

|  |  |
| --- | --- |
| **Intake: APD1F2409CS(CYB)** | **Module Code:** **CT108-3-1-PYP** |

|  |
| --- |
| **Hand Out Date: Monday, 21 October 2024 Week 3** |
| **Submission Date:** **Sunday, 19 January 2024 Week 12** |

**TITLE:**

***UNIVERSITY MANAGEMENT SYSTEM (UMS)***

**Module Lecturer: Mrs. Aziah Abdollah**

**Group ZPYP**

|  |  |  |
| --- | --- | --- |
| **No.** | **Name** | **TP** |
| **1** | AHMED MIRAHUSAIN ALVI | TP084807 |
| **2** | ANDREW WIJAYA | TP079319 |
| **3** | MAHRUS SHAMSUL AHSAN | TP085562 |
| **4** | ABDALLAH MOHAMED MAHMOUD MOHAMED MAHMOUD | TP085097 |
| **5** | MOHAMMED YOUSEF MOHAMMED MOHAMMED | TP085042 |

Table of Contents

[ACKNOWLEDGEMENT 1](#_Toc188221221)

[ABSTRACT 2](#_Toc188221222)

[CHAPTER 1: UNIVERSITY MANAGEMENT SYSTEM 3](#_Toc188221223)

[**1.1** Introduction 3](#_Toc188221224)

[**1.2** Assumptions 4](#_Toc188221225)

[CHAPTER 2: PROGRAM DESIGN & PLANNING 6](#_Toc188221226)

[**2.1** UMS Main Menu 6](#_Toc188221227)

[**2.2** Administrator Role 10](#_Toc188221228)

[**2.3** Lecturer Role 23](#_Toc188221229)

[**2.4** Student Role 29](#_Toc188221230)

[**2.5** Registrar Role 35](#_Toc188221231)

[**2.6** Accountant Role 47](#_Toc188221232)

[CHAPTER 3: SOURCE CODE & EXPLAINATION 53](#_Toc188221233)

[***3.1*** *Application of Storage Types* 53](#_Toc188221234)

[**3.1.1** Lists 53](#_Toc188221235)

[***3.1.2*** *Text Files* 54](#_Toc188221236)

[**3.2** Application of Control Structure 55](#_Toc188221237)

[**3.2.1** *Iteration (for loop)* 55](#_Toc188221238)

[***3.2.2*** *Iteration: (while loop)* 56](#_Toc188221239)

[**3.2.3** Selective: if-else nested if 57](#_Toc188221240)

[***3.3*** *Application of Try…Except* 58](#_Toc188221241)

[***3.4*** *Application of Validation* 59](#_Toc188221242)

[***3.5*** *The Flow of Add Function* 60](#_Toc188221243)

[***3.6*** *The Flow of Update Function* 61](#_Toc188221244)

[***3.7*** *The Flow of Delete/Remove Function* 62](#_Toc188221245)

[***3.8*** *The Flow of Generating Reports* 63](#_Toc188221246)

[***3.9*** *Additional Features* 64](#_Toc188221247)

[CHAPTER 4: INPUT/OUTPUT SAMPLE 67](#_Toc188221248)

[***4.1*** *UMS Main Menu* 67](#_Toc188221249)

[***4.2*** *Administrator Role* 70](#_Toc188221250)

[***4.3*** *Lecturer Role* 77](#_Toc188221251)

[***4.4*** *Student Role* 83](#_Toc188221252)

[***4.6*** *Registrar Role* 90](#_Toc188221253)

[***4.6*** *Accountant Role* 94](#_Toc188221254)

[CONCLUSION 100](#_Toc188221255)

[REFERENCES 101](#_Toc188221256)

[APPENDIX A: TEXT FILES 102](#_Toc188221257)

[APPENDIX B: WORKLOAD MATRIX 107](#_Toc188221258)

ACKNOWLEDGEMENT

We extend our sincere gratitude to everyone who played a role in the successful completion of this document. Above all, we are deeply indebted to our mentor, Ms. Aziah Abdollah, for her invaluable guidance, insightful feedback and constant encouragement, which have been instrumental in shaping the quality and direction of this work.

We also would like to express our sincere appreciation to our colleagues and peers. Their spirit of collaboration, constructive discussions and shared knowledge have greatly enriched our understanding and perspective.

A special note of thanks goes to our friends—Mohammed Yousef Mohammed Mohammed, Mahrus Shamsul Ahsan, Ahmed Mirahusain Alvi, Andrew Wijaya, and Abdallah Mohamed Mahmoud—whose steadfast support, motivation, and camaraderie have been a constant source of inspiration throughout this journey.

Finally, we acknowledge the importance of the resources that have supported this work, including books, research articles, and online materials. These resources have been invaluable in providing the necessary foundation and depth to our efforts.

To everyone who encouraged and believed in us, we owe our success to your kindness and support. Thank you for making this accomplishment a reality.

ABSTRACT

This paper explores the development and implementation university management system. The system features 5 primary roles in the system including administrator, registrar, accountant, lecturer, and student, each playing an important role in the system. The report has been structured into 4 sections, providing detailed explanation of the design, implementation and usage of the system.

Section 1introduces the project, outlining all the assumptions made in order to create this system. These assumptions laid the foundation of the system. section 2 includes the pseudocode needed to create the system. The pseudocode has been further divided to showcase code design for each role. Section 3 talks about the source code and its explanation based on its functionality. The codes are explained according to certain criteria provided. Section 5 includes all the input, and output for each role. In this section, it is shown what happens when an user uses the system. It shows the results when the user enters valid and invalid input. The report then ends with the conclusion, references and appendix.

CHAPTER 1: UNIVERSITY MANAGEMENT SYSTEM

**1.1** Introduction

The University had an ineffective system to manage data and procedures for the five users: Students, lecturer, administrators, Accountant, Registrar. This system caused many problems. For example, it was easy to cause human errors. While operating the university menu, the users could make mistakes like choosing the wrong option. However, with an automatic system, equipped with validations, recorded data, User Friendly, the university can easily do its things with rapid, and less to make errors. Moreover, installing an automated system can be expensive due to the first investment in equipment, it could improve efficiency and reducing the staff needed to do the thing, reducing cost by time to time. Therefore, compared to the manual system, the automated one is better for the smooth running of the university, and it would also give it a modern touch.

Our team was tasked to improve system to become more efficient system. Processes like manage lecturer, register student, the fee that has been paid by student, record student grade and others would be automated. Validations and other functionalities are implemented in the system, to ensure more effective management and boost overall efficiency and accuracy.

**1.2** *Assumptions*

* The goal for this project is to make an automatic system which is to support the five primary users which are student, Lecturer, Administrator, Registrar, Accountant Each will have their respective menu page.
* There is a login feature that was needed to access each role function, in which the Username is “UMSLOGIN,” and the password is “12589”. This had been made to ensure the security of the program .
* Some of the things that administrators can do with the system are change View All Data, add or remove students, add classes, control teachers, and make reports. What registrars do, on the other hand, is manage enrollments, add new students, keep student records up to date, send out papers, keep an eye on student information, and delete student records.
* What do lecturers do for school? They must look at the modules they have been given, write down grades, look at names of students, keep track of attendance, and go over student scores. Students can also see what classes are available, choose to take or not take, see a record of their attendance, and see their grades. Last but not least, accountants handle money problems. It is their job to keep track of payments, print fee receipts, record school fees, and look at payments that are still due.
* Error management: When managing errors, try-except blocks are used to make sure that the program can know what to do if it faces the incorrect or missing input / data without crashing. The system is strengthened and stabilized this way to avoid redundant.
* We also implemented a modular program in which all roles are in different files, different save data which are all in text file and compiled through the one folder which all can be accessed through the UMS main logic menu.
* There are eight files being used which are “Security.txt” for login, “attendance.txt” for record student present data, “fee.txt” to record how much had the student paid, “Grades.txt” for record student score in test, ”Lecturer.txt” to record the lecturer info and show the lecturer info, “Modules.txt” to show the module data, “Student.txt” to record the student info and show the student info”

CHAPTER 2: PROGRAM DESIGN & PLANNING

**2.1** UMS Main Menu

BEGIN UniversityManagementSystem

FUNCTION get\_valid\_input(prompt, required=True)

WHILE TRUE DO

DISPLAY prompt

GET value

SET value TO value.strip()

IF value IS NOT EMPTY OR required IS FALSE THEN

RETURN value

ELSE

DISPLAY "This field cannot be empty. Please try again."

ENDIF

ENDWHILE

ENDFUNCTION

FUNCTION login()

DISPLAY "Please enter Login credentials"

DISPLAY 'UserName: "UMSLOGIN", Password: "12589"'

PROMPT "Enter Username: "

GET username USING get\_valid\_input(prompt)

PROMPT "Enter password: "

GET password USING get\_valid\_input(prompt)

TRY

OPEN "security.txt" FOR READ AS file

READ file LINE BY LINE INTO users

FOR EACH user IN users DO

SPLIT user BY "," INTO stored\_name AND stored\_password

IF username EQUALS stored\_name AND password EQUALS stored\_password THEN

DISPLAY "Login successful!"

RETURN TRUE

ENDIF

ENDFOR

DISPLAY "Invalid username or password. Try again."

RETURN FALSE

CATCH FileNotFoundError EXCEPTION

DISPLAY "Security file not found."

RETURN FALSE

ENDTRY

ENDFUNCTION

FUNCTION main\_menu()

WHILE TRUE DO

DISPLAY "-------Welcome to University Management System-----------"

SET login\_result TO CALL login()

IF login\_result IS FALSE THEN

DISPLAY "Access Denied! Please try logging in again."

CONTINUE

ENDIF

DISPLAY "\n---------- Main Menu ----------"

DISPLAY "1. Administrator"

DISPLAY "2. Lecturer"

DISPLAY "3. Student"

DISPLAY "4. Registrar"

DISPLAY "5. Accountant"

DISPLAY "6. Exit"

PROMPT "Enter your choice: "

GET choice

IF choice EQUALS "1" THEN

CALL Administrator.admin\_menu()

ELSE IF choice EQUALS "2" THEN

CALL Lecturer.lecturer\_menu()

ELSE IF choice EQUALS "3" THEN

CALL Student.student\_menu()

ELSE IF choice EQUALS "4" THEN

CALL Registrar.registrar\_menu()

ELSE IF choice EQUALS "5" THEN

CALL Accountant.accountant\_menu()

ELSE IF choice EQUALS "6" THEN

DISPLAY "Exiting now..."

BREAK

ELSE

DISPLAY "Invalid choice, please try again."

ENDIF

ENDWHILE

ENDFUNCTION

CALL main\_menu()

END UniversityManagementSystem

**2.2** Administrator Role

BEGIN AdministratorMenu

FUNCTION admin\_menu()

FUNCTION validate\_numeric\_input(prompt)

WHILE TRUE DO

DISPLAY prompt

GET value

SET value TO value.strip()

IF value IS NUMERIC THEN

RETURN value

ELSE

DISPLAY "Invalid input. Please enter a numeric value."

ENDIF

ENDWHILE

ENDFUNCTION

FUNCTION validate\_date\_input(prompt)

WHILE TRUE DO

DISPLAY prompt

GET date\_str

TRY

PARSE date\_str AS DATE (FORMAT: YYYY-MM-DD)

RETURN date\_str

CATCH ValueError EXCEPTION

DISPLAY "Invalid date format. Please use YYYY-MM-DD."

ENDTRY

ENDWHILE

ENDFUNCTION

FUNCTION manage\_module()

DISPLAY "\nModule Management"

PROMPT "Choose an option (Add, Update, Remove): "

GET option

SET option TO option.lower()

TRY

IF option EQUALS "add" THEN

PROMPT "Enter module code: "

GET module\_code

PROMPT "Enter module name: "

GET module\_name

PROMPT "Enter module credits: "

GET credits USING validate\_numeric\_input(prompt)

APPEND (module\_code, module\_name, credits) TO "modules.txt"

DISPLAY "Module added successfully."

ELSE IF option EQUALS "update" THEN

PROMPT "Enter module code to update: "

GET module\_code

SET found TO FALSE

READ ALL LINES FROM "modules.txt" INTO lines

OPEN "modules.txt" FOR WRITE

FOR EACH line IN lines DO

IF line STARTS WITH (module\_code + ",") THEN

PROMPT "Enter new module name: "

GET module\_name

PROMPT "Enter new course credits: "

GET credits USING validate\_numeric\_input(prompt)

WRITE (module\_code, module\_name, credits) TO FILE

SET found TO TRUE

ELSE

WRITE line TO FILE

ENDIF

ENDFOR

IF found THEN

DISPLAY "Module updated successfully."

ELSE

DISPLAY "Module not found."

ENDIF

ELSE IF option EQUALS "remove" THEN

PROMPT "Enter a module code to remove: "

GET module\_code

SET found TO FALSE

READ ALL LINES FROM "modules.txt" INTO lines

OPEN "modules.txt" FOR WRITE

FOR EACH line IN lines DO

IF NOT line STARTS WITH (module\_code + ",") THEN

WRITE line TO FILE

ELSE

SET found TO TRUE

ENDIF

ENDFOR

IF found THEN

DISPLAY "Module removed successfully."

ELSE

DISPLAY "Module not found."

ENDIF

ELSE

DISPLAY "Invalid option. Please try again."

ENDIF

CATCH FileNotFoundError EXCEPTION

DISPLAY "Error: File not found. Ensure 'modules.txt' exists."

ENDTRY

ENDFUNCTION

FUNCTION manage\_student()

DISPLAY "\nStudent Management"

PROMPT "Choose an option (Add, Update, Remove): "

GET option

SET option TO option.lower()

TRY

IF option EQUALS "add" THEN

PROMPT "Enter student ID: "

GET student\_id USING validate\_numeric\_input(prompt)

PROMPT "Enter student name: "

GET student\_name

PROMPT "Enter department: "

GET department

PROMPT "Enter Date of Admission [YYYY-MM-DD]: "

GET AdmissionDate USING validate\_date\_input(prompt)

APPEND (student\_id, student\_name, department, AdmissionDate) TO "students.txt"

DISPLAY "Student added successfully."

ELSE IF option EQUALS "update" THEN

PROMPT "Enter student ID to update: "

GET student\_id USING validate\_numeric\_input(prompt)

SET found TO FALSE

READ ALL LINES FROM "students.txt" INTO lines

OPEN "students.txt" FOR WRITE

FOR EACH line IN lines DO

IF line STARTS WITH (student\_id + ",") THEN

PROMPT "Enter new student name: "

GET student\_name

PROMPT "Enter new department: "

GET department

WRITE (student\_id, student\_name, department) TO FILE

SET found TO TRUE

ELSE

WRITE line TO FILE

ENDIF

ENDFOR

IF found THEN

DISPLAY "Student updated successfully."

ELSE

DISPLAY "Student not found."

ENDIF

ELSE IF option EQUALS "remove" THEN

PROMPT "Enter student ID to remove: "

GET student\_id USING validate\_numeric\_input(prompt)

SET found TO FALSE

READ ALL LINES FROM "students.txt" INTO lines

OPEN "students.txt" FOR WRITE

FOR EACH line IN lines DO

IF NOT line STARTS WITH (student\_id + ",") THEN

WRITE line TO FILE

ELSE

SET found TO TRUE

ENDIF

ENDFOR

IF found THEN

DISPLAY "Student removed successfully."

ELSE

DISPLAY "Student not found."

ENDIF

ELSE

DISPLAY "Invalid option. Please try again."

ENDIF

CATCH FileNotFoundError EXCEPTION

DISPLAY "Error: File not found. Ensure 'students.txt' exists."

ENDTRY

ENDFUNCTION

FUNCTION manage\_lecturer()

DISPLAY "\nLecturer Management"

PROMPT "Choose an option (Add, Update, Remove): "

GET option

SET option TO option.lower()

TRY

IF option EQUALS "add" THEN

PROMPT "Enter lecturer ID: "

GET lecturer\_id USING validate\_numeric\_input(prompt)

PROMPT "Enter lecturer name: "

GET lecturer\_name

PROMPT "Enter department: "

GET department

PROMPT "Enter Date of Registration [YYYY-MM-DD]: "

GET HiringDate USING validate\_date\_input(prompt)

APPEND (lecturer\_id, lecturer\_name, department, HiringDate) TO "lecturers.txt"

DISPLAY "Lecturer added successfully."

ELSE IF option EQUALS "update" THEN

PROMPT "Enter lecturer ID to update: "

GET lecturer\_id USING validate\_numeric\_input(prompt)

SET found TO FALSE

READ ALL LINES FROM "lecturers.txt" INTO lines

OPEN "lecturers.txt" FOR WRITE

FOR EACH line IN lines DO

IF line STARTS WITH (lecturer\_id + ",") THEN

PROMPT "Enter new lecturer name: "

GET lecturer\_name

PROMPT "Enter new department: "

GET department

WRITE (lecturer\_id, lecturer\_name, department) TO FILE

SET found TO TRUE

ELSE

WRITE line TO FILE

ENDIF

ENDFOR

IF found THEN

DISPLAY "Lecturer updated successfully."

ELSE

DISPLAY "Lecturer not found."

ENDIF

ELSE IF option EQUALS "remove" THEN

PROMPT "Enter lecturer ID to remove: "

GET lecturer\_id USING validate\_numeric\_input(prompt)

SET found TO FALSE

READ ALL LINES FROM "lecturers.txt" INTO lines

OPEN "lecturers.txt" FOR WRITE

FOR EACH line IN lines DO

IF NOT line STARTS WITH (lecturer\_id + ",") THEN

WRITE line TO FILE

ELSE

SET found TO TRUE

ENDIF

ENDFOR

IF found THEN

DISPLAY "Lecturer removed successfully."

ELSE

DISPLAY "Lecturer not found."

ENDIF

ELSE

DISPLAY "Invalid option. Please try again."

ENDIF

CATCH FileNotFoundError EXCEPTION

DISPLAY "Error: File not found. Ensure 'lecturers.txt' exists."

ENDTRY

ENDFUNCTION

FUNCTION view\_all\_data()

TRY

DISPLAY "\nViewing All Data:"

SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)

DISPLAY "Data viewed on: " + timestamp

DISPLAY "\nAll Students Data:"

READ ALL CONTENTS OF "students.txt" AND DISPLAY

DISPLAY "\nAll Modules Data:"

READ ALL CONTENTS OF "modules.txt" AND DISPLAY

DISPLAY "\nAll Lecturers Data:"

READ ALL CONTENTS OF "lecturers.txt" AND DISPLAY

CATCH FileNotFoundError EXCEPTION

DISPLAY "Error: One or more files are missing. Ensure all required files exist."

ENDTRY

ENDFUNCTION

FUNCTION generate\_reports()

TRY

DISPLAY "\nGenerating Reports:"

SET student\_count TO COUNT LINES IN "students.txt"

SET module\_count TO COUNT LINES IN "modules.txt"

SET lecturer\_count TO COUNT LINES IN "lecturers.txt"

SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)

DISPLAY "Reports generated on: " + timestamp

DISPLAY "Total Students: " + student\_count

DISPLAY "Active Modules: " + module\_count

DISPLAY "Total Lecturers: " + lecturer\_count

CATCH FileNotFoundError EXCEPTION

DISPLAY "Error: One or more files are missing. Ensure all required files exist."

ENDTRY

ENDFUNCTION

WHILE TRUE DO

DISPLAY "\n---------- University Administrator ------------"

DISPLAY "1. Manage Modules"

DISPLAY "2. Manage Students"

DISPLAY "3. Manage Lecturers"

DISPLAY "4. View All Data"

DISPLAY "5. Generate Reports"

DISPLAY "6. Exit"

PROMPT "Enter your choice: "

GET choice

IF choice EQUALS "1" THEN

CALL manage\_module()

ELSE IF choice EQUALS "2" THEN

CALL manage\_student()

ELSE IF choice EQUALS "3" THEN

CALL manage\_lecturer()

ELSE IF choice EQUALS "4" THEN

CALL view\_all\_data()

ELSE IF choice EQUALS "5" THEN

CALL generate\_reports()

ELSE IF choice EQUALS "6" THEN

DISPLAY "Exiting Administrator menu."

BREAK

ELSE

DISPLAY "Invalid choice. Please try again."

ENDIF

ENDWHILE

ENDFUNCTION

END AdministratorMenu

**2.3** Lecturer Role

*BEGIN LecturerMenu*

*FUNCTION lecturer\_menu()*

*FUNCTION view\_assigned\_modules(lecturer\_id)*

*DISPLAY "Viewing Assigned Modules..."*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)*

*DISPLAY "time-stamp: " + timestamp*

*TRY*

*OPEN "modules.txt" FOR READ*

*DISPLAY "Modules assigned:"*

*FOR EACH line IN file DO*

*SPLIT line BY "," INTO details*

*IF LENGTH OF details IS 4 AND details[3] EQUALS lecturer\_id THEN*

*DISPLAY details[0] + " - " + details[1] + " (" + details[2] + " credits)"*

*ENDIF*

*ENDFOR*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Modules file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION view\_student\_list(module\_code)*

*DISPLAY "Student List for Module: " + module\_code*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)*

*DISPLAY "time-stamp: " + timestamp*

*TRY*

*OPEN "enrolments.txt" FOR READ*

*DISPLAY "Enrolled Students:"*

*SET found TO FALSE*

*FOR EACH line IN file DO*

*SET line TO line.strip()*

*IF line IS EMPTY THEN*

*CONTINUE*

*ENDIF*

*TRY*

*SPLIT line BY "," INTO student\_id AND mod\_code*

*IF mod\_code EQUALS module\_code THEN*

*DISPLAY "Student ID: " + student\_id*

*SET found TO TRUE*

*ENDIF*

*CATCH ValueError EXCEPTION*

*DISPLAY "Skipping invalid line: " + line*

*ENDTRY*

*ENDFOR*

*IF NOT found THEN*

*DISPLAY "No students are enrolled in module " + module\_code + "."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Enrollments file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION record\_grades()*

*DISPLAY "Recording Grades..."*

*PROMPT "Enter Module Code: "*

*GET module\_code*

*PROMPT "Enter Student ID: "*

*GET student\_id*

*PROMPT "Enter Grade: "*

*GET grade*

*TRY*

*OPEN "grades.txt" FOR APPEND*

*WRITE (module\_code + "," + student\_id + "," + grade) TO file*

*DISPLAY "Grade recorded successfully."*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Grades file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION track\_attendance()*

*DISPLAY "Tracking Attendance..."*

*PROMPT "Enter Module Code: "*

*GET module\_code*

*PROMPT "Enter Student ID: "*

*GET student\_id*

*PROMPT "Enter Attendance Status (Present/Absent): "*

*GET attendance\_status*

*TRY*

*OPEN "attendance.txt" FOR APPEND*

*WRITE (module\_code + "," + student\_id + "," + attendance\_status) TO file*

*DISPLAY "Attendance recorded successfully."*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Attendance file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION view\_student\_grades(module\_code)*

*DISPLAY "Viewing Grades for Module: " + module\_code*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)*

*DISPLAY "time-stamp: " + timestamp*

*TRY*

*OPEN "grades.txt" FOR READ*

*DISPLAY "Student Grades:"*

*FOR EACH line IN file DO*

*SPLIT line BY "," INTO mod\_code, student\_id, grade*

*IF mod\_code EQUALS module\_code THEN*

*DISPLAY "Student ID: " + student\_id + ", Grade: " + grade*

*ENDIF*

*ENDFOR*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Grades file not found."*

*ENDTRY*

*ENDFUNCTION*

*WHILE TRUE DO*

*DISPLAY "---------- Lecturer Menu ----------"*

*DISPLAY "1. View Assigned Modules"*

*DISPLAY "2. View Student List for a Module"*

*DISPLAY "3. Record Grades"*

*DISPLAY "4. Track Attendance"*

*DISPLAY "5. View Student Grades"*

*DISPLAY "6. Exit"*

*PROMPT "Enter your choice: "*

*GET choice*

*IF choice EQUALS "1" THEN*

*PROMPT "Enter your Lecturer ID: "*

*GET lecturer\_id*

*CALL view\_assigned\_modules(lecturer\_id)*

*ELSE IF choice EQUALS "2" THEN*

*PROMPT "Enter Module Code: "*

*GET module\_code*

*CALL view\_student\_list(module\_code)*

*ELSE IF choice EQUALS "3" THEN*

*CALL record\_grades()*

*ELSE IF choice EQUALS "4" THEN*

*CALL track\_attendance()*

*ELSE IF choice EQUALS "5" THEN*

*PROMPT "Enter Module Code: "*

*GET module\_code*

*CALL view\_student\_grades(module\_code)*

*ELSE IF choice EQUALS "6" THEN*

*DISPLAY "Exiting Lecturer Menu."*

*BREAK*

*ELSE*

*DISPLAY "Invalid choice, please try again."*

*ENDIF*

*ENDWHILE*

*ENDFUNCTION*

*END LecturerMenu*

**2.4** Student Role

*BEGIN StudentMenu*

*FUNCTION student\_menu()*

*FUNCTION view\_available\_modules()*

*DISPLAY "Available Modules:"*

*TRY*

*OPEN "modules.txt" FOR READ*

*DISPLAY "Module Code - Module Name (Credits)"*

*FOR EACH line IN file DO*

*SPLIT line BY "," INTO details*

*IF LENGTH OF details IS GREATER THAN OR EQUAL TO 3 THEN*

*DISPLAY details[0] + " - " + details[1] + " (" + details[2] + " credits)"*

*ENDIF*

*ENDFOR*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Modules file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION enroll\_in\_module(student\_id)*

*DISPLAY "Enroll in Module..."*

*PROMPT "Enter Module Code: "*

*GET module\_code*

*TRY*

*OPEN "enrolments.txt" FOR APPEND*

*WRITE (student\_id + "," + module\_code) TO file*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD:HH:MM:SS)*

*DISPLAY "Enrolled in module " + module\_code + " successfully on " + timestamp + "."*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Enrollments file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION unroll\_from\_module(student\_id)*

*DISPLAY "Unroll from Module..."*

*PROMPT "Enter Module Code: "*

*GET module\_code*

*TRY*

*SET found TO FALSE*

*READ ALL LINES FROM "enrolments.txt" INTO lines*

*OPEN "enrolments.txt" FOR WRITE*

*FOR EACH line IN lines DO*

*IF line.strip() NOT EQUALS (student\_id + "," + module\_code) THEN*

*WRITE line TO file*

*ELSE*

*SET found TO TRUE*

*ENDIF*

*ENDFOR*

*IF found THEN*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD:HH:MM:SS)*

*DISPLAY "Unrolled from module " + module\_code + " successfully on " + timestamp + "."*

*ELSE*

*DISPLAY "No enrollment found for module " + module\_code + "."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Enrollments file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION view\_grades(student\_id)*

*DISPLAY "Your Grades:"*

*TRY*

*OPEN "grades.txt" FOR READ*

*FOR EACH line IN file DO*

*SPLIT line BY "," INTO mod\_code, sid, grade*

*IF sid EQUALS student\_id THEN*

*DISPLAY "Module Code: " + mod\_code + ", Grade: " + grade*

*ENDIF*

*ENDFOR*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Grades file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION access\_attendance\_record(student\_id)*

*PROMPT "Enter Module Code: "*

*GET module\_code*

*DISPLAY "Attendance Records for Student ID " + student\_id + " in Module " + module\_code + ":"*

*TRY*

*OPEN "attendance.txt" FOR READ*

*SET total\_classes TO 0*

*SET attended\_classes TO 0*

*FOR EACH line IN file DO*

*SET line TO line.strip()*

*IF line IS EMPTY THEN*

*CONTINUE*

*ENDIF*

*TRY*

*SPLIT line BY "," INTO mod\_code, sid, status*

*IF sid EQUALS student\_id AND mod\_code EQUALS module\_code THEN*

*INCREMENT total\_classes BY 1*

*IF status.lower() EQUALS "present" THEN*

*INCREMENT attended\_classes BY 1*

*ENDIF*

*ENDIF*

*CATCH ValueError EXCEPTION*

*DISPLAY "Skipping invalid line: " + line*

*ENDTRY*

*ENDFOR*

*IF total\_classes IS GREATER THAN 0 THEN*

*SET attendance\_percentage TO (attended\_classes / total\_classes) \* 100*

*DISPLAY "Total Classes: " + total\_classes*

*DISPLAY "Classes Attended: " + attended\_classes*

*DISPLAY "Attendance Percentage: " + attendance\_percentage + "%"*

*ELSE*

*DISPLAY "No attendance records found for Module " + module\_code + " with your Student ID."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Attendance file not found."*

*ENDTRY*

*ENDFUNCTION*

*WHILE TRUE DO*

*DISPLAY "---------- Student Menu ----------"*

*DISPLAY "1. View Available Modules"*

*DISPLAY "2. Enroll in Module"*

*DISPLAY "3. Unroll from Module"*

*DISPLAY "4. View Grades"*

*DISPLAY "5. Access Attendance Record"*

*DISPLAY "6. Exit"*

*PROMPT "Enter your choice: "*

*GET choice*

*IF choice EQUALS "1" THEN*

*CALL view\_available\_modules()*

*ELSE IF choice EQUALS "2" THEN*

*PROMPT "Enter your Student ID: "*

*GET student\_id*

*CALL enroll\_in\_module(student\_id)*

*ELSE IF choice EQUALS "3" THEN*

*PROMPT "Enter your Student ID: "*

*GET student\_id*

*CALL unroll\_from\_module(student\_id)*

*ELSE IF choice EQUALS "4" THEN*

*PROMPT "Enter your Student ID: "*

*GET student\_id*

*CALL view\_grades(student\_id)*

*ELSE IF choice EQUALS "5" THEN*

*PROMPT "Enter your Student ID: "*

*GET student\_id*

*CALL access\_attendance\_record(student\_id)*

*ELSE IF choice EQUALS "6" THEN*

*DISPLAY "Exiting Student Menu."*

*BREAK*

*ELSE*

*DISPLAY "Invalid choice, please try again."*

*ENDIF*

*ENDWHILE*

*ENDFUNCTION*

*END StudentMenu*

**2.5** Registrar Role

*BEGIN RegistrarMenu*

*FUNCTION registrar\_menu()*

*FUNCTION get\_valid\_input(prompt, required=True)*

*WHILE TRUE DO*

*DISPLAY prompt*

*GET value*

*SET value TO value.strip()*

*IF value IS NOT EMPTY OR required IS FALSE THEN*

*RETURN value*

*ELSE*

*DISPLAY "This field cannot be empty. Please try again."*

*ENDIF*

*ENDWHILE*

*ENDFUNCTION*

*FUNCTION validate\_numeric\_input(prompt)*

*WHILE TRUE DO*

*DISPLAY prompt*

*GET value*

*SET value TO value.strip()*

*IF value IS NUMERIC THEN*

*RETURN value*

*ELSE*

*DISPLAY "Invalid input. Please enter a numeric value."*

*ENDIF*

*ENDWHILE*

*ENDFUNCTION*

*FUNCTION validate\_date\_input(prompt)*

*WHILE TRUE DO*

*DISPLAY prompt*

*GET date\_str*

*TRY*

*PARSE date\_str AS DATE (FORMAT: YYYY-MM-DD)*

*RETURN date\_str*

*CATCH ValueError EXCEPTION*

*DISPLAY "Invalid date format. Please use YYYY-MM-DD."*

*ENDTRY*

*ENDWHILE*

*ENDFUNCTION*

*FUNCTION register\_new\_student()*

*DISPLAY "Register New Student"*

*PROMPT "Enter Student ID: "*

*GET student\_id USING validate\_numeric\_input(prompt)*

*IF CALL check\_student\_id\_exists(student\_id) THEN*

*DISPLAY "Student ID " + student\_id + " has already been registered."*

*RETURN*

*ENDIF*

*PROMPT "Enter Student Name: "*

*GET student\_name USING get\_valid\_input(prompt)*

*PROMPT "Enter Program Name: "*

*GET program USING get\_valid\_input(prompt)*

*TRY*

*OPEN "students.txt" FOR APPEND*

*WRITE (student\_id + "," + student\_name + "," + program) TO file*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)*

*DISPLAY "Student registered on: " + timestamp*

*DISPLAY "Student registered successfully."*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Students file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION check\_student\_id\_exists(student\_id)*

*TRY*

*OPEN "students.txt" FOR READ*

*FOR EACH line IN file DO*

*SPLIT line BY "," INTO existing\_student\_id AND IGNORE*

*IF student\_id EQUALS existing\_student\_id THEN*

*RETURN TRUE*

*ENDIF*

*ENDFOR*

*RETURN FALSE*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Students file not found."*

*RETURN FALSE*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION update\_student\_record()*

*DISPLAY "Update Student Record"*

*PROMPT "Enter Student ID to update: "*

*GET student\_id USING validate\_numeric\_input(prompt)*

*TRY*

*READ ALL LINES FROM "students.txt" INTO lines*

*SET updated TO FALSE*

*OPEN "students.txt" FOR WRITE*

*FOR EACH line IN lines DO*

*IF line STARTS WITH (student\_id + ",") THEN*

*PROMPT "Enter Updated Student Name: "*

*GET student\_name USING get\_valid\_input(prompt)*

*PROMPT "Enter Updated Program Name: "*

*GET program USING get\_valid\_input(prompt)*

*WRITE (student\_id + "," + student\_name + "," + program) TO file*

*SET updated TO TRUE*

*ELSE*

*WRITE line TO file*

*ENDIF*

*ENDFOR*

*IF updated THEN*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD:HH:MM:SS)*

*DISPLAY "Student information updated on: " + timestamp*

*DISPLAY "Student record updated successfully."*

*ELSE*

*DISPLAY "Student ID not found."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Students file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION manage\_enrollments()*

*DISPLAY "Manage Enrollments"*

*PROMPT "Enter Student ID: "*

*GET student\_id USING validate\_numeric\_input(prompt)*

*PROMPT "Enter Module Code: "*

*GET module\_code USING get\_valid\_input(prompt)*

*SET module\_exists TO FALSE*

*TRY*

*OPEN "modules.txt" FOR READ*

*FOR EACH line IN file DO*

*IF line.strip() EQUALS module\_code THEN*

*SET module\_exists TO TRUE*

*BREAK*

*ENDIF*

*ENDFOR*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Modules file not found."*

*ENDTRY*

*IF NOT module\_exists THEN*

*DISPLAY "Module code " + module\_code + " does not exist."*

*RETURN*

*ENDIF*

*PROMPT "Enter Action (Enroll/Unenroll): "*

*GET action*

*SET action TO action.lower()*

*TRY*

*IF action EQUALS "enroll" THEN*

*OPEN "enrolments.txt" FOR APPEND*

*WRITE (student\_id + "," + module\_code) TO file*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD:HH:MM:SS)*

*DISPLAY "Student enrolled in " + module\_code + " on: " + timestamp*

*ELSE IF action EQUALS "unenroll" THEN*

*READ ALL LINES FROM "enrolments.txt" INTO lines*

*SET found TO FALSE*

*OPEN "enrolments.txt" FOR WRITE*

*FOR EACH line IN lines DO*

*IF line.strip() NOT EQUALS (student\_id + "," + module\_code) THEN*

*WRITE line TO file*

*ELSE*

*SET found TO TRUE*

*ENDIF*

*ENDFOR*

*IF found THEN*

*DISPLAY "Student " + student\_id + " unenrolled from module " + module\_code + "."*

*ELSE*

*DISPLAY "Enrollment record not found."*

*ENDIF*

*ELSE*

*DISPLAY "Invalid action."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Enrollments file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION issue\_transcript()*

*DISPLAY "Issue Transcript"*

*PROMPT "Enter Student ID: "*

*GET student\_id USING validate\_numeric\_input(prompt)*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)*

*TRY*

*DISPLAY "Transcript for Student ID: " + student\_id*

*DISPLAY "Modules and Grades:"*

*OPEN "grades.txt" FOR READ*

*SET found TO FALSE*

*FOR EACH line IN file DO*

*SPLIT line BY "," INTO mod\_code, sid, grade*

*IF sid EQUALS student\_id THEN*

*DISPLAY "Module: " + mod\_code + ", Grade: " + grade*

*SET found TO TRUE*

*ENDIF*

*ENDFOR*

*IF NOT found THEN*

*DISPLAY "No grades found for this student."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Grades file not found."*

*ENDTRY*

*DISPLAY "Transcript generated on: " + timestamp*

*ENDFUNCTION*

*FUNCTION delete\_student\_record()*

*DISPLAY "Delete Student Record"*

*PROMPT "Enter Student ID to delete: "*

*GET student\_id USING validate\_numeric\_input(prompt)*

*TRY*

*READ ALL LINES FROM "students.txt" INTO lines*

*SET deleted TO FALSE*

*OPEN "students.txt" FOR WRITE*

*FOR EACH line IN lines DO*

*IF NOT line STARTS WITH (student\_id + ",") THEN*

*WRITE line TO file*

*ELSE*

*SET deleted TO TRUE*

*ENDIF*

*ENDFOR*

*IF deleted THEN*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD:HH:MM:SS)*

*DISPLAY "Student record deleted on: " + timestamp*

*DISPLAY "Student record deleted successfully."*

*ELSE*

*DISPLAY "Student ID not found."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Students file not found."*

*ENDTRY*

*ENDFUNCTION*

*FUNCTION view\_student\_information()*

*DISPLAY "View Student Information"*

*PROMPT "Enter Student ID to view: "*

*GET student\_id USING validate\_numeric\_input(prompt)*

*TRY*

*OPEN "students.txt" FOR READ*

*SET found TO FALSE*

*FOR EACH line IN file DO*

*IF line STARTS WITH (student\_id + ",") THEN*

*SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD:HH:MM:SS)*

*DISPLAY "Student info viewed on: " + timestamp*

*DISPLAY "Student Details:"*

*SPLIT line BY "," INTO id, name, program*

*DISPLAY "ID: " + id + ", Name: " + name + ", Program: " + program*

*SET found TO TRUE*

*BREAK*

*ENDIF*

*ENDFOR*

*IF NOT found THEN*

*DISPLAY "Student ID not found."*

*ENDIF*

*CATCH FileNotFoundError EXCEPTION*

*DISPLAY "Students file not found."*

*ENDTRY*

*ENDFUNCTION*

*WHILE TRUE DO*

*DISPLAY "---------- Registrar Menu ----------"*

*DISPLAY "1. Register New Student"*

*DISPLAY "2. Update Student Record"*

*DISPLAY "3. Manage Enrollments"*

*DISPLAY "4. Issue Transcript"*

*DISPLAY "5. Delete Student Record"*

*DISPLAY "6. View Student Information"*

*DISPLAY "7. Exit"*

*PROMPT "Enter your choice: "*

*GET choice*

*IF choice EQUALS "1" THEN*

*CALL register\_new\_student()*

*ELSE IF choice EQUALS "2" THEN*

*CALL update\_student\_record()*

*ELSE IF choice EQUALS "3" THEN*

*CALL manage\_enrollments()*

*ELSE IF choice EQUALS "4" THEN*

*CALL issue\_transcript()*

*ELSE IF choice EQUALS "5" THEN*

*CALL delete\_student\_record()*

*ELSE IF choice EQUALS "6" THEN*

*CALL view\_student\_information()*

*ELSE IF choice EQUALS "7" THEN*

*DISPLAY "Exiting Registrar Menu."*

*BREAK*

*ELSE*

*DISPLAY "Invalid choice, please try again."*

*ENDIF*

*ENDWHILE*

*ENDFUNCTION*

*END RegistrarMenu*

**2.6** Accountant Role

BEGIN AccountantMenu

FUNCTION validate\_numeric\_input(prompt)

WHILE TRUE DO

DISPLAY prompt

GET value

SET value TO value.strip()

IF value IS NUMERIC THEN

RETURN value

ELSE

DISPLAY "Invalid input. Please enter a numeric value."

ENDIF

ENDWHILE

ENDFUNCTION

FUNCTION validate\_date\_input(prompt)

WHILE TRUE DO

DISPLAY prompt

GET date\_str

TRY

PARSE date\_str AS DATE (FORMAT: YYYY-MM-DD)

RETURN date\_str

CATCH ValueError EXCEPTION

DISPLAY "Invalid date format. Please use YYYY-MM-DD."

ENDTRY

ENDWHILE

ENDFUNCTION

FUNCTION accountant\_menu()

FUNCTION record\_tuition\_fees()

DISPLAY "Record Tuition Fees"

PROMPT "Enter Student ID: "

GET student\_id

PROMPT "Enter Amount Paid: "

GET amount\_paid USING validate\_numeric\_input(prompt)

PROMPT "Enter Date of Payment (YYYY-MM-DD): "

GET date\_of\_payment USING validate\_date\_input(prompt)

TRY

OPEN "fees.txt" FOR APPEND

WRITE (student\_id + "," + amount\_paid + "," + date\_of\_payment) TO file

DISPLAY "Tuition fee payment recorded successfully."

CATCH FileNotFoundError EXCEPTION

DISPLAY "Fees file not found."

ENDTRY

ENDFUNCTION

FUNCTION view\_outstanding\_fees()

DISPLAY "View Outstanding Fees"

SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)

DISPLAY "Reports generated on: " + timestamp

TRY

DISPLAY "Outstanding Fees:"

OPEN "fees.txt" FOR READ

FOR EACH line IN file DO

SPLIT line BY "," INTO details

IF LENGTH OF details EQUALS 3 THEN

SET student\_id, amount\_paid, date\_of\_payment TO details

DISPLAY "Student ID: " + student\_id + ", Outstanding Fees: " + amount\_paid

ENDIF

ENDFOR

CATCH FileNotFoundError EXCEPTION

DISPLAY "Fees file not found."

ENDTRY

ENDFUNCTION

FUNCTION update\_payment\_records()

DISPLAY "Update Payment Records"

PROMPT "Enter Student ID: "

GET student\_id

PROMPT "Enter Updated Amount Paid: "

GET new\_payment USING validate\_numeric\_input(prompt)

PROMPT "Enter Updated Payment Date (YYYY-MM-DD): "

GET payment\_date USING validate\_date\_input(prompt)

TRY

READ ALL LINES FROM "fees.txt" INTO lines

SET updated TO FALSE

OPEN "fees.txt" FOR WRITE

FOR EACH line IN lines DO

IF line STARTS WITH (student\_id + ",") THEN

WRITE (student\_id + "," + new\_payment + "," + payment\_date) TO file

SET updated TO TRUE

ELSE

WRITE line TO file

ENDIF

ENDFOR

IF updated THEN

DISPLAY "Payment record updated successfully."

ELSE

DISPLAY "Student ID not found in payment records."

ENDIF

CATCH FileNotFoundError EXCEPTION

DISPLAY "Fees file not found."

ENDTRY

ENDFUNCTION

FUNCTION issue\_fee\_receipts()

DISPLAY "Issue Fee Receipt"

PROMPT "Enter Student ID: "

GET student\_id

TRY

OPEN "fees.txt" FOR READ

FOR EACH line IN file DO

IF line STARTS WITH (student\_id + ",") THEN

SPLIT line BY "," INTO details

DISPLAY "Fee Receipt:"

DISPLAY "Student ID: " + details[0]

DISPLAY "Amount Paid: " + details[1]

DISPLAY "Payment Date: " + details[2]

RETURN

ENDIF

ENDFOR

DISPLAY "No fee record found for the given Student ID."

CATCH FileNotFoundError EXCEPTION

DISPLAY "Fees file not found."

ENDTRY

ENDFUNCTION

FUNCTION view\_financial\_summary()

DISPLAY "View Financial Summary"

SET timestamp TO CURRENT DATE AND TIME (FORMAT: YYYY-MM-DD HH:MM:SS)

DISPLAY "Reports generated on: " + timestamp

TRY

SET total\_fees\_collected TO 0

OPEN "fees.txt" FOR READ

FOR EACH line IN file DO

SPLIT line BY "," INTO details

IF LENGTH OF details IS GREATER THAN OR EQUAL TO 2 THEN

ADD details[1] (AS FLOAT) TO total\_fees\_collected

ENDIF

ENDFOR

DISPLAY "Total Fees Collected: " + total\_fees\_collected

CATCH FileNotFoundError EXCEPTION

DISPLAY "Fees file not found."

CATCH ValueError EXCEPTION

DISPLAY "Error reading fee amounts. Ensure all data is numeric."

ENDTRY

ENDFUNCTION

ENDFUNCTION

END AccountantMenu

# **CHAPTER 3: SOURCE CODE & EXPLAINATION**

## ***3.1*** *Application of Storage Types*

**3.1.1** Lists

***A computer code on a black background

Description automatically generated***Lists are at the centre of how our system, it handles the collection of records whether that is from a student or from accountants. Data from our txt files are read into lists, enabling reliable and easier processing and operations. An example of this is the outstanding fee function that reads a txt file for example our student.txt line by line, then creates a dictionary for each record and stores all dictionaries in a list.

Figure 3.1.1 List

This allows us to filter, search for and update records for student. The structure of the list allows for data to be accessed sequentially or modified which is important for displaying all student records.

### ***3.1.2*** *Text Files*

Text files are used for constant storage. In our system we have used txt files to ensure that data remains connected across our different program execution.

**A screen shot of a computer program

Description automatically generated**Figure 3.1.2 Text file

In this screenshot, we have taken a section of our delete student\_record function which shows how we are able to write lists back into a text file. This process ensures changes are made during runtime, such as deleting the student record from the data.

**3.2** Application of Control Structure

### **3.2.1** *Iteration (for loop)*

Loops (for and while) allow for repetitive actions, for our university management system we used this to iterate through student grade records to find a match. An example of this is how we use the loop to iterate through the student grade list by processing each student grade one by one.

*A computer screen shot of text

Description automatically generated*

Figure 3.2.1 Iteration for loop

### ***3.2.2*** *Iteration: (while loop)*

A screen shot of a computer program

Description automatically generated

Figure 3.2.2 While Loop

While loop has a slightly different with the For loop. For loop has a limit like how long we want this output or program to be executed while loop do not have a limit. In figure 3.2.2 the while loop is true means that the main menu will repeat every time until the user exit which mean it is break which make it become false.

**3.2.3** Selective: if-else nested if

Nested if means that inside the if there is still another if. In figure 3.2.3 it shows that there are “if Sid == ...” means it is the first condition, and there is also if status. Lower () =...” mean it is the second condition. So, to output to record present in the data, the system needs to make sure that the student is in the class and module and the lecturer input that the student is present on that class

A screen shot of a computer program

Description automatically generated

Figure 3.2.3 Selective if else nested if

## ***3.3*** *Application of Try…Except*

Our university management system utilizes try-except blocks to handle potential runtime errors, which helps us avoid program problems which crashes. An example of this is when extracting a header from data. The use of try-except ensures the system will see it useful when the data is missing or not properly formatted. If we did not use this to handle errors, we would run into a bigger issue which can usually cause syntax errors.

A screen shot of a computer code

Description automatically generated

Figure 3.3.1 Try...except block

## ***3.4*** *Application of Validation*

*A screen shot of a computer program

Description automatically generatedA computer screen with text

Description automatically generated*Figure 3.4.1 validate numeric input

Figure 3.4.2 validate date input

Validation ensures that only correct and meaningful data is processed. In figure 3. 4.1 we found a way to make sure the user to input number if not followed it will ask the user to input again. Same as 3.4.2 and 3.4.3 which has the same function as figure 3.4.1 but has different function.

This prevents invalid updates to number and date as if a user attempts to modify a number and date that does not exist it will validate and then push them to the else block. We also ensured that if a user is to make such a mistake, we would inform them through the print block which allows them to be informed of their mistake and then correct their input.

## ***3.5*** *The Flow of Add Function*

A screen shot of a computer code

Description automatically generatedFigure 3.5.1 add function

In figure 3.5.1 For the flow of add functions, the user input the lecturer ID, name, department, and hiring date. After the user input everything, the code will open the lecturer.txt file and add the lecturer data inside the txt and print “lecturer added successfully.”

## A computer screen shot of code Description automatically generated***3.6*** *The Flow of Update Function*

Figure 3.6.1 Update function

In figure 3.6.1, the flow of update is it will ask the user to enter which module to update and the code will open the modules. txt and search for it. If it was found inside the data, it will ask the user to input the new module name and the course credit and it will save the data inside and print module updated successfully. If it does not find any data related to the module name, it will print module not found.

## ***3.7*** *The Flow of Delete/Remove Function*

A computer screen shot of a program

Description automatically generated

Figure 3.7.1 delete function

In figure 3.7.1 for the flow for delete function is it will ask the user to put student ID and it will open the student file and check for the student ID. If they find the student ID in the record, the data related to that student Id will be deleted. While if they do not find it will say print student not found.

## ***3.8*** *The Flow of Generating Reports*

A screen shot of a computer program

Description automatically generated3.8.1 Figure 3.8.0 Generating Reports

In figure 3.8.1 the code will start by opening the txt that was needed to generate the reports like student name, module id, lecturer name. After that, the code will start to print all the information that was found in the record.

## ***3.9*** *Additional Features*

A computer screen with text and images

Description automatically generated

Figure 3.9.1 flag feature

Flag feature is like a switch. Its function is to avoid buggy release. The flow of this can be seen in figure 3.9.1 the flag feature is located at updated = false. So, it is a switch, in this case, false mean on and True means off. So, this can tell the program when to stop writing the data and when to start writing the data. The flow can be seen in figure 3.9.1 first it will ask for input which are the student ID, new payment, and the payment date. And after the opening the fee.txt first we will tell the program that the data is error or still need to be updated which is “updated == false” and after we re-enter all the correct one, we will tell the data is already correct by “updated ==” correct.”

A computer screen with colorful text

Description automatically generated

Figure 3.9.2 Security feature

We have a login feature or the security feature. How this work if the user put the right username and password it will enter. However, if not it will decline. The flow is the program will ask the user for username and password. After getting it from user, the program will check the security.txt which has a username and password inside. If it same with the username or password inside it will let the user continue however if the program found that the username or password is wrong, it will tell the user to re-enter the username and password till correct.

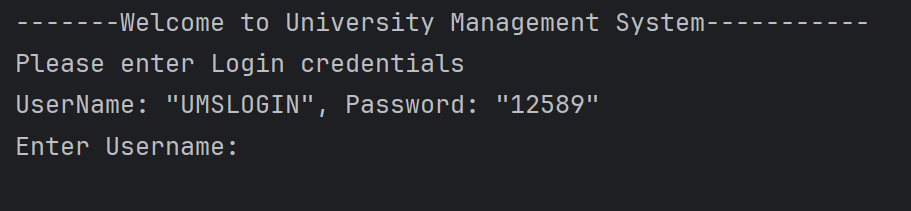
**A screen shot of a computer code

Description automatically generated**Figure 3.9.3 Timestamp

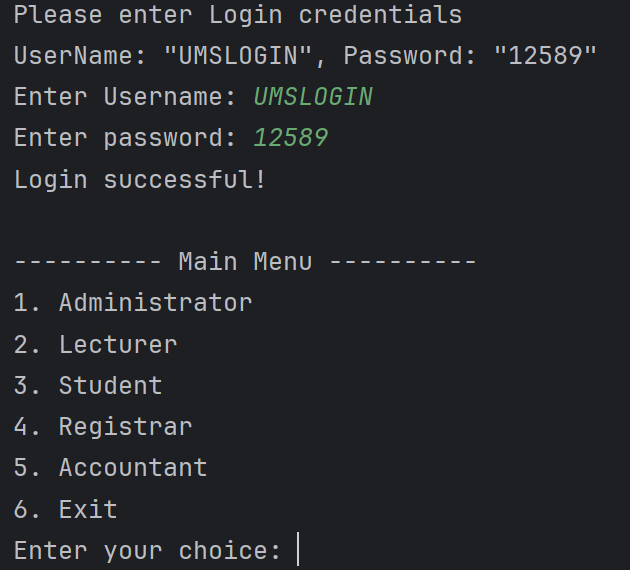
Time stamp is used to give the user information about what time and date use the function. The flow of time stamp is first we need to import date and time and then need to use a variable to represent the datetime and just call the variable to be used. As in figure 3.9.3 after the output come out it will also show the time and date that the output come at.

# **CHAPTER 4: INPUT/OUTPUT SAMPLE**

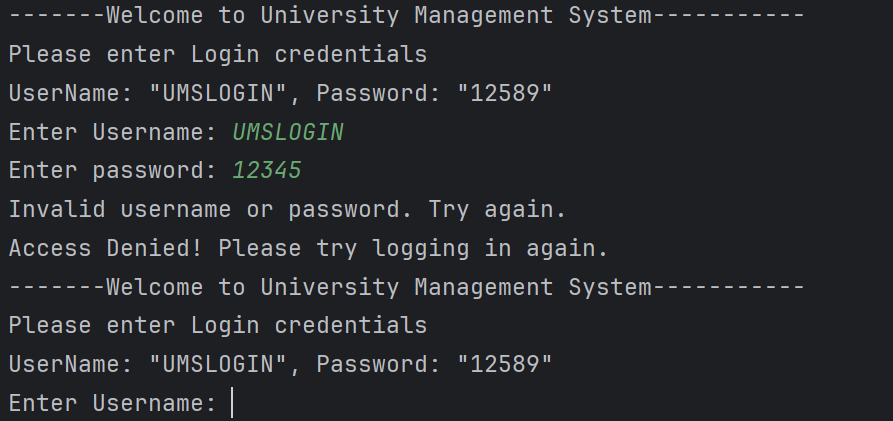
## ***4.1*** *UMS Main Menu*



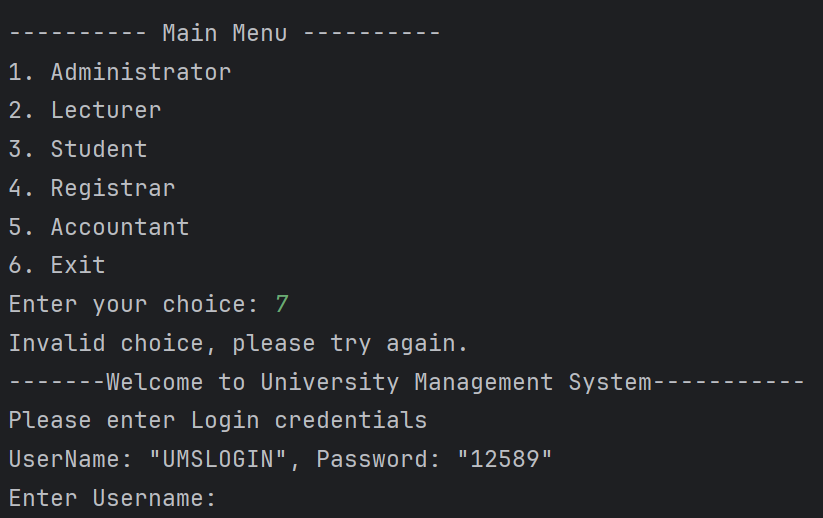
Upon entering the university management system, you are welcomed by this main menu. The user is required to enter the username and password to access the features in the system. The username and password have been provided for simplicity purposes.



Upon entering the username and password correctly, the user is presented with the main menu. Here the user can choose what feature to access based on their role.

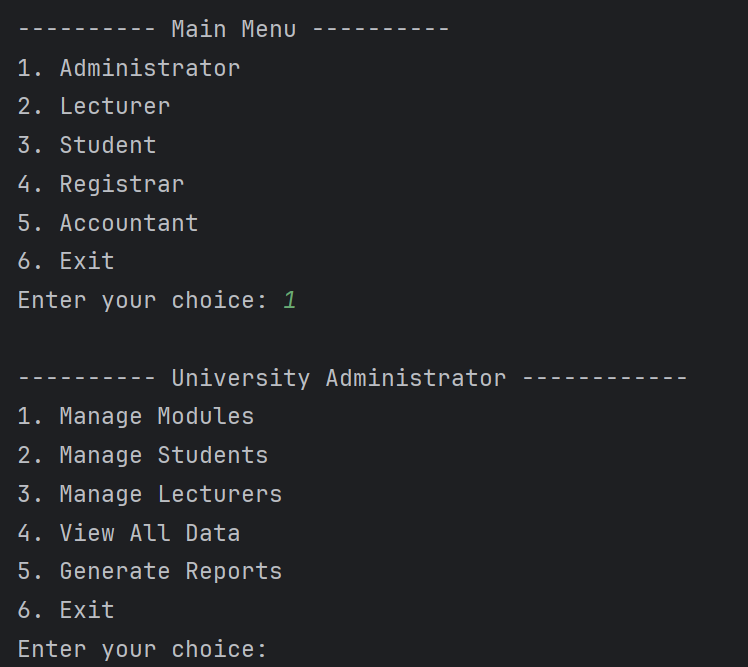


If the user enters an invalid username or password, an error message will be shown stating that the username or password entered was invalid and access will be denied. The user will then be asked to enter the username and password again.

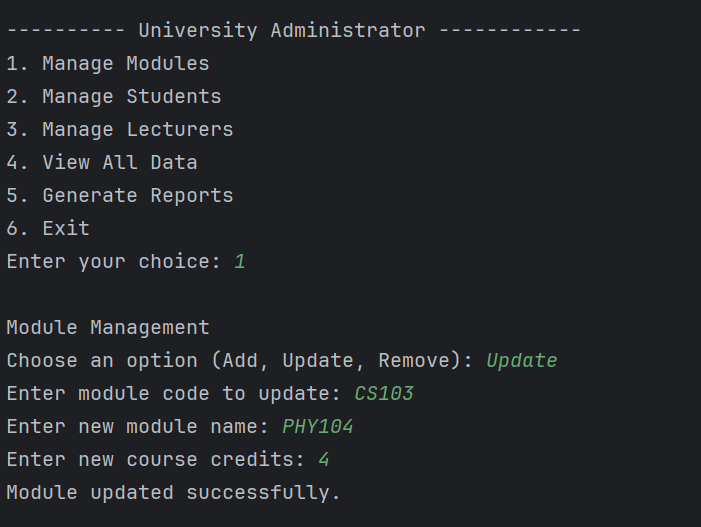
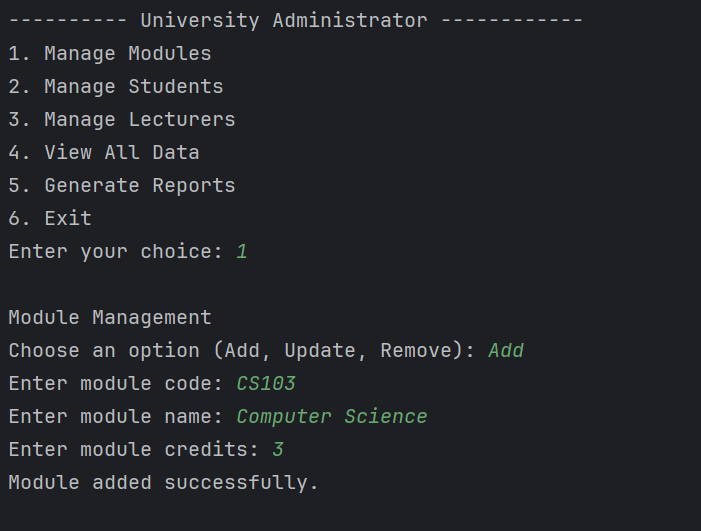


If the user enters an invalid choice in the main menu, an error message will be shown stating the choice was invalid and the user will be asked to enter the username and password once again

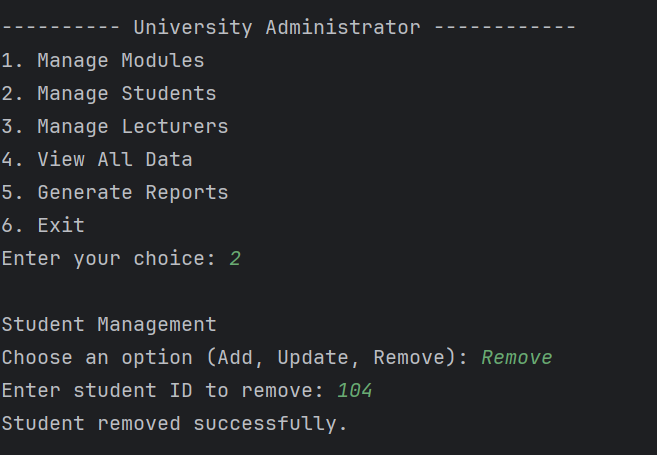
## ***4.2*** *Administrator Role*



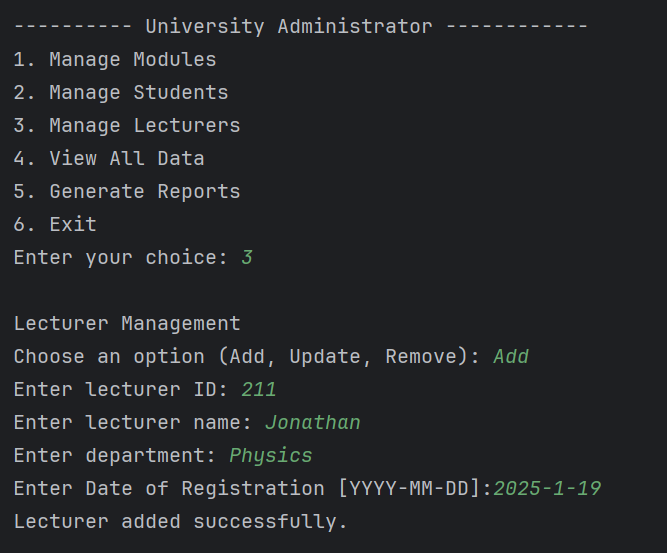
When the user selects Administrator (1) from the main menu it brings up the University Administrator Menu, from where the user will be able to add and remove modules, students, and lecturers, as well as view data and reports.

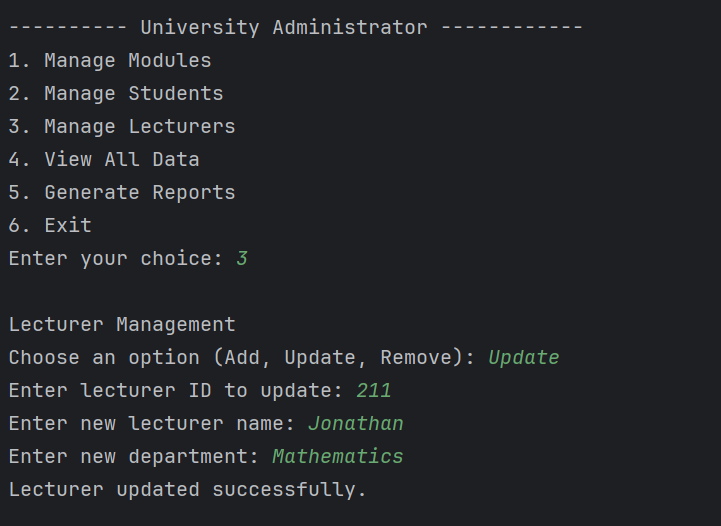


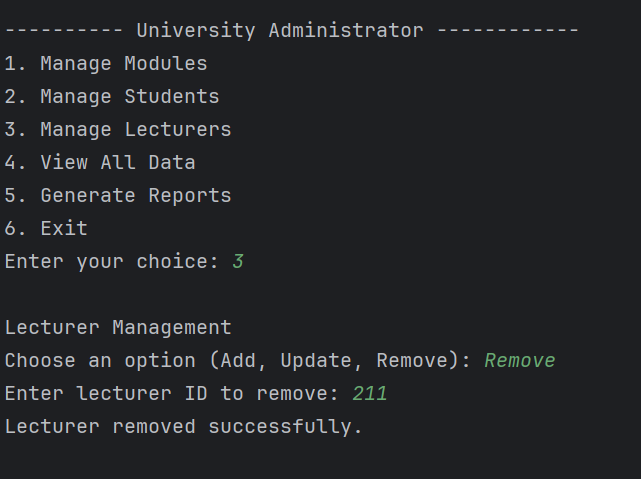
A user can manage the modules by selecting an option either to Add, Update, or Remove a module.



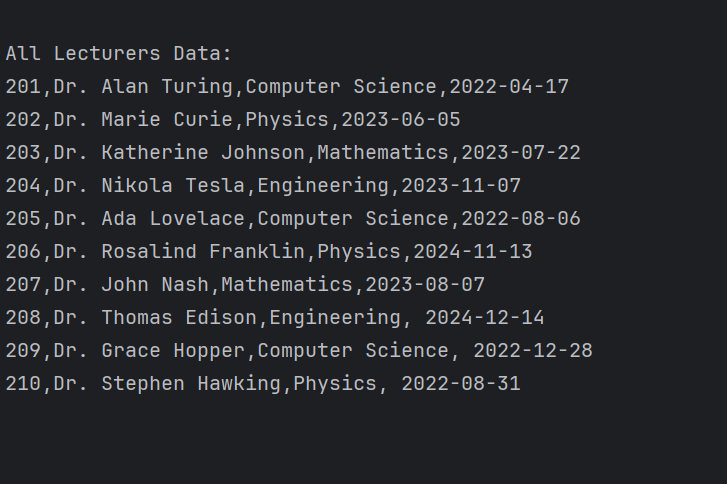
The user can manage students by selecting to Add, Update, or Remove student records.



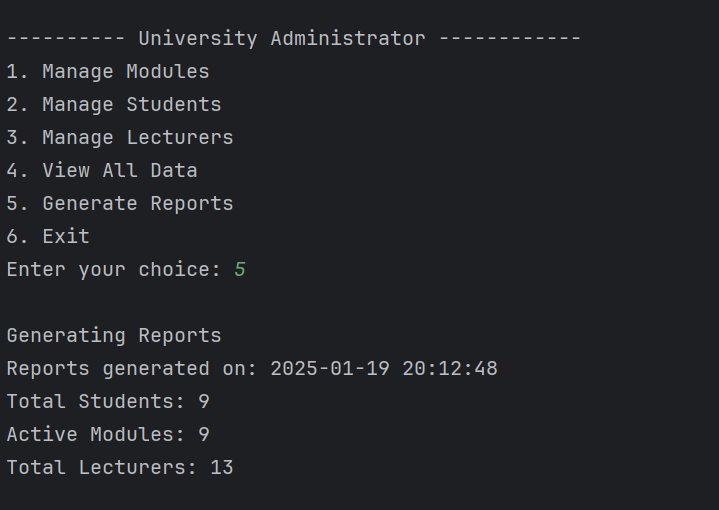




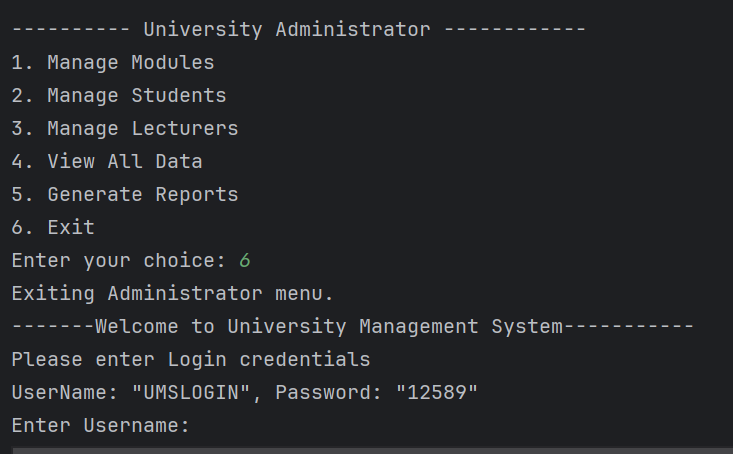
The user can manage lecturers by choosing an option to Add, Update, or Remove lecturer data. In case the user chooses Remove, he will be asked for the ID of the lecturer, and if that ID does not exist, an error message will appear stating that the lecturer was not found.



The user can View all Data which has records of students details, module details, as well as the lecturers details. Aside from that, this system provides the users with access logs that include the dates and times when the said information was accessed. bomb

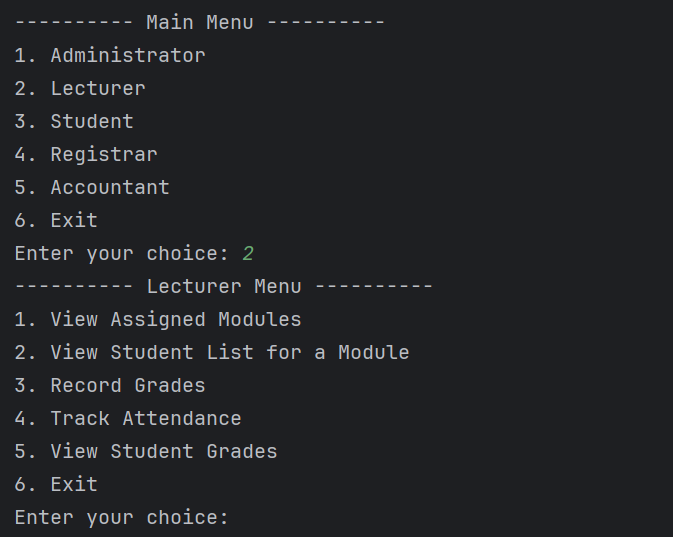


The user can also Generate Reports that will tell the user the Total Students, Active Modules, and the Total Lecturers.

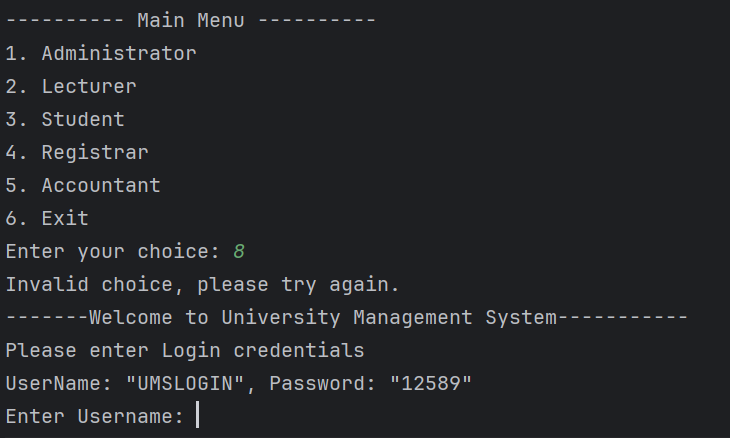


IF the user decides to leave the University Administrator Menu, he can choose Option 6 which will then Exit the user from the Administrator Menu and back to the Login.

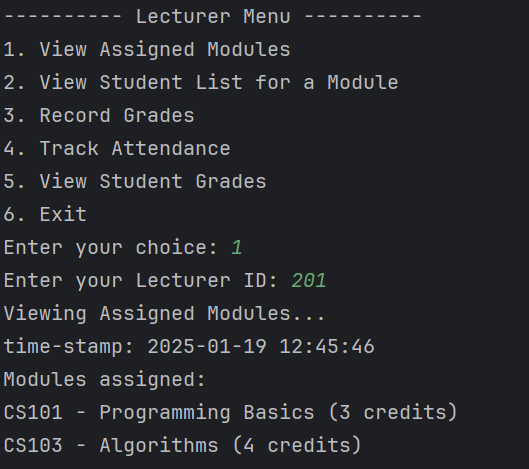
## ***4.3*** *Lecturer Role*



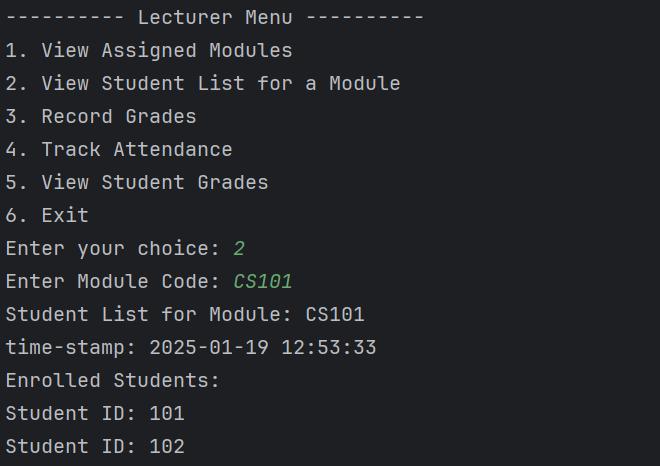
When the user chooses the Lecturer option (2) from the Main Menu, he will then be able to see the Lecturer Menu.



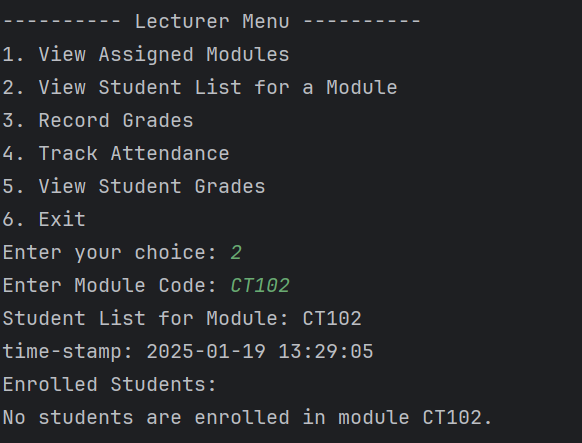
If the user enters an invalid choice, an error message will be shown stating that the choice is invalid, and the user will be asked to enter their choice again.



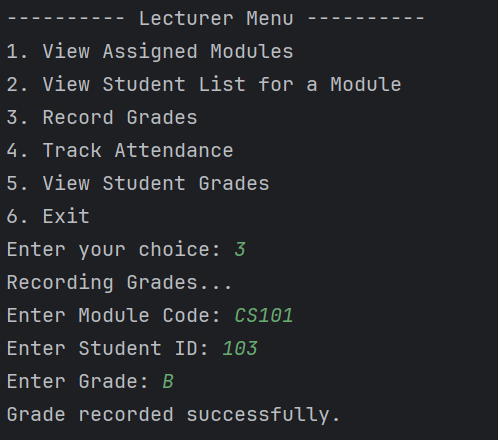
If the user chooses option 1, they will be able to view the assigned modules by entering on of the Lecturer ID that is provided in the Lecturer text file.



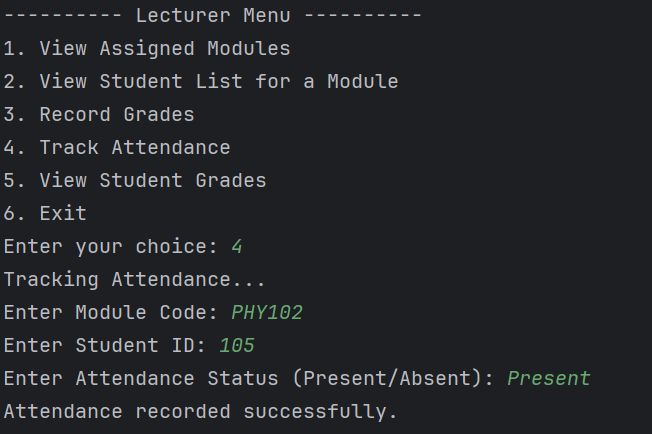
If the user selects option 2, he or she will be able to see list of students who have registered for a particular module by entering Module Code. The system will then show the information of the students who have registered for that Module.



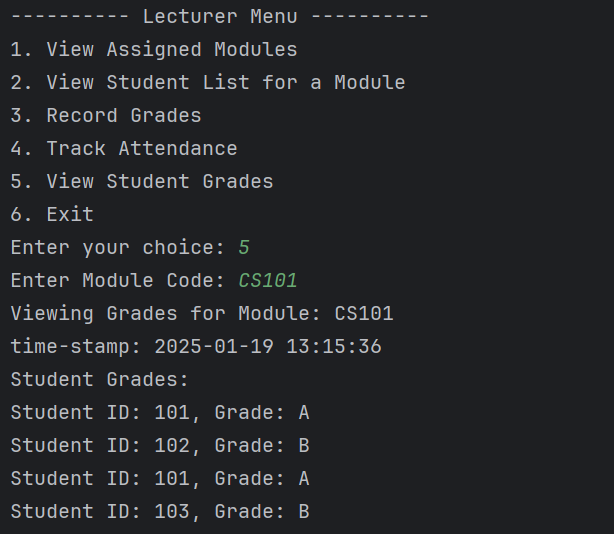
If the user enters a wrong Module Code, he will then be shown an error saying stating that no students are enrolled in this module



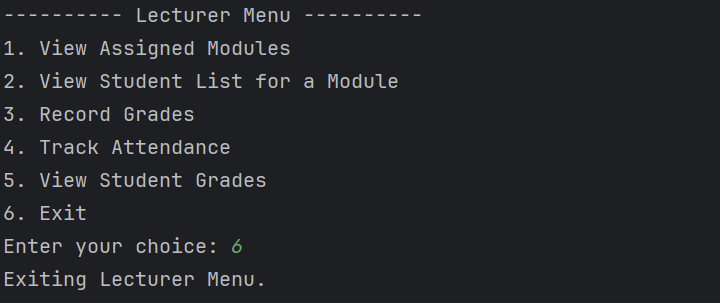
If the User Selects option 3, he will then be able to Record Grades for a student by typing the module Code as well as the Student ID, then the user will be able to choose which grade the student will receive



If the user selects option 4, the user can track attendance by entering the Module Code, Student ID, and choosing if the student is "Present" or "Absent." The attendance record is then saved in the Attendance file.

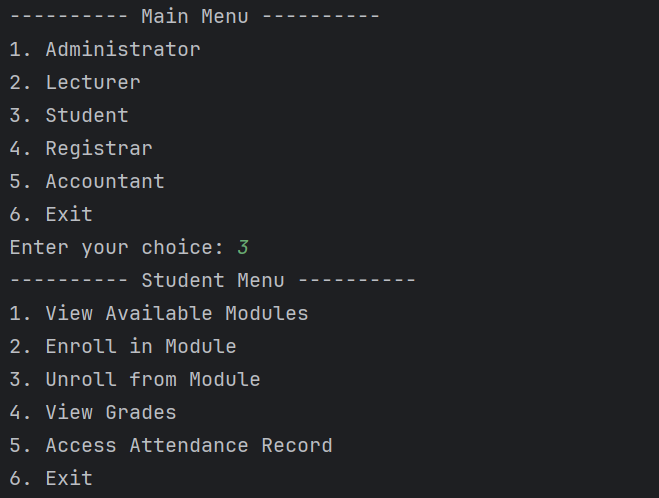


If the user selects option 5, the user can view Students grades for a course by simply typing the Module Code.

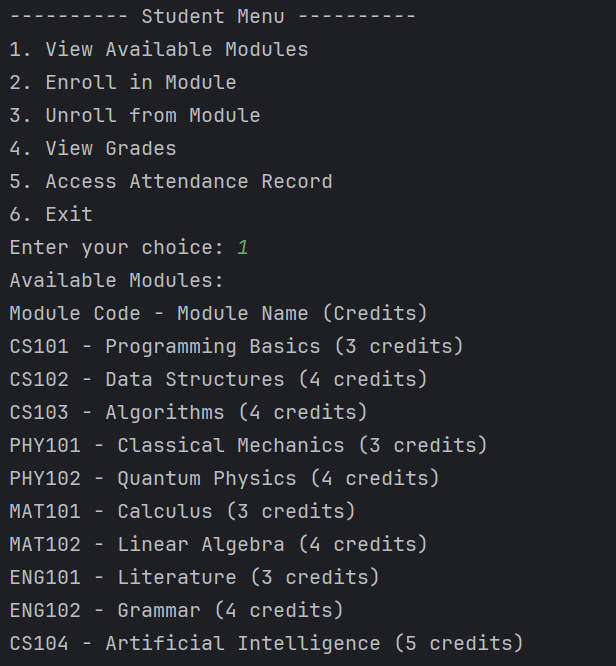


If the user decides to leave the Lecturer Menu, he can choose option 6 where it will then Exit the Lecturer Menu.

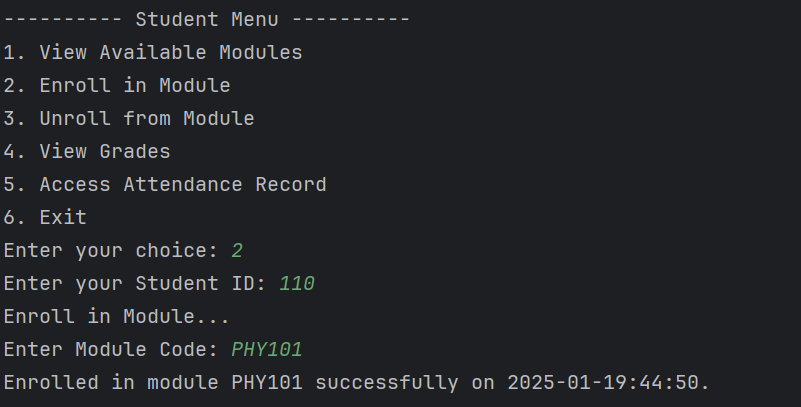
## ***4.4*** *Student Role*

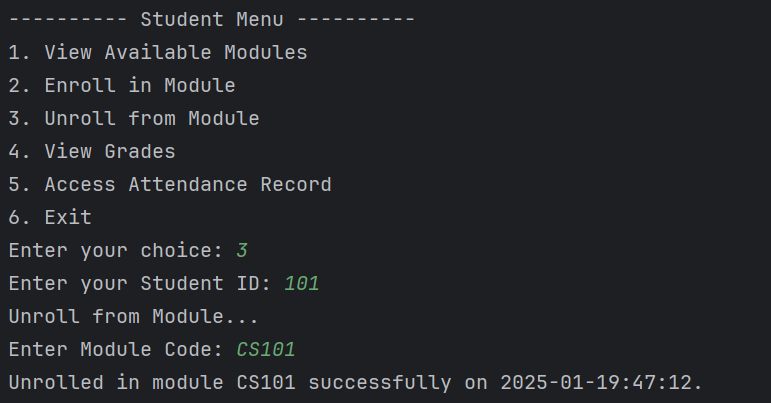


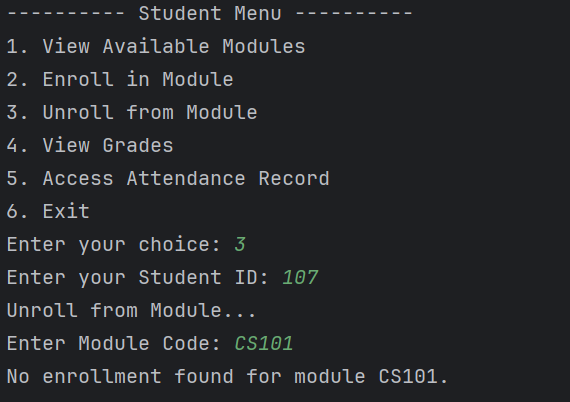
When the user selects the student option (3) from the Main Menu, they will be directed to the Student Menu.



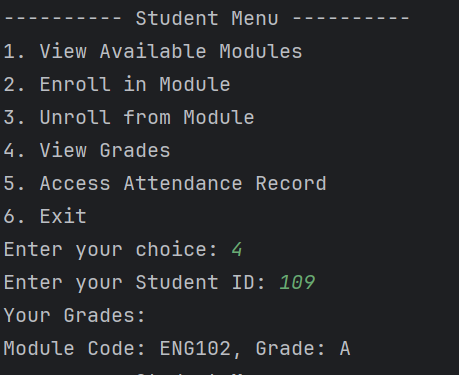
If the user selects option 1, the user gets a list of available modules as the output. The system shows the Module Code, Module Name and the number of credit hours given to each module. This enables the student to view all courses available for them to take before having to sign up for them.

 If the user selects option 2, they can register for a module by entering the student ID and the Module code. When the enrolment is affected, a message confirming this will be generated accompanied with the time stamp.

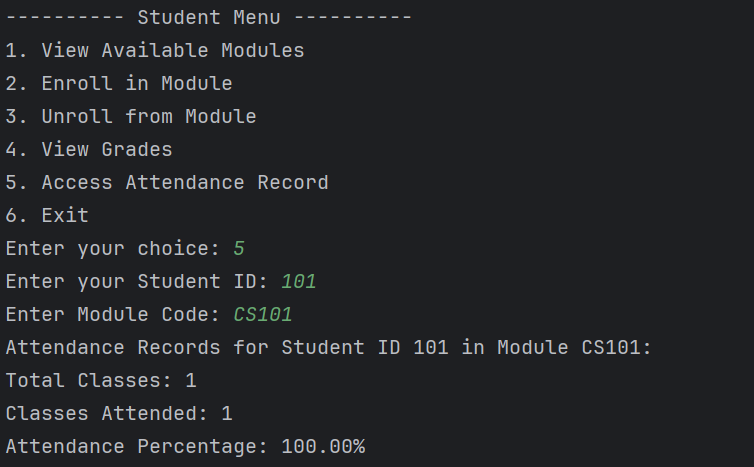
 When the user chooses option 3, they can unroll from a module by typing in their Student ID and the Module Code.



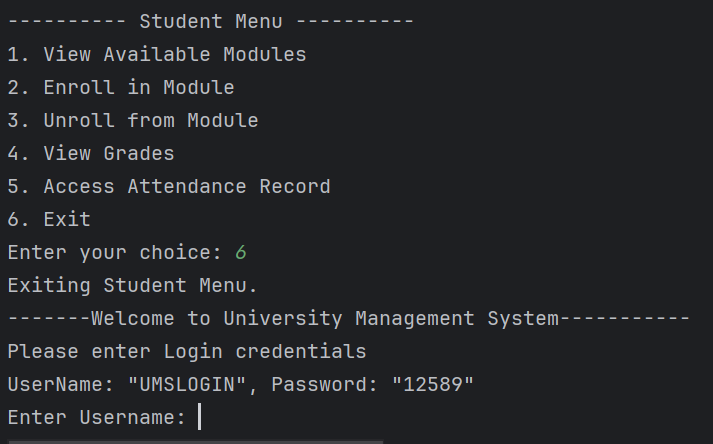
In the case that the student is not accorded to the specified module, the system gives a message that no enrolment was found. If the unrolling is complete, a message confirming this, along with the time stamp is displayed.



When the use chooses option 4, they can view their grades after entering their Student ID. This means the system will show module codes and the scores which has been obtained for each module.



If the user chooses option 5, he will be able to check his attendance record of a certain module. When the student types of his Student ID and the Module Code the system will show the Total classes, number of classes attended and the attendance percentage for the stated module.

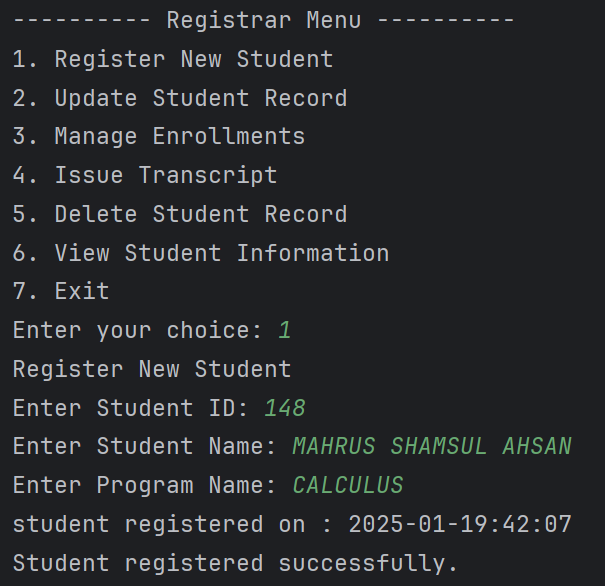


If the user selects option 6, they exit the Student Menu and return to the Main Menu. The system logs out the student from the current session and displays the login screen.

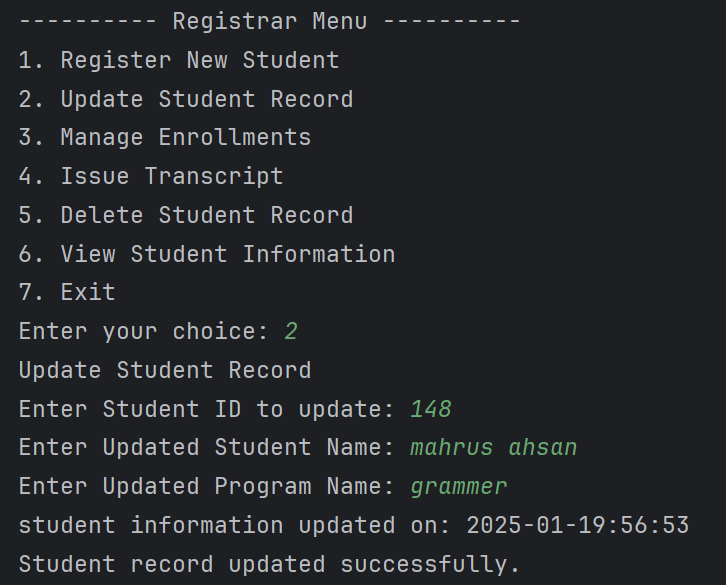
## ***4.6*** *Registrar Role*



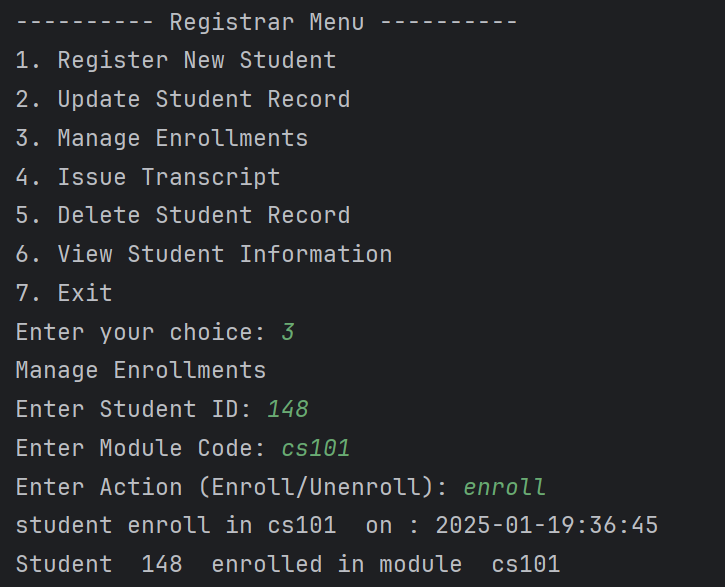
Upon choosing number 5, the user will be shown the registrar menu. Here the user will have 7 options to choose from.



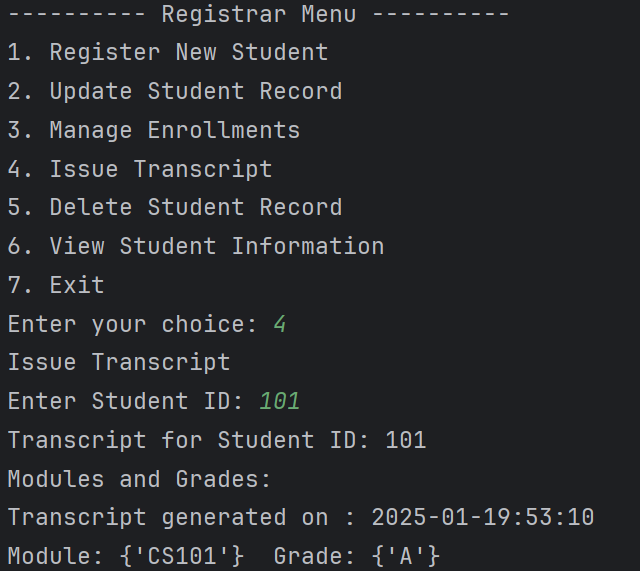
Choosing option 1 will allow the user to register a new student. The user will have to enter the new student ID, name of the new student and the program name. If added successfully, a message will be shown stating that the student has been registered successfully.



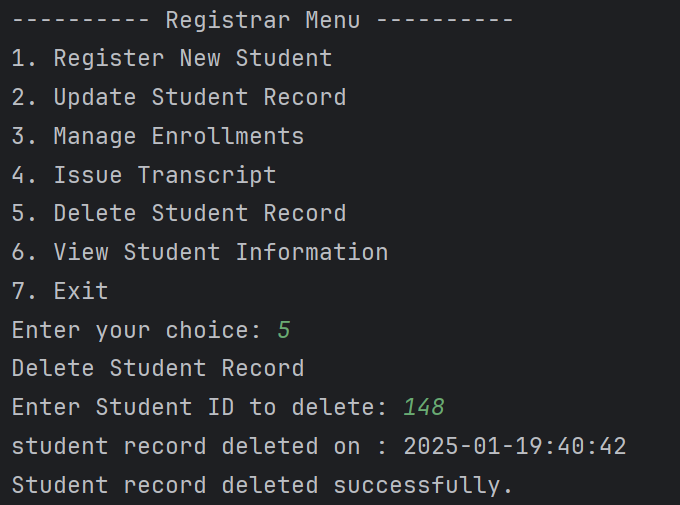
Choosing option 2 will allow the user to update student record. The user will have to enter the student ID, updated student name and updated program name. Once successfully added a time stamp will be shown and a message will also be shown stating the student record was updated successfully.



Choosing option 3 will allow the user to manage the enrollment of students. The user will have to enter the student ID, module code, and whether to enroll or unenroll. A time stamp and a message will be shown if the student was successfully enrolled or unenrolled.



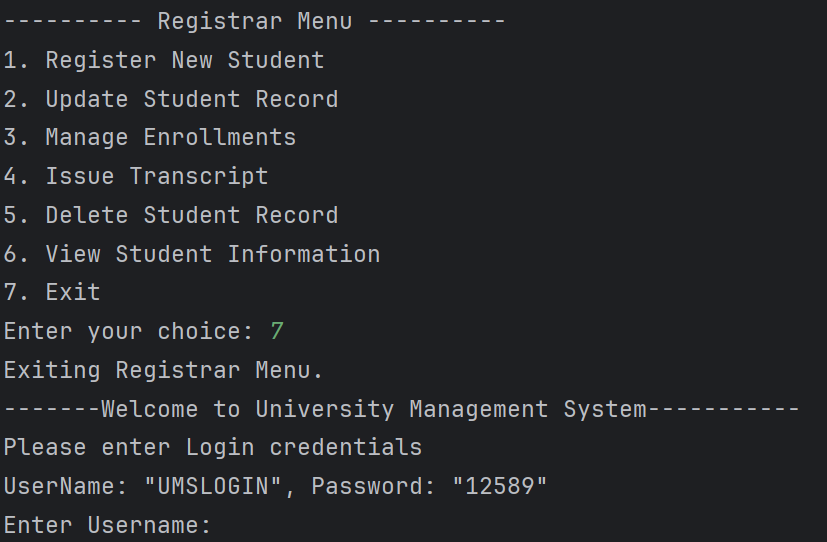
Choosing option 4 will allow the user to issue transcript for a student. The user will have to enter the student ID. Then a transcript will be generated stating the student ID, and the grade of the student. A time stamp will also be generated.



Choosing option 5 will allow the user to delete student record. The user will have to enter the student ID. A time stamp will and generated and a message will be shown if the deletion was successful.

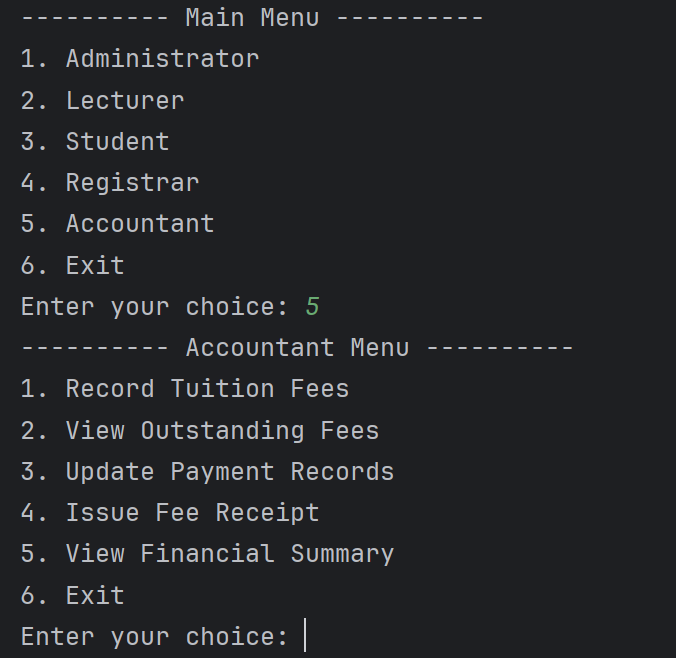


Choosing option 6 will allow the user to view any student's information. The user will have to enter the student ID. The user will then be shown the student ID, name, and program the student is enrolled in. A time stamp will be generated.

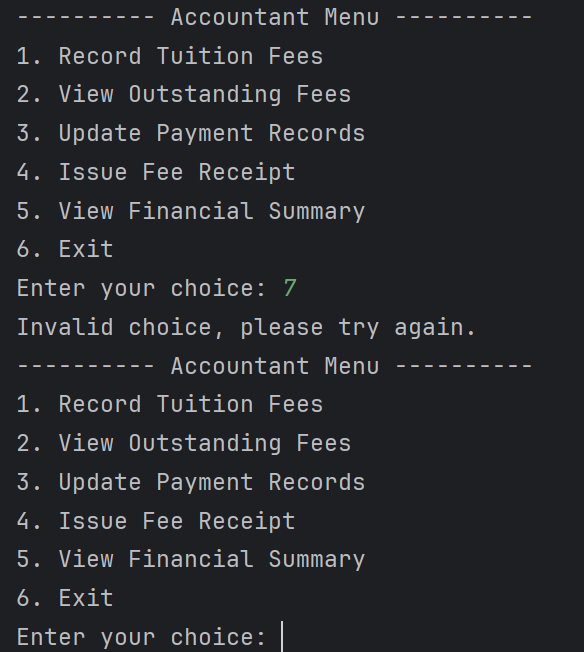


Choosing option 7 will allow the user to exit the registrar menu and return to the UMS login menu.

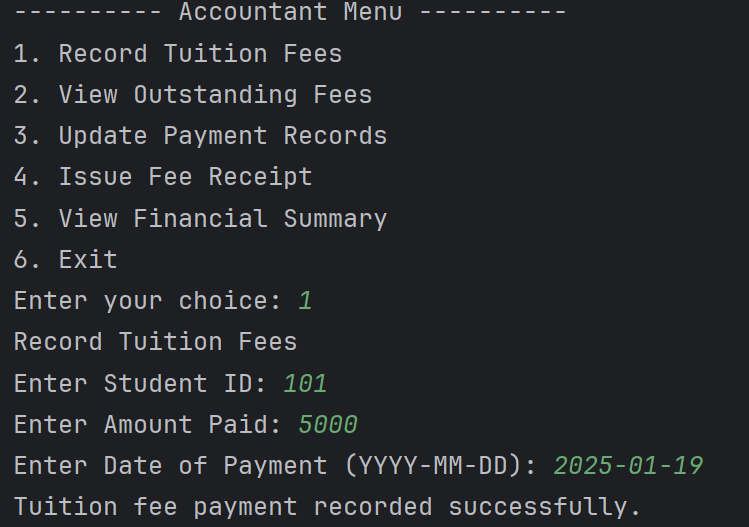
## ***4.6*** *Accountant Role*



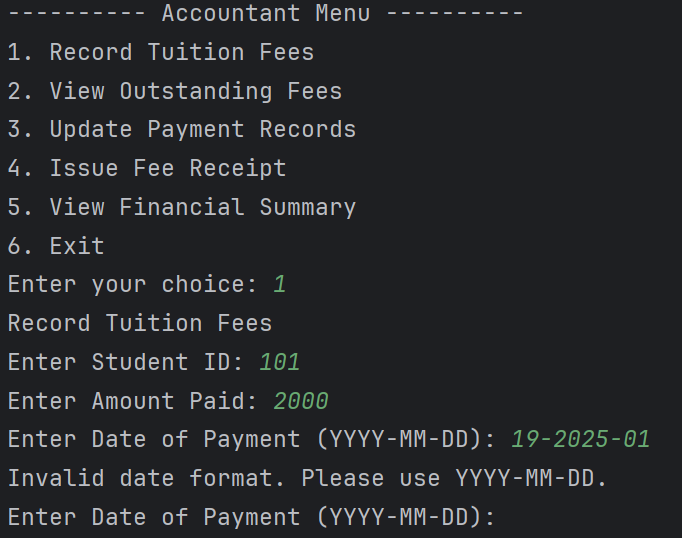
Upon entering number 5 to access the accountant feature, the user is presented with the accountant menu. The user can choose six options from this menu.



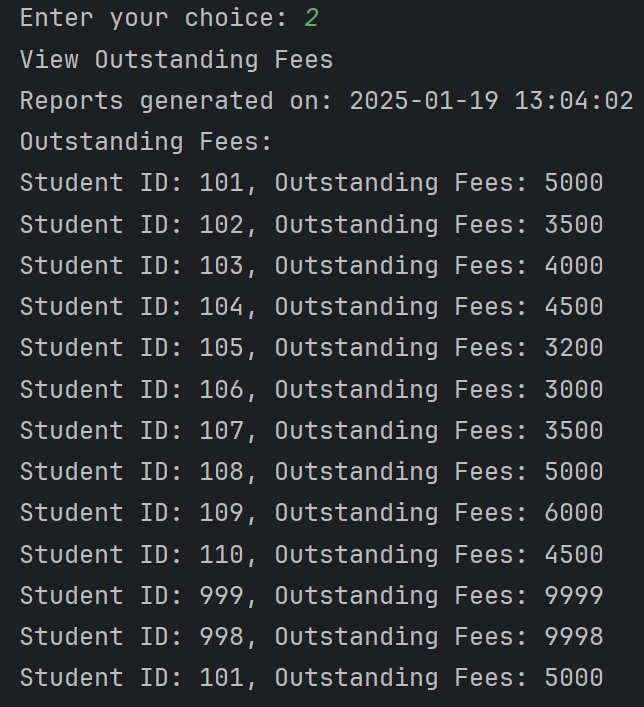
If the user enters an invalid choice, an error message will be shown stating that the choice is invalid, and the user will be asked to enter their choice again.



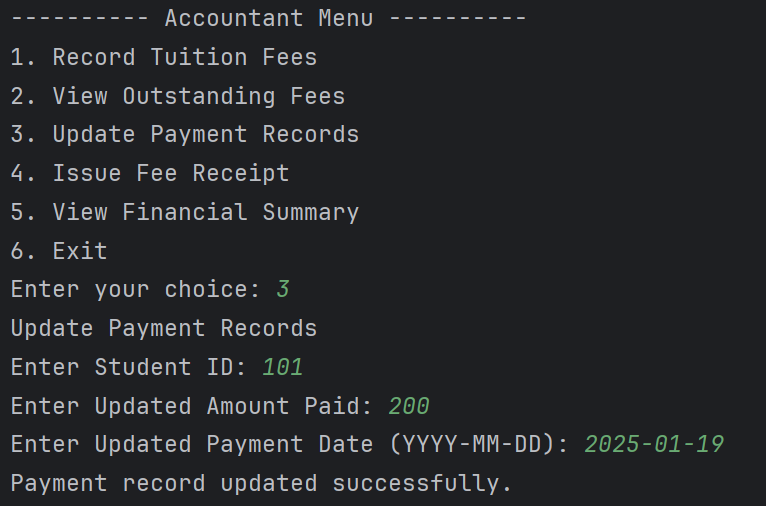
Choosing option 1 will allow the user to record tuition fees. The user will be asked to enter the student ID, the amount paid, and the date of payment. Once the user has entered the details, they will get a message if the record has been added successfully.



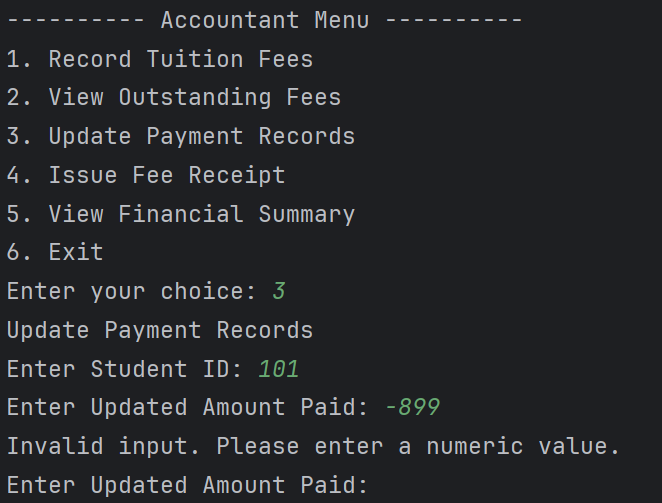
If the user enters the date in a wrong format, an error message will be shown, and the user will be asked to enter the date again.



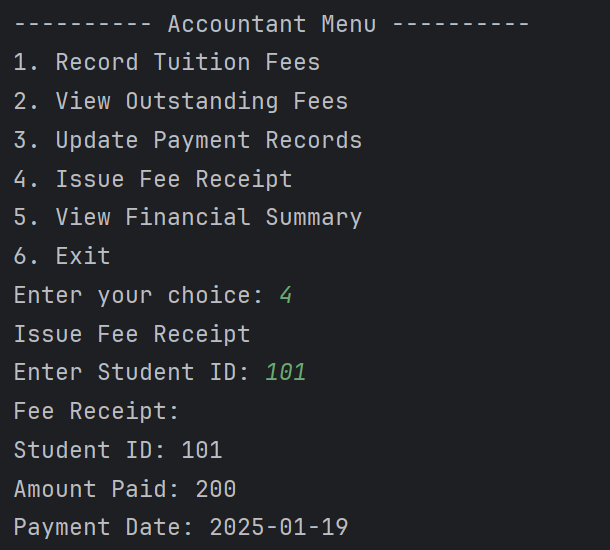
Choosing option 2 will allow the user to view all outstanding fees. They will be able to see the student ID and the amount of money outstanding. The time stamp will be shown as well.



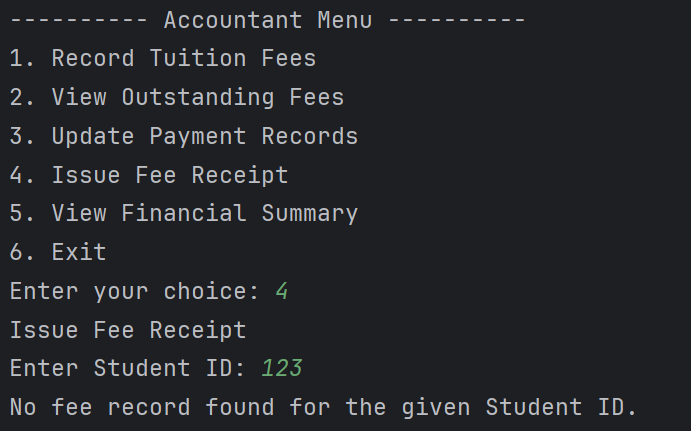
Choosing option 3 will allow the user to update any payment record. The user will be asked to enter the student ID, updated amount paid, and the payment date. If updated successfully, the user will be shown a message stating the record has been updated successfully.



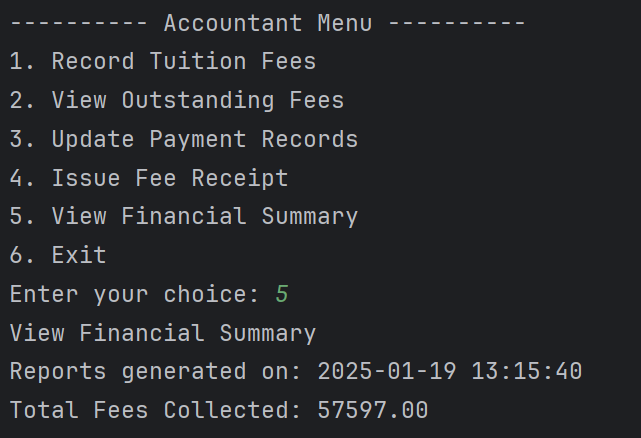
If the user enters an invalid input for the amount paid an error message will be shown and the user will be asked to enter the amount again.



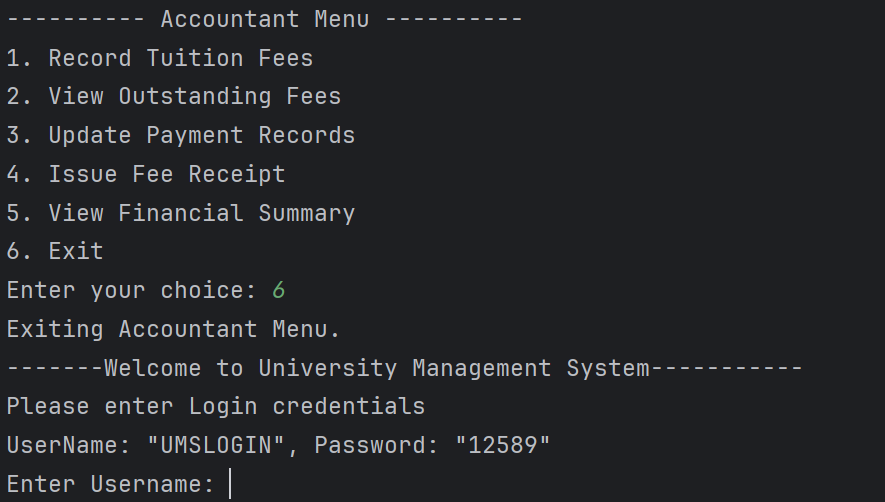
Choosing option 4 will allow the user to issue a fee receipt. The user will be asked to enter the student ID. The receipt will then be issued stating the student ID, the amount paid and the date of payment.



If the user enters the wrong student ID, an error message will be shown stating that no fee record found for the given student ID.



Choosing option 5 will allow the user to view the financial summary. The user will be shown the time stamp of the report and the total fees collected.



Choosing option 6 will allow the user to exit the accountant menu and go back to the login menu. The user will be required to login again in order to choose any of the features.

# **CONCLUSION**

This University Management System (UMS) is a helpful solution for managing nearly all the operations of the university and has more specific operations for administrators, lecturers, students, registrars, and accountants to perform in their various roles. Developed with Python, the system uses such basic programming activities as functions, files, and input/output validation wherein no additional libraries are used for data processing. Combined with the isolated deployments, this means that each feature is separated into its own functions, which provides the beginning for a clean, structured, readable, and maintainable codebase. This way, roles reduce security and usability risks and allow users to see only the functionality that matches their positions. Moreover, error checking and input formatting reduce the likelihood of errors and make data processing more accurate. However, as the system advances and works with high volumes of data, the use of simple databases as text files for data storage (CSV) will be an issue.

Changing to the database-based solution would dramatically improve both scalability and performance. But as basic for current implementation, making more improvement could enhance the usability of the system, more function such as logging, detailed report and GUI need to be added on the system. Thus, UMS system has confirmed a reasonable approach to developing basic programming concepts and offers a stable option for managing university activities.

This is especially ideal for small and medium educational facilities; and there exist quite pronounced possibilities for the subsequent optimization with respect to the growth in data storing capacity, response speed, as well as the degree of easiness for users.

# **REFERENCES**

GeeksforGeeks. (2024, December 8). *Python Try Except*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python-try-except/

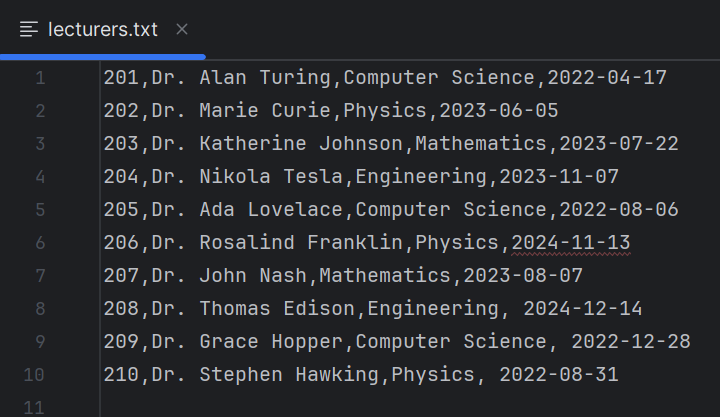
GeeksforGeeks. (2024, August 9). *Python3 – if , if..else, Nested if, if-elif statements*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python3-if-if-else-nested-if-if-elif-statements/?ref=outind

Programiz. (n.d.). *Python File Operation*. Retrieved from Programiz: https://www.programiz.com/python-programming/file-operation

W3Schools. (n.d.). *Python Lists*. Retrieved from W3Schools: https://www.w3schools.com/python/python\_lists.asp

APPENDIX A: TEXT FILES

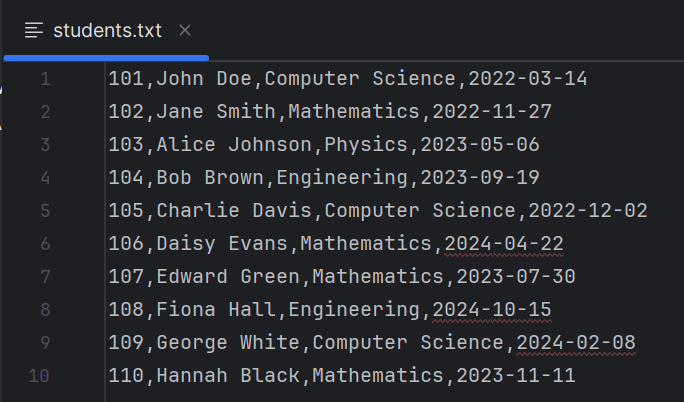
**Lecturer Text file**



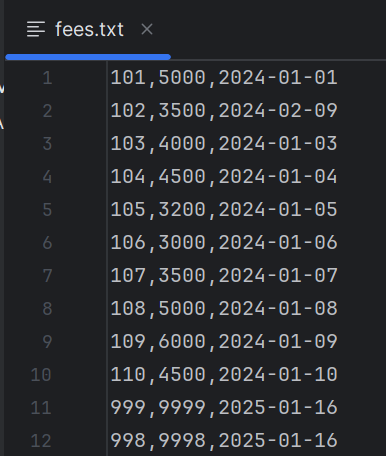
**Modules Text file**



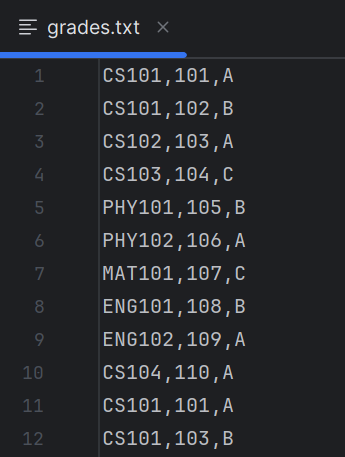
**Students Text file**



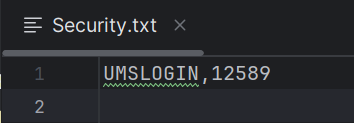
**Fees Text file**



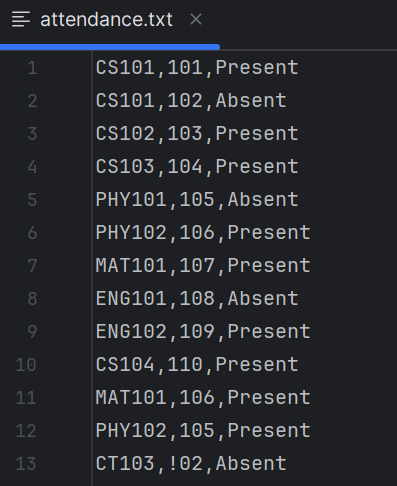
**Grades Text file**



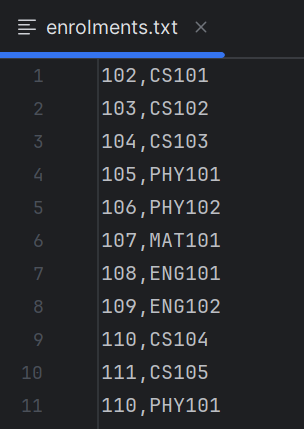
**Security Text file**



**Attendance Text file**



**Enrolments Text file**



# **APPENDIX B: WORKLOAD MATRIX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task/Activity** | **Description** | **Assigned Member** | **Percentage** |
| **Requirement Analysis** | Define project objectives, scope, and requirements. | |  | | --- | | Ahmed Mirahusain Alvi TP084807 | | ANDREW WIJAYA TP079319 | | Mahrus Shamsul AhsanTP085562 | | Abdallah Mohamed Mahmoud Mohamed Mahmoud TP085097 | | Mohammed Yousef Mohammed Mohammed TP085042 | | |  | | --- | | 20% | | 20% | | 20% | | 20% | | 20% | |
| **System Design** | Flowcharts **/** Pseudocode | |  | | --- | | Ahmed Mirahusain Alvi TP084807 | | ANDREW WIJAYA TP079319 | | Mahrus Shamsul AhsanTP085562 | | Abdallah Mohamed Mahmoud Mohamed Mahmoud TP085097 | | Mohammed Yousef Mohammed Mohammed TP085042 | | |  | | --- | | 20% | | 20% | | 20% | | 20% | | 20% | |
| **Backend Development** | Implement core functional | |  | | --- | | Ahmed Mirahusain Alvi TP084807 (Administrator) | | ANDREW WIJAYA TP079319 (Register) | | Mahrus Shamsul AhsanTP085562 (Accountant) | | Abdallah Mohamed Mahmoud Mohamed Mahmoud TP085097  (Lecturer) | | Mohammed Yousef Mohammed Mohammed TP085042  (Student) | | |  | | --- | | 20% | | 20% | | 20% | | 20% | | 20% | |
| **Documentation** | |  | | --- | | Table of contents | | Acknowledgement | | Abstract | | Introduction and assumptions of your system developed | | Explanation of programming concepts applied with sample of source code based-on the system developed. | | Screenshots of sample input/output with explanation | | Conclusion | | |  | | --- | | Ahmed Mirahusain Alvi TP084807 | | ANDREW WIJAYA TP079319 | | Mahrus Shamsul AhsanTP085562 | | Abdallah Mohamed Mahmoud Mohamed Mahmoud TP085097 | | Mohammed Yousef Mohammed Mohammed TP085042 | | |  | | --- | | 20% | | 20% | | 20% | | 20% | | 20% | |