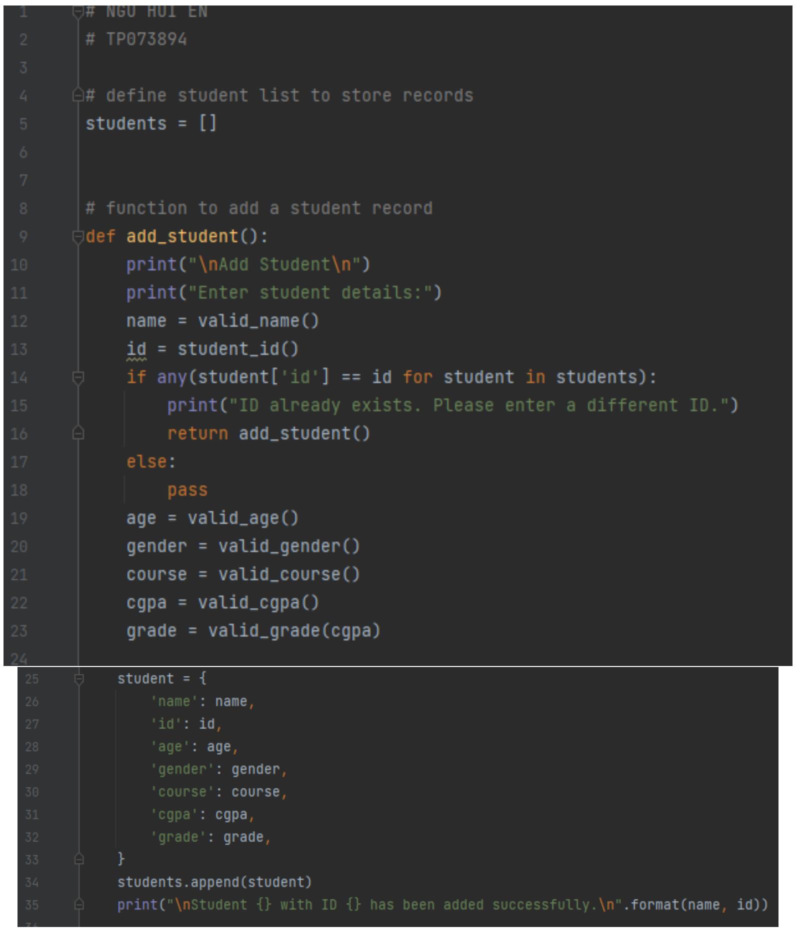
Author: Ngu Hui En

1.1 Overview

This student management system is a program for managing student records. It allows the user to add, delete, search, update, and view all student records. In general, the program asks for the student's name, student ID, age, gender, course name and CGPA, and calculates the degree classification based on the CGPA. The records are stored in a list of dictionaries. The name and gender were stored as string while ID and age were stored as integers. The CGPA was stored as float. Student ID is unique to each student.

The program first defines an empty list called “students” to store the records. It then defines several functions for prompting the user to enter student details, validating the input data and performing the various operations on the records. The “add_student” function is used to add a student record. The programming concept used is functions, conditionals, loops and dictionaries. It starts by prompting the user to enter the student's details, validates the input data, checks if the ID already exists in the list, calculates the degree classification based on the CGPA, stores it in a dictionary and adds the record to the “students” list (Figure 1.1, Figure 1.2). Finally, the function prints a success message with the student’s name and ID.

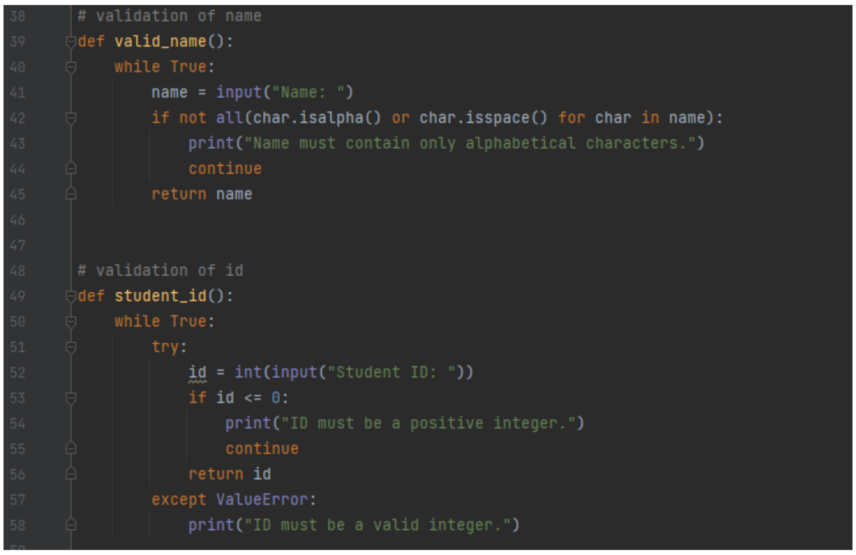


```
# NGU HUI EN
# TP073894

# define student list to store records
students = []


# function to add a student record
def add_student():
    print("\nAdd Student\n")
    print("Enter student details:")
    name = valid_name()
    id = student_id()
    if any(student['id'] == id for student in students):
        print("ID already exists. Please enter a different ID.")
        return add_student()
    else:
        pass
    age = valid_age()
    gender = valid_gender()
    course = valid_course()
    cgpa = valid_cgpa()
    grade = valid_grade(cgpa)

    student = {
        'name': name,
        'id': id,
        'age': age,
        'gender': gender,
        'course': course,
        'cgpa': cgpa,
        'grade': grade,
    }
    students.append(student)
    print("\nStudent {} with ID {} has been added successfully.\n".format(name, id))
```

Figure 1.1. The python code for defining the function of adding students.

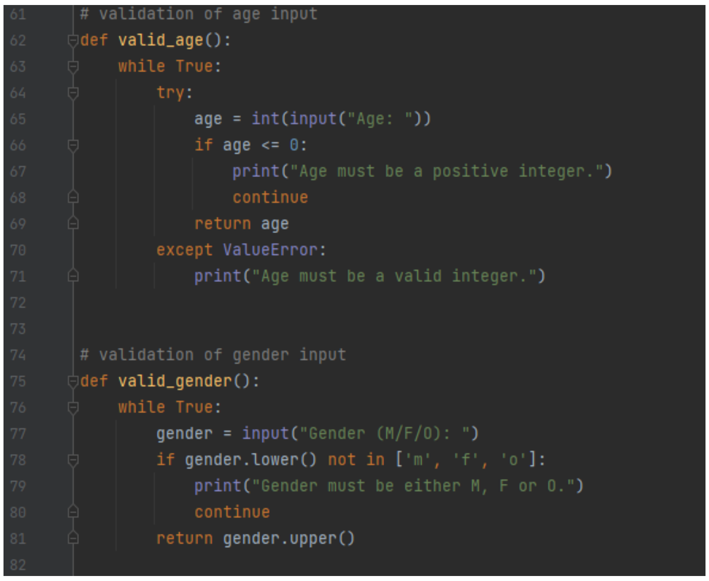
Validation of student records was performed under different functions. The "valid_name" function validates the name input contains only alphabetical characters (Figure 1.2). The "student_id" function validates the ID input to ensure it is a valid positive integer (Figure 1.3). The "valid_age" function validates the age input to ensure it is a valid positive integer(Figure 1.3). The "valid_gender" function validates the gender input to ensure it is either M, F, or O (Figure 1.4). The "valid_course" function validates the course name input contains only alphabetical characters. The "valid_cgpa" function validates the CGPA input to ensure it is a valid float between 0.00 and 4.00 (Figure 1.4). The "valid_grade" function calculates the degree classification based on the CGPA (Figure 1.5).

The programming concept used in this code is functions and exception handling. Each validation function uses a while loop to continuously prompt the user to enter a valid input until the condition is met. If the input is not valid, the loop continues and the user is prompted again. When a valid input is entered, the function returns the input value. The exception handling concept is used in the "student_id()", "valid_age()", and "valid_cgpa()" functions to handle errors that may occur if the user enters an invalid input such as a string instead of an integer or a float. If an error occurs, the function prints an error message and prompts the user to enter a valid input again. Finally, the "valid_grade()" function defines the grade based on the CGPA value passed to it as a parameter. It uses conditional statements to determine the grade based on the range of CGPA.



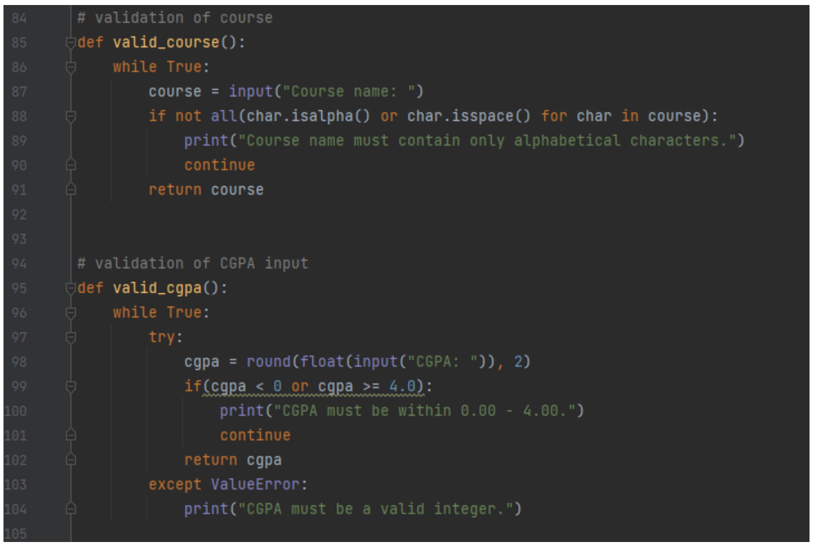
```python
# validation of name
def valid_name():
    while True:
        name = input("Name: ")
        if not all(char.isalpha() or char.isspace() for char in name):
            print("Name must contain only alphabetical characters.")
            continue
        return name


# validation of id
def student_id():
    while True:
        try:
            id = int(input("Student ID: "))
            if id <= 0:
                print("ID must be a positive integer.")
                continue
            return id
        except ValueError:
            print("ID must be a valid integer.")
```

Figure 1.2. The python code for the validation of name and ID.



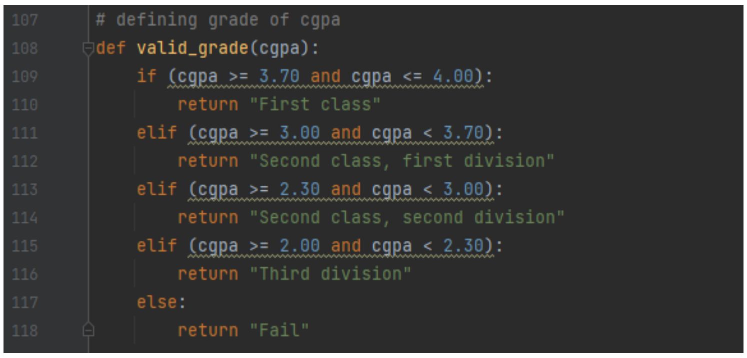
```python
# validation of age input
def valid_age():
    while True:
        try:
            age = int(input("Age: "))
            if age <= 0:
                print("Age must be a positive integer.")
                continue
            return age
        except ValueError:
            print("Age must be a valid integer.")


# validation of gender input
def valid_gender():
    while True:
        gender = input("Gender (M/F/O): ")
        if gender.lower() not in ['m', 'f', 'o']:
            print("Gender must be either M, F or O.")
            continue
        return gender.upper()

```

Figure 1.3. The python code for the validation of age and gender.



```python
# validation of course
def valid_course():
    while True:
        course = input("Course name: ")
        if not all(char.isalpha() or char.isspace() for char in course):
            print("Course name must contain only alphabetical characters.")
            continue
        return course


# validation of CGPA input
def valid_cgpa():
    while True:
        try:
            cgpa = round(float(input("CGPA: ")), 2)
            if(cgpa < 0 or cgpa >= 4.0):
                print("CGPA must be within 0.00 - 4.00.")
                continue
            return cgpa
        except ValueError:
            print("CGPA must be a valid integer.")

```

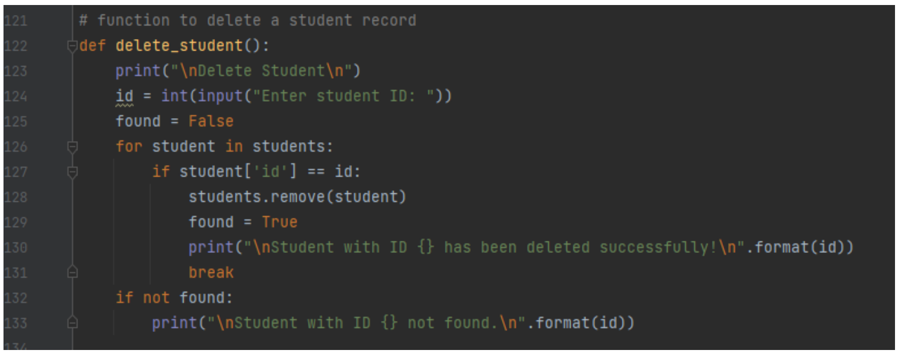
Figure 1.4. The python code for the validation of course and CGPA.



```python
# defining grade of cgpa
def valid_grade(cgpa):
    if (cgpa >= 3.70 and cgpa <= 4.00):
        return "First class"
    elif (cgpa >= 3.00 and cgpa < 3.70):
        return "Second class, first division"
    elif (cgpa >= 2.30 and cgpa < 3.00):
        return "Second class, second division"
    elif (cgpa >= 2.00 and cgpa < 2.30):
        return "Third division"
    else:
        return "Fail"
```

Figure 1.5. The python code for defining the grade of CGPA.

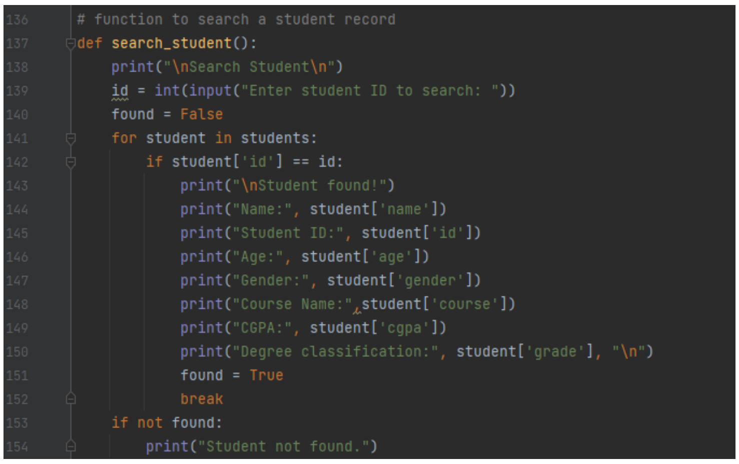
Next, the "delete_student" function prompts the user to enter the student's ID to delete the record, searches for the record in the "students" list, and deletes the record if found (Figure 1.6). If no match is found, the function will print a message indicating that the student with the entered ID was not found. The programming concept used in this code is iteration, conditional statements and input/output operations. The "for" loop used for iteration and "if" statement used for conditional checking. The "input()" and "print()" functions are used to prompt the user for input and output the result of the operation.



```python
# function to delete a student record
def delete_student():
    print("\nDelete Student\n")
    id = int(input("Enter student ID: "))
    found = False
    for student in students:
        if student['id'] == id:
            students.remove(student)
            found = True
            print("\nStudent with ID {} has been deleted successfully!\n".format(id))
            break
    if not found:
        print("\nStudent with ID {} not found.\n".format(id))
```

Figure 1.6. The python code for defining the deletion of student record.

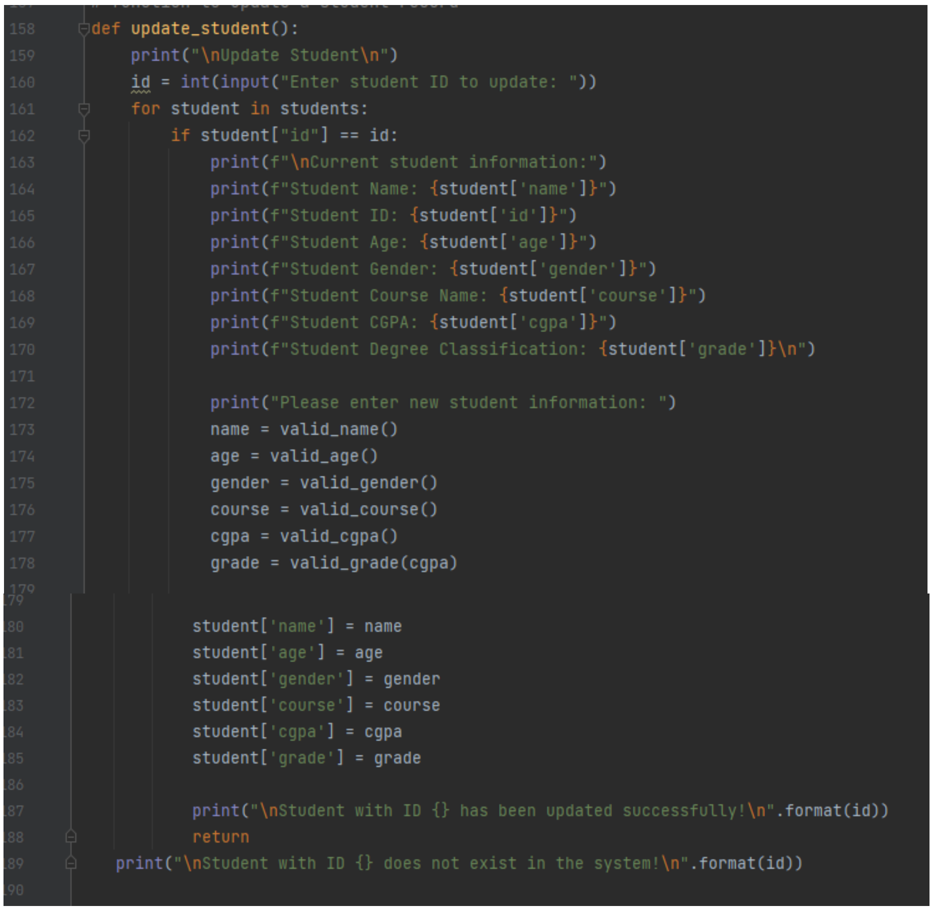
The "search_student()" function prompts the user to enter the student's ID to search for the record, searches for the record in the "students" list, and displays the record if found (Figure 1.7). If no match is found, the function prints a message indicating that the student with the entered ID was not found. The programming concept used in this code is the same as delete function which are iteration, conditional statements, and input/output operations.



```python
# function to search a student record
def search_student():
    print("\nSearch Student\n")
    id = int(input("Enter student ID to search: "))
    found = False
    for student in students:
        if student['id'] == id:
            print("\nStudent found!")
            print("Name:", student['name'])
            print("Student ID:", student['id'])
            print("Age:", student['age'])
            print("Gender:", student['gender'])
            print("Course Name:",student['course'])
            print("CGPA:", student['cgpa'])
            print("Degree classification:", student['grade'], "\n")
            found = True
            break
    if not found:
        print("Student not found.")
```

Figure 1.7. The python code for defining the searching of student records.

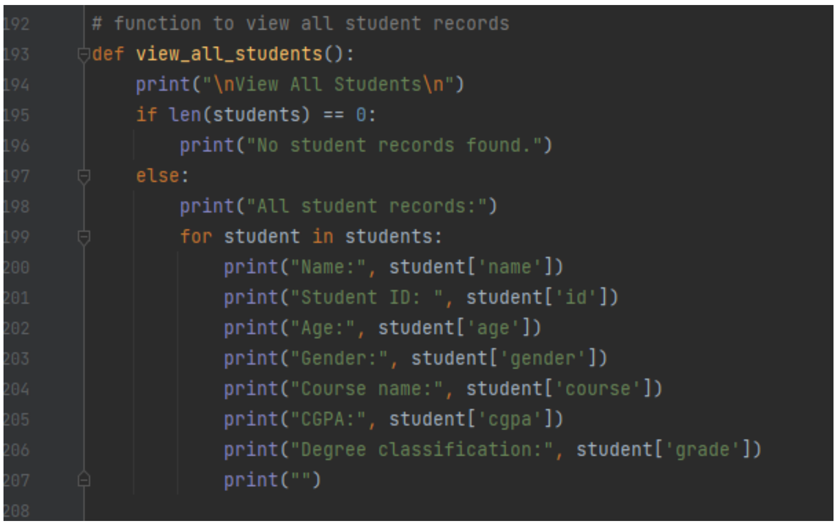
The “update_student” function prompts the user to enter the student's ID to update the record. If the ID exists in the “students” list, it will display the current record, prompt the user to enter the new record details, validate the input data, calculate the degree classification based on the CGPA, and update the record (Figure 1.8). The function calls the following validation functions to ensure that the new information entered by the user is valid: valid_name(), valid_age(), valid_gender(), valid_course(), valid_cgpa(), and valid_grade(). If the ID entered by the user does not exist in the list, the function prints a message indicating that the student does not exist. Overall, the programming concepts used in this code are functions, lists, dictionaries, and loops. The function update_student() utilises these concepts to modify the information of a student in the list of dictionaries students. The use of validation functions also ensures that the user inputs valid data for each field.



```python
def update_student():
    print("\nUpdate Student\n")
    id = int(input("Enter student ID to update: "))
    for student in students:
        if student["id"] == id:
            print(f"\nCurrent student information:")
            print(f"Student Name: {student['name']}")
            print(f"Student ID: {student['id']}")
            print(f"Student Age: {student['age']}")
            print(f"Student Gender: {student['gender']}")
            print(f"Student Course Name: {student['course']}")
            print(f"Student CGPA: {student['cgpa']}")
            print(f"Student Degree Classification: {student['grade']}\n")

            print("Please enter new student information: ")
            name = valid_name()
            age = valid_age()
            gender = valid_gender()
            course = valid_course()
            cgpa = valid_cgpa()
            grade = valid_grade(cgpa)

            student['name'] = name
            student['age'] = age
            student['gender'] = gender
            student['course'] = course
            student['cgpa'] = cgpa
            student['grade'] = grade

            print("\nStudent with ID {} has been updated successfully!\n".format(id))
            return
    print("\nStudent with ID {} does not exist in the system!\n".format(id))
```

Figure 1.8. The python code for defining the update of the student record.

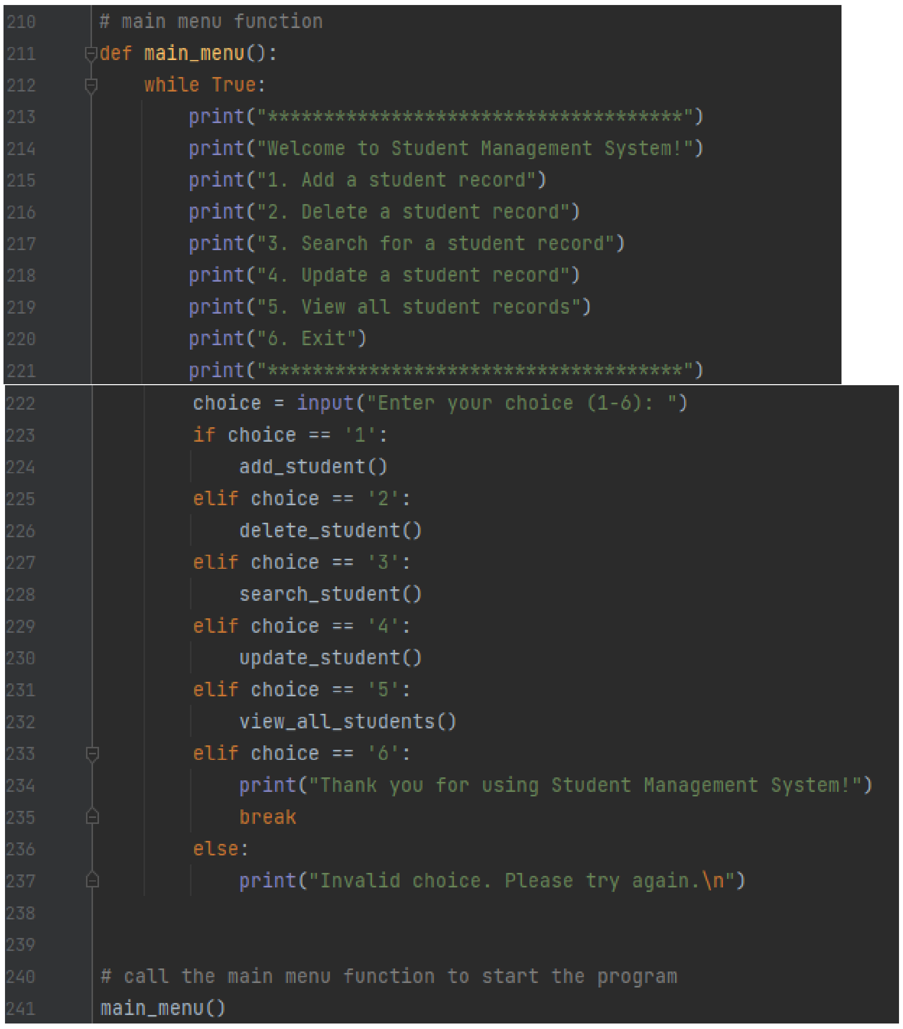
The “view_all_students” function displays all the records stored in the “students” list (Figure 1.9). It uses a loop to iterate over all the elements in the students list and prints the details of each student using the print() function. The len() function is used to check if the students list is empty, and if so, it prints a message stating that no student records were found. The function uses string concatenation to combine the student details with the appropriate labels for each field, such as “Name:”, “Student ID:”, “Age:”, “Gender:”, “Course name:”, “CGPA:”, and “Degree classification:”. The print() function is called for each field to display the label and the corresponding value. Finally, an empty string is printed after each student record to add a blank line between them for better readability.



```python
# function to view all student records
def view_all_students():
    print("\nView All Students\n")
    if len(students) == 0:
        print("No student records found.")
    else:
        print("All student records:")
        for student in students:
            print("Name:", student['name'])
            print("Student ID: ", student['id'])
            print("Age:", student['age'])
            print("Gender:", student['gender'])
            print("Course name:", student['course'])
            print("CGPA:", student['cgpa'])
            print("Degree classification:", student['grade'])
            print("")
```

Figure 1.9. The python code for defining the viewing of all student records.

The “main_menu” function displays the menu options for the user to choose which functions to be performed (Figure 1.10). It is a menu-driven program. Based on the user's input, the function calls other functions to perform the required operations, such as adding, deleting, searching, updating, or viewing student records. This process is repeated until the user chooses to exit the program.



```python
# main menu function
def main_menu():
    while True:
        print("*************************************")
        print("Welcome to Student Management System!")
        print("1. Add a student record")
        print("2. Delete a student record")
        print("3. Search for a student record")
        print("4. Update a student record")
        print("5. View all student records")
        print("6. Exit")
        print("*************************************")
        choice = input("Enter your choice (1-6): ")
        if choice == '1':
            add_student()
        elif choice == '2':
            delete_student()
        elif choice == '3':
            search_student()
        elif choice == '4':
            update_student()
        elif choice == '5':
            view_all_students()
        elif choice == '6':
            print("Thank you for using Student Management System!")
            break
        else:
            print("Invalid choice. Please try again.\n")


# call the main menu function to start the program
main_menu()
```

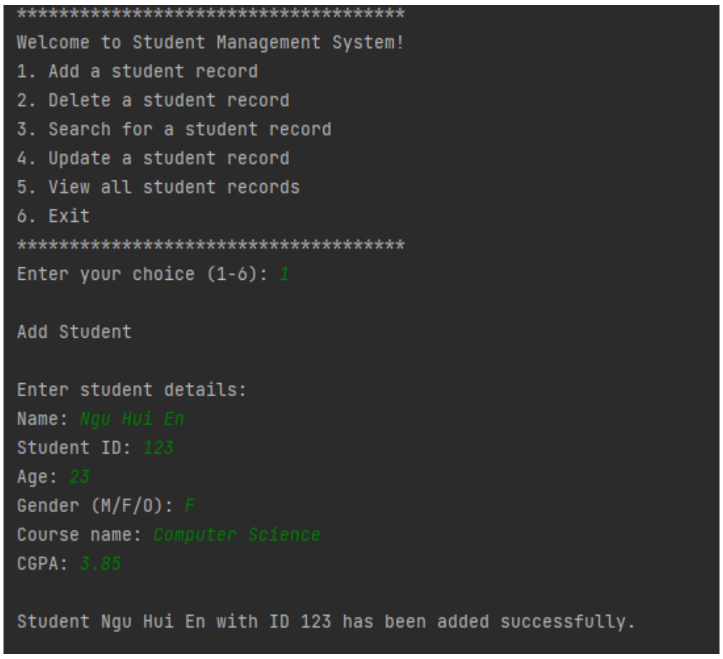
Figure 1.10. The python code for defining the main menu.

1.2 Input/Output with Explanation

Two student records were used to test the input and output of the program. Menu will be displayed every time before asking the use for entering the choice.

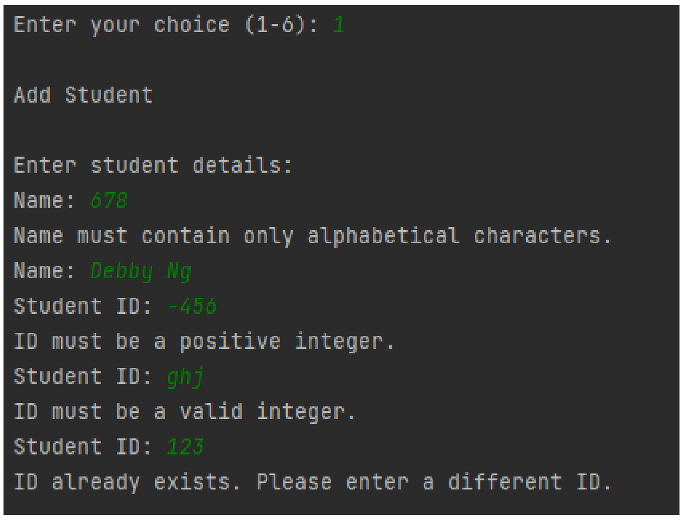
1.2.1 Add and View Student Record

The menu will display everything to ask the user for choice. After entering ‘1’, the adding student record function will be performed. The student’s information on name, student ID, age, gender, course name and CGPA are required to be entered (Figure 1.11). If invalid names such as numbers are inserted, the interface will display invalid message and require the user to enter the valid name again (Figure 1.12). Same goes to the student ID, only positive integers are accepted. Duplication of student ID is not allowed. Figure 1.13 shows the results of entering invalid and valid age, gender, and CGPA under adding student function. The age only accepted positive integers. The input of gender accepted uppercase and lowercase of ‘m’, ‘f’ and ‘o’. If lowercase is imputed, it will return it to uppercase in the list. It does not accept other alphabetical characters and numbers as value. The user is required to enter the value again if invalid input is entered. Course name only accept alphabetical characters. Furthermore, the CGPA value should be within 0.00 to 4.00. If the value inserted is more than 2 decimal points and within the CGPA value, it will round up the value into 2 decimal points. If all the student details inserted are valid, an adding success statement will be shown.



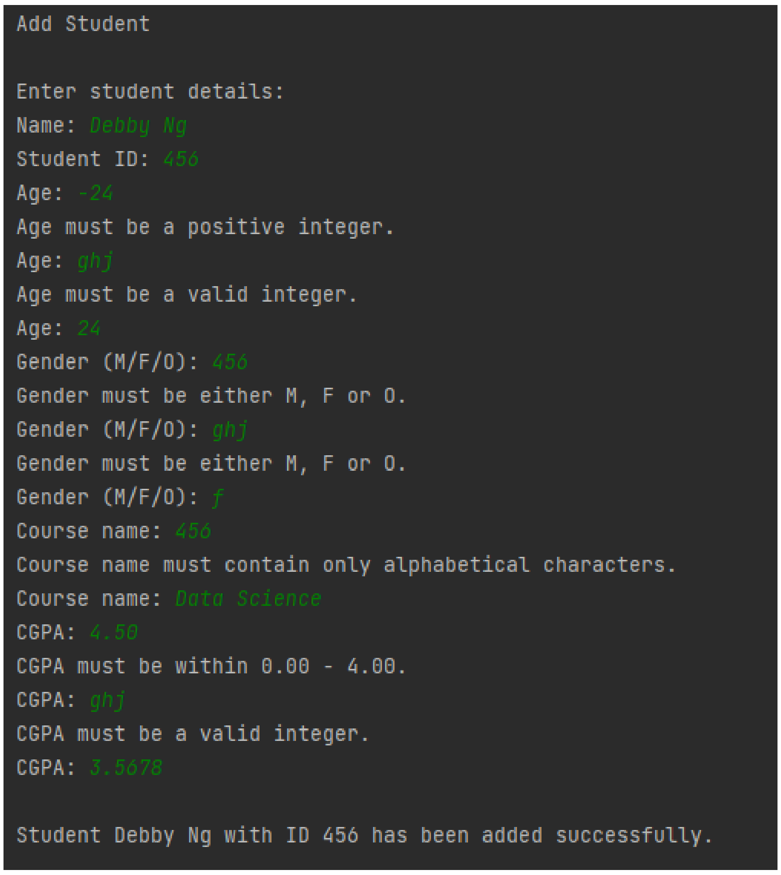
```
*************************************
Welcome to Student Management System!
1. Add a student record
2. Delete a student record
3. Search for a student record
4. Update a student record
5. View all student records
6. Exit
*************************************
Enter your choice (1-6): 1

Add Student

Enter student details:
Name: Ngu Hui En
Student ID: 123
Age: 23
Gender (M/F/O): F
Course name: Computer Science
CGPA: 3.85

Student Ngu Hui En with ID 123 has been added successfully.
```

Figure 1.11. The menu display with the results of valid input after successfully adding the student record.



```
Enter your choice (1-6): 1

Add Student

Enter student details:
Name: 678
Name must contain only alphabetical characters.
Name: Debby Ng
Student ID: -456
ID must be a positive integer.
Student ID: ghj
ID must be a valid integer.
Student ID: 123
ID already exists. Please enter a different ID.
```

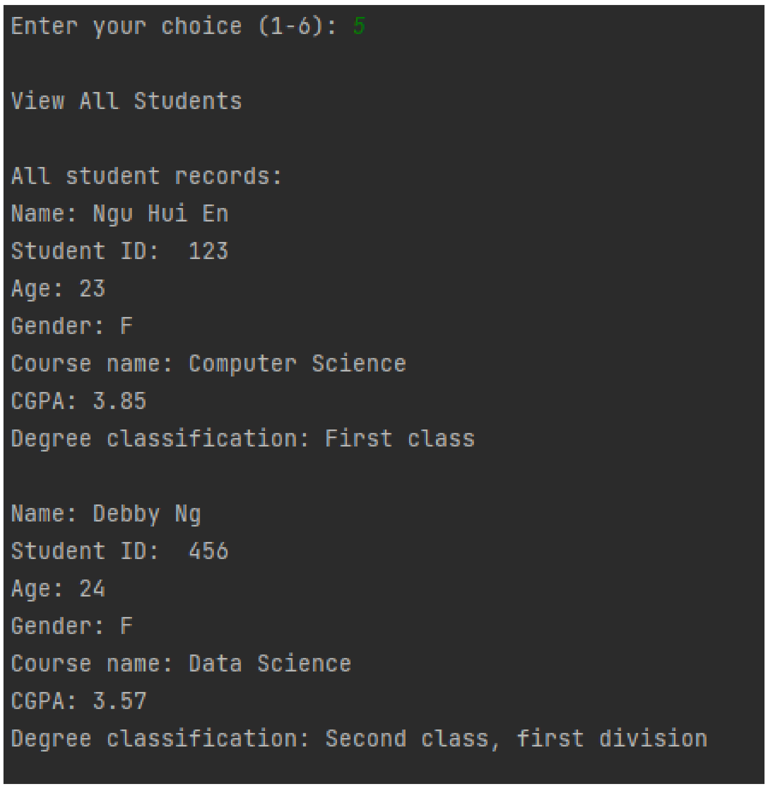
Figure 1.12. The results of entering invalid name and student ID under adding student function.



```
Add Student

Enter student details:
Name: Debby Ng
Student ID: 456
Age: -24
Age must be a positive integer.
Age: ghj
Age must be a valid integer.
Age: 24
Gender (M/F/O): 456
Gender must be either M, F or O.
Gender (M/F/O): ghj
Gender must be either M, F or O.
Gender (M/F/O): f
Course name: 456
Course name must contain only alphabetical characters.
Course name: Data Science
CGPA: 4.50
CGPA must be within 0.00 - 4.00.
CGPA: ghj
CGPA must be a valid integer.
CGPA: 3.5678

Student Debby Ng with ID 456 has been added successfully.
```

Figure 1.13. The results of entering invalid and valid age, gender, course name and CGPA under adding student function.

After entering the ‘5’ as choice, all the student records inserted will be displayed which involve the information on name, student ID, age, gender, CGPA and degree classification (Figure 1.14). The user will then return to the menu.

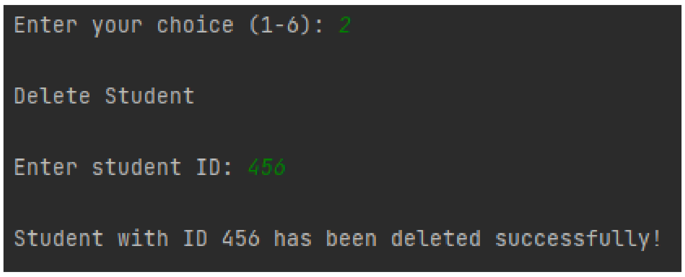


```
Enter your choice (1-6): 5

View All Students

All student records:
Name: Ngu Hui En
Student ID:  123
Age: 23
Gender: F
Course name: Computer Science
CGPA: 3.85
Degree classification: First class

Name: Debby Ng
Student ID:  456
Age: 24
Gender: F
Course name: Data Science
CGPA: 3.57
Degree classification: Second class, first division
```

Figure 1.14. The view all students interface after adding the student records.

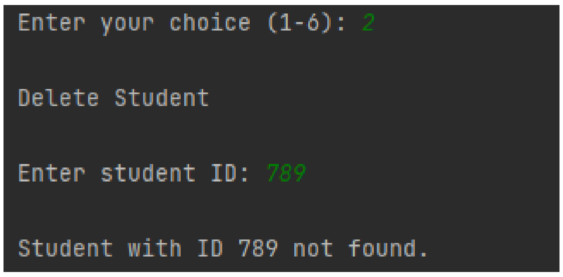
1.2.2 Delete and View Student Record

For choice ‘2’, a student record will be deleted by inserting a valid student ID in the list (Figure 1.15). If invalid student ID is inserted, the interface will show a message that the student is not being found (Figure 1.16). Thus, no student record can be deleted with invalid student ID. The user will then return to the menu and ask the user for choice. After deleting it, the view of all student records will show the existing student records (Figure 1.17).



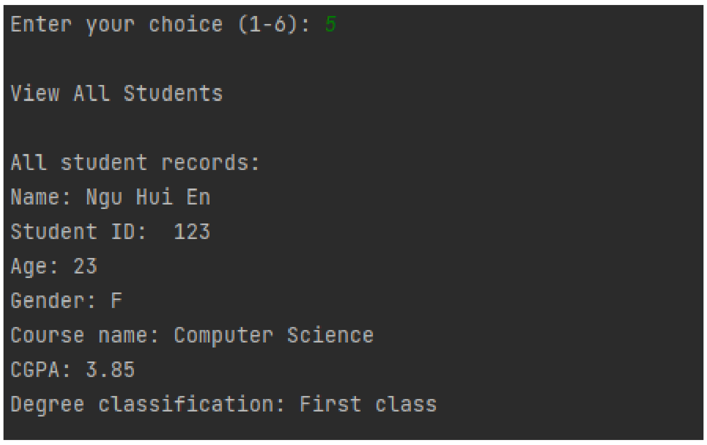
```
Enter your choice (1-6): 2

Delete Student

Enter student ID: 456

Student with ID 456 has been deleted successfully!
```

Figure 1.15. The result of deleting a student successfully by inserting a valid student ID.



```
Enter your choice (1-6): 2

Delete Student

Enter student ID: 789

Student with ID 789 not found.
```

Figure 1.16. The result of deleting a student unsuccessfully by inserting an invalid student ID.

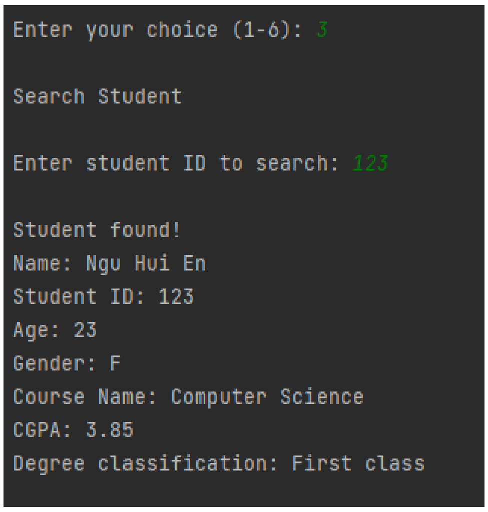


```
Enter your choice (1-6): 5

View All Students

All student records:
Name: Ngu Hui En
Student ID:  123
Age: 23
Gender: F
Course name: Computer Science
CGPA: 3.85
Degree classification: First class
```

Figure 1.17. The viewing of all student records after deleting a student record.

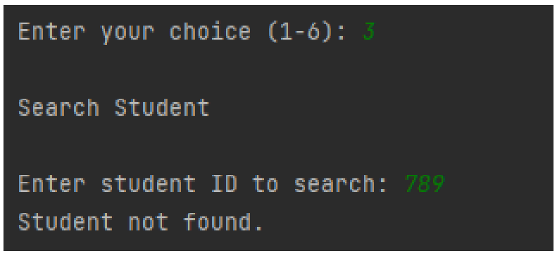
1.2.3 Search Student Record

For choice ‘3’, it will perform the searching of student records. The student records which involve name, student ID, age, gender, course name, CGPA and degree classification will be shown if the student ID inserted is valid and exists (Figure 1.18). “Student not found” message will be displayed if the student ID inserted is invalid and does not exist (Figure 1.19). The screen will return to the menu display.



```
Enter your choice (1-6): 3

Search Student

Enter student ID to search: 123

Student found!
Name: Ngu Hui En
Student ID: 123
Age: 23
Gender: F
Course Name: Computer Science
CGPA: 3.85
Degree classification: First class
```

Figure 1.18. The result of inserting valid student ID to be searched.

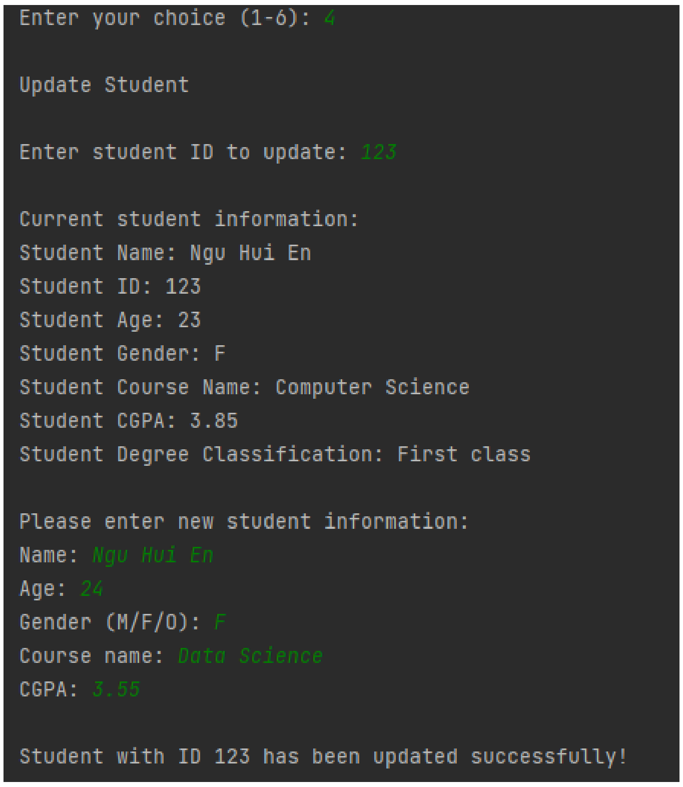


```
Enter your choice (1-6): 3

Search Student

Enter student ID to search: 789
Student not found.
```

Figure 1.19. The result of inserting invalid student ID to be searched.

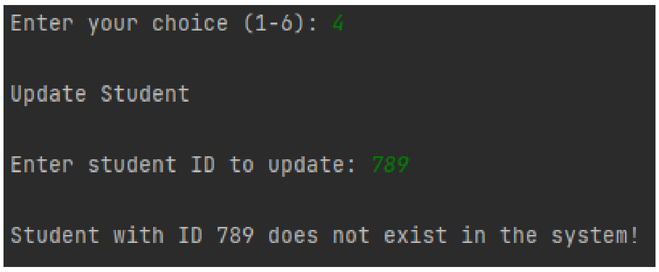
1.2.4 Update and View Student Record

Choice ‘4’ is to update the existing student record with new details except updating the student ID. After adding the valid and existing student ID to be updated, the current information will be displayed and the user is required to enter the updated name, age, gender, course name and CGPA. After successfully adding the valid input, it will update the student record (Figure 1.20). If invalid and non-existing student ID is inserted, it will display a non-existing message to the user (Figure 1.21). The user will then return to the menu. Figure 1.22 shows the updated results of viewing the all of the student records.



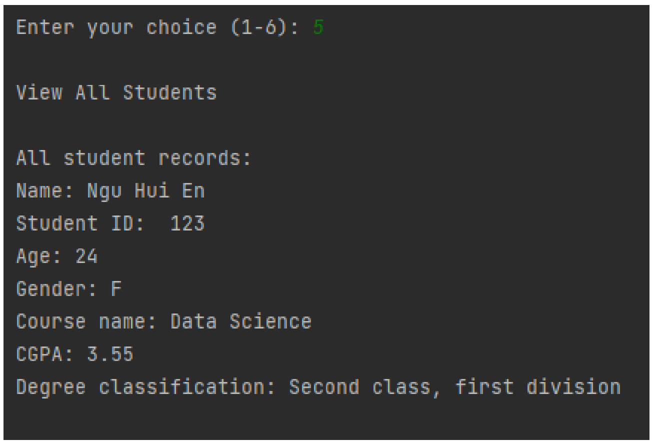
```
Enter your choice (1-6): 4

Update Student

Enter student ID to update: 123

Current student information:
Student Name: Ngu Hui En
Student ID: 123
Student Age: 23
Student Gender: F
Student Course Name: Computer Science
Student CGPA: 3.85
Student Degree Classification: First class

Please enter new student information:
Name: Ngu Hui En
Age: 24
Gender (M/F/O): F
Course name: Data Science
CGPA: 3.55

Student with ID 123 has been updated successfully!
```

Figure 1.20. The result of inserting valid student ID to be updated.



```
Enter your choice (1-6): 4

Update Student

Enter student ID to update: 789

Student with ID 789 does not exist in the system!
```

Figure 1.21. The result of inserting invalid student ID under searching function.

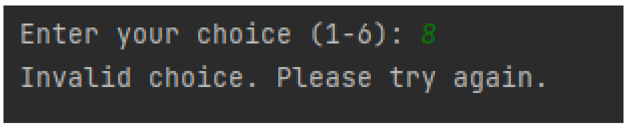


```
Enter your choice (1-6): 5

View All Students

All student records:
Name: Ngu Hui En
Student ID:  123
Age: 24
Gender: F
Course name: Data Science
CGPA: 3.55
Degree classification: Second class, first division
```

Figure 1.22. The updated view of all student records after updating the student record.

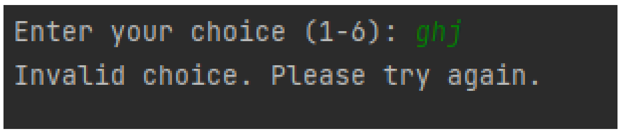
1.2.5 Menu display

When running the program, a menu will be displayed every time to ask the enter for entering the choice. Choice has to be entered in order to perform certain functions. Input that excluding integers 1 to 6 will display "Invalid choice. Please try again." (Figure 1.23, Figure 1.24). The user will then return to the menu. It will stop asking the user for choice until the user choose '6' to exit the program (Figure 1.25).



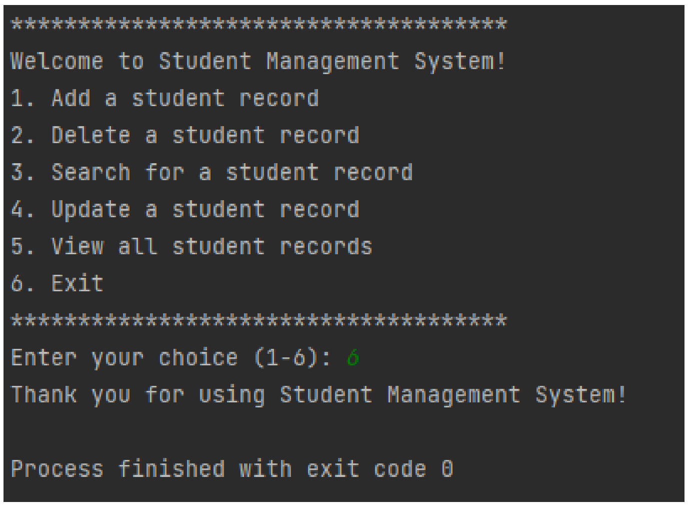
```
Enter your choice (1-6): 8
Invalid choice. Please try again.
```

Figure 1.23. The results after invalid choice were entered.



```
Enter your choice (1-6): ghj
Invalid choice. Please try again.
```

Figure 1.24. The results after invalid choice were entered.



```
*************************************
Welcome to Student Management System!
1. Add a student record
2. Delete a student record
3. Search for a student record
4. Update a student record
5. View all student records
6. Exit
*************************************
Enter your choice (1-6): 6
Thank you for using Student Management System!

Process finished with exit code 0
```

Figure 1.25. The interface of exiting the programme.