(tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                   "(PERFORMANCE , DEGREE)"], showindex=student,
               tablefmt='fancy_grid'))
```

4. Save and display the output in a file

```
import sys


class file(object):
    def __init__(self):
        self.terminal = sys.stdout
        self.file = open("student.log", "a")

    def write(self, message):
        self.terminal.write(message)
        self.file.write(message)

    def flush(self):
        pass

    def close(self):
        self.file.close()
        pass
```

5. Input validation
    a. Integer input validation

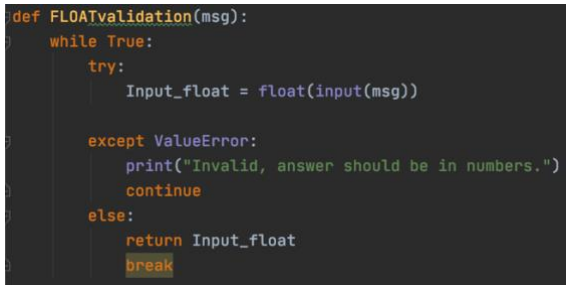
Figure 5 extra feature 5



```
def INTvalidation(msg):
    while True:
        try:
            Input_int = int(input(msg))

        except ValueError:
            print("Invalid, answer should be in numbers.")
            continue
        else:
            return Input_int
            break
```

    b. Float input validation

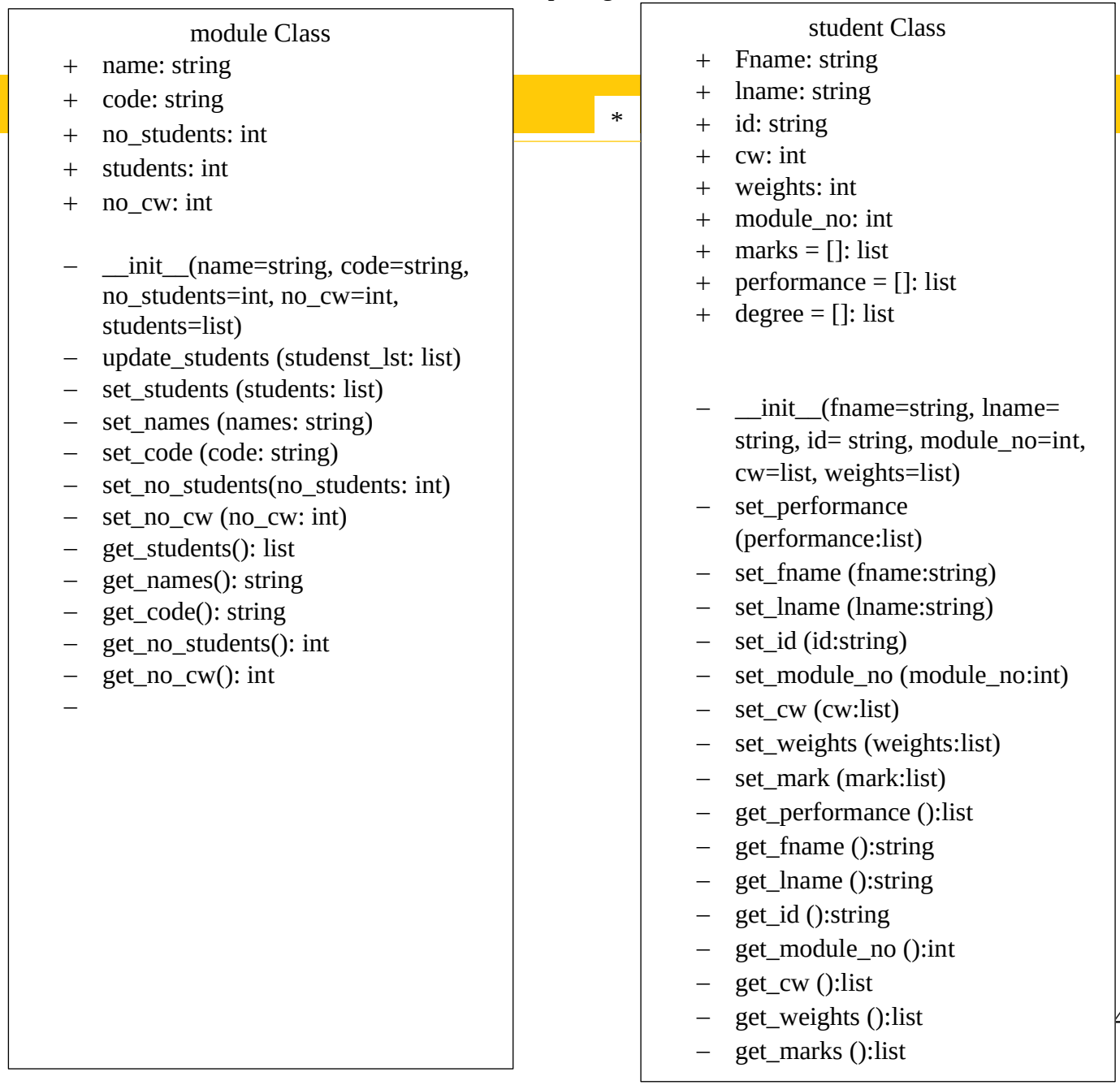
Figure 6 extra feature 6



```
def FLOATvalidation(msg):
    while True:
        try:
            Input_float = float(input(msg))

        except ValueError:
            print("Invalid, answer should be in numbers.")
            continue
        else:
            return Input_float
            break
```

CLASS DIAGRAM

Figure 7 class diagram



FUNCTIONS

Table 2 functions

Function Name	Arguments	Return	Explanation
same_module_data	none	none	This function is called when the user chooses the option of entering the same module data. It allows the user to enter the module data (name and code) at first and then asks the user for the assessments and its weights in each semester. After that the student_data function is called
diff_module_data	none	None	This function is called when the user chooses the option of entering different module data.

			It allows the user to enter the module data (name and code), the assessments and its weights in each semester. After that the student_data function is called
student_data	student_number (int) module_num (int) semester (int)	students (list)	In this function the user enters the amount of students in each semester and then their data.
validate_students	main_studentslst_sem1 (list) main_studentslst_sem2 (list)	none	This function makes sure that each student attended at least 3 modules per semester. If the student attended less than 3 modules the user is asked to enter them in more modules.
displaydata	none	none	This function is called when the user chooses option 1 from the menu, which displays the student data in a table.

avg_CW	none	assessment (int) module_code (str) avg (int)	This function is called when the user chooses option 2 from the menu, which calculates the average score per assessment.
avg_module	none	module_code (str) avg (int)	This function is called when the user chooses option 3 from the menu, which calculates the average score per module.
totalScore_students	none	stu_table (list)	This function is called when the user chooses option 4 from the menu, which calculates the total score of a student.
academicPerformance	Semester (int)	stu_table (list)	This function is called when the user chooses option 5 from the menu, which shows the academic performance for each student.
sort_alph	list (list) index (int)	lesser (list) [pivot] (list of 0) Greater (list)	This function is called when the user chooses option 6 from the menu, which sorts the

			students alphabetically depending on the user's choice, whether by first name or last name.
sort_grade	list (list) index (int)	lesser (list) [pivot] (list of 0) Greater (list)	This function is called when the user chooses option 8 from the menu, which sorts the students depending their total scores.
max_cw	module_code (str) list (list) idx (integer) cw (integer)	none	This function is called when the user chooses option 9 from the menu, which finds the maximum of a chosen assessment
min_cw	module_code (str) list (list) idx (integer) cw (integer)	none	This function is called when the user chooses option 9 from the menu, which finds the minimum of a chosen assessment

max_module	module_code (str) list (list) idx (integer)	none	This function is called when the user chooses option 10 from the menu, which finds the maximum of a chosen module
min_module	module_code (str) list (list) idx (integer)	none	This function is called when the user chooses option 10 from the menu, which finds the minimum of a chosen module
stu_info	stu_idx (int)	none	This function is called in the 4 predefined functions (min_cw, max_cw, min_module, max_module). This function gives the user the option to view the data of the student that got a particular grade.
INTvalidation	message (any)	Input_int (int)	This function is implemented in all integer inputs, to make sure the user enters an integer.

			It enters a while loop if the user enters anything other than an integer until he/she enters an integer
FLOATvalidation	Message (any)	Input_float (float)	This function is implemented in all float inputs, to make sure the user enters a float. It enters a while loop if the user enters anything other than a float until he/she enters an float
input_menu	none	none	This function is called first thing in the program which allows the user to choose between 2 options, whether they want to enter the same or different module data for both semesters.

menu	none	none	After the user is done with entering all module/student data this function is called which display a menu of all the predefined functions.

ALGORITHM TEMPLATE

Table 3 sorting algorithm template

Name	quicksort
Input	Unsorted list
Output	Sorted list
Content	The pivot is chosen as the first index of the list, it then divides the list into two, making the left side list lower than the pivot and the right side list greater than the pivot. This is

	repeated for each of the divided list until the whole list is sorted. The unsorted list is sorted depending on the user's option whether by first name, last name or total score.

Solution	if len(lst) == 0:
	  return []
	else:
	  pivot = lst[0]
	  lesser = sort_grade([x for x in lst[1:] if x[index] < pivot[index]], index)
	  greater = sort_grade([x for x in lst[1:] if x[index] >= pivot[index]], index)
	  return greater + [pivot] + lesser

Analysis	Quicksort is a divide-and-conquer algorithm. It works by selecting a pivot element from the list and dividing the other elements into two lists, according to whether they are less than

	or greater than the pivot. The divided lists are then sorted recursively. This algorithm takes O(*n* log *n*) comparisons to sort *n* items. Best case scenario: O(n log n) Average case scenario: O(n log n) Worst case scenario: O(n^2)
Variations	Selection sort, insertion sort, merge sort, bubble sort, shell sort

How does the quick sort work when the user chooses the option of sorting by total score?

20	100	10	80	60	50	90

1. The pivot is selected as the first element. It will divide the list into two parts, the right part will be the great and the left part will be the smaller.

20	100	10	80	60	50	90

2. We compare the selected pivot with the next index

100 > 20 = TRUE

3. We compare the selected pivot with the element after and so on.

10 > 20 = FALSE

80 > 20 = TRUE

20	100	80	10	60	50	90

60 > 20 = TRUE

20	100	80	60	10	50	90

50 > 20 = TRUE

20	100	80	60	50	10	90

90 > 20 = TRUE

20	100	80	60	50	90	10

4. Now we know the pivot's right place, the left of the pivot is greater and the right is smaller.

90	100	80	60	50	20	10

5. Now quicksort the left part, the first index is the pivot

90	100	80	60	50	20	10

100 > 90 = TRUE

80 > 90 = FALSE

50 > 90 = FALSE

60 > 90 = FALSE

6. The left of the 90 pivot is now the greater and the right is the smaller.

100	90	80	60	50	20	10

7. Now 100 is the pivot, it is at the right place so now we quick sort the right of 90 and select the first element as the pivot, which is 80 and so on.

100	90	80	60	50	20	10

60 > 80 = FALSE

50 > 80 = FALSE

100	90	80	60	50	20	10

50 > 60 = FALSE

100	90	80	60	50	20	10

8. The right part of pivot 20 is sorted, now we quick sort the left part, making 10 the pivot

100	90	80	60	50	20	10

9. 10 is at the right place, so the final sorted list is

100	90	80	60	50	20	10

SCENARIOS

Table 4 Scenarios

SCENARIOS	
Inputs	Outputs

```
def input_menu():
    print("----------FIRST WE'LL START BY ENTERING THE MODULE DATA----------")

    print("\n1- Enter the same modules in both semesters")
    print("2- Enter different modules in both semesters")

    same_module = INTvalidation("\nPlease choose the option that suits you (1 or 2): ")
    while same_module not in range(1, 3):
        print("Pleace chose 1 or 2.")
        same_module = INTvalidation("\nPlease choose the option that suits you (1 or 2): ")

    if same_module == 1:
        same_module_data()
        # validate on number of students per modules in every semester
        validate_numberOfStudentsPerModule(main_studentslst_sem1, main_studentslst_sem2)

    elif same_module == 2:
        diff_module_data()
        # validate on number of students per modules in every semester
        validate_numberOfStudentsPerModule(main_studentslst_sem1, main_studentslst_sem2)
```

Figure 8: input_menu() function

Scenario 1:

Output when the function is called:

```
----------PROGRAM HAS STARTED----------

----------FIRST WE'LL START BY ENTERING THE MODULE DATA----------

1- Enter the same modules in both semesters
2- Enter different modules in both semesters

Please choose the option that suits you (1 or 2):
```

Figure 9 input_menu() function output

Scenario 1:

```
Please choose the option that suits you (1 or 2): 1

You can enter up to 6 modules.
Enter number of modules:
```

Figure 10 input_menu() function scenario 1

same_module_data() function will be called.

Scenario 2:

The user inputs 1

Scenario 2:

The user inputs 2

Scenario 3:

The user inputs something else other than 1 or 2

Scenario 4:

The user inputs a string

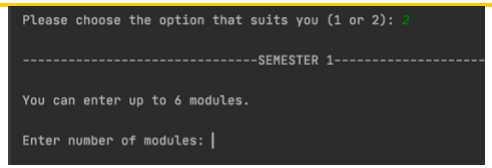


Figure 11 input_menu() function scenario 2

diff_module_data() function will be called.

Scenario 3:

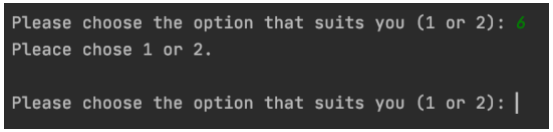


Figure 12 input_menu() function scenario 3

It will get in the while loop until the user chooses 1 or 2.

Scenario 4:

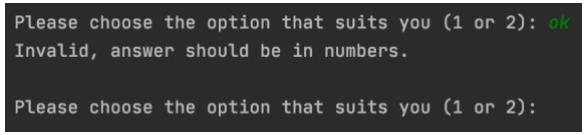


Figure 13 input_menu() function scenario 4

INTvalidation() function will be called

input_menu() function scenario 1

```
def same_module_data():
    module_num = INTvalidation("Enter number of modules: ")
    while module_num not in range(0, 7):
        print("\nYou can enter up to 6 modules ONLY.")
        module_num = INTvalidation("Enter number of modules: ")
    for j in range(module_num):
        module_name = input("\nEnter module {} name: ".format(j + 1)).upper()
        module_code = input("Enter module {} code: ".format(j + 1)).upper()
        for sem in range(2):
            print("\n------------------------------SEMESTER {} in MODULE {}------------------------------\n".format(
            cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            while cw_num not in range(0, 11):
                print("You can enter up to 10 courseworks only.")
                cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            sum = 0
            while sum != 100:
                sum = 0
                weight_list = []
                for k in range(cw_num):
                    weight_list.append(FLOATvalidation("Enter the weight of CW {}: ".format(k + 1)))
                    sum += weight_list[k]
                if sum != 100:
                    print("Invalid, sum of weights should be equal to 100.")
            print("\nYou can enter up to 1,000 students.")
            student_num = INTvalidation("Enter number of students you want to enter: ")
            while student_num not in range(0, 1001):
                print("You can enter up to 1,000 students ONLY.")
                student_num = INTvalidation("Enter number of students you want to enter: ")
            students_lst = student_data(student_num, cw_num, weight_list, module_num, sem)
```

Figure 14 same_module_data() function

Scenario 1:

The user entered all inputs correctly:

- The number of modules (module_num)
- The module name (module_name)
- The module code (module_code)
- The number of CWs (cw_num)
- The weight for each CW (weight_list)
- The number of students (student_num)

Scenario 1:

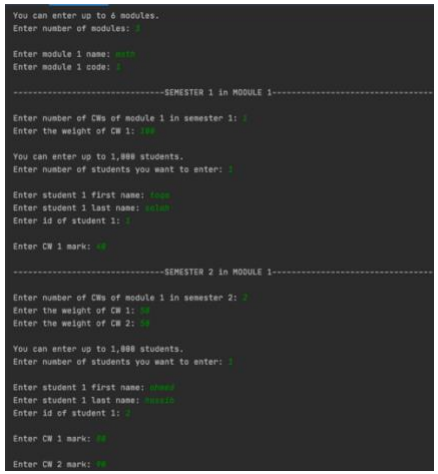


Figure 15 same_module_data() scenario 1

Figure 15 will be repeated 2 more times since the user chose to enter 3 modules(note: the user can enter up to 6 modules). This is the best case scenario since all inputs were entered right.

Scenario 2:

Scenario 2:

The user entered the following input wrongly:

- The number of modules (module_num)
- The weight for each CW (weight_list)
- The number of students (student_num)

Note: the student_data() function is called inside the same_module_data() function after asked the user to enter the number of students.

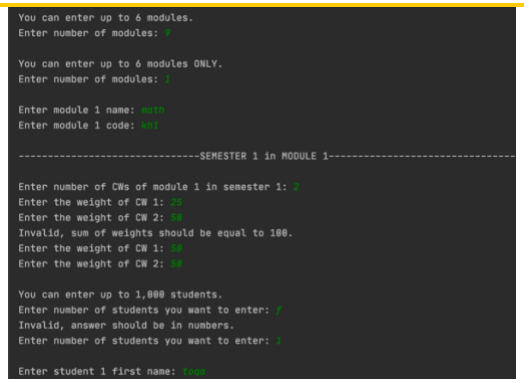


Figure 16 same_module_data() scenario 2

Figure 16 shows the input validation, every time the user enters an invalid input (whether a string to an integer input or exceeding the limit), the program enters a while loop until a valid answer is entered.

input_menu() function scenario 2

```
def diff_module_data():
    for sem in range(2):
        print("\n------------------------------SEMESTER {}------------------------------\n".format(sem + 1))
        print("You can enter up to 6 modules.")
        module_num = INTvalidation("\nEnter number of modules: ")
        while module_num not in range(0, 7):
            print("You can enter up to 6 modules ONLY.")
            module_num = INTvalidation("\nEnter number of modules: ")
        for j in range(module_num):
            module_name = input("\nEnter module {} name: ".format(j + 1)).upper()
            module_code = input("Enter module {} code: ".format(j + 1)).upper()
            cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            while cw_num not in range(0, 11):
                print("You can enter up to 10 courseworks only.")
                cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            print(" ")
            sum = 0
            while sum != 100:
                sum = 0
                weight_list = []
                for k in range(cw_num):
                    weight_list.append(FLOATvalidation("Enter the weight of cw: " + str(k + 1) + " "))
                    sum += weight_list[k]
                if sum != 100:
                    print("Invalid, sum of weights should be equal to 100.")
            print("\nYou can enter up to 1,000 students.")
            student_num = INTvalidation("Enter number of students you want to enter: ")
            while student_num not in range(0, 1001):
                print("You can enter up to 1,000 students ONLY.")
                student_num = INTvalidation("Enter number of students you want to enter: ")
            # call student fn to enter student info
            students_lst = student_data(student_num, cw_num, weight_list, module_num, sem)
```

Figure 17 diff_module_data() function

The inputs of the diff_module_data() function is the same as same_module_data() function.

```
------------------------------SEMESTER 1------------------------------

You can enter up to 6 modules.

Enter number of modules: 1

Enter module 1 name: math
Enter module 1 code: 1
Enter number of CWs of module 1 in semester 1: 1

Enter the weight of cw: 1 100

You can enter up to 1,000 students.
Enter number of students you want to enter: 1

Enter student 1 first name: toqa
Enter student 1 last name: salah
Enter id of student 1: 1

Enter CW 1 mark: 45

------------------------------SEMESTER 2------------------------------

You can enter up to 6 modules.

Enter number of modules: 1

Enter module 1 name: english
Enter module 1 code: 2
Enter number of CWs of module 1 in semester 2: 1

Enter the weight of cw: 1 100

You can enter up to 1,000 students.
Enter number of students you want to enter: 1
```

Figure 18 diff_module_data() function output

Figure 18 show the output if the diff_module_data function is called. Notice how in semester 2 they asked for another module data, however in Figure 15 same_module_data() scenario 1, in semester 2 the program did not ask for another module data.

Student_data() function

```
def student_data(student_number, module_num, semester):

    students = []
    for i in range(student_number):
        students_fname = str(input("\nEnter student {} first name: ".format(i + 1))).capitalize()
        students_lname = str(input("Enter student {} last name: ".format(i + 1))).capitalize()
        ID = input("Enter id of student {}: ".format(i + 1)).upper()
        CWs_list = []
        for k in range(0, cw_num):
            grades = (FLOATvalidation("\nEnter CW {} mark: ".format(k + 1)))
            while grades > 100 or grades < 0:
                print("Invalid, grade should be less than or equal 100.")
                grades = (FLOATvalidation("Enter CW {} mark: ".format(k + 1)))
            CWs_list.append((grades * weight_list[k]) / 100)
```

Figure 19 student_data() function

Scenario 1:

The user entered all inputs correctly:

- The student's first name (student_fname).
- The student's last name (student_lname).
- The student's ID (ID).
- The student's grades (grades).

Scenario 2:

The user entered the following input incorrect:

- The student's grades (grades).

Scenario 1:

```
Enter number of students you want to enter: 1

Enter student 1 first name: toqa
Enter student 1 last name: salah
Enter id of student 1: 1

Enter CW 1 mark: 60
```

Figure 20 student_data() function scenario 1

Figure 20 shows the output if all inputs are entered correctly.

Scenario 2:

```
Enter number of students you want to enter: 2

Enter student 1 first name: toqa
Enter student 1 last name: salah
Enter id of student 1: 1

Enter CW 1 mark: 110
Invalid, grade should be less than or equal 100.
Enter CW 1 mark:
```

Figure 21 student_data() function scenario 2

Figure 21 shows the output if inputs are entered wrongly. The grade of the student should be less than or equal to 100. If a number more than 100 is entered, the program enters a while loop until a valid number is entered.

```python
def displaydata():
    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')
    module_code = input('Enter the module code: ')
    stu_table = []
    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    academicPerformance(semester)
                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                           student.get_mark(), student.get_performance()]
                    stu_table.append(stu)
                print(tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                                   "(PERFORMANCE , DEGREE)"], showindex=student,
                               tablefmt='fancy_grid'))
    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    academicPerformance(semester=2)
                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                           student.get_mark(), student.get_performance()]
                    stu_table.append(stu)
                print(tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                                   "(PERFORMANCE , DEGREE)"], showindex=student,
                               tablefmt='fancy_grid'))
```

Figure 22 displaydata() function

displaydata() function will be called.

The input of the display data function is:

- The number of the semester (semester)
- The code of the module (module_code)

Scenario 1:

```
Enter your selection: 1
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
```

	FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE	(PERFORMANCE , DEGREE)
0	Toqa	Salah	1	[10.0, 20.0]	30	('Unsatisfactory', 'Fail')
1	Kariman	Alkuraish	2	[35.0, 17.5]	52.5	('satisfying', 'Lower second 2:2')
2	Ahmed	Hassib	3	[45.0, 50.0]	95	('Excellent to outstanding', 'First')

Figure 23 displaydata() function scenario 1

Figure 23 shows the output of the displaydata() function, if all input are entered correctly.

Scenario 2:

```
Enter your selection: 1
Please choose 1 or 2.
Enter the required semester: 3
Please choose 1 or 2.
Enter the required semester:
```

Figure 24 displaydata() function scenario 2

If the user chooses a semester other than 1 or 2, the program enter a while loop until 1 or 2 is entered.

menu() function scenario 2:

If the user selects 2

```
def avg_CW():
    avg = 0
    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')
    module_code = input('Enter the module code: ')
    assessment = INTvalidation('Enter the assessment number: ')
    while assessment not in range(1, cw_num + 1):
        print("\nPlease choose a valid number.")
        assessment = INTvalidation('Enter the assessment number: ')
    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    assessments = student.get_cw()
                    avg = avg + (assessments[assessment - 1] / module.get_no_students())
        return "\nThe average score of CW {} in module {} is: {}".format(assessment, module_code, avg)
    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    assessments = student.get_cw()
                    avg = avg + (assessments[assessment - 1] / module.get_no_students())
        return "\nThe average score of CW {} in module {} is: {}".format(assessment, module_code, avg)
```

Figure 25 avg_CW() function

avg_CW() function will be called.

In scenario 1 all inputs were entered correctly:

- The number of the semester (semester)
- The code of the module (module_code)
- The assessment number (assessment

In scenario 2 the following input was entered wrong:

- The assessment number (assessment

Scenario 1:

```
Enter your selection: 2
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
Enter the assessment number: 1

The average score of CW 1 in module 1 is: 30.0
```

Figure 26 avg_CW() function scenario 1

Scenario 2:

```
Enter your selection: 2
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
Enter the assessment number: q
Invalid, answer should be in numbers.
Enter the assessment number:
```

Figure 27 avg_CW() function scenario 2

If the user enters a string in any integer input, the program enters a while loop until an integer is entered.

menu() function scenario 3:

If the user selects 3

```
def avg_module():
    avg = 0
    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')
    module_code = input('Enter the module code: ')
    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    marks = student.get_mark()
                    avg = avg + (marks / module.get_no_students())
        return "The average score for module {} is: {}".format(module_code, avg)
    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    marks = student.get_mark()
                    avg = avg + (marks / module.get_no_students())
        return "The average score for module {} is: {}".format(module_code, avg)
```

Figure 28 avg_module() function

avg_module() function will be called.

In scenario 1 all inputs were entered correctly:

- The number of the semester (semester)
- The code of the module (module_code)

In scenario 2 the following input was entered wrong:

- The assessment number (assessment

Scenario 1:

```
Enter your selection: 3
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
The average score for module 1 is: 59.166
```

Figure 29 avg_module() function scenario 1

Scenario 2:

```
Enter your selection: >? 3
Please choose 1 or 2.
Enter the required semester: >? 1
Enter the module code: >? 7
The average score for module 7 is: 0
```

Figure 30 avg_module() function scenario 2

If the user enters a module code the is not found, the program will then calculate it as 0, and display the menu again.

menu() function scenario 4:

If the user selects 4

When the user selects 4, the totalScore_students() function gets called

```
def totalScore_students():
    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')
    module_code = input('Enter the module code: ')
    stu_table = []
    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_mark()]
                    stu_table.append(stu)
                return tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "TOTAL GRADE"], showindex=student,
                                tablefmt='fancy_grid')
    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_mark()]
                    stu_table.append(stu)
                return tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "TOTAL GRADE"], showindex=student,
                                tablefmt='fancy_grid')
```

Figure 31 totalScore_students() function

```
Enter your selection: 4
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 2
```

	FIRST NAME	LAST NAME	ID	TOTAL GRADE
0	Toqa	Salah	1	44
1	Ahmed	Hassib	2	77.4
2	Omar	Sharaky	3	38.5

Figure 32 totalScore_students() function output

menu() function scenario 5:

If the user selects 5

When the user selects 5, academicPerformance() function gets called

```
def academicPerformance(semester):
    stu_table = []
    if semester == 1:
        for module in semester1:
            for student in module.get_students():
                totalmarks = student.get_mark()
                if totalmarks > 70:
                    performance = "Excellent to outstanding"
                    degree = "First"
                elif totalmarks >= 60:
                    performance = "good to very good"
                    degree = "Upper second 2:1"
                elif totalmarks >= 50:
                    performance = "satisfying"
                    degree = "Lower second 2:2"
                elif totalmarks >= 40:
                    performance = "Sufficient"
                    degree = "Third 3"
                else:
                    performance = "Unsatisfactory"
                    degree = "Fail"
                student.set_performance(performance, degree)
                stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_mark(),
                       student.get_performance()]
                stu_table.append(stu)
            return tabulate(stu_table,
                            headers=["FIRST NAME", "LAST NAME", "ID", "TOTAL GRADE", "( PERFORMANCE , DEGREE )"],
                            showindex=student, tablefmt='fancy_grid')
```

Figure 33 academicPerformance() function

```
Please choose 1 or 2.
Enter the required semester: 2
```

	FIRST NAME	LAST NAME	ID	TOTAL GRADE	(PERFORMANCE , DEGREE)
0	Ziad	Abdallah	4	83	('Excellent to outstanding', 'First')
1	Kariman	Ahmed	5	61.25	('good to very good', 'Upper second 2:1')
2	Mariam	Waleed	6	49.85	('Sufficient', 'Third 3')

Figure 34 academicPerformance() function output

menu() function scenario 6:

If the user selects 6 or 7

When the user selects 6 or 7, the sort_alph() function gets called

```python
def sort_alph(list, idx):
    if len(list) == 0:
        return []
    else:
        pivot = list[0]
        lesser = sort_alph([x for x in list[1:] if x[idx] < pivot[idx]], idx)
        greater = sort_alph([x for x in list[1:] if x[idx] >= pivot[idx]], idx)
        return lesser + [pivot] + greater
```

Figure 35 sort_alph() function

```
Enter your selection: 6
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
```

FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE
Ahmed	Hassib	2	[56.0]	56
Omar	Sharaky	3	[65.0]	65
Toqa	Salah	1	[34.0]	34

```
Enter your selection: 7
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 2
```

FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE
Ahmed	Hassib	2	[33.0, 44.4]	77.4
Toqa	Salah	1	[11.5, 32.5]	44
Omar	Sharaky	3	[22.0, 16.5]	38.5

Figure 36 sort_alph() function output

menu() function scenario 7:

If the user selects 7

When the user selects 8, the sort_grade() function gets called

```
def sort_grade(list, idx):
    if len(list) == 0:
        return []
    else:
        pivot = list[0]
        lesser = sort_grade([x for x in list[1:] if x[idx] < pivot[idx]], idx)
        greater = sort_grade([x for x in list[1:] if x[idx] >= pivot[idx]], idx)
        return greater + [pivot] + lesser
```

Figure 37 sort_grade() function

```
Enter your selection: 8
Please choose 1 or 2.
Enter the required semester: 2
Enter the module code: 3
```

FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE
Mariam	Waleed	6	[89.0]	89
Kariman	Ahmed	5	[67.9]	67.9
Ziad	Abdallah	4	[34.0]	34

Figure 38 sort_grade() function output

menu() function scenario 8:

If the user selects 9

Scenario 1:

If the user chooses minimum, the min_cw() function will be called.

```
def min_cw(module_code, list, idx, cw):
    max = list[0][idx][cw]

    stu_idx = 0
    for i in range(len(list)):
        if list[i][idx][cw] < max:
            max = list[i][idx][cw]
            stu_idx = i
    print("\nthe max of module {} is: {}".format(module_code, max))
    stu_info(stu_idx)
```

Figure 39 min_cw() function

Scenario 2:

Scenario 1:

```
Enter your selection: 9
Please choose 1 or 2.
Enter the required semester: 2
Enter the module code: 2
Enter the assessment number: 1
Do you want to find the minimum or maximum (min for minimum & max for maximum)min

The min of CW 1 module 2 is: 40.00

Do you want to find out who go this grade (yes or no)?
```

Figure 41 min_cw() function output

If the user chooses maximum, the max_cw() function will be called.

```
def max_cw(module_code, list, idx, cw):
    max = list[0][idx][cw]

    stu_idx = 0
    for i in range(len(list)):
        if list[i][idx][cw] > max:
            max = list[i][idx][cw]
            stu_idx = i
    print("\nthe max of module {} is: {}".format(module_code, max))
    stu_info(stu_idx)
```

Figure 40 max_cw() function

Scenario 2:

```
Enter your selection: 9
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
Enter the assessment number: 1
Do you want to find the minimum or maximum (min for minimum & max for maximum)max

The max of CW 1 module 1 is: 33.00

Do you want to find out who go this grade (yes or no)?
```

Figure 42 max_cw() function output

menu() function scenario 9:

If the user selects 10

Scenario 1:

If the user chooses minimum, the min_mod() function will be called.

```
def min_module(module_code, list, idx):
    min = list[0][idx]
    stu_idx = 0

    for i in range(len(list)):
        if list[i][idx] < min:
            min = list[i][idx]
            stu_idx = i

    print("The max of module {} is: {:.2f}".format(module_code, min))
    stu_info(stu_idx)
```

Figure 43 min_module function()

Scenario 2:

If the user chooses maximum, the max_mod () function will be called.

Scenario 1:

```
Enter your selection: 10
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
Do you want to find the minimum or maximum (min for minimum & max for maximum)min
The minimum of module 1 is: 30.00

Do you want to find out who go this grade (yes or no)?
```

Figure 45 min_module function() output

```python
def max_module(module_code, list, idx):
    max = list[0][idx]

    stu_idx = 0
    for i in range(len(list)):
        if list[i][idx] > max:
            max = list[i][idx]
            stu_idx = i
    print("\nThe max of module {} is: {:.2f}".format(module_code, max))
    stu_info(stu_idx)
```

Figure 44 max_module function()

Scenario 2:

```
Enter your selection: 10
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 3
Do you want to find the minimum or maximum (min for minimum & max for maximum)max

The max of module 3 is: 90.00

Do you want to find out who go this grade (yes or no)?
```

Figure 46 max_module function() output

minimum and maximum functions scenario1:

inside of each function, the stu_info() function is called

```python
def stu_info(stu_idx):

    while True:
        choice = input("\nDo you want to find out who go this grade (yes or no)? ").lower()

        if choice == "yes":
            fstudent = main_studentslst_sem1[stu_idx].get_fname()
            lstudent = main_studentslst_sem1[stu_idx].get_lname()
            idstudent = main_studentslst_sem1[stu_idx].get_id()

            print(tabulate([[fstudent, lstudent, idstudent]], headers=["FRIST NAME", "LAST NAME", "ID"],
                           tablefmt='fancy_grid'))
            break
        elif choice == "no":

            print("Okay.")
            break
        else:
            print("Please answer with yes or no.")
```

Figure 47 stu_info() function

```
Enter your selection: 10
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
Do you want to find the minimum or maximum (min for minimum & max for maximum)min
The minimum of module 1 is: 30.00

Do you want to find out who go this grade (yes or no)? yes
╒═══════════════╤═══════════════╤══════╕
│ FRIST NAME    │ LAST NAME     │   ID │
╞═══════════════╪═══════════════╪══════╡
│ Toqa          │ Salah         │    1 │
╘═══════════════╧═══════════════╧══════╛
```

Figure 48 stu_info() function output

menu() function scenario 6:

If the user selects 11

```
elif selection == 11:
    print("Thanks for using the program ")
    quit()
```

Figure 49 selection 11

```
Enter your selection: 11
Thanks for using the program

Process finished with exit code 0
```

Figure 50 selection 11 output

TEST CASE SCENARIOS

FIRST TEST CASE SCENARIO:

The following snapshots show a complete valid scenario with inputs showing how my program can execute all its functionalities correctly. All inputs are colored in green.

Figure 51 first test case valid scenario

```
-----------------------------------PROGRAM HAS STARTED-----------------------------

----------------------FIRST WE'LL START BY ENTERING THE MODULE DATA-----------------

1- Enter the same modules in both semesters
2- Enter different modules in both semesters

Please choose the option that suits you (1 or 2): 1

You can enter from 3 to 6 modules.
Enter number of modules: 3

Enter module 1 name: programming and algorithms
Enter module 1 code (should be in numbers): 1

-------------------------------SEMESTER 1 in MODULE 1--------------------------------

Enter number of CWs of module 1 in semester 1: 1
Enter the weight of CW 1: 100

You can enter up to 1,000 students.
Enter number of students you want to enter: 3

Enter student 1 first name: toqa
Enter student 1 last name: salah
Enter id of student 1: 1

Enter CW 1 mark: 34

Enter student 2 first name: ahmed
Enter student 2 last name: hassib
Enter id of student 2: 2

Enter CW 1 mark: 56
```

```
Enter student 3 first name: omar
Enter student 3 last name: sharaky
Enter id of student 3: 3

Enter CW 1 mark: 65

------------------------------SEMESTER 2 in MODULE 1--------------------------------

Enter number of CWs of module 1 in semester 2: 2
Enter the weight of CW 1: 50
Enter the weight of CW 2: 50

You can enter up to 1,000 students.
Enter number of students you want to enter: 3

Enter student 1 first name: ziad
Enter student 1 last name: abdallah
Enter id of student 1: 4

Enter CW 1 mark: 99

Enter CW 2 mark: 67

Enter student 2 first name: kariman
Enter student 2 last name: ahmed
Enter id of student 2: 5

Enter CW 1 mark: 55

Enter CW 2 mark: 67.5

Enter student 3 first name: mariam
Enter student 3 last name: waleed
Enter id of student 3: 6

Enter CW 1 mark: 34.7
```

```
Enter CW 2 mark: 65

Enter module 2 name: ethical hacking and cyber security
Enter module 2 code (should be in numbers): 2

------------------------------SEMESTER 1 in MODULE 2--------------------------------

Enter number of CWs of module 2 in semester 1: 2
Enter the weight of CW 1: 50
Enter the weight of CW 2: 50

You can enter up to 1,000 students.
Enter number of students you want to enter: 3

Enter student 1 first name: taqa
Enter student 1 last name: salah
Enter id of student 1: 1

Enter CW 1 mark: 23

Enter CW 2 mark: 65

Enter student 2 first name: ahmed
Enter student 2 last name: hassib
Enter id of student 2: 2

Enter CW 1 mark: 66

Enter CW 2 mark: 88.8

Enter student 3 first name: omar
Enter student 3 last name: sharaky
Enter id of student 3: 3

Enter CW 1 mark: 44
```

```
Enter CW 2 mark: 33

------------------------------SEMESTER 2 in MODULE 2--------------------------------

Enter number of CWs of module 2 in semester 2: 3
Enter the weight of CW 1: 20
Enter the weight of CW 2: 20
Enter the weight of CW 3: 60

You can enter up to 1,000 students.
Enter number of students you want to enter: 3

Enter student 1 first name: ziad
Enter student 1 last name: abdallah
Enter id of student 1: 4

Enter CW 1 mark: 67

Enter CW 2 mark: 87

Enter CW 3 mark: 54

Enter student 2 first name: kariman
Enter student 2 last name: ahmed
Enter id of student 2: 5

Enter CW 1 mark: 22

Enter CW 2 mark: 45

Enter CW 3 mark: 65

Enter student 3 first name: mariam
Enter student 3 last name: waleed
Enter id of student 3: 6
```

```
Enter CW 1 mark: 22

Enter CW 2 mark: 33

Enter CW 3 mark: 43

Enter module 3 name: computer systems and networks
Enter module 3 code (should be in numbers): 3

------------------------------SEMESTER 1 in MODULE 3--------------------------------

Enter number of CWs of module 3 in semester 1: 4
Enter the weight of CW 1: 25
Enter the weight of CW 2: 25
Enter the weight of CW 3: 25
Enter the weight of CW 4: 25

You can enter up to 1,000 students.
Enter number of students you want to enter: 3

Enter student 1 first name: toqa
Enter student 1 last name: salah
Enter id of student 1: 1

Enter CW 1 mark: 44

Enter CW 2 mark: 55

Enter CW 3 mark: 55

Enter CW 4 mark: 65

Enter student 2 first name: ahmed
Enter student 2 last name: hassib
Enter id of student 2: 2
```

```
Enter CW 1 mark: 77

Enter CW 2 mark: 65

Enter CW 3 mark: 45

Enter CW 4 mark: 78

Enter student 3 first name: omar
Enter student 3 last name: sharaky
Enter id of student 3: 3

Enter CW 1 mark: 90

Enter CW 2 mark: 56

Enter CW 3 mark: 33

Enter CW 4 mark: 56.7

------------------------------SEMESTER 2 in MODULE 3---------------------------------

Enter number of CWs of module 3 in semester 2: 1
Enter the weight of CW 1: 100

You can enter up to 1,000 students.
Enter number of students you want to enter: 3

Enter student 1 first name: ziad
Enter student 1 last name: abdallah
Enter id of student 1: 4

Enter CW 1 mark: 34

Enter student 2 first name: kariman
Enter student 2 last name: ahmed
```

```
Enter id of student 2: 5

Enter CW 1 mark: 67.9

Enter student 3 first name: mariam
Enter student 3 last name: waleed
Enter id of student 3: 6

Enter CW 1 mark: 89

Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 1
```

```
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
```

	FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE	(PERFORMANCE , DEGREE)
0	Toqa	Salah	1	[34.0]	34	('Unsatisfactory', 'Fail')
1	Ahmed	Hassib	2	[56.0]	56	('satisfying', 'Lower second 2:2')
2	Omar	Sharaky	3	[65.0]	65	('good to very good', 'Upper second 2:1')

```
Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program
```

```
Enter your selection: 2
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
Enter the assessment number: 1

The average score of CW 1 in module 1 is: 51.66666666666667

Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 3
Please choose 1 or 2.
Enter the required semester: 2
Enter the module code: 3
```

```
The average score for module 3 is: 63.63333333333334

Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 4
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 2
```

```
┌─────┬────────────┬────────────┬──────┬─────────────┐
│     │ FIRST NAME │ LAST NAME  │  ID  │ TOTAL GRADE │
├─────┼────────────┼────────────┼──────┼─────────────┤
│  0  │ Toqa       │ Salah      │    1 │          44 │
├─────┼────────────┼────────────┼──────┼─────────────┤
│  1  │ Ahmed      │ Hassib     │    2 │        77.4 │
├─────┼────────────┼────────────┼──────┼─────────────┤
│  2  │ Omar       │ Sharaky    │    3 │        38.5 │
└─────┴────────────┴────────────┴──────┴─────────────┘

Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 5
```

```
Please choose 1 or 2.
Enter the required semester: 2
```

	FIRST NAME	LAST NAME	ID	TOTAL GRADE	(PERFORMANCE , DEGREE)
0	Ziad	Abdallah	4	83	('Excellent to outstanding', 'First')
1	Kariman	Ahmed	5	61.25	('good to very good', 'Upper second 2:1')
2	Mariam	Waleed	6	49.85	('Sufficient', 'Third 3')

```
Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program
```

```
Enter your selection: 6
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
```

FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE
Ahmed	Hassib	2	[56.0]	56
Omar	Sharaky	3	[65.0]	65
Toqa	Salah	1	[34.0]	34

```
Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module
```

```
11- End the program


Enter your selection: 7
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 2
```

FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE
Ahmed	Hassib	2	[33.0, 44.4]	77.4
Toqa	Salah	1	[11.5, 32.5]	44
Omar	Sharaky	3	[22.0, 16.5]	38.5

```
Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module
```

```
10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 8
Please choose 1 or 2.
Enter the required semester: 2
Enter the module code: 3
```

FIRST NAME	LAST NAME	ID	GRADES	TOTAL GRADE
Mariam	Waleed	6	[89.0]	89
Kariman	Ahmed	5	[67.9]	67.9
Ziad	Abdallah	4	[34.0]	34

```
Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students
```

```
9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 9
Please choose 1 or 2.
Enter the required semester: 1
Enter the module code: 1
Enter the assessment number: 1
Do you want to find the minimum or maximum (min for minimum & max for maximum)max
╒═══════════════╤═══════════╤═══════════════╕
│   MODULE CODE │   SEMESTER │   MAXIMUM GRADE │
╞═══════════════╪═══════════╪═══════════════╡
│             1 │          1 │             65 │
╘═══════════════╧═══════════╧═══════════════╛
Do you want to view student data of this grade (yes or no): yes
╒═══════════════╤═══════════════╤══════╤═══════════════╕
│ FIRST NAME    │ LAST NAME     │   ID │   MAXIMUM GRADE │
╞═══════════════╪═══════════════╪══════╪═══════════════╡
│ Omar          │ Sharaky       │    3 │             65 │
╘═══════════════╧═══════════════╧══════╧═══════════════╛

Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module

3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module
```

```
6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 10
Please choose 1 or 2.
Enter the required semester: 2
Enter the module code: 1
Do you want to find the minimum or maximum (min for minimum & max for maximum)min
╒═══════════════╤══════════════╤═══════════════╕
│   MODULE CODE │   SEMESTER   │   MINIMUM GRADE │
╞═══════════════╪══════════════╪═══════════════╡
│             1 │            2 │         49.85 │
╘═══════════════╧══════════════╧═══════════════╛
Do you want to view student data of this grade (yes or no): yes
╒═══════════════╤═══════════════╤══════╤═══════════════╕
│ FIRST NAME    │ LAST NAME     │   ID │   MINIMUM GRADE │
╞═══════════════╪═══════════════╪══════╪═══════════════╡
│ Mariam        │ Waleed        │    6 │         49.85 │
╘═══════════════╧═══════════════╧══════╧═══════════════╛

Please choose one of the following:

1- Display data of students in a module

2- calculate and display the avg score of a specific assessment in a module
```

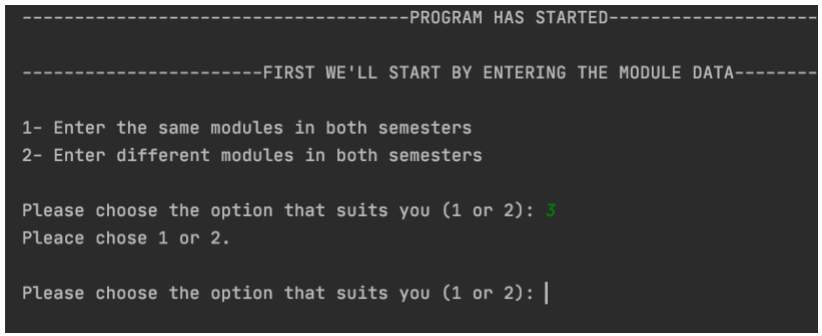
```
3- calculate and display the average score of a module

4- Calculate and display total marks for students in a module

5- Calculate and display academic performance for students in a module

6- Sort the output alphabetically based on the first name of the students

7- Sort the output alphabetically based on the last name of the students

8- Sort the output based on the total score of the students

9- Find the student with the lowest/highest mark in a certain assessment in a module

10- Find the student with the lowest/highest mark in a certain module

11- End the program


Enter your selection: 11
Thanks for using the program

Process finished with exit code 0
```

SECOND TEST CASE SCENARIO:

The following snapshots show multiple scenarios with invalid inputs, specifying the invalid input and the expected behavior of my program. All invalid inputs here are taken in the part where the user enters module/student data.

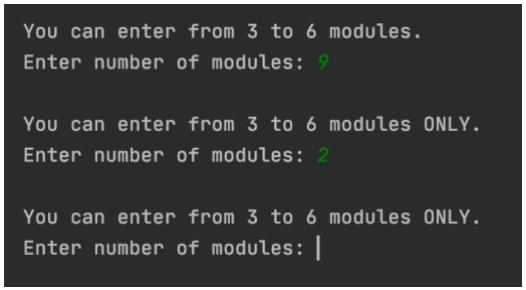
At the beginning of the code, the program gives the user 2 options to choose from if they enter any number other than 1 or 2, the program enters a while loop until a valid answer is entered

Figure 52 second test case scenario 1



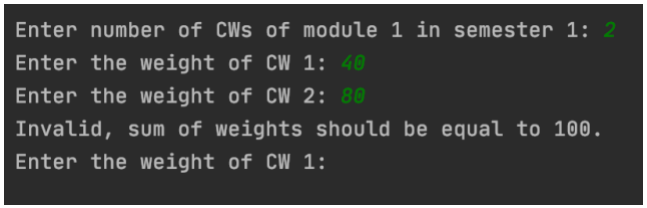
Here, the program gives the user the option to enter modules from 3 to 6, if they enter any number less than 3 or exceeding 6, the program enters a while loop until a valid answer is entered

Figure 53 second test case scenario 2



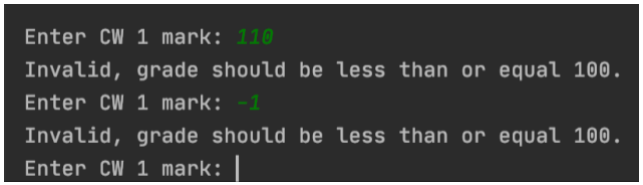
Here, the weights of all the assignments should equal to 100, if the sum is less than or more than 100, the program enters a while loop until a valid answer is entered

Figure 54 second test case scenario 3



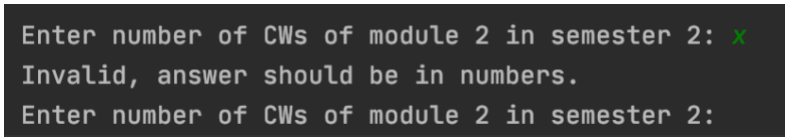
Here, the grade of a student per assignment should be less than or equal to 100, if the grade is more than 100, the program enters a while loop until a valid answer is entered.

Figure 55 second test case scenario 4



Moreover, if any integer input receives a string from the user, the program enters a while loop until a string is entered.

Figure 56 second test case scenario 5

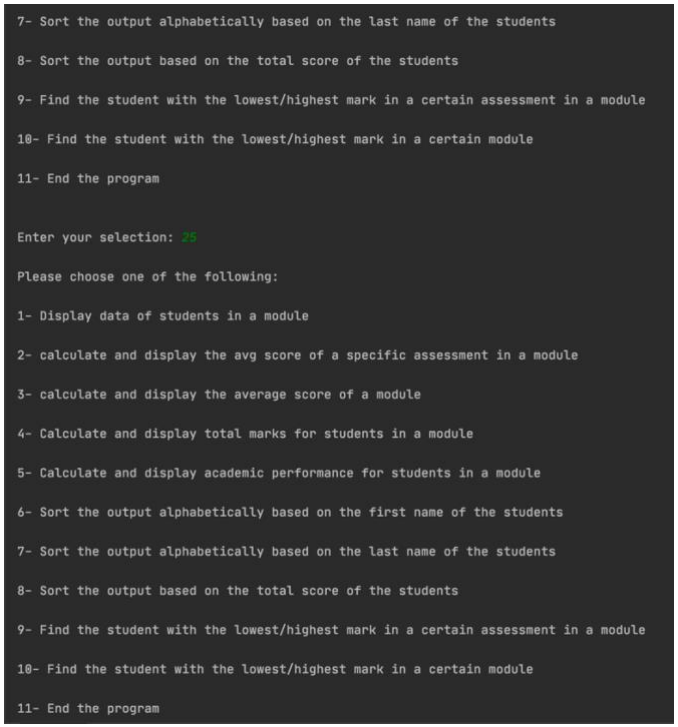


THIRD TEST CASE SCENARIO:

The following snapshots show another scenarios with another invalid inputs after the user input all module/student data, specifying the invalid input and the expected behavior of the program.

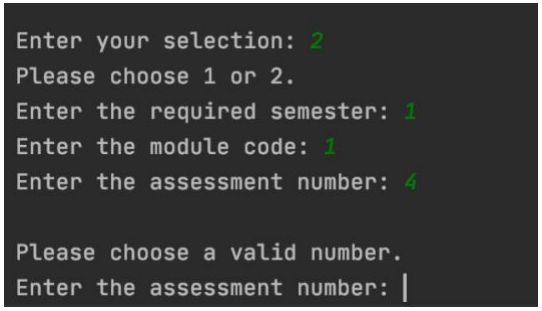
When the menu is displayed for the user, and they choose a number more than 11, the program enters a while loop to display the menu every time an invalid input is entered until a valid input is entered.

Figure 57 third test case scenario 1



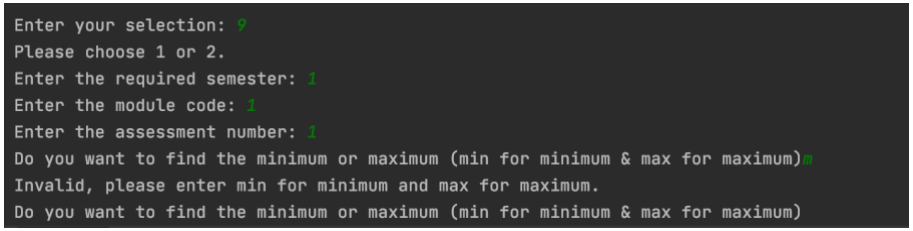
Here, after executing a selection from the menu, if the user enters an exceeding number of assessments, the program enters a while loop until a valid answer is entered.

Figure 58 third test case scenario 2



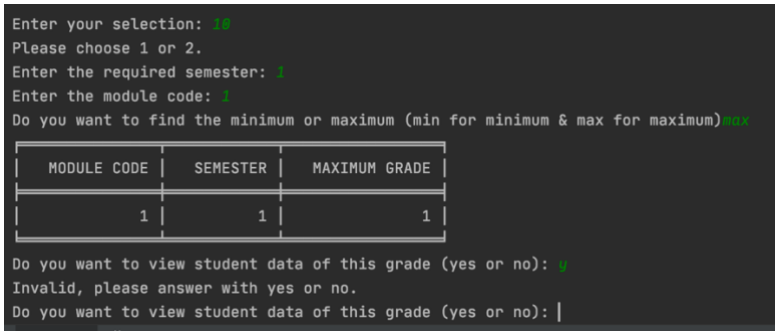
In option 9 and 10 in the menu, if the user answers with something else other than max or min (when is asked “Do you want to find the minimum or maximum”), the program enters a while loop until one of them is entered.

Figure 59 third test case scenario 3



In option 9 and 10 in the menu, if the user answers with something else other than yes or no (when is asked “Do you want to view student data of this grade”), the program enters a while loop until one of them is entered.

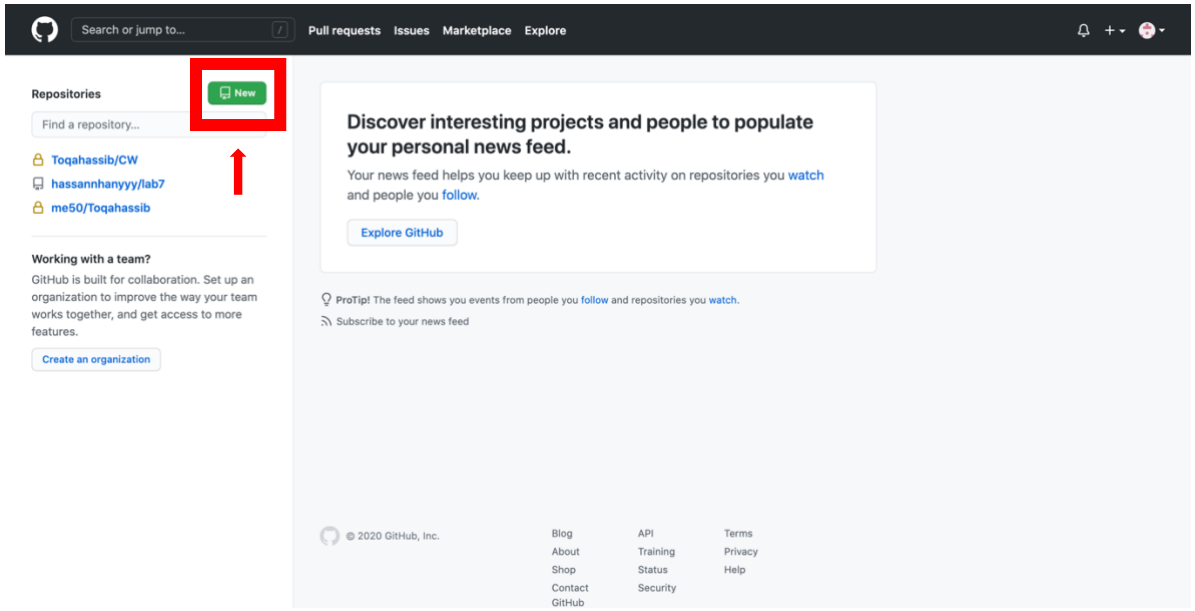
Figure 60 third test case scenario 4



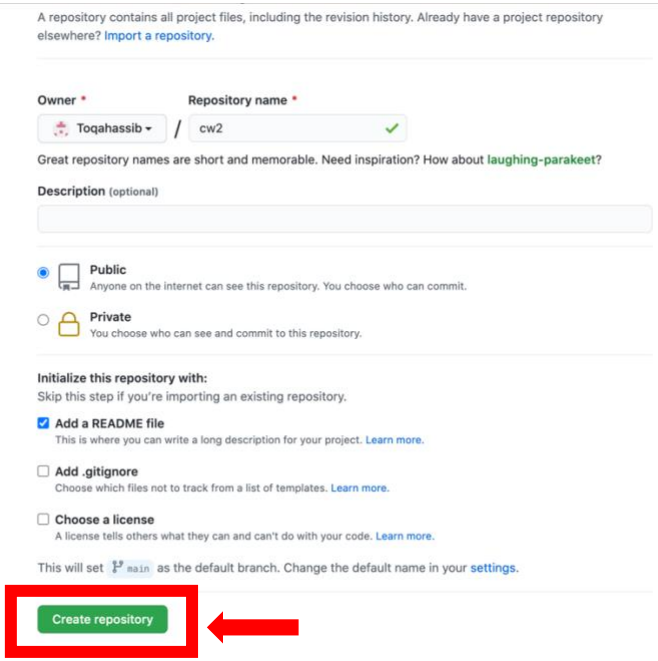
SOFTWARE DEVELOPMENT USING GITHUB

In this section I will explain a scenario where I am the team leader of a group of 4 (including myself) and all of us are working on the code. I will list the workflow in steps using GitHub commands explaining how each team member (including myself) will contribute in the development to deliver the complete working program.

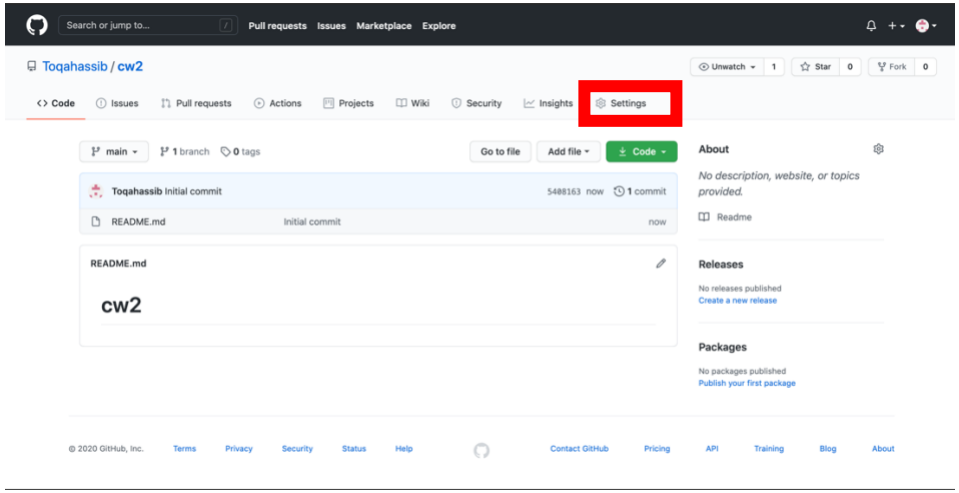
1. Me, the team leader, first created a new repository on my account and shared its name among my team.



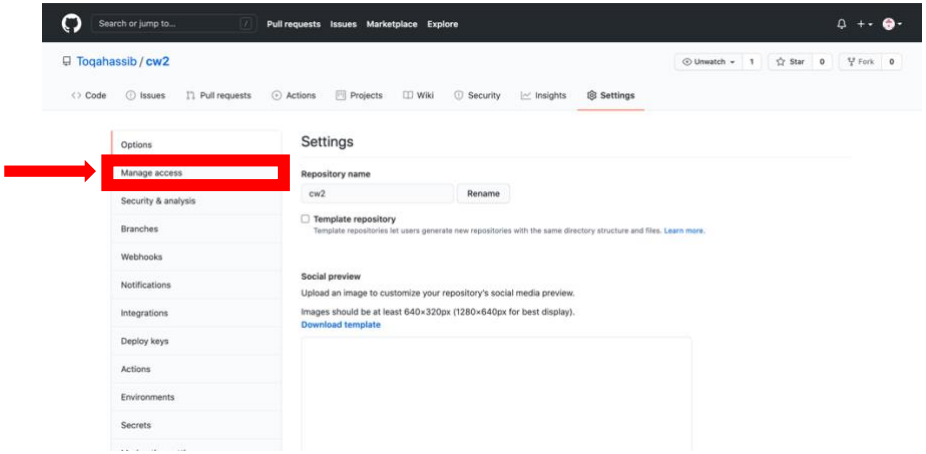
2. I then named the repository, as cw2, and checked add a README file, then clicked on create repository.


A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.
Owner *
Repository name *
Toqahassib
/
cw2
Great repository names are short and memorable. Need inspiration? How about laughing-parakeet?
Description (optional)
Public
Anyone on the internet can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
Initialize this repository with:
Skip this step if you're importing an existing repository.
Add a README file
This is where you can write a long description for your project. Learn more.
Add .gitignore
Choose which files not to track from a list of templates. Learn more.
Choose a license
A license tells others what they can and can't do with your code. Learn more.
This will set main as the default branch. Change the default name in your settings.
Create repository

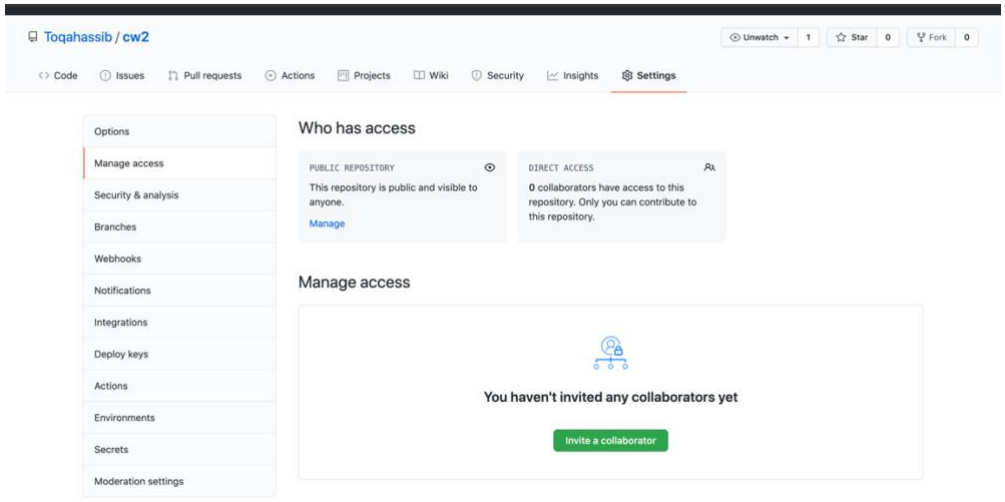
3. After the repository is created, I invited my team members to collaborate in this repository. This is done by clicking on settings.

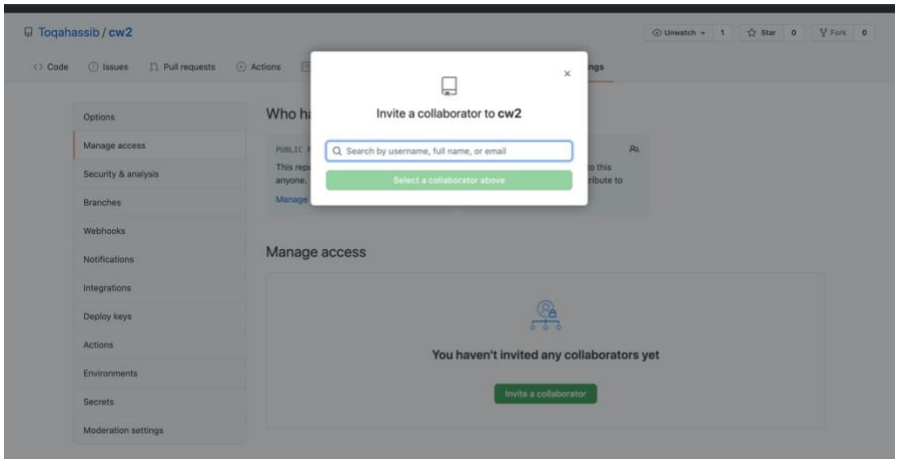


Then, on the left panel I clicked on manage access



After that, I invited my team members by clicking on invite a collaborator and then searched their GitHub usernames.





Finally, an invitation was sent to all my team members and they accepted it.

4. After that, all of us set our Git names and emails by doing the following.

```
Toqas-MBP:~ mac$ git config --global user.name toqahassib
Toqas-MBP:~ mac$ git config --global user.email toqahassib@gmail.com
Toqas-MBP:~ mac$
```

5. Then, we cloned our git repo.

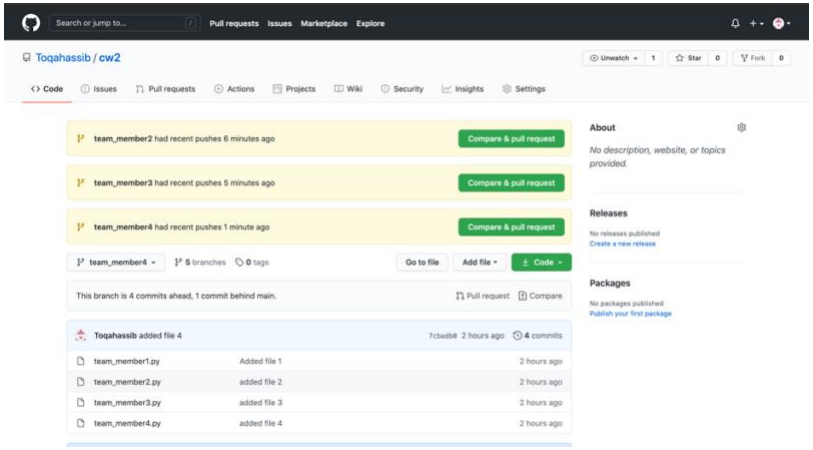
```
Toqas-MBP:~ mac$ git clone https://github.com/toqahassib/cw2.git
Cloning into 'cw2'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
Toqas-MBP:~ mac$
```

6. Now, the following steps only my team members followed

7. Each team member created a branch and pushed their files. By using the following commands

git checkout -b “username”
git add “filename”
git commit -m “message”
git push -u “URL” “branch name”

```
Toqas-MBP:CW2 mac$ git checkout -b team_member2
Switched to a new branch 'team_member2'
Toqas-MBP:CW2 mac$ git add team_member2.py
Toqas-MBP:CW2 mac$ git commit -m "added file 2"
[team_member2 dd0304b] added file 2
 1 file changed, 859 insertions(+)
 create mode 100644 team_member2.py
Toqas-MBP:CW2 mac$ git push -u https://github.com/Toqahassib/cw2.git team_member2
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 4.20 KiB | 4.20 MiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'team_member2' on GitHub by visiting:
remote:      https://github.com/Toqahassib/cw2/pull/new/team_member2
remote:
To https://github.com/Toqahassib/cw2.git
 * [new branch]      team_member2 -> team_member2
Branch 'team_member2' set up to track remote branch 'team_member2' from 'https://github.com/Toqahassib/cw2.git'.
```



8. After all team members pushed their files, I pulled the updated version of the repo and then merged my own branch to the main using the following commands

```
git checkout main
git pull origin username
git checkout main
git merge branch name
```

9. Finally, I pushed the repo again using the following command

```
git push -u origin
```

10. After that all my team members switched to their main branch and pulled the Git repository, using the following commands

```
git checkout main
git pull
git checkout main
```

11. The final file is running on my machine.

```
/Users/mac/PycharmProjects/CW2/venv/bin/python /Users/mac/PycharmProjects/CW2/Final.py
-----------------------------------PROGRAM HAS STARTED-----------------------------------

----------------------FIRST WE'LL START BY ENTERING THE MODULE DATA----------------------

1- Enter the same modules in both semesters
2- Enter different modules in both semesters

Please choose the option that suits you (1 or 2):
```

CONCLUSION

To conclude, this coursework is an extended version of coursework 1, where it implemented extra features which is to apply the concept of multiple modules. Working with classes/OOP made the program very organized which makes it reusable. This program contains many options for the user, not only can the user enter students in different modules but also in different semesters. This code is designed with two main extra features, first the user has an option to store the same modules in two semesters, and second the user has an option to store different modules in two semesters. This gives the user various options and makes the program extremely flexible.

At the beginning of this assignment, I had very little experience working with classes/OOP, so a large portion of time was dedicated to research, understanding and testing, as well as looking at alternative implementations to figure out what was most efficient. Finally, this made me improve my coding/programming skills and it also enhanced my problem solving skills.

REFERENCES

- https://www.tutorialspoint.com/uml/uml_class_diagram.htm
- https://www.geeksforgeeks.org/how-to-create-a-list-of-object-in-python-class/#:~:text=We%20can%20create%20list%20of,array%20of%20structures%20in%20C
- https://stackoverflow.com/questions/10619905/quicksort-a-list-containing-lists-using-python
- https://www.101computing.net/number-only/#:~:text=In%20most%20of%20your%20Python,the%20input()%20function%3A%20e.g.&text=1-,username%3Dinput(%22What%20is%20your%20username%3F%22,will%20need%20to%20retrieve%20numbers
- https://en.wikipedia.org/wiki/Quicksort#Parallelization
- https://stackoverflow.com/questions/14906764/how-to-redirect-stdout-to-both-file-and-console-with-scripting

APPENDIX

```
from tabulate import tabulate

import sys


class file(object):
    def __init__(self):
        self.terminal = sys.stdout
        self.file = open("student.log", "a")

    def write(self, message):
        self.terminal.write(message)
        self.file.write(message)

    def flush(self):
        pass

    def close(self):
        self.file.close()
        pass
```

```python
class student:
    def __init__(self, fname="", lname="", id="", module_no=0, cw=[], weights=[]):
        self.fname = fname
        self.lname = lname
        self.id = id
        self.cw = cw
        self.weights = weights
        self.module_no = module_no
        self.marks = []
        self.performance = []
        self.degree = []

    def set_performance(self, performance, degree):
        self.performance = performance
        self.degree = degree

    def set_fname(self, fname):
        self.fname = fname

    def set_lname(self, lname):
        self.lname = lname

    def set_id(self, id):
        self.id = id

    def set_module_no(self, module_no):
        self.module_no = module_no

    def set_cw(self, cw):
        self.cw = cw
```

```
    def set_weights(self, weights):
        self.weights = weights

    def set_mark(self, mark):
        self.mark = mark

    def get_marks(self):
        return self.marks

    def get_performance(self):
        return self.performance, self.degree

    def get_fname(self):
        return self.fname

    def get_lname(self):
        return self.lname

    def get_id(self):
        return self.id

    def get_module_no(self):
        return self.module_no

    def get_cw(self):
        return self.cw

    def get_weights(self):
        return self.weights

    def get_mark(self):
        return self.mark
```

```python
class module:

    def __init__(self, name="", code="", no_students=0, no_cw=0, students=[]):
        self.name = name
        self.code = code
        self.no_students = no_students

        self.students = students
        self.no_cw = no_cw

    def update_students(self, students_lst):
        for i in range(len(students_lst)):
            self.students.append(students_lst[i])

    def set_students(self, students):
        self.students = students

    def set_names(self, name):
        self.name = name

    def set_code(self, code):
        self.code = code

    def set_no_students(self, no_students):
        self.no_students = no_students

    def set_no_cw(self, no_cw):
        self.no_cw = no_cw

    def get_students(self):
        return self.students

    def get_names(self):
        return self.name
```

```
    def get_code(self):
        return self.code

    def get_no_students(self):
        return self.no_students

    def get_no_cw(self):
        return self.no_cw


module_obj = {}


def same_module_data():
    global cw_num, weight_list, students_lst

    print("\nYou can enter from 3 to 6 modules.")
    module_num = INTvalidation("Enter number of modules: ")

    while module_num not in range(3, 7):
        print("\nYou can enter from 3 to 6 modules ONLY.")
        module_num = INTvalidation("Enter number of modules: ")

    for j in range(module_num):
        module_name = input("\nEnter module {} name: ".format(j + 1)).upper()

        module_code = input("Enter module {} code (should be in numbers): ".format(j + 1)).upper()

        for sem in range(2):
            print("\n------------------------------SEMESTER {} in MODULE {}------------------------------\n".format(
                sem + 1, j + 1))

            cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            while cw_num not in range(0, 11):
                print("You can enter up to 10 courseworks only.")
```

```
            cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))

        sum = 0
        while sum != 100:
            sum = 0
            weight_list = []
            for k in range(cw_num):
                weight_list.append(FLOATvalidation("Enter the weight of CW {}: ".format(k + 1)))
                sum += weight_list[k]
            if sum != 100:
                print("Invalid, sum of weights should be equal to 100.")

        print("\nYou can enter up to 1,000 students.")
        student_num = INTvalidation("Enter number of students you want to enter: ")
        while student_num not in range(0, 1001):
            print("You can enter up to 1,000 students ONLY.")
            student_num = INTvalidation("Enter number of students you want to enter: ")

        # call student fn to enter student info
        students_lst = student_data(student_num, module_num, sem)

        module_obj[j] = module(module_name, module_code, student_num, cw_num, students_lst)

        if sem == 0:
            semester1.append(module_obj[j])

        elif sem == 1:
            semester2.append(module_obj[j])

    modules.append(semester1)
    modules.append(semester2)
```

```
def diff_module_data():
    for sem in range(2):
        print("\n------------------------------SEMESTER {}-------------------------------\n".format(sem + 1))
        print("You can enter from 3 to 6 modules.")
        module_num = INTvalidation("\nEnter number of modules: ")

        while module_num not in range(3, 7):
            print("You can enter from 3 to 6 modules ONLY.")
            module_num = INTvalidation("\nEnter number of modules: ")

        for j in range(module_num):

            module_name = input("\nEnter module {} name: ".format(j + 1)).upper()
            module_code = input("Enter module {} code (should be in numbers): ".format(j + 1)).upper()

            cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))
            while cw_num not in range(0, 11):
                print("You can enter up to 10 courseworks only.")
                cw_num = INTvalidation("Enter number of CWs of module {} in semester {}: ".format(j + 1, sem + 1))

            print(" ")

            sum = 0
            while sum != 100:
                sum = 0
                weight_list = []
                for k in range(cw_num):
                    weight_list.append(FLOATvalidation("Enter the weight of cw: " + str(k + 1) + " "))
                    sum += weight_list[k]
                if sum != 100:
                    print("Invalid, sum of weights should be equal to 100.")

            print("\nYou can enter up to 1,000 students.")
            student_num = INTvalidation("Enter number of students you want to enter: ")
            while student_num not in range(0, 1001):
```

```
            while student_num not in range(0, 1001):
                print("You can enter up to 1,000 students ONLY.")
                student_num = INTvalidation("Enter number of students you want to enter: ")

            # call student fn to enter student info
            students_lst = student_data(student_num, cw_num, weight_list, module_num, sem)

            module_obj[j] = module(module_name, module_code, student_num, cw_num, students_lst)

            if sem == 0:
                semester1.append(module_obj[j])

            elif sem == 1:
                semester2.append(module_obj[j])

    modules.append(semester1)
    modules.append(semester2)
    print(modules)


obj = {}


def student_data(student_number, module_num, semester):
    students = []
    for i in range(student_number):
        students_fname = str(input("\nEnter student {} first name: ".format(i + 1))).capitalize()
        students_lname = str(input("Enter student {} last name: ".format(i + 1))).capitalize()
        ID = input("Enter id of student {}: ".format(i + 1)).upper()
        CWs_list = []
        for k in range(0, cw_num):
            grades = (FLOATvalidation("\nEnter CW {} mark: ".format(k + 1)))
            while grades > 100 or grades < 0:
                print("Invalid, grade should be less than or equal 100.")
                grades = (FLOATvalidation("Enter CW {} mark: ".format(k + 1)))
```

```
            CWs_list.append((grades * weight_list[k]) / 100)

        mark = sum(CWs_list)
        obj[i] = student()
        obj[i].set_fname(students_fname)
        obj[i].set_lname(students_lname)
        obj[i].set_id(ID)
        obj[i].set_module_no(module_num)
        obj[i].set_cw(CWs_list)
        obj[i].set_weights(weight_list)
        obj[i].set_mark(mark)

        # academicPerformance(semester)
        obj[i].set_performance(performance, degree)
        students.append(obj[i])

        if semester == 0:
            main_studentslst_sem1.append(obj[i])
        elif semester == 1:
            main_studentslst_sem2.append(obj[i])

    return students


def validate_students(main_studentslst_sem1, main_studentslst_sem2):
    # for semester 1
    count = 0
    for student in (main_studentslst_sem1):
        id_target = student.get_id()
        for my_student in (main_studentslst_sem1):
            if id_target == my_student.get_id():
                count = count + 1

        if count < 3:
```

```
            print(
                "\nStudent {} {} with ID: {} has registered in less than 3 modules in semester 1, please add more "
                "modules.".format(student.get_fname(), student.get_lname(), id_target))

            module_code = input('Enter the module code: ')
            for mod in semester1:
                if module_code == mod.get_code():
                    students_lst = student_data(1, student.get_module_no(), 0)
                    mod.update_students(students_lst)

        count = 0

    # for semester 2
    count = 0
    for student in (main_studentslst_sem2):
        id_target = student.get_id()
        for my_student in (main_studentslst_sem2):
            if id_target == my_student.get_id():
                count = count + 1

        if count < 3:
            print(
                "Student {} {} with ID: {} has registered in less than 3 modules in semester 2, please add more "
                "modules.".format(student.get_fname(), student.get_lname(), id_target))

            module_code = input('Enter the module code: ')

            for mod in semester2:
                if module_code == mod.get_code():
                    students_lst = student_data(1, student.get_module_no(), 1)
                    mod.update_students(students_lst)

        count = 0
```

```
def displaydata():
    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')

    module_code = input('Enter the module code: ')
    stu_table = []
    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    academicPerformance(semester)

                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                           student.get_mark()]

                    stu_table.append(stu)

                print(tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE"],
                               showindex=student, tablefmt='fancy_grid'))

    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    academicPerformance(semester=2)

                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                           student.get_mark()]

                    stu_table.append(stu)
                print(tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE"],
                               showindex=student, tablefmt='fancy_grid'))
```

```
def avg_CW():
    avg = 0

    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')

    module_code = input('Enter the module code: ')

    assessment = INTvalidation('Enter the assessment number: ')
    while assessment not in range(1, cw_num + 1):
        print("\nPlease choose a valid number.")
        assessment = INTvalidation('Enter the assessment number: ')

    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    assessments = student.get_cw()
                    avg = avg + (assessments[assessment - 1] / module.get_no_students())

        return "\nThe average score of CW {} in module {} is: {:.2f}".format(assessment, module_code, avg)

    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    assessments = student.get_cw()
                    avg = avg + (assessments[assessment - 1] / module.get_no_students())

        return "\nThe average score of CW {} in module {} is: {:.2f}".format(assessment, module_code, avg)
```

```python
def avg_module():
    avg = 0

    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')

    module_code = input('Enter the module code: ')

    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    marks = student.get_mark()
                    avg = avg + (marks / module.get_no_students())
        return "The average score for module {} is: {:.2f}".format(module_code, avg)

    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    marks = student.get_mark()
                    avg = avg + (marks / module.get_no_students())

        return "The average score for module {} is: {:.2f}".format(module_code, avg)


# (c)
def totalScore_students():
    semester = 0
    while semester not in range(1, 3):
        print("Please choose 1 or 2.")
        semester = INTvalidation('Enter the required semester: ')
    module_code = input('Enter the module code: ')
```

```
    stu_table = []
    if semester == 1:
        for module in semester1:
            if module_code == module.get_code():
                for student in module.get_students():
                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_mark()]
                    stu_table.append(stu)
                return tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "TOTAL GRADE"], showindex=student,
                                tablefmt='fancy_grid')
    if semester == 2:
        for module in semester2:
            if module_code == module.get_code():
                for student in module.get_students():
                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_mark()]
                    stu_table.append(stu)
                return tabulate(stu_table, headers=["FIRST NAME", "LAST NAME", "ID", "TOTAL GRADE"], showindex=student,
                                tablefmt='fancy_grid')


# (d)
def academicPerformance(semester):
    stu_table = []
    if semester == 1:
        for module in semester1:
            for student in module.get_students():
                totalmarks = student.get_mark()
                if totalmarks > 70:
                    performance = "Excellent to outstanding"
                    degree = "First"
                elif totalmarks >= 60:
                    performance = "good to very good"
                    degree = "Upper second 2:1"
                elif totalmarks >= 50:
                    performance = "satisfying"
                    degree = "Lower second 2:2"
```

```
            elif totalmarks >= 40:
                performance = "Sufficient"
                degree = "Third 3"
            else:
                performance = "Unsatisfactory"
                degree = "Fail"
            student.set_performance(performance, degree)
            stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_mark(),
                   student.get_performance()]
            stu_table.append(stu)
        return tabulate(stu_table,
                        headers=["FIRST NAME", "LAST NAME", "ID", "TOTAL GRADE", "( PERFORMANCE , DEGREE )"],
                        showindex=student, tablefmt='fancy_grid')

if semester == 2:
    for module in semester2:
        for student in module.get_students():
            totalmarks = student.get_mark()

            if totalmarks > 70:
                performance = "Excellent to outstanding"
                degree = "First"
            elif totalmarks >= 60:
                performance = "good to very good"
                degree = "Upper second 2:1"
            elif totalmarks >= 50:
                performance = "satisfying"
                degree = "Lower second 2:2"
            elif totalmarks >= 40:
                performance = "Sufficient"
                degree = "Third 3"
            else:
                performance = "Unsatisfactory"
                degree = "Fail"
```

```
                student.set_performance(performance, degree)

                stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_mark(),
                       student.get_performance()]
                stu_table.append(stu)

            return tabulate(stu_table,
                            headers=["FIRST NAME", "LAST NAME", "ID", "TOTAL GRADE", "( PERFORMANCE , DEGREE )"],
                            showindex=student, tablefmt='fancy_grid')


# (e)
def sort_alph(list, idx):
    if len(list) == 0:
        return []
    else:
        pivot = list[0]
        lesser = sort_alph([x for x in list[1:] if x[idx] < pivot[idx]], idx)
        greater = sort_alph([x for x in list[1:] if x[idx] >= pivot[idx]], idx)
        return lesser + [pivot] + greater


# (e)
def sort_grade(list, idx):
    if len(list) == 0:
        return []
    else:
        pivot = list[0]
        lesser = sort_grade([x for x in list[1:] if x[idx] < pivot[idx]], idx)
        greater = sort_grade([x for x in list[1:] if x[idx] >= pivot[idx]], idx)
        return greater + [pivot] + lesser
```

```
def max_cw(module_code, list, idx, cw):
    max = list[0][idx][cw]

    stu_idx = 0
    for i in range(len(list)):
        if list[i][idx][cw] > max:
            max = list[i][idx][cw]
            stu_idx = i
    print("\nThe maximum of CW {} module {} is: {:.2f}".format(cw + 1, module_code, max))
    stu_info(stu_idx)


# (f)
def min_cw(module_code, list, idx, cw):
    max = list[0][idx][cw]

    stu_idx = 0
    for i in range(len(list)):
        if list[i][idx][cw] < max:
            max = list[i][idx][cw]
            stu_idx = i
    print("\nThe minimum of CW {} module {} is: {:.2f}".format(cw + 1, module_code, max))
    stu_info(stu_idx)


# (g)
def max_module(module_code, list, idx):
    max = list[0][idx]

    stu_idx = 0
    for i in range(len(list)):
        if list[i][idx] > max:
            max = list[i][idx]
            stu_idx = i
```

```
    print("\nThe maximum of module {} is: {:.2f}".format(module_code, max))
    stu_info(stu_idx)


# (g)
def min_module(module_code, list, idx):
    min = list[0][idx]
    stu_idx = 0

    for i in range(len(list)):
        if list[i][idx] < min:
            min = list[i][idx]
            stu_idx = i

    print("The minimum of module {} is: {:.2f}".format(module_code, min))
    stu_info(stu_idx)


# (h)
def stu_info(stu_idx):

    while True:
        choice = input("\nDo you want to find out who go this grade (yes or no)? ").lower()

        if choice == "yes":
            fstudent = main_studentslst_sem1[stu_idx].get_fname()
            lstudent = main_studentslst_sem1[stu_idx].get_lname()
            idstudent = main_studentslst_sem1[stu_idx].get_id()

            print(tabulate([[fstudent, lstudent, idstudent]], headers=["FRIST NAME", "LAST NAME", "ID"],
                           tablefmt='fancy_grid'))
            break
        elif choice == "no":
```

```
            print("Okay.")
            break
        else:
            print("Please answer with yes or no.")


def INTvalidation(msg):
    while True:
        try:
            Input_int = int(input(msg))

        except ValueError:
            print("Invalid, answer should be in numbers.")
            continue
        else:
            return Input_int
            break


def FLOATvalidation(msg):
    while True:
        try:
            Input_float = float(input(msg))

        except ValueError:
            print("Invalid, answer should be in numbers.")
            continue
        else:
            return Input_float
            break


def input_menu():
    print("----------------------FIRST WE'LL START BY ENTERING THE MODULE DATA-------------------------")
```

```
    print("\n1- Enter the same modules in both semesters")
    print("2- Enter different modules in both semesters")

    which_module = INTvalidation("\nPlease choose the option that suits you (1 or 2): ")
    while which_module not in range(1, 3):
        print("Pleace chose 1 or 2.")
        which_module = INTvalidation("\nPlease choose the option that suits you (1 or 2): ")

    if which_module == 1:
        same_module_data()
        # validate on number of students per modules in every semester
        validate_students(main_studentslst_sem1, main_studentslst_sem2)


    elif which_module == 2:
        diff_module_data()
        # validate on number of students per modules in every semester
        validate_students(main_studentslst_sem1, main_studentslst_sem2)


def menu():
    selection = 0

    while selection != 11:
        print("\nPlease choose one of the following: ")

        print("\n1- Display data of students in a module\n")

        print("2- calculate and display the avg score of a specific assessment in a module\n")
        print("3- calculate and display the average score of a module\n")
        print("4- Calculate and display total marks for students in a module \n")
        print("5- Calculate and display academic performance for students in a module \n")
        print("6- Sort the output alphabetically based on the first name of the students\n")
        print("7- Sort the output alphabetically based on the last name of the students\n")
        print("8- Sort the output based on the total score of the students\n")
```

```
    print("9- Find the student with the lowest/highest mark in a certain assessment in a module \n")
    print("10- Find the student with the lowest/highest mark in a certain module \n")
    print("11- End the program \n")

    selection = INTvalidation("\nEnter your selection: ")

    # (a) calculate and display the avg score of a specifc assignment in a module
    if selection == 1:
        displaydata()

    if selection == 2:
        print(avg_CW())

    # (b) calculate and display the avg score of a module
    elif selection == 3:
        print(avg_module())

    # (c) Calculate and display total marks for students in a module
    elif selection == 4:
        print(totalScore_students())

    # (d) calculate the academic performance for each student over all modules attended
    elif selection == 5:
        semester = 0
        while semester not in range(1, 3):
            print("Please choose 1 or 2.")
            semester = INTvalidation('Enter the required semester: ')

        print(academicPerformance(semester))

    # (e) Sort the output alphabetically based on the first name of the students
    elif selection == 6:
        semester = 0
        while semester not in range(1, 3):
            print("Please choose 1 or 2.")
```

```
            semester = INTvalidation('Enter the required semester: ')
        module_code = input('Enter the module code: ')
        stu_table = []
        if semester == 1:
            for module in semester1:
                if module_code == module.get_code():
                    for student in module.get_students():
                        stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                               student.get_mark()]
                        stu_table.append(stu)
                    print(tabulate(sort_alph(stu_table, 0),
                                   headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                            "(PERFORMANCE , DEGREE)"], tablefmt='fancy_grid'))

        if semester == 2:
            for module in semester2:
                if module_code == module.get_code():
                    for student in module.get_students():
                        stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                               student.get_mark()]
                        stu_table.append(stu)
                    print(tabulate(sort_alph(stu_table, 0),
                                   headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                            "(PERFORMANCE , DEGREE)"], tablefmt='fancy_grid'))

    # (e) Sort the output alphabetically based on the last name of the students
    elif selection == 7:
        semester = 0
        while semester not in range(1, 3):
            print("Please choose 1 or 2.")
            semester = INTvalidation('Enter the required semester: ')
        module_code = input('Enter the module code: ')
        stu_table = []
        if semester == 1:
            for module in semester1:
```

```
                if module_code == module.get_code():
                    for student in module.get_students():
                        stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                               student.get_mark()]
                        stu_table.append(stu)
                    print(tabulate(sort_alph(stu_table, 1),
                                   headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                            "(PERFORMANCE , DEGREE)"], tablefmt='fancy_grid'))
        if semester == 2:
            for module in semester2:
                if module_code == module.get_code():
                    for student in module.get_students():
                        stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                               student.get_mark()]
                        stu_table.append(stu)
                    print(tabulate(sort_alph(stu_table, 1),
                                   headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                            "(PERFORMANCE , DEGREE)"], tablefmt='fancy_grid'))

    # (e) Sort the output based on the total score of the students
    elif selection == 8:
        semester = 0
        while semester not in range(1, 3):
            print("Please choose 1 or 2.")
            semester = INTvalidation('Enter the required semester: ')

        module_code = input('Enter the module code: ')
        stu_table = []

        if semester == 1:
            for module in semester1:
                if module_code == module.get_code():
                    for student in module.get_students():
                        stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                               student.get_mark()]
```

```
                    stu_table.append(stu)

                print(tabulate(sort_grade(stu_table, 4),
                               headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                        "(PERFORMANCE , DEGREE)"], tablefmt='fancy_grid'))

        if semester == 2:
            for module in semester2:
                if module_code == module.get_code():
                    for student in module.get_students():
                        stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                               student.get_mark()]

                        stu_table.append(stu)

                    print(tabulate(sort_grade(stu_table, 4),
                                   headers=["FIRST NAME", "LAST NAME", "ID", "GRADES", "TOTAL GRADE",
                                            "(PERFORMANCE , DEGREE)"], tablefmt='fancy_grid'))

    # (f) max/min for specific assessment
    elif selection == 9:
        semester = 0
        while semester not in range(1, 3):
            print("Please choose 1 or 2.")
            semester = INTvalidation('Enter the required semester: ')

        module_code = input('Enter the module code: ')
        ass = INTvalidation("Enter the assessment number: ")
        while ass not in range(1, cw_num + 1):
            print("\nPlease choose a valid number.")
            ass = INTvalidation('Enter the assessment number: ')

        stu_table = []
        while True:
            minmax = input("Do you want to find the minimum or maximum (min for minimum & max for maximum)").lower()
```

```
            if minmax == "max":
                if semester == 1:
                    for module in semester1:
                        if module_code == module.get_code():
                            for student in module.get_students():
                                stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                       student.get_mark()]
                                stu_table.append(stu)
                            max_cw(module_code, stu_table, 3, ass - 1)
                if semester == 2:
                    for module in semester2:
                        if module_code == module.get_code():
                            for student in module.get_students():
                                stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                       student.get_mark()]
                                stu_table.append(stu)
                            max_cw(module_code, stu_table, 3, ass - 1)
                    break
                elif minmax == "min":
                    if semester == 1:
                        for module in semester1:
                            if module_code == module.get_code():
                                for student in module.get_students():
                                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                           student.get_mark()]
                                    stu_table.append(stu)
                                min_cw(module_code, stu_table, 3, ass - 1)

                    if semester == 2:
                        for module in semester2:
                            if module_code == module.get_code():
                                for student in module.get_students():
                                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                           student.get_mark()]
                                    stu_table.append(stu)
```

```
                            min_cw(module_code, stu_table, 3, ass - 1)
                    break
                else:
                    print("Invalid, please enter min for minimum and max for maximum.")

        # (g) max/min for specific module
        elif selection == 10:
            semester = 0
            while semester not in range(1, 3):
                print("Please choose 1 or 2.")
                semester = INTvalidation('Enter the required semester: ')

            module_code = input('Enter the module code: ')
            stu_table = []
            while True:
                minmax = input("Do you want to find the minimum or maximum (min for minimum & max for maximum)").lower()

                if minmax == "max":
                    if semester == 1:
                        for module in semester1:
                            if module_code == module.get_code():
                                for student in module.get_students():
                                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                           student.get_mark()]
                                    stu_table.append(stu)
                                max_module(module_code, stu_table, 4)

                    if semester == 2:
                        for module in semester2:
                            if module_code == module.get_code():
                                for student in module.get_students():
                                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                           student.get_mark()]
                                    stu_table.append(stu)
                                max_module(module_code, stu_table, 4)
```

```
                    break
                elif minmax == "min":
                    if semester == 1:
                        for module in semester1:
                            if module_code == module.get_code():
                                for student in module.get_students():
                                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                           student.get_mark()]
                                    stu_table.append(stu)
                                min_module(module_code, stu_table, 4)

                    if semester == 2:
                        for module in semester2:
                            if module_code == module.get_code():
                                for student in module.get_students():
                                    stu = [student.get_fname(), student.get_lname(), student.get_id(), student.get_cw(),
                                           student.get_mark()]
                                    stu_table.append(stu)
                                min_module(module_code, stu_table, 4)
                    break
                else:
                    print("Invalid, please enter min for minimum and max for maximum.")

        elif selection == 11:
            print("Thanks for using the program ")
            quit()


performance = ""
degree = ""

modules = []

semester1 = []
semester2 = []
```

```
main_studentslst_sem1 = []
main_studentslst_sem2 = []

stu_table = []

print("-----------------------------------PROGRAM HAS STARTED--------------------------------\n")

dr = input("\nPlease enter your name: ")

print("Hello Dr.", dr.capitalize())
print("\n---------------------------------------GRADING SYSTEM---------------------------------\n")

filechoice = input("Do you want to save the output in a text file (yes or no)? ")

input_menu()

menu()
```