Coursework 1 – Q3

a) **Design the data structure to manage student information, which includes their name, ID, and a list of modules. Each module should have a title, module ID, and marks.**

For the data structure I separated the information of the students and the information of the modules. This allowed me to access them independently, so the lists of modules between students may vary but the information of the all the available modules are stored somewhere else.

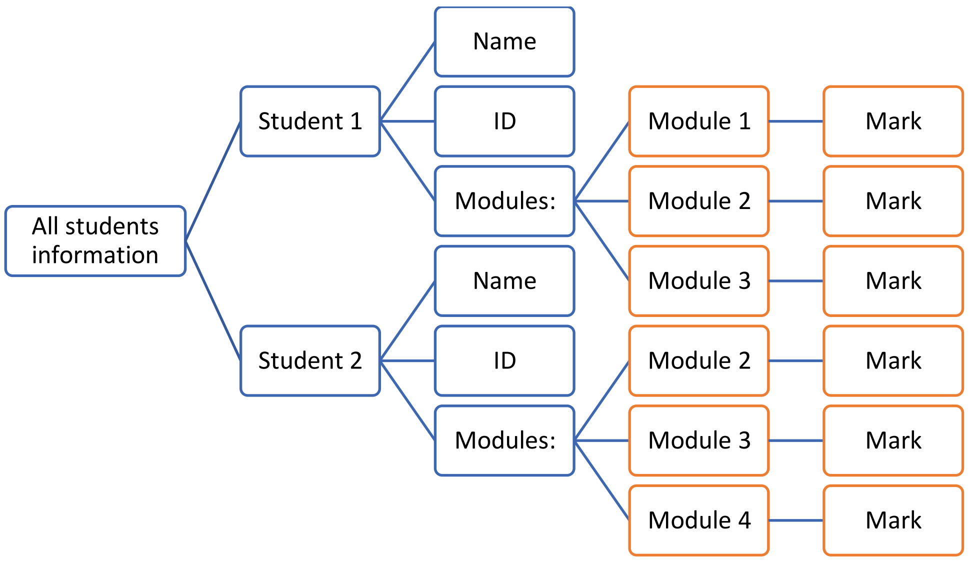


Figure 1. Data structure of student information

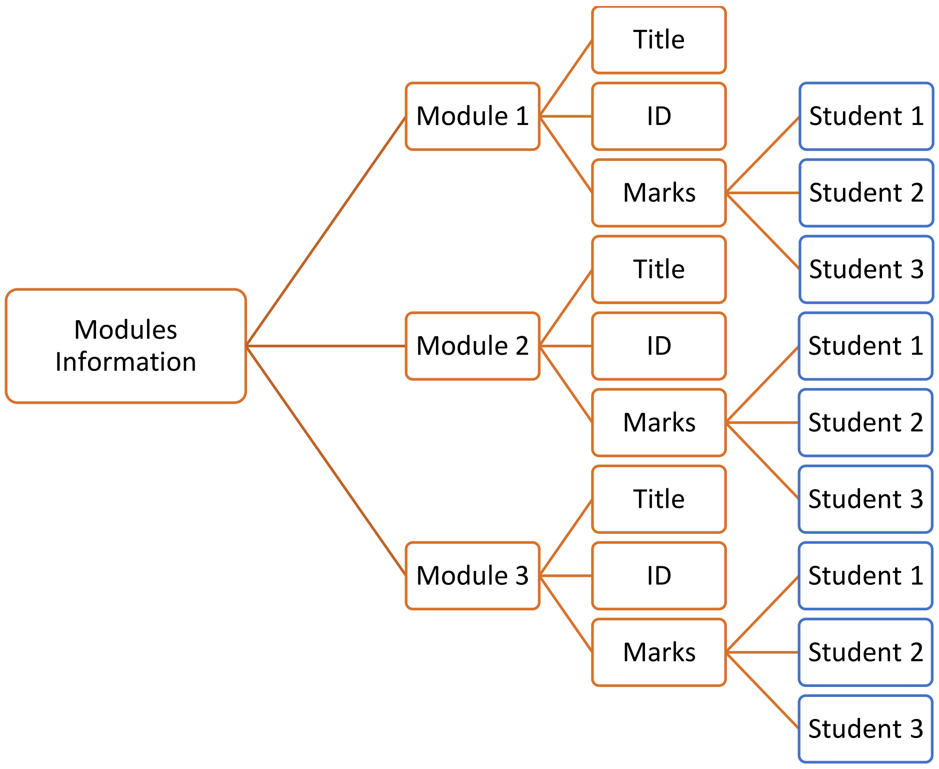
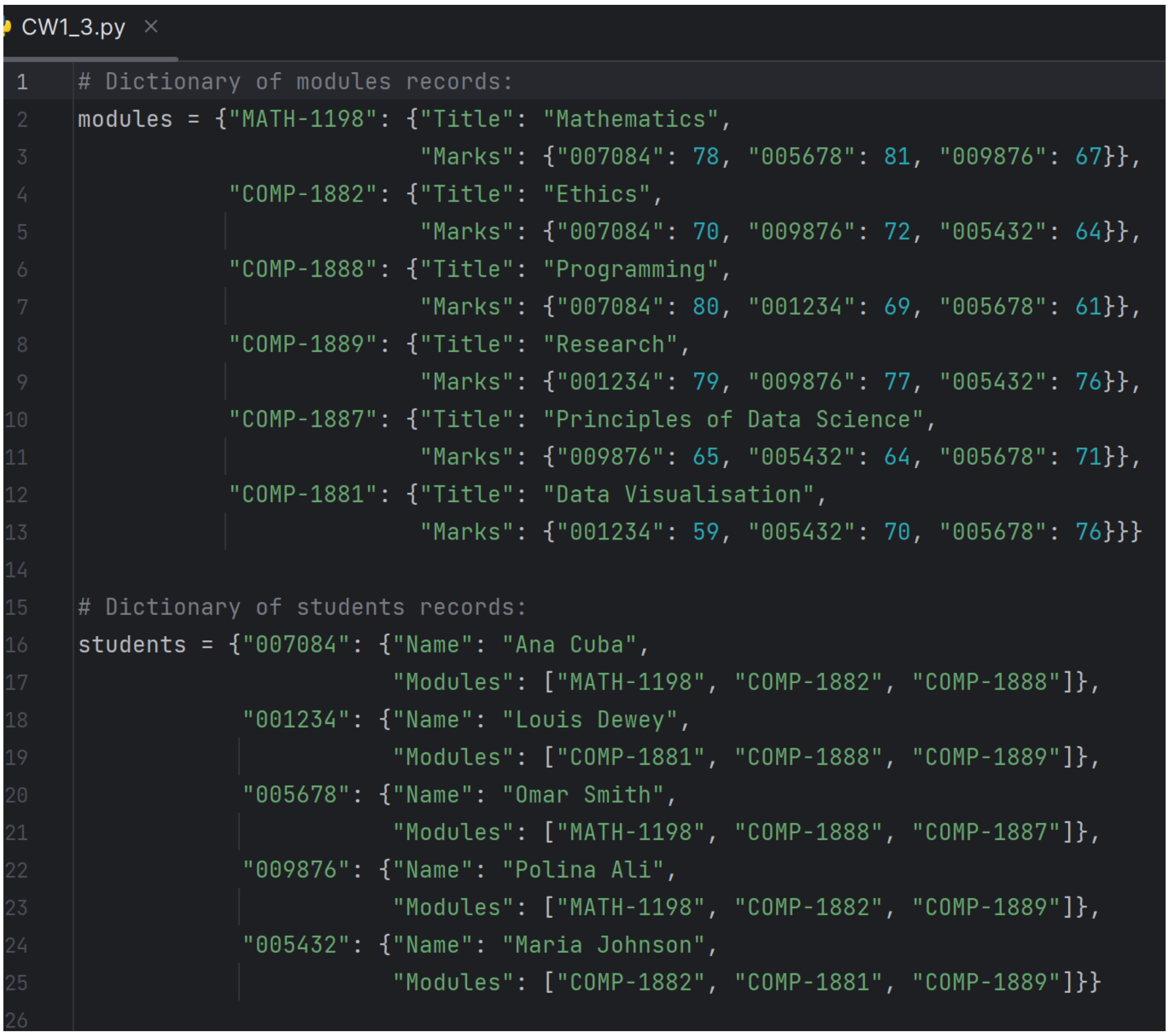


Figure 2. Data Structure of modules information

Following these structures, I created 2 dictionaries: Modules and Students. In the Students dictionary I used the student IDs as keys and the rest of information as a dictionary with the names and list of modules for each student. In the Modules dictionary the keys are the Module IDs, and the values are dictionaries containing the title of the module and marks of students.


CW1_3.py

```python
# Dictionary of modules records:
modules = {"MATH-1198": {"Title": "Mathematics",
                         "Marks": {"007084": 78, "005678": 81, "009876": 67}},
           "COMP-1882": {"Title": "Ethics",
                         "Marks": {"007084": 70, "009876": 72, "005432": 64}},
           "COMP-1888": {"Title": "Programming",
                         "Marks": {"007084": 80, "001234": 69, "005678": 61}},
           "COMP-1889": {"Title": "Research",
                         "Marks": {"001234": 79, "009876": 77, "005432": 76}},
           "COMP-1887": {"Title": "Principles of Data Science",
                         "Marks": {"009876": 65, "005432": 64, "005678": 71}},
           "COMP-1881": {"Title": "Data Visualisation",
                         "Marks": {"001234": 59, "005432": 70, "005678": 76}}}

# Dictionary of students records:
students = {"007084": {"Name": "Ana Cuba",
                       "Modules": ["MATH-1198", "COMP-1882", "COMP-1888"]},
            "001234": {"Name": "Louis Dewey",
                       "Modules": ["COMP-1881", "COMP-1888", "COMP-1889"]},
            "005678": {"Name": "Omar Smith",
                       "Modules": ["MATH-1198", "COMP-1888", "COMP-1887"]},
            "009876": {"Name": "Polina Ali",
                       "Modules": ["MATH-1198", "COMP-1882", "COMP-1889"]},
            "005432": {"Name": "Maria Johnson",
                       "Modules": ["COMP-1882", "COMP-1881", "COMP-1889"]}}
```

Figure 3. Dictionaries in Python

b) Enable end-users to add or remove a module for a specific student.
c) View the student profile, including the list of modules, by using the student ID.
d) Identify the students with the highest and lowest average marks.
e) Display the students in ascending order based on their average marks.

The program was developed in two parts:
First part: b) and c) because these ask for a specific student.
Second part: d) and e) because these use the information of all the students.

FIRST PART

We are required to enable the user to add or remove a module, or to show the profile of a specific student. So, the program must receive the student identification and what action to do.

The pseudo code of the program:

1. Define the dictionaries modules and students.

2. Define the functions: to add a module, to remove a module and to show the profile of the student.

3. Ask for a student ID and check if it exists.

4. Ask for the action to perform: 1) To add a module, 2) to remove a module or 3) to show the profile of the student.

5. If we need to add or remove a module: Ask for the module. Then call for the respective function and print the new list of modules.

6. If we need to show the profile: call for the function and print the result.

Functions to add or remove a module:

When adding a module, we must check if the module ID is in the dictionary of modules and then, that it is not in the existing list of modules of the student.

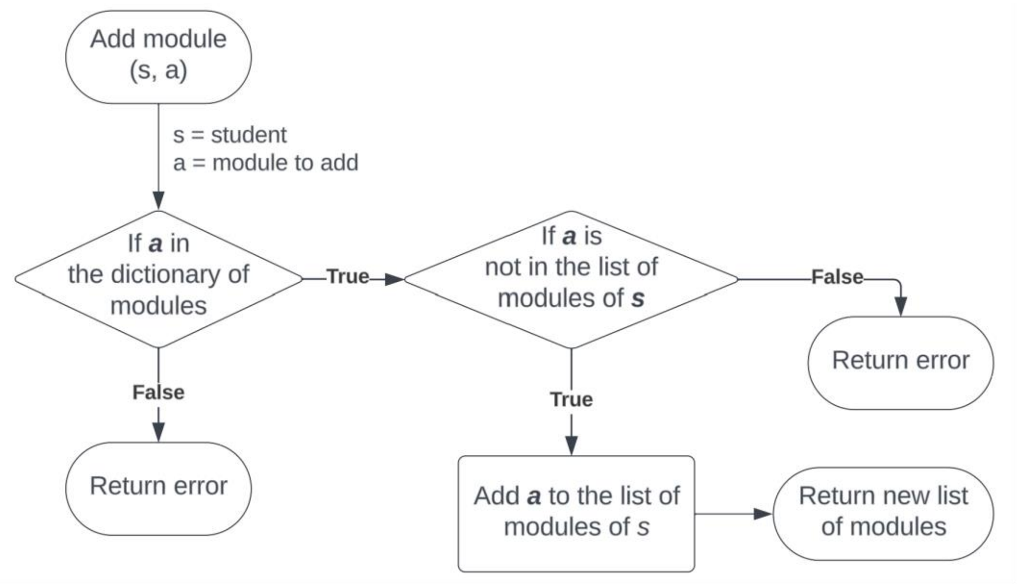

Add module (s, a)
s = student
a = module to add
If **a** in the dictionary of modules
True
If **a** is not in the list of modules of **s**
False
Return error
False
True
Return error
Add **a** to the list of modules of *s*
Return new list of modules

Figure 4. Flow chart of function to add a module for a specific student.

When removing a module, we just check that the module ID entered exists in the list of modules of the student.

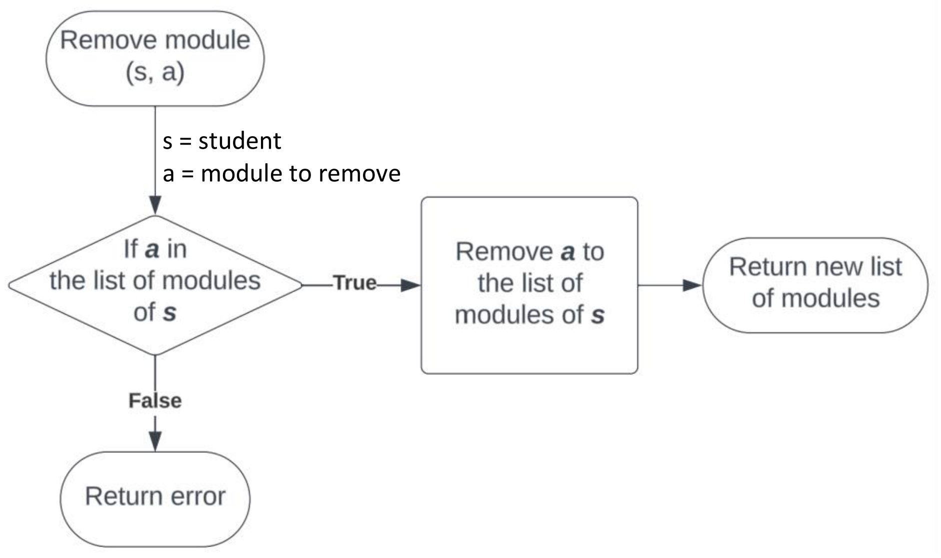
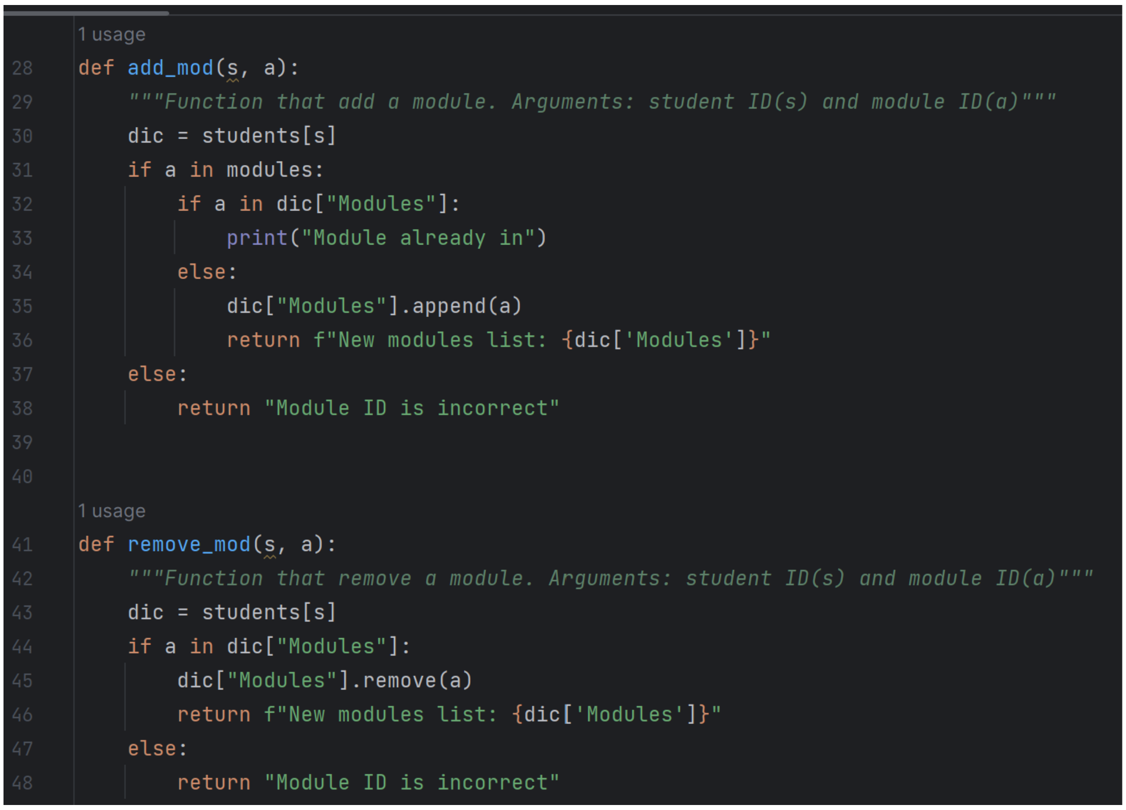

Remove module (s, a)
s = student
a = module to remove
If **a** in the list of modules of **s**
True
Remove **a** to the list of modules of **s**
Return new list of modules
False
Return error

Figure 5. Flow chart of function to remove a module for a specific student.

The functions look like this:



```
1 usage
def add_mod(s, a):
    """Function that add a module. Arguments: student ID(s) and module ID(a)"""
    dic = students[s]
    if a in modules:
        if a in dic["Modules"]:
            print("Module already in")
        else:
            dic["Modules"].append(a)
            return f"New modules list: {dic['Modules']}"
    else:
        return "Module ID is incorrect"


1 usage
def remove_mod(s, a):
    """Function that remove a module. Arguments: student ID(s) and module ID(a)"""
    dic = students[s]
    if a in dic["Modules"]:
        dic["Modules"].remove(a)
        return f"New modules list: {dic['Modules']}"
    else:
        return "Module ID is incorrect"
```

Figure 6. Functions to add and remove modules for a student.

Function to show the profile of the student:

In the students' dictionary we have the student IDs as keys and a dictionary with the names and list of modules for each student. We could just call for this information. But if we want to see the grades of the student as well, we can get them from the modules' dictionary, and show a more complete profile.

So, first we need a function to get the list of grades of a student.

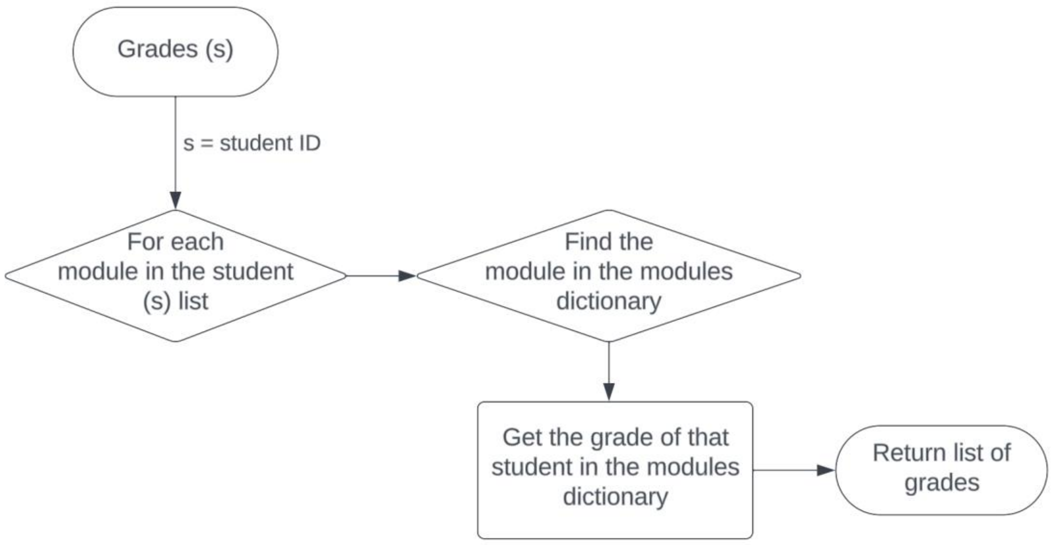
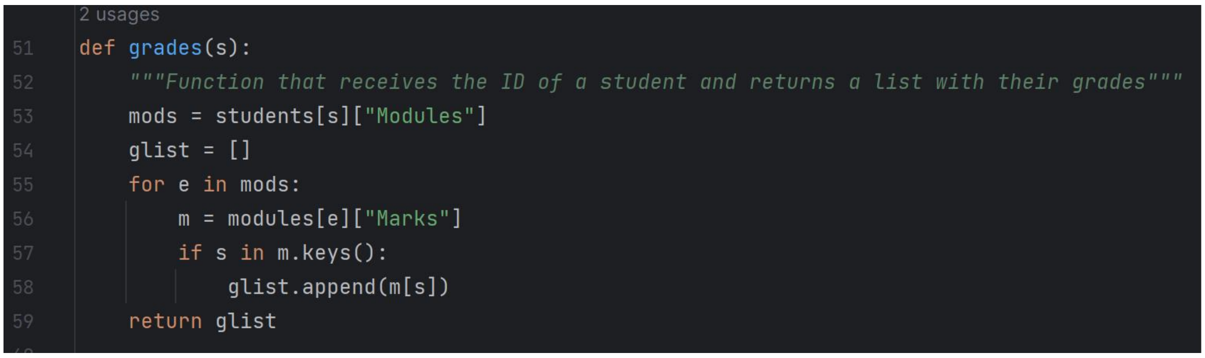

Grades (s)
s = student ID
For each module in the student (s) list
Find the module in the modules dictionary
Get the grade of that student in the modules dictionary
Return list of grades

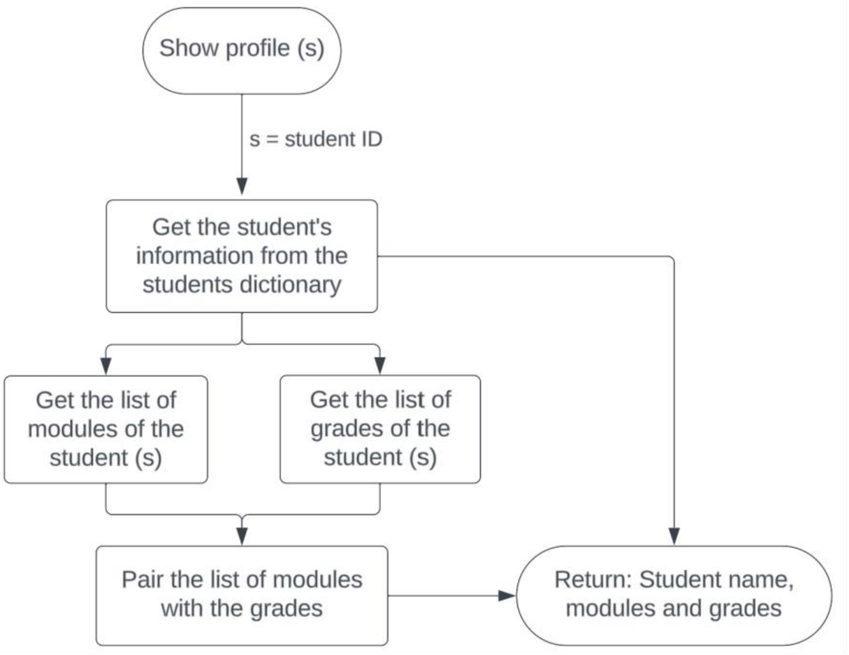
Figure 7. Flow chart of function to get the list of grades for a specific student.



```
2 usages
def grades(s):
    """Function that receives the ID of a student and returns a list with their grades"""
    mods = students[s]["Modules"]
    glist = []
    for e in mods:
        m = modules[e]["Marks"]
        if s in m.keys():
            glist.append(m[s])
    return glist
```

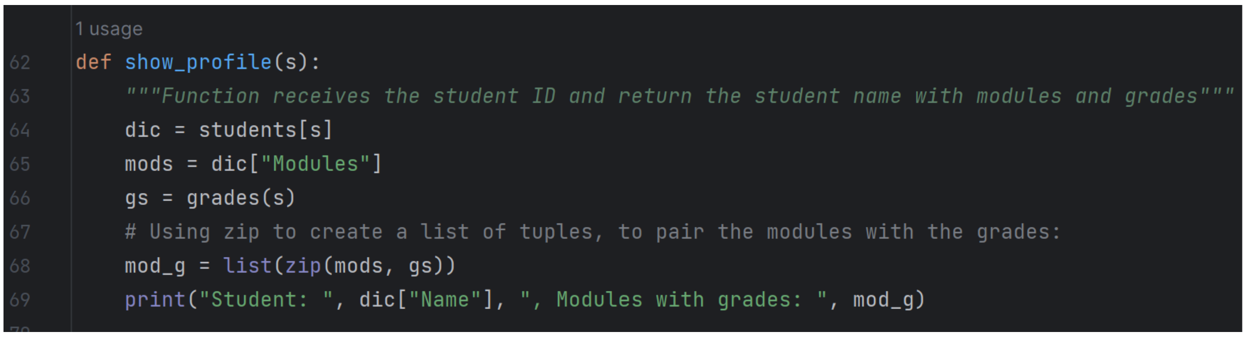
Figure 8. Function to get the list of grades of a student.

Then we can make the function to show the profile, with the grades of the student.



Show profile (s)
s = student ID
Get the student's information from the students dictionary
Get the list of modules of the student (s)
Get the list of grades of the student (s)
Pair the list of modules with the grades
Return: Student name, modules and grades

Figure 9. Flow chart of function to show the profile of a student.



```
1 usage
def show_profile(s):
    """Function receives the student ID and return the student name with modules and grades"""
    dic = students[s]
    mods = dic["Modules"]
    gs = grades(s)
    # Using zip to create a list of tuples, to pair the modules with the grades:
    mod_g = list(zip(mods, gs))
    print("Student: ", dic["Name"], ", Modules with grades: ", mod_g)
```

Figure 10. Function to show the profile of a student.

Now, to ask for the student ID and what action to perform:

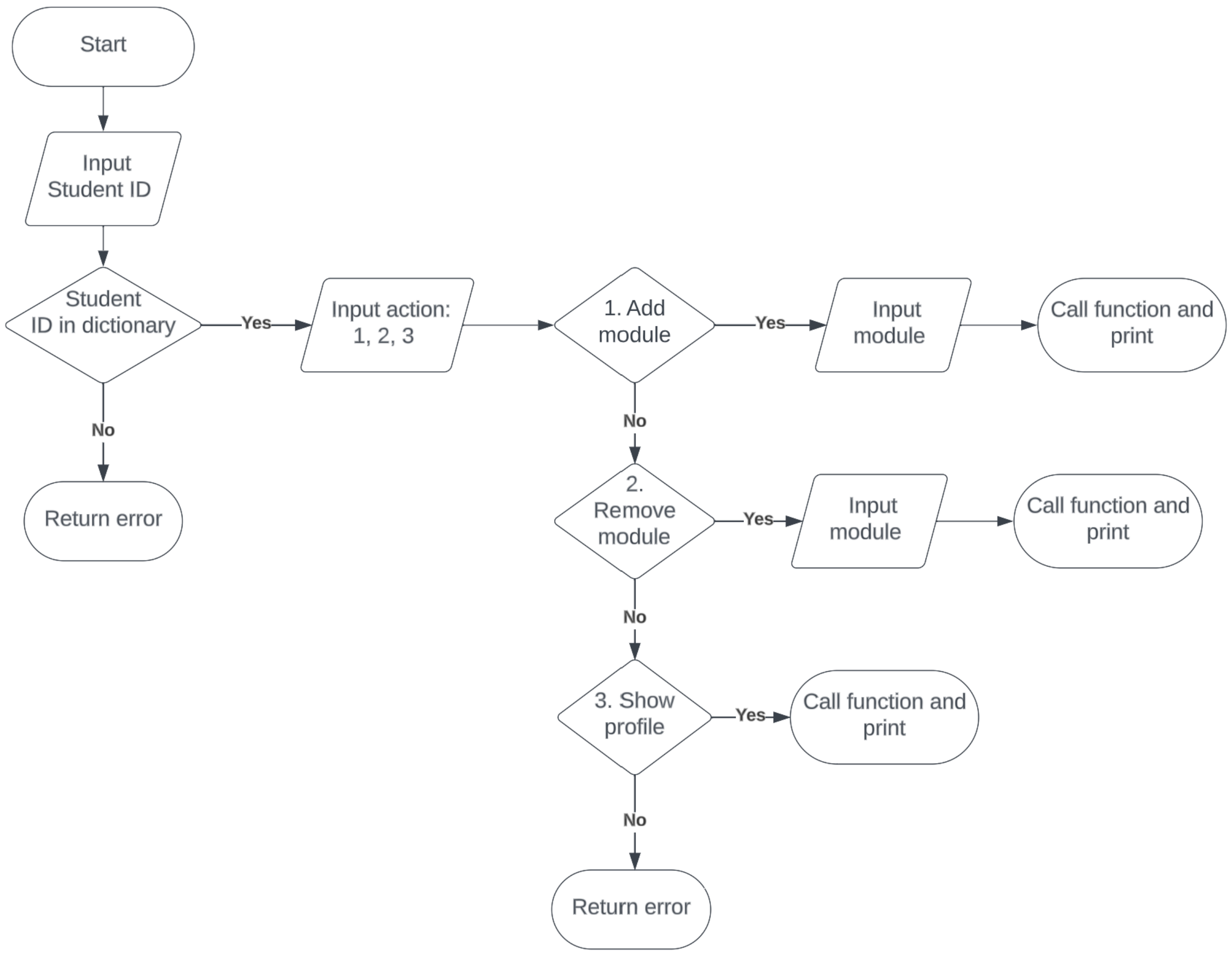
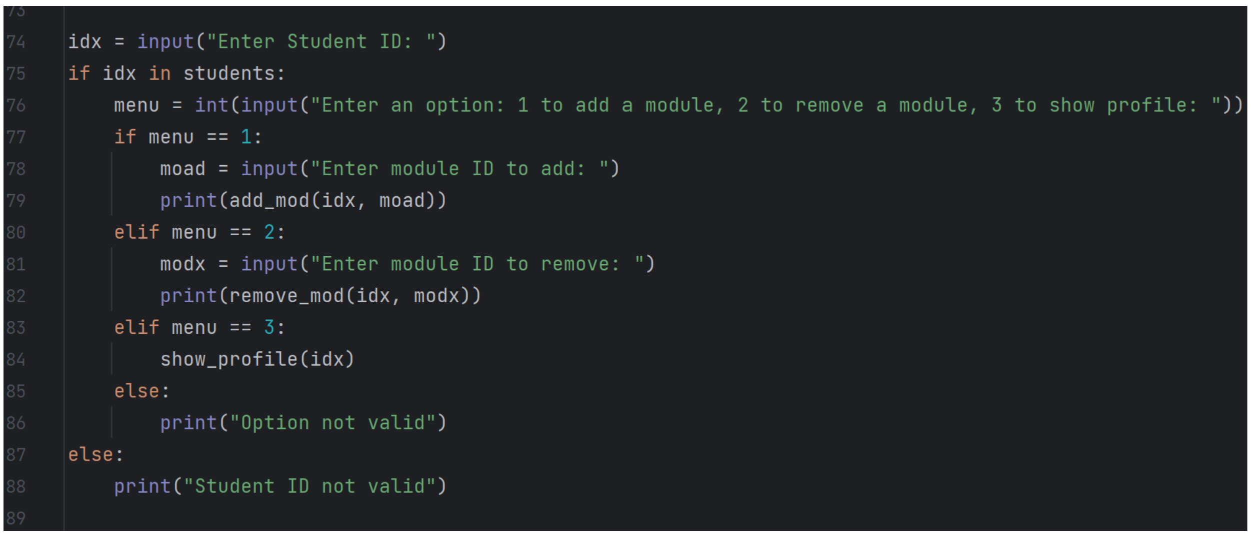


Figure 11. Flow chart for asking the action and call the functions.

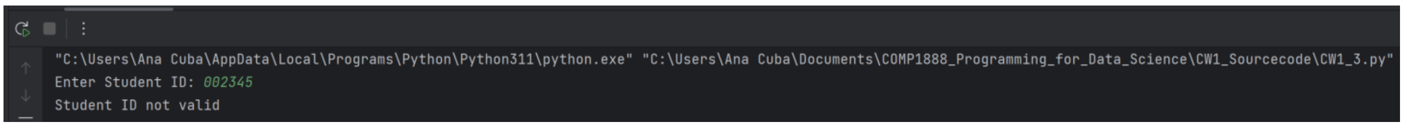
And this in python:



```
idx = input("Enter Student ID: ")
if idx in students:
    menu = int(input("Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: "))
    if menu == 1:
        moad = input("Enter module ID to add: ")
        print(add_mod(idx, moad))
    elif menu == 2:
        modx = input("Enter module ID to remove: ")
        print(remove_mod(idx, modx))
    elif menu == 3:
        show_profile(idx)
    else:
        print("Option not valid")
else:
    print("Student ID not valid")
```

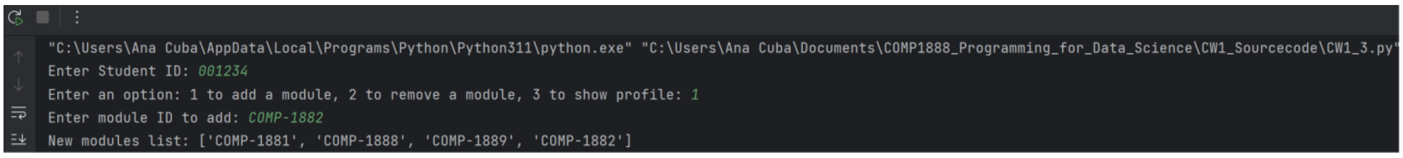
Figure 12. Code to input the Student ID and the action to perform.

Results for the first part:



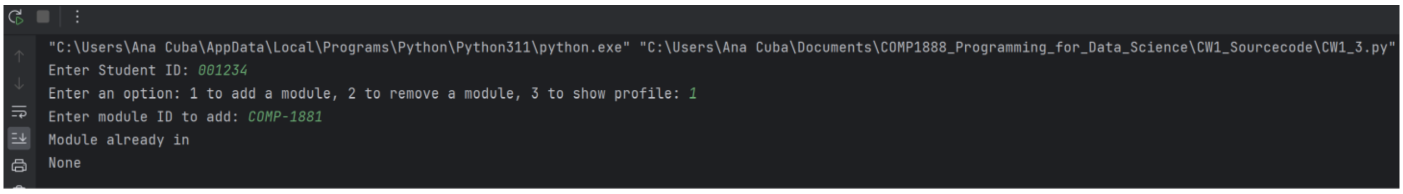
```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 002345
Student ID not valid
```

Figure 13. Running the program with wrong student ID.



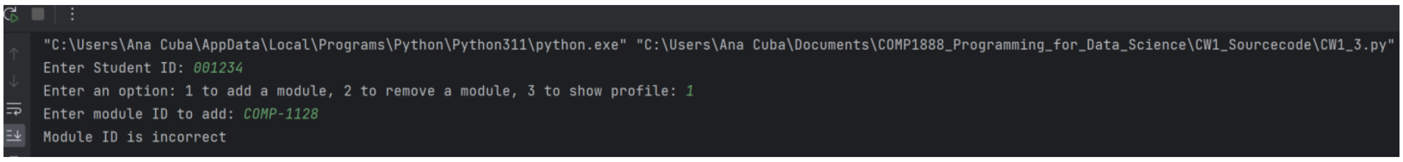
```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 001234
Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: 1
Enter module ID to add: COMP-1882
New modules list: ['COMP-1881', 'COMP-1888', 'COMP-1889', 'COMP-1882']
```

Figure 14. Running the program and choosing to add a module.



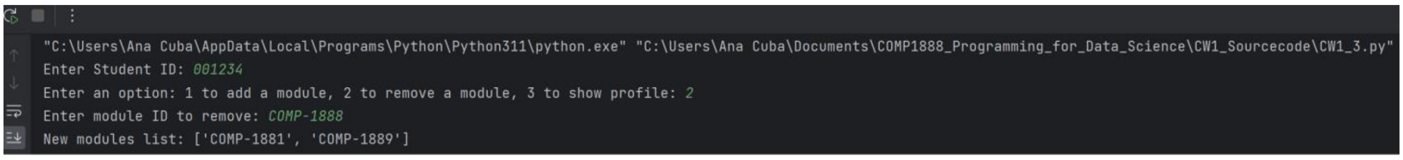
```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 001234
Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: 1
Enter module ID to add: COMP-1881
Module already in
None
```

Figure 15. Running the program, choosing to add a module and entering a module already in.



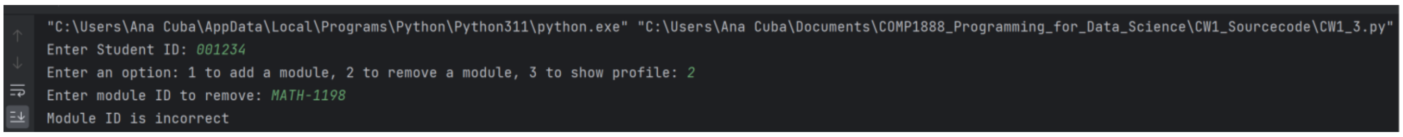
```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 001234
Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: 1
Enter module ID to add: COMP-1128
Module ID is incorrect
```

Figure 16. Running a program, choosing a module not in the module dictionary to add.



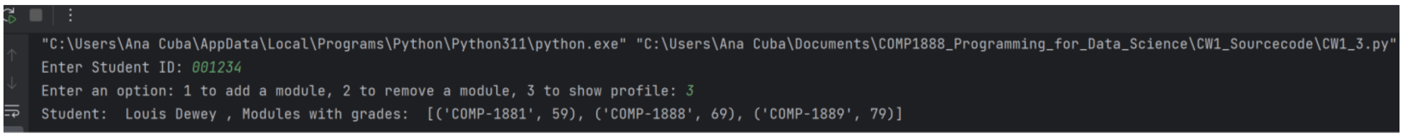
```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 001234
Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: 2
Enter module ID to remove: COMP-1888
New modules list: ['COMP-1881', 'COMP-1889']
```

Figure 17. Running the program and choosing to remove a module.



```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 001234
Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: 2
Enter module ID to remove: MATH-1198
Module ID is incorrect
```

Figure 18. Running the program and choosing a module that is not in the student modules list to remove.



```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 001234
Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: 3
Student:  Louis Dewey , Modules with grades:  [('COMP-1881', 59), ('COMP-1888', 69), ('COMP-1889', 79)]
```

Figure 19. Running the program and choosing to see the profile.

SECOND PART

We are required to identify the students with the highest and lowest average marks. For this I already have a function that get a list of marks of a student. So, I need to do this for every student and with that list of grades get the average for each student.

1. Get the list of marks of every student.
2. Define a function to get the average mark for every student.
3. Create a list of average marks.
4. Find the highest and lowest value of that list.
5. Show the students who got those marks.

So, the logic for this part is as follows:

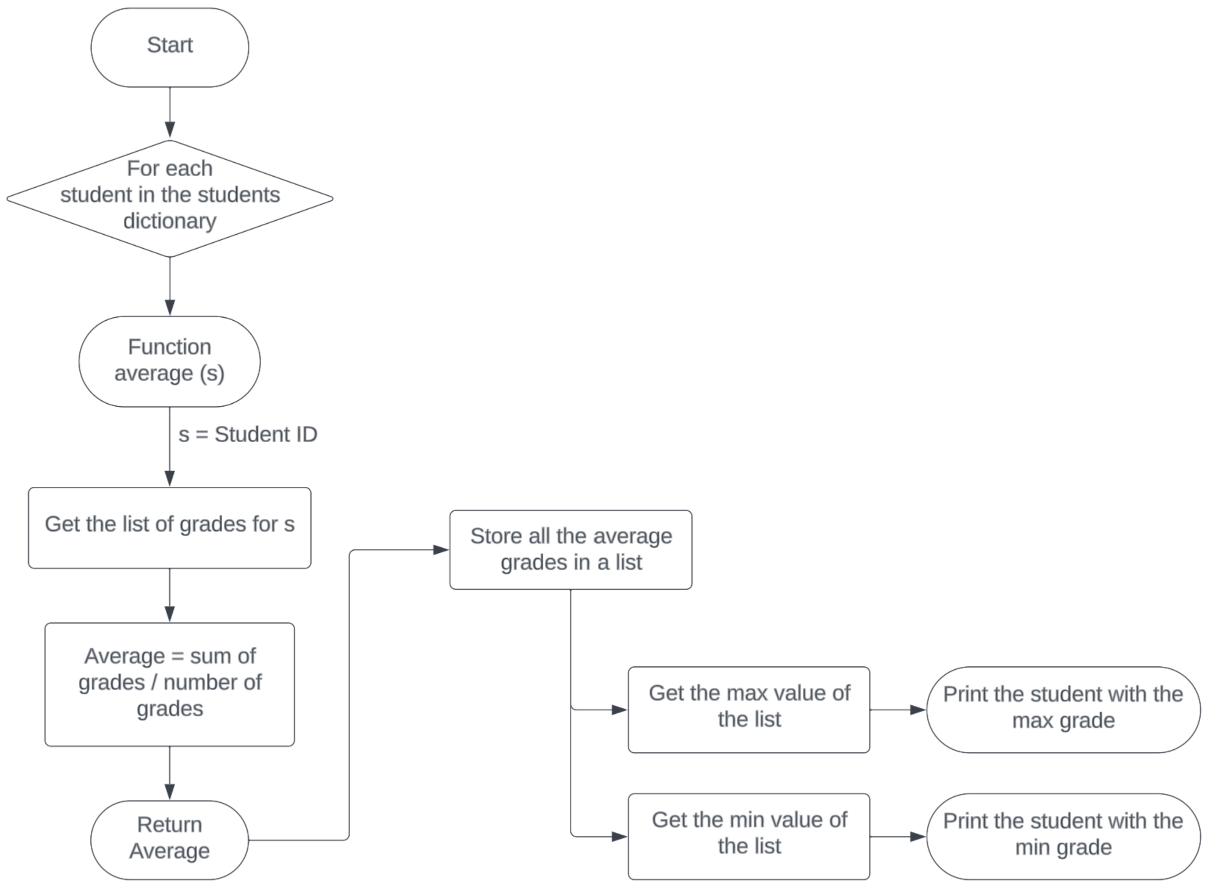
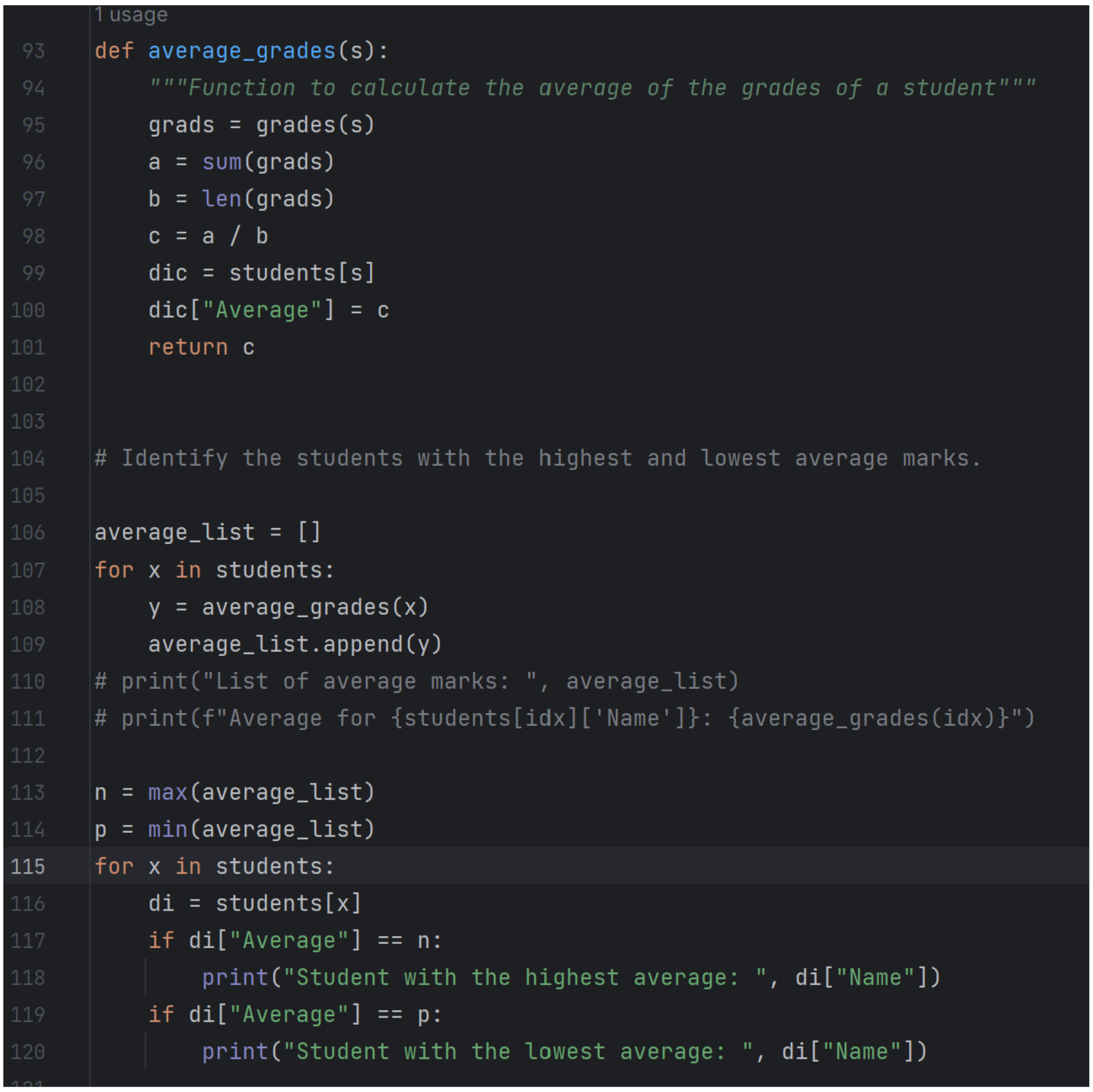

Start
For each student in the students dictionary
Function average (s)
s = Student ID
Get the list of grades for s
Average = sum of grades / number of grades
Return Average
Store all the average grades in a list
Get the max value of the list
Print the student with the max grade
Get the min value of the list
Print the student with the min grade

Figure 20. Flow chart to get the student with the highest and lowest average marks.

And the program in python:



```python
1 usage
def average_grades(s):
    """Function to calculate the average of the grades of a student"""
    grads = grades(s)
    a = sum(grads)
    b = len(grads)
    c = a / b
    dic = students[s]
    dic["Average"] = c
    return c


# Identify the students with the highest and lowest average marks.

average_list = []
for x in students:
    y = average_grades(x)
    average_list.append(y)
# print("List of average marks: ", average_list)
# print(f"Average for {students[idx]['Name']}: {average_grades(idx)}")

n = max(average_list)
p = min(average_list)
for x in students:
    di = students[x]
    if di["Average"] == n:
        print("Student with the highest average: ", di["Name"])
    if di["Average"] == p:
        print("Student with the lowest average: ", di["Name"])
```

Figure 21. Code to show the students with the highest and lowest average.

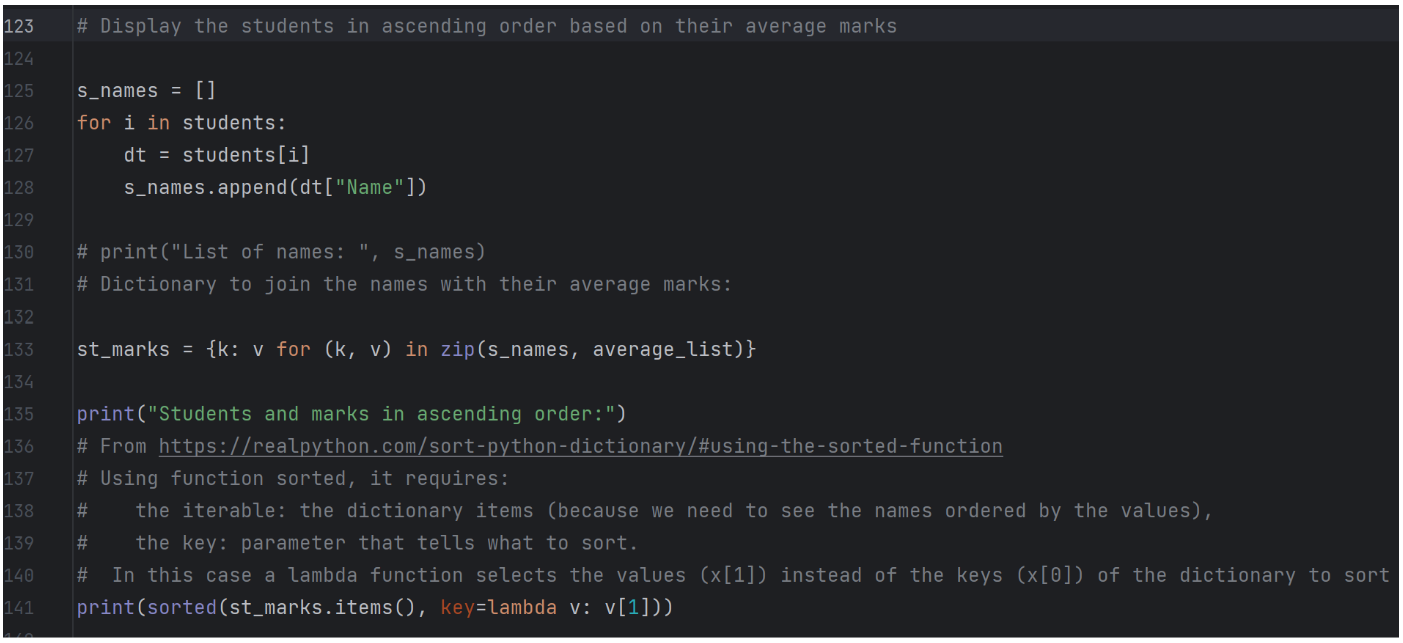
Then we are required to show the students in order based in their average marks.

The logic for this part:

1. Get the list with the average marks of the students, which I already got in the last part.
2. Get the list of the names of the students.
3. Pair the students' names with their average marks. I used a zip function in a dictionary comprehension.
4. Sort the pairs by the average marks. I used the function sorted.

This part of the program will execute each time we run the program.

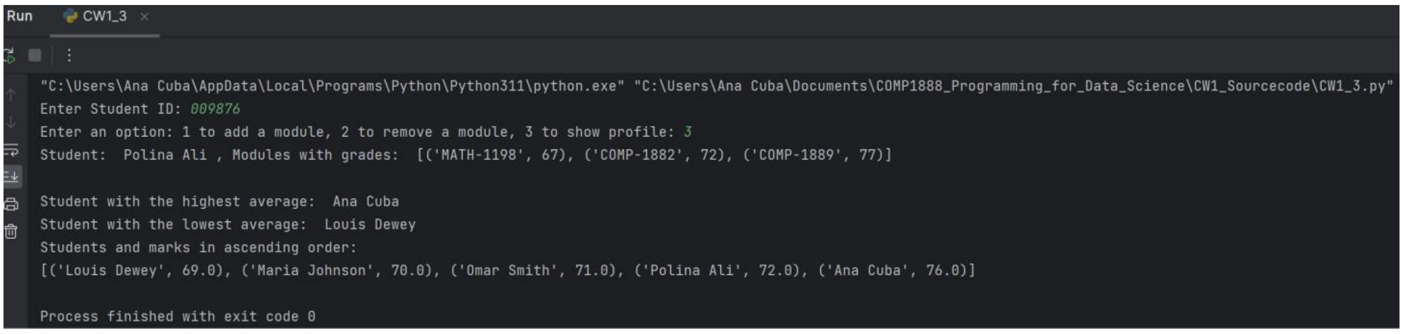
And this part of the program in python:



```python
# Display the students in ascending order based on their average marks

s_names = []
for i in students:
    dt = students[i]
    s_names.append(dt["Name"])

# print("List of names: ", s_names)
# Dictionary to join the names with their average marks:

st_marks = {k: v for (k, v) in zip(s_names, average_list)}

print("Students and marks in ascending order:")
# From https://realpython.com/sort-python-dictionary/#using-the-sorted-function
# Using function sorted, it requires:
#    the iterable: the dictionary items (because we need to see the names ordered by the values),
#    the key: parameter that tells what to sort.
#  In this case a lambda function selects the values (x[1]) instead of the keys (x[0]) of the dictionary to sort
print(sorted(st_marks.items(), key=lambda v: v[1]))
```

Figure 22. Code to show the list of students sorted by average marks.

The results with the second part:



```
"C:\Users\Ana Cuba\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\Ana Cuba\Documents\COMP1888_Programming_for_Data_Science\CW1_Sourcecode\CW1_3.py"
Enter Student ID: 009876
Enter an option: 1 to add a module, 2 to remove a module, 3 to show profile: 3
Student:  Polina Ali , Modules with grades:  [('MATH-1198', 67), ('COMP-1882', 72), ('COMP-1889', 77)]

Student with the highest average:  Ana Cuba
Student with the lowest average:  Louis Dewey
Students and marks in ascending order:
[('Louis Dewey', 69.0), ('Maria Johnson', 70.0), ('Omar Smith', 71.0), ('Polina Ali', 72.0), ('Ana Cuba', 76.0)]

Process finished with exit code 0
```

Figure 23. Running the program.

2.2 Code that you wrote

```
# Dictionary of modules records:
modules = {"MATH-1198": {"Title": "Mathematics",
                         "Marks": {"007084": 78, "005678": 81, "009876": 67}},
           "COMP-1882": {"Title": "Ethics",
                         "Marks": {"007084": 70, "009876": 72, "005432": 64}},
           "COMP-1888": {"Title": "Programming",
                         "Marks": {"007084": 80, "001234": 69, "005678": 61}},
           "COMP-1889": {"Title": "Research",
                         "Marks": {"001234": 79, "009876": 77, "005432": 76}},
           "COMP-1887": {"Title": "Principles of Data Science",
                         "Marks": {"009876": 65, "005432": 64, "005678": 71}},
           "COMP-1881": {"Title": "Data Visualisation",
                         "Marks": {"001234": 59, "005432": 70, "005678": 76}}}

# Dictionary of students records:
students = {"007084": {"Name": "Ana Cuba",
                       "Modules": ["MATH-1198", "COMP-1882", "COMP-1888"]},
            "001234": {"Name": "Louis Dewey",
                       "Modules": ["COMP-1881", "COMP-1888", "COMP-1889"]},
            "005678": {"Name": "Omar Smith",
                       "Modules": ["MATH-1198", "COMP-1888", "COMP-1887"]},
            "009876": {"Name": "Polina Ali",
                       "Modules": ["MATH-1198", "COMP-1882", "COMP-1889"]},
            "005432": {"Name": "Maria Johnson",
                       "Modules": ["COMP-1882", "COMP-1881", "COMP-1889"]}}


def add_mod(s, a):
    """Function that add a module. Arguments: student ID(s) and module ID(a)"""
    dic = students[s]
    if a in modules:
        if a in dic["Modules"]:
            print("Module already in")
        else:
            dic["Modules"].append(a)
            return f"New modules list: {dic['Modules']}"
    else:
        return "Module ID is incorrect"


def remove_mod(s, a):
    """Function that remove a module. Arguments: student ID(s) and module
ID(a)"""
    dic = students[s]
    if a in dic["Modules"]:
        dic["Modules"].remove(a)
        return f"New modules list: {dic['Modules']}"
    else:
        return "Module ID is incorrect"
```

```
def grades(s):
    """Function that receives the ID of a student and returns a list with their
grades"""
    mods = students[s]["Modules"]
    glist = []
    for e in mods:
        m = modules[e]["Marks"]
        if s in m.keys():
            glist.append(m[s])
    return glist


def show_profile(s):
    """Function receives the student ID and return the student name with
modules and grades"""
    dic = students[s]
    mods = dic["Modules"]
    gs = grades(s)
    # Using zip to create a list of tuples, to pair the modules with the
grades:
    mod_g = list(zip(mods, gs))
    print("Student: ", dic["Name"], ", Modules with grades: ", mod_g)


# Menu to ask user what function to do:

idx = input("Enter Student ID: ")
if idx in students:
    menu = int(input("Enter an option: 1 to add a module, 2 to remove a module,
3 to show profile: "))
    if menu == 1:
        moad = input("Enter module ID to add: ")
        print(add_mod(idx, moad))
    elif menu == 2:
        modx = input("Enter module ID to remove: ")
        print(remove_mod(idx, modx))
    elif menu == 3:
        show_profile(idx)
    else:
        print("Option not valid")
else:
    print("Student ID not valid")

print()


def average_grades(s):
    """Function to calculate the average of the grades of a student"""
    grads = grades(s)
    a = sum(grads)
    b = len(grads)
    c = a / b
    dic = students[s]
    dic["Average"] = c
    return c


# Identify the students with the highest and lowest average marks.

average_list = []
for x in students:
    y = average_grades(x)
    average_list.append(y)
# print("List of average marks: ", average_list)
# print(f"Average for {students[idx]['Name']}: {average_grades(idx)}")

n = max(average_list)
p = min(average_list)
for x in students:
    di = students[x]
    if di["Average"] == n:
        print("Student with the highest average: ", di["Name"])
    if di["Average"] == p:
        print("Student with the lowest average: ", di["Name"])


# Display the students in ascending order based on their average marks

s_names = []
for i in students:
    dt = students[i]
    s_names.append(dt["Name"])

# print("List of names: ", s_names)
# Dictionary to join the names with their average marks:

st_marks = {k: v for (k, v) in zip(s_names, average_list)}

print("Students and marks in ascending order:")
# From https://realpython.com/sort-python-dictionary/#using-the-sorted-function
# Using function sorted, it requires:
#    the iterable: the dictionary items (because we need to see the names
ordered by the values),
#    the key: parameter that tells what to sort.
#  In this case a lambda function selects the values (x[1]) instead of the keys
(x[0]) of the dictionary to sort
print(sorted(st_marks.items(), key=lambda v: v[1]))
```