COMP1811 – Python Project Report

Name	Chisel Chavush	Student ID	001131628
Partner's name	Baris Celiktutan	Partner SIDs	001120868
Partner's name	Emir Meneksheli	Partner SIDs	001084980

1. Brief statement of features you have completed

(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)
Indicate the feature each partner implemented by replacing "developed by" in red below with partner name.

1.1 Circle the parts of the coursework you have **fully completed and are fully working**. Please be accurate.	**Features** ***[Chisel Chavush/ Emir Meneksheli]*** **F1:** i ☒ ii ☒ iii ☒ ***[Baris Celiktutan/ Emir Meneksheli]*** **F2:** i ☐ ii ☒ iii ☒
1.2 Circle the parts of the coursework you have **partly completed or are partly working.**	**Features** **F1:** i ☐ ii ☐ iii ☐ **F2:** i ☐ ii ☐ iii ☐
Briefly explain your answer if you circled any parts in 1.2	

#In our project we all worked on each part together to learn every possibility while we are working on this coursework. Because all codes was connected to each other and to the database.

#Chisel Chavush worked on Feature 1 / Sub-feature i and Feature 2 / Sub-feature i. (2 sub-features)

#Emir Meneksheli worked on Feature 1 / Sub-feature ii and Feature 2 / Sub-feature ii. (2 sub-features)

#Baris Celiktutan worked on Feature 1 / Sub-feature iii and Feature 2 / Sub – feature iii (2 sub-features)

2. Concise List of Bugs and Weaknesses

*A concise list of bugs and/or weaknesses in your work (if you don't think there are any, then say so). Bugs that are declared in this list will lose you fewer marks than ones that you don't declare! (**100-200 word**, but word count depends heavily on the number of bugs and weaknesses identified.)*
(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)

2.1 Bugs

List each bug plus a brief description

**Enter Button:*

We have a bug when you click enter button to switch the frames, more then one times. At the beginning we were not be able to click enter button on keyboard, the program wasn't recognize this button. Then we add the; main.bind("<Return>", clicked) command to activate it. (Highlighted on screenshot)!

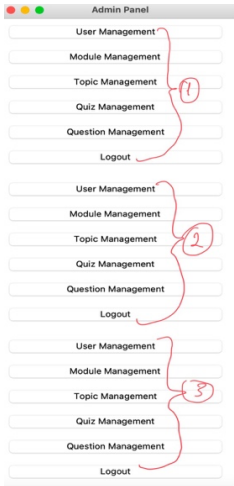
This issue continues even while student taking the quiz. (Everywhere actually when you click Enter Button!)

```
97        self.loginButton = Button(self.mainFrame, text='Login', width=33, command=check)
98        self.loginButton.place(x=10, y=130)
99        main.bind("<Return>", clicked)
```

Then we have another function to check this click!

```
37        # Creating a clicked element to clicked on any button or element
38        def clicked(event):
39            check()
```

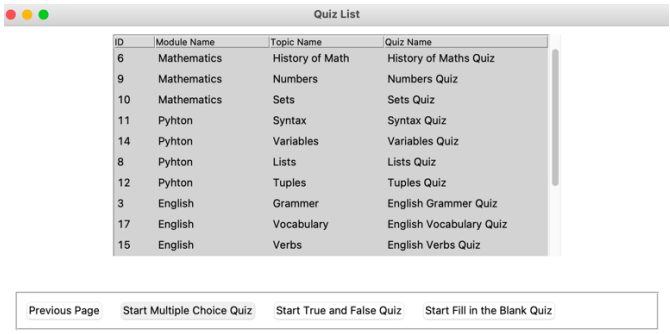
The Enter button activated, but then we realized on the management panel when you clicked 2 times or more on this button it was repeating it self as a loop. (Clicked Enter 3 times on the pic.)



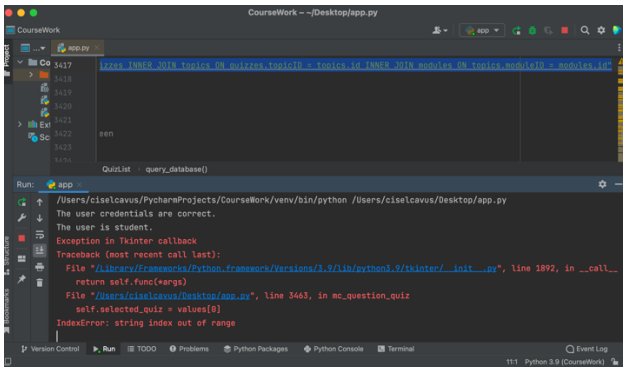
In the terminal we can see it is login in the same page while we are clicking on Enter Button.

*Selecting Quiz

If student try to get the quiz without choosing from quiz list it is causing an error on the terminal .



This is the error we are receiving on the terminal. We can fix that with using selection requirements on



2.2 Weaknesses

List each weakness plus a brief description

*We used lots of classes, to keep clear our codes but this was our weakness because we could use few main classes and calling them as a Child Class using inheritance. We wasn't clear about inheritance as well when we started to the coursework, but now we know how to improve and how to call other classes as (Parent/Child class).

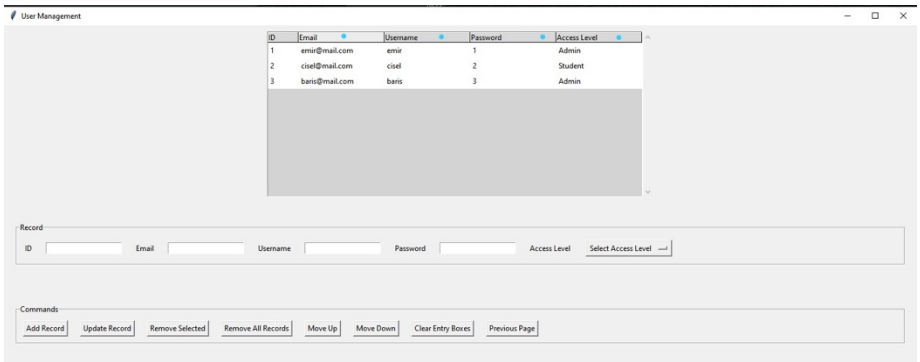
*We used less object, we could use more object to create more reusable codes.

*We connect database in most of the class and it makes our codes longer. We could create a Database class and call that class as a child class wherever we need.

*Process working slowly

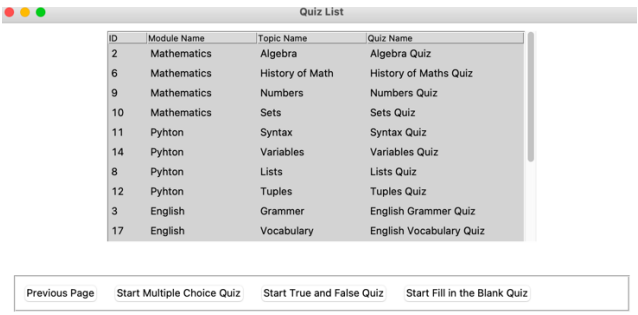
*We connected to the database almost in every class, so because of that it makes extra process and it cause the program work little bit slowly.

*Our data's not coming in alphabetic order. (This wasn't any requirements but it could avoid the complicated display.)



* In the my report panel, total score is including every question type, so student can not see which type of quiz he got in the reports.

* Student have to choose which type of question would like to see in the quiz. The program randomizing in selected types not all of them.



*We realized our codes running more faster on windows operating system. In IOS operating system it's a bit more slow.

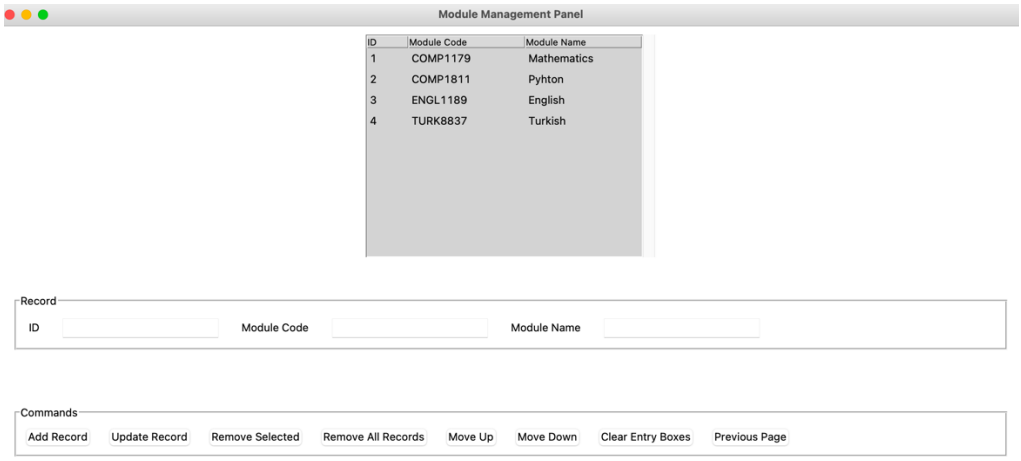
3. Description of the features implemented

Describe your implementation of the required features and how well do they work. Provide some exposition of the design decisions made and indicate how the features developed were integrated.
(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)

We have completed all the requirements as you can see below. But we have implemented few features which is we thought it can be useful for user.

****We created a login page which is directly connected with database. In the database we have the username/password and access level column, with that columns program checking the if user is admin or student.*

**** We add few different buttons as well which may be require if admin add extra quizzes, modules, topic or any type of questions.*



You can see the Remove all records button/Move Up/Move Down/Clear Entry boxes buttons. For example clear entry boxes button can delete all values at the same time form entry box so user will gain time instead of deleting one by one.

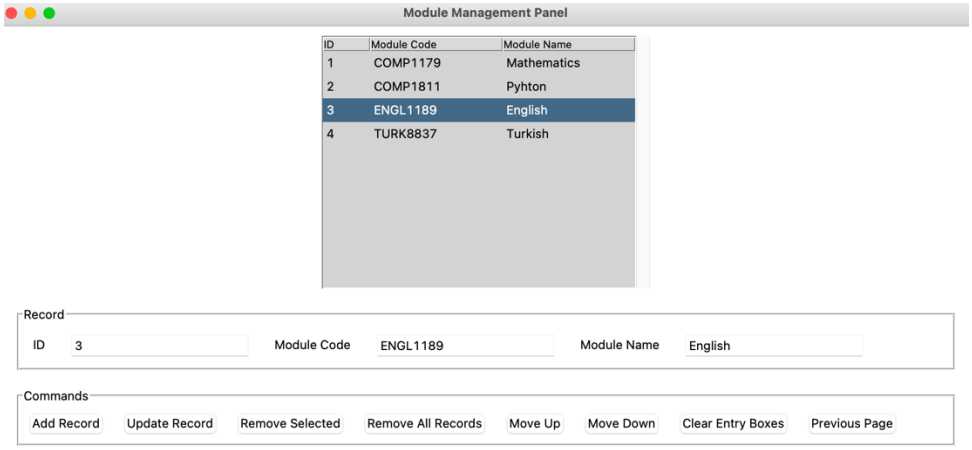
REQUIREMENTS Explanation

Feature 1:

*Sub-feature i: This picture is showing the database columns on database application. When user add/update/remove the records it is directly showing on database . We used the dynamic typing to be able to do update in our codes. ***

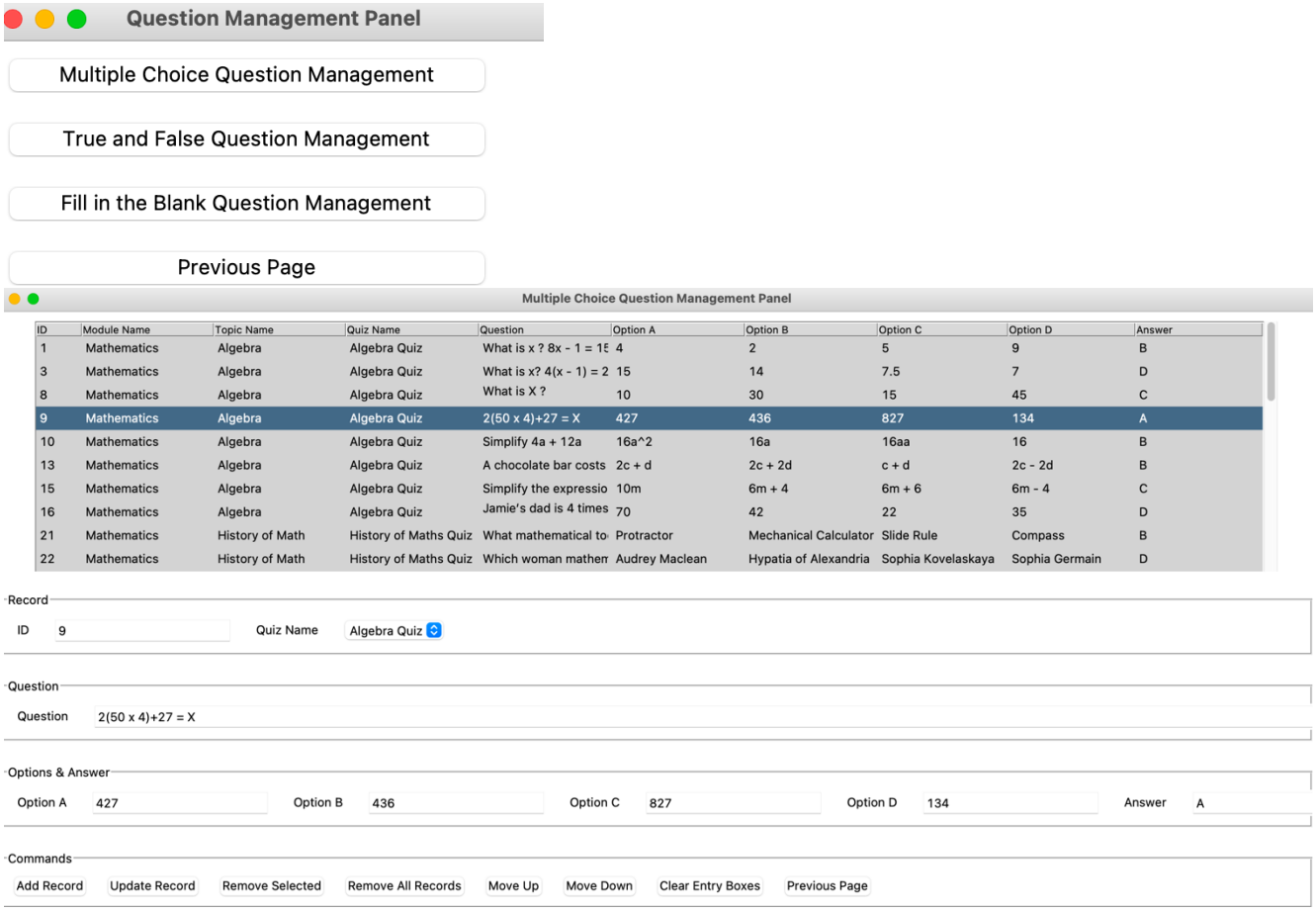
				id	moduleCode	moduleName
Edit	Copy	Delete		1	COMP1179	Mathematics
Edit	Copy	Delete		2	COMP1811	Pyhton
Edit	Copy	Delete		3	ENGL1189	English
Edit	Copy	Delete		4	TURK8837	Turkish

In the second picture which is Admin Interface, there is buttons to add/update/remove the Module ID, Module Code, Module Name. User is be able to do all of this functions on this panel. We wanted create a useful design to be clear to the users who can manage everything easily. Each button next to each other so it is easily to access. When user click on ant row from the table (dark blue selected) they can update the any information of the module or directly they can directly add a new module to the database using this interface. As user can add the module, user will be able to delete any module which is already exist on database. We have few other buttons for user convenience such as Remove all Records, so basically user can remove all the records directly. When user try to add the new module which is already exist on database, user is going to see warning message as this module already exist!



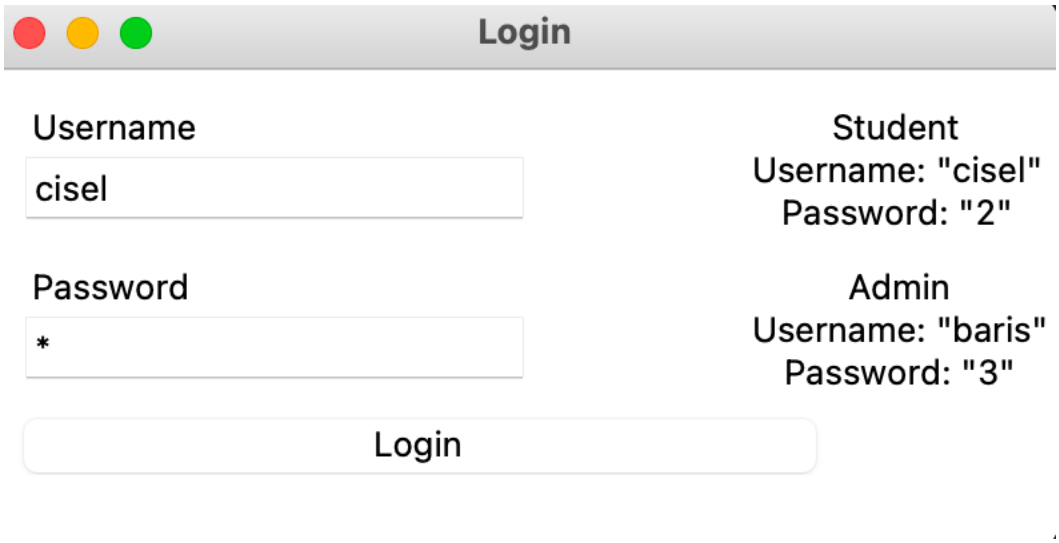
Sub-feature ii :

In this section we have a question management panel to manage any changes on the questions. At the same time user be able to add/ update/delete the question from the database. On the Question Management panel we have three kind of question type to work on. Once you click on any type of question, user will be able to add / update/ delete question on this frame. To do any change on the any specific question user can click on it and all the information will be shown on the panel, ten basically user can edit whatever like to.



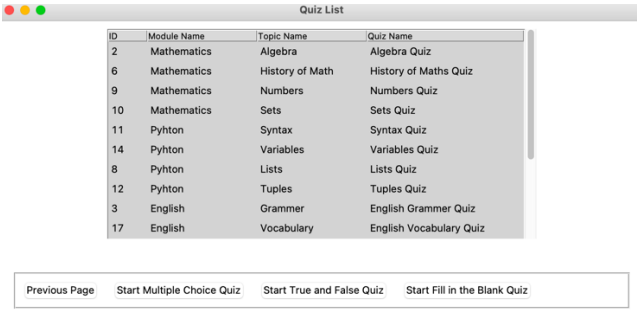
Sub-features iii

In that section we don't have selection menu, we want to be more realistic. In our database we created a table as "user". In our GUI we created a login page which is checking if user is student or admin. With this function if user is admin, frame switching to the admin panel, which is management page. If user is student, it is switching to the student page which is including my report, take quiz and logout buttons. Basically we separated users like this.

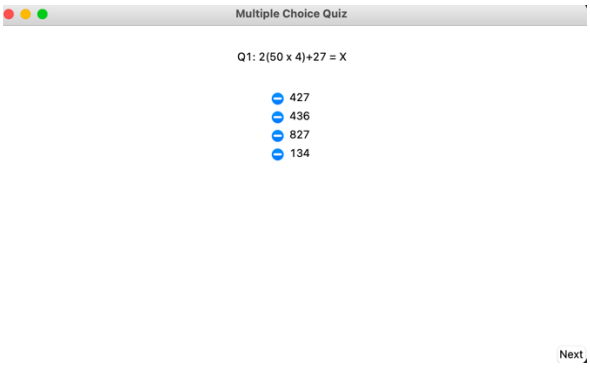


Feature 2:

Sub-feature i: In this section, our program displaying questions randomly, and possible answers. We have 3 different button to give the opportunity to the student to choose if student would like to get multiple choice, true/false or filling the blank questions. For all of this options, questions coming randomly from database and displaying possible answers. In each frame we are displaying only one question in the root and for next question student have to click on next button.

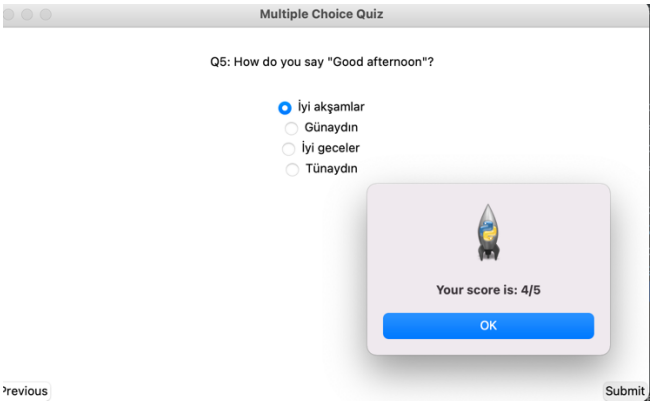


Student can choose any Quiz from the Quiz list, then user have to choose which type of quiz would like to get.

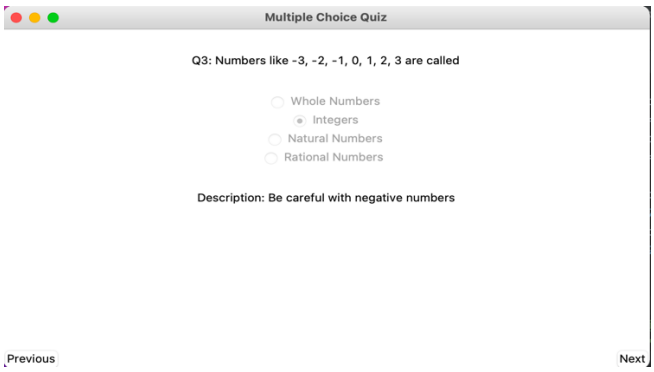


Let's say student wants to take multiple questions Quiz, this an example in ss. These questions coming randomly and displaying possible answers. Then student can choose any answer and click on next button.

Sub-feature ii:

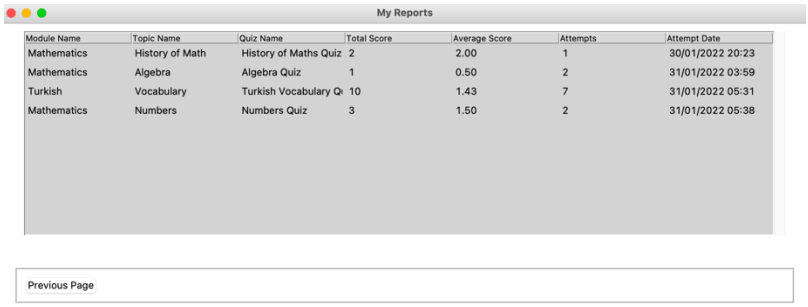


When students, finished the exam. Score is displaying on the screen as a message box.



When student finish the exam, user can go to previous page for each question and display the feedback(description, hint), why question was correct or wrong.

Sub-feature iii:

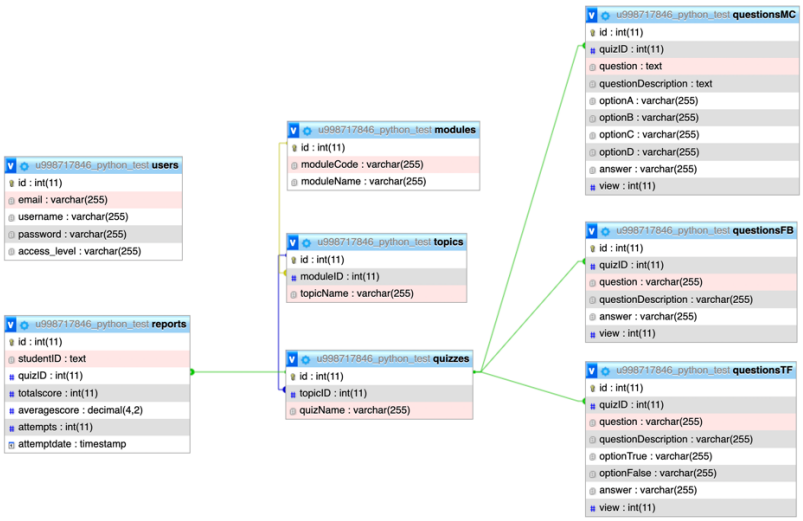


Module Name	Topic Name	Quiz Name	Total Score	Average Score	Attempts	Attempt Date
Mathematics	History of Math	History of Maths Quiz	2	2.00	1	30/01/2022 20:23
Mathematics	Algebra	Algebra Quiz	1	0.50	2	31/01/2022 03:59
Turkish	Vocabulary	Turkish Vocabulary Qu	10	1.43	7	31/01/2022 05:31
Mathematics	Numbers	Numbers Quiz	3	1.50	2	31/01/2022 05:38

Inside of the students frame, there is a "My Reports" button and its allow the users to display results about the quiz taken already. User storing all the data's on the database and inside of the "My Reports" table.

4.Classes and OOP Features

*List all the classes used in your program and include the attributes and behaviours for each. You may use a class diagram to illustrate these classes. Your narrative for section 3.2 should describe the design decisions you made and the OOP techniques used. Each partner must list the classes they developed separately and provide an exposition on the choice of classes, class design and OOP features implemented. (**200-400 words for each partner**). (THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*



3.1 Classes Used

*class main

```
def __init__(self, root):
```

*class AdminPanel

```
def __init__(self, root):
```

*class UserManagementPanel

```
def __init__(self, root):
```

* class ModuleManagementPanel

```
def __init__(self, root):
```

* class TopicManagementPanel

```
def __init__(self, root):
```

* class QuizManagementPanel

```
def __init__(self, root):
```

* class QuestionManagement

```
def __init__(self, root):
```

* class MCQuestionManagement

```
def __init__(self, root):
```

* class TFQuestionManagement

```
def __init__(self, root):
```

* class FBQuestionManagement

```
def __init__(self, root):
```

* class StudentPanel

```
def __init__(self, root):
```

* class MyReports

```
def __init__(self, root):
```

* class QuizList

```
def __init__(self, root):
```

* class MCQuiz(Quizlist)

```
def __init__(self, root, selected_quiz):
```

*class TFQuiz(Quizlist)

```
def __init__(self, root, selected_quiz):
```

*class FBQuiz(Quizlist)

```
def __init__(self, root, selected_quiz):
```

3.2 Brief Explanation of Class Design and OOP Features Used

Object-oriented programming (OOP) is a programming paradigm that allows us to package together data states and functionality to modify those data states, while keeping the details hidden away (like with the lightbulb).
We have classes to create and manage new objects and support with inheritance. A key ingredient in object-oriented programming and a mechanism of reusing code.

In our codes as an OOP Features we try to use Abstraction, Inheritance, and at the same time we used the encapsulation as get and set methods.

In our first class which is main class we created the root (window) and we call the same root in our every other class. All of our buttons, entry boxes and table's created as a dynamic, as needs to be in OOP. Everything is connected to the database. Whatever user(admin/student) enters on the root directly it will connect and checked by database. Everything was carefully thought through and tried to be implemented. In our last class we have the counting function to understand the user success (How many question is true or false) and to understand the attempts.

1) class Main:

Main class is creating the main root which is including the login page as well. In this class we gave the entry boxes as username and password and these entries connected to the database to understand if user is student or admin. So basically we created dynamic frames and dynamic buttons for the OOP in that class, and we have the click function to click on the button to switch the frames.

2) class AdminPanel:

In admin panel we are using the same root but different frame. We are only switching the frames on the same root. In this class we gave the management frames. As an OOP methods and functions if user click on any management panel its destroying available frames and switching to the new one.

3) class UserManagementPanel:

In that class we have dynamic panel because we are connecting to the database to manage the users. In that class we have lots of OOP functions such as add/update/delete buttons. All of those buttons connected to the

database and its updating all the data's in the moment. (Details of the add record functions as an example,all the other buttons are similar.)

```
# Adding record tested by Chisel
# Add new record to database
def add_record():

    # If there is no value at the entry boxes it will show the
messagebox.showinfo()
    if len(email_entry.get()) == 0 or len(username_entry.get()) == 0 or len(
            password_entry.get()) == 0 or access_level.get() == "Select Access
Level":
        messagebox.showinfo("Warning!", "Please enter all fields.")
        query_database()
        return

    # Connecting to the database
    conn = connect(
        host="mysql.hastivo.com",
        user="wedpswfy",
        password="gMR&m5ZQRCdp",
        database="question_bank"
    )

    # Create a cursor instance
    c = conn.cursor()

    select_query = "SELECT * FROM users WHERE email=%s OR username=%s"
    user_values = (email_entry.get(), username_entry.get())

    # Execute query
    c.execute(select_query, user_values)

    # Fetch user
    user = c.fetchall()

    # Check if the email and username of the entered user exist.
    if user:
        # Existing user
        print('Existing user!')

        messagebox.showinfo("Warning!", "The user is exist.")

        # Clear The Entry Boxes
        clear_entries()

    else:
        # Not existing user
        print('New user.')

        # SQL query to add users to the "user" table in the database.
        insert_user = "INSERT INTO users (email, username, password, access_level)
VALUES (%s, %s, %s, %s)"
        user_credentials = (email_entry.get(), username_entry.get(),
password_entry.get(), access_level.get())

        # Execute query
        c.execute(insert_user, user_credentials)

        # Clear The Entry Boxes
        clear_entries()

    # Commit changes
    conn.commit()
```

```
    # Close connection
    conn.close()

    # Clear The Treeview Table
    my_tree.delete(*my_tree.get_children())

    # Run to pull data from database on start
    query_database()
```

4) class ModuleManagementPanel:

In this class we are connecting to the Module's table from the database. As a requirement we have to be able to adding, updating, and deleting the modules. We are doing all of these requirements in that class. This class using the same root as other classes. Again we have a query to get data's from database.(Please see below as an example!)

```
query = "SELECT * FROM modules"
```

This query calling the module table from the database and displaying data's in dynamic window.

5) class TopicManagementPanel:

Same as ModuleManagementPanel, in that class we are managing the Topics for the modules. In our database we have Module table and we are creating a connection between Module and Topic table. We are using INNER JOIN statement to connect.

```
query = "SELECT * FROM topics INNER JOIN modules ON topics.moduleID = modules.id"
```

6) class QuizManagementPanel:

In that class admin can add/edit/delete any Quiz but if admin would like to add new quiz, has to choose from available topic. There is a selection list to choose it.

7) class QuestionManagement:

User have to choose which kind of question type would like to Update. In that class we have only dynamic buttons to switching frames for question type options. When user choose question type its destroying existing frame.

8) class MCQuestionManagement:

User be able to add/edit/delete any multiple choise question in that class. We connected to the database again to exchange the data's between root and database.

9) class TFQuestionManagement:

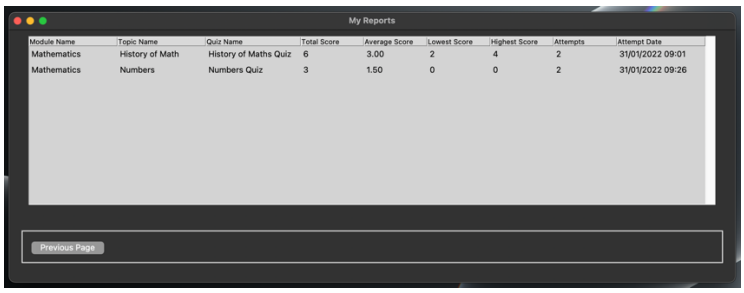
Same logic in that class as well user can do any changes on True/False Questions.

10) class FBQuestionManagement:

Same as True/False question. Only student have entry boxes to add answer.

11) Class MyReports:

In this we are keeping all the record. Actually we are receiving all of those record from database. They are working simultaneously.



12) Class QuizList:

In that class we gave the available Quizzes to student to choose it. The query is;

```
query = "SELECT * FROM quizzes INNER JOIN topics ON quizzes.topicID = topics.id
INNER JOIN modules ON topics.moduleID = modules.id"
```

In the GUI student can see all the available options.

13) Class MCQuiz(QuizList):
14) Class TFQuiz(QuizList):
15) class FBQuiz(QuizList):

In this three class we used inheritance as a SuperClass. These classes is which are we are displaying the questions and Answers on the root. It is connected with Quizlist class and with database. When student finished the exam ,can go back to read the description why that question was correct or wrong.

5.Code for the Classes Created

*Add the **code for each of the classes you have implemented yourself** here. If you have contributed to parts of classes, please highlight those parts in a different colour. Copy and paste relevant code - actual code please, no screenshots! Make it easy for the tutor to read. Add explanation if necessary – though your in-code comments should be clear enough. You will lose marks if screenshots are provided instead of code.*
(COMPLETE THIS SECTION INDIVIDUALLY – only list the code for the classes you developed individually. DO NOT provide a listing of the entire code. You will be marked down if a full code listing is provided.)

4.1 Class AdminPanel:

```
# Starting to crate Admin Panel class and design of the frames.
class AdminPanel:

    def __init__(self, root):
        admin_panel_frame = Frame(root)
        admin_panel_frame.pack(fill='both', expand=1)

        root.title("Admin Panel")
        root.geometry('355x250')


        #managment panels will be tested(black box)-Baris
        # If user going to click in user management button admin_panel_frame will
destroy and UserManagmentPanel will be open on the window. Same for all
definitions below
        def user_management():
            admin_panel_frame.destroy()
            UserManagementPanel(root)

        def module_management_panel():
            admin_panel_frame.destroy()
            ModuleManagementPanel(root)

        def topic_management_panel():
            admin_panel_frame.destroy()
            TopicManagementPanel(root)

        def quiz_management_panel():
            admin_panel_frame.destroy()
            QuizManagementPanel(root)

        def question_management_panel():
            admin_panel_frame.destroy()
            QuestionManagement(root)

        #logout function will be test(black box)-Baris
        def logout():
            are_you_sure = messagebox.askyesno("Alert", "Are you sure you want to
logout?")

            if are_you_sure:
                admin_panel_frame.destroy()
                main(root)

        # Creating the buttons for the management frames
        Button(admin_panel_frame, text='User Management', width=33,
command=user_management).place(x=10, y=10)
        Button(admin_panel_frame, text='Module Management', width=33,
command=module_management_panel).place(x=10, y=50)
```

```
        Button(admin_panel_frame, text='Topic Management', width=33,
command=topic_management_panel).place(x=10, y=90)
        Button(admin_panel_frame, text='Quiz Management', width=33,
command=quiz_management_panel).place(x=10, y=130)
        Button(admin_panel_frame, text='Question Management',
command=question_management_panel, width=33).place(x=10, y=170)
        Button(admin_panel_frame, text='Logout', command=logout,
width=33).place(x=10, y=210)
```

4.2 Class TopicManagementPanel

```
4.3     class TopicManagementPanel:

       def __init__(self, root):
           topic_management_panel_frame = Frame(root)
           topic_management_panel_frame.pack(fill='both', expand=1)

           root.title("Topic Management Panel")
           root.geometry('1230x550')

           def query_database():
               # Clear the Treeview
               for record in my_tree.get_children():
                   my_tree.delete(record)

               conn = connect(
                   host="sql582.main-hosting.eu",
                   user="u998717846_test_user",
                   password="7$Mw9Q=e",
                   database="u998717846_python_test"
               )

               c = conn.cursor()
               # print(conn)

               query = "SELECT * FROM topics INNER JOIN modules ON topics.moduleID
   = modules.id"

               c.execute(query)
               records = c.fetchall()

               # Add our data to the screen
               global count
               count = 0

               for record in records:
                   if count % 2 == 0:
                       my_tree.insert(
                           parent="",
                           index="end",
                           iid=count,
                           text="",
                           values=(record[0], record[5], record[2]),
                           tags=("evenrow",)
                       )
                   else:
```

```
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[5], record[2]),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(topic_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("ID", "Module Name", "Topic Name")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Module Name", anchor=W, width=140)
        my_tree.column("Topic Name", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
        my_tree.heading("Module Name", text="Module Name", anchor=W)
        my_tree.heading("Topic Name", text="Topic Name", anchor=W)
```

```
        # Add Record Entry Boxes
        data_frame = LabelFrame(topic_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)

        # MODULE MENU START #

        conn = connect(
            host="sql582.main-hosting.eu",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        query = "SELECT * FROM modules"

        c = conn.cursor()
        # print(conn)

        c.execute(query)
        records = c.fetchall()

        # datatype of menu text
        module_menu = StringVar()

        module_list = dict()
        for record in records:
            module_list[record[0]] = record[2]

        module_name_label = Label(data_frame, text="Module Name")
        module_name_label.grid(row=0, column=2, padx=10, pady=10)
        module_name_menu = OptionMenu(data_frame, module_menu,
*module_list.values())
        module_menu.set("Select Module")
        module_name_menu.grid(row=0, column=3, padx=10, pady=10)

        # MODULE MENU END #

        topic_name_label = Label(data_frame, text="Topic Name")
        topic_name_label.grid(row=0, column=4, padx=10, pady=10)
        topic_name_entry = Entry(data_frame)
        topic_name_entry.grid(row=0, column=5, padx=10, pady=10)

        # Move Row Up
        def up():
            rows = my_tree.selection()
            for row in rows:
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

        # Move Row Down
        def down():
            rows = my_tree.selection()
            for row in reversed(rows):
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

        # Clear entry boxes
        def clear_entries():
            # Clear entry boxes
            id_entry.delete(0, END)
            module_menu.set("Select Module")
```

```
            topic_name_entry.delete(0, END)

        # Select Record
        def select_record(e):
            # Clear entry boxes
            id_entry.delete(0, END)
            module_menu.set("Select Module")

            topic_name_entry.delete(0, END)

            # Grab record number
            selected = my_tree.focus()

            # Grab record values
            values = my_tree.item(selected, 'values')

            # Output to entry boxes
            id_entry.insert(0, values[0])
            module_menu.set(values[1])
            topic_name_entry.insert(0, values[2])

        # Remove one record
        def remove_one():
            x = my_tree.selection()[0]
            my_tree.delete(x)

            conn = connect(
                host="sql582.main-hosting.eu",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM topics WHERE id=" + id_entry.get())

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                conn = connect(
                    host="sql582.main-hosting.eu",
```

```
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute("TRUNCATE TABLE topics")

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if module_menu.get() == "Select Module" or
len(topic_name_entry.get()) == 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            conn = connect(
                host="sql582.main-hosting.eu",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            query = "SELECT * FROM modules WHERE
moduleName='{}'".format(module_menu.get())
            c.execute(query)
            result = c.fetchone()

            query2 = "INSERT INTO topics (moduleID, topicName) VALUES (%s, %s)"
            value = (result[0], topic_name_entry.get())
            c.execute(query2, value)

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()
```

```
            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if module_menu.get() == "Select Module" or
len(topic_name_entry.get()) == 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            conn = connect(
                host="sql582.main-hosting.eu",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            fetch_modules = "SELECT * FROM modules WHERE
moduleName='{}'".format(module_menu.get())
            c.execute(fetch_modules)
            module_result = c.fetchone()

            update_topic = "UPDATE topics SET moduleID = %s, topicName = %s
WHERE id = %s"
            value = (module_result[0], topic_name_entry.get(), id_entry.get())
            c.execute(update_topic, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Update record
            my_tree.item(selected, text="", values=(
                module_result[0], topic_name_entry.get()))

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(topic_management_panel_frame,
text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)
```

```
        add_button = Button(button_frame, text="Add Record",
command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)

        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)

        def previous_page():
            topic_management_panel_frame.destroy()
            AdminPanel(root)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()
```

4.4 Class QuestionManagement

```
4.5     class QuestionManagement:

      def __init__(self, root):
        question_management_panel_frame = Frame(root)
        question_management_panel_frame.pack(fill='both', expand=1)

        root.title("Question Management Panel")
        root.geometry('355x170')

        def mc_question_management_panel():
            question_management_panel_frame.destroy()
            MCQuestionManagement(root)

        def tf_question_management_panel():
            question_management_panel_frame.destroy()
            TFQuestionManagement(root)

        def fb_question_management_panel():
            question_management_panel_frame.destroy()
            FBQuestionManagement(root)
```

```
        def previous_page():
            question_management_panel_frame.destroy()
            AdminPanel(root)

        Button(question_management_panel_frame, text='Multiple Choice Question
Management',
               command=mc_question_management_panel, width=33).place(x=10,
y=10)
        Button(question_management_panel_frame, text='True and False Question
Management', width=33,
               command=tf_question_management_panel).place(x=10, y=50)
        Button(question_management_panel_frame, text='Fill in the Blank
Question Management', width=33,
               command=fb_question_management_panel).place(x=10, y=90)
        Button(question_management_panel_frame, text='Previous Page',
command=previous_page, width=33).place(x=10,

y=130)
```

4.5 Class MCQuiz(QuizList)

```
lass MCQuiz(QuizList):

    def __init__(self, root, selected_quiz):
        self.selected_quiz = selected_quiz
        self.questions = []
        self.options = []

        self.mc_question_page_1 = Frame(root, width=33)
        self.mc_question_page_2 = Frame(root, width=33)
        self.mc_question_page_3 = Frame(root, width=33)
        self.mc_question_page_4 = Frame(root, width=33)
        self.mc_question_page_5 = Frame(root, width=33)
        self.mc_question_page_1.pack(fill='both', expand=1)

        root.title("Multiple Choice Quiz")
        root.geometry('700x400')

        super().__init__(root)

        # Normal version
        # print(self.get_tf_questions()[0][2])

        # Random Shuffle
        random.shuffle(self.get_mc_questions())
        random.shuffle(self.get_options(self.get_mc_questions()[0][0]))

        self.q1_value = StringVar()
        self.q2_value = StringVar()
        self.q3_value = StringVar()
        self.q4_value = StringVar()
        self.q5_value = StringVar()

        self.q1_value.set(0)
        self.q2_value.set(0)
        self.q3_value.set(0)
        self.q4_value.set(0)
```

```
        self.q5_value.set(0)

        # Question Labels
        self.question1_label = Label(self.mc_question_page_1, text='Q1: ' +
self.get_mc_questions()[0][2], wraplength=650)
        self.question2_label = Label(self.mc_question_page_2, text='Q2: ' +
self.get_mc_questions()[1][2], wraplength=650)
        self.question3_label = Label(self.mc_question_page_3, text='Q3: ' +
self.get_mc_questions()[2][2], wraplength=650)
        self.question4_label = Label(self.mc_question_page_4, text='Q4: ' +
self.get_mc_questions()[3][2], wraplength=650)
        self.question5_label = Label(self.mc_question_page_5, text='Q5: ' +
self.get_mc_questions()[4][2], wraplength=650)
        self.question1_label.pack(fill='x', ipady=25)

        # Question Description Labels
        self.q1_description_label = Label(self.mc_question_page_1,
text='Description: ' + self.get_mc_questions()[0][3], wraplength=650)
        self.q2_description_label = Label(self.mc_question_page_2,
text='Description: ' + self.get_mc_questions()[1][3], wraplength=650)
        self.q3_description_label = Label(self.mc_question_page_3,
text='Description: ' + self.get_mc_questions()[2][3], wraplength=650)
        self.q4_description_label = Label(self.mc_question_page_4,
text='Description: ' + self.get_mc_questions()[3][3], wraplength=650)
        self.q5_description_label = Label(self.mc_question_page_5,
text='Description: ' + self.get_mc_questions()[4][3], wraplength=650)

        # Next Buttons
        self.next_button_1 = Button(self.mc_question_page_1, text='Next',
command=partial(self.change_frame, self.mc_question_page_1,
self.mc_question_page_2))
        self.next_button_2 = Button(self.mc_question_page_2, text='Next',
command=partial(self.change_frame, self.mc_question_page_2,
self.mc_question_page_3))
        self.next_button_3 = Button(self.mc_question_page_3, text='Next',
command=partial(self.change_frame, self.mc_question_page_3,
self.mc_question_page_4))
        self.next_button_4 = Button(self.mc_question_page_4, text='Next',
command=partial(self.change_frame, self.mc_question_page_4,
self.mc_question_page_5))
        self.submit_button = Button(self.mc_question_page_5, text='Submit',
command=self.submit)
        self.next_button_1.place(rely=1, relx=1, anchor="se")

        # Question 1
        self.q1_radio_list = []
        for q1_options in self.get_options(self.get_mc_questions()[0][0])[0]:
            self.q1_radio_button = Radiobutton(self.mc_question_page_1,
text=q1_options, value=q1_options,
                                               variable=self.q1_value)

            self.q1_radio_list.append(self.q1_radio_button)
            self.q1_radio_button.pack()

        # Question 2
        self.question2_label.pack(fill='x', ipady=25)

        self.q2_radio_list = []
        for q2_options in self.get_options(self.get_mc_questions()[1][0])[0]:
            self.q2_radio_button = Radiobutton(self.mc_question_page_2,
text=q2_options, value=q2_options,
                                               variable=self.q2_value)

            self.q2_radio_list.append(self.q2_radio_button)
```

```
            self.q2_radio_button.pack()

        self.next_button_2.place(rely=1, relx=1, anchor="se")

        # Question 3
        self.question3_label.pack(fill='x', ipady=25)

        self.q3_radio_list = []
        for q3_options in self.get_options(self.get_mc_questions()[2][0])[0]:
            self.q3_radio_button = Radiobutton(self.mc_question_page_3,
text=q3_options, value=q3_options,
                                                variable=self.q3_value)

            self.q3_radio_list.append(self.q3_radio_button)
            self.q3_radio_button.pack()

        self.next_button_3.place(rely=1, relx=1, anchor="se")

        # Question 4
        self.question4_label.pack(fill='x', ipady=25)

        self.q4_radio_list = []
        for q4_options in self.get_options(self.get_mc_questions()[3][0])[0]:
            self.q4_radio_button = Radiobutton(self.mc_question_page_4,
text=q4_options, value=q4_options,
                                                variable=self.q4_value)

            self.q4_radio_list.append(self.q4_radio_button)
            self.q4_radio_button.pack()

        self.next_button_4.place(rely=1, relx=1, anchor="se")

        # Question 5
        self.question5_label.pack(fill='x', ipady=25)

        self.q5_radio_list = []
        for q5_options in self.get_options(self.get_mc_questions()[4][0])[0]:
            self.q5_radio_button = Radiobutton(self.mc_question_page_5,
text=q5_options, value=q5_options,
                                                variable=self.q5_value)

            self.q5_radio_list.append(self.q5_radio_button)
            self.q5_radio_button.pack()

        self.submit_button.place(rely=1, relx=1, anchor="se")
        self.exit_button = Button(self.mc_question_page_5, text='Exit',
command=self.exit_quiz)

        # Previous Buttons
        self.previous_button_1 = Button(self.mc_question_page_2, text='Previous',
command=partial(self.change_frame, self.mc_question_page_2,
self.mc_question_page_1))
        self.previous_button_2 = Button(self.mc_question_page_3, text='Previous',
command=partial(self.change_frame, self.mc_question_page_3,
self.mc_question_page_2))
        self.previous_button_3 = Button(self.mc_question_page_4, text='Previous',
command=partial(self.change_frame, self.mc_question_page_4,
self.mc_question_page_3))
        self.previous_button_4 = Button(self.mc_question_page_5, text='Previous',
command=partial(self.change_frame, self.mc_question_page_5,
self.mc_question_page_4))
        self.previous_button_1.place(rely=1, relx=0, anchor="sw")
        self.previous_button_2.place(rely=1, relx=0, anchor="sw")
        self.previous_button_3.place(rely=1, relx=0, anchor="sw")
```

```
        self.previous_button_4.place(rely=1, relx=0, anchor="sw")

        self.correct = 0
        self.options = ['A', 'B', 'C', 'D']

    def change_frame(self, frame_to_forget, frame_to_pack):
        frame_to_forget.forget()
        frame_to_pack.pack(fill='both', expand=1)

    def submit(self):

        self.submit_button.destroy()
        self.exit_button.place(rely=1, relx=1, anchor="se")

        self.q1_description_label.pack(fill='x', ipady=25)
        self.q2_description_label.pack(fill='x', ipady=25)
        self.q3_description_label.pack(fill='x', ipady=25)
        self.q4_description_label.pack(fill='x', ipady=25)
        self.q5_description_label.pack(fill='x', ipady=25)

        for my_radio_list in [self.q1_radio_list, self.q2_radio_list,
self.q3_radio_list, self.q4_radio_list, self.q5_radio_list]:
            for radio_button in my_radio_list:
                radio_button.config(state=DISABLED)

        value_get_list = [
            self.q1_value.get(),
            self.q2_value.get(),
            self.q3_value.get(),
            self.q4_value.get(),
            self.q5_value.get()
        ]

        # DB Connection START

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()

        # DB Connection END

        for index in range(5):
            option_index = 4

            for option in self.options:

                # select_view = "SELECT view FROM questionsMC WHERE
id='{}'".format(self.get_mc_questions()[index][0])
                # c.execute(select_view)
                # select_view = c.fetchone()
                #
                # # Increase the attempt
                # view = select_view[0]
                # view += 1
                #
                # update_view = "UPDATE questionsMC SET view='{}' WHERE
id='{}'".format(view, self.get_mc_questions()[index][0])
                # c.execute(update_view)
                # conn.commit()
```

```
                if self.get_mc_questions()[index][8] == option:
                    if self.get_mc_questions()[index][option_index] ==
value_get_list[index]:
                        self.correct += 1
                option_index += 1

        # Select the report from the database
        select_report = "SELECT attempts, totalscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
            usernameEntry.get(), self.selected_quiz)
        c.execute(select_report)
        report = c.fetchone()

        score = self.correct

        if report:

            # Increase the attempt
            attempts = report[0]
            attempts += 1

            total_score = score + report[1]
            average_score = total_score / attempts

            # Update the attempt on the database
            update_report = "UPDATE reports SET totalscore='{}',
averagescore='{}', attempts ='{}' WHERE studentID='{}' AND quizID='{}'".format(
                total_score, average_score, attempts, usernameEntry.get(),
self.selected_quiz)
            c.execute(update_report)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the attempt on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        else:

            # Insert a report to the database
            insert_report = "INSERT INTO reports (studentID, quizID, totalscore,
averagescore, lowestscore, highestscore, attempts) VALUES (%s, %s, %s, %s, %s, %s,
%s)"
```

```
            report_value = (usernameEntry.get(), self.selected_quiz, score, score,
6, -1, 1)
            c.execute(insert_report, report_value)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the attempt on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        # Display correct answer
        print(f"Your score is: {score}/5")
        messagebox.showinfo("Result", f"Your score is: {score}/5")

    def get_mc_questions(self):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT * FROM questionsMC WHERE
quizID='{}'".format(self.selected_quiz)

        c.execute(query)

        my_list = c.fetchall()

        for question in my_list:
            self.questions.append(question)
        return self.questions

    def get_options(self, questionID):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
```

```
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT optionA, optionB, optionC, optionD FROM questionsMC WHERE
id='{}'".format(questionID)

        c.execute(query)

        my_list = c.fetchall()

        return my_list

    def exit_quiz(self):
        self.mc_question_page_1.destroy()
        self.mc_question_page_2.destroy()
        self.mc_question_page_3.destroy()
        self.mc_question_page_4.destroy()
        self.mc_question_page_5.destroy()
        StudentPanel(root)
```

4.5 Class TFQuiz(QuizList)

```
class TFQuiz(QuizList):

    def __init__(self, root, selected_quiz):
        self.selected_quiz = selected_quiz
        self.questions = []
        self.options = []

        self.tf_question_page_1 = Frame(root, width=33)
        self.tf_question_page_2 = Frame(root, width=33)
        self.tf_question_page_3 = Frame(root, width=33)
        self.tf_question_page_4 = Frame(root, width=33)
        self.tf_question_page_5 = Frame(root, width=33)
        self.tf_question_page_1.pack(fill='both', expand=1)

        root.title("True and False Quiz")
        root.geometry('700x400')

        super().__init__(root)

        # Normal version
        # print(self.get_tf_questions()[0][2])

        # Random Shuffle
        random.shuffle(self.get_tf_questions())
        random.shuffle(self.get_options(self.get_tf_questions()[0][0]))

        self.q1_value = StringVar()
        self.q2_value = StringVar()
        self.q3_value = StringVar()
        self.q4_value = StringVar()
        self.q5_value = StringVar()

        self.q1_value.set(0)
        self.q2_value.set(0)
        self.q3_value.set(0)
        self.q4_value.set(0)
```

```
        self.q5_value.set(0)

        # Question Labels
        self.question1_label = Label(self.tf_question_page_1, text='Q1: ' +
self.get_tf_questions()[0][2], wraplength=650)
        self.question2_label = Label(self.tf_question_page_2, text='Q2: ' +
self.get_tf_questions()[1][2], wraplength=650)
        self.question3_label = Label(self.tf_question_page_3, text='Q3: ' +
self.get_tf_questions()[2][2], wraplength=650)
        self.question4_label = Label(self.tf_question_page_4, text='Q4: ' +
self.get_tf_questions()[3][2], wraplength=650)
        self.question5_label = Label(self.tf_question_page_5, text='Q5: ' +
self.get_tf_questions()[4][2], wraplength=650)
        self.question1_label.pack(fill='x', ipady=25)

        # Question Description Labels
        self.q1_description_label = Label(self.tf_question_page_1,
text='Description: ' + self.get_tf_questions()[0][3], wraplength=650)
        self.q2_description_label = Label(self.tf_question_page_2,
text='Description: ' + self.get_tf_questions()[1][3], wraplength=650)
        self.q3_description_label = Label(self.tf_question_page_3,
text='Description: ' + self.get_tf_questions()[2][3], wraplength=650)
        self.q4_description_label = Label(self.tf_question_page_4,
text='Description: ' + self.get_tf_questions()[3][3], wraplength=650)
        self.q5_description_label = Label(self.tf_question_page_5,
text='Description: ' + self.get_tf_questions()[4][3], wraplength=650)

        # Next Buttons
        self.next_button_1 = Button(self.tf_question_page_1, text='Next',
command=partial(self.change_frame, self.tf_question_page_1,
self.tf_question_page_2))
        self.next_button_2 = Button(self.tf_question_page_2, text='Next',
command=partial(self.change_frame, self.tf_question_page_2,
self.tf_question_page_3))
        self.next_button_3 = Button(self.tf_question_page_3, text='Next',
command=partial(self.change_frame, self.tf_question_page_3,
self.tf_question_page_4))
        self.next_button_4 = Button(self.tf_question_page_4, text='Next',
command=partial(self.change_frame, self.tf_question_page_4,
self.tf_question_page_5))
        self.submit_button = Button(self.tf_question_page_5, text='Submit',
command=self.submit)
        self.next_button_1.place(rely=1, relx=1, anchor="se")

        # Question 1
        self.q1_radio_list = []
        for q1_options in self.get_options(self.get_tf_questions()[0][0])[0]:
            self.q1_radio_button = Radiobutton(self.tf_question_page_1,
text=q1_options, value=q1_options,
                                               variable=self.q1_value)

            self.q1_radio_list.append(self.q1_radio_button)
            self.q1_radio_button.pack()

        # Question 2
        self.question2_label.pack(fill='x', ipady=25)

        self.q2_radio_list = []
        for q2_options in self.get_options(self.get_tf_questions()[1][0])[0]:
            self.q2_radio_button = Radiobutton(self.tf_question_page_2,
text=q2_options, value=q2_options,
                                               variable=self.q2_value)

            self.q2_radio_list.append(self.q2_radio_button)
```

```
            self.q2_radio_button.pack()

        self.next_button_2.place(rely=1, relx=1, anchor="se")

        # Question 3
        self.question3_label.pack(fill='x', ipady=25)

        self.q3_radio_list = []
        for q3_options in self.get_options(self.get_tf_questions()[2][0])[0]:
            self.q3_radio_button = Radiobutton(self.tf_question_page_3,
text=q3_options, value=q3_options,
                                                variable=self.q3_value)

            self.q3_radio_list.append(self.q3_radio_button)
            self.q3_radio_button.pack()

        self.next_button_3.place(rely=1, relx=1, anchor="se")

        # Question 4
        self.question4_label.pack(fill='x', ipady=25)

        self.q4_radio_list = []
        for q4_options in self.get_options(self.get_tf_questions()[3][0])[0]:
            self.q4_radio_button = Radiobutton(self.tf_question_page_4,
text=q4_options, value=q4_options,
                                                variable=self.q4_value)

            self.q4_radio_list.append(self.q4_radio_button)
            self.q4_radio_button.pack()

        self.next_button_4.place(rely=1, relx=1, anchor="se")

        # Question 5
        self.question5_label.pack(fill='x', ipady=25)

        self.q5_radio_list = []
        for q5_options in self.get_options(self.get_tf_questions()[4][0])[0]:
            self.q5_radio_button = Radiobutton(self.tf_question_page_5,
text=q5_options, value=q5_options,
                                                variable=self.q5_value)

            self.q5_radio_list.append(self.q5_radio_button)
            self.q5_radio_button.pack()

        self.submit_button.place(rely=1, relx=1, anchor="se")
        self.exit_button = Button(self.tf_question_page_5, text='Exit',
command=self.exit_quiz)

        # Previous Buttons
        self.previous_button_1 = Button(self.tf_question_page_2, text='Previous',
command=partial(self.change_frame, self.tf_question_page_2,
self.tf_question_page_1))
        self.previous_button_2 = Button(self.tf_question_page_3, text='Previous',
command=partial(self.change_frame, self.tf_question_page_3,
self.tf_question_page_2))
        self.previous_button_3 = Button(self.tf_question_page_4, text='Previous',
command=partial(self.change_frame, self.tf_question_page_4,
self.tf_question_page_3))
        self.previous_button_4 = Button(self.tf_question_page_5, text='Previous',
command=partial(self.change_frame, self.tf_question_page_5,
self.tf_question_page_4))
        self.previous_button_1.place(rely=1, relx=0, anchor="sw")
        self.previous_button_2.place(rely=1, relx=0, anchor="sw")
        self.previous_button_3.place(rely=1, relx=0, anchor="sw")
```

```
        self.previous_button_4.place(rely=1, relx=0, anchor="sw")

        self.correct = 0
        self.options = ['True', 'False']

    def change_frame(self, frame_to_forget, frame_to_pack):
        frame_to_forget.forget()
        frame_to_pack.pack(fill='both', expand=1)

    def submit(self):

        self.submit_button.destroy()
        self.exit_button.place(rely=1, relx=1, anchor="se")

        self.q1_description_label.pack(fill='x', ipady=25)
        self.q2_description_label.pack(fill='x', ipady=25)
        self.q3_description_label.pack(fill='x', ipady=25)
        self.q4_description_label.pack(fill='x', ipady=25)
        self.q5_description_label.pack(fill='x', ipady=25)

        for my_radio_list in [self.q1_radio_list, self.q2_radio_list,
self.q3_radio_list, self.q4_radio_list, self.q5_radio_list]:
            for radio_button in my_radio_list:
                radio_button.config(state=DISABLED)

        value_get_list = [
            self.q1_value.get(),
            self.q2_value.get(),
            self.q3_value.get(),
            self.q4_value.get(),
            self.q5_value.get()
        ]

        # DB Connection START

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()

        # DB Connection END

        for index in range(5):
            option_index = 5

            for option in self.options:

                if self.get_tf_questions()[index][5] == option:
                    if self.get_tf_questions()[index][option_index] ==
value_get_list[index]:
                        self.correct += 1
                option_index += 1

        # Select the report from the database
        select_report = "SELECT attempts, totalscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
            usernameEntry.get(), self.selected_quiz)
        c.execute(select_report)
        report = c.fetchone()
```

```
        score = self.correct

        if report:

            # Increase the attempt
            attempts = report[0]
            attempts += 1

            total_score = score + report[1]
            average_score = total_score / attempts

            # Update the attempt on the database
            update_report = "UPDATE reports SET totalscore='{}',
averagescore='{}', attempts ='{}' WHERE studentID='{}' AND quizID='{}'".format(
                total_score, average_score, attempts, usernameEntry.get(),
self.selected_quiz)
            c.execute(update_report)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the attempt on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        else:

            # Insert a report to the database
            insert_report = "INSERT INTO reports (studentID, quizID, totalscore,
averagescore, lowestscore, highestscore, attempts) VALUES (%s, %s, %s, %s, %s, %s,
%s)"
            report_value = (usernameEntry.get(), self.selected_quiz, score, score,
6, -1, 1)
            c.execute(insert_report, report_value)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]
```

```
            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the attempt on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        # Display correct answer
        print(f"Your score is: {score}/5")
        messagebox.showinfo("Result", f"Your score is: {score}/5")

    def get_tf_questions(self):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT * FROM questionsTF WHERE
quizID='{}'".format(self.selected_quiz)

        c.execute(query)

        my_list = c.fetchall()

        for question in my_list:
            self.questions.append(question)
        return self.questions

    def get_options(self, questionID):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT optionTrue, optionFalse FROM questionsTF WHERE
id='{}'".format(questionID)

        c.execute(query)

        my_list = c.fetchall()

        return my_list
```

```
    def exit_quiz(self):
        self.tf_question_page_1.destroy()
        self.tf_question_page_2.destroy()
        self.tf_question_page_3.destroy()
        self.tf_question_page_4.destroy()
        self.tf_question_page_5.destroy()
        StudentPanel(root)
```

6.Testing

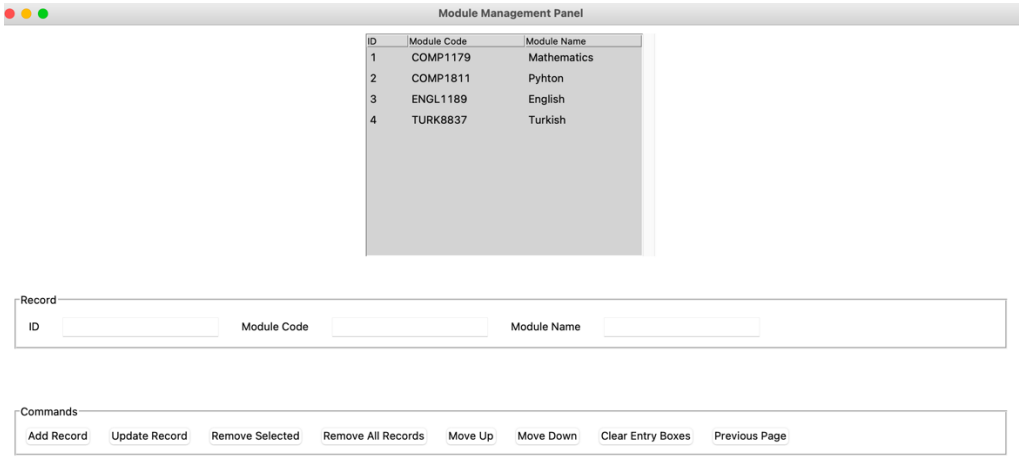
Describe the process you took to test your code and to make sure the program functions as required. Provide the detailed test plan used. Also, indicate the testing you did after integrating your code with your partner's. (COMPLETE THIS SECTION INDIVIDUALLY)

#CASE	Test Case (What is being tested?)	Actions	Inputs	Expected Outputs	Actual Output	Pass/Fail	Corrective Action
1	Logout	Pressed Logout	Clicked on Logout	Log out from root	Log out from root	Pass	-
2	Module Menu	Seeing the modules from the database	Connected to the database and clicked Modules	Printed module list	Printed module list	Pass	-
3	Question Management Panel	Clicked Question Management Panel	Clicking the Question Management Panel Button	Open the new Panel	Opened new panel	Pass	-
4	MC Quiz Panel	Viewing the Questions as Random on the Panel	Connecting the database, and calling the Parent class	Printed Questions as Random on the root	Printed Questions as Random on the root	Pass	-
5	MC Quiz Panel	Viewing the Answer as Random on the Panel	Connecting the database, and calling the Parent class	Printed answes listed with choosing buttons	Printed answes listed with choosing buttons	Pass	-

7. Annotated Screenshots Demonstrating Implementation

Provide screenshots that demonstrate the features implemented. Annotate each screenshot and if necessary, provide a brief description for **each** (**up to 100 words**) to explain the code in action. Make sure the screenshots make clear what you have implemented and achieved.
(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)

7.1 Feature F1 (*Chisel /Baris/Emir*)



This is the general view on the root how the buttons look like, user can add any modules directly from that root. If this module already exist on the database, it is going to give error messages as “This module already exist”. With this user can’t add the same module two times to the database.

Sub-feature I- screenshots

Adding Record:

```
def add_record():

    # If there is no value at the entry boxes it will show the
messagebox.showinfo()
    if len(module_code_entry.get()) == 0 or len(module_name_entry.get()) ==
0:
        messagebox.showinfo("Warning!", "Please enter all fields.")
        query_database()
        return

    query = "INSERT INTO modules (moduleCode, moduleName) VALUES (%s, %s)"
    value = (module_code_entry.get(), module_name_entry.get())

    conn = connect(
        host="sql582.main-hosting.eu",
        user="u998717846_test_user",
        password="7$Mw9Q=e",
        database="u998717846_python_test"
    )

    # Create a cursor instance
    c = conn.cursor()

    # Execute query
    c.execute(query, value)

    # Commit changes
    conn.commit()

    # Close connection
    conn.close()

    # Clear The Entry Boxes
    clear_entries()

    # Clear The Treeview Table
    my_tree.delete(*my_tree.get_children())

    # Run to pull data from database on start
    query_database()
```

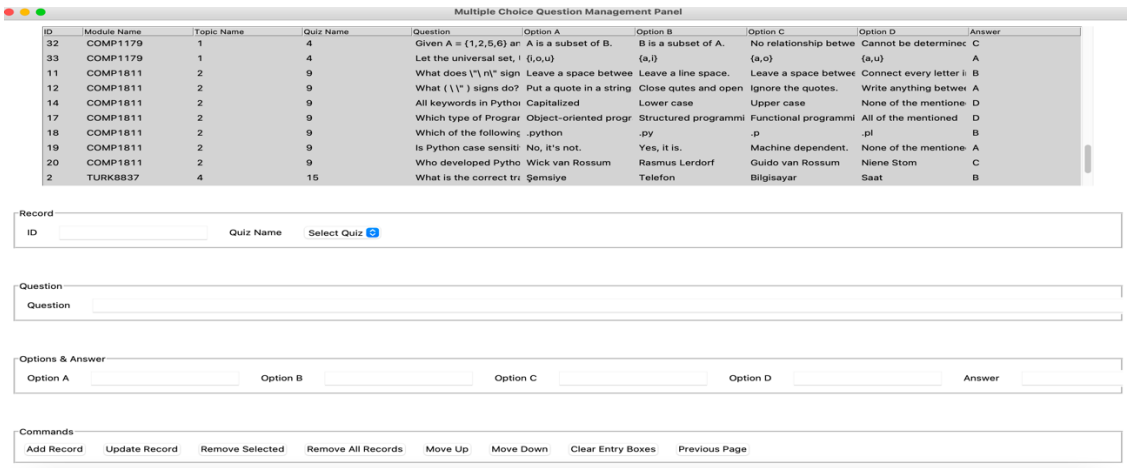
Updating Records

```
def update_record():

    # If there is no value at the entry boxes it will show the
messagebox.showinfo()
    if len(module_code_entry.get()) == 0 or len(module_name_entry.get()) ==
0:
        messagebox.showinfo("Warning!", "Please enter all fields.")
        query_database()
        return

    # Grab the record number
    selected = my_tree.focus()

    # Update record
    my_tree.item(selected, text="", values=(
        module_code_entry.get(), module_name_entry.get()))

    # If there is no value at the entry boxes it will show the
messagebox.showinfo()
    if len(module_code_entry.get()) == 0 or len(module_name_entry.get()) ==
0:
        messagebox.showinfo("Warning!", "Please enter all fields.")
        query_database()
        return

    query = "UPDATE modules SET moduleCode = %s, moduleName = %s WHERE id =
%s"
    value = (module_code_entry.get(), module_name_entry.get(),
id_entry.get())

    conn = connect(
        host="sql582.main-hosting.eu",
        user="u998717846_test_user",
        password="7$Mw9Q=e",
        database="u998717846_python_test"
    )

    # Create a cursor instance
    c = conn.cursor()

    # Execute query
    c.execute(query, value)

    # Commit changes
    conn.commit()

    # Close our connection
    conn.close()

    # Clear The Entry Boxes
    clear_entries()

    # Clear The Treeview Table
    my_tree.delete(*my_tree.get_children())

    # Run to pull data from database on start
    query_database()
```

Deleting a record:

```
def remove_one():
    x = my_tree.selection()[0]
    my_tree.delete(x)

    conn = connect(
        host="sql582.main-hosting.eu",
        user="u998717846_test_user",
        password="7$Mw9Q=e",
        database="u998717846_python_test"
    )

    # Create a cursor instance
    c = conn.cursor()

    # Execute query
    c.execute("DELETE FROM modules WHERE id=" + id_entry.get())

    # Commit changes
    conn.commit()

    # Close connection
    conn.close()

    # Clear The Entry Boxes
    clear_entries()

    # Informative message box
    messagebox.showinfo("Deleted!", "Successfully deleted.")
```

a. SUB-FEATURE II- SCREENSHOTS …



Same thing in this feature as well. User can add/edit or delete any question from the database.

Add Record:

```
def add_record():

    # If there is no value at the entry boxes it will show the
messagebox.showinfo()
    if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
            option_a_entry.get()) == 0 or len(option_b_entry.get()) == 0 or
len(
            option_c_entry.get()) == 0 or len(option_d_entry.get()) == 0 or
len(
            answer_entry.get()) == 0:
        messagebox.showinfo("Warning!", "Please enter all fields.")
        query_database()
        return

    conn = connect(
        host="sql582.main-hosting.eu",
        user="u998717846_test_user",
        password="7$Mw9Q=e",
        database="u998717846_python_test"
    )

    # Create a cursor instance
    c = conn.cursor()

    # Fetch selected quiz
    fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
    c.execute(fetch_quiz)
    quiz_result = c.fetchone()

    # Insert question
    insert_question = "INSERT INTO questionsMC (quizID, question, optionA,
optionB, optionC, optionD, answer) VALUES (%s, %s, %s, %s, %s, %s, %s)"
    value = (
    quiz_result[0], question_entry.get(), option_a_entry.get(),
option_b_entry.get(), option_c_entry.get(),
    option_d_entry.get(), answer_entry.get())
    c.execute(insert_question, value)

    # Commit changes
    conn.commit()

    # Close connection
    conn.close()

    # Clear The Entry Boxes
    clear_entries()

    # Clear The Treeview Table
    my_tree.delete(*my_tree.get_children())

    # Run to pull data from database on start
    query_database()
```

Updating Records:

```
# Update record
def update_record():

    # If there is no value at the entry boxes it will show the
messagebox.showinfo()
    if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
            option_a_entry.get()) == 0 or len(option_b_entry.get()) == 0 or
len(
            option_c_entry.get()) == 0 or len(option_d_entry.get()) == 0 or
len(
            answer_entry.get()) == 0:
        messagebox.showinfo("Warning!", "Please enter all fields.")
        query_database()
        return

    # Grab the record number
    selected = my_tree.focus()

    conn = connect(
        host="sql582.main-hosting.eu",
        user="u998717846_test_user",
        password="7$Mw9Q=e",
        database="u998717846_python_test"
    )

    # Create a cursor instance
    c = conn.cursor()

    # Fetch selected quiz
    fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
    c.execute(fetch_quiz)
    quiz_result = c.fetchone()

    # Update question
    updateQuestion = "UPDATE questionsMC SET quizID = %s, question = %s,
optionA = %s, optionB = %s, optionC = %s, optionD = %s, answer = %s WHERE
id = %s"
    value = (
    quiz_result[0], question_entry.get(), option_a_entry.get(),
option_b_entry.get(), option_c_entry.get(),
    option_d_entry.get(), answer_entry.get(), id_entry.get())
    c.execute(updateQuestion, value)

    # Commit changes
    conn.commit()

    # Close our connection
    conn.close()

    # Update record
    my_tree.item(selected, text="", values=(
        quiz_result[0], question_entry.get(), option_a_entry.get(),
option_b_entry.get(), option_c_entry.get(),
        option_d_entry.get(), answer_entry.get()))

    # Clear The Entry Boxes
    clear_entries()
```

Deleting Records:

```
# Remove one record
def remove_one():
    x = my_tree.selection()[0]
    my_tree.delete(x)

    conn = connect(
        host="sql582.main-hosting.eu",
        user="u998717846_test_user",
        password="7$Mw9Q=e",
        database="u998717846_python_test"
    )

    # Create a cursor instance
    c = conn.cursor()

    # Execute query
    c.execute("DELETE FROM questionsMC WHERE id=" + id_entry.get())

    # Commit changes
    conn.commit()

    # Close connection
    conn.close()

    # Clear The Entry Boxes
    clear_entries()

    # Informative message box
    messagebox.showinfo("Deleted!", "Successfully deleted.")
```

b. SUB-FEATURE III- SCREENSHOTS …

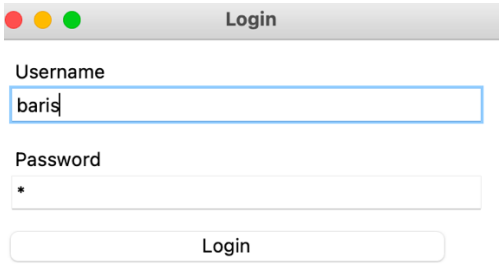
In this section, we have the login page. Which is checking the access level from the data base and understanding if its Student or Admin:

Student:

Username: cisel Password:2

Admin:

Username: baris Password:3

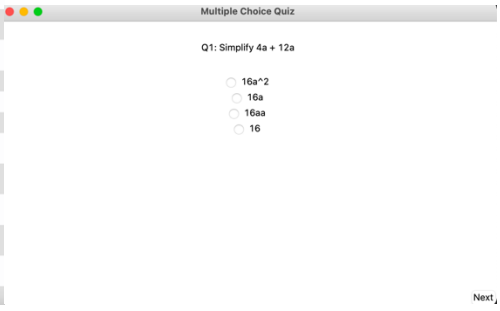


7.2 Feature F2 (*Chisel /Baris/Emir*)

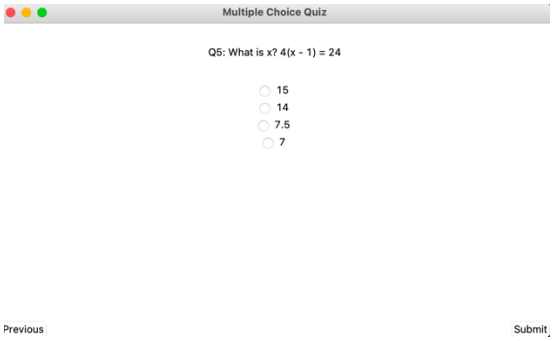
a. Sub-feature I- screenshots …

In our display we are taking 5 random question from database. First screenshot is from database. Next to that one is Quiz GUI. So all of the question including any other Question type coming randomly. (An example Multiple Choice quiz)

Edit Copy Delete	1	2	8x - 1 = 15 What is x ?	Answer of the question is 2. If 8x = 16, x = 2	4	2	5	9	B	8
Edit Copy Delete	2	16	What is the correct translation for "telephone"?		Şemsiye	Telefon	Bilgisayar	Saat	B	6
Edit Copy Delete	3	2	What is x? 4(x - 1) = 24	Answer of the question is 7. If 4x - 4 = 21, 4x is...	15	14	7.5	7	D	15
Edit Copy Delete	4	16	How do you say "Good morning"?		İyi akşamlar	Günaydın	İyi geceler	Tünaydın	B	6
Edit Copy Delete	5	16	How do you say "Good afternoon"?		İyi akşamlar	Günaydın	İyi geceler	Tünaydın	D	6
Edit Copy Delete	6	16	How do you say "Good night"?		İyi akşamlar	Tünaydın	İyi Geceler	Günaydın	C	6
Edit Copy Delete	7	16	What is the correct translation for "Please"?		Özür dilerim	Lütfen	Teşekkürler	Rica ederim	B	6
Edit Copy Delete	8	2	What is X ? (15 x 2) - 15 = X	Answer of the question is 15. We have 30 in the br...	10	30	15	45	C	13
Edit Copy Delete	9	2	2(50 x 4)+27 = X	Answer of the question is 427. We have 200 in brac...	427	436	827	134	A	13
Edit Copy Delete	10	2	Simplify 4a + 12a	Answer of the question is 16a. It's like adding ap...	16a^2	16a	16aa	16	B	14
Edit Copy Delete	11	11	What does \"\ n\" signs do?		Leave a space between letters of a string.	Leave a line space.	Leave a space between words in a string sentence.	Connect every letter in a string	B	0
Console					Put a quote in a	Close quotes		Write anything		

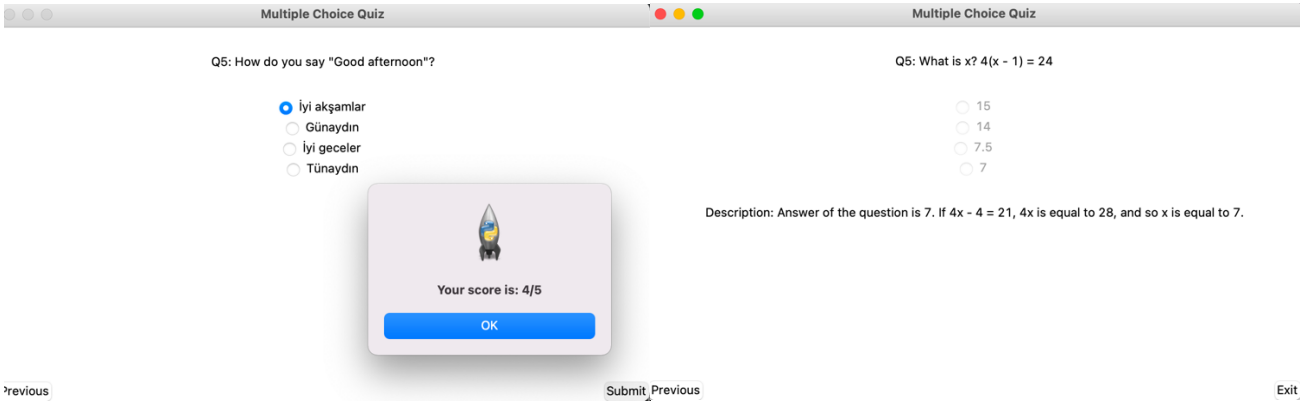


At the fifth question we have submit button. Once student clicked on that button, taken quiz will be saved on my reports page and at the same time on database. Once quiz completed after than clicked submit button, student can go back to previous page to see the description for each question.

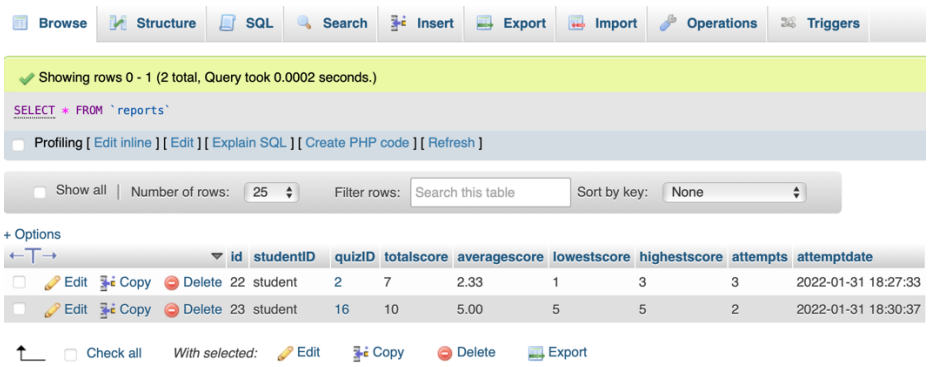


b. SUB-FEATURE II- SCREENSHOTS …

As I explained in the Features Implemented section, in this Sub-Feature student can see the result and description about the question.

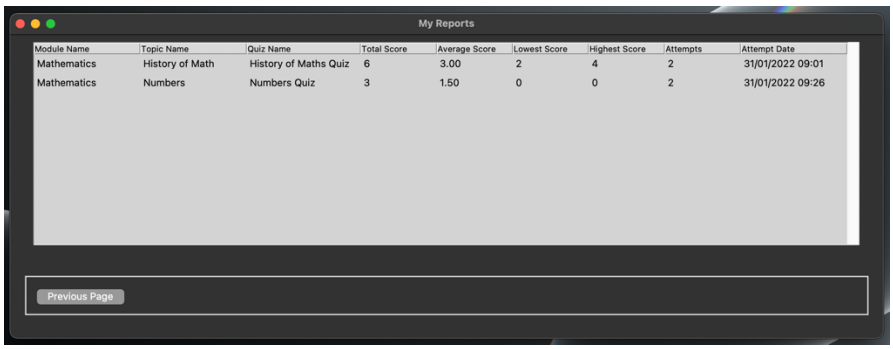


And all of taken quiz records saved on Report table, in our program. In our database we have a table named results. All of the stored data's coming from that table. In that table student can see Total Score, Average Score, Hihest/Lowest Score, Attempts and Date of the quiz.



c. SUB-FEATURE III- SCREENSHOTS …

In that section an achievements report that displays: the number of times a quiz was taken for that module, the average score achieved, and the lowest and highest score achieved;



8. Evaluation

Give a reflective, critical self-evaluation of your experience developing the project and discuss what you would do if you had more time to work on the project. Answer the following questions for the reflection and write ***350-400 words overall****. Please include an actual word count for this section.*
(COMPLETE THIS SECTION INDIVIDUALLY)

8.1 Evaluate how well your design and implementation meet the requirements

I think most of the design and implementation meet with the requirements. As an extra we used the login page to check the access level to understand if user is student or admin. Some of our codes not reusable we know that and we will try to improve it. We have the add/edit/delete function for all module,topic and questions. Even we have the same function for users as well.

From my view our design was well. We didn't want use colours because we thought we are doing an official page, we wanted to be more clear. Our buttons was well designed as well. All of the root is changing for each panel its depends on what is on the root.

8.2 Evaluate you own and your group's performance

Everybody in our group puts lots of effort for each part. We try to work all together for each Features and Sub-Features, it was a bit hard because everybody has something to do and it was difficult to arrange a time for working together, but somehow, we manage to do it. We stayed in the library many nights to complete the project and arranged meetings online for many day.

I can say I'm believing we worked well and I tried to do my best.

8.2.1 What went well?

Our design and most of our functions went well. I believe we learned to creating a GUI and frames really well.

8.2.2 What went less well?

From my side it was hard to use classes and objects as an inheritance, I wasn't that much comfortable while I was using it.

8.2.3 What was learnt?

One of our group member's was experienced about programming at the beginning it was hard to understand what was he doing but then he teach us how to do and implementing to our codes. We learned lots of things about OOP and the logic of the working with small piece of codes then put them all together. At the same time we learned to work as a group.

8.2.4 How would a similar task be completed differently?

I'm believing that task could be more easier and can complete with less code. But it was my first task with programming language and When I understand the using OOP (classes-methods) it was a bit late we have not use that much inheritance because at the beginning we thought everything will be separate from each other, but now I realized it could be more easier with classes and inheritance.

8.2.5 How could the module be improved?

The module was designed really well since first week. I can access whatever I would like to learn including office hours. Only thing maybe we can have small task as an exercises to understand more clearly. Because I completed the all the exercises but when I started to work on Coursework in some part I felt lost and I searched a lot from different sources. Except that everything was really well.

8.3 SELF-ASSESSMENT

Please assess yourself objectively for each section shown below and then enter the total mark you expect to get. Marks for each assessment criteria are indicated between parentheses.

CODE DEVELOPMENT (70)

Features Implemented [30]

Sub-feature i (up to 8)

Sub-features have not been implemented – 0
Attempted, not complete or very buggy – 1 or 2
Implemented and functioning without errors but not integrated – 3 or 4
Implemented and fully integrated but buggy – 5 or 6
Implemented, fully integrated and functioning without errors – 7 or 8

Sub-feature ii (up to 10)

Sub-features have not been implemented – 0
Attempted, not complete or very buggy – 1 or 2
Implemented and functioning without errors but not integrated – 3 to 5
Implemented and fully integrated but buggy – 6 to 8
Implemented, fully integrated and functioning without errors – 9 or 10

Sub-feature iii (up to 12)

Sub-features has not been implemented – 0
Attempted, not complete or very buggy – 1 to 3
Implemented and functioning without errors but not integrated – 4 to 6
Implemented and fully integrated but buggy – 7 to 9
Implemented, fully integrated and functioning without errors – 10 to 12

For this criterion I think I got: 30 out of 30

Use of OOP techniques [25]

Abstraction (up to 10)

No classes have been created – 0
Classes have been created superficially and not instantiated or used – 1 or 2
Classes have been created but only some have been instantiated and used – 3 or 4
Useful classes and objects have been created and used correctly – 5 to 7
The use of classes and objects exceeds the specification – 8 to 10

Encapsulation (up to 10)

No encapsulation has been used – 0
Class variables and methods have been encapsulated superficially – 1 to 3
Class variables and methods have been encapsulated correctly – 4 to 6
The use of encapsulation exceeds the specification – 7 to 10

Inheritance (up to 5)

No inheritance has been used – 0
Classes have been inherited superficially – 1
Classes have been inherited correctly – 2 to 4
The use of inheritance exceeds the specification – 5

Bonus marks will be awarded for the appropriate use of polymorphism (bonus marks up to 10)

For this criterion I think I got: 20 out of 25

Quality of Code [15]

Code Duplication (up to 8)

Code contains too many unnecessary code repetition – 0
Regular occurrences of duplicate code – 1 to 3
Occasional duplicate code – 4 to 5
Very little duplicate code – 6 to 7
No duplicate code – 8

PEP8 Conventions and naming of variables, methods and classes (up to 4) **

PEP8 and naming convention has not been used – 0
PEP8 and naming convention has been used occasionally – 1
PEP8 and naming convention has been used, but not regularly – 2
PEP8 and naming convention has been used regularly – 3
PEP8 convention used professionally and all items have been named correctly – 4

In-code Comments (up to 3)

No in-code comments – 0
Code contains occasional in-code comments – 1
Code contains useful and regular in-code comments – 2
Thoroughly commented, good use of docstrings, and header comments describing.py files – 3

For this criterion I think I got: 11 out of 15

DOCUMENTATION (20)

Design (up to 10) clear exposition about the design and decisions for OOP use

The documentation cannot be understood on first reading or mostly incomplete – 0
The documentation is readable, but a section(s) are missing – 1 to 3
The documentation is complete – 4 to 6
The documentation is complete and of a high standard – 7 to 10

Testing (5)

Testing has not been demonstrated in the documentation – 0
Little white box testing has been documented – 1 or 2
White box testing has been documented for all the coursework – 3 or 4
White box testing has been documented for the whole system – 5

Evaluation (5)

No evaluation was shown in the documentation – 0
The evaluation shows a lack of thought – 1 or 2
The evaluation shows thought – 3 or 4
The evaluation shows clear introspection, demonstrates increased awareness – 5

For this criterion I think I got: 15 out of 20

ACCEPTANCE TESTS - DEMONSTRATIONS (10)

Final Demo (up to 10)

Not attended or no work demonstrated – 0
Work demonstrated was not up to the standard expected – 1 to 3
Work demonstrated was up to the standard expected – 4 to 7
Work demonstrated exceeded the standard expected – 8 to 10

For this criterion I think I got: 8 out of 10

I think my overall mark would be: 84 out of 100

9. Group Pro forma

Describe the division of work and agree percentage contributions. The pro forma must be signed by all group members and an identical copy provided in each report. If you cannot agree percentage contributions, please indicate so in the notes column and provide your reasoning.
(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)

Partner ID	Tasks/Features Completed	%Contribution	Signature	Notes
1	Feature 1 i / Feature 2 i	33.3	Chisel	
2	Feature 1 ii / Feature 2 ii	33.3	Emir	
3	Feature 1 iii/ Feature 2 iii	33.3	Baris	
	Total	99.9		

Bibliography

Funtrivia.com. 2022. *History of Mathematics Quiz | Scientists & Inventors | 10 Questions*. [online] Available at: <https://www.funtrivia.com/trivia-quiz/People/History-of-Mathematics-2905.html> [Accessed 6 January 2022].

GitHub. 2022. *Search · python · YihangLou/FasterRCNN-Encapsulation-Cplusplus*. [online] Available at: <https://github.com/YihangLou/FasterRCNN-Encapsulation-Cplusplus/search?q=python> [Accessed 20 January 2022].

QUESTIONS, N., 2022. *Number Forms Multiple Choice Questions Worksheets | Turtle Diary*. [online] Turtlediary.com. Available at: <https://www.turtlediary.com/worksheets/number-forms-multiple-choice-questions.html> [Accessed 13 January 2022].

APPENDIX A: CODE LISTING

*Provide a complete listing of all the *.py files in your PyCharm project. Make sure your code is well commented and applies professional Python convention (refer to PEP 8 for details). The code listed here must match that uploaded to Moodle. Please copy and paste the actual code – no screenshots please! You will lose marks if screenshots are provided instead of code.*

(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)

PLEASE DON'T FORGET TO INSTALL PACKAGES IN PYCHARM!

```
# Importing the packages and library files.
# Before run the program, make sure you download the required packages from Python
Packages which is provided at below.

from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from mysql.connector import connect, Error
import random as random
from functools import partial
from fpdf import FPDF

# Before run the program, make sure you download the required packages from Python
Packages which is provided at top.

root = Tk()

# Used StringVar() to manage the value of a widget such as entry boxes.
usernameEntry = StringVar()
passwordEntry = StringVar()


# Created the first class which is going to be the root main. Then we are going to
use this class as an inheritance below.
class Main:
    # Created the first frame and giving the details of the frame.
    def __init__(self, root):
        root.title("Login")
        root.geometry("530x180")
        # root.resizable(0, 0)

        self.root = root
        self.rootFrame = Frame(self.root)
        self.rootFrame.pack(fill="both", expand=1)

        # Creating the entry box physical details.
        self.usernameLabel = Label(self.rootFrame, text='Username')
        self.usernameLabel.place(x=10, y=10)
        self.usernameEntry = Entry(self.rootFrame, textvariable=usernameEntry,
width=35)
        self.usernameEntry.place(x=10, y=30)

        self.passwordLabel = Label(self.rootFrame, text='Password')
        self.passwordLabel.place(x=10, y=70)
        self.passwordEntry = Entry(self.rootFrame, textvariable=passwordEntry,
width=35, show='*')
        self.passwordEntry.place(x=10, y=90)
        self.usernameEntry.delete(0, END)
        self.passwordEntry.delete(0, END)

        # Credentials
```

```
        self.studentCredentialsLabel = Label(self.rootFrame,
text='Student\nUsername: "cisel"\nPassword: "2"')
        self.studentCredentialsLabel.place(x=380, y=10)

        self.adminCredentialsLabel = Label(self.rootFrame, text='Admin\nUsername:
"baris"\nPassword: "3"')
        self.adminCredentialsLabel.place(x=380, y=70)

        # Clicked button will be tested(Black Box!) - Baris
        # Creating a clicked element to clicked on any button or element
        def clicked(event):
            check()

        # test finished

        # Check function will be tested(black box)-Baris
        # Checking the username and password
        def check():
            username = usernameEntry.get()
            password = passwordEntry.get()

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if len(username) == 0 or len(password) == 0:
                messagebox.showinfo("Warning!", "Please enter your username and
password.")
                return

            # Connecting to the database to check the username and password.
            try:
                with connect(
                        host="auth-db582.hostinger.com",
                        user="u998717846_test_user",
                        password="7$Mw9Q=e",
                        database="u998717846_python_test"
                ) as connection:

                    query = "SELECT * FROM users WHERE username = %s AND password
= %s"
                    value = (username, password)

                    with connection.cursor() as cursor:
                        cursor.execute(query, value)
                        result = cursor.fetchone()

                        if result:
                            # We gave the print command to see on terminal for
proof if the code is working.
                            print('The user credentials are correct.')

                            # If the value showing "admin" root frame will switch
to the admin.panel.frame
                            if result[4] == "Admin":
                                print('The user is admin.')

                                self.rootFrame.destroy()
                                AdminPanel(root)
                                root.mainloop()

                            # If the value showing "student" root frame will
switch to the student.panel.frame
                            if result[4] == "Student":
                                print('The user is student.')
```

```
                                self.rootFrame.destroy()
                                StudentPanel(root)
                                root.mainloop()

                        # If the entered value is won't matching with database
program will print wrong on terminal and messagebox.show info will show the
(Warning! Invalid username and password)
                        else:
                            # We gave the print command to see on terminal for
proof.
                            print(
                                'The user credentials are invalid!')  # We gave
the print command to see on terminal for proof.
                            messagebox.showinfo('Warning!', 'Invalid username or
password!')

            # Checking the errors
            except Error as e:
                print(e)

        # Created the main button to click on it for log in.
        self.mainButton = Button(self.rootFrame, text='Login', width=33,
command=check)
        self.mainButton.place(x=10, y=130)

        root.bind("<Return>", clicked)


# Starting to crate Admin Panel class and design of the frames.
class AdminPanel:

    def __init__(self, root):
        admin_panel_frame = Frame(root)
        admin_panel_frame.pack(fill='both', expand=1)

        root.title("Admin Panel")
        root.geometry('355x250')

        # Managment panels will be tested(black box)-Baris
        # If user going to click in user management button admin_panel_frame will
destroy and UserManagmentPanel will be open on the window. Same for all
definitions below
        def user_management():
            admin_panel_frame.destroy()
            UserManagementPanel(root)

        def module_management_panel():
            admin_panel_frame.destroy()
            ModuleManagementPanel(root)

        def topic_management_panel():
            admin_panel_frame.destroy()
            TopicManagementPanel(root)

        def quiz_management_panel():
            admin_panel_frame.destroy()
            QuizManagementPanel(root)

        def question_management_panel():
            admin_panel_frame.destroy()
            QuestionManagement(root)

        # logout function will be test(black box)-Baris
        def logout():
```

```
            are_you_sure = messagebox.askyesno("Alert", "Are you sure you want to
logout?")

            if are_you_sure:
                admin_panel_frame.destroy()
                Main(root)

        # Creating the buttons for the management frames
        Button(admin_panel_frame, text='User Management', width=33,
command=user_management).place(x=10, y=10)
        Button(admin_panel_frame, text='Module Management', width=33,
command=module_management_panel).place(x=10, y=50)
        Button(admin_panel_frame, text='Topic Management', width=33,
command=topic_management_panel).place(x=10, y=90)
        Button(admin_panel_frame, text='Quiz Management', width=33,
command=quiz_management_panel).place(x=10, y=130)
        Button(admin_panel_frame, text='Question Management',
command=question_management_panel, width=33).place(x=10,

y=170)
        Button(admin_panel_frame, text='Logout', command=logout,
width=33).place(x=10, y=210)


# Creating the user management frame and designing the buttons.
class UserManagementPanel:

    def __init__(self, root):
        user_management_panel_frame = Frame(root)
        user_management_panel_frame.pack(fill='both', expand=1)

        root.title("User Management")
        root.geometry('1490x550')

        # Connecting the database again to add/edit/delete/update the all records
        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            # Giving the query to access the table.
            query = "SELECT * FROM users"

            c.execute(query)
            records = c.fetchall()

            # Add our data to the screen
            global count
            count = 0

            for record in records:
                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
```

```
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[1], record[2], record[3],
record[4]),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[1], record[2], record[3],
record[4]),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(user_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("ID", "Email", "Username", "Password", "Access
Level")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
```

```
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Email", anchor=W, width=140)
        my_tree.column("Username", anchor=W, width=140)
        my_tree.column("Password", anchor=W, width=140)
        my_tree.column("Access Level", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
        my_tree.heading("Email", text="Email", anchor=W)
        my_tree.heading("Username", text="Username", anchor=W)
        my_tree.heading("Password", text="Password", anchor=W)
        my_tree.heading("Access Level", text="Access Level", anchor=W)

        # Add record part will be test (black box) by Emir
        # Add Record Entry Boxes
        data_frame = LabelFrame(user_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)

        email_label = Label(data_frame, text="Email")
        email_label.grid(row=0, column=2, padx=10, pady=10)
        email_entry = Entry(data_frame)
        email_entry.grid(row=0, column=3, padx=10, pady=10)

        username_label = Label(data_frame, text="Username")
        username_label.grid(row=0, column=4, padx=10, pady=10)
        username_entry = Entry(data_frame)
        username_entry.grid(row=0, column=5, padx=10, pady=10)

        password_label = Label(data_frame, text="Password")
        password_label.grid(row=0, column=6, padx=10, pady=10)
        password_entry = Entry(data_frame)
        password_entry.grid(row=0, column=7, padx=10, pady=10)

        # Access level will be test(black box)-Emir
        # Checking the access level to understand if user student or admin
        access_level = StringVar()
        access_levels = ["Admin", "Student"]

        access_level_label = Label(data_frame, text="Access Level")
        access_level_label.grid(row=0, column=8, padx=10, pady=10)
        access_level_menu = OptionMenu(data_frame, access_level, *access_levels)
        access_level.set("Select Access Level")
        access_level_menu.grid(row=0, column=9, padx=10, pady=10)

        # Move Row Up
        def up():
            rows = my_tree.selection()
            for row in rows:
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

        # Move Row Down
        def down():
            rows = my_tree.selection()
            for row in reversed(rows):
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

        # Clear entry boxes will be checked by Emir
        # Clear entry boxes
```

```
        def clear_entries():
            # Clear entry boxes
            id_entry.delete(0, END)
            email_entry.delete(0, END)
            username_entry.delete(0, END)
            password_entry.delete(0, END)
            access_level.set("Select Access Level")

        # Select record function will be test by Emir
        # Select Record
        def select_record(e):
            # Clear entry boxes
            id_entry.delete(0, END)
            email_entry.delete(0, END)
            username_entry.delete(0, END)
            password_entry.delete(0, END)
            access_level.set("Select Access Level")

            # Grab record number
            selected = my_tree.focus()

            # Grab record values
            values = my_tree.item(selected, 'values')

            # output to entry boxes to see what is chosen
            id_entry.insert(0, values[0])
            email_entry.insert(0, values[1])
            username_entry.insert(0, values[2])
            password_entry.insert(0, values[3])
            access_level.set(values[4])

        # Remove record function will be checked by Emir
        # Remove one record
        def remove_one():

            # If there is no selected record it will show the
messagebox.showinfo()
            if len(my_tree.focus()) == 0:
                messagebox.showinfo("Warning!", "Please select the record you want
to delete.")
                query_database()
                return

            x = my_tree.selection()[0]
            my_tree.delete(x)

            # Connecting to the database
            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM users WHERE id=" + id_entry.get())

            # Commit changes
            conn.commit()

            # Close connection
```

```
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        # remove all entries tested by Chisel
        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                # Connecting to the database
                conn = connect(
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute("TRUNCATE TABLE users")

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Adding record tested by Chisel
        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if len(email_entry.get()) == 0 or len(username_entry.get()) == 0 or
len(
                    password_entry.get()) == 0 or access_level.get() == "Select
Access Level":
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Connecting to the database
```

```
            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            select_query = "SELECT * FROM users WHERE email=%s OR username=%s"
            user_values = (email_entry.get(), username_entry.get())

            # Execute query
            c.execute(select_query, user_values)

            # Fetch user
            user = c.fetchall()

            # Check if the email and username of the entered user exist.
            if user:
                # Existing user
                print('Existing user!')

                messagebox.showinfo("Warning!", "The user is exist.")

                # Clear The Entry Boxes
                clear_entries()

            else:
                # Not existing user
                print('New user.')

                # SQL query to add users to the "user" table in the database.
                insert_user = "INSERT INTO users (email, username, password,
access_level) VALUES (%s, %s, %s, %s)"
                user_credentials = (email_entry.get(), username_entry.get(),
password_entry.get(), access_level.get())

                # Execute query
                c.execute(insert_user, user_credentials)

                # Clear The Entry Boxes
                clear_entries()

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update selected record tested by Chisel
        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if len(email_entry.get()) == 0 or len(username_entry.get()) == 0 or
```

```
len(
                    password_entry.get()) == 0 or access_level.get() == "Select
Access Level":
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            # Update record
            my_tree.item(selected, text="", values=(
                email_entry.get(), username_entry.get(), password_entry.get(),
access_level.get()))

            # Calling queries
            query = "UPDATE users SET email = %s, username = %s, password = %s,
access_level = %s WHERE id = %s"
            value = (email_entry.get(), username_entry.get(),
password_entry.get(), access_level.get(), id_entry.get())

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute(query, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(user_management_panel_frame, text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)

        add_button = Button(button_frame, text="Add Record", command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)
```

```
        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)

        # Going to previous page tested by Chisel
        # If user click on previous page button it will take the frames to back
        def previous_page():
            user_management_panel_frame.destroy()
            AdminPanel(root)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()


# creating a class for ModuleManagementPanel.
class ModuleManagementPanel:

    def __init__(self, root):
        module_management_panel_frame = Frame(root)
        module_management_panel_frame.pack(fill='both', expand=1)

        root.title("Module Management Panel")
        root.geometry('1230x550')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            query = "SELECT * FROM modules"

            c.execute(query)
            records = c.fetchall()

            # Add our data to the screen
            global count
            count = 0
```

```
            for record in records:
                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[1], record[2]),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[1], record[2]),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(module_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("ID", "Module Code", "Module Name")
```

```
        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Module Code", anchor=W, width=140)
        my_tree.column("Module Name", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
        my_tree.heading("Module Code", text="Module Code", anchor=W)
        my_tree.heading("Module Name", text="Module Name", anchor=W)

        # Add Record Entry Boxes
        data_frame = LabelFrame(module_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)

        module_code_label = Label(data_frame, text="Module Code")
        module_code_label.grid(row=0, column=2, padx=10, pady=10)
        module_code_entry = Entry(data_frame)
        module_code_entry.grid(row=0, column=3, padx=10, pady=10)

        module_name_label = Label(data_frame, text="Module Name")
        module_name_label.grid(row=0, column=4, padx=10, pady=10)
        module_name_entry = Entry(data_frame)
        module_name_entry.grid(row=0, column=5, padx=10, pady=10)

        # Move Row Up
        def up():
            rows = my_tree.selection()
            for row in rows:
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

        # Move Row Down
        def down():
            rows = my_tree.selection()
            for row in reversed(rows):
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

        # Clear entry boxes
        def clear_entries():
            # Clear entry boxes
            id_entry.delete(0, END)
            module_code_entry.delete(0, END)
            module_name_entry.delete(0, END)

        # Select Record
        def select_record(e):
            # Clear entry boxes
            id_entry.delete(0, END)
            module_code_entry.delete(0, END)
            module_name_entry.delete(0, END)

            # Grab record number
            selected = my_tree.focus()

            # Grab record values
            values = my_tree.item(selected, 'values')

            # Output to entry boxes
```

```
            id_entry.insert(0, values[0])
            module_code_entry.insert(0, values[1])
            module_name_entry.insert(0, values[2])

        # Remove one record
        def remove_one():

            # If there is no selected record it will show the
messagebox.showinfo()
            if len(my_tree.focus()) == 0:
                messagebox.showinfo("Warning!", "Please select the record you want
to delete.")
                query_database()
                return

            x = my_tree.selection()[0]
            my_tree.delete(x)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM modules WHERE id=" + id_entry.get())

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                conn = connect(
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()
```

```
                # Execute query
                c.execute("TRUNCATE TABLE modules")

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if len(module_code_entry.get()) == 0 or len(module_name_entry.get())
== 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            ##########

            # Connecting to the database
            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            check_query = "SELECT * FROM modules WHERE moduleCode=%s OR
moduleName=%s"
            check_values = (module_code_entry.get(), module_name_entry.get())

            # Execute query
            c.execute(check_query, check_values)

            # Fetch user
            check = c.fetchall()

            # Check if the email and username of the entered user exist.
            if check:
                # Existing user
                print('Existing module!')

                messagebox.showinfo("Warning!", "The module is exist.")

                # Clear The Entry Boxes
                clear_entries()

            else:
                # Not existing user
```

```
                print('New module.')

                query = "INSERT INTO modules (moduleCode, moduleName) VALUES (%s,
%s)"
                value = (module_code_entry.get(), module_name_entry.get())

                conn = connect(
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute(query, value)

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

            ##########

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if len(module_code_entry.get()) == 0 or len(module_name_entry.get())
== 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            # Update record
            my_tree.item(selected, text="", values=(
                module_code_entry.get(), module_name_entry.get()))

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if len(module_code_entry.get()) == 0 or len(module_name_entry.get())
== 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            query = "UPDATE modules SET moduleCode = %s, moduleName = %s WHERE id
= %s"
```

```
            value = (module_code_entry.get(), module_name_entry.get(),
id_entry.get())

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute(query, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(module_management_panel_frame, text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)

        add_button = Button(button_frame, text="Add Record", command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)

        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)

        def previous_page():
            module_management_panel_frame.destroy()
            AdminPanel(root)
```

```
        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()


class TopicManagementPanel:

    def __init__(self, root):
        topic_management_panel_frame = Frame(root)
        topic_management_panel_frame.pack(fill='both', expand=1)

        root.title("Topic Management Panel")
        root.geometry('1230x550')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            query = "SELECT * FROM topics INNER JOIN modules ON topics.moduleID =
modules.id"

            c.execute(query)
            records = c.fetchall()

            # Add our data to the screen
            global count
            count = 0

            for record in records:
                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[5], record[2]),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[5], record[2]),
                        tags=("oddrow",)
                    )
```

```
                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(topic_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("ID", "Module Name", "Topic Name")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Module Name", anchor=W, width=140)
        my_tree.column("Topic Name", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
        my_tree.heading("Module Name", text="Module Name", anchor=W)
        my_tree.heading("Topic Name", text="Topic Name", anchor=W)

        # Add Record Entry Boxes
        data_frame = LabelFrame(topic_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)
```

```
        # MODULE MENU START #

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        query = "SELECT * FROM modules"

        c = conn.cursor()
        # print(conn)

        c.execute(query)
        records = c.fetchall()

        # datatype of menu text
        module_menu = StringVar()

        module_list = dict()
        for record in records:
            module_list[record[0]] = record[2]

        module_name_label = Label(data_frame, text="Module Name")
        module_name_label.grid(row=0, column=2, padx=10, pady=10)
        module_name_menu = OptionMenu(data_frame, module_menu,
*module_list.values())
        module_menu.set("Select Module")
        module_name_menu.grid(row=0, column=3, padx=10, pady=10)

        # MODULE MENU END #

        topic_name_label = Label(data_frame, text="Topic Name")
        topic_name_label.grid(row=0, column=4, padx=10, pady=10)
        topic_name_entry = Entry(data_frame)
        topic_name_entry.grid(row=0, column=5, padx=10, pady=10)

        # Move Row Up
        def up():
            rows = my_tree.selection()
            for row in rows:
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

        # Move Row Down
        def down():
            rows = my_tree.selection()
            for row in reversed(rows):
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

        # Clear entry boxes
        def clear_entries():
            # Clear entry boxes
            id_entry.delete(0, END)
            module_menu.set("Select Module")
            topic_name_entry.delete(0, END)

        # Select Record
        def select_record(e):
            # Clear entry boxes
            id_entry.delete(0, END)
            module_menu.set("Select Module")
```

```
            topic_name_entry.delete(0, END)

            # Grab record number
            selected = my_tree.focus()

            # Grab record values
            values = my_tree.item(selected, 'values')

            # Output to entry boxes
            id_entry.insert(0, values[0])
            module_menu.set(values[1])
            topic_name_entry.insert(0, values[2])

        # Remove one record
        def remove_one():

            # If there is no selected record it will show the
messagebox.showinfo()
            if len(my_tree.focus()) == 0:
                messagebox.showinfo("Warning!", "Please select the record you want
to delete.")
                query_database()
                return

            x = my_tree.selection()[0]
            my_tree.delete(x)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM topics WHERE id=" + id_entry.get())

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                conn = connect(
```

```
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute("TRUNCATE TABLE topics")

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if module_menu.get() == "Select Module" or len(topic_name_entry.get())
== 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            query = "SELECT * FROM modules WHERE
moduleName='{}'".format(module_menu.get())
            c.execute(query)
            result = c.fetchone()

            query2 = "INSERT INTO topics (moduleID, topicName) VALUES (%s, %s)"
            value = (result[0], topic_name_entry.get())
            c.execute(query2, value)

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
```

```
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if module_menu.get() == "Select Module" or len(topic_name_entry.get())
== 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            fetch_modules = "SELECT * FROM modules WHERE
moduleName='{}'".format(module_menu.get())
            c.execute(fetch_modules)
            module_result = c.fetchone()

            update_topic = "UPDATE topics SET moduleID = %s, topicName = %s WHERE
id = %s"
            value = (module_result[0], topic_name_entry.get(), id_entry.get())
            c.execute(update_topic, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Update record
            my_tree.item(selected, text="", values=(
                module_result[0], topic_name_entry.get()))

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(topic_management_panel_frame, text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)
```

```
        add_button = Button(button_frame, text="Add Record", command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)

        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)

        def previous_page():
            topic_management_panel_frame.destroy()
            AdminPanel(root)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()


class QuizManagementPanel:

    def __init__(self, root):
        quiz_management_panel_frame = Frame(root)
        quiz_management_panel_frame.pack(fill='both', expand=1)

        root.title("Quiz Management Panel")
        root.geometry('1230x550')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)
```

```
            query = "SELECT * FROM quizzes INNER JOIN topics ON quizzes.topicID =
topics.id INNER JOIN modules ON topics.moduleID = modules.id"

            c.execute(query)
            records = c.fetchall()

            # Add our data to the screen
            global count
            count = 0

            for record in records:
                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[8], record[5], record[2]),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[8], record[5], record[2]),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(quiz_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
```

```
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("ID", "Module Name", "Topic Name", "Quiz Name")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Module Name", anchor=W, width=140)
        my_tree.column("Topic Name", anchor=W, width=140)
        my_tree.column("Quiz Name", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
        my_tree.heading("Module Name", text="Module Name", anchor=W)
        my_tree.heading("Topic Name", text="Topic Name", anchor=W)
        my_tree.heading("Quiz Name", text="Quiz Name", anchor=W)

        # Add Record Entry Boxes
        data_frame = LabelFrame(quiz_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)

        # TOPIC MENU START #

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        query = "SELECT * FROM topics"

        c = conn.cursor()
        # print(conn)

        c.execute(query)
        records = c.fetchall()

        # Datatype of menu text
        topic_menu = StringVar()

        topic_list = dict()
        for record in records:
            topic_list[record[0]] = record[2]

        topic_name_label = Label(data_frame, text="Topic Name")
        topic_name_label.grid(row=0, column=2, padx=10, pady=10)
        topic_name_menu = OptionMenu(data_frame, topic_menu, *topic_list.values())
        topic_menu.set("Select Topic")
        topic_name_menu.grid(row=0, column=3, padx=10, pady=10)

        # TOPIC MENU END #
```

```
        quiz_name_label = Label(data_frame, text="Quiz Name")
        quiz_name_label.grid(row=0, column=4, padx=10, pady=10)
        quiz_name_entry = Entry(data_frame)
        quiz_name_entry.grid(row=0, column=5, padx=10, pady=10)

        # Move Row Up
        def up():
            rows = my_tree.selection()
            for row in rows:
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

        # Move Row Down
        def down():
            rows = my_tree.selection()
            for row in reversed(rows):
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

        # Clear entry boxes
        def clear_entries():
            # Clear entry boxes
            id_entry.delete(0, END)
            topic_menu.set("Select Topic")
            quiz_name_entry.delete(0, END)

        # Select Record
        def select_record(e):
            # Clear entry boxes
            id_entry.delete(0, END)
            topic_menu.set("Select Topic")
            quiz_name_entry.delete(0, END)

            # Grab record number
            selected = my_tree.focus()

            # Grab record values
            values = my_tree.item(selected, 'values')

            # Output to entry boxes
            id_entry.insert(0, values[0])
            topic_menu.set(values[2])
            quiz_name_entry.insert(0, values[3])

        # Remove one record
        def remove_one():

            # If there is no selected record it will show the
messagebox.showinfo()
            if len(my_tree.focus()) == 0:
                messagebox.showinfo("Warning!", "Please select the record you want
to delete.")
                query_database()
                return

            x = my_tree.selection()[0]
            my_tree.delete(x)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )
```

```
            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM quizzes WHERE id=" + id_entry.get())

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                conn = connect(
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute("TRUNCATE TABLE quizzes")

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if topic_menu.get() == "Select Topic" or len(quiz_name_entry.get()) ==
0:
```

```
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Fetch selected topic
            fetch_topic = "SELECT * FROM topics WHERE
topicName='{}'".format(topic_menu.get())
            c.execute(fetch_topic)
            topic_result = c.fetchone()

            insert_quiz = "INSERT INTO quizzes (topicID, quizName) VALUES (%s,
%s)"
            value = (topic_result[0], quiz_name_entry.get())
            c.execute(insert_quiz, value)

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if topic_menu.get() == "Select Topic" or len(quiz_name_entry.get()) ==
0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()
```

```
            # Fetch selected topic
            fetch_topic = "SELECT * FROM topics WHERE
topicName='{}'".format(topic_menu.get())
            c.execute(fetch_topic)
            topic_result = c.fetchone()

            update_quiz = "UPDATE quizzes SET topicID = %s, quizName = %s WHERE id
= %s"
            value = (topic_result[0], quiz_name_entry.get(), id_entry.get())
            c.execute(update_quiz, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Update record
            my_tree.item(selected, text="", values=(
                topic_result[0], quiz_name_entry.get()))

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(quiz_management_panel_frame, text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)

        add_button = Button(button_frame, text="Add Record", command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)

        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)

        def previous_page():
            quiz_management_panel_frame.destroy()
            AdminPanel(root)

        previous_page_button = Button(button_frame, text="Previous Page",
```

```
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()


class QuestionManagement:

    def __init__(self, root):
        question_management_panel_frame = Frame(root)
        question_management_panel_frame.pack(fill='both', expand=1)

        root.title("Question Management Panel")
        root.geometry('355x170')

        def mc_question_management_panel():
            question_management_panel_frame.destroy()
            MCQuestionManagement(root)

        def tf_question_management_panel():
            question_management_panel_frame.destroy()
            TFQuestionManagement(root)

        def fb_question_management_panel():
            question_management_panel_frame.destroy()
            FBQuestionManagement(root)

        def previous_page():
            question_management_panel_frame.destroy()
            AdminPanel(root)

        Button(question_management_panel_frame, text='Multiple Choice Question
Management',
               command=mc_question_management_panel, width=33).place(x=10, y=10)
        Button(question_management_panel_frame, text='True and False Question
Management', width=33,
               command=tf_question_management_panel).place(x=10, y=50)
        Button(question_management_panel_frame, text='Fill in the Blank Question
Management', width=33,
               command=fb_question_management_panel).place(x=10, y=90)
        Button(question_management_panel_frame, text='Previous Page',
command=previous_page, width=33).place(x=10,

y=130)


class MCQuestionManagement:

    def __init__(self, root):
        mc_question_management_panel_frame = Frame(root)
        mc_question_management_panel_frame.pack(fill='both', expand=1)

        root.title("Multiple Choice Question Management Panel")
        root.geometry('1490x700')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)
```

```
            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            query = "SELECT * FROM questionsMC INNER JOIN quizzes ON
questionsMC.quizID = quizzes.id INNER JOIN topics ON quizzes.topicID = topics.id
INNER JOIN modules ON topics.moduleID = modules.id;"

            c.execute(query)
            records = c.fetchall()

            # Add our data to the screen
            global count
            count = 0

            for record in records:
                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(
                            record[0], record[18], record[15], record[12],
record[2], record[4], record[5], record[6],
                            record[7], record[8]),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(
                            record[0], record[18], record[15], record[12],
record[2], record[4], record[5], record[6],
                            record[7], record[8]),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
```

```
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(mc_question_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = (
            "ID", "Module Name", "Topic Name", "Quiz Name", "Question", "Option
A", "Option B", "Option C", "Option D",
            "Answer")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Module Name", anchor=W, width=140)
        my_tree.column("Quiz Name", anchor=W, width=140)
        my_tree.column("Topic Name", anchor=W, width=140)
        my_tree.column("Question", anchor=W, width=140)
        my_tree.column("Option A", anchor=W, width=140)
        my_tree.column("Option B", anchor=W, width=140)
        my_tree.column("Option C", anchor=W, width=140)
        my_tree.column("Option D", anchor=W, width=140)
        my_tree.column("Answer", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
        my_tree.heading("Module Name", text="Module Name", anchor=W)
        my_tree.heading("Quiz Name", text="Quiz Name", anchor=W)
        my_tree.heading("Topic Name", text="Topic Name", anchor=W)
        my_tree.heading("Question", text="Question", anchor=W)
        my_tree.heading("Option A", text="Option A", anchor=W)
        my_tree.heading("Option B", text="Option B", anchor=W)
        my_tree.heading("Option C", text="Option C", anchor=W)
        my_tree.heading("Option D", text="Option D", anchor=W)
        my_tree.heading("Answer", text="Answer", anchor=W)

        # Add Record Entry Boxes
        data_frame = LabelFrame(mc_question_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)
```

```
        # QUIZ MENU START #

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        query = "SELECT * FROM topics INNER JOIN modules ON topics.moduleID =
modules.id INNER JOIN quizzes ON topics.id = quizzes.topicID"

        c = conn.cursor()
        c.execute(query)
        records = c.fetchall()

        # Datatype of menu text
        quiz_menu = StringVar()

        quiz_list = dict()
        for record in records:
            quiz_list[record[6]] = record[8]

        quiz_name_label = Label(data_frame, text="Quiz Name")
        quiz_name_label.grid(row=0, column=2, padx=10, pady=10)
        quiz_name_menu = OptionMenu(data_frame, quiz_menu, *quiz_list.values())
        quiz_menu.set("Select Quiz")
        quiz_name_menu.grid(row=0, column=3, padx=10, pady=10)

        # QUIZ MENU END #

        # Add Question Entry Boxes
        question_frame = LabelFrame(mc_question_management_panel_frame,
text="Question")
        question_frame.pack(fill="x", expand="yes", padx=20)

        question_label = Label(question_frame, text="Question")
        question_label.grid(row=0, column=0, padx=10, pady=10)
        question_entry = Entry(question_frame, width=148)
        question_entry.grid(row=0, column=1, padx=10, pady=10)

        # Add Options Entry Boxes
        option_frame = LabelFrame(mc_question_management_panel_frame,
text="Options & Answer")
        option_frame.pack(fill="x", expand="yes", padx=20)

        option_a_label = Label(option_frame, text="Option A")
        option_a_label.grid(row=0, column=0, padx=10, pady=10)
        option_a_entry = Entry(option_frame)
        option_a_entry.grid(row=0, column=1, padx=10, pady=10)

        option_b_label = Label(option_frame, text="Option B")
        option_b_label.grid(row=0, column=2, padx=10, pady=10)
        option_b_entry = Entry(option_frame)
        option_b_entry.grid(row=0, column=3, padx=10, pady=10)

        option_c_label = Label(option_frame, text="Option C")
        option_c_label.grid(row=0, column=4, padx=10, pady=10)
        option_c_entry = Entry(option_frame)
        option_c_entry.grid(row=0, column=5, padx=10, pady=10)

        option_d_label = Label(option_frame, text="Option D")
        option_d_label.grid(row=0, column=6, padx=10, pady=10)
        option_d_entry = Entry(option_frame)
```

```
    option_d_entry.grid(row=0, column=7, padx=10, pady=10)

    # datatype of menu text
    answer_entry = StringVar()

    answer_list = ['A', 'B', 'C', 'D']

    answer_label = Label(option_frame, text="Answer")
    answer_label.grid(row=0, column=8, padx=10, pady=10)
    answer_menu = OptionMenu(option_frame, answer_entry, *answer_list)
    answer_entry.set("Select Correct Answer")
    answer_menu.grid(row=0, column=9, padx=10, pady=10)

    # Move Row Up
    def up():
        rows = my_tree.selection()
        for row in rows:
            my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

    # Move Row Down
    def down():
        rows = my_tree.selection()
        for row in reversed(rows):
            my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

    # Clear entry boxes
    def clear_entries():
        # Clear entry boxes
        id_entry.delete(0, END)
        quiz_menu.set("Select Quiz")
        question_entry.delete(0, END)
        option_a_entry.delete(0, END)
        option_b_entry.delete(0, END)
        option_c_entry.delete(0, END)
        option_d_entry.delete(0, END)
        answer_entry.set("Select Correct Answer")

    # Select Record
    def select_record(e):
        # Clear entry boxes
        id_entry.delete(0, END)
        quiz_menu.set("Select Quiz")
        question_entry.delete(0, END)
        option_a_entry.delete(0, END)
        option_b_entry.delete(0, END)
        option_c_entry.delete(0, END)
        option_d_entry.delete(0, END)
        answer_entry.set("Select Correct Answer")

        # Grab record number
        selected = my_tree.focus()

        # Grab record values
        values = my_tree.item(selected, 'values')

        # Output to entry boxes
        id_entry.insert(0, values[0])
        quiz_menu.set(values[3])
        question_entry.insert(0, values[4])
        option_a_entry.insert(0, values[5])
        option_b_entry.insert(0, values[6])
        option_c_entry.insert(0, values[7])
        option_d_entry.insert(0, values[8])
        answer_entry.set(values[9])
```

```
        # Remove one record
        def remove_one():

            # If there is no selected record it will show the
messagebox.showinfo()
            if len(my_tree.focus()) == 0:
                messagebox.showinfo("Warning!", "Please select the record you want
to delete.")
                query_database()
                return

            x = my_tree.selection()[0]
            my_tree.delete(x)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM questionsMC WHERE id=" + id_entry.get())

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                conn = connect(
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute("TRUNCATE TABLE questionsMC")
```

```
                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
                    option_a_entry.get()) == 0 or len(option_b_entry.get()) == 0
or len(
                option_c_entry.get()) == 0 or len(option_d_entry.get()) == 0 or
len(answer_entry.get()) == 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Fetch selected quiz
            fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
            c.execute(fetch_quiz)
            quiz_result = c.fetchone()

            # Insert question
            insert_question = "INSERT INTO questionsMC (quizID, question, optionA,
optionB, optionC, optionD, answer) VALUES (%s, %s, %s, %s, %s, %s, %s)"
            value = (
                quiz_result[0], question_entry.get(), option_a_entry.get(),
option_b_entry.get(), option_c_entry.get(),
                option_d_entry.get(),
                answer_entry.get())
            c.execute(insert_question, value)

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()
```

```
            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
                    option_a_entry.get()) == 0 or len(option_b_entry.get()) == 0
or len(
                option_c_entry.get()) == 0 or len(option_d_entry.get()) == 0 or
len(answer_entry.get()) == 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Fetch selected quiz
            fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
            c.execute(fetch_quiz)
            quiz_result = c.fetchone()

            # Update question
            update_question = "UPDATE questionsMC SET quizID = %s, question = %s,
optionA = %s, optionB = %s, optionC = %s, optionD = %s, answer = %s WHERE id = %s"
            value = (
                quiz_result[0], question_entry.get(), option_a_entry.get(),
option_b_entry.get(), option_c_entry.get(),
                option_d_entry.get(), answer_entry.get(), id_entry.get())
            c.execute(update_question, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Update record
            my_tree.item(selected, text="", values=(
                quiz_result[0], question_entry.get(), option_a_entry.get(),
option_b_entry.get(), option_c_entry.get(),
                answer_entry.get()))

            # Clear The Entry Boxes
            clear_entries()
```

```
            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(mc_question_management_panel_frame,
text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)

        add_button = Button(button_frame, text="Add Record", command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)

        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)

        def previous_page():
            mc_question_management_panel_frame.destroy()
            QuestionManagement(root)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()


class TFQuestionManagement:

    def __init__(self, root):
        tf_question_management_panel_frame = Frame(root)
        tf_question_management_panel_frame.pack(fill='both', expand=1)

        root.title("True and False Question Management Panel")
        root.geometry('1490x700')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
```

```
                my_tree.delete(record)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            query = "SELECT * FROM questionsTF INNER JOIN quizzes ON
questionsTF.quizID = quizzes.id INNER JOIN topics ON quizzes.topicID = topics.id
INNER JOIN modules ON topics.moduleID = modules.id;"

            c.execute(query)
            records = c.fetchall()

            # Add our data to the screen
            global count
            count = 0

            for record in records:
                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(
                            record[0], record[16], record[13], record[10],
record[2], record[4], record[5], record[6]),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(
                            record[0], record[16], record[13], record[10],
record[2], record[4], record[5], record[6]),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
```

```
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(tf_question_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("ID", "Module Name", "Topic Name", "Quiz Name",
"Question", "False", "True", "Answer")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Module Name", anchor=W, width=140)
        my_tree.column("Quiz Name", anchor=W, width=140)
        my_tree.column("Topic Name", anchor=W, width=140)
        my_tree.column("Question", anchor=W, width=140)
        my_tree.column("True", anchor=W, width=140)
        my_tree.column("False", anchor=W, width=140)
        my_tree.column("Answer", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
        my_tree.heading("Module Name", text="Module Name", anchor=W)
        my_tree.heading("Quiz Name", text="Quiz Name", anchor=W)
        my_tree.heading("Topic Name", text="Topic Name", anchor=W)
        my_tree.heading("Question", text="Question", anchor=W)
        my_tree.heading("True", text="True", anchor=W)
        my_tree.heading("False", text="False", anchor=W)
        my_tree.heading("Answer", text="Answer", anchor=W)

        # Add Record Entry Boxes
        data_frame = LabelFrame(tf_question_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)

        # QUIZ MENU START #

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
```

```
            database="u998717846_python_test"
        )

        query = "SELECT * FROM topics INNER JOIN modules ON topics.moduleID =
modules.id INNER JOIN quizzes ON topics.id = quizzes.topicID"

        c = conn.cursor()
        c.execute(query)
        records = c.fetchall()

        # datatype of menu text
        quiz_menu = StringVar()

        quiz_list = dict()
        for record in records:
            quiz_list[record[6]] = record[8]

        quiz_name_label = Label(data_frame, text="Quiz Name")
        quiz_name_label.grid(row=0, column=2, padx=10, pady=10)
        quiz_name_menu = OptionMenu(data_frame, quiz_menu, *quiz_list.values())
        quiz_menu.set("Select Quiz")
        quiz_name_menu.grid(row=0, column=3, padx=10, pady=10)

        # QUIZ MENU END #

        # Add Question Entry Boxes
        question_frame = LabelFrame(tf_question_management_panel_frame,
text="Question")
        question_frame.pack(fill="x", expand="yes", padx=20)

        question_label = Label(question_frame, text="Question")
        question_label.grid(row=0, column=0, padx=10, pady=10)
        question_entry = Entry(question_frame, width=148)
        question_entry.grid(row=0, column=1, padx=10, pady=10)

        # Add Options Entry Boxes
        option_frame = LabelFrame(tf_question_management_panel_frame,
text="Options & Answer")
        option_frame.pack(fill="x", expand="yes", padx=20)

        true_label = Label(option_frame, text="True")
        true_label.grid(row=0, column=0, padx=10, pady=10)
        true_entry = Entry(option_frame)
        true_entry.grid(row=0, column=1, padx=10, pady=10)

        false_label = Label(option_frame, text="False")
        false_label.grid(row=0, column=2, padx=10, pady=10)
        false_entry = Entry(option_frame)
        false_entry.grid(row=0, column=3, padx=10, pady=10)

        answer_label = Label(option_frame, text="Answer")
        answer_label.grid(row=0, column=4, padx=10, pady=10)
        answer_entry = Entry(option_frame)
        answer_entry.grid(row=0, column=5, padx=10, pady=10)

        # Move Row Up
        def up():
            rows = my_tree.selection()
            for row in rows:
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

        # Move Row Down
        def down():
            rows = my_tree.selection()
```

```
            for row in reversed(rows):
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

        # Clear entry boxes
        def clear_entries():
            # Clear entry boxes
            id_entry.delete(0, END)
            quiz_menu.set("Select Quiz")
            question_entry.delete(0, END)
            true_entry.delete(0, END)
            false_entry.delete(0, END)
            answer_entry.delete(0, END)

        # Select Record
        def select_record(e):
            # Clear entry boxes
            id_entry.delete(0, END)
            quiz_menu.set("Select Quiz")
            question_entry.delete(0, END)
            true_entry.delete(0, END)
            false_entry.delete(0, END)
            answer_entry.delete(0, END)

            # Grab record number
            selected = my_tree.focus()

            # Grab record values
            values = my_tree.item(selected, 'values')

            # Output to entry boxes
            id_entry.insert(0, values[0])
            quiz_menu.set(values[3])
            question_entry.insert(0, values[4])
            true_entry.insert(0, values[5])
            false_entry.insert(0, values[6])
            answer_entry.insert(0, values[7])

        # Remove one record
        def remove_one():

            # If there is no selected record it will show the
messagebox.showinfo()
            if len(my_tree.focus()) == 0:
                messagebox.showinfo("Warning!", "Please select the record you want
to delete.")
                query_database()
                return

            x = my_tree.selection()[0]
            my_tree.delete(x)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM questionsTF WHERE id=" + id_entry.get())
```

```
            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                conn = connect(
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute("TRUNCATE TABLE questionsTF")

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
                    true_entry.get()) == 0 or len(false_entry.get()) == 0 or len(
                answer_entry.get()) == 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return
```

```
            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Fetch selected quiz
            fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
            c.execute(fetch_quiz)
            quiz_result = c.fetchone()

            # Insert question
            insert_question = "INSERT INTO questionsTF (quizID, question,
optionTrue, optionFalse, answer) VALUES (%s, %s, %s, %s, %s)"
            value = (quiz_result[0], question_entry.get(), true_entry.get(),
false_entry.get(), answer_entry.get())
            c.execute(insert_question, value)

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
                    true_entry.get()) == 0 or len(false_entry.get()) == 0 or len(
                answer_entry.get()) == 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()
```

```
            # Fetch selected quiz
            fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
            c.execute(fetch_quiz)
            quiz_result = c.fetchone()

            # Update question
            update_question = "UPDATE questionsTF SET quizID = %s, question = %s,
optionTrue = %s, optionFalse = %s, answer = %s WHERE id = %s"
            value = (quiz_result[0], question_entry.get(), true_entry.get(),
false_entry.get(), answer_entry.get(),
                     id_entry.get())
            c.execute(update_question, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Update record
            my_tree.item(selected, text="", values=(
                quiz_result[0], question_entry.get(), true_entry.get(),
false_entry.get(), answer_entry.get()))

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(tf_question_management_panel_frame,
text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)

        add_button = Button(button_frame, text="Add Record", command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)

        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)
```

```
        def previous_page():
            tf_question_management_panel_frame.destroy()
            QuestionManagement(root)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()


class FBQuestionManagement:

    def __init__(self, root):
        fb_question_management_panel_frame = Frame(root)
        fb_question_management_panel_frame.pack(fill='both', expand=1)

        root.title("Fill in the Blank Question Management Panel")
        root.geometry('1490x700')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            query = "SELECT * FROM questionsFB INNER JOIN quizzes ON
questionsFB.quizID = quizzes.id INNER JOIN topics ON quizzes.topicID = topics.id
INNER JOIN modules ON topics.moduleID = modules.id;"

            c.execute(query)
            records = c.fetchall()

            # Add our data to the screen
            global count
            count = 0

            for record in records:
                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[14], record[11], record[8],
record[2], record[4]),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
```

```
                        index="end",
                        iid=count,
                        text="",
                        values=(record[0], record[14], record[11], record[8],
record[2], record[4]),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(fb_question_management_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("ID", "Module Name", "Topic Name", "Quiz Name",
"Question", "Answer")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("ID", anchor=W, width=50)
        my_tree.column("Module Name", anchor=W, width=140)
        my_tree.column("Quiz Name", anchor=W, width=140)
        my_tree.column("Topic Name", anchor=W, width=140)
        my_tree.column("Question", anchor=W, width=140)
        my_tree.column("Answer", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("ID", text="ID", anchor=W)
```

```
        my_tree.heading("Module Name", text="Module Name", anchor=W)
        my_tree.heading("Quiz Name", text="Quiz Name", anchor=W)
        my_tree.heading("Topic Name", text="Topic Name", anchor=W)
        my_tree.heading("Question", text="Question", anchor=W)
        my_tree.heading("Answer", text="Answer", anchor=W)

        # Add Record Entry Boxes
        data_frame = LabelFrame(fb_question_management_panel_frame, text="Record")
        data_frame.pack(fill="x", expand="yes", padx=20)

        id_label = Label(data_frame, text="ID")
        id_label.grid(row=0, column=0, padx=10, pady=10)
        id_entry = Entry(data_frame)
        id_entry.grid(row=0, column=1, padx=10, pady=10)

        # QUIZ MENU START #

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        query = "SELECT * FROM topics INNER JOIN modules ON topics.moduleID =
modules.id INNER JOIN quizzes ON topics.id = quizzes.topicID"

        c = conn.cursor()
        c.execute(query)
        records = c.fetchall()

        # datatype of menu text
        quiz_menu = StringVar()

        quiz_list = dict()
        for record in records:
            quiz_list[record[6]] = record[8]

        quiz_name_label = Label(data_frame, text="Quiz Name")
        quiz_name_label.grid(row=0, column=2, padx=10, pady=10)
        quiz_name_menu = OptionMenu(data_frame, quiz_menu, *quiz_list.values())
        quiz_menu.set("Select Quiz")
        quiz_name_menu.grid(row=0, column=3, padx=10, pady=10)

        # QUIZ MENU END #

        # Add Question Entry Boxes
        question_frame = LabelFrame(fb_question_management_panel_frame,
text="Question")
        question_frame.pack(fill="x", expand="yes", padx=20)

        question_label = Label(question_frame, text="Question")
        question_label.grid(row=0, column=0, padx=10, pady=10)
        question_entry = Entry(question_frame, width=148)
        question_entry.grid(row=0, column=1, padx=10, pady=10)

        # Add Options Entry Boxes
        option_frame = LabelFrame(fb_question_management_panel_frame,
text="Options & Answer")
        option_frame.pack(fill="x", expand="yes", padx=20)

        answer_label = Label(option_frame, text="Answer")
        answer_label.grid(row=0, column=0, padx=10, pady=10)
        answer_entry = Entry(option_frame)
```

```
        answer_entry.grid(row=0, column=1, padx=10, pady=10)

        # Move Row Up
        def up():
            rows = my_tree.selection()
            for row in rows:
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) - 1)

        # Move Row Down
        def down():
            rows = my_tree.selection()
            for row in reversed(rows):
                my_tree.move(row, my_tree.parent(row), my_tree.index(row) + 1)

        # Clear entry boxes
        def clear_entries():
            # Clear entry boxes
            id_entry.delete(0, END)
            quiz_menu.set("Select Quiz")
            question_entry.delete(0, END)
            answer_entry.delete(0, END)

        # Select Record
        def select_record(e):
            # Clear entry boxes
            id_entry.delete(0, END)
            quiz_menu.set("Select Quiz")
            question_entry.delete(0, END)
            answer_entry.delete(0, END)

            # Grab record number
            selected = my_tree.focus()

            # Grab record values
            values = my_tree.item(selected, 'values')

            # Output to entry boxes
            id_entry.insert(0, values[0])
            quiz_menu.set(values[3])
            question_entry.insert(0, values[4])
            answer_entry.insert(0, values[5])

        # Remove one record
        def remove_one():

            # If there is no selected record it will show the
messagebox.showinfo()
            if len(my_tree.focus()) == 0:
                messagebox.showinfo("Warning!", "Please select the record you want
to delete.")
                query_database()
                return

            x = my_tree.selection()[0]
            my_tree.delete(x)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
```

```
            c = conn.cursor()

            # Execute query
            c.execute("DELETE FROM questionsFB WHERE id=" + id_entry.get())

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Informative message box
            messagebox.showinfo("Deleted!", "Successfully deleted.")

        def remove_all():
            # Informative message box
            response = messagebox.askyesno("Warning!", "Are you sure?\nAll data
will be deleted!")

            # Add logic for message box
            if response == 1:
                # Clear the Treeview
                for record in my_tree.get_children():
                    my_tree.delete(record)

                conn = connect(
                    host="auth-db582.hostinger.com",
                    user="u998717846_test_user",
                    password="7$Mw9Q=e",
                    database="u998717846_python_test"
                )

                # Create a cursor instance
                c = conn.cursor()

                # Execute query
                c.execute("TRUNCATE TABLE questionsFB")

                # Commit changes
                conn.commit()

                # Close connection
                conn.close()

                # Clear The Entry Boxes
                clear_entries()

                # Recreate The Table
                # create_table_again()

                # Informative message box
                messagebox.showinfo("Deleted!", "Successfully deleted.")

        # Add new record to database
        def add_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
                    answer_entry.get()) == 0:
```

```
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            # Fetch selected quiz
            fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
            c.execute(fetch_quiz)
            quiz_result = c.fetchone()

            # Insert question
            insert_question = "INSERT INTO questionsFB (quizID, question, answer)
VALUES (%s, %s, %s)"
            value = (quiz_result[0], question_entry.get(), answer_entry.get())
            c.execute(insert_question, value)

            # Commit changes
            conn.commit()

            # Close connection
            conn.close()

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Update record
        def update_record():

            # If there is no value at the entry boxes it will show the
messagebox.showinfo()
            if quiz_menu.get() == "Select Quiz" or len(question_entry.get()) == 0
or len(
                    answer_entry.get()) == 0:
                messagebox.showinfo("Warning!", "Please enter all fields.")
                query_database()
                return

            # Grab the record number
            selected = my_tree.focus()

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
```

```
            c = conn.cursor()

            # Fetch selected quiz
            fetch_quiz = "SELECT * FROM quizzes WHERE
quizName='{}'".format(quiz_menu.get())
            c.execute(fetch_quiz)
            quiz_result = c.fetchone()

            # Update question
            update_question = "UPDATE questionsFB SET quizID = %s, question = %s,
answer = %s WHERE id = %s"
            value = (quiz_result[0], question_entry.get(), answer_entry.get(),
id_entry.get())
            c.execute(update_question, value)

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

            # Update record
            my_tree.item(selected, text="", values=(
                quiz_result[0], question_entry.get(), answer_entry.get()))

            # Clear The Entry Boxes
            clear_entries()

            # Clear The Treeview Table
            my_tree.delete(*my_tree.get_children())

            # Run to pull data from database on start
            query_database()

        # Add Buttons
        button_frame = LabelFrame(fb_question_management_panel_frame,
text="Commands")
        button_frame.pack(fill="x", expand="yes", padx=20)

        add_button = Button(button_frame, text="Add Record", command=add_record)
        add_button.grid(row=0, column=0, padx=10, pady=10)

        update_button = Button(button_frame, text="Update Record",
command=update_record)
        update_button.grid(row=0, column=1, padx=10, pady=10)

        remove_one_button = Button(button_frame, text="Remove Selected",
command=remove_one)
        remove_one_button.grid(row=0, column=2, padx=10, pady=10)

        remove_all_button = Button(button_frame, text="Remove All Records",
command=remove_all)
        remove_all_button.grid(row=0, column=3, padx=10, pady=10)

        move_up_button = Button(button_frame, text="Move Up", command=up)
        move_up_button.grid(row=0, column=4, padx=10, pady=10)

        move_down_button = Button(button_frame, text="Move Down", command=down)
        move_down_button.grid(row=0, column=5, padx=10, pady=10)

        select_record_button = Button(button_frame, text="Clear Entry Boxes",
command=clear_entries)
        select_record_button.grid(row=0, column=6, padx=10, pady=10)
```

```
        def previous_page():
            fb_question_management_panel_frame.destroy()
            QuestionManagement(root)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=7, padx=10, pady=10)

        # Bind the treeview
        my_tree.bind("<ButtonRelease-1>", select_record)

        # Run to pull data from database on start
        query_database()


class StudentPanel:

    def __init__(self, root):
        student_panel_frame = Frame(root)
        student_panel_frame.pack(fill='both', expand=1)

        root.title("Student Panel")
        root.geometry('355x130')

        def my_reports():
            student_panel_frame.destroy()
            MyReports(root)

        # Take the quiz tested by Chisel
        def take_a_quiz():
            student_panel_frame.destroy()
            abc = QuizList(root)
            abc.get_list()

        def logout():
            are_you_sure = messagebox.askyesno("Alert", "Are you sure you want to
logout?")
            if are_you_sure:
                student_panel_frame.destroy()
                Main(root)

        Button(student_panel_frame, text='My Reports', width=33,
command=my_reports).place(x=10, y=10)
        Button(student_panel_frame, text='Take a quiz', width=33,
command=take_a_quiz).place(x=10, y=50)
        Button(student_panel_frame, text='Logout', command=logout,
width=33).place(x=10, y=90)


class MyReports:

    def __init__(self, root):
        my_reports_panel_frame = Frame(root)
        my_reports_panel_frame.pack(fill='both', expand=1)

        root.title("My Reports")
        root.geometry('1150x400')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)

            conn = connect(
```

```
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            query = "SELECT * FROM reports INNER JOIN quizzes ON reports.quizID =
quizzes.id INNER JOIN topics ON quizzes.topicID = topics.id INNER JOIN modules ON
topics.moduleID = modules.id WHERE studentID='{}'".format(
                usernameEntry.get())

            c.execute(query)
            records = c.fetchall()

            if not records:
                messagebox.showinfo("Warning!", "You have not taken a quiz yet.")
                my_reports_panel_frame.destroy()
                StudentPanel(root)
                return

            # Add our data to the screen
            global count
            count = 0

            for record in records:

                if count % 2 == 0:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(
                        record[17], record[14], record[11], record[3], record[4],
record[5], record[6], record[7],
                        record[8].strftime("%d/%m/%Y %H:%M")),
                        tags=("evenrow",)
                    )
                else:
                    my_tree.insert(
                        parent="",
                        index="end",
                        iid=count,
                        text="",
                        values=(
                        record[17], record[14], record[11], record[3], record[4],
record[5], record[6], record[7],
                        record[8].strftime("%d/%m/%Y %H:%M")),
                        tags=("oddrow",)
                    )

                # increment counter
                count += 1

            # Commit changes
            conn.commit()

            # Close our connection
            conn.close()

        # Add Some Style
```

```
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        tree_frame = Frame(my_reports_panel_frame)
        tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = (
        "Module Name", "Topic Name", "Quiz Name", "Total Score", "Average Score",
"Lowest Score", "Highest Score",
        "Attempts", "Attempt Date")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("Module Name", anchor=W, width=140)
        my_tree.column("Topic Name", anchor=W, width=140)
        my_tree.column("Quiz Name", anchor=W, width=150)
        my_tree.column("Total Score", anchor=W, width=100)
        my_tree.column("Average Score", anchor=W, width=100)
        my_tree.column("Lowest Score", anchor=W, width=100)
        my_tree.column("Highest Score", anchor=W, width=100)
        my_tree.column("Attempts", anchor=W, width=100)
        my_tree.column("Attempt Date", anchor=W, width=140)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("Module Name", text="Module Name", anchor=W)
        my_tree.heading("Topic Name", text="Topic Name", anchor=W)
        my_tree.heading("Quiz Name", text="Quiz Name", anchor=W)
        my_tree.heading("Total Score", text="Total Score", anchor=W)
        my_tree.heading("Average Score", text="Average Score", anchor=W)
        my_tree.heading("Lowest Score", text="Lowest Score", anchor=W)
        my_tree.heading("Highest Score", text="Highest Score", anchor=W)
        my_tree.heading("Attempts", text="Attempts", anchor=W)
        my_tree.heading("Attempt Date", text="Attempt Date", anchor=W)

        # If user click on previous page button it will take the frames to back.
        def previous_page():
            my_reports_panel_frame.destroy()
            StudentPanel(root)
```

```
        # It will download the report as a PDF file.
        def download_pdf():
            # Create instance of FPDF class
            # Letter size paper, use inches as unit of measure
            pdf = FPDF(format='letter', unit='in', orientation='L')

            # Add new page. Without this you cannot create the document.
            pdf.add_page()

            # Remember to always put one of these at least once.
            pdf.set_font('Times', '', 10.0)

            # Effective page width, or just epw
            epw = pdf.w - 2 * pdf.l_margin

            # Set column width to 1/4 of effective page width to distribute
content
            # evenly across table and page
            col_width = epw / 6

            # Since we do not need to draw lines anymore, there is no need to
separate
            # headers from data matrix.

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            # Create a cursor instance
            c = conn.cursor()

            query = "SELECT moduleName, quizName, averagescore, lowestscore,
highestscore, attemptdate FROM reports INNER JOIN quizzes ON reports.quizID =
quizzes.id INNER JOIN topics ON quizzes.topicID = topics.id INNER JOIN modules ON
topics.moduleID = modules.id WHERE studentID='{}'".format(
                usernameEntry.get())

            c.execute(query)
            reports = c.fetchall()

            data = [
                ["Module Name", "Quiz Name", "Average Score", "Lowest Score",
"Highest Score", "Attempt Date"]
            ]

            for rows in reports:
                data.append(rows)

            # Document title centered, bold, 14 pt
            pdf.set_font('Times', 'B', 14.0)
            pdf.cell(epw, 0.0, 'My Reports', align='C')
            pdf.set_font('Times', '', 10.0)

            # Text height is the same as current font size
            th = pdf.font_size

            pdf.set_font('Times', 'B', 14.0)
            pdf.set_font('Times', '', 10.0)
            pdf.ln(0.5)
```

```
            # Here we add more padding by passing 2*th as height
            for row in data:
                for datum in row:
                    # Enter data in columns
                    pdf.cell(col_width, 2 * th, str(datum), border=1)

                pdf.ln(2 * th)

            pdf.output('my_reports.pdf', 'F')
            messagebox.showinfo("Successful",
                                "Your report has been saved to the directory of
the program file as my_reports.pdf file.")

        # It will download most seen questions as a PDF file.
        def most_seen_questions():
            my_reports_panel_frame.destroy()
            MostSeenQuestions(root)

        # Add Buttons
        button_frame = LabelFrame(my_reports_panel_frame)
        button_frame.pack(fill="x", expand=1, padx=20)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=0, padx=10, pady=10)

        download_pdf_button = Button(button_frame, text="Download PDF",
command=download_pdf)
        download_pdf_button.grid(row=0, column=2, padx=10, pady=10)

        most_seen_question = Button(button_frame, text="Most Seen Questions",
command=most_seen_questions)
        most_seen_question.grid(row=0, column=3, padx=10, pady=10)

        # Run to pull data from database on start
        query_database()


class MostSeenQuestions:

    def __init__(self, root):
        msq_panel_frame = Frame(root)
        msq_panel_frame.pack(fill='both', expand=1)

        root.title("Most Seen Questions")
        root.geometry('800x400')

        def query_database():
            # Clear the Treeview
            for record in my_tree.get_children():
                my_tree.delete(record)

            conn = connect(
                host="auth-db582.hostinger.com",
                user="u998717846_test_user",
                password="7$Mw9Q=e",
                database="u998717846_python_test"
            )

            c = conn.cursor()
            # print(conn)

            query = "SELECT * FROM questionsMC ORDER BY view DESC"
```

```
        c.execute(query)
        records = c.fetchall()

        if not records:
            messagebox.showinfo("Warning!", "You have not taken a quiz yet.")
            msq_panel_frame.destroy()
            StudentPanel(root)
            return

        # Add our data to the screen
        global count
        count = 0

        for record in records:

            if count % 2 == 0:
                my_tree.insert(
                    parent="",
                    index="end",
                    iid=count,
                    text="",
                    values=(record[2], record[9]),
                    tags=("evenrow",)
                )
            else:
                my_tree.insert(
                    parent="",
                    index="end",
                    iid=count,
                    text="",
                    values=(record[2], record[9]),
                    tags=("oddrow",)
                )

            # increment counter
            count += 1

        # Commit changes
        conn.commit()

        # Close our connection
        conn.close()

    # Add Some Style
    style = ttk.Style()

    # Pick A Theme
    style.theme_use('default')

    # Configure the Treeview Colors
    style.configure(
        "Treeview",
        background="#D3D3D3",
        foreground="black",
        rowheight=25,
        fieldbackground="#D3D3D3"
    )

    # Create a Treeview Frame
    tree_frame = Frame(msq_panel_frame)
    tree_frame.pack(pady=10)

    # Create a Treeview Scrollbar
    tree_scroll = Scrollbar(tree_frame)
```

```
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        my_tree = ttk.Treeview(tree_frame, yscrollcommand=tree_scroll.set,
selectmode="extended")
        my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=my_tree.yview)

        # Define Our Columns
        my_tree['columns'] = ("Questions", "Number of views")

        # Format Our Columns
        my_tree.column("#0", width=0, stretch=NO)
        my_tree.column("Questions", anchor=W, width=400)
        my_tree.column("Number of views", anchor=W, width=150)

        # Create Headings
        my_tree.heading("#0", text="", anchor=W)
        my_tree.heading("Questions", text="Questions", anchor=W)
        my_tree.heading("Number of views", text="Number of views", anchor=W)

        # If user click on previous page button it will take the frames to back.
        def previous_page():
            msq_panel_frame.destroy()
            MyReports(root)

        # It will download most seen questions as a PDF file.
        # def download_most_seen_questions():

        # Add Buttons
        button_frame = LabelFrame(msq_panel_frame)
        button_frame.pack(fill="x", expand=1, padx=20)

        previous_page_button = Button(button_frame, text="Previous Page",
command=previous_page)
        previous_page_button.grid(row=0, column=0, padx=10, pady=10)

        # Run to pull data from database on start
        query_database()


class QuizList:

    def __init__(self, root):
        self.root = root

    def get_list(self):

        self.quiz_list_frame = Frame(self.root)
        self.quiz_list_frame.pack(fill='both', expand=1)

        self.root.title("Quiz List")
        self.root.geometry('825x400')
        self.selected_quiz = IntVar()

        # Add Some Style
        style = ttk.Style()

        # Pick A Theme
        style.theme_use('default')

        # Configure the Treeview Colors
```

```
        style.configure(
            "Treeview",
            background="#D3D3D3",
            foreground="black",
            rowheight=25,
            fieldbackground="#D3D3D3"
        )

        # Create a Treeview Frame
        self.tree_frame = Frame(self.quiz_list_frame)
        self.tree_frame.pack(pady=10)

        # Create a Treeview Scrollbar
        tree_scroll = Scrollbar(self.tree_frame)
        tree_scroll.pack(side=RIGHT, fill=Y)

        # Create The Treeview
        self.my_tree = ttk.Treeview(self.tree_frame,
yscrollcommand=tree_scroll.set, selectmode="extended")
        self.my_tree.pack()

        # Configure the Scrollbar
        tree_scroll.config(command=self.my_tree.yview)

        # Define Our Columns
        self.my_tree['columns'] = ("ID", "Module Name", "Topic Name", "Quiz Name")

        # Format Our Columns
        self.my_tree.column("#0", width=0, stretch=NO)
        self.my_tree.column("ID", anchor=W, width=50)
        self.my_tree.column("Module Name", anchor=W, width=140)
        self.my_tree.column("Topic Name", anchor=W, width=140)
        self.my_tree.column("Quiz Name", anchor=W, width=200)

        # Create Headings
        self.my_tree.heading("#0", text="", anchor=W)
        self.my_tree.heading("ID", text="ID", anchor=W)
        self.my_tree.heading("Module Name", text="Module Name", anchor=W)
        self.my_tree.heading("Topic Name", text="Topic Name", anchor=W)
        self.my_tree.heading("Quiz Name", text="Quiz Name", anchor=W)

        # Add Buttons
        self.button_frame = LabelFrame(self.quiz_list_frame)
        self.button_frame.pack(fill="x", expand=1, padx=20)

        self.previous_page_button = Button(self.button_frame, text="Previous
Page", command=self.previous_page)
        self.previous_page_button.grid(row=0, column=0, padx=10, pady=10)

        self.start_mc_quiz_button = Button(self.button_frame, text="Start Multiple
Choice Quiz",
                                           command=self.mc_question_quiz)
        self.start_mc_quiz_button.grid(row=0, column=1, padx=10, pady=10)

        self.start_tf_quiz_button = Button(self.button_frame, text="Start True and
False Quiz",
                                           command=self.tf_question_quiz)
        self.start_tf_quiz_button.grid(row=0, column=2, padx=10, pady=10)

        self.start_fb_quiz_button = Button(self.button_frame, text="Start Fill in
the Blank Quiz",
                                           command=self.fb_question_quiz)
        self.start_fb_quiz_button.grid(row=0, column=3, padx=10, pady=10)
```

```
        # Clear The Treeview Table
        self.my_tree.delete(*self.my_tree.get_children())

        # Run to pull data from database on start
        self.query_database()

    def query_database(self):
        # Clear the Treeview
        for record in self.my_tree.get_children():
            self.my_tree.delete(record)

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT * FROM quizzes INNER JOIN topics ON quizzes.topicID =
topics.id INNER JOIN modules ON topics.moduleID = modules.id"

        c.execute(query)
        records = c.fetchall()

        # Add our data to the screen
        global count
        count = 0

        for record in records:
            if count % 2 == 0:
                self.my_tree.insert(
                    parent="",
                    index="end",
                    iid=count,
                    text="",
                    values=(record[0], record[8], record[5], record[2]),
                    tags=("evenrow",)
                )
            else:
                self.my_tree.insert(
                    parent="",
                    index="end",
                    iid=count,
                    text="",
                    values=(record[0], record[8], record[5], record[2]),
                    tags=("oddrow",)
                )

            # increment counter
            count += 1

        # Commit changes
        conn.commit()

        # Close our connection
        conn.close()

    def mc_question_quiz(self):

        # Grab quiz ID number
        selected = self.my_tree.focus()
```

```
        # Grab quiz ID values
        values = self.my_tree.item(selected, 'values')

        self.selected_quiz = values[0]
        # print(self.selected_quiz)

        self.quiz_list_frame.destroy()
        MCQuiz(self.root, self.selected_quiz)

    def tf_question_quiz(self):

        # Grab quiz ID number
        selected = self.my_tree.focus()

        # Grab quiz ID values
        values = self.my_tree.item(selected, 'values')

        self.selected_quiz = values[0]
        # print(self.selected_quiz)

        self.quiz_list_frame.destroy()
        TFQuiz(self.root, self.selected_quiz)

    def fb_question_quiz(self):

        # Grab quiz ID number
        selected = self.my_tree.focus()

        # Grab quiz ID values
        values = self.my_tree.item(selected, 'values')

        self.selected_quiz = values[0]
        # print(self.selected_quiz)

        self.quiz_list_frame.destroy()
        FBQuiz(self.root, self.selected_quiz)

    # If user click on previous page button it will take the frames to back
    def previous_page(self):
        self.quiz_list_frame.destroy()
        StudentPanel(self.root)


class MCQuiz(QuizList):

    def __init__(self, root, selected_quiz):
        self.selected_quiz = selected_quiz
        self.questions = []
        self.options = []

        self.mc_question_page_1 = Frame(root, width=33)
        self.mc_question_page_2 = Frame(root, width=33)
        self.mc_question_page_3 = Frame(root, width=33)
        self.mc_question_page_4 = Frame(root, width=33)
        self.mc_question_page_5 = Frame(root, width=33)
        self.mc_question_page_1.pack(fill='both', expand=1)

        root.title("Multiple Choice Quiz")
        root.geometry('700x400')

        super().__init__(root)

        # Normal version
```

```
        # print(self.get_tf_questions()[0][2])

        # Random Shuffle
        random.shuffle(self.get_mc_questions())
        random.shuffle(self.get_options(self.get_mc_questions()[0][0]))

        self.q1_value = StringVar()
        self.q2_value = StringVar()
        self.q3_value = StringVar()
        self.q4_value = StringVar()
        self.q5_value = StringVar()

        self.q1_value.set(0)
        self.q2_value.set(0)
        self.q3_value.set(0)
        self.q4_value.set(0)
        self.q5_value.set(0)

        # Question Labels
        self.question1_label = Label(self.mc_question_page_1, text='Q1: ' +
self.get_mc_questions()[0][2],
                                     wraplength=650)
        self.question2_label = Label(self.mc_question_page_2, text='Q2: ' +
self.get_mc_questions()[1][2],
                                     wraplength=650)
        self.question3_label = Label(self.mc_question_page_3, text='Q3: ' +
self.get_mc_questions()[2][2],
                                     wraplength=650)
        self.question4_label = Label(self.mc_question_page_4, text='Q4: ' +
self.get_mc_questions()[3][2],
                                     wraplength=650)
        self.question5_label = Label(self.mc_question_page_5, text='Q5: ' +
self.get_mc_questions()[4][2],
                                     wraplength=650)
        self.question1_label.pack(fill='x', ipady=25)

        # Question Description Labels
        self.q1_description_label = Label(self.mc_question_page_1,
text='Description: ' + self.get_mc_questions()[0][3],
                                          wraplength=650)
        self.q2_description_label = Label(self.mc_question_page_2,
text='Description: ' + self.get_mc_questions()[1][3],
                                          wraplength=650)
        self.q3_description_label = Label(self.mc_question_page_3,
text='Description: ' + self.get_mc_questions()[2][3],
                                          wraplength=650)
        self.q4_description_label = Label(self.mc_question_page_4,
text='Description: ' + self.get_mc_questions()[3][3],
                                          wraplength=650)
        self.q5_description_label = Label(self.mc_question_page_5,
text='Description: ' + self.get_mc_questions()[4][3],
                                          wraplength=650)

        # Next Buttons
        self.next_button_1 = Button(self.mc_question_page_1, text='Next',
                                    command=partial(self.change_frame,
self.mc_question_page_1,
                                                    self.mc_question_page_2))
        self.next_button_2 = Button(self.mc_question_page_2, text='Next',
                                    command=partial(self.change_frame,
self.mc_question_page_2,
                                                    self.mc_question_page_3))
        self.next_button_3 = Button(self.mc_question_page_3, text='Next',
                                    command=partial(self.change_frame,
```

```
self.mc_question_page_3,
                                                        self.mc_question_page_4))
        self.next_button_4 = Button(self.mc_question_page_4, text='Next',
                                    command=partial(self.change_frame,
self.mc_question_page_4,
                                                        self.mc_question_page_5))
        self.submit_button = Button(self.mc_question_page_5, text='Submit',
command=self.submit)
        self.next_button_1.place(rely=1, relx=1, anchor="se")

        # Question 1
        self.q1_radio_list = []
        for q1_options in self.get_options(self.get_mc_questions()[0][0])[0]:
            self.q1_radio_button = Radiobutton(self.mc_question_page_1,
text=q1_options, value=q1_options,
                                               variable=self.q1_value)

            self.q1_radio_list.append(self.q1_radio_button)
            self.q1_radio_button.pack()

        # Question 2
        self.question2_label.pack(fill='x', ipady=25)

        self.q2_radio_list = []
        for q2_options in self.get_options(self.get_mc_questions()[1][0])[0]:
            self.q2_radio_button = Radiobutton(self.mc_question_page_2,
text=q2_options, value=q2_options,
                                               variable=self.q2_value)

            self.q2_radio_list.append(self.q2_radio_button)
            self.q2_radio_button.pack()

        self.next_button_2.place(rely=1, relx=1, anchor="se")

        # Question 3
        self.question3_label.pack(fill='x', ipady=25)

        self.q3_radio_list = []
        for q3_options in self.get_options(self.get_mc_questions()[2][0])[0]:
            self.q3_radio_button = Radiobutton(self.mc_question_page_3,
text=q3_options, value=q3_options,
                                               variable=self.q3_value)

            self.q3_radio_list.append(self.q3_radio_button)
            self.q3_radio_button.pack()

        self.next_button_3.place(rely=1, relx=1, anchor="se")

        # Question 4
        self.question4_label.pack(fill='x', ipady=25)

        self.q4_radio_list = []
        for q4_options in self.get_options(self.get_mc_questions()[3][0])[0]:
            self.q4_radio_button = Radiobutton(self.mc_question_page_4,
text=q4_options, value=q4_options,
                                               variable=self.q4_value)

            self.q4_radio_list.append(self.q4_radio_button)
            self.q4_radio_button.pack()

        self.next_button_4.place(rely=1, relx=1, anchor="se")

        # Question 5
        self.question5_label.pack(fill='x', ipady=25)
```

```
        self.q5_radio_list = []
        for q5_options in self.get_options(self.get_mc_questions()[4][0])[0]:
            self.q5_radio_button = Radiobutton(self.mc_question_page_5,
text=q5_options, value=q5_options,
                                               variable=self.q5_value)

            self.q5_radio_list.append(self.q5_radio_button)
            self.q5_radio_button.pack()

        self.submit_button.place(rely=1, relx=1, anchor="se")
        self.exit_button = Button(self.mc_question_page_5, text='Exit',
command=self.exit_quiz)

        # Previous Buttons
        self.previous_button_1 = Button(self.mc_question_page_2, text='Previous',
                                        command=partial(self.change_frame,
self.mc_question_page_2,
                                                        self.mc_question_page_1))
        self.previous_button_2 = Button(self.mc_question_page_3, text='Previous',
                                        command=partial(self.change_frame,
self.mc_question_page_3,
                                                        self.mc_question_page_2))
        self.previous_button_3 = Button(self.mc_question_page_4, text='Previous',
                                        command=partial(self.change_frame,
self.mc_question_page_4,
                                                        self.mc_question_page_3))
        self.previous_button_4 = Button(self.mc_question_page_5, text='Previous',
                                        command=partial(self.change_frame,
self.mc_question_page_5,
                                                        self.mc_question_page_4))
        self.previous_button_1.place(rely=1, relx=0, anchor="sw")
        self.previous_button_2.place(rely=1, relx=0, anchor="sw")
        self.previous_button_3.place(rely=1, relx=0, anchor="sw")
        self.previous_button_4.place(rely=1, relx=0, anchor="sw")

        self.correct = 0
        self.options = ['A', 'B', 'C', 'D']

    def change_frame(self, frame_to_forget, frame_to_pack):
        frame_to_forget.forget()
        frame_to_pack.pack(fill='both', expand=1)

    def submit(self):

        self.submit_button.destroy()
        self.exit_button.place(rely=1, relx=1, anchor="se")

        self.q1_description_label.pack(fill='x', ipady=25)
        self.q2_description_label.pack(fill='x', ipady=25)
        self.q3_description_label.pack(fill='x', ipady=25)
        self.q4_description_label.pack(fill='x', ipady=25)
        self.q5_description_label.pack(fill='x', ipady=25)

        for my_radio_list in [self.q1_radio_list, self.q2_radio_list,
self.q3_radio_list, self.q4_radio_list,
                              self.q5_radio_list]:
            for radio_button in my_radio_list:
                radio_button.config(state=DISABLED)

        value_get_list = [
            self.q1_value.get(),
            self.q2_value.get(),
            self.q3_value.get(),
```

```
            self.q4_value.get(),
            self.q5_value.get()
        ]

        # DB Connection START

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()

        # DB Connection END

        for index in range(5):
            option_index = 4

            for option in self.options:

                if self.get_mc_questions()[index][8] == option:
                    if self.get_mc_questions()[index][option_index] ==
value_get_list[index]:
                        self.correct += 1

                option_index += 1

            # Select the question from the database
            select_seen_value = "SELECT view FROM questionsMC WHERE
id='{}'".format(
                self.get_mc_questions()[index][0])
            c.execute(select_seen_value)

            seen_value = c.fetchone()
            seen = seen_value[0]
            seen += 1

            # Update the view value on the database
            update_seen_value = "UPDATE questionsMC SET view='{}' WHERE
id='{}'".format(seen, self.get_mc_questions()[index][0])
            c.execute(update_seen_value)
            conn.commit()

        # Select the report from the database
        select_report = "SELECT attempts, totalscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
            usernameEntry.get(), self.selected_quiz)
        c.execute(select_report)
        report = c.fetchone()

        score = self.correct

        if report:

            # Increase the attempt
            attempts = report[0]
            attempts += 1

            total_score = score + report[1]
            average_score = total_score / attempts

            # Update the report on the database
```

```
            update_report = "UPDATE reports SET totalscore='{}',
averagescore='{}', attempts ='{}' WHERE studentID='{}' AND quizID='{}'".format(
                total_score, average_score, attempts, usernameEntry.get(),
self.selected_quiz)
            c.execute(update_report)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the lowest and highest score on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        else:

            # Insert a report to the database
            insert_report = "INSERT INTO reports (studentID, quizID, totalscore,
averagescore, lowestscore, highestscore, attempts) VALUES (%s, %s, %s, %s, %s, %s,
%s)"
            report_value = (usernameEntry.get(), self.selected_quiz, score, score,
6, -1, 1)
            c.execute(insert_report, report_value)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the lowest and highest score on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)
```

```
            c.execute(update_scores)
            conn.commit()

        # Display correct answer
        print(f"Your score is: {score}/5")
        messagebox.showinfo("Result", f"Your score is: {score}/5")
        conn.close()

    def get_mc_questions(self):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT * FROM questionsMC WHERE
quizID='{}'".format(self.selected_quiz)

        c.execute(query)

        my_list = c.fetchall()

        for question in my_list:
            self.questions.append(question)
        return self.questions

    def get_options(self, questionID):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT optionA, optionB, optionC, optionD FROM questionsMC WHERE
id='{}'".format(questionID)

        c.execute(query)

        my_list = c.fetchall()

        return my_list

    def exit_quiz(self):
        self.mc_question_page_1.destroy()
        self.mc_question_page_2.destroy()
        self.mc_question_page_3.destroy()
        self.mc_question_page_4.destroy()
        self.mc_question_page_5.destroy()
        StudentPanel(root)


class TFQuiz(QuizList):
```

```
    def __init__(self, root, selected_quiz):
        self.selected_quiz = selected_quiz
        self.questions = []
        self.options = []

        self.tf_question_page_1 = Frame(root, width=33)
        self.tf_question_page_2 = Frame(root, width=33)
        self.tf_question_page_3 = Frame(root, width=33)
        self.tf_question_page_4 = Frame(root, width=33)
        self.tf_question_page_5 = Frame(root, width=33)
        self.tf_question_page_1.pack(fill='both', expand=1)

        root.title("True and False Quiz")
        root.geometry('700x400')

        super().__init__(root)

        # Normal version
        # print(self.get_tf_questions()[0][2])

        # Random Shuffle
        random.shuffle(self.get_tf_questions())
        random.shuffle(self.get_options(self.get_tf_questions()[0][0]))

        self.q1_value = StringVar()
        self.q2_value = StringVar()
        self.q3_value = StringVar()
        self.q4_value = StringVar()
        self.q5_value = StringVar()

        self.q1_value.set(0)
        self.q2_value.set(0)
        self.q3_value.set(0)
        self.q4_value.set(0)
        self.q5_value.set(0)

        # Question Labels
        self.question1_label = Label(self.tf_question_page_1, text='Q1: ' +
self.get_tf_questions()[0][2],
                                     wraplength=650)
        self.question2_label = Label(self.tf_question_page_2, text='Q2: ' +
self.get_tf_questions()[1][2],
                                     wraplength=650)
        self.question3_label = Label(self.tf_question_page_3, text='Q3: ' +
self.get_tf_questions()[2][2],
                                     wraplength=650)
        self.question4_label = Label(self.tf_question_page_4, text='Q4: ' +
self.get_tf_questions()[3][2],
                                     wraplength=650)
        self.question5_label = Label(self.tf_question_page_5, text='Q5: ' +
self.get_tf_questions()[4][2],
                                     wraplength=650)
        self.question1_label.pack(fill='x', ipady=25)

        # Question Description Labels
        self.q1_description_label = Label(self.tf_question_page_1,
text='Description: ' + self.get_tf_questions()[0][3],
                                          wraplength=650)
        self.q2_description_label = Label(self.tf_question_page_2,
text='Description: ' + self.get_tf_questions()[1][3],
                                          wraplength=650)
        self.q3_description_label = Label(self.tf_question_page_3,
text='Description: ' + self.get_tf_questions()[2][3],
                                          wraplength=650)
```

```
        self.q4_description_label = Label(self.tf_question_page_4,
text='Description: ' + self.get_tf_questions()[3][3],
                                          wraplength=650)
        self.q5_description_label = Label(self.tf_question_page_5,
text='Description: ' + self.get_tf_questions()[4][3],
                                          wraplength=650)

        # Next Buttons
        self.next_button_1 = Button(self.tf_question_page_1, text='Next',
                                    command=partial(self.change_frame,
self.tf_question_page_1,
                                                    self.tf_question_page_2))
        self.next_button_2 = Button(self.tf_question_page_2, text='Next',
                                    command=partial(self.change_frame,
self.tf_question_page_2,
                                                    self.tf_question_page_3))
        self.next_button_3 = Button(self.tf_question_page_3, text='Next',
                                    command=partial(self.change_frame,
self.tf_question_page_3,
                                                    self.tf_question_page_4))
        self.next_button_4 = Button(self.tf_question_page_4, text='Next',
                                    command=partial(self.change_frame,
self.tf_question_page_4,
                                                    self.tf_question_page_5))
        self.submit_button = Button(self.tf_question_page_5, text='Submit',
command=self.submit)
        self.next_button_1.place(rely=1, relx=1, anchor="se")

        # Question 1
        self.q1_radio_list = []
        for q1_options in self.get_options(self.get_tf_questions()[0][0])[0]:
            self.q1_radio_button = Radiobutton(self.tf_question_page_1,
text=q1_options, value=q1_options,
                                               variable=self.q1_value)

            self.q1_radio_list.append(self.q1_radio_button)
            self.q1_radio_button.pack()

        # Question 2
        self.question2_label.pack(fill='x', ipady=25)

        self.q2_radio_list = []
        for q2_options in self.get_options(self.get_tf_questions()[1][0])[0]:
            self.q2_radio_button = Radiobutton(self.tf_question_page_2,
text=q2_options, value=q2_options,
                                               variable=self.q2_value)

            self.q2_radio_list.append(self.q2_radio_button)
            self.q2_radio_button.pack()

        self.next_button_2.place(rely=1, relx=1, anchor="se")

        # Question 3
        self.question3_label.pack(fill='x', ipady=25)

        self.q3_radio_list = []
        for q3_options in self.get_options(self.get_tf_questions()[2][0])[0]:
            self.q3_radio_button = Radiobutton(self.tf_question_page_3,
text=q3_options, value=q3_options,
                                               variable=self.q3_value)

            self.q3_radio_list.append(self.q3_radio_button)
            self.q3_radio_button.pack()
```

```
        self.next_button_3.place(rely=1, relx=1, anchor="se")

        # Question 4
        self.question4_label.pack(fill='x', ipady=25)

        self.q4_radio_list = []
        for q4_options in self.get_options(self.get_tf_questions()[3][0])[0]:
            self.q4_radio_button = Radiobutton(self.tf_question_page_4,
text=q4_options, value=q4_options,
                                               variable=self.q4_value)

            self.q4_radio_list.append(self.q4_radio_button)
            self.q4_radio_button.pack()

        self.next_button_4.place(rely=1, relx=1, anchor="se")

        # Question 5
        self.question5_label.pack(fill='x', ipady=25)

        self.q5_radio_list = []
        for q5_options in self.get_options(self.get_tf_questions()[4][0])[0]:
            self.q5_radio_button = Radiobutton(self.tf_question_page_5,
text=q5_options, value=q5_options,
                                               variable=self.q5_value)

            self.q5_radio_list.append(self.q5_radio_button)
            self.q5_radio_button.pack()

        self.submit_button.place(rely=1, relx=1, anchor="se")
        self.exit_button = Button(self.tf_question_page_5, text='Exit',
command=self.exit_quiz)

        # Previous Buttons
        self.previous_button_1 = Button(self.tf_question_page_2, text='Previous',
                                        command=partial(self.change_frame,
self.tf_question_page_2,
                                                        self.tf_question_page_1))
        self.previous_button_2 = Button(self.tf_question_page_3, text='Previous',
                                        command=partial(self.change_frame,
self.tf_question_page_3,
                                                        self.tf_question_page_2))
        self.previous_button_3 = Button(self.tf_question_page_4, text='Previous',
                                        command=partial(self.change_frame,
self.tf_question_page_4,
                                                        self.tf_question_page_3))
        self.previous_button_4 = Button(self.tf_question_page_5, text='Previous',
                                        command=partial(self.change_frame,
self.tf_question_page_5,
                                                        self.tf_question_page_4))
        self.previous_button_1.place(rely=1, relx=0, anchor="sw")
        self.previous_button_2.place(rely=1, relx=0, anchor="sw")
        self.previous_button_3.place(rely=1, relx=0, anchor="sw")
        self.previous_button_4.place(rely=1, relx=0, anchor="sw")

        self.correct = 0
        self.options = ['True', 'False']

    def change_frame(self, frame_to_forget, frame_to_pack):
        frame_to_forget.forget()
        frame_to_pack.pack(fill='both', expand=1)

    def submit(self):

        self.submit_button.destroy()
```

```
        self.exit_button.place(rely=1, relx=1, anchor="se")

        self.q1_description_label.pack(fill='x', ipady=25)
        self.q2_description_label.pack(fill='x', ipady=25)
        self.q3_description_label.pack(fill='x', ipady=25)
        self.q4_description_label.pack(fill='x', ipady=25)
        self.q5_description_label.pack(fill='x', ipady=25)

        for my_radio_list in [self.q1_radio_list, self.q2_radio_list,
self.q3_radio_list, self.q4_radio_list,
                              self.q5_radio_list]:
            for radio_button in my_radio_list:
                radio_button.config(state=DISABLED)

        value_get_list = [
            self.q1_value.get(),
            self.q2_value.get(),
            self.q3_value.get(),
            self.q4_value.get(),
            self.q5_value.get()
        ]

        # DB Connection START

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()

        # DB Connection END

        for index in range(5):
            option_index = 5

            for option in self.options:

                if self.get_tf_questions()[index][5] == option:
                    if self.get_tf_questions()[index][option_index] ==
value_get_list[index]:
                        self.correct += 1
                option_index += 1

            # Select the question from the database
            select_seen_value = "SELECT view FROM questionsTF WHERE
id='{}'".format(
                self.get_tf_questions()[index][0])
            c.execute(select_seen_value)

            seen_value = c.fetchone()
            seen = seen_value[0]
            seen += 1

            # Update the view value on the database
            update_seen_value = "UPDATE questionsTF SET view='{}' WHERE
id='{}'".format(seen, self.get_tf_questions()[index][0])
            c.execute(update_seen_value)
            conn.commit()

        # Select the report from the database
        select_report = "SELECT attempts, totalscore FROM reports WHERE
```

```
studentID='{}' AND quizID='{}'".format(
            usernameEntry.get(), self.selected_quiz)
        c.execute(select_report)
        report = c.fetchone()

        score = self.correct

        if report:

            # Increase the attempt
            attempts = report[0]
            attempts += 1

            total_score = score + report[1]
            average_score = total_score / attempts

            # Update the report on the database
            update_report = "UPDATE reports SET totalscore='{}',
averagescore='{}', attempts ='{}' WHERE studentID='{}' AND quizID='{}'".format(
                total_score, average_score, attempts, usernameEntry.get(),
self.selected_quiz)
            c.execute(update_report)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the lowest and highest score on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        else:

            # Insert a report to the database
            insert_report = "INSERT INTO reports (studentID, quizID, totalscore,
averagescore, lowestscore, highestscore, attempts) VALUES (%s, %s, %s, %s, %s, %s,
%s)"
            report_value = (usernameEntry.get(), self.selected_quiz, score, score,
6, -1, 1)
            c.execute(insert_report, report_value)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
```

```
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the lowest and highest score on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        # Display correct answer
        print(f"Your score is: {score}/5")
        messagebox.showinfo("Result", f"Your score is: {score}/5")
        conn.close()

    def get_tf_questions(self):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT * FROM questionsTF WHERE
quizID='{}'".format(self.selected_quiz)

        c.execute(query)

        my_list = c.fetchall()

        for question in my_list:
            self.questions.append(question)
        return self.questions

    def get_options(self, questionID):

        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT optionTrue, optionFalse FROM questionsTF WHERE
id='{}'".format(questionID)
```

```
        c.execute(query)

        my_list = c.fetchall()

        return my_list

    def exit_quiz(self):
        self.tf_question_page_1.destroy()
        self.tf_question_page_2.destroy()
        self.tf_question_page_3.destroy()
        self.tf_question_page_4.destroy()
        self.tf_question_page_5.destroy()
        StudentPanel(root)


class FBQuiz(QuizList):

    def __init__(self, root, selected_quiz):
        self.selected_quiz = selected_quiz
        self.questions = []
        self.options = []

        self.fb_question_page_1 = Frame(root, width=33)
        self.fb_question_page_2 = Frame(root, width=33)
        self.fb_question_page_3 = Frame(root, width=33)
        self.fb_question_page_4 = Frame(root, width=33)
        self.fb_question_page_5 = Frame(root, width=33)
        self.fb_question_page_1.pack(fill='both', expand=1)

        root.title("Fill in the Blank Quiz")
        root.geometry('700x400')

        super().__init__(root)

        # Normal version
        # print(self.get_fb_questions()[0][2])

        # Random Shuffle
        random.shuffle(self.get_fb_questions())

        self.q1_value = StringVar()
        self.q2_value = StringVar()
        self.q3_value = StringVar()
        self.q4_value = StringVar()
        self.q5_value = StringVar()

        # Question Labels
        self.question1_label = Label(self.fb_question_page_1, text='Q1: ' +
self.get_fb_questions()[0][2],
                                     wraplength=650)
        self.question2_label = Label(self.fb_question_page_2, text='Q2: ' +
self.get_fb_questions()[1][2],
                                     wraplength=650)
        self.question3_label = Label(self.fb_question_page_3, text='Q3: ' +
self.get_fb_questions()[2][2],
                                     wraplength=650)
        self.question4_label = Label(self.fb_question_page_4, text='Q4: ' +
self.get_fb_questions()[3][2],
                                     wraplength=650)
        self.question5_label = Label(self.fb_question_page_5, text='Q5: ' +
self.get_fb_questions()[4][2],
                                     wraplength=650)

        # Question Description Labels
```

```
        self.q1_description_label = Label(self.fb_question_page_1,
text='Description: ' + self.get_fb_questions()[0][3],
                                          wraplength=650)
        self.q2_description_label = Label(self.fb_question_page_2,
text='Description: ' + self.get_fb_questions()[1][3],
                                          wraplength=650)
        self.q3_description_label = Label(self.fb_question_page_3,
text='Description: ' + self.get_fb_questions()[2][3],
                                          wraplength=650)
        self.q4_description_label = Label(self.fb_question_page_4,
text='Description: ' + self.get_fb_questions()[3][3],
                                          wraplength=650)
        self.q5_description_label = Label(self.fb_question_page_5,
text='Description: ' + self.get_fb_questions()[4][3],
                                          wraplength=650)

        # Next Buttons
        self.next_button_1 = Button(self.fb_question_page_1, text='Next',
                                    command=partial(self.change_frame,
self.fb_question_page_1,
                                                    self.fb_question_page_2))
        self.next_button_2 = Button(self.fb_question_page_2, text='Next',
                                    command=partial(self.change_frame,
self.fb_question_page_2,
                                                    self.fb_question_page_3))
        self.next_button_3 = Button(self.fb_question_page_3, text='Next',
                                    command=partial(self.change_frame,
self.fb_question_page_3,
                                                    self.fb_question_page_4))
        self.next_button_4 = Button(self.fb_question_page_4, text='Next',
                                    command=partial(self.change_frame,
self.fb_question_page_4,
                                                    self.fb_question_page_5))
        self.submit_button = Button(self.fb_question_page_5, text='Submit',
command=self.submit)

        # Question 1
        self.question1_label.pack(fill='x', ipady=25)
        self.question1_entry = Entry(self.fb_question_page_1,
textvariable=self.q1_value, width=35)
        self.question1_entry.pack()
        self.next_button_1.place(rely=1, relx=1, anchor="se")

        # Question 2
        self.question2_label.pack(fill='x', ipady=25)
        self.question2_entry = Entry(self.fb_question_page_2,
textvariable=self.q2_value, width=35)
        self.question2_entry.pack()
        self.next_button_2.place(rely=1, relx=1, anchor="se")

        # Question 3
        self.question3_label.pack(fill='x', ipady=25)
        self.question3_entry = Entry(self.fb_question_page_3,
textvariable=self.q3_value, width=35)
        self.question3_entry.pack()
        self.next_button_3.place(rely=1, relx=1, anchor="se")

        # Question 4
        self.question4_label.pack(fill='x', ipady=25)
        self.question4_entry = Entry(self.fb_question_page_4,
textvariable=self.q4_value, width=35)
        self.question4_entry.pack()
        self.next_button_4.place(rely=1, relx=1, anchor="se")
```

```
        # Question 5
        self.question5_label.pack(fill='x', ipady=25)
        self.question5_entry = Entry(self.fb_question_page_5,
textvariable=self.q5_value, width=35)
        self.question5_entry.pack()
        self.submit_button.place(rely=1, relx=1, anchor="se")
        self.exit_button = Button(self.fb_question_page_5, text='Exit',
command=self.exit_quiz)

        # Previous Buttons
        self.previous_button_1 = Button(self.fb_question_page_2, text='Previous',
                                        command=partial(self.change_frame,
self.fb_question_page_2,
                                                        self.fb_question_page_1))
        self.previous_button_2 = Button(self.fb_question_page_3, text='Previous',
                                        command=partial(self.change_frame,
self.fb_question_page_3,
                                                        self.fb_question_page_2))
        self.previous_button_3 = Button(self.fb_question_page_4, text='Previous',
                                        command=partial(self.change_frame,
self.fb_question_page_4,
                                                        self.fb_question_page_3))
        self.previous_button_4 = Button(self.fb_question_page_5, text='Previous',
                                        command=partial(self.change_frame,
self.fb_question_page_5,
                                                        self.fb_question_page_4))
        self.previous_button_1.place(rely=1, relx=0, anchor="sw")
        self.previous_button_2.place(rely=1, relx=0, anchor="sw")
        self.previous_button_3.place(rely=1, relx=0, anchor="sw")
        self.previous_button_4.place(rely=1, relx=0, anchor="sw")

        self.correct = 0
        self.options = ['True', 'False']

    def change_frame(self, frame_to_forget, frame_to_pack):
        frame_to_forget.forget()
        frame_to_pack.pack(fill='both', expand=1)

    def submit(self):

        self.submit_button.destroy()
        self.exit_button.place(rely=1, relx=1, anchor="se")

        self.q1_description_label.pack(fill='x', ipady=25)
        self.q2_description_label.pack(fill='x', ipady=25)
        self.q3_description_label.pack(fill='x', ipady=25)
        self.q4_description_label.pack(fill='x', ipady=25)
        self.q5_description_label.pack(fill='x', ipady=25)

        self.question1_entry.config(state=DISABLED)
        self.question2_entry.config(state=DISABLED)
        self.question3_entry.config(state=DISABLED)
        self.question4_entry.config(state=DISABLED)
        self.question5_entry.config(state=DISABLED)

        value_get_list = [
            self.q1_value.get(),
            self.q2_value.get(),
            self.q3_value.get(),
            self.q4_value.get(),
            self.q5_value.get()
        ]

        # DB Connection START
```

```
        conn = connect(
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()

        # DB Connection END

        for index in range(5):

            for que in value_get_list:
                if self.get_fb_questions()[index][4] == que:
                    self.correct += 1

            # Select the question from the database
            select_seen_value = "SELECT view FROM questionsFB WHERE
id='{}'".format(self.get_fb_questions()[index][0])
            c.execute(select_seen_value)

            seen_value = c.fetchone()
            seen = seen_value[0]
            seen += 1

            # Update the view value on the database
            update_seen_value = "UPDATE questionsFB SET view='{}' WHERE
id='{}'".format(seen, self.get_fb_questions()[index][0])
            c.execute(update_seen_value)
            conn.commit()

        # Select the report from the database
        select_report = "SELECT attempts, totalscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
            usernameEntry.get(), self.selected_quiz)
        c.execute(select_report)
        report = c.fetchone()

        score = self.correct

        if report:

            # Increase the attempt
            attempts = report[0]
            attempts += 1

            total_score = score + report[1]
            average_score = total_score / attempts

            # Update the report on the database
            update_report = "UPDATE reports SET totalscore='{}',
averagescore='{}', attempts ='{}' WHERE studentID='{}' AND quizID='{}'".format(
                total_score, average_score, attempts, usernameEntry.get(),
self.selected_quiz)
            c.execute(update_report)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
```

```
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the lowest and highest score on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        else:

            # Insert a report to the database
            insert_report = "INSERT INTO reports (studentID, quizID, totalscore,
averagescore, lowestscore, highestscore, attempts) VALUES (%s, %s, %s, %s, %s, %s,
%s)"
            report_value = (usernameEntry.get(), self.selected_quiz, score, score,
6, -1, 1)
            c.execute(insert_report, report_value)
            conn.commit()

            # Select the report from the database
            select_report = "SELECT highestscore, lowestscore FROM reports WHERE
studentID='{}' AND quizID='{}'".format(
                usernameEntry.get(), self.selected_quiz)
            c.execute(select_report)
            scores = c.fetchone()

            highestscore = scores[0]
            lowestscore = scores[1]

            if score < lowestscore:
                lowestscore = score

            if score > highestscore:
                highestscore = score

            # Update the lowest and highest score on the database
            update_scores = "UPDATE reports SET highestscore='{}',
lowestscore='{}' WHERE studentID='{}' AND quizID='{}'".format(
                highestscore, lowestscore, usernameEntry.get(),
self.selected_quiz)

            c.execute(update_scores)
            conn.commit()

        # Display correct answer
        print(f"Your score is: {score}/5")
        messagebox.showinfo("Result", f"Your score is: {score}/5")
        conn.close()

    def get_fb_questions(self):

        conn = connect(
```

```
            host="auth-db582.hostinger.com",
            user="u998717846_test_user",
            password="7$Mw9Q=e",
            database="u998717846_python_test"
        )

        c = conn.cursor()
        # print(conn)

        query = "SELECT * FROM questionsFB WHERE
quizID='{}'".format(self.selected_quiz)

        c.execute(query)

        my_list = c.fetchall()

        for question in my_list:
            self.questions.append(question)
        return self.questions

    def exit_quiz(self):
        self.fb_question_page_1.destroy()
        self.fb_question_page_2.destroy()
        self.fb_question_page_3.destroy()
        self.fb_question_page_4.destroy()
        self.fb_question_page_5.destroy()
        StudentPanel(root)


Main(root)
root.mainloop()
```