PYTHON PROGRAMMING

Intake Code: APU1F2311CS

Lecturer: Sathiapriya A/P Ramiah

Submission Date: Sunday, February 25th 2024

GROUP MEMBERS	
Name	**TP Number**
Abdullah Al Faiyaz	TP 077873
Emmanuel Naranendhya Tyatan	TP 078574
Faiyad Mahabub Tasin	TP 077983
Yeremia Samuel Aditya Marfianto	TP 078403

APU CAFE MANAGEMENT SYSTEM

Table of Content

Introduction and Assumptions

Introduction

The APU Cafe program is aimed at helping or advancing students in their programming language learning. The program assigns trainers to students who are looking to learn more about the languages Python, Java, or C. This program uses an external library called tabulate to display some of the tables. Everyone who wants to run this program must first install it using this command: python -m pip install tabulate. The entry point of this program is in main.py.

Assumptions

Unregistered administrators are able to register themselves as new administrators. A newly registered administrator will have the password “123”. As administrators, users are able to register a new trainer with a starting module. Afterward, they can add more modules to that trainer, even ones with the same language and level. A trainer has no limit on how many modules they can teach at once.

A trainer’s salary is calculated based on the level of the module they’re teaching and how many students are enrolled in that module. A trainer teaching a beginner-level module will be paid **1000** for each student that is enrolled. A trainer teaching an intermediate-level module will be paid **1500** for each student that is enrolled. A trainer teaching an advanced level module will be paid **2000** for each student that is enrolled. If a trainer teaches multiple modules with different levels, the salary will be the sum of the previously mentioned calculations.

All the students will be registered in the system by the lecturer before the students start using the system. All students would be briefed about the system before they start using it. For example the students would know that they won't be able to view the schedule if they have not paid for their modules ,they need to change the password the first time they log in as it is 123, and they are also able to request for as many modules as they wish.

Design of The Program

Abdullah Al Faiyaz

api/student.py

```
try
 LOAD students from students json file using the json module
 READ students
 LOAD subjects from modules json file using the json module
 READ subjects
Catch FileNotFoundError as e
 Print “utils.red(f"Error: The {e.filename.replace("database/",
")} was not found.")”

Function change_password (student_id, new_password)
 LOOP i FROM 0 TO len(students) - 1 STEP 1
   IF students[i]["id"] == student_id THEN
     SET student["password"] = new_password
   ENDIF
 ENDLOOP
 SAVE students to students json file using the json module
ENDFUNCTION


Function change_contact (student_id, new_contact)
 LOOP student FROM 0 TO len(students) - 1 STEP 1
   IF student["id"] == student_id THEN
     SET student["contact_number"] = new_contact
   ENDIF
 ENDLOOP
 SAVE students to students json file using the json module
ENDFUNCTION


Function view_schedule (student_id)
```

```
 LOOP student FROM 0 TO len(students) - 1 STEP 1
   IF student["id"] == student_id THEN
     SET module_ids = [student["modules"][i] LOOP i FROM 0 TO
len(student["modules"]) -1 STEP 1]
     LOOP i FROM 0 TO len(student["modules"]) - 1 STEP 1
       SET module_id = student["modules"][i]
       PRINT "{i + 1}. {subjects[module_id]['name']}"
     ENDLOOP
     PROMPT user to Enter your choice:
     READ module_num - 1
     SET module_id = module_ids[module_num]
     LOOP i FROM 0 TO
len(subjects[list(subjects)[i]]["students"]) - 1 STEP 1
               IF subjects[list(subjects)[i]]["students"][i]["id"]
     == student_id THEN
                  PRINT red(f"Please do your payment. You still
     have outstanding: {get["invoice"]}"))
                   CASE OF utils.input_number("Do you want to pay
     now?\n1. Pay\n2. Exit\nEnter choice: ", 2)
                  = 1: CALL payment(student_id)
                  = 2: return False
                     ENDCASE
              ENDIF
            ENDLOOP
     SET matching_schedule = [subjects[module_id]["schedule"] [i]
LOOP i FROM 0 TO len(subjects[module_id]["schedule"]) -1 STEP 1]
     IF len(matching_schedule) == 0 THEN
         Print "There is no schedule given by the trainer at the
moment."
         Return
     ENDIF
     LOOP i FROM 0 TO len(matching_schedule) -1 STEP 1
       row = (list(matching_schedule[i].values()))
       row.insert(0, subjects[module_id]['name'])
     ENDLOOP
```

```
     SET rows = [row]
     SET headers = ["Module", "Date", "Time", "Location"]
     SET table = tb.tabulate(rows, headers=headers,
tablefmt="grid")
     Print table
   ENDIF
ENDFUNCTION


FUNCTION view_all_schedule(student_id)
 SET rows = []
 LOOP i FROM 0 TO len(students) - 1 STEP 1
   IF students[i]["id"] == student_id THEN
     SET module_ids = [student["modules"][i] LOOP i FROM 0 TO
len(student["modules"]) -1 STEP 1]
     LOOP i FROM 0 TO len(module_ids) – 1 STEP 1
           LOOP i FROM 0 TO len(subjects) – 1 STEP 1
            LOOP i FROM 0 TO
len(subjects[list(subjects)[i]]["students"]) - 1 STEP 1
              IF subjects[list(subjects)[i]]["students"][i]["id"]
     == student_id THEN
                  PRINT red(f"Please do your payment. You still
     have outstanding: {get["invoice"]}"))
                   CASE OF utils.input_number("Do you want to pay
     now?\n1. Pay\n2. Exit\nEnter choice: ", 2)
                  = 1: CALL payment(student_id)
                  = 2: return False
                     ENDCASE
             ENDIF
           ENDLOOP
            IF module_id == subjects[i] THEN
              SET matching_schedule =
[subjects[module_id]["schedule"] [i] LOOP i FROM 0 TO
len(subjects[module_id]["schedule"]) -1 STEP 1]
               IF len(matching_schedule) == 0 THEN
```

```
                    Print "There is no schedule given by the
trainer at the moment."
                    Return
                ENDIF
                LOOP i FROM 0 TO len(matching_schedule) -1 STEP 1
                            row =
(list(matching_schedule[i].values()))
                            row.insert(0,
subjects[module_id]['name'])
                ENDLOOP
                APPEND row to rows
              ENDIF
            ENDLOOP
      ENDLOOP
    ENDIF
 ENDLOOP
 SET headers = ["Module", "Date", "Time", "Location"]
 SET table = tb.tabulate(rows, headers=headers, tablefmt="grid")
 Print table
ENDFUNCTION


FUNCTION request (student_id)
 SET request_id = generate_unique_uuid()
 SET request_data = {
       "sender": student_id,
       "status": "pending",
       "module_request_type": module,
       "module_request_level": level,
    }
 LOOP i FROM 0 TO len(students) - 1 STEP 1
    IF students[i]["id"] == student_id THEN
      student["requests"][request_id] = request_data
    ENDIF
 ENDLOOP
```

```
 SAVE students to students json file using the json module
ENDFUNCTION

FUNCTION delete_requests (student_id, request_id)
 LOOP i FROM 0 TO len(students) - 1 STEP 1
   IF students[i]["id"] == student_id THEN
       IF student["requests"][request_id]["status"] == "pending"
THEN
           DELETE student["requests"][request_id]
           SAVE students to students json file using the json
module
       ELSE
           `return False
       ENDIF`
 ENDLOOP
ENDFUNCTION


FUNCTION  show_request_status (student_id)
   LOOP i FROM 0 TO len(students) - 1 STEP 1
       IF students[i]["id"] == student_id THEN
           LOOP i FROM 0 TO
len(list(student["requests"].values())) - 1 STEP 1
               PRINT "f"{i+1}. Module:
{requests['module_request_type']}, Level:
{requests['module_request_level']}, Status:
{requests['status']}""
           ENDLOOP
           return len(list(student["requests"].values()))
       ENDIF
   ENDLOOP
ENDFUNCTION


FUNCTION show_invoice(student_id)
```

```
    SET module_id_and_invoice = []
    LOOP i FROM 0 TO len(students) - 1 STEP 1
        IF student["id"] == student_id THEN
            LOOP i FROM 0 TO len(student["modules"])-1 STEP 1
                LOOP i FROM 0 TO
len(subjects[student["modules"][i]]["students"]) -1 STEP 1
                    IF subjects[module]["students"][i]["id"] ==
student_id THEN
                        IF
subjects[module]["students"][i]["invoice"] == 0 THEN
                            continue
                        ENDIF
                        module_id_and_invoice.append([ module,
student_module_data["invoice"] ])
                        PRINT f"{i + 1}. Module name
{subjects[module]['name']}, Outstanding:
{student_module_data['invoice']}"
                    ENDIF
                ENDLOOP
            ENDLOOP
        ENDIF
    ENDLOOP
    IF len(module_id_and_invoice) == 0 THEN
        PRINT "No outstanding invoices."
        return False


FUNCTION  payment(student_id)
    SET module_id_and_invoice = []
    LOOP i FROM 0 TO len(students) - 1 STEP 1
        IF student["id"] == student_id THEN
            LOOP i FROM 0 TO len(student["modules"])-1 STEP 1
                LOOP i FROM 0 TO
len(subjects[student["modules"][i]]["students"]) -1 STEP 1
```

```
                    IF subjects[module]["students"][i]["id"] ==
student_id THEN
                        IF
subjects[module]["students"][i]["invoice"] == 0 THEN
                            continue
                        ENDIF
                        module_id_and_invoice.append([ module,
student_module_data["invoice"] ])
                        PRINT f"{i + 1}. Module name
{subjects[module]['name']}, Outstanding:
{student_module_data['invoice']}"
                    ENDIF
                ENDLOOP
            ENDLOOP
        ENDIF
    ENDLOOP
    IF len(module_id_and_invoice) == 0 THEN
        PRINT "No outstanding invoices."
        return
    ENDIF
    SET choice = input_number("For which module do you want to
make the payment for? ", len( module_id_and_invoice )) - 1
    SET selected_module_id = module_id_and_invoice[choice][0]
    SET to_pay = input_number("How much do you want to pay? ",
module_id_and_invoice[choice][1])
    LOOP i FROM 0 TO
len(subjects[selected_module_id]["students"]) -1 STEP 1
        SET student_module_data["invoice"] -= to_pay
    PRINT green("Payment successful, Thank you")
    SAVE students to students json file using the json module
ENDFUNCTION

FUNCTION get (student_id)
   LOOP i FROM 0 TO len(students) - 1 STEP 1
       IF students[i]["id"] == student_id THEN
```

```
            return students[i]
        ENDIF
    ENDLOOP
ENDFUNCTION
```

display/student.py

```
SET id to the id provided as a parameter when running the program
from the CLI
CALL student.get(id) returning student_data

FUNCTION edit_profile ()
    print "1. Change password"
    print "2. Change contact number"
    print "3. Exit"
    CASE OF utils.input_number("\nEnter choice: ", 3)
        = 1: PROMPT user to New Password:
             READ new_password
             PROMPT user to Confirm Password:
             READ confirm_password
             DOWHILE len(new_password) < 6
                PRINT "red("Enter a longer password (password ⩾
6)""
             ENDDO
             DOWHILE new_password ≠ confirm_password
                PRINT "red("Your does not match")"
                PROMPT user to New Password:
                READ new_password
                PROMPT user to Confirm Password:
                READ confirm_password
             ENDDO
             PRINT "green("Your password has been changed")"
            CALL student.change_password(id, new_password)
            main_display()
        = 2: DOWHILE True
```

```
                try
                    PROMPT user to New number: +60
                    READ new_contact_num
                    DOWHILE len(str(new_contact_num)) ≠ 8
                        PRINT "red("Enter a malaysian number (9
digits)")"
                        PROMPT user to New number: +60
                        READ new_contact_num
                    CALL student.change_contact(id,
str(new_contact_num))
                    break
                Catch ValueError
                    PRINT "red("Please enter numbers not words")"
            ENDDO
            PRINT "green("Your contact number has been changed")"
            CALL main_display()
        = 3: CALL main_display()
    ENDCASE

FUNCTION display_schedule ()
    PRINT "1. Class-Specific Schedule"
    PRINT "2. Show all schedule"
    PRINT "3. exit"

    CASE OF utils.input_number("\nEnter choice: ", 3)
        = 1: CALL student.view_schedule(id)
             CALL main_display()
        = 2: CALL student.view_all_schedule(id)
             CALL main_display()
        = 3: CALL main_display()
    ENDCASE


FUNCTION manage_requests ()
    PRINT "1. Send request"
```

```
   PRINT "2. Show all request status"
   PRINT "3. Delete request"
   PRINT "4. exit"
   CASE OF utils.input_number("\nEnter choice: ", 4)
       =1: SET module = utils.select("Requested module type?",
["Python", "Java", "C"])
           SET level = utils.select("Requested module level?",
["Beginner", "Intermediate", "Advanced"])
           CALL student.request(id, module, level)
           CALL main_display()
       =2: CALL student.show_request_status(id)
           CALL main_display()
       =3: CALL student.show_request_status(id) returning allowed
           IF allowed == 0 THEN
               print "green("You have not requested for any
modules!!")"
           ENDIF
           PROMPT user to "Enter choice: "
           READ selected_index
           SET selected_request_id =
list(student_data["requests"].keys())[selected_index]
           CALL student.delete_requests(id, selected_request_id)
returning pending
           IF pending == False THEN
              PRINT "green("Your request has already been
Accepted you cannot delete in now")"
           ENDIF
           CALL main_display()
       =4: CALL main_display()
   ENDCASE


FUNCTION view_invoices ()
   PRINT "1. Show all invoices"
   PRINT "2. Pay"
```

```
   PRINT "3. Exit"
   CASE OF utils.input_number("\nEnter choice: ", 3)
       =1: CALL student.show_invoice(id)
          SET has_invoice = student.show_invoice(id)
            IF NOT has_invoice THEN
                main_display()
            PRINT "1. Pay"
            PRINT "2. EXIT"
            CASE OF utils.input_number("\nEnter choice: ", 2)
                =1: CALL student.payment(id)
                    PRINT "green("Payment successful, Thank you")"
                    CALL main_display()
                =2: CALL main_display()
            ENDCASE
       =2: CALL student.payment(id)
           PRINT "green("Payment successful, Thank you")"
           CALL main_display()
       =3: CALL main_display()
   ENDCASE
ENDFUNCTION


FUNCTION main_display()
   PRINT "\nStudent Dashboard:"
   PRINT "1. Edit profile"
   PRINT "2. View schedule"
   PRINT "3. Manage enrollment requests"
   PRINT "4. View invoices"
   PRINT "5. Exit"

   CASE OF utils.input_number("\nEnter choice: ", 3)
       =1: edit_profile()
       =2: display_schedule()
       =3: manage_requests()
       =4: view_invoices()
```

```
        =5: exit()
    ENDCASE
ENDFUNCTION
```

utils.py

```
FUNCTION green(text)
    return f"\033[92m{text}\033[0m"
ENDFUNCTION


FUNCTION red(text)
    return f"\033[0;31m{text}\033[0m"
ENDFUNCTION
```

Emmanuel Naranendhya Tyatan

api/trainer.py

```
LOAD trainers data from database/trainers.json file
READ trainers data

try
  LOAD modules data from database/modules.json file
Catch File not found error
  PRINT “Error: the ‘database/modules.json’ file does not exist”
Endtry

FUNCTION get(trainer_id)
  LOOP trainer in trainers
     IF trainer[“id] == trainer_id THEN
         RETURN
     ENDIF
  ENDLOOP
ENDFUNCTION

FUNCTION update_own_profile(id)
   LOOP trainer in trainers
       IF trainer[“id] == id THEN
           PROMPT user to input new name
           READ new_name
           PROMPT user to input new password
           READ new_password
           SET trainer[“name”] == new_name
           SET trainer[“password] == new_password
           SAVE trainers to “database/trainers.json” file
           PRINT “Trainer profile updated”
           PRINT new name and password
       ENDIF
   ENDLOOP
ENDFUNCTION
```

```
FUNCTION change_module
  LOOP trainer in trainers
     IF id == trainer[“id]
        SET module_ids = [module_id for module_id in
        trainer[“modules”]
        PRINT “Which module name to change?”

        LOOP i, FROM 0 TO LENGTH(trainer[“modules”]) -1 STEP 1
           PRINT “{i + 1}. {CALL get(module_id_["name"] from
           module API file}"
        ENDLOOP

        PROMPT user to enter the choice in module_num
        READ module_num
        SET selected_module_id = module_ids[module_num]
        PROMPT user to enter what module name they want to change
        to
        READ new_name
        CALL module.change_module_name(selected_module_id,
        new_name)
        PRINT ”Module name changed to {new_name}”
     ENDIF
  ENDLOOP
ENDFUNCTION

FUNCTION send_feedback(id)
  LOAD data into "database/feedback.json"
  READ data
  SET feedback_type = utils.select(“Choose the feedback type: “,
  [“suggestion”, “complaint”]
  PROMPT user to input a message
  READ feedback_text
  GENERATE feedback_id by using uuid4
  SET feedback TO {
```

```
                “sender”: trainer_id
                “type”: feedback_type
                “message”: feedback_text
                }
  APPEND feedback to feedbacks
  Try
    SAVE feedbacks data to "database/feedback.json" file
  Catch file not found error
    PRINT (“Error: the file ‘database/feedback.json’ does not
    exist.”)

  PRINT “Feedback sent successfully. Thank you!”
ENDFUNCTION

FUNCTION add_schedule(id)
  LOOP trainer in trainers
     IF trainer[“id”] == id THEN
       SET modules_ids = [module_id for module_id in
                         trainer[“modules”]
       LOOP i, FROM 0 TO LENGTH(trainer[“modules”]) -1 STEP 1
         PRINT ”{i+1}. {modules[module_id][‘name’]}”
         PROMPT user to enter his/her choice
         READ module_num
         SET module_id = module_ids[module_num]
         DOWHILE True
            PROMPT user to input the date for the class schedule
            READ date
            Try
               SET datetime.strptime(date, format:%d/%m/%Y)
            Catch Value Error
               PRINT (“Invalid date format. Please use the
               correct date format (dd/mm/yyyy”)
         ENDDO
         DOWHILE True
            PROMPT user to input the time for the class schedule
```

```
            READ time
            Try
               SET datetime.strptime(time, format: %H:%M)
            Catch Value Error
               PRINT(Invalid time format. Please use the
               correct time format (hours:minutes”)
         ENDDO
         PROMPT user to input the place for the class schedule
         READ place
         APPEND date, time, and place to modules[module_id]
         [“schedule”]
         PRINT (“Coaching Class Added”)
         PRINT date, time, place
         SAVE modules to “database/modules.json”
       ENDLOOP
     ENDIF
  ENDLOOP
ENDFUNCTION


FUNCTION delete_schedule(id)
  for trainer in trainers
    IF trainer[“id”] == id
       SET module_ids = [module_id for module_id in
       trainer["modules"]]
       LOOP i FROM 0 TO LENGTH(trainer["modules"]) - 1 STEP 1
          PRINT (f”{i + 1}. +
          modules[trainer["modules"][i]]['name']”)
       ENDLOOP
       PROMPT "Enter your choice: "
       READ module_num, DECREMENT by 1 for index
       SET module_id = module_ids[module_num]

       SET padding = 2
       CALCULATE padding and max lengths for name, TP
```

```
        number, and email for display formatting
        SET header = f"|{'Date'.center(max_date_length)}|{'Time'.
        center(max_time_length)}|{'Location'.center(max_location_
        length)}|"
        SET separator = '+' + '-' * (max_date_length) + '+' + '-'
        * (max_time_length) + '+' + '-' * (max_location_length)
        + '+'
        PRINT separator
        PRINT header
        PRINT separator
        LOOP i FROM 0 TO LENGTH(modules[module_id[“schedule”]) - 1
        STEP 1
          SET date = schedule['date'].center(max_date_length)
          SET time = schedule['time'].center(max_time_length)
          SET location =schedule['location'].center(max_location_
          length)
          PRINT date, time, and location
          PRINT separator
        ENDLOOP
        PROMPT “Enter your choice: “
        READ schedule_num, DECREMENT by 1 for index
        DELETE modules[module_id]["schedule"][schedule_num]
        PRINT “Schedule deleted!”

        SAVE to “database/modules.json
    ENDIF
ENDFUNCTION

FUNCTION list_enroll_students(id)
  LOAD students data from "database/students.json"
  READ data
  LOOP trainer in trainers
    IF trainer[“id”] == id THEN
      module_ids = [module_id for module_id in
      trainer[“module”]]
```

```
      LOOP i, module_id in enumerate(trainer[“modules”])
         PRINT (f”{i+1}. {modules[module_id][‘name]}”)
      ENDLOOP

      PROMPT "Enter your choice: "
      READ module_num, DECREMENT by 1 for index
      SET module_id = module_ids[module_num]

        LOOP i, student in enumerate (students)
          IF len(modules[module_id][“students” ]) = 0
            PRINT "No student enrolled."
          ENDIF
          LOOP student_in_md in modules[module_id]["students"]
            IF student [“id”] == student_in_md[“id”] THEN
              SET padding = 2
              CALCULATE padding and max lengths for name, TP
              number, and email for display formatting
              SET header = f”|{'Name'.center(max_name_length)}|
              {'TPNumber'.center(max_tp_number_length)}|{'Email’.
              center(max_email_length)}|"
              SET separator = '+' + '-' * (max_name_length) + '+'
+ '-' * (max_tp_number_length) + '+' + '-' * (max_email_length) +
'+'

              PRINT separator
              PRINT header
              PRINT separator
              SET name_cell = f"{student['name'].
              center(max_name_length)}"
              SET tp_number_cell = f"{student['tp_number'].
              center(max_tp_number_length)}"
              SET email_cell = f"{student['email'].center(max_
              email_length)}"
              PRINT name_cell, tp_number_cell, and email_cell
              with formatted table
```

```
              PRINT separator

              WAIT 0.5 seconds
            ENDIF
          ENDLOOP
        ENDLOOP
    ENDIF
  ENDLOOP
END FUNCTION

FUNCTION view_invoice(id)
  LOAD students into JSON from "database/students.json" file
  READ students
  LOOP trainer in trainers
    IF trainer[“id”] == id THEN
       SET module_ids = [module_id for module_id in
                         trainer[“modules”]
       LOOP i, module_id in enumerate(trainer[“modules”]):
            PRINT (f”{i+1}. {modules[module_id][‘name]}”)
       ENDLOOP

       SET module_num from user and decrement by 1 for zero-based
       index
       PROMPT user to enter their choice in module_num
       SET module_id = module_ids[module_num]

       LOOP module in list(modules.values())
          IF trainer[“id”] == module[“trainer”]
             SET padding = 2
             SET student_name_length = 0
             SET outstanding_length = 0
             SET max_module_name_length = LENGTH of module's name
             + padding * 6

          IF modules[module_id]["students"] == 0 THEN
```

```
   PRINT "There is no student enrolled."
      RETURN
ENDIF

LOOP student in students
  LOOP student_in_md in modules[module_id]["students"]
    IF student [“id”] == student_in_md[“id”] THEN
      UPDATE max_student_name_length to max LENGTH of
      student's name + padding * 6
      UPDATE max_outstanding_length to max LENGTH of
      student_in_md's invoice + padding * 6
    ENDIF
  ENDLOOP
ENDLOOP

SET header with 'Module Name', 'Student Name', and
'Outstanding' centered with calculated lengths
SET separator line based on max lengths

PRINT separator
PRINT header
PRINT separator

INITIALIZE student_name = ""
INITIALIZE outstanding = ""
SET module_name = module[“name”].center
(max_module_name_length)

LOOP student in students
  LOOP student_in_md in modules[module_id]["students"]
    IF student[“id] == student_in_md [“id”] THEN
      SET student_name = student[name].center
      (max_student_name_length)
    ENDIF
  ENDLOOP
```

```
          ENDLOOP
          SET outstanding = student_in_md[“invoice”].center
          (max_outstanding_length)
          PRINT formatted row with module_name, student_name, and
          outstanding
          PRINT separator
       ENDLOOP
    ENDIF
  ENDLOOP
ENDFUNCTION
```

display/trainer.py

```
START
  SET id to the first line command
  GET trainer_info using id

  FUNCTION main()
    DOWHILE True
       PRINT "——————Trainer Menu———————"
       PRINT "1. Add Schedule"
       PRINT "2. Delete Schedule"
       PRINT "3. List of Enrolled Students"
       PRINT "4. Students Invoice"
       PRINT "5. Send Feedback"
       PRINT "6. Update Own Profile"
       PRINT "7. Exit"
       PROMPT "Enter your choice: "
       READ choice

       IF choice IS "1"
           CALL add_schedule with id
       ELSE IF choice IS "2"
           CALL delete_class_schedule_part with id
       ELSE IF choice IS "3"
```

```
            CALL list_enroll_student with id
        ELSE IF choice IS "4"
            CALL view_invoice with id
        ELSE IF choice IS "5"
            CALL send_feedback with id
        ELSE IF choice IS "6"
            CALL update_own_profile with id
        ELSE IF choice IS "7"
            PRINT "Thank you."
            BREAK
        ELSE
            PRINT "Invalid choice. Please enter a number between 1
            to 7."
        ENDIF
    ENDDO
  END FUNCTION

  CALL main
END
```

Faiyaad Mahabub

api/lecturer.py

```
Open and load data into “database\lectures.json” as lecturers

Function  get (lecturer_id):
  Loop lecturer in lecturers

     If lecturer[“id”] matches lecturer_id Then

        Return lecturer

     EndIf

  EndLoop

EndFunction

Function change_name (lecturer_id, new_name):
   Loop lecturer in lecturers:

       If lecturer[“id”] matches lecturer_id Then

          lecturer[“name”] = new_name

       EndIf

   EndLoop

   Open “database/lecturers.json” in write mode as write_file
     Write JSON content of lecturers as write_file

EndFunction
```

```
Function change_password (lecturer_id, new_password):
  Loop lecturer in lecturers:

      If lecturer[“id”] matches lecturer_id Then

         lecturer[“password”] = new_password

      EndIF

  EndLoop

  Open “database/lecturers.json” in write mode as write_file
      Write JSON content of lecturers as write_file
EndFunction

Function register (name,  password):
  Append the lecturers for
       {“id”: str(uuid4()), “name”: name, “password”: password}


  Open “database/lecturers.json” in write mode as write_file
     Write JSON contents of lecturers as write_file

EndFunction
```

display/lecturer.py

```
SET lecturer_id to command line argument at position 1
Get data for lecturer using lecturer_id

Function display_edit_profile():
  Print (“1. Edit name”)
  Print (“2. Edit password”)
  Print (“3. Back”)

  Case of Prompt user for choice  in integer
           Read user’s choice

            = Prompt user for new_name
              Read new_name
              change name in lecturer for “lecturer_id” in “new_name”
              Call display_edit_profile()

            = new_password: Prompt user for new password
              Change password in lecturer by lecturer_id in
              new_password
              Call display_edit_profile

            = Call display_dashboard
   Otherwise
     Display “Invalid input number”

   EndCase

EndFunction
```

```
Function display_view_requests():
  students = Get all information from student data in student.json.py
  Create an empty list for students_under_lecturer
  Create an empty list for requests

  Loop student_data in students
   If lecturer_id matches the id for lecturer in student_data Then
         Append student_data in students_under_lecturer

      If length of student_data in “requests” is greater then 0 Then

          Loop request_data in student_data “request” for all values
            Append requests for
            “name”: student_data[“name”],
            “module_request_type”: request_data[“module_request_type”],
            “module_request_level”: request_data[“module_request_level”]

          EndLoop

       EndIF

   EndIf

  EndLoop

  If length requests is greater then 0 Then

   Loop index for request_data in requests

     Display request data’s
             “name”,”module_request_type”,”module_request_level”

   EndLoop

  Else
       Display “No enrollment requests found”

  EndIf
```

```
  Display “Press enter to continue…”
  Prompt user for the input
  Call display_dashboard()
EndFunction


Function display_manage_students():
  Print “1. Register student”
  Print “2. Delete student ”
  Print “3. Approve enrollment requests”
  Print “4. Update student module”
  Print “5. Back”

  Function register_students()
    Prompt user for name
    Read name

    Prompt user for tp_number
    Read tp_number

    Prompt user for contact_number
    Read contact_number

    Prompt user for address
    Read address

    Select module_type among items [“python”,”java”, “c”]
    Select module_level among items [“beginner”, “intermediate”,
                                     “advanced”]
    Select enrollment_month among items [
                                            “January”
                                            “February”
                                            “March”
                                            “April”
                                            “May”
                                            “June”
                                            “July”
                                            “August”
```

```
                                                “September”
                                                “October”
                                                “November”
                                                “December    ]
modules = get module_type and module_level from module into dic
          get_filtered_modules
If length of module is 0 Then
    Display in red
                “Error:a module for {module_type}” with level of
                 {module_level} does not exist. This is possibly
                 because trainer for this module does not exist
    Call display_manage_students()
    Return
 EndIf

 student_id = student.register(
                name,tp_number,contact_number,address,lecturer_id
               )
 selected_module_id =
             list(modules.key())[randint(0,len(modules.keys()-1))]

 student.add_modules(student_id,selected_module_id,enrolment_month)

 Display in green “Successfully registered student with
                  id{student_id}”

 Call display_manage_students()

 EndFunction


 Function delete_profile():
   students = get all data from student
   Loop as index for student_data in students
      Display “{with index+1 for
                student_data[“name”]}{student_data[tp_number]}”
   EndLoop
   Prompt user for a number to select student for deletion
   Read student_index
```

```
  selected_student_id = students[student_index][“id”]
  Delete selected student id from data student

  Remove selected student id for all modules from module
  Display in green “Successfully deleted student with
                   id{selected_student_id} ”
  Call display_manage_students()

EndFunction

Function approve_requests():
  students= Get all data from student
  Create an empty list for student_under_lecturer
  Create an empty list for request_ids

  Loop as index for students in student_data
    If student_data[“lecturer”] matches lecturer_id Then
       Append student_data in students_under_lecturer
       Display “{i+1}. {student_data[‘name’]}”
    EndIf
  EndLoop

  Prompt user for a number to select student for approval
  Read choice
  If request with status other then “approved” in
     student_under_lecturer[choice][“requests”] Then
        modules_datas= create empty list
        Loop as index for request_details in module_datas
          Append request_details at start in request_ids
          request_data=request_details with indentation of 1
          Display “{request_data[‘module_request_type’]}”
        EndLoop
        Prompt user for module choice
        Read module_choice
        enrollment_month =Select enrollment month from [
                                                    “January”
                                                    “February”
                                                    “March”
```

```
                                                    “April”
                                                    “May”
                                                    “June”
                                                    “July”
                                                    “August”
                                                    “September”
                                                    “October ”
                                                    “November”
                                                    “December”
                                                              ]



        module_type=
             module_datas[module_choice][1][“module_request_type”]
        available_modules = from modules get_filtered_modules for
                      module_type,module_level
         If length of available_modules is 0 Then
            Display “No {module_type} module with level
                    {module_level} is available!”
             Call display_manage_students()
             Return
          EndIf

      Approve request in student data at students_under_lecturer
      for choice “id”, module_datas takes in module_choice at
      start

      add module in student for (student_under_lecturer in choice
      for “id”

      Set selected_module_id to a random key from modules
      dictionary)
  Else
   Display “no module request found”

  EndIf
  Call display_manage_students
```

```
EndFunction


Function update_student_module():
 students= get all information for students
 Create empty list for students_under_lecturer

  Loop as index i for student_data in students
    If student_data of lecturer “id” matches with lecturer_id Then
      Append student_data in student_under_lecturer
      Display “{i+1}. {module_id}{module_data[“name”]}”
    EndIf
  EndLoop

  Prompt user to enter choice for modules
  Read choice

  Loop as index for module_id
    Students_under_lecturer for choice in “modules”
    module_data = get module_id in module
    Display “{module_id} {module_data[“name”]}”
  EndLoop

  Prompt user for module_choice
  Read module_choice
  module_type= select the module type to change from list
              [“python”, “java”, “c”]
  module_level= select module level to change from list
               [“beginner”, “intermediate”, “advanced”]
  enrollment_month= select month of enrollment from list
                   [“Jan”,”Nov”, “Dec”]
  modules = get filtered_modules from modules for module_type,
           module_level
  Remove module in student for students under lecturer “id”,
  students_under_lecturer[choice][“modules”][module_choice]

  add modules for students that are under lecturer “id”
    Set selected_module_id to a random key from modules dictionary
```

```
        for enrollment_month
      Call display_manage_students()


    EndFunction

    Case of Prompt user to choose a number between 5
            Read chosen number
       = Call register_student()
       = Call delete_profile()
       = Call approve_request()
       = Call update_student_module()
       = Call display_dashboard()
    Otherwise
      “Not valid input number”
    EndCase
EndFunction


Function display_dashboard()
  Print “Lecturer dashboard”
  Print “Manage students”
  Print “View all enrollment requests”
  Print “Exit”
  Case of prompt user for a number for choosing
          Read user’s choice
          =Call display_edit_profile()
          =Call display_manage_students()
          =Call display_view_requests()
          =exit
  Otherwise
   Display “Invalid input number”
  EndCase
EndFunction

Call display_dashboard()
```

Yeremia Samuel Aditya Marfianto

api/administrator.py

```
Load data from admin.json file
READ data

FUNCTION register(name, password)
  new_admin = {
    "id": generate_unique_uuid(),
    "name": name,
    "password": password
  }

  Append new_admin to data

  Save data to admin.json file
ENDFUNCTION

FUNCTION change_name(administrator_id, new_name)
  for admin in data:
    if admin["id"] == administrator_id:
        admin["name"] = new_name

  Save data to admin.json file
ENDFUNCTION

FUNCTION change_password(administrator_id, new_password)
  for admin in data:
    if admin["id"] == administrator_id:
        admin["password"] = new_password

  Save data to admin.json file
ENDFUNCTION

FUNCTION get(administrator_id)
```

```
  for admin in data:
    if admin["id"] == administrator_id:
      return admin
ENDFUNCTION
```

display/administrator.py

```
SET id to the id provided as a parameter when running the program
from the CLI

FUNCTION display_dashboard()
  PRINT "\nAdministrator Dashboard"
  PRINT "1. Edit profile"
  PRINT "2. Manage trainer"
  PRINT "3. View feedbacks"
  PRINT "4. Exit"

  PROMPT user to enter a number between 1 and 4
  READ choice

  CASE OF choice
    = 1: display_edit_profile()
    = 2: display_manage_trainers()
    = 3: display_view_feedbacks()
    = 4: exit()
  ENDCASE
ENDFUNCTION

FUNCTION display_edit_profile()
  PRINT "1. Change name"
  PRINT "2. Change password"
  PRINT "3. Back"

  PROMPT user to enter a number between 1 and 3
  READ choice
```

```
  CASE OF choice
    = 1: PROMPT user to input a name to change to
         READ changed_name
         CHANGE the name of the administrator in the json file
         PRINT "Name successfully changed to {changed_name}"
         display_edit_profile()
    = 2: PROMPT user to input a password to change to
         READ changed_password
         CHANGE the password of the administrator in the json
file
         PRINT "Password successfully changed to
{changed_password}"
         display_edit_profile()
    = 3: display_dashboard()
  ENDCASE
ENDFUNCTION

FUNCTION display_manage_trainers()
  PRINT "1. Register trainer"
  PRINT "2. Delete trainer"
  PRINT "3. Manage trainers' modules"
  PRINT "4. Check trainer salaries"
  PRINT "5. Back"

  PROMPT user to input a number between 1 and 5
  READ choice

  CASE OF choice
    = 1: display_register_trainer()
    = 2: display_delete_trainer()
    = 3: display_manage_trainer_modules()
    = 4: display_trainer_salaries()
    = 5: display_dashboard()
  ENDCASE
```

```
ENDFUNCTION

FUNCTION display_register_trainer()
  PROMPT user to input name for trainer
  READ name

  PROMPT user to select one of these languages: python, java, c
  READ module_language

  PROMPT user to select one of these levels: beginner,
intermediate, advanced
  READ module_level

  trainer_id = CALL register(name) from trainer API file
  module_id = CALL create(
    trainer_id, module_language, module_level,
f"{module_language} {module_level}"
  ) from module API File

  CALL add_module(trainer_id, module_id) from trainer API file

  PRINT "Successfully registered trainer with id {trainer_id}!"

  display_manage_trainers()
ENDFUNCTION

FUNCTION display_delete_trainer()
  trainers = CALL get_all() from trainer API file
  SET trainer_ids = []

  LOOP i FROM 0 TO len(trainers) - 1 STEP 1
    trainer_data = trainers[i]

    Append trainer_data["id"] to trainer_ids
```

```
    trainer_module_ids = trainer_data["modules"]
    SET module_names = []

    LOOP i FROM 0 TO len(trainer_module_ids) - 1 STEP 1
      module_data = CALL get(module_id) from module API file

      Append module_data["name"] to module_names
    ENDLOOP

    module_names = ", ".join(module_names)

    PRINT "{i + 1}. {trainer_data["name"]} {trainer_data["id"]}
{module_names}"
  ENDLOOP

  PRINT "\n1. Delete a trainer"
  PRINT "2. Cancel"

  PROMPT user to input a number between 1 and 2
  READ choice

  CASE OF choice
    = 1: PROMPT user to input a number between 1 and
len(trainer_ids)
         READ selected_id
         selected_id -= 1
    = 2: display_manage_trainers()
         return
  ENDCASE

  is_deleted = CALL delete(trainer_ids[selected_id]) from trainer
API file

  IF (is_deleted) THEN
    PRINT "Successfully deleted trainer with id {id}"
```

```
  ELSE
    PRINT "Trainer with the id {id} does not exist!"
  ENDIF

  display_manage_trainers()
ENDFUNCTION

FUNCTION display_trainer_salaries()
  trainers = CALL get_all() from trainer API file

  LOOP i FROM 0 TO len(trainers) - 1 STEP 1
    SET trainer_data = trainers[i]

    trainer_module_ids = trainer_data["modules"]
    SET module_datas = []

    LOOP i FROM 0 TO len(trainer_module_ids) - 1 STEP 1
      module_data = CALL get(trainer_module_ids[i]) from trainer
API file

      Append module_data to module_datas
    ENDLOOP

    SET salary = 0

    LOOP i FROM 0 TO len(module_datas) - 1 STEP 1
      module_data = module_datas[i]

      LOOP i FROM 0 TO len(module_data["students"]) - 1 STEP 1
        IF (module_data["students"][i]["invoice"] == 0) THEN
          CASE OF module_data["level"]
            = "advanced": salary += 2000
            = "intermediate": salary += 1500
            = "beginner": salary += 1000
          ENDCASE
```

```
        ENDIF
      ENDLOOP
    ENDLOOP

    PRINT "{index + 1}. {trainer_data['name']}
{trainer_data['id']} {salary}"
  ENDLOOP

  PRINT "\nPress enter to continue..."
  PROMPT user to press enter

  display_manage_trainers()
ENDFUNCTION

FUNCTION display_manage_trainer_modules()
  trainers = CALL get_all() from trainer API file

  SET trainer_ids = []

  FUNCTION get_selected_trainer_id()
    PROMPT user to input a number between 1 and len(trainer_ids)
    READ selected_id
    selected_id -= 1

    return trainer_ids[selected_id]
  ENDFUNCTION

  FUNCTION add_module()
    selected_trainer_id = get_selected_trainer_id()

    LOOP i FROM 0 TO len(CALL get(selected_trainer_id)["modules"]
from trainer API file) - 1 STEP 1
      module_id = CALL get(selected_trainer_id)["modules"][i]
from trainer API file
```

```
      module_data = CALL get(module_id) from module API file

      PRINT "{i + 1}. {module_data["name"]}
({module_data["level"]})"
    ENDLOOP

    PRINT "1. Add a module"
    PRINT "2. Cancel"

    PROMPT user to input a number between 1 and 2
    READ choice

    IF choice == 2 THEN
      display_manage_trainer_modules()
      return
    ENDIF

    PROMPT user to select one of these languages: python, java, c
    READ module_language

    PROMPT user to select one of these levels: beginner,
intermediate, advanced
    READ module_level

    module_id = CALL create(selected_trainer_id, module_language,
module_level, "{module_language} {module_level}") from module API
file

    CALL add_module(selected_trainer_id, module_id) from trainer
API file

    PRINT "Successfully added module for trainer!"
    display_manage_trainer_modules()
  ENDFUNCTION
```

```
  FUNCTION remove_module()
    selected_trainer_id = get_selected_trainer_id()

    LOOP i FROM 0 TO len(CALL get(selected_trainer_id)["modules"]
from trainer API file) - 1 STEP 1
      module_id = CALL get(selected_trainer_id)["modules"][i]
from trainer API file

      module_data = CALL get(module_id) from module API file

      PRINT "{i + 1}. {module_data["name"]}"
    ENDLOOP

    PRINT "1. Remove a module"
    PRINT "2. Cancel"

    PROMPT user to input a number between 1 and 2
    READ choice

    IF choice == 2 THEN
      display_manage_trainer_modules()
      return
    ENDIF

    PROMPT user to input a number between 1 and len(module_ids)
    READ selected_module_num
    selected_module_num -= 1

    selected_module_id = module_ids[selected_module_num]

    CALL remove_module(selected_trainer_id, selected_module_id)
from trainer API file
    CALL delete(selected_module_id) from module API file

    PRINT "Successfully deleted module from trainer!"
```

```
    display_manage_trainer_modules()
  ENDFUNCTION

  LOOP i FROM 0 TO len(trainers) - 1 STEP 1
    trainer_data = trainers[i]

    Append trainer_data["id"] to trainer_ids

    trainer_module_ids = trainer_data["modules"]

    SET modules_names = []

    LOOP i FROM 0 TO len(trainer_module_ids) - 1 STEP 1
      CALL get(trainer_module_ids[i]) from module API file
    ENDLOOP

    module_names = ", ".join(module_names)

    PRINT "{i + 1}. {trainer_data["name"]} {trainer_data["id"]}
{module_names}"
  ENDLOOP

  PRINT "\n1. Add module"
  PRINT "2. Remove module"
  PRINT "3. Cancel"

  PROMPT user to input a number between 1 and 3
  READ menu_choice

  CASE OF menu_choice
    = 1: add_module()
    = 2: remove_module()
    = 3: display_manage_trainers()
  ENDCASE
ENDFUNCTION
```

```
FUNCTION display_view_feedbacks(filter_type = "")
  feedbacks = CALL get_all() from feedback API file
  SET items_count = 0

  LOOP i FROM 0 TO len(feedbacks) STEP 1
    SET current_feedback = feedbacks[i]

    IF filter_type ≠ "" and current_feedback["type"] ≠
filter_type:
      continue
    ENDIF

    items_count += 1

    feedback_sender, feedback_type, feedback_message = values of
current_feedback
    feedback_sender = CALL get(feedback_sender)["name"] from
trainer API file

    IF items_count > 0 THEN
      PRINT "{feedback_type.capitalize()}
{feedback_sender.capitalize()} {feedback_message}"
    ELSE
      PRINT "No feedback with type {filter_type} found."

      PROMPT user to press enter

      display_view_feedbacks()
    ENDIF

    IF filter_type == "" THEN
      PRINT "\n1. Filter type to suggestion"
      PRINT "\n2. Filter type to complaint"
      PRINT "\n3. Back""
```

```
      PROMPT user to input a number between 1 and 3
      READ choice

      CASE OF choice
        = 1: display_view_feedbacks("suggestion")
        = 2: display_view_feedbacks("complaint")
        = 3: display_dashboard()
      ENDCASE
    ELSE
      PRINT "\n1. Back"

      PROMPT user to input the number 1
      READ choice

      IF choice == 1 THEN
        display_view_feedbacks()
      ENDIF
    ENDIF
  ENDLOOP
ENDFUNCTION

START
  display_dashboard()
END
```

api/trainer.py

```
Load data from trainers.json file
READ trainers

FUNCTION register(name)
  trainer_id = generate_random_uuid()

  new_trainer = {
```

```
    "id": trainer_id,
    "name": name,
    "password": "123",
    "modules": [],
    "feedbacks": []
  }

  Append new_trainer to trainers

  Save trainers to trainers.json file

  return trainer_id
ENDFUNCTION

FUNCTION add_module(trainer_id, module_id)
  LOOP i FROM 0 TO len(trainers) - 1 STEP 1
    IF trainer["id"] == trainer_id THEN
      Append module_id to trainer["modules"]
    ENDIF
  ENDLOOP

  Save trainers to trainers.json file
ENDFUNCTION

FUNCTION get_all()
  return trainers
ENDFUNCTION

FUNCTION delete(id)
  SET deleted_status = False

  LOOP i FROM 0 TO len(trainers) - 1 STEP 1
    user = trainers[i]

    IF user["id"] == id THEN
```

```
      LOOP i FROM 0 TO len(user[modules]) STEP 1
        module_uuid = user[modules][i]

        CALL delete(module_uuid) from module API file
      ENDLOOP

      trainer[i].remove()

      Save trainers to trainers.json file

      deleted_status = True
    ENDIF
  ENDLOOP

  return deleted_status
ENDFUNCTION

FUNCTION get(trainer_id)
  LOOP i FROM 0 TO len(trainers) - 1 STEP 1
    trainer = trainers[i]

    IF trainer["id"] == trainer_id THEN
      return trainer
    ENDIF
  ENDLOOP
ENDFUNCTION

FUNCTION remove_module(trainer_id, module_id)
  LOOP i FROM 0 TO len(trainers) - 1 STEP 1
    trainer = trainers[i]

    IF trainer["id"] == trainer_id THEN
      trainer["modules"].remove(module_id)
    ENDIF
  ENDLOOP
```

```
  Save trainers to trainers.json file
ENDFUNCTION
```

api/module.py

```
Load data from modules.json file
READ data

FUNCTION get(module_id)
  return data[module_id]
ENDFUNCTION

FUNCTION change_module_name(module_id, new_name)
  data[module_id][“name”] = new_name

  Save data to database/modules.json file
ENDFUNCTION

FUNCTION create(trainer_id, language, level, module_name)
  uuid = generate_random_uuid()

  data[uuid] = {
    "trainer": trainer_id
    "language": language,
    "level": level,
    "name": module_name,
    "schedule": [],
    "students": []
  }

  Save data to modules.json file

  return uuid
ENDFUNCTION
```

```
FUNCTION add_student(module_id, student_id, enrollment_month)
  SET invoice = 0

  CASE OF get(module_id)["level"]
    = "advanced": invoice = 2000
    = "intermediate": invoice = 1500
    = "beginner": invoice = 1000
  ENDCASE

  Append {"id": student_id, "enrolment_month": enrollment_month,
"invoice": invoice} to data[module_id]["students"]

  Save data to modules.json file
ENDFUNCTION

FUNCTION remove_student_from_all_modules(student_id)
  LOOP i FROM 0 TO len(data.items()) - 1 STEP 1
    module_id = list(data.items())[i][0]
    module_data = list(data.items())[i][1]

    LOOP i FROM 0 TO len(module_data["students"]) - 1 STEP 1
      module_student = module_data["students"][i]

      IF (module_student["id"] == student_id) THEN
        data[module_id] ["students"][i].remove()
      ENDIF
    ENDLOOP
  ENDLOOP

  Save data to modules.json file
ENDFUNCTION

FUNCTION get_filtered_modules(language, level)
  modules = {}
```

```
  LOOP i FROM 0 TO len(data.items()) - 1 STEP 1
    module_id = list(data.items())[i][0]
    module_data = list(data.items())[i][1]

    IF module_data["language"] == language and
module_data["level"] == level THEN
      modules[module_id] = module_data
    ENDIF
  ENDLOOP

  return modules
ENDFUNCTION

FUNCTION remove_student_from_module(student_id, module_id)
  LOOP i FROM 0 TO len(data[module_id]["students"]) STEP 1
    student_data = data[module_id]["students"][i]

    IF student_data["id"] == student_id THEN
      student_data.remove()
    ENDIF
  ENDLOOP

  Save data to modules.json file
ENDFUNCTION

FUNCTION delete(module_id)

  LOOP i FROM 0 TO len(data[module_id]["students"]) STEP 1
    student_data = data[module_id]["students"][i]

    CALL remove_module(student_data["id"], module_id) from
student API file
  ENDLOOP
```

```
  Save data to modules.json file
ENDFUNCTION
```

api/feedback.py

```
Load data from feedbacks.json file
READ data

FUNCTION create(sender, type, message)
    feedback_uuid = generate_random_uuid()

    data[feedback_uuid] = {
        "sender": sender,
        "type": type,
        "message": message
    }

    Save data to feedbacks.json file

    return feedback_uuid
ENDFUNCTION

FUNCTION get_all()
    return the values of data
ENDFUNCTION
```

utils.py

```
FUNCTION select(message, items)
  PRINT message

  LOOP i FROM 0 TO len(items) - 1 STEP 1
    PRINT "{i + 1}. {items[i]}"
  ENDLOOP
```

```
  RETURN items[input_number("\nEnter choice: ", len(items)) - 1]
ENDFUNCTION

FUNCTION input_number(message, choice_count)
  DOWHILE (True)
    PRINT message
    PROMPT user to input a number
    READ choice

    IF (1 ⩽ choice ⩽ choice_count) THEN
      return choice
    ELSE
      PRINT "Input a number between 0 and {choice_count}"
      CONTINUE
    ENDIF
  ENDDO
ENDFUNCTION

FUNCTION green(text)
    return "\033[92m{text}\033[0m"
ENDFUNCTION

FUNCTION red(text)
    return "\033[0;31m{text}\033[0m"
ENDFUNCTION

FUNCTION bold(text)
    return "\033[1m{text}"
ENDFUNCTION
```

main.py

```
FUNCTION handle_login()
  SET attempt_count = 1
```

```
  type = role_choice

  PRINT "Logging in as {type.capitalize()}..."

  PROMPT user to input name
  READ name

  DOWHILE (attempt_count ⩽ 4)
    PROMPT user to input password
    READ password

    status = CALL check(type, name, password) from user API file

    IF not status["exists"] THEN
      PRINT "User does not exist!"
      return
    ENDIF

    IF status["exists"] and not status["match"] THEN
      IF 3 - attempt_count ⩾ 0 THEN
        PRINT "Wrong password! {3 - attempt_count} chance(s)
left"
      ENDIF

      attempt_count += 1
      continue
    ENDIF

    IF status["match"] THEN
       Programmatically run the python file of {type} with the
status["id"] as the first argument
    ENDIF
  ENDDO
ENDFUNCTION
```

```
FUNCTION handle_register()
  type = role_choice

  PRINT "Register asa {type}"

  PROMPT user to input name
  READ name

  PROMPT user to input password
  READ password

  IF CALL check(type, name, password)["exists"] from user API
file THEN
    PRINT "User already exists!"
    return
  ENDIF

  id = CALL register(type, name, password) from user API file

  Programmatically run the python file of {type} with the id as
the first argument
ENDFUNCTION

START
  PROMPT user to select which role they want to enter as:
administrator, lecturer, trainer, or student
  READ role_choice

  SET action_choice_count = 1

  PRINT "Action:"
  PRINT "1. Login"
  IF role_choice in ["administrator", "lecturer"] THEN
    PRINT "2. Register"
    action_choice_count = 2
```

```
  ENDIF

  PROMPT user to input a number between 1 and action_choice_count
  READ action_choice

  CASE OF action_choice
    = 1: handle_login()
    = 2: handle_register()
  ENDCASE
END
```

api/user.py

```
FUNCTION load_data(type)
  Load data from database/{type}.json
  READ data

  return data
ENDFUNCTION

FUNCTION check(type, name, password)
  data = load_data(type)

  SET user_exists = False
  SET password_matches = False
  SET id = False

  LOOP i FROM 0 TO len(data) - 1 STEP 1
     user = data[i]

     IF user_exists THEN
       break
     ENDIF

     IF user["name"].lower() == name.lower() THEN
```

```
      user_exists = True

      IF user["password"] == password THEN
        password_matches = True

        id = user["id"]
      ENDIF
    ENDIF
  ENDLOOP

  return {
    "exists": user_exists,
    "match": password_matches,
    "id": id
  }
ENDFUNCTION

FUNCTION register(type, name, password)
  data = load_data(type)

  id = generate_random_uuid()

  Append {
    "id": id,
    "name": name,
    "password": password
  } to data

  Save data to database/{type}s.json file

  return id
ENDFUNCTION
```

Explanation of Programming Concepts

Abdullah Al Faiyaz

File and Error Handling

```
try:
    students = json.load(open("database/students.json"))
    subjects = json.load(open("database/modules.json"))
except FileNotFoundError as e:
    print(red(f"The {e.filename.replace("database/", "")} was not found."))
    exit()
```

These lines of code are used to open and read the json file as an object in python. open("database/students.json") is used to open the json file and json.load() is used to convert the data in json to an object that can be used in python; try: and except FileNotFoundError as e: is used for error handling if the file that is being opened is missing then the (The “name of the missing file” was not found.) statement will be displayed. I am using except FileNotFoundError as e: to display the name of the missing file. e.filename shows the name of the file and I am removing database/ from the file name as all the files containing the data of the students and modules are in the database . By using .replace I am replacing “database/” with a blank space.

Function and For loop

```
def get(student_id):
    for student in students:
        if student["id"] == student_id:
            return student
```

The code in the image is of a function that is used to get the data of the student that has logged. In this code I have used a for loop to iterate over the student list in a JSON file, with each student's data stored in a dictionary. From the dictionary we are taking the id of the students and comparing the id of the student that has logged-in student using an if statement. When a match is found, all the data of that student is returned when the function is called, as I use the return statement.

While loop and User Input

```
while True:
    try:
        new_contact_num = int(input("New number: +60 "))
        while len(str(new_contact_num)) != 8:
            print(red("Enter a malaysian number (9 digits)"))
            new_contact_num = int(input("New number: +60 "))
        student.change_contact(id, str(new_contact_num))
        break
    except ValueError:
        print(red("Please enter numbers not words"))
print(green("Your contact number has been changed"))
main_display()
```

The code in the image is of a while loop the first while loop is in an infinite loop and the second while loop is a conditional so before that we ask the user(student) to input their contact number as the input can be only integers I have implemented error handling with try: and except except ValueError: just in case the user(student) puts in a string by chance. The condition for the second loop is to check if the length of the contact number is 9 digits or not; to check the length of the contact number the integer input is converted to a string, and the len() is used to check the length of the contact number. If the contact number is 9 digits then the while loop will break and display that their contact number has been changed in green colour. If it is not 9 digits it will go to the second loop and display that Enter a malaysian number (9 digits) in red colour. The loop will continue until a 9 digit contact number is given or the user uses a string. If the user uses a string then it will display Please enter numbers not words and continue the first loop.

Match case and Function calling

```
match input_number("Enter choice: ", 3):
    case 1:
        student.view_schedule(id)
        main_display()
    case 2:
        student.view_all_schedule(id)
        main_display()
    case 3:
        main_display()
```

In the code above I am using match case to call functions

Emmanuel Naranendhya Tyatan

File Proccessing and Error Handling:

```
7   trainers = json.load(open("database/trainers.json"))
8   try:
9       modules = json.load(open("database/modules.json"))
10  except FileNotFoundError:
11      print("Error: the 'database/modules.json' does not exist.")
12
```

The code above indicates the way to open JSON file and store it inside that particular JSON file. Using the command of 'json.load', it will load trainers information from the JSON files. In the next line, it also loads modules information with 'try' command to handle any exceptions. If the modules file doesn't exist, it will print an error regarding the file does not exist. This certainly used for handle the issues and prevent the messiness in lines of code.

```
with open("trainers.json", "w") as json_file:
    json.dump(trainers, json_file)
```

For this type of file processing, it would open the JSON file (in this matter, it will open 'trainers.json') in write mode ('w'), which allow the program to write the data into the file.

Function

```
    Emmanuel Nara
13  def get(trainer_id):
14      for trainer in trainers:
15          if trainer["id"] == trainer_id:
16              return trainer
17
```

The code snippet is designed to search a collection of trainer records and find an id that matches "trainer_id" parameter.

Looping

- For Loop

```
55  def add_schedule(id):
56      for trainer in trainers:
57          if id == trainer["id"]:
58              module_ids = [module_id for module_id in trainer["modules"]]
59
60              for i, module_id in enumerate(trainer["modules"]):
61                  print(f"{i+1}. {modules[module_id]['name']}")
62
63              module_num = int(input("Enter your choice: ")) - 1
64              module_id = module_ids[module_num]
```

From the provided code snippet, I use two loops. The first loop uses to iterate through the list of trainer that were stored in trainers. If the provided id that were passed as input to the function matches with trainer's id, it will retrieves the the list of modules that are associated with trainer and stores it in a variable called module_ids. As for the second loop, it is use to iterate through the list of modules and display the modules that were stored in that list. It also use enumerate command, so the modules name are numbered and easy to implemented it in selection.

- While Loop

```
11  def main():
12      while True:
13          print("------Trainer Menu-------")
14          print("1. Add Schedule")
15          print("2. Delete Schedule")
16          print("3. Update Module Name")
17          print("4. List of Enrolled Students")
18          print("5. Students Invoice")
19          print("6. Send Feedback")
20          print("7. Update Own Profile")
21          print("8. Exit")
```

The code snippet above indicates the continuous looping that present a trainer menu containing the functions of add and delete schedule, update module name, list of enrolled students, student invoice menu, send feedback menu, update trainer's profile menu, and exit menu.

Selection

```
34          if id == trainer["id"]:
35              new_name = input("Enter trainer's new name: ")
36              new_password = input("Enter new password: ")
37              trainer["name"] = new_name
38              trainer["password"] = new_password
39
40              with open("database/trainers.json", "w") as json_file:
41                  json.dump(trainers, json_file)
42
43              print("\nTrainer Updated")
44              print(f"Name: {new_name}\nPassword: {new_password}\n")
```

In the code snippet above, the use of 'if' statement indicates if the variable id matches with the key "id" in the trainer dictionary, it asks the user to enter their new name and password. These new details are set to the trainer's record in the list, the entire updated details are saved in JSON file specifically in 'database/trainers.json', and then it prints a confirmation message to the console indicating the trainer has been updated, showing the new name and password.

Print()

```
print(f"{i+1}. {modules[module_id]['name']}")
```

The line of code above uses print function with f-string to print out the current index in the loop incremented by 1 (because it always started from 0) to make the count start from number 1. It also prints the name of the module using key ‘name’ corresponding to the current ‘module_id’ being iterated over. Each module’s name is printed alongside its number in the list, creating a numbered list of modules on the terminal.

Input()

```
feedback_text = input("Enter the message: ")
```

The code above uses the input statement to ask the user to enter their feedback message. The message itself can be in the form of complaint or suggestion, and when the user types their feedback it will stored as a string in the variable ‘feedback_text’.

Conversion (int())

```
schedule_num = int(input("Enter choice: ")) - 1
```

The code snippet above, is to prompts the user to enter their choice number of selected schedules, the ‘int()’ command attempts to convert the user’s input into an integer, This is required because the command will return data as numerical operations so it will use numbers and it is easier for the user to prompt rather using string.

List

```
modules[module_id]["schedule"].append({"Date": date, "Time": time, "location": place})
```

The code above is adding a new entry to the “schedule” list within a module. It started with assigning the modules as the dictionary and calling [module_id] to access a specific module by its id within the modules dictionary. And then I place [“schedule”] as a key to indicate every module in the modules dictionary and append it to the the same list a new schedule information of date, time, and place which is a list to containing new schedule items consist of date, time, and place.

List Comprehension

```
padding = 2
max_date_length = max(
    len(schedule["date"]) for schedule in modules[module_id]["schedule"]
) + (padding * 3)
max_time_length = max(
    len(schedule["time"]) for schedule in modules[module_id]["schedule"]
) + (padding * 3)
max_location_length = max(
    len(schedule["location"]) for schedule in modules[module_id]["schedule"]
) + (padding * 3)
```

The code snippet above is to make a new list of schedule containing the existing information from modules[module_id][“schedule”]. And to calculate the maximum length of the value or information in the existing list. So we could calculate how much space we need to make a table.

Faiyad Mahabub Tasin

Function

```
def get(lecturer_id):
    for lecturer in lecturers:
        if lecturer["id"] == lecturer_id:
            return lecturer
```

This is a function name get which takes the parameter lecturer_id where it iterates through the list of " lecturer in lecturers". 'lecturers' is the variable used to assign the loading and opening of the lecturer's JSON file. In the list in the JSON file, if "id" is found and matched with lecturer_id which is in 'display\lecturer.py' then return the corresponding information to a lecturer. This function is a simple retrieval function where it retrieves the lecturer's information based on their ID from loaded JSON data in the lecturer's JSON file.

Selection

```
def get(lecturer_id):
    for lecturer in lecturers:
        if lecturer["id"] == lecturer_id:
            return lecturer
```

In the function "check" in "api\lecturer.py" with a taken parameter of name, and password. The selection used is if-else statements. First, it iterates for the 'user' in lecturers that is assigned to open and load the lecturer's JSON file. The parameters 'name' and 'password' are

used to check stored data in JSON file. Where “user_exist” is assigned as a Boolean flag to see if the lecturer with ‘name’ exists in the JSON file or not even for “password_matches” as well. If” user_exists” in boolean false in the first ‘if’ block, then break iteration. In the next ‘if’ block it matches the lecturer’s ‘name’ stored in the JSON file with the name. If it’s a match, then turn it into the boolean ‘TRUE’ and move to another ‘If’ block statement where if the ‘password’ matches the password then change the boolean to ‘TRUE’ and break iteration. Where it returns strings “exists” and “match” as either boolean “TRUE” or “FALSE”

Input

```
match int(input("\nEnter choice: ")):
```

The input function was taken from the “display_edit_profile()” function where it uses a matching case to select through the option for editing the lecturer’s name from the “name” list in lecturers.json as an option of “Edit name”. “Edit password “of the lecturer in the list “password” is edited as well after prompting the user choice using the input function and option “Back” to move back to the “display_dashboard()” function to display options for choosing further actions. The input function is used to prompt the user choice for select the above options.

File Processing

```
with open("database/lecturers.json", "w") as write_file:
    json.dump(lecturers, write_file)
```

This snippet is taken from the function “change_name” in “api\lecturer.py” where this file processing is used to open “database/lecturers.json” as “write_file” in write mode where you

can write contents into the JSON database as the function is used to change the lecturer's name and password when "display_edit_profile" function is used in "display\lecturer.py".

Print

```
print("1. Register student")
print("2. Delete student")
print("3. Approve enrolment requests")
print("4. Update student module")
print("5. Back")
```

The print function is used to display the choices for the user for selection. Where the user selects its choice to move into that particular action.

List

```
enrolment_month = select(
    message: "Enrolment month?",
    items: [
        "January",
        "February",
        "March",
        "April",
        "May",
        "June",
        "July",
        "August",
        "September",
        "October",
        "November",
        "December",
    ],
).lower()
```

This is a list of months for the enrollment of students. This is where lecturers choose the month of enrollment for the students by using the select option.

String "len" function

```
if len(students_under_lecturer[choice]["requests"]) > 0:
```

The function "len" is used to calculate and return the length of the list in the dictionary "students_under_lecturer" where it accesses the dictionary "requests" with the chosen lecturer's choice. This "len" function calculates if there is any empty list or has a student in the request list in the student's JSON file. If there is an empty list it won't enter into the loop in the "if" block statement but will enter the "else" statement displaying "No module requests found". If it contains any students the len will calculate it showing more than 0 and enter the "if" block statement where it would move into loop statement "for" to iterate through the list of "requests"

Yeremia Samuel Aditya Marfianto

Looping

```
table = []

for index, module_id in enumerate(trainer.get(selected_trainer_id)["modules"]):
    module_data = module.get(module_id)

    table.append([index + 1, f"{module_data["name"]}"])

print(tabulate(table, ["No.", "Module"], tablefmt="rounded_grid"))
```

This code snippet above uses a for loop to iterate over a trainer's modules. In the for loop statement, A list containing the index, module name, and module level is created. After the loop ends, a table is printed using the previously mentioned data with the tabulate external library.

Selection

```
match input_number("\nEnter choice: ", 3):
    case 1:
        add_module()
    case 2:
        remove_module()
    case 3:
        display_manage_trainers()
        return
```

This code snippet of the selection concept uses a match statement to determine which menu to display next according to user input.

input()

```
choice = int(input(f"{message}\033[92m"))
```

This code uses the input function to ask the user to input a choice in the form of a number. The function first displays the message variable before getting the user input. It then converts the string output into an integer.

List

```
return items[input_number("\nEnter choice: ", len(items)) - 1]
```

This code snippet returns the nth element of the items list according to user input.

print()

```
print(red(f"Error: Input a number between {bold("1")} and {choice_count}!"))
```

This line of code uses the print function and f-string to print an error message when a user inputs a number that is larger than choice_count or lower than 0.

Function

```
def display_dashboard():
    print("Administrator Dashboard")
    print("1. Edit profile")
    print("2. Manage trainer")
    print("3. View feedbacks")
    print("4. Exit")

    match input_number("\nEnter choice: ", 4):
        case 1:
            display_edit_profile()
        case 2:
            display_manage_trainers()
        case 3:
            display_view_feedbacks()
        case 4:
            exit()
```

This function displays the main menu which administrators see after they login.

List comprehension

```
module_names = ", ".join(
    module.get(module_id)["name"] for module_id in trainer_module_ids
)
```

The code above maps the trainer_module_ids list as another list containing the module names. The module name list then gets joined with a comma as a separator.

JSON file loading and writing

```
with open("database/administrators.json") as file:
    data = json.load(file)
```

This code loads the json file from database/administrators.json into the variable “data”.

```
with open("database/administrators.json", "w") as file:
    json.dump(data, file, indent=2)
```

This code writes the contents of variable “data” into database/administrators.json with an indentation of 2 and auto-formatting.

Additional Features

Abdullah AL Faiyaz - Red and Green Output Colors

```
def green(text):
    return f"\033[92m{text}\033[0m"


FaiyazWiz
def red(text):
    return f"\033[0;31m{text}\033[0m"
```

These two functions change the color of the text that you put in the parameters of the function. It is located in the utils.py file.

Yeremia Samuel Aditya Marfianto - Utility functions

```
def select(message, items):
    print(message)

    for index, item in enumerate(items):
        print(f"{index + 1}. {item}")

    return items[input_number("\nEnter choice: ", len(items)) - 1]


def input_number(message, choice_count):
    while True:
        try:
            choice = int(input(f"{message}\033[92m"))
            print("\033[0m")
            if 1 ≤ choice ≤ choice_count:
                return choice
            else:
                print(red(f"Error: Input a number between {bold("1")} and {choice_count}!"))
                continue
        except ValueError:
            print(red("Error: Input a number!"))
            continue
```

This source code is located in the utils.py file, which contains handy functions to make creating menus easier. The select function is created to make selections easier in menus. It prints a message before listing all the selection items which the user can then select by entering the index number of the item. The input_number function handles the logic for when a user needs to input a range of numbers. The function also handles the potential errors that may be caused when trying to input a range of numbers.

Yeremia Samuel Aditya Marfianto - Filtering feedback by type

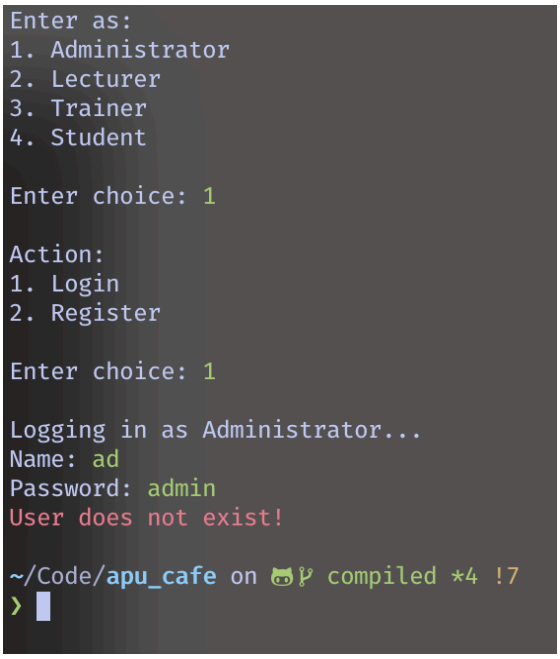
Feedbacks in this system have two types: suggestion and complaint. The functionality to view feedbacks sent by trainers in the administrator menu is extended by adding an option to filter between those two types of suggestions. The source code can be found in display/administrator.py from line 283 to 338.

Sample Input and Output of The Program

Login Menu

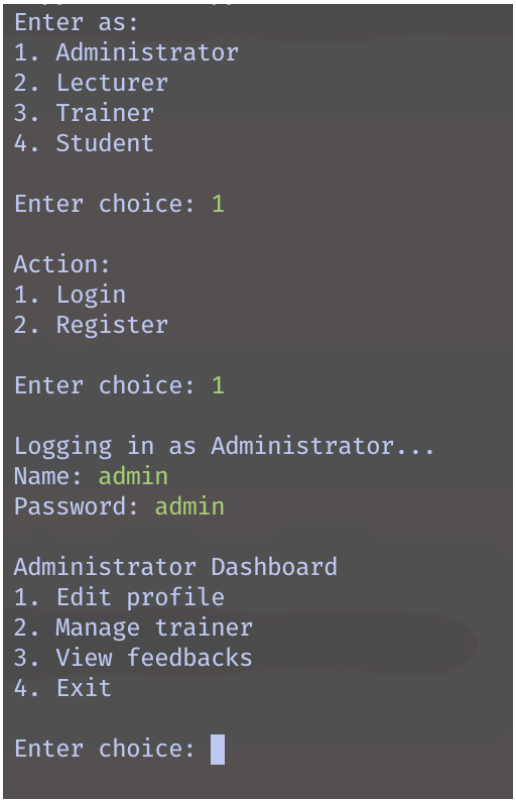
```
Enter as:
1. Administrator
2. Lecturer
3. Trainer
4. Student

Enter choice: 1

Action:
1. Login
2. Register

Enter choice: 1

Logging in as Administrator...
Name: admin
Password: 1
Wrong password! 2 chance(s) left
Password: 123
Wrong password! 1 chance(s) left
Password: 1234
Wrong password! 0 chance(s) left
Password: ad

~/Code/apu_cafe on  compiled *4 !7
❯
```

The above screenshot shows how users are given 4 chances to input the correct password.

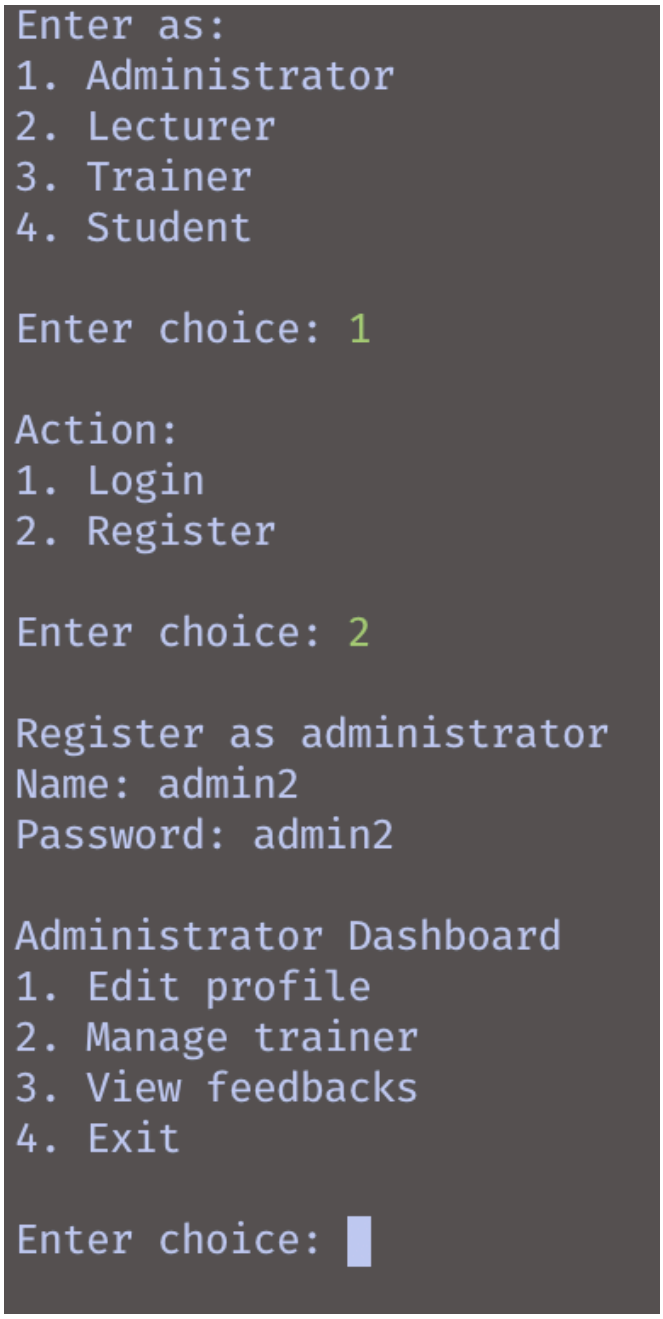


It will output an error if the given username doesn't exist.

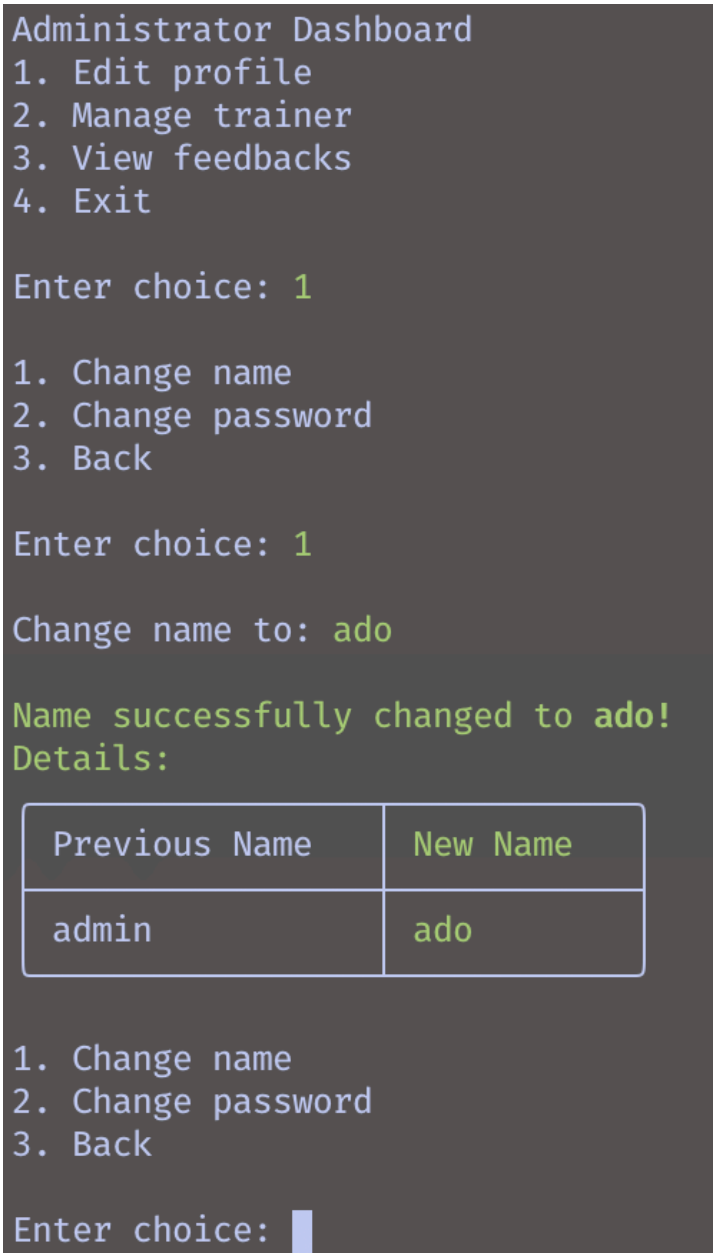
Administrator Menu



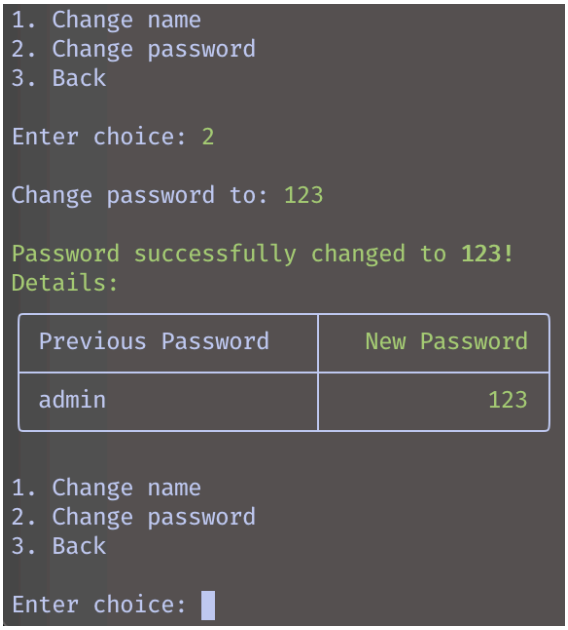
Logging in as an administrator: In this menu, after a user has successfully logged in as an administrator, they will be shown the dashboard menu, which acts as an entry point to all of the sub-menus.



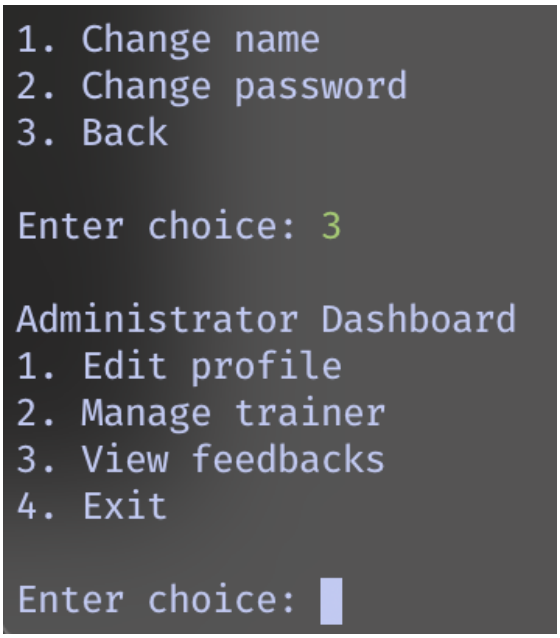
Registering as an administrator: In this menu, the user is prompted to enter a name and a password to complete the registration process.



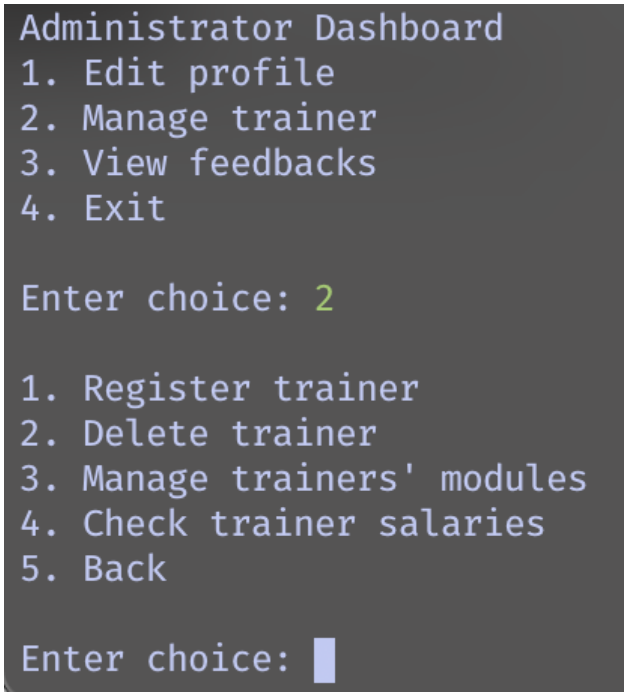
Changing the administrator's own username: In this menu, the user is prompted to enter a new username. A confirmation text appears as soon as the user presses enter and the previous name and new name will be displayed side by side afterwards.



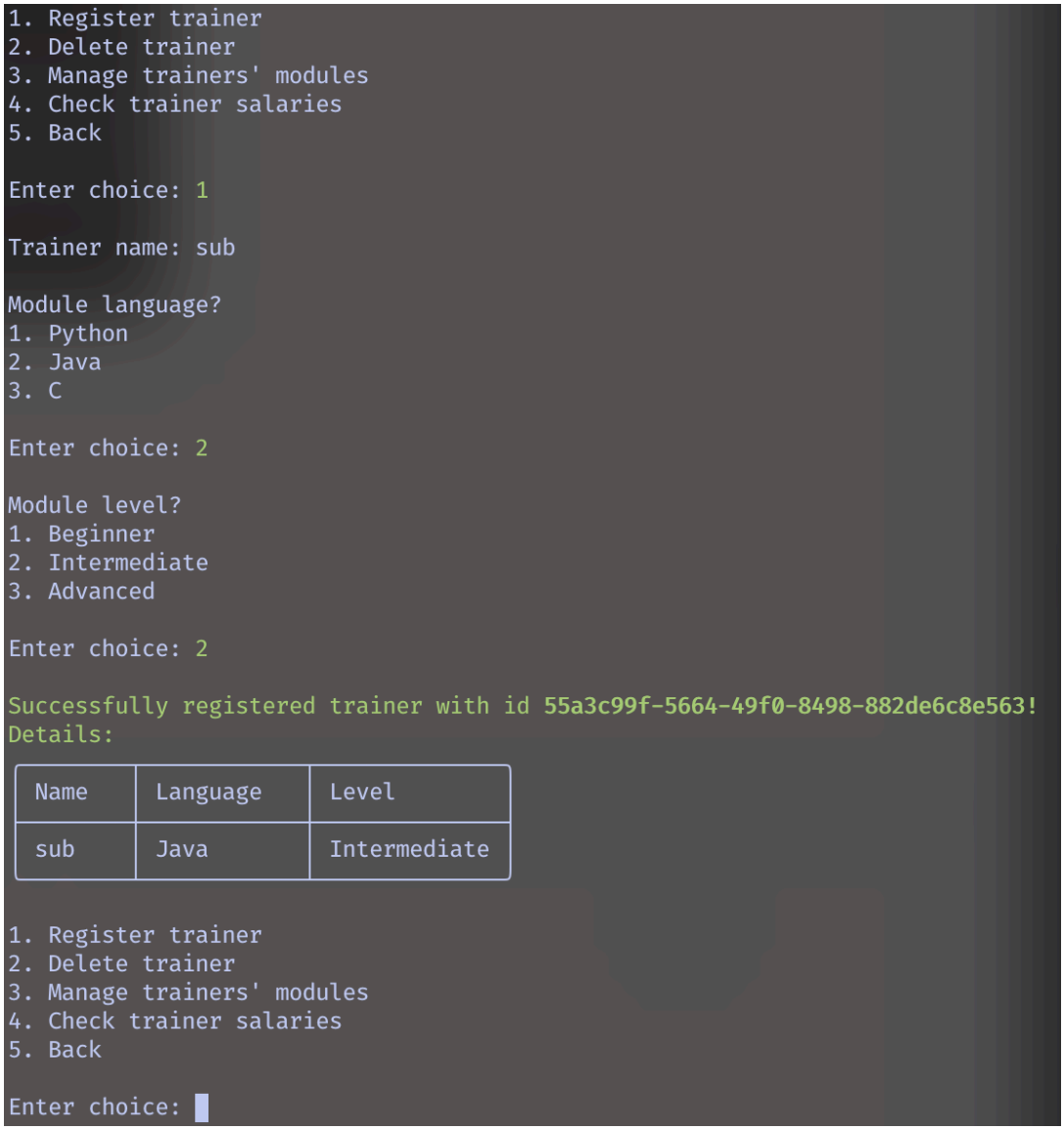
Changing the administrator's own password: This menu has the same action flow as changing the username.



Going back to the admin dashboard from the edit profile menu: Administrators can go back to the main dashboard menu after going into most sub-menus.



Entering the manage trainer menu: In this menu, administrators can do all the things related to managing a trainer such as: registering a trainer, deleting a trainer, managing a trainer's module, and checking trainers' salaries.



Registering a trainer: When registering a trainer, the administrator is prompted to enter the trainer's name, select which module they will be teaching, and the level of that module. After entering all of those details, a successful confirmation text will appear alongside the details of the newly registered trainer displayed in a table format.

```
1. Register trainer
2. Delete trainer
3. Manage trainers' modules
4. Check trainer salaries
5. Back

Enter choice: 2
```

No.	Name	Id	Module name(s)
1	asep	a07d2389-f146-4f85-bba5-dc62300fb331	Advanced Python
2	kon	30f95807-ed01-4127-a7d4-362a6e73cc2a	c advanced
3	bibo	852e8a60-fe7a-428d-8efb-9a93b9c146ce	java advanced
4	jun	8361eb6c-e52d-432d-96ee-3dd4c71bfda4	python intermediate
5	pon	c85cb8df-bb31-4fe0-9e22-c82421b57099	java beginner
6	bob	5595c50b-687e-4013-8a39-b7a24dacb6b1	java intermediate
7	yor	a5f9bc92-4e64-466b-8fbd-da523758af0f	c beginner
8	lon	d4e0f684-0023-4ca0-b8ea-4442e5546664	c intermediate
9	jon	c7ff8d3a-ad7d-40bf-81e4-8ac383054b4d	python beginner
10	sub	55a3c99f-5664-49f0-8498-882de6c8e563	java intermediate

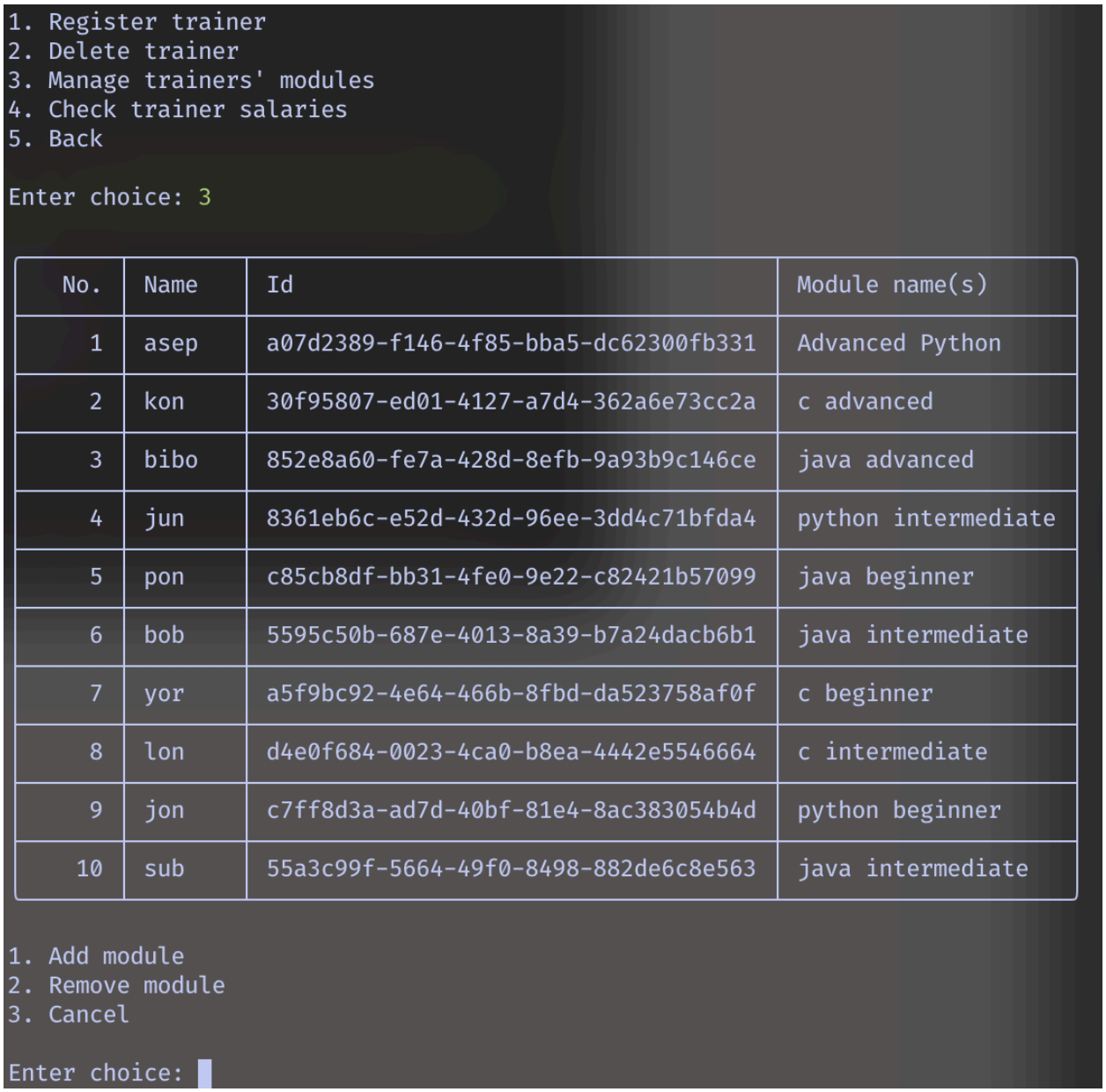
```
1. Delete a trainer
2. Cancel

Enter choice: 1

Which trainer to delete? 10

Successfully deleted trainer with id 71f2743a-bb31-4748-99c4-4fb9805234bc!
Details:
```

Name	Id
sub	55a3c99f-5664-49f0-8498-882de6c8e563

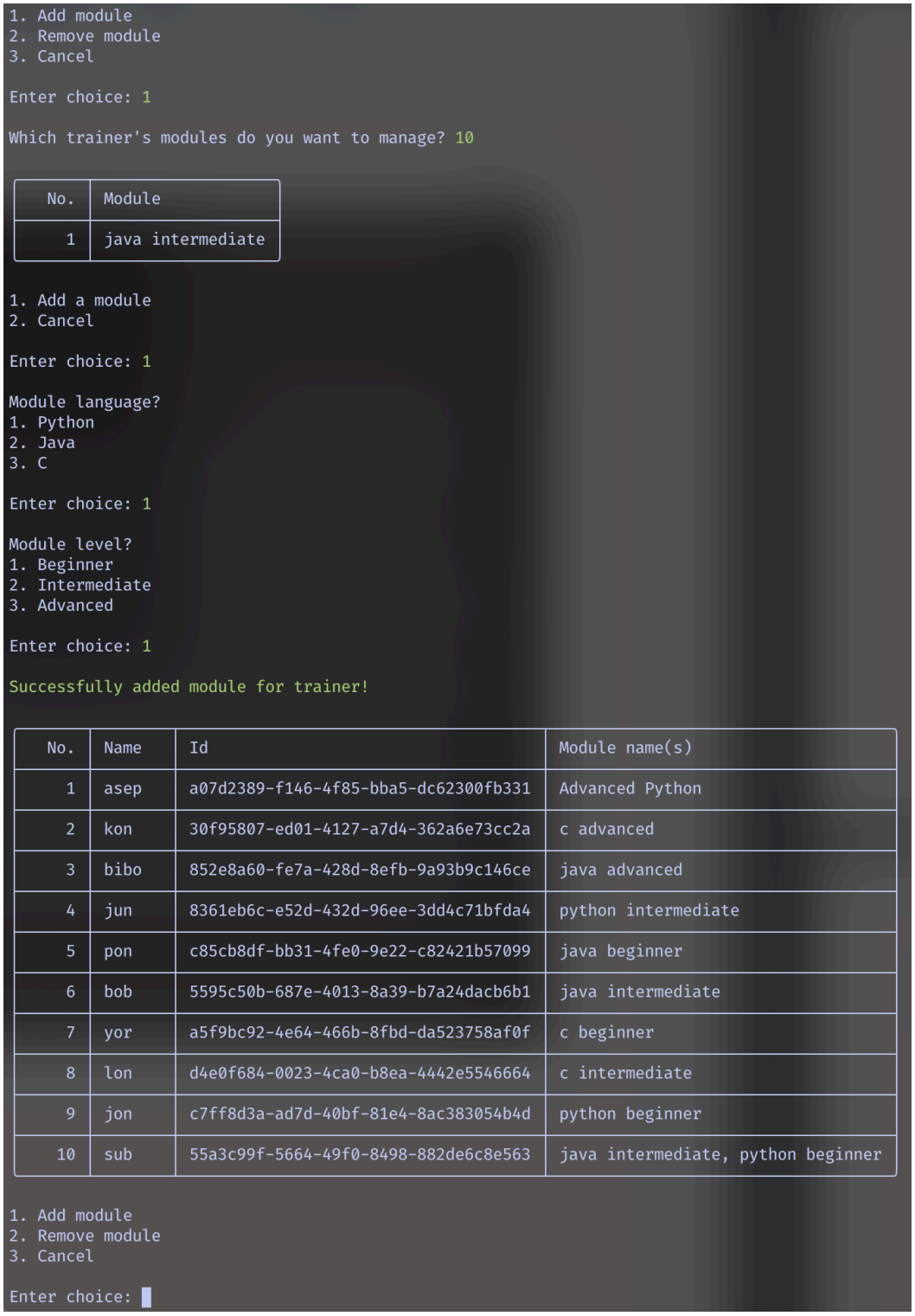
```
1. Register trainer
2. Delete trainer
3. Manage trainers' modules
4. Check trainer salaries
5. Back

Enter choice:
```

Deleting a trainer: In this menu, a table is displayed listing all available trainers with their ids and module names. From there, the user can choose to delete a trainer or cancel the action. If

they choose to delete a trainer, they will be prompted to enter the index of the trainer they want to delete, and afterwards a successful confirmation text will appear.



Manage trainer's modules menu: In this menu, admins can add or remove modules for trainers. They are first shown a table of all of the trainers available with the other necessary information to make a decision on whether to add or remove a module from someone.



```
1. Add module
2. Remove module
3. Cancel

Enter choice: 1

Which trainer's modules do you want to manage? 10
```

No.	Module
1	java intermediate

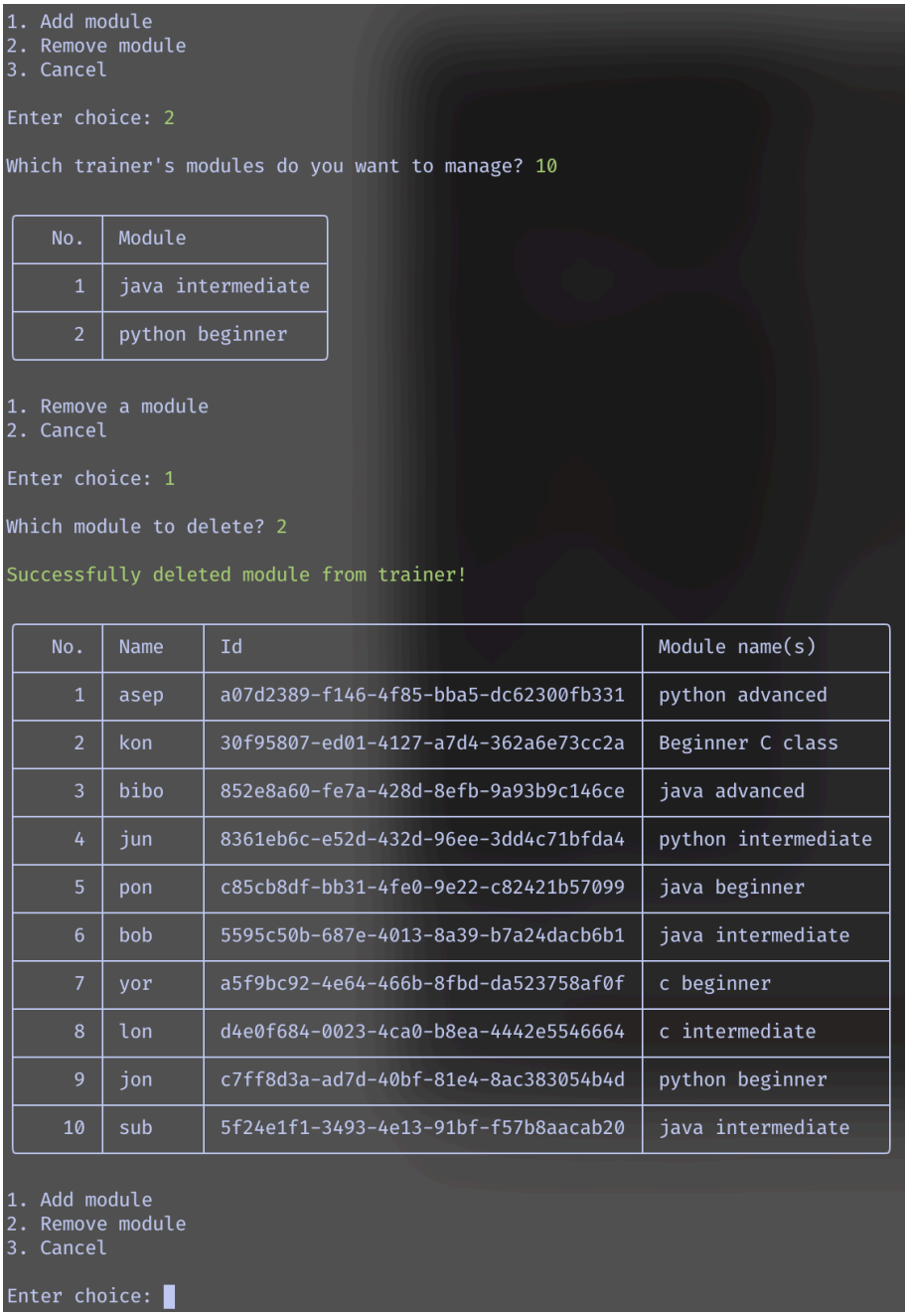
```
1. Add a module
2. Cancel

Enter choice: 1

Module language?
1. Python
2. Java
3. C

Enter choice: 1

Module level?
1. Beginner
2. Intermediate
3. Advanced

Enter choice: 1

Successfully added module for trainer!
```

No.	Name	Id	Module name(s)
1	asep	a07d2389-f146-4f85-bba5-dc62300fb331	Advanced Python
2	kon	30f95807-ed01-4127-a7d4-362a6e73cc2a	c advanced
3	bibo	852e8a60-fe7a-428d-8efb-9a93b9c146ce	java advanced
4	jun	8361eb6c-e52d-432d-96ee-3dd4c71bfda4	python intermediate
5	pon	c85cb8df-bb31-4fe0-9e22-c82421b57099	java beginner
6	bob	5595c50b-687e-4013-8a39-b7a24dacb6b1	java intermediate
7	yor	a5f9bc92-4e64-466b-8fbd-da523758af0f	c beginner
8	lon	d4e0f684-0023-4ca0-b8ea-4442e5546664	c intermediate
9	jon	c7ff8d3a-ad7d-40bf-81e4-8ac383054b4d	python beginner
10	sub	55a3c99f-5664-49f0-8498-882de6c8e563	java intermediate, python beginner

```
1. Add module
2. Remove module
3. Cancel

Enter choice:
```

Adding a module to a trainer: When adding a module to a trainer, the admin will be prompted to enter the index of the trainer for whom they want to add a module, based on the table. Following that, a table listing the current module names that the trainer has will be

displayed, and the admin can choose to proceed with adding a module or cancel the action. If they choose to proceed with adding a module, they will be prompted to select the module language and level. After selecting the necessary items, a successful confirmation text will be displayed, and the admin will be taken back to the "Manage Trainers' Modules" menu.



```
1. Add module
2. Remove module
3. Cancel

Enter choice: 2

Which trainer's modules do you want to manage? 10
```

No.	Module
1	java intermediate
2	python beginner

```
1. Remove a module
2. Cancel

Enter choice: 1

Which module to delete? 2

Successfully deleted module from trainer!
```

No.	Name	Id	Module name(s)
1	asep	a07d2389-f146-4f85-bba5-dc62300fb331	python advanced
2	kon	30f95807-ed01-4127-a7d4-362a6e73cc2a	Beginner C class
3	bibo	852e8a60-fe7a-428d-8efb-9a93b9c146ce	java advanced
4	jun	8361eb6c-e52d-432d-96ee-3dd4c71bfda4	python intermediate
5	pon	c85cb8df-bb31-4fe0-9e22-c82421b57099	java beginner
6	bob	5595c50b-687e-4013-8a39-b7a24dacb6b1	java intermediate
7	yor	a5f9bc92-4e64-466b-8fbd-da523758af0f	c beginner
8	lon	d4e0f684-0023-4ca0-b8ea-4442e5546664	c intermediate
9	jon	c7ff8d3a-ad7d-40bf-81e4-8ac383054b4d	python beginner
10	sub	5f24e1f1-3493-4e13-91bf-f57b8aacab20	java intermediate

```
1. Add module
2. Remove module
3. Cancel

Enter choice:
```

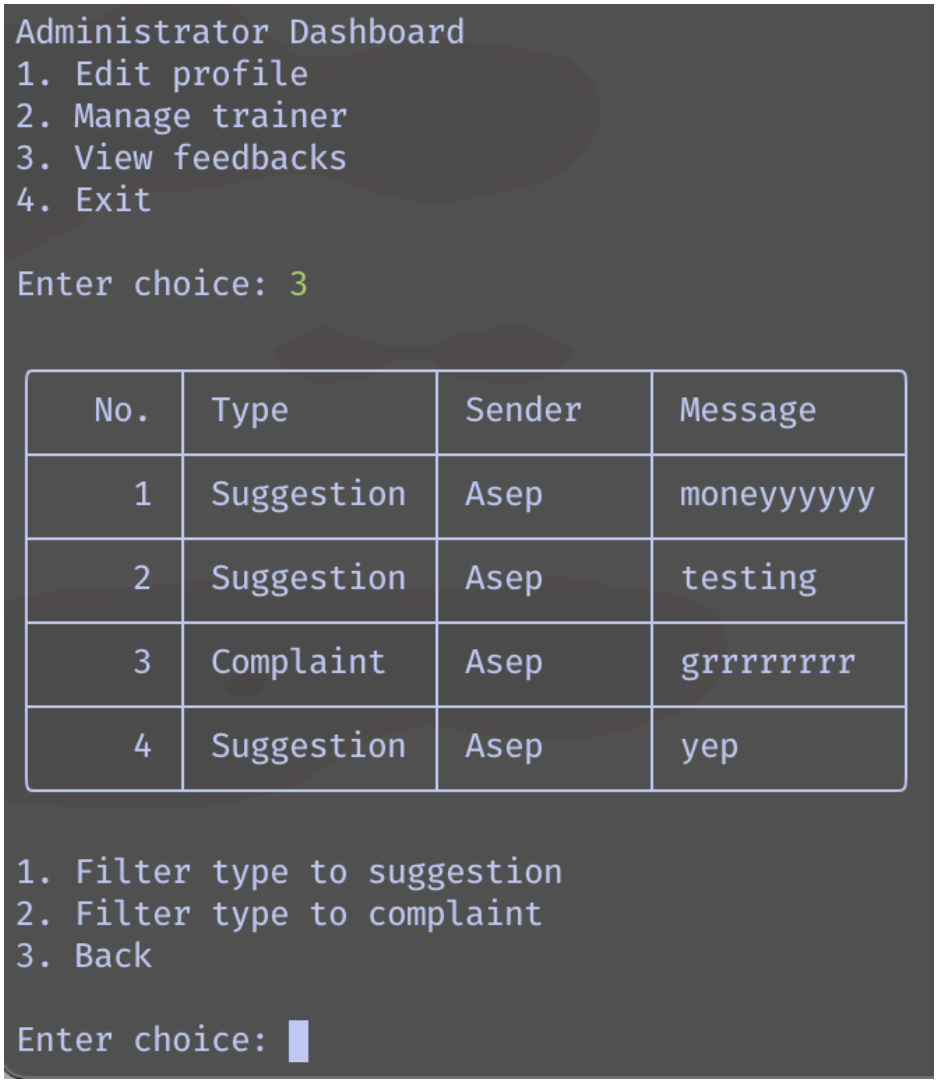
Removing a module from a trainer: This menu follows a similar action flow to adding a module, except that after the admin enters the trainer's index and proceeds to remove a module, they will be prompted to enter the index of the module they want to remove from the trainer.

```
1. Register trainer
2. Delete trainer
3. Manage trainers' modules
4. Check trainer salaries
5. Back

Enter choice: 4
```

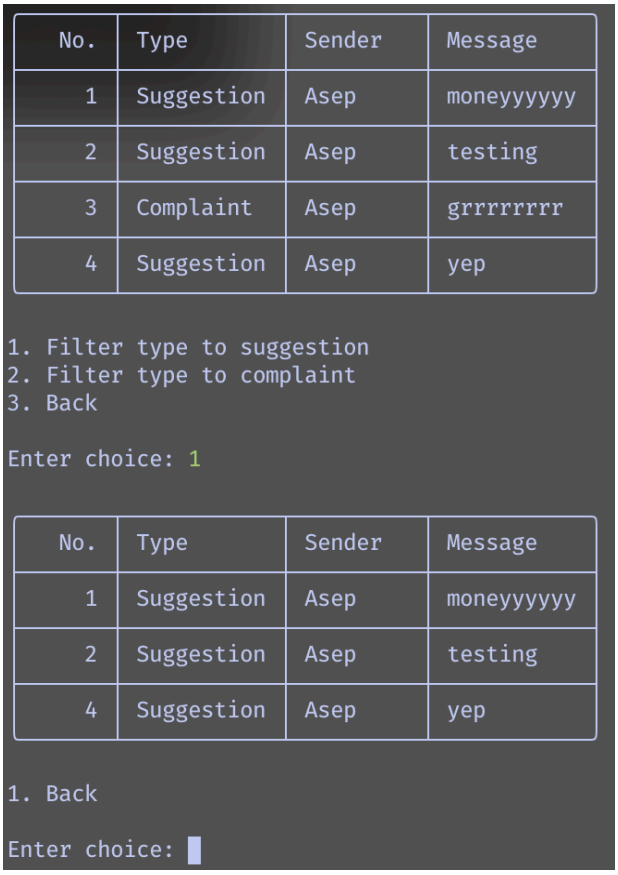
No.	Name	Module(s)	Salary
1	asep	python (advanced)	2000
2	kon	c (advanced)	2000
3	bibo	java (advanced)	0
4	jun	python (intermediate)	0
5	pon	java (beginner)	1000
6	bob	java (intermediate)	0
7	yor	c (beginner)	0
8	lon	c (intermediate)	0
9	jon	python (beginner)	1000

```
Press enter to continue...

1. Register trainer
2. Delete trainer
3. Manage trainers' modules
4. Check trainer salaries
5. Back

Enter choice:
```

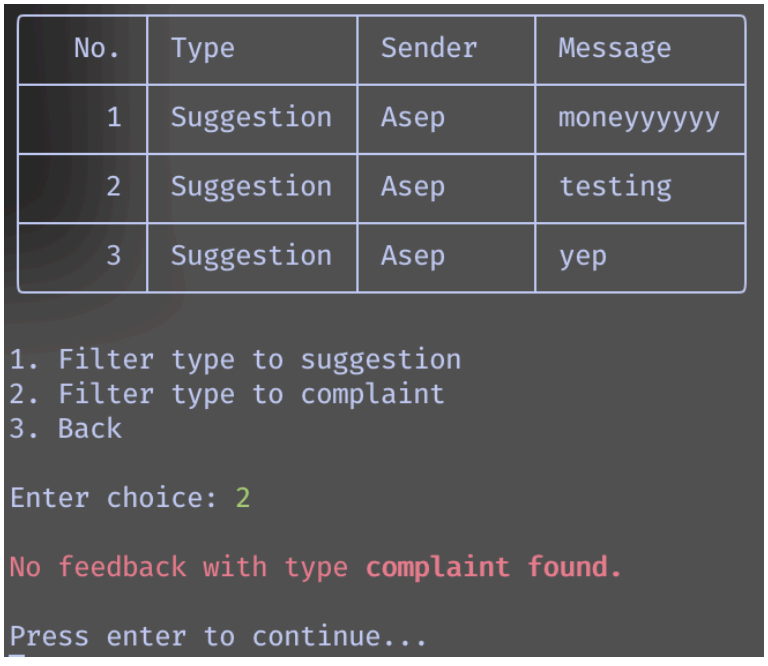
Check trainer salaries menu: Administrators are able to check all of the trainers' salaries which is displayed in a table format.



View feedbacks menu: Administrators can see all of the feedbacks that were given my trainers in a table format. They can also filter the feedbacks by type if they choose to do so.

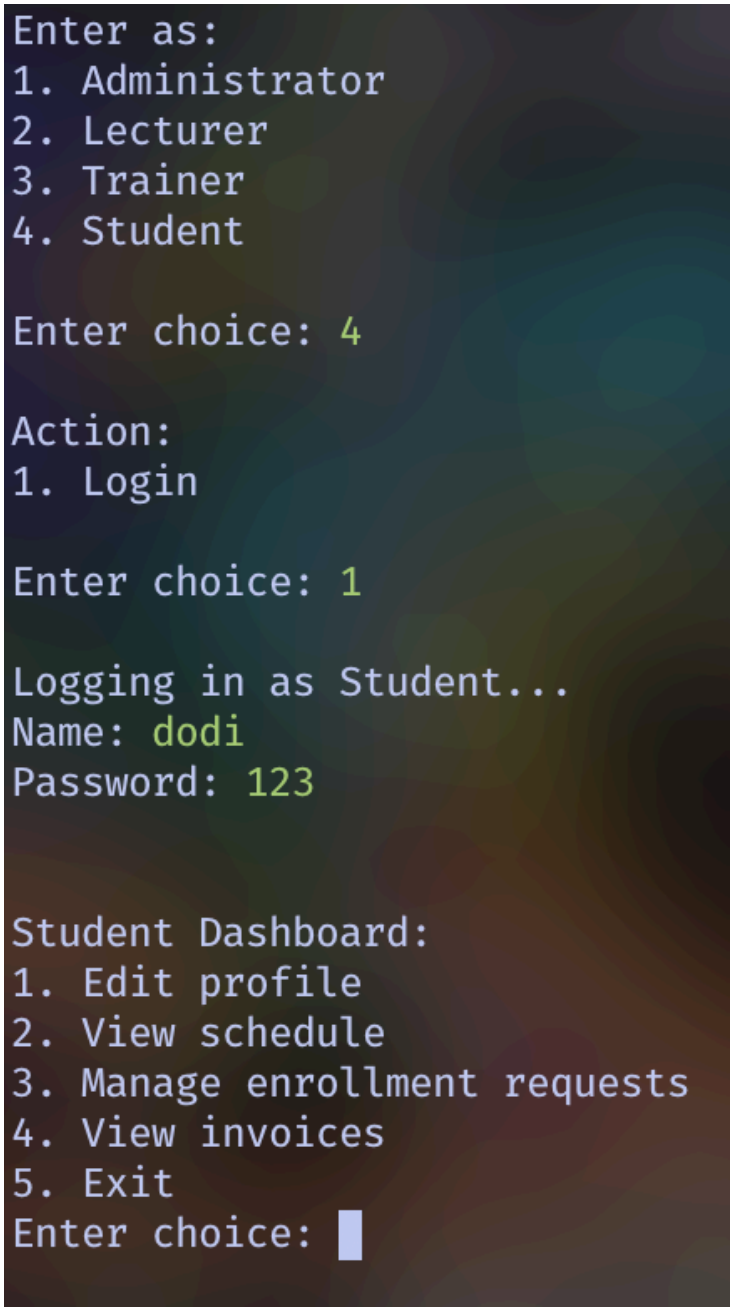


Filtering feedback type to suggestions only

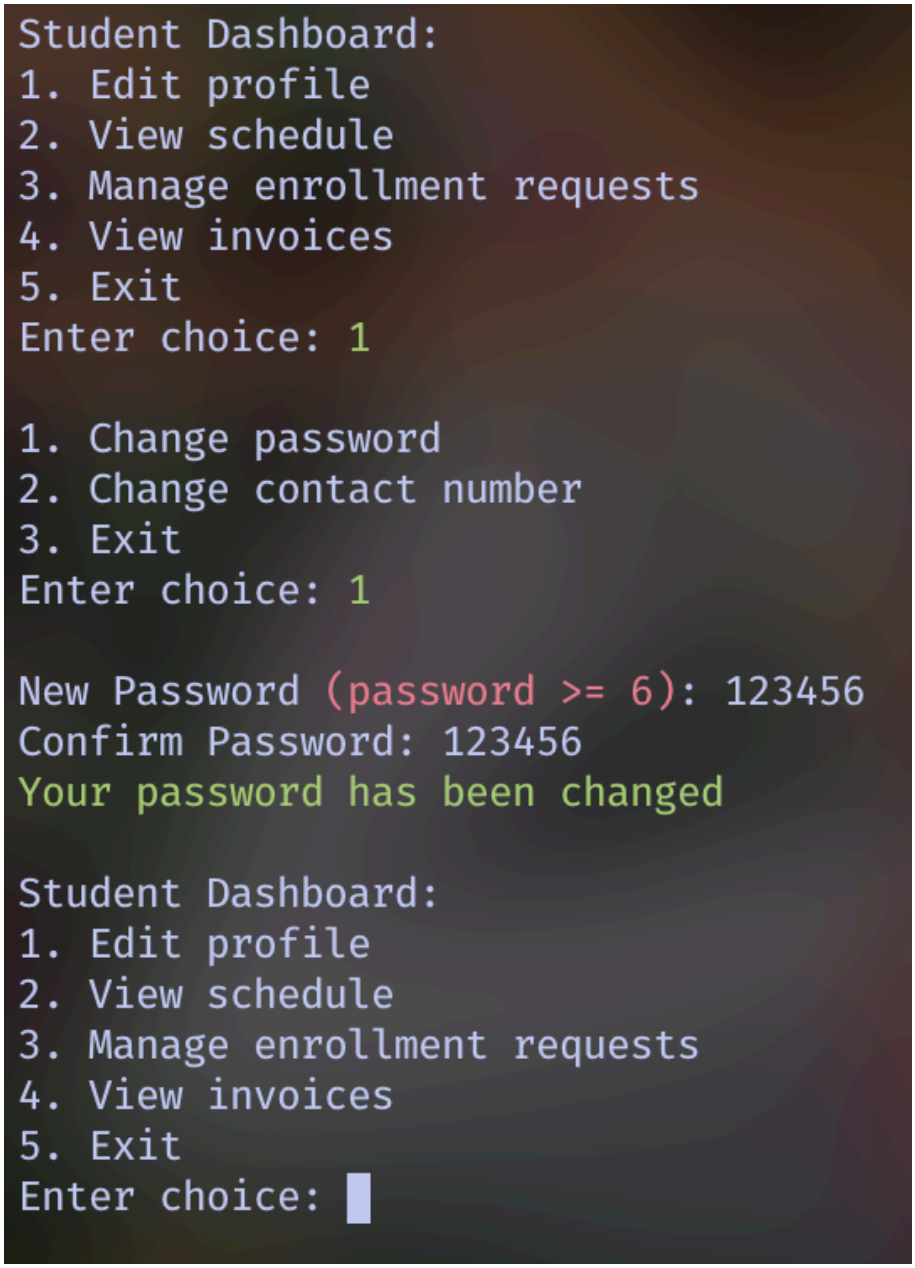


Output when no feedback with the specific type isn't found.

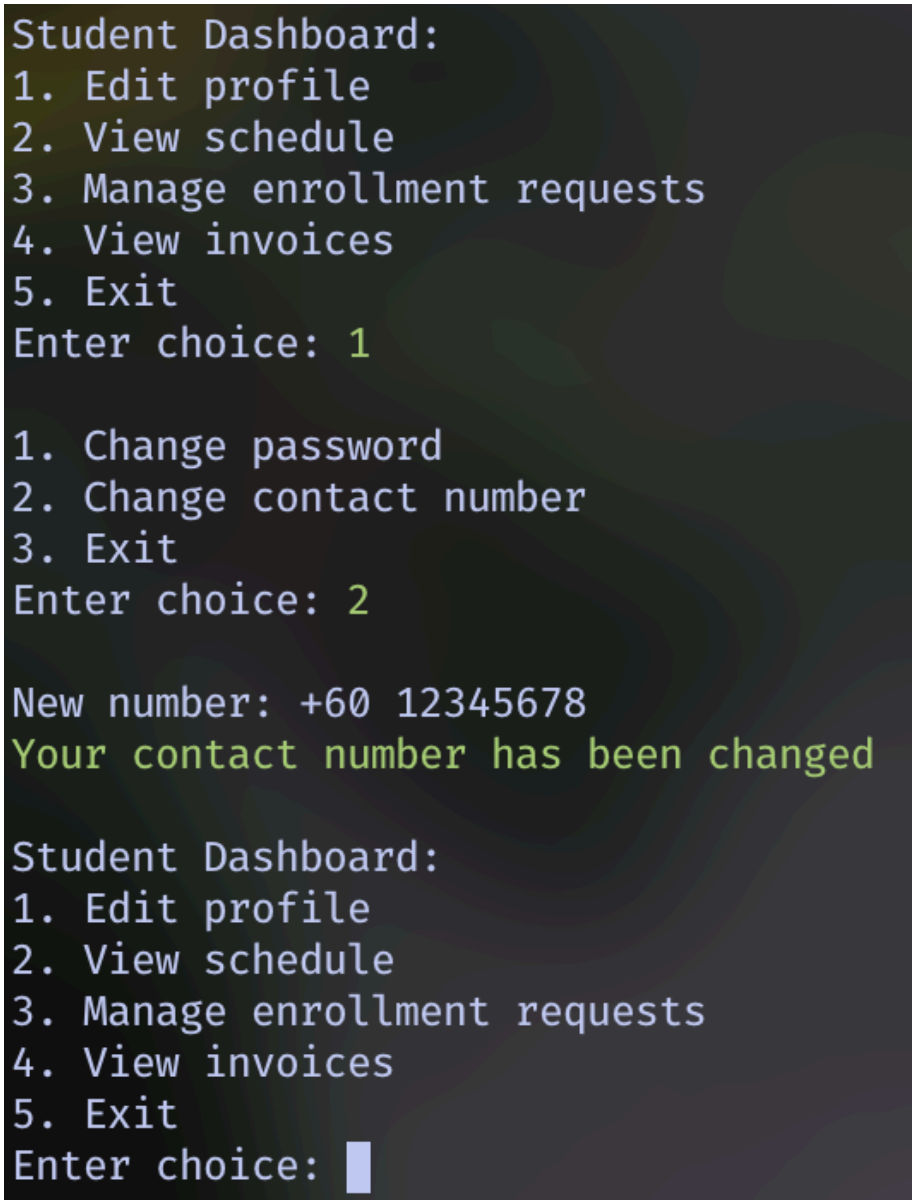
Student Menu



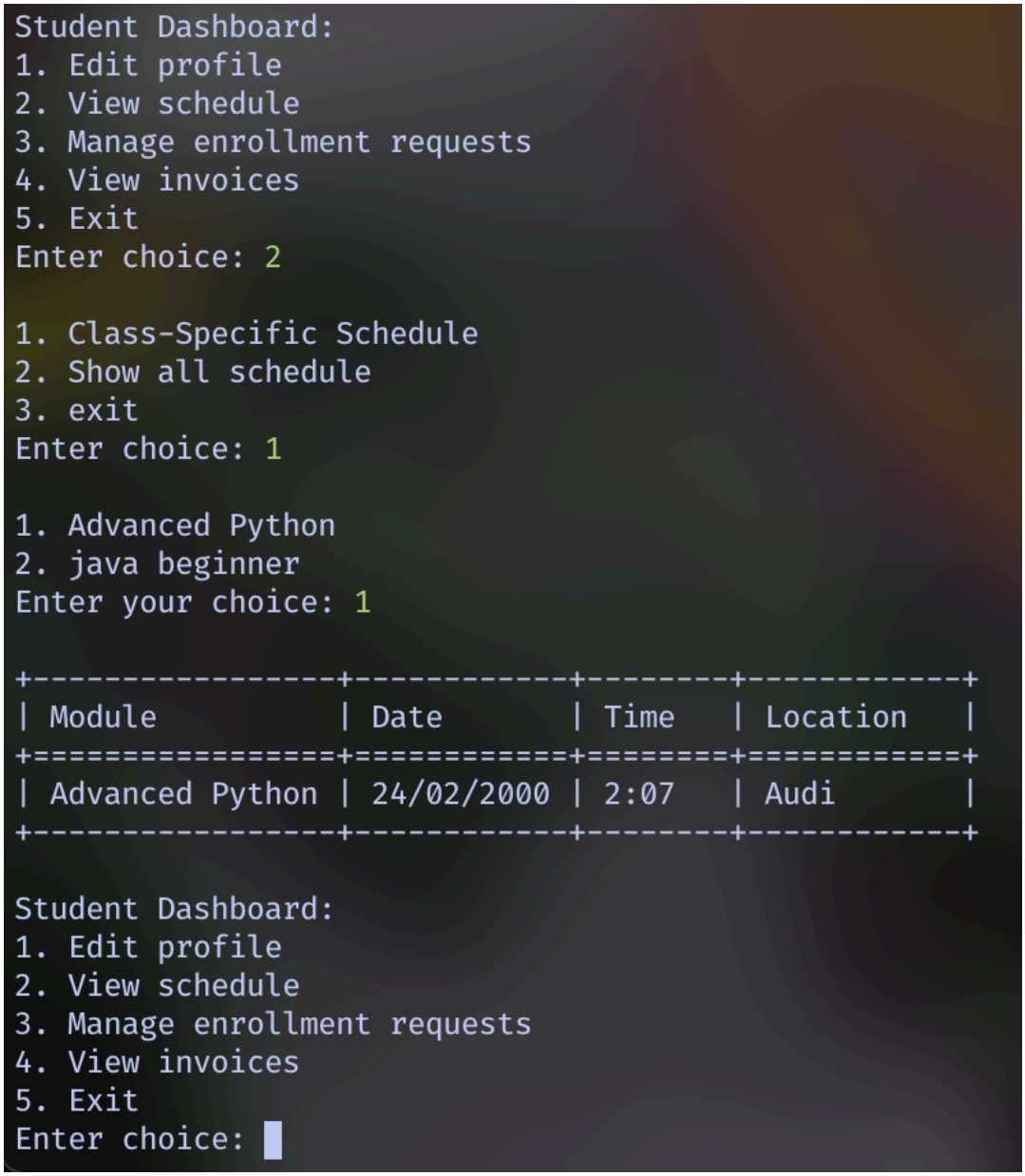
This is the student menu that will be displayed after the student logs in; the student is given options to choose from 1-5. The student can enter the number of the option the student wants to enter in Enter choice:.



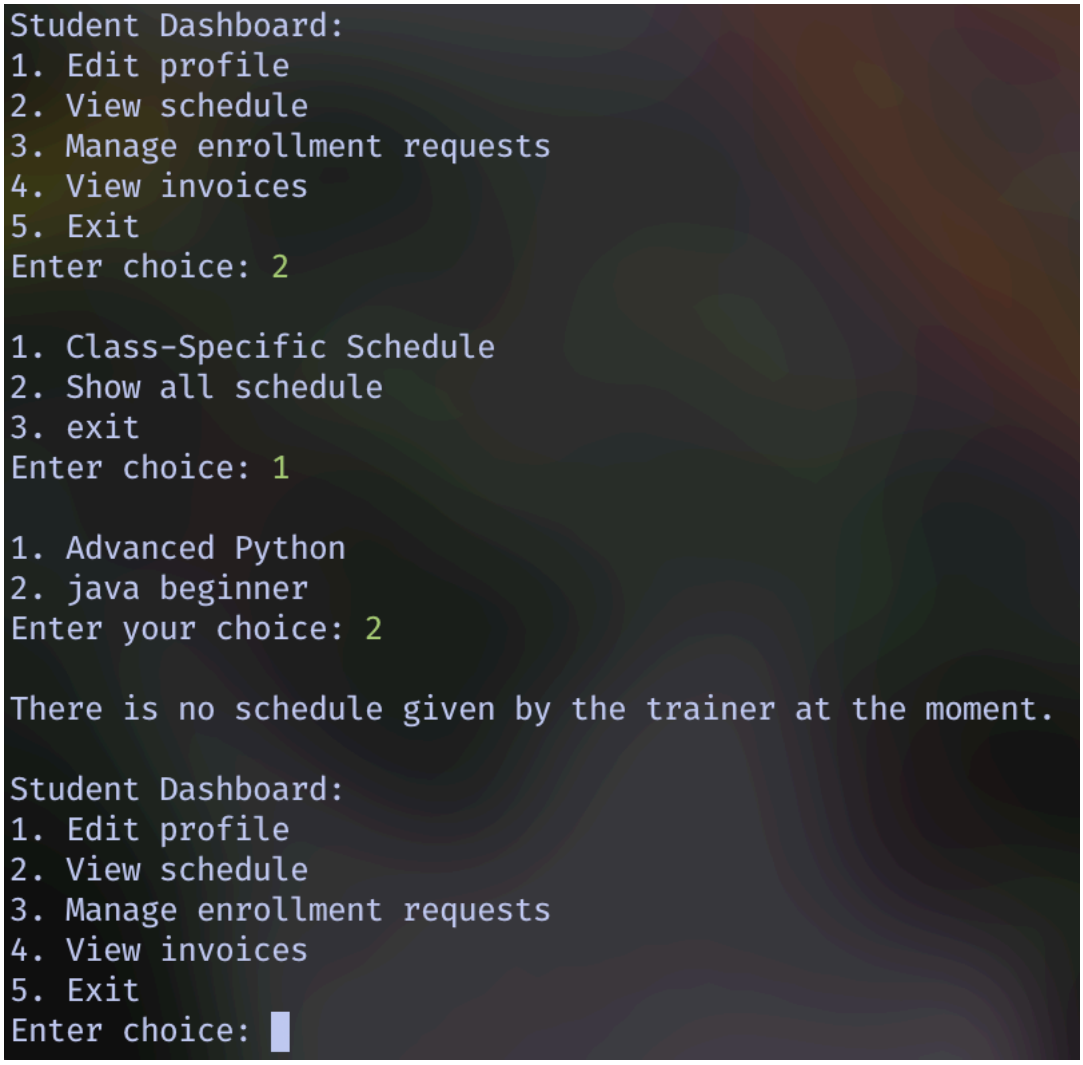
In the edit profile you can change your password and the new password has to be at least 6 characters.



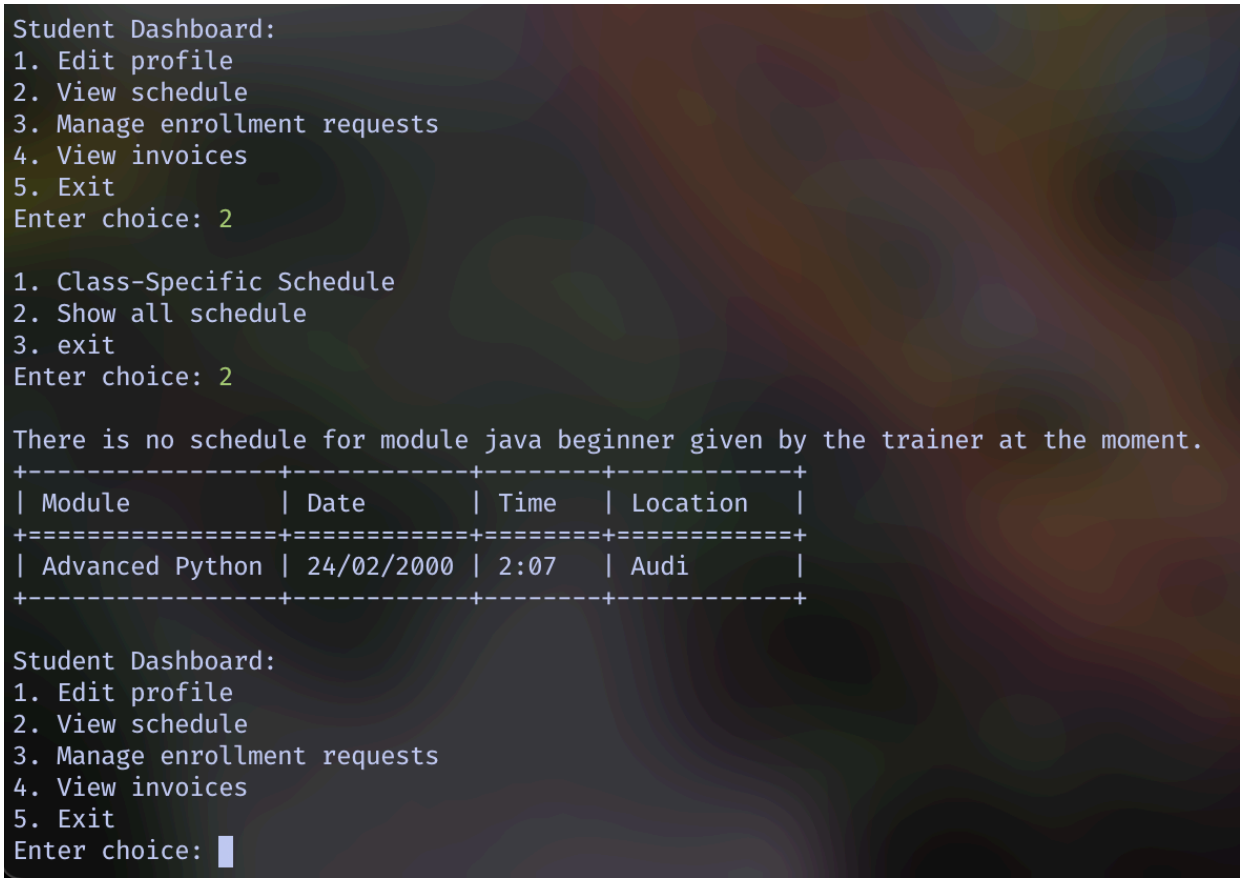
In the edit profile the student can also change their contact number and the student is only allowed to change it to a Malaysian number.



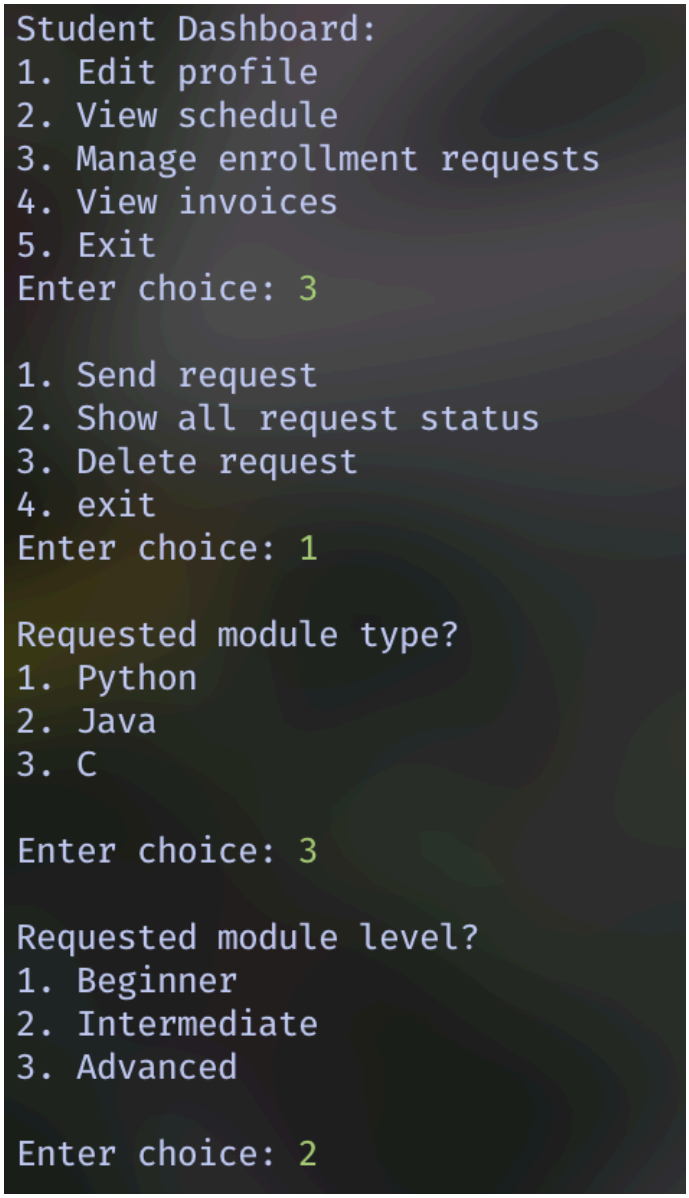
In view schedule the student can view class specific schedule and in class class specific schedule the student is given the option to see the schedule of one specific class as shown in the image.



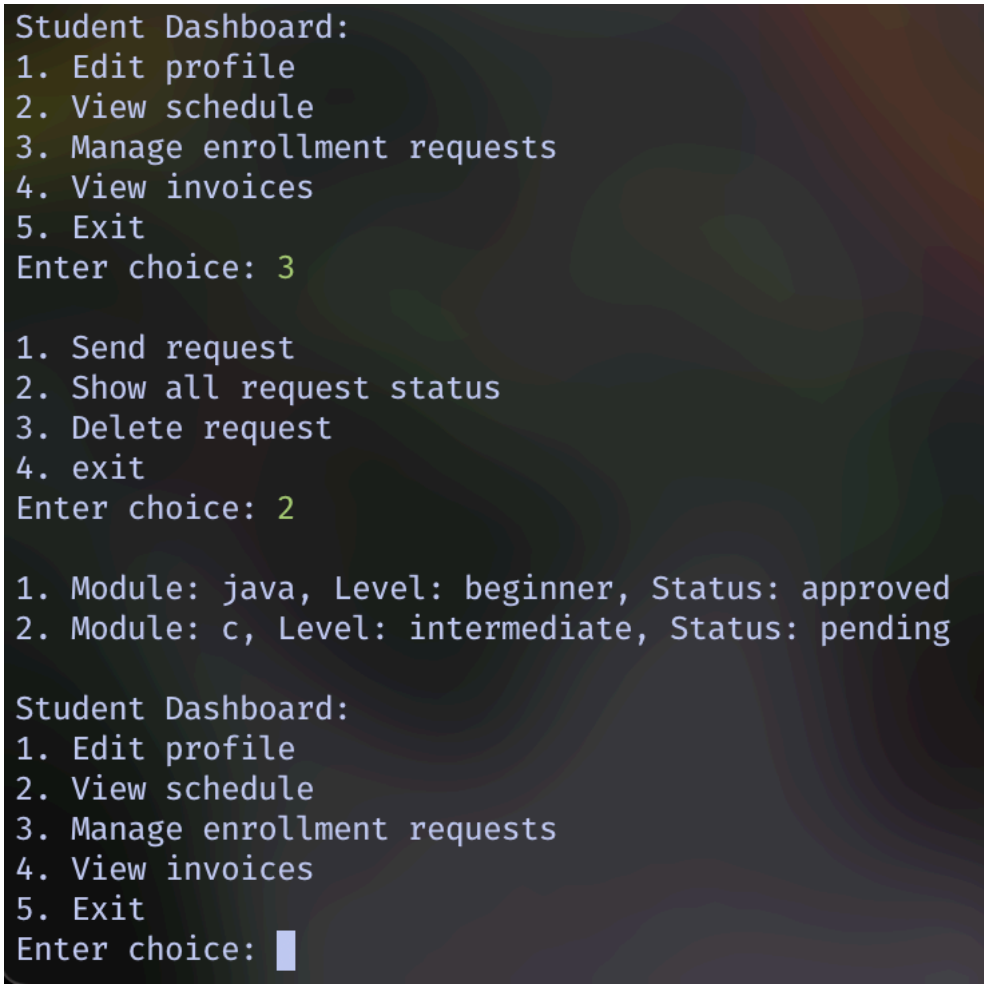
This image shows the result if a trainer has not yet given a schedule.



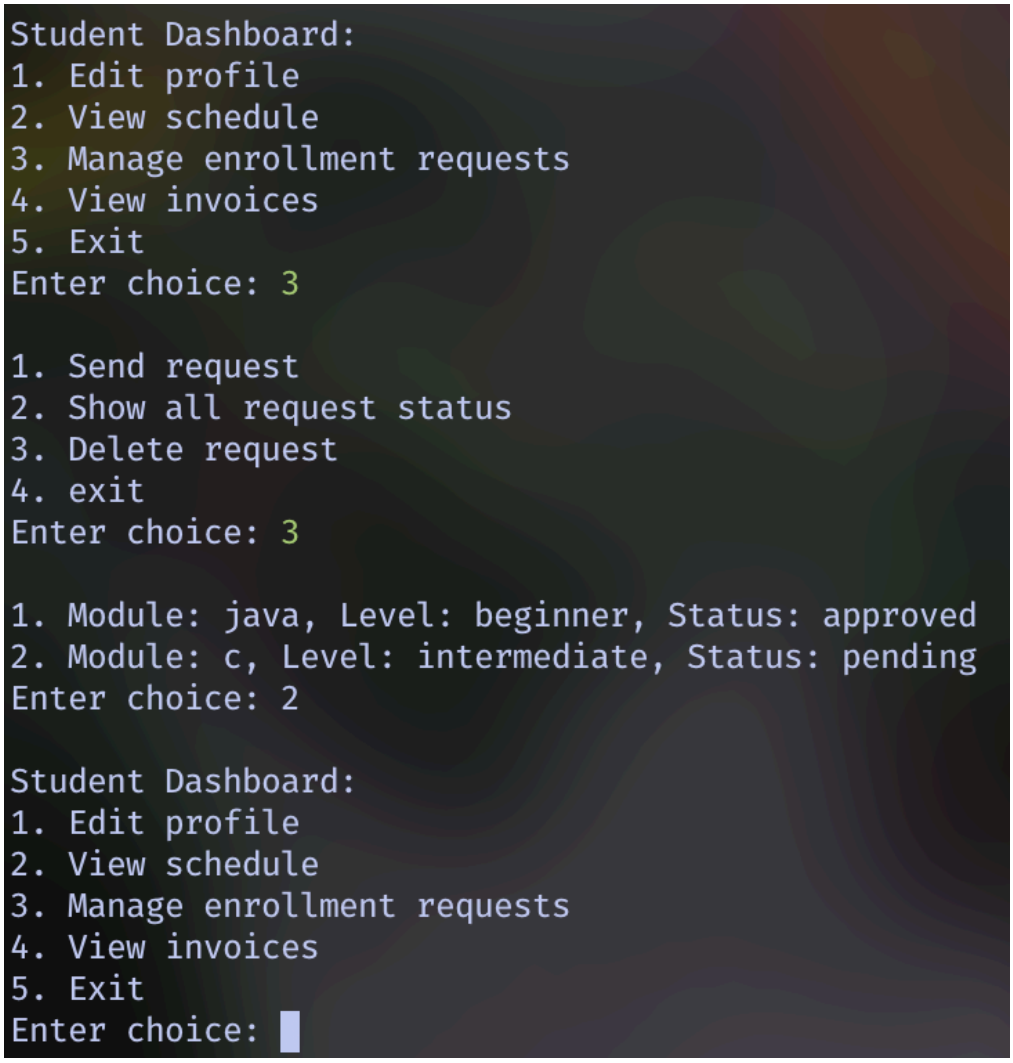
This image shows the result of choosing the show all schedule option.



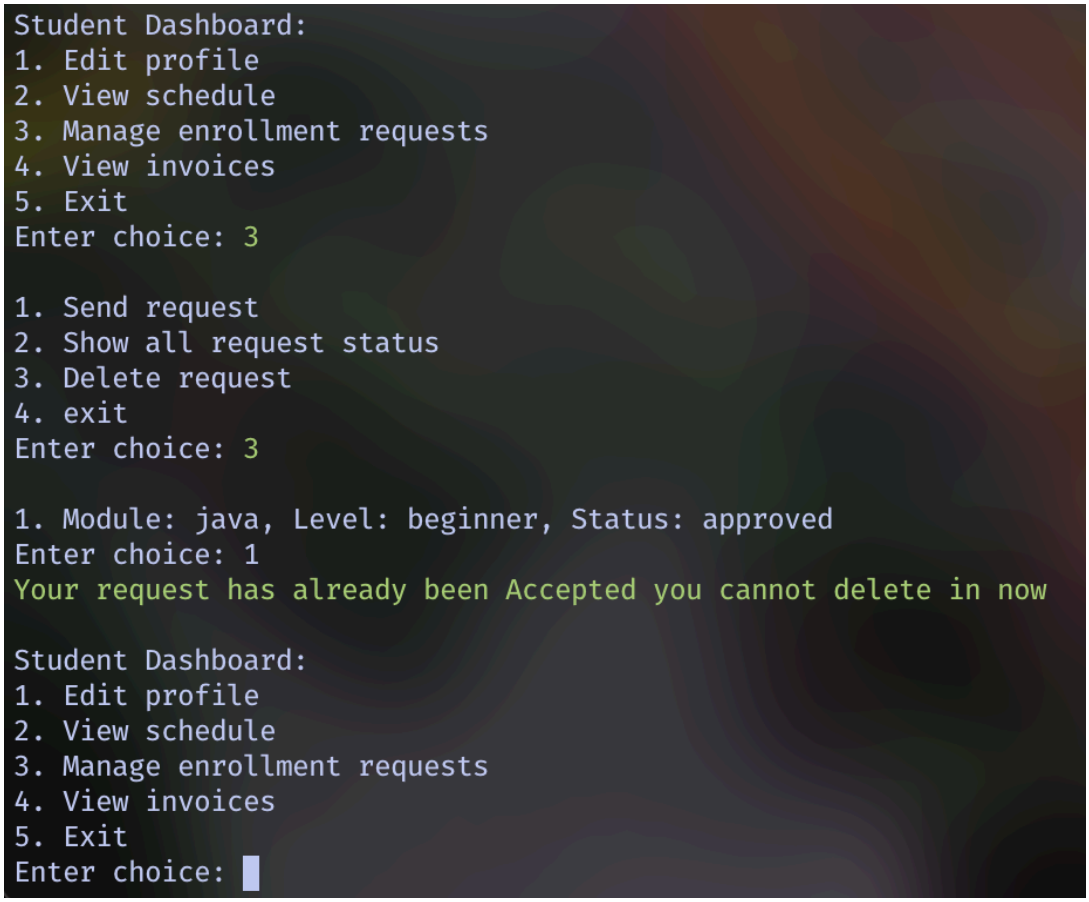
In this image shows the manage request option results after entering manage request the student has chosen to send a request to the lecturer to apply to a new module or a new level.



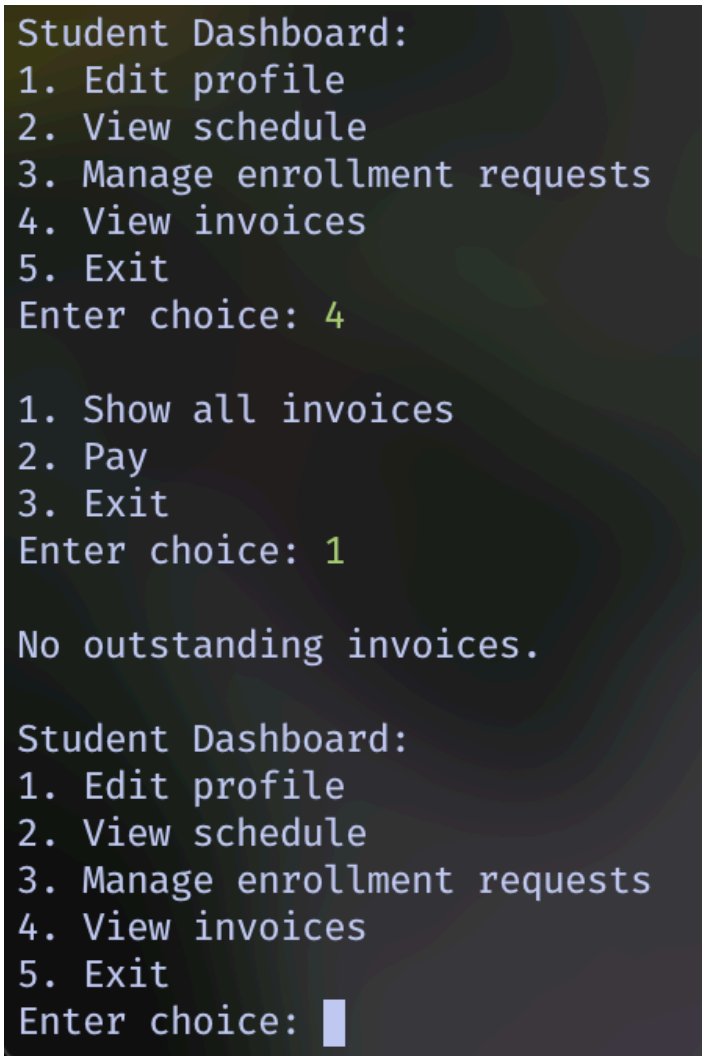
In manage requests you can also check the status of your request if it has been approved or not.



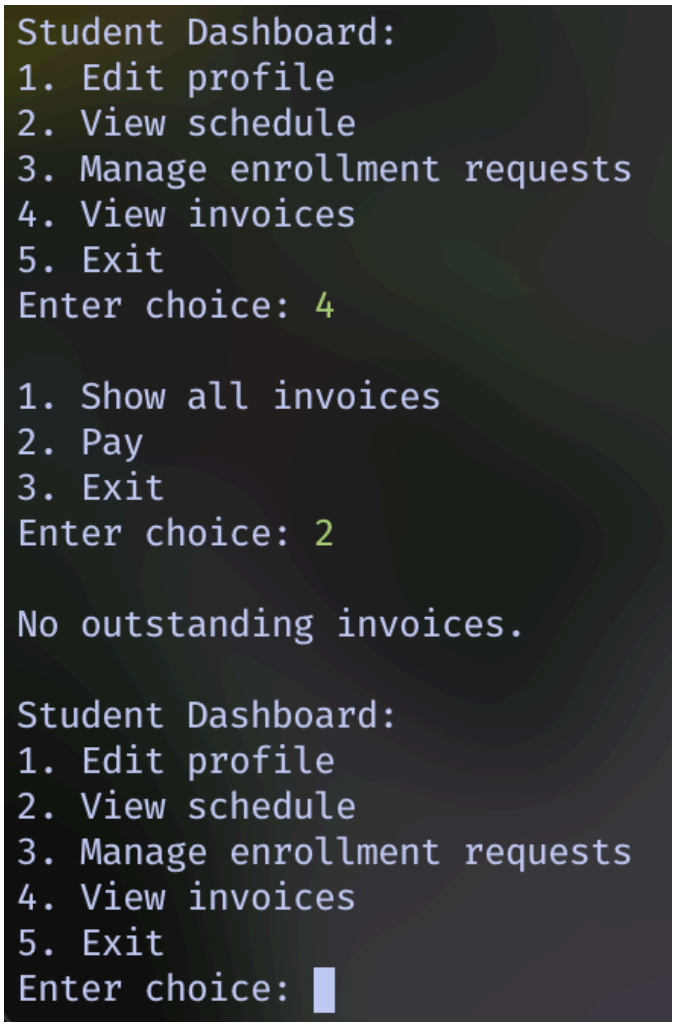
In manage requests you can also delete requests if the request is still pending.



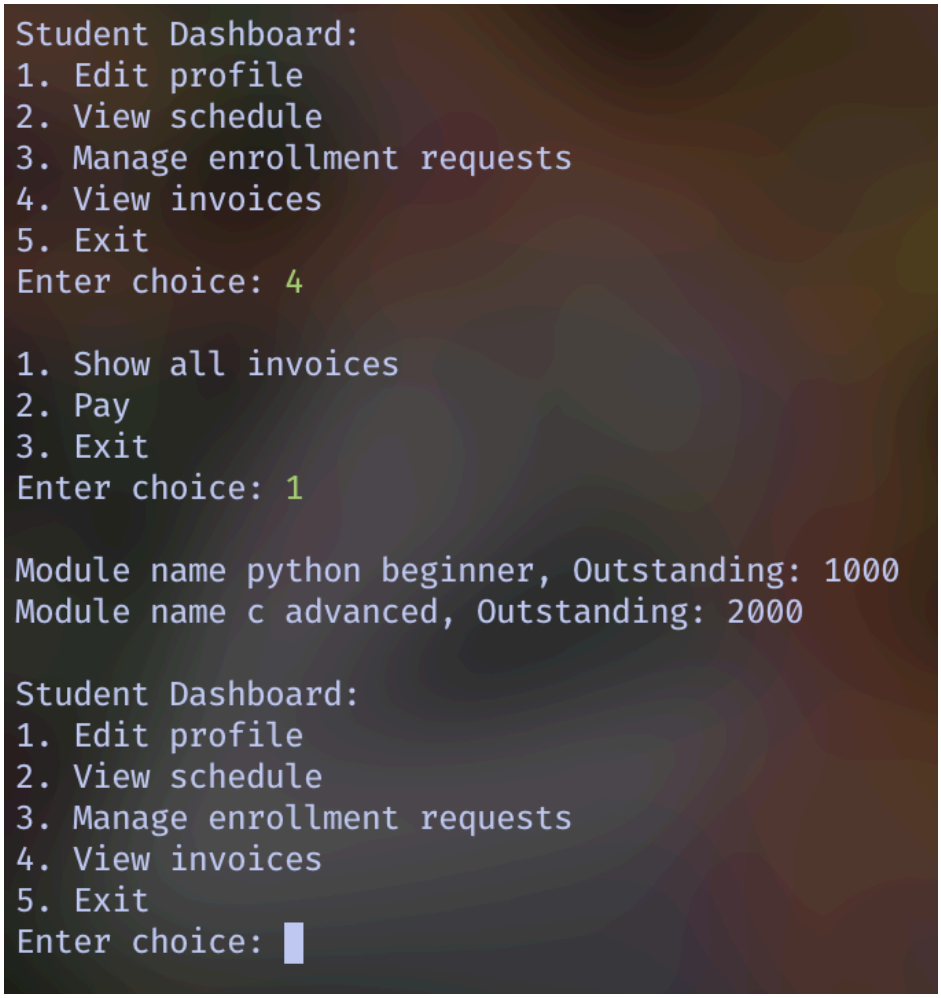
This image shows the result of the request has been approved and the student still wants to delete it.



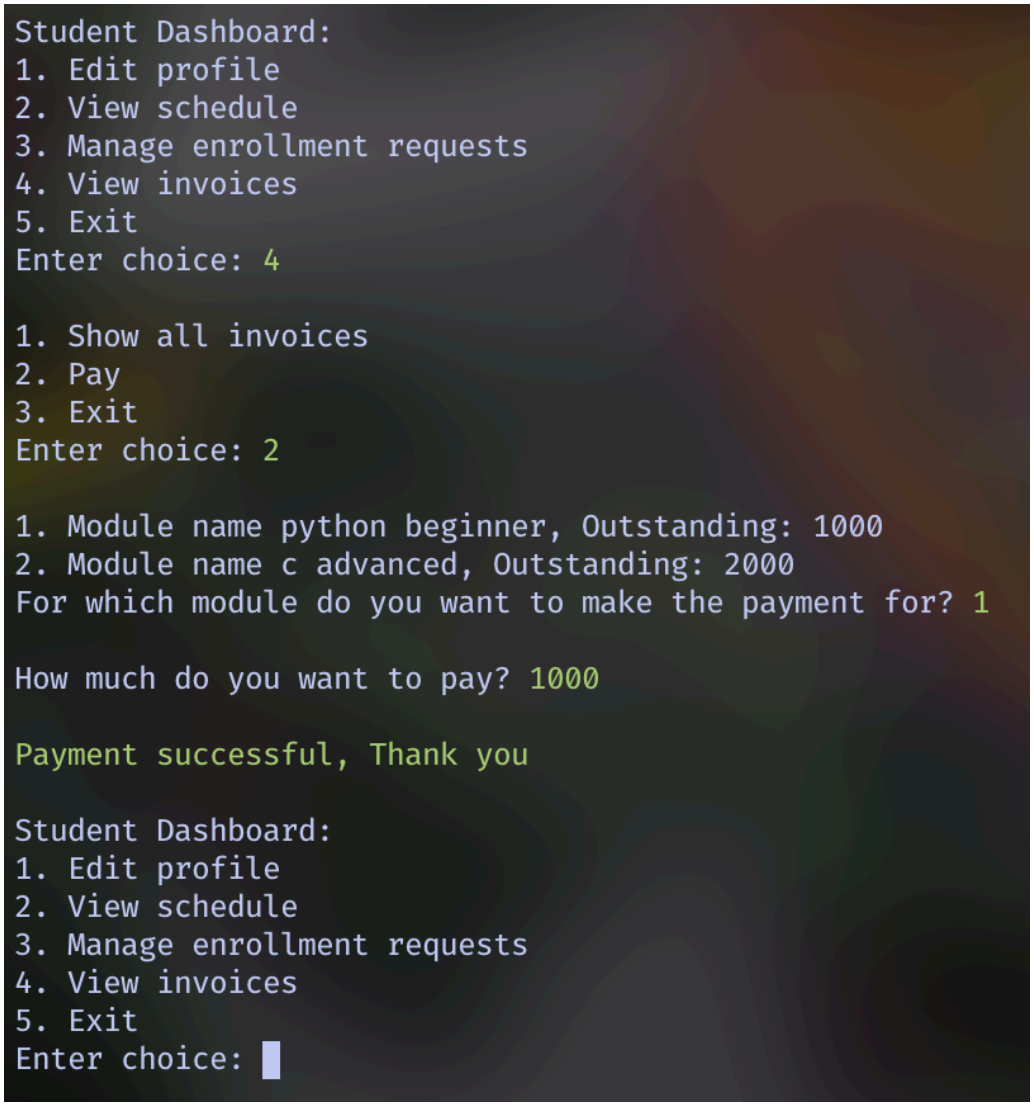
In view invoice the student can check all the invoices you have by entering show all invoice.



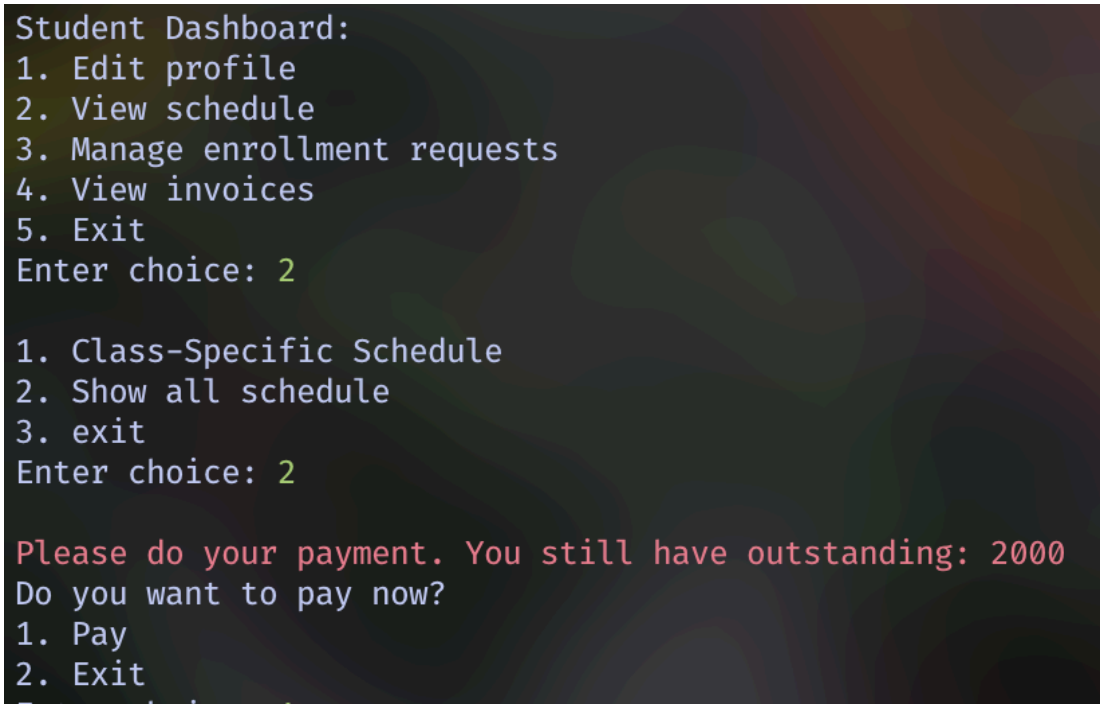
In view invoices the student can pay for the modules the student is taking and the image shows the result of a student who has paid for all the modules.



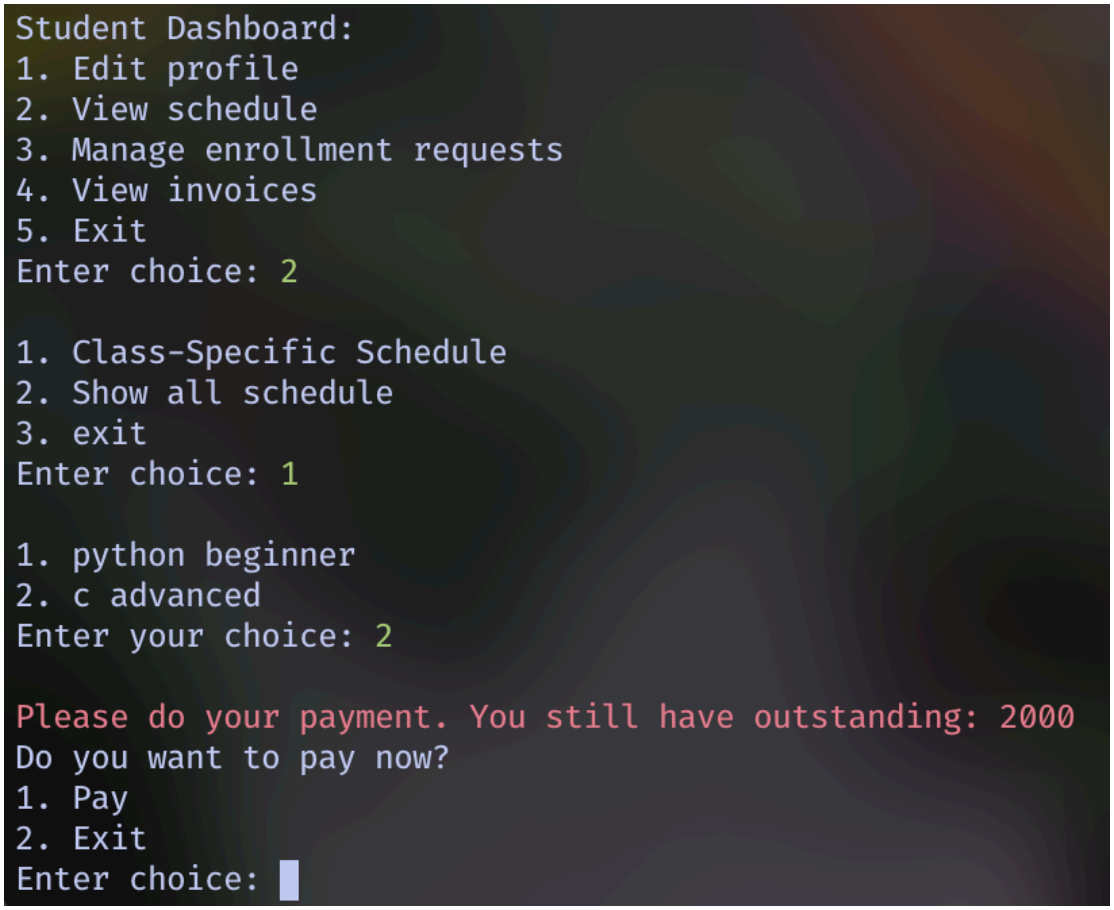
In this image shows a student who has not paid for these particular modules which is being checked by entering show all invoices.



In this image shows a student who has not paid for these particular modules so the student is given the option to pay for whichever module the student wants.



This image shows the result of a student who has not paid for their module and is trying to view the schedule.



This image shows the result of a student who has not paid for their module and is trying to view the schedule of that particular module.

Lecture Menu

```
Enter as:
1. Administrator
2. Lecturer
3. Trainer
4. Student

Enter choice: 2

Action:
1. Login
2. Register

Enter choice: 1
```

Login as lecturer

```
Logging in as Lecturer...
Name: Faiyaad
Password: 123
Wrong password!
Password: 678
Wrong password!
Password: 567
```

Trying to log in but the password was wrong so the user retyped the password and tried to log in again and was successful in the end. There are only three chances for typing in the correct password, otherwise, the program finishes

Registration of Lecturer

```
Enter as:
1. Administrator
2. Lecturer
3. Trainer
4. Student

Enter choice: 2

Action:
1. Login
2. Register

Enter choice: 2

Register as lecturer
Name: wolly
Password: 123

Process finished with exit code 0
```

A new user for the lecturer has been registered successfully using option 2 from “Action.”

Edit Profile Option For Lecturer

```
Lecturer dashboard
1. Edit profile
2. Manage students
3. View all enrollment requests
4. Exit

Enter choice: 1

1. Edit name
2. Edit password
3. Back

Enter choice: 1


Enter new name: Faiyaad
1. Edit name
2. Edit password
3. Back

Enter choice: 2
```

In the Lecturer dashboard where the user has chosen option 1 for the edit profile option the name is changed to Faiyaad

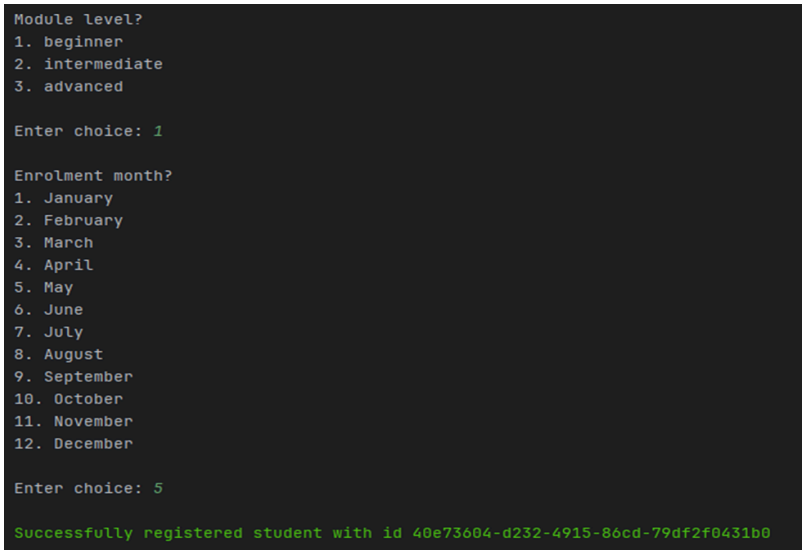
```
1. Edit name
2. Edit password
3. Back

Enter choice: 2


Enter new password: 456
1. Edit name
2. Edit password
3. Back

Enter choice: 3

Lecturer dashboard
1. Edit profile
2. Manage students
3. View all enrollment requests
4. Exit

Enter choice:
```

Where “2” is chosen afterward to enter a new password and 3 is chosen to go back to the dashboard.

Registration Of Students

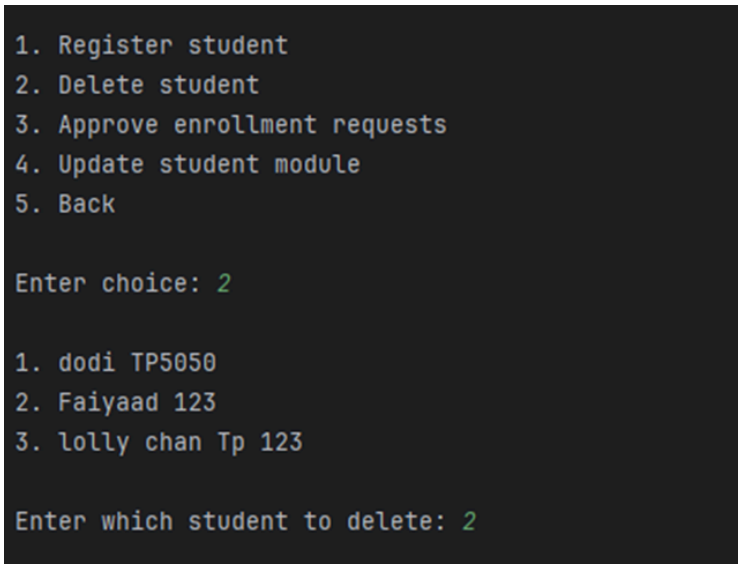
```
1. Register student
2. Delete student
3. Approve enrollment requests
4. Update student module
5. Back

Enter choice: 1

Name: lolly chan
TP Number: Tp 123
Contact number: 67890
Address: blok:k
Module type?
1. python
2. java
3. c

Enter choice: 2
```

Used for Registration of students and assigning modules for the enrollment of the student.



Choosing the module level of the student and month of enrollment to be registered successfully

Delete Student Option



Used to delete student “Faiyaad” from enrolled students.

Approve Enrollment Requests Of Students

```
1. Register student
2. Delete student
3. Approve enrollment requests
4. Update student module
5. Back

Enter choice: 3

1. dodi
2. lolly chan

Enter choice: 1

1. java

Enter choice: 1
```

Used to approve request enrollment for students such as dodi requesting to join class for java

```
Enrolment month?
1. January
2. February
3. March
4. April
5. May
6. June
7. July
8. August
9. September
10. October
11. November
12. December

Enter choice: 5
```

This is used to choose the month of enrollment for student dodi

```
1. Register student
2. Delete student
3. Approve enrollment requests
4. Update student module
5. Back

Enter choice: 3

1. dodi
2. lolly chan

Enter choice: 2

No module requests found.
```

No request is seen for student “lolly chan”

Update student modules, level, and month of Enrollment

```
No module requests found.
1. Register student
2. Delete student
3. Approve enrollment requests
4. Update student module
5. Back

Enter choice: 4

1. dodi
2. lolly chan

Enter choice: 1

1. e5a9029c-f74f-4f25-8e44-7d11c37fa02c Advanced Python
2. f27575ee-9c15-4d2f-b738-547bf8b953d7 java beginner
3. f27575ee-9c15-4d2f-b738-547bf8b953d7 java beginner

Which module to change? 2

Module type to change to?
1. python
2. java
3. c
```

This is used to update modules for students if the student doesn’t want to continue with the module further or if the lecturer added a particular student to a different module by mistake.

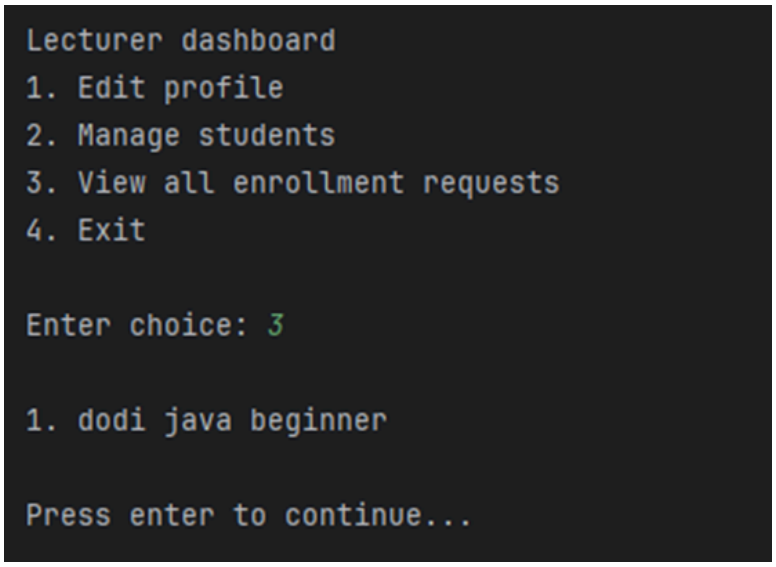
```
Module level to change to?
1. beginner
2. intermediate
3. advanced

Enter choice: 2

Enrollment month:
1. Jan
2. Nov
3. Dec

Enter choice: 1
```

This section is used to choose the level and enrollment month for the updated module.

```
1. Register student
2. Delete student
3. Approve enrollment requests
4. Update student module
5. Back

Enter choice: 5

Lecturer dashboard
1. Edit profile
2. Manage students
3. View all enrollment requests
4. Exit
```

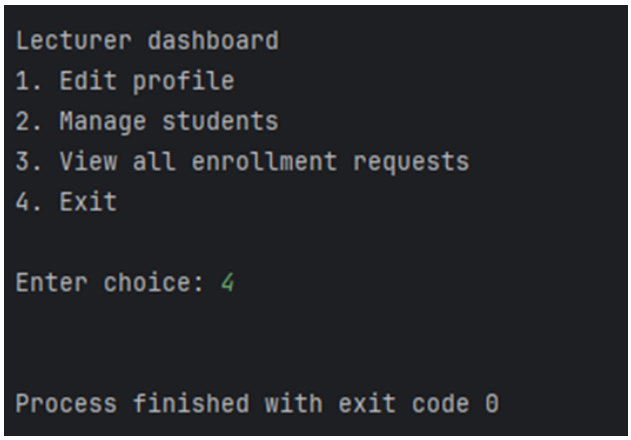
Where the “Back” option is used to go back to the lecturer’s dashboard.

View All Enrollment Requests



This option is used to see all enrollment requests from students to enroll in the modules and press enter to continue to the lecturer's dashboard

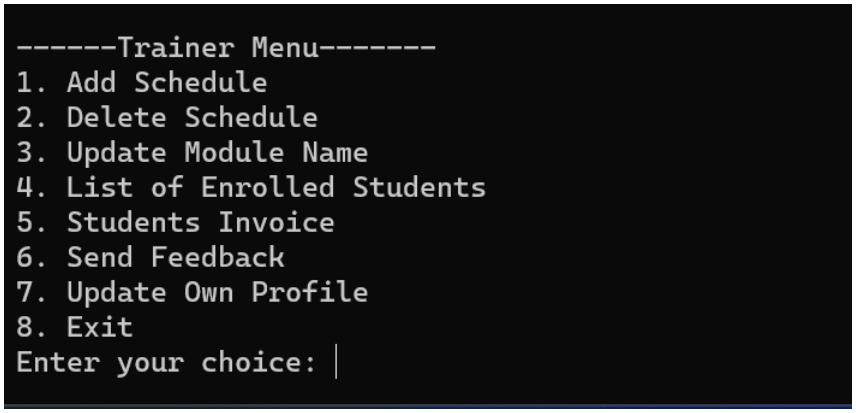
Exit



The exit option stops the program from running and you are done for the day

Trainer's Menu

Login page in the main menu and input a number that corresponds to the selected role. And login as a trainer with input the name and password.



This is the Trainer's Menu, In here listed 8 items of Trainer's duty

```
------Trainer Menu-------
1. Add Schedule
2. Delete Schedule
3. Update Module Name
4. List of Enrolled Students
5. Students Invoice
6. Send Feedback
7. Update Own Profile
8. Exit
Enter your choice: 1
1. python advanced
Enter your choice: 1
Enter the date for the class schedule (dd/mm/yyyy): 24-02-2024
Invalid date format. Please use the correct date format (dd/mm/yyyy)

Enter the date for the class schedule (dd/mm/yyyy): 24/02/2024
Enter the time for the class schedule (hh:mm): 1.40
Invalid time format. Please use the correct time format (hours:minute)

Enter the time for the class schedule (hh:mm): 1:40
Enter the place for the class schedule: B-08-04

Coaching Class Added:
Date: 24/02/2024
Time: 1:40
Location: B-08-04
```

In this page, if we input 1, it will go directly to the add schedule page where it will display the coaching class information, that consists of modules and schedules. If we press the current module it will ask the user to input the schedule of that particular module. There are also validations added in there, if the user inputs the wrong format it will print an error message and ask user to input the right one.

```
------Trainer Menu-------
1. Add Schedule
2. Delete Schedule
3. Update Module Name
4. List of Enrolled Students
5. Students Invoice
6. Send Feedback
7. Update Own Profile
8. Exit
Enter your choice: 2
1. python advanced
Enter your choice: 1
```

On this page, if users input number 2, it will display the list of modules that the trainer teaches.

```
Enter your choice: 1
+----------------+-----------+----------------+
|      Date      |    Time   |    Location    |
+----------------+-----------+----------------+
|   24/12/2020   |   10:20   |    B-69-20     |
+----------------+-----------+----------------+
|   24/02/2024   |    1:45   |   Auditorium   |
+----------------+-----------+----------------+
|   24/02/2000   |    2:07   |      Audi      |
+----------------+-----------+----------------+
|   24/02/2024   |    1:40   |    B-08-04     |
+----------------+-----------+----------------+
Enter choice: 1
Schedule deleted!
```

On this page, the user will choose the module they want to edit. And it will display the list of schedules of that particular module. Here the user can choose which schedule that he wants to delete, after choosing it will print “Schedule deleted!”, the schedule in that module has been deleted.

```
------Trainer Menu-------
1. Add Schedule
2. Delete Schedule
3. Update Module Name
4. List of Enrolled Students
5. Students Invoice
6. Send Feedback
7. Update Own Profile
8. Exit
Enter your choice: 3
Which module name to change?
1. python advanced
Enter your choice: 1
What do you want to change the name to? Intermediate Python class
Module name changed to Intermediate Python class
```

On this page, if the user chooses number 3, he will be prompted to select a list of modules he wants to edit. The user is asked to change the module name and a message appears saying the module name has been changed.

```
------Trainer Menu-------
1. Add Schedule
2. Delete Schedule
3. Update Module Name
4. List of Enrolled Students
5. Students Invoice
6. Send Feedback
7. Update Own Profile
8. Exit
Enter your choice: 4
1. python advanced
Enter your choice: 1
+-------------+----------------+-------------------------------+
|     Name    |   TP Number    |             Email             |
+-------------+----------------+-------------------------------+
|     dodi    |     TP5050     |     tp5050@mail.apu.edu.my    |
+-------------+----------------+-------------------------------+
```

On this page, If user input number 4, he will be asked to choose a list of available modules. After that it displayed the information of students that were enrolled in that particular module. The information consists of the student's name, tp number. and email.

```
------Trainer Menu-------
1. Add Schedule
2. Delete Schedule
3. Update Module Name
4. List of Enrolled Students
5. Students Invoice
6. Send Feedback
7. Update Own Profile
8. Exit
Enter your choice: 5
1. python advanced
Enter your choice: 1
+--------------------------+----------------+-------------+
|        Module Name       |  Student Name  | Outstanding |
+--------------------------+----------------+-------------+
|      python advanced     |      dodi      |      0      |
+--------------------------+----------------+-------------+
```

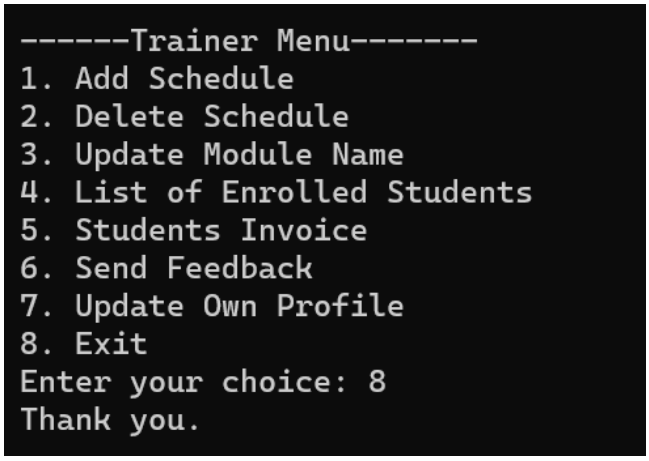
On this page, if user input number 5, he will be directed to see the student’s invoice option. same like the other options. It displays a module list and asks the user to prompt the available modules. After prompting the correct number, it will display the information regarding student’s invoice, that consist of module name, student name, and student’s current outstanding.

```
------Trainer Menu-------
1. Add Schedule
2. Delete Schedule
3. Update Module Name
4. List of Enrolled Students
5. Students Invoice
6. Send Feedback
7. Update Own Profile
8. Exit
Enter your choice: 6
Choose the feedback type:
1. suggestion
2. complaint

Enter choice: 1

Enter the message: The classroom should have bigger screen so all of the students could see
Feedback sent successfully. Thank you!
```

On this page, when user input number 6, it asks the user to select 2 options consisting of suggestion and complaint. After choosing which type of feedback, it directs the user to type a message, it could be about the class, the system, etc. Last, it displays a confirmation message that feedback is successfully sent.

```
------Trainer Menu-------
1. Add Schedule
2. Delete Schedule
3. Update Module Name
4. List of Enrolled Students
5. Students Invoice
6. Send Feedback
7. Update Own Profile
8. Exit
Enter your choice: 7
Enter trainer's new name: Jonathan
Enter new password: 456

Trainer Updated
Name: Jonathan
Password: 456
```

On this page, trainers can update their profile. First, they will be asked to enter their new name and password. After that, the system will print the updated trainer information based on the trainer’s input

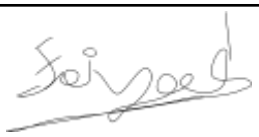


The last option in the Trainer Menu is Exit. In this option, the user can exit the system by selecting number 8, it will then appear a message saying “Thank you” and the user can log out of the system.

Conclusion

The code above is for the APU CAFE program system which helps students to learn coding languages easily. While making this project we faced many difficulties, by overcoming those difficulties we learned a lot and by making this project we also learned how to work in a team efficiently. We have tried our best to make a robust and functional project for the students. We are optimistic that we have fulfilled the requirements for the assignment. We would also like to show our gratitude to our lecturer for guiding us throughout this project.

Work Breakdown Structure

Name	Task / Description / Responsibility	Signature
Abdullah Al Faiyaz	Student part, assisting with lecturer part and trainer part, assumptions for students, conclusion, making of report format.	
Emmanuel Naranendhya Tyatan	Trainer part. Conclusion, Make the cover page, and create the table of contents, making the report format.	
Faiyad Mahabub Tasin	Lecturer part, Creating of Table Of Contents, Making of Report Format.	
Yeremia Samuel Aditya Marfianto	Administrator part, creating the main login menu, unifying the codebase, defining the data structures needed for the program, assisting with every part, assumptions for administrators, assumptions for trainers, making the report format.