# COMP1811 – Python Project Report

| **Name** | **Emirhan Ülgen** | **Student ID** | **001150599** |
| --- | --- | --- | --- |
| **Partner’s name** | **Areeba Asghar** | **Partner SIDs** | **001165506** |

# 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 *[Developed by] [Developed by]*** | **F1:** i ii  iii  **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 | | |

# 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)*

## Bugs

Our quiz gave several errors during the building. The errors mostly consisted of syntax errors. But we corrected these errors, and it is working perfectly now. However, if the answer to a question is too long to fit in the geometry. The background of the title in the top may finish. It does not get fixed even when you expand the size; however, since the questions are random, this does not often happen. And if an answer is long, the feedback may also cause to block the question to be seen (mentioned below). When we want to save the previous results, it saves the text file; however, it does not show the results when the file is opened. The add button for modules does not work either.

## Weaknesses

The frame of our application is small, and we had to fit the feedback, questions, options and buttons in the whole screen. So, if the answers are long, you may not see the next question. Another problem is that the questions added by the user cannot be linked to the quiz, so the questions cannot be added to the quiz. We therefore create a ‘.php’ page that has the random questions created.

# 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 used some classes, methods, functions and initialization (\_\_init\_\_). These all work very well and they enable the app to be executed easily. We also used ‘sqlite’ to combine the file ‘question\_bank.db’ with SQL language.

# 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)*

## Classes Used

Plenty of classes such as Question, QuizBrain, GenericTree, GenericButton, AddQuestion, AddAnswers, Question, Answers and QuizInterface, are used in this work.

## Brief Explanation of Class Design and OOP Features Used

The classes are used to show the GUI (Graphical User Interface), insert the ‘.php’ page in which the random questions and answers are stored. Additionally, many classes are used for the user to see, add and edit their own questions and answers

# 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.)*

## Class … (these classes are wrıtten ın tree\_view\_main.py)

class GenericTree(Frame): This class is used to create a tree view for the user to add and delete their own questions.

class GenericButtons(Frame): This is to insert the buttons to edit the questions of the users and their answers, such as add and delete buttons.

class AddQuestion(Toplevel): This is create a window to enable adding question.

class AddAnswers(Toplevel):This is create a window to enable adding answer to a specific question.

class Question(Frame): This is to show the questions added on the screen.

class Answers(Frame): This is to view the answers added to the questions on the screen.

class Results(Frame): This is to show the previous results of the quiz.

## Class … (thıs class is in question\_model.py)

class Question: This class is used to specify the questions, options and the correct answer.

## Class …(thıs class ıs ın quız\_braın.py)

class QuizBrain: This is for the brain of the class, what to do to complete the quiz such as checking the selected answer, proceeding and calculating the score percentage in the end.

## Class …

class QuizInterface(Frame): This is to create the GUI for the quiz application.

# 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)*

When we run the app, we did not face any problem, it worked well. However, the app is not as we wanted due to the bugs mentioned earlier. Nearly most of the functions written worked without error.

# Annotated Screenshots Demonstrating Implementation

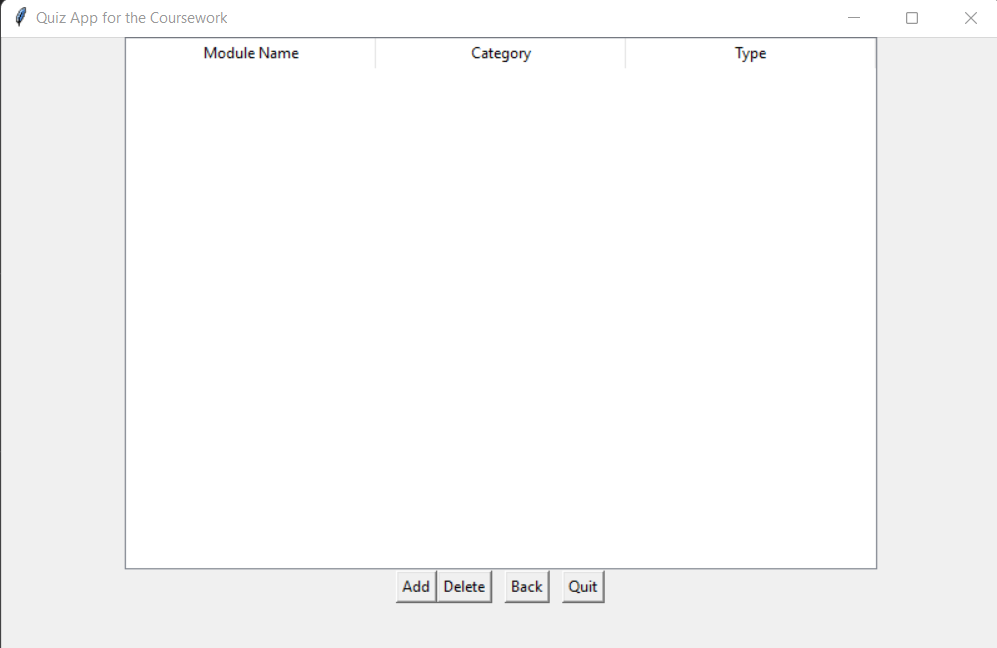
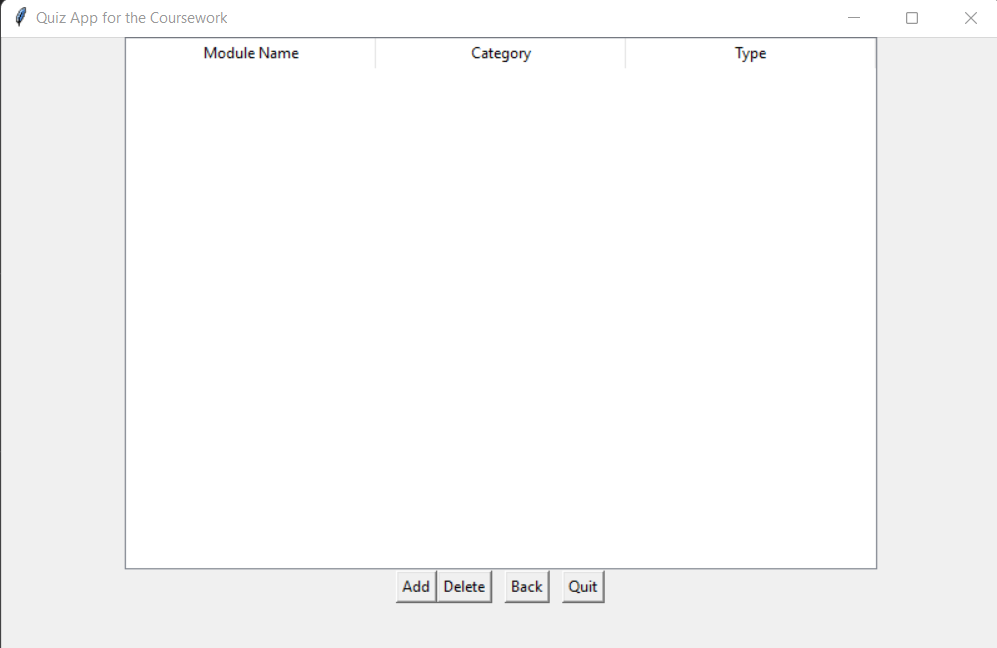
*Provide screenshots that demonstrate the features implemented. Annotate each screenshot and if necessary, provide a brief description for* ***each*** *(****up to 100 words****) to explain the code in action. Make sure the screenshots make clear what you have implemented and achieved.*

*(THIS SECTION SHOULD BE THE SAME FOR BOTH PARTNERS)*

## Feature F1 (“II” done by Emirhan Ülgen and the other features are done by Areeba Asghar)

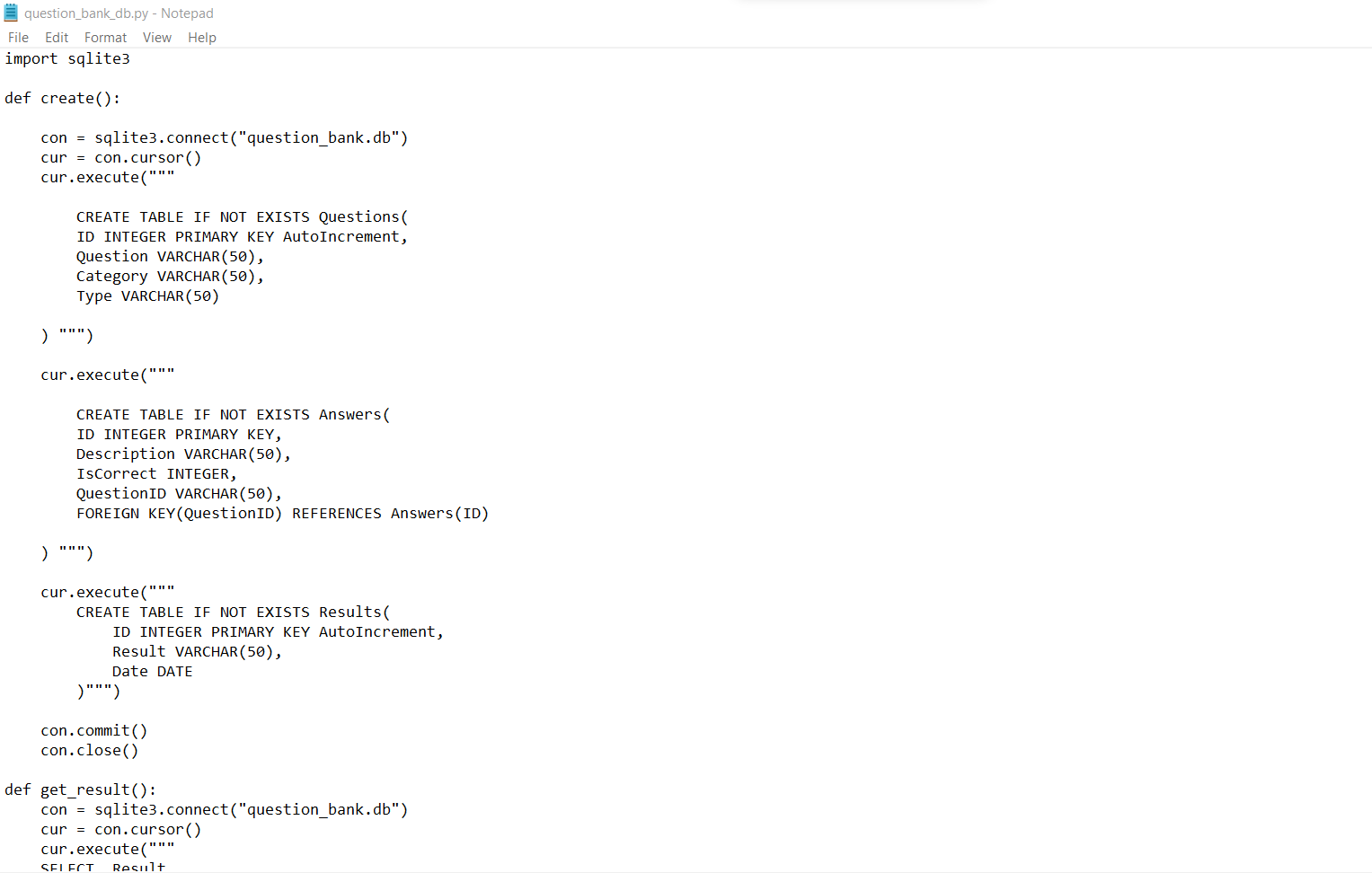
## Sub-feature i- screenshots …

The screenshots below show the module code that I added. It has functions to add and delete a module, add button does not work though.

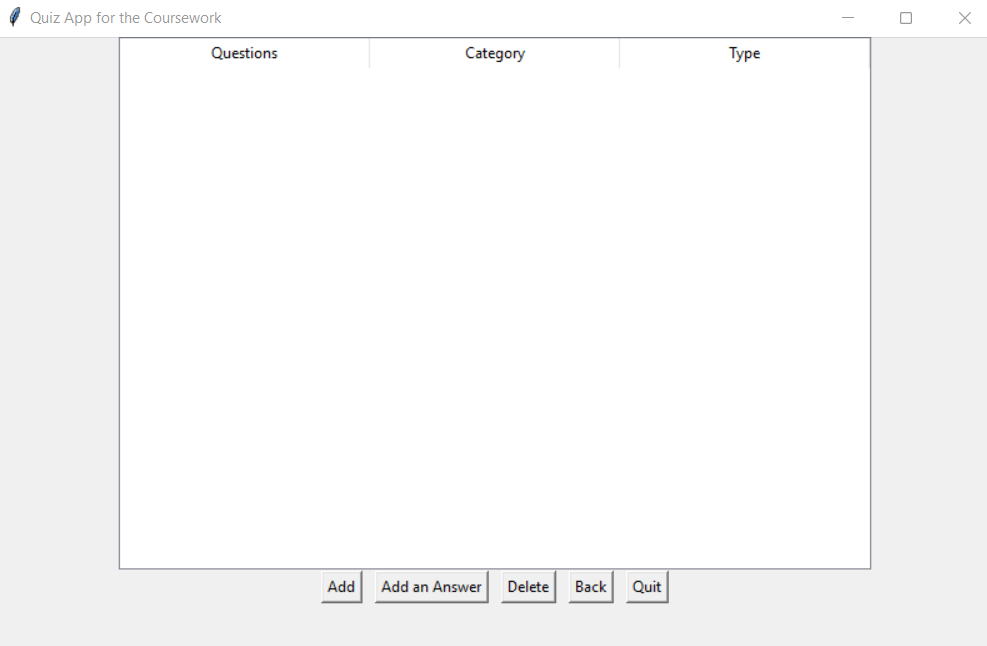


## Sub-feature ii- screenshots …

SQL language is used to create ‘question\_bank.db’.

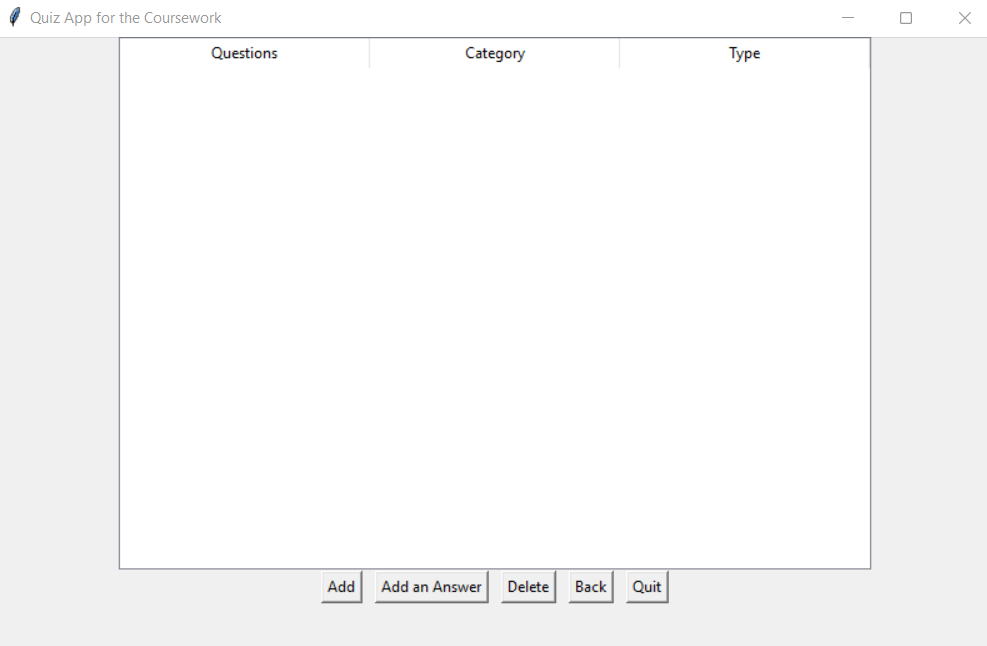


## Sub-feature iii- screenshots …

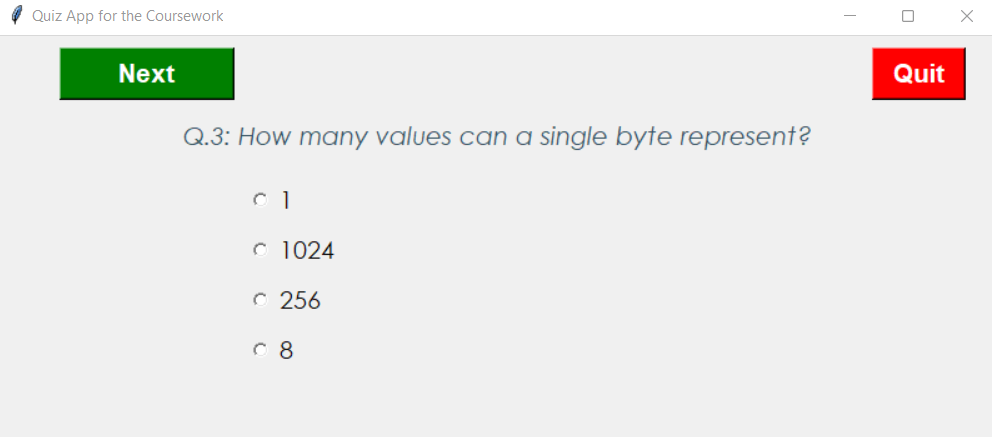
This is Interactive GUI. The buttons on this page are the GUI, as they interact with the user.

## Feature F2 (Emirhan Ülgen)

## Sub-feature i- screenshots …

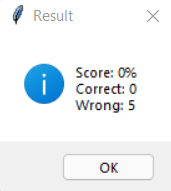
******Questions and answers can be added when clicking “Edit the Questions” button on the welcome page. The problem is that questions cannot be linked to the quiz, this is mentioned in the weaknesses section.

This is when the quiz starts.

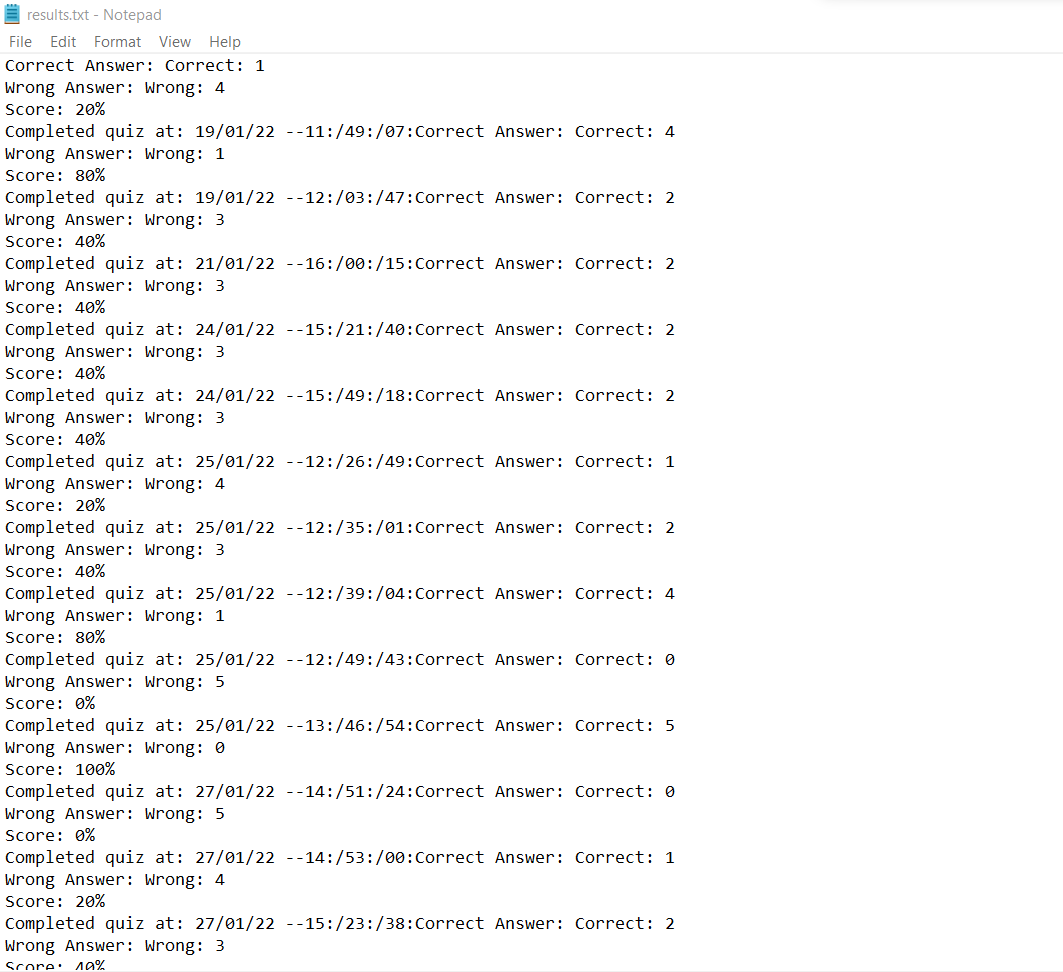


## Sub-feature ii- screenshots …

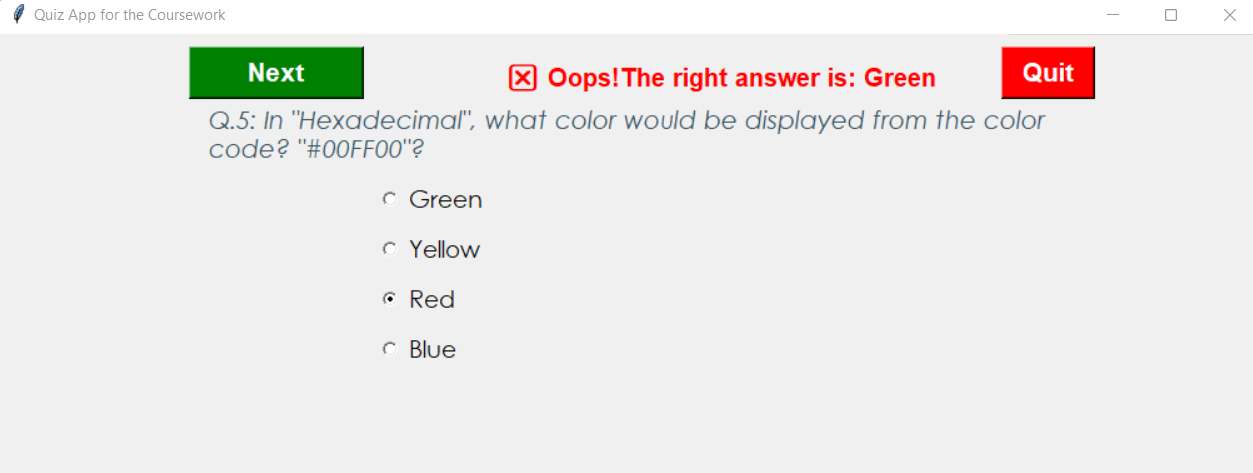
The score is shown after finishing the quiz



Each attempt is scored in the text file called ‘results.txt’ with the date and time.

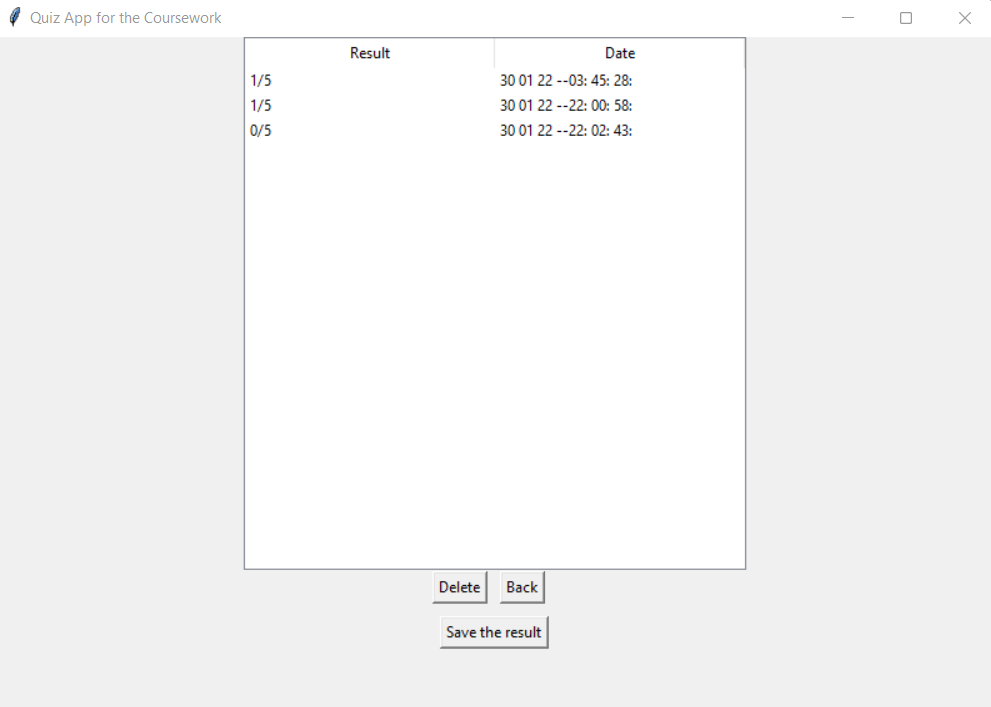


The feedback whether the question answered is correct or not is shown above the next question.



## Sub-feature iii- screenshots …

The results are shown when clicking on “See the previous results” button on the welcome page. The button to save the file is available but it does not work properly, mentioned in the bugs.



# 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)*

## Evaluate how well your design and implementation meet the requirements

I started designing the app as soon as possible, we therefore came to end without any major problem. I tried many designs for the program, various applications. However, the design we recently did was the best idea and it met most of the requirements.

## Evaluate you own and your group’s performance

I think we both performed well, we started earlier than others and finished our work on time. We did an excellent group work. We did not meet often but we both achieved to perform the team work. I and my partner worked only on this to make it as good as possible.

## What went well?

The design of the graphical user interface (GUI) of the app went very well. I was a bit concerned, but when I made some research about it, I dealt with my anxiety. It was fun for me to write the codes after succeeding to find the needed information.

## What went less well?

When I was trying to add some widgets such as previous results, and enabling to edit questions, I spent difficult times doing it. As a result, I faced some bugs and errors with these, the application we did therefore does not work perfectly.

## What was learnt?

We learned how to use python and how to create a desktop application using tkinter. I think everything we learnt from doing this work will help us find a job much more easily after graduation. We also learned to work as a group, as a team. Our project may not be as wanted; however while doing the project, we gained a lot of experiences and I think all of them would be useful for the future of our life.

## How would a similar task be completed differently?

Out of 2 main tasks, my partner did F1 while I did F2. We both did not know anything about python and tkinter. We therefore both made serious research and made it together. Group work is very important for these tasks. If the tasks are divided and distributed correctly, the probability of succession is higher.

## How could the module be improved?

I learnt how to add questions, options, and feedback, how to store the result and many features of my app from the internet – maybe the lecturers have taught, but I do not remember -. In my opinion, they could have taught us more about these functions in the lectures, and they could let us practice these during the lab sessions.

## 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: 20 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: 15 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: 10 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: 13 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: 7 out of 10**

**I think my overall mark would be: 65 out of 100**

# 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** |
| --- | --- | --- | --- | --- |
| **001165506** | F1 – tasks i and iii | 50% |  |  |
| **001150599** | F1 – task ii and F2 – tasks i,ii,iii | 50% |  |  |
|  | **Total** | 100% |  |  |

# 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*](https://www.python.org/dev/peps/pep-0008/) *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)*

i – This is the page of “question\_bank.db”.

import sqlite3

def create():

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("""

        CREATE TABLE IF NOT EXISTS Modules(

        ID INTEGER PRIMARY KEY AutoIncrement,

        Module VARCHAR(50),

        ModuleCategory VARCHAR(50),

        ModuleType VARCHAR(50)

        )""")

    cur.execute("""

        CREATE TABLE IF NOT EXISTS Questions(

        ID INTEGER PRIMARY KEY AutoIncrement,

        Question VARCHAR(50),

        Category VARCHAR(50),

        Type VARCHAR(50)

    ) """)

    cur.execute("""

        CREATE TABLE IF NOT EXISTS Answers(

        ID INTEGER PRIMARY KEY,

        Description VARCHAR(50),

        IsCorrect INTEGER,

        QuestionID VARCHAR(50),

        FOREIGN KEY(QuestionID) REFERENCES Answers(ID)

    ) """)

    cur.execute("""

        CREATE TABLE IF NOT EXISTS Results(

            ID INTEGER PRIMARY KEY AutoIncrement,

            Result VARCHAR(50),

            Date DATE

        )""")

    con.commit()

    con.close()

def get\_module():

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("""

    SELECT  Module,

            ModuleCategory,

            ModuleType

            FROM Modules""")

    con.commit()

    results = cur.fetchall()

    con.close()

    return results

def insert\_module(text, module\_category, module\_type):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    query = """

            INSERT INTO Results(

                Module

                ModuleCategory,

                ModuleType

            ) VALUES (

                ?,

                ?,

                ?

            )"""

    cur.execute(query, [text, module\_category, module\_type])

    con.commit()

    con.close()

def delete\_module(text, module\_category, module\_type):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("DELETE FROM Results WHERE Module = ? AND ModuleCategory = ? AND ModuleType = ?", [text, module\_category, module\_type])

    con.commit()

    con.close()

def get\_result():

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("""

    SELECT  Result,

            Date

            FROM Results""")

    con.commit()

    results = cur.fetchall()

    con.close()

    return results

def insert\_result(rslt,  result\_date):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    query = """

            INSERT INTO Results(

                Result,

                Date

            ) VALUES (

                ?,

                ?

            )"""

    cur.execute(query, [rslt,  result\_date])

    con.commit()

    con.close()

def delete\_result(rslt, result\_date):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("DELETE FROM Results WHERE Result = ? AND Date = ?", [rslt,  result\_date])

    con.commit()

    con.close()

def insert\_question(text, question\_category, question\_type):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    query = """

            INSERT INTO Questions(

                Question,

                Category,

                Type

            ) VALUES (

                ?,

                ?,

                ?

            )"""

    cur.execute(query, [text, question\_category, question\_type])

    con.commit()

    con.close()

def get\_question():

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("""

    SELECT  Question,

            Category,

            Type

            FROM Questions""")

    con.commit()

    results = cur.fetchall()

    con.close()

    return results

def delete\_question(text, question\_category, question\_type):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    query = "DELETE FROM Questions WHERE Question = ? AND Category = ? AND Type = ?"

    cur.execute(query, [text, question\_category, question\_type])

    con.commit()

    con.close()

def insert\_answer(answer\_description, answer\_is\_correct, question\_id):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("""INSERT INTO Answers

                    (

                    Description,

                    IsCorrect,

                    QuestionID

                    ) VALUES(

                        ?,

                        ?,

                        ?

                    )""", [answer\_description, answer\_is\_correct, question\_id])

    con.commit()

    con.close()

def get\_answer():

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("""

    SELECT  Description,

            IsCorrect,

            QuestionID

            FROM Answers""")

    con.commit()

    results = cur.fetchall()

    con.close()

    return results

def delete\_answer(answer\_description, answer\_is\_correct):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    query = "DELETE FROM Answers WHERE Description = ? AND IsCorrect = ?"

    print(query, answer\_description, answer\_is\_correct)

    cur.execute(query, [answer\_description, answer\_is\_correct])

    con.commit()

    con.close()

def get\_id(text, question\_category, question\_type):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("SELECT ID FROM Questions WHERE Question = ? AND Category = ? AND Type = ?", [text, question\_category, question\_type])

    result = cur.fetchone()

    con.commit()

    con.close()

    return result[0]

def get\_answers(id):

    con = sqlite3.connect("question\_bank.db")

    cur = con.cursor()

    cur.execute("SELECT Description, IsCorrect FROM Answers WHERE QuestionID = ?", [id])

    result = cur.fetchall()

    con.commit()

    con.close()

    return result

i – This is the page called “main.py” where the functions of welcome page are located and the app should be launched in this page.

#this is for enabling to edit the questions, such as selecting, deleting, updating, etc. In addition, the app should be run on this page.

from cProfile import label

from cgitb import text

from distutils import text\_file

from doctest import master

from logging import root

from logging.handlers import QueueListener

from multiprocessing.connection import answer\_challenge

from select import select

import tkinter

from tkinter import \*

from tkinter import filedialog

from tkinter.ttk import Combobox, Treeview

from unicodedata import category

from unittest import result

from question\_bank\_db import \*

from quiz\_ui import \*

from question\_model import Question

from quiz\_data import question\_data

from quiz\_brain import QuizBrain

from random import choices, shuffle

from quiz\_ui import QuizInterface

from tkinter.filedialog import asksaveasfile

import html

#for gathering everything from other pages needed to launch the program

question\_bank = []

for question in question\_data:

    choices = []

    question\_text = html.unescape(question["question"])

    correct\_answer = html.unescape(question["correct\_answer"])

    incorrect\_answers = question["incorrect\_answers"]

    for ans in incorrect\_answers:

        choices.append(html.unescape(ans))

    choices.append(correct\_answer)

    shuffle(choices)

    new\_question = Question(question\_text, correct\_answer, choices)

    question\_bank.append(new\_question)

quiz = QuizBrain(question\_bank)

#the title and the size of the app

class Root(Tk):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        self.title("Quiz App for the Coursework")

        self.geometry("800x700")

#the welcome screen of the app

class Main(Frame):

    def \_\_init\_\_(self, master):

        Frame.\_\_init\_\_(self, master)

        self.master = master

        self.label = Label(self, text="WELCOME TO THE QUIZ", fg="white", bg="blue", width=30, font=("Century Gothic", 15))

        self.label.pack(padx=30, pady=30)

        self.edit\_button = Button(self, text="Edit your Questions", command=self.edit, width=30, height=5)

        self.edit\_button.pack(padx=30, pady=30)

        self.result\_button = Button(self, text="See your Previous Results", command=self.see\_result, width=30, height=5)

        self.result\_button.pack(padx=30, pady=30)

        self.quiz\_button = Button(self, text="Start the Quiz", command=self.start\_quiz, width=30, height=5)

        self.quiz\_button.pack(padx=30, pady=30)

        self.quit\_button = Button(self, text="Quit", command=self.master.destroy, width=30, height=5)

        self.quit\_button.pack(padx=30, pady=30)

#to direct to the results page when click on "See the Previous Results" button

    def see\_result(self):

        result = Results(root)

        result.pack()

        result.refresh()

        main.pack\_forget()

#to direct to the edit question page when click on "Edit your Questions" button

    def edit(self):

        question.pack()

        question.refresh()

        main.pack\_forget()

#to start the quiz when clicking on "Start the Quiz" button

    def start\_quiz(self):

        quiz\_ui = QuizInterface(quiz, root, main)

        quiz\_ui.pack()

        self.pack\_forget()

root = Root()

main = Main(root)

main.pack()

#functions of the layout of the app

class GenericTree(Frame):

    def \_\_init\_\_(self, master, dict, items):

        Frame.\_\_init\_\_(self, master)

        self.master = master

        keys = dict.keys()

        self.items = items

        self.columns = list(keys)

        self.tree = Treeview(self.master, columns = self.columns, show="headings", height=20)

        self.tree.pack()

        for key in keys:

            self.tree.heading(key, text=dict[key], anchor=CENTER)

        self.refresh(items)

    def insrt(self, values):

        self.tree.insert('', END, values=values)

    def refresh(self, items):

        self.delete\_all()

        for item in items:

            self.insrt(item)

    def delete\_all(self):

        items = self.tree.get\_children()

        for item in items:

            self.tree.delete(item)

    def selected\_item(self):

        selected\_item = self.tree.selection()

        return selected\_item

#class for buttons

class GenericButtons(Frame):

    def \_\_init\_\_(self, master, add, delete, answer\_button, back):

        Frame.\_\_init\_\_(self, master)

        self.master = master

        self.add\_bttn = Button(self, text="Add", command=add)

        self.add\_bttn.grid(row=0, column=0)

        self.add\_an\_answer\_button = Button(self, text="Add an Answer", command=answer\_button)

        self.add\_an\_answer\_button.grid(row=0, column=1, padx=10)

        delete\_button = Button(self, text="Delete", command=delete)

        delete\_button.grid(row=0, column=2)

        b\_button = Button(self, text="Back", command=back)

        b\_button.grid(row=0, column=3, padx=10)

        self.q\_button = Button(self, text="Quit", command=self.close)

        self.q\_button.grid(row=0, column=4)

    def close(self):

        root.destroy()

 # functions to add a module

class AddModule(Toplevel):

    def \_init\_(self, master, module\_category, module\_type):

        Toplevel.\_\_init\_\_(self, master)

        self.master = master

        self.geometry("600x460")

        label = Label(self, text="Add Question")

        label.pack()

        self.text = Text(self)

        self.text.pack()

        self.module\_category = Combobox(self, values=module\_category)

        self.module\_category.pack()

        self.module\_type = Combobox(self, values=module\_type)

        self.module\_type.pack()

        addbttn = Button(self, text="Add", command=self.add)

        addbttn.pack()

    def add(self):

        module\_type = self.module\_type.get()

        module\_category = self.module\_category.get()

        text = self.text.get("1.0", 'end')

        insert\_question(text, module\_category, module\_type)

        module.refresh()

        self.destroy()

class Module(Frame):

    def \_\_init\_\_(self, master):

        Frame.\_\_init\_\_(self, master)

        self.master = master

        self.module\_dictionary = {'module' : 'Module Name',

            'module\_category': 'Category',

            'module\_type': 'Type'

            }

        self.module = get\_module()

        self.tree = GenericTree(self, self.module\_dictionary, self.module)

        self.tree.pack()

        self.button = GenericButtons(self, self.add\_mdl, self.delete\_mdl, lambda:None, self.back)

        self.button.add\_an\_answer\_button.destroy()

        self.button.pack()

    def back(self):

        main.pack()

        self.pack\_forget()

    def add\_mdl(self):

        self.module\_category = [

            'Science: Computers',

            'Science: Mathematics',

            'General Knowledge']

        self.module\_type = [

            'Multiple',

            'True/False'

        ]

        AddModule(self, self.module\_category, self.module\_type)

    def delete\_mdl(self):

        item = self.tree.selected\_item()[0]

        itm = item

        itm = self.tree.tree.item(itm)['values']

        self.module\_category = itm[0]

        self.module\_type = itm[2]

        delete\_question(self.module\_category, self.module\_type)

        self.refresh()

    def refresh(self):

        self.tree.refresh(get\_module())

#functions to add a question

class AddQuestion(Toplevel):

    def \_\_init\_\_(self, master, question\_category, question\_type):

        Toplevel.\_\_init\_\_(self, master)

        self.master = master

        self.geometry("600x600")

        label = Label(self, text="Add Question")

        label.pack()

        self.text = Text(self)

        self.text.pack()

        self.question\_type = Combobox(self, values=question\_type)

        self.question\_type.pack()

        self.question\_category = Combobox(self, values=question\_category)

        self.question\_category.pack()

        addbttn = Button(self, text="Add", command=self.add)

        addbttn.pack()

    def add(self):

        question\_type = self.question\_type.get()

        question\_category = self.question\_category.get()

        text = self.text.get("1.0", 'end')

        insert\_question(text, question\_category, question\_type)

        question.refresh()

        self.destroy()

#functions to add an answer to a question

class AddAnswers(Toplevel):

    def \_\_init\_\_(self, master, answer\_is\_correct, question\_id):

        Toplevel.\_\_init\_\_(self, master)

        self.master = master

        self.geometry("600x460")

        label = Label(self, text="Add an Answer")

        label.pack()

        self.question\_id = question\_id

        self.answer\_description = Text(self)

        self.answer\_description.pack()

        self.answer\_is\_correct = Combobox(self, values=answer\_is\_correct)

        self.answer\_is\_correct.pack()

        answer\_bttn = Button(self, text="Add", command=self.answer\_button)

        answer\_bttn.pack()

    def answer\_button(self):

        answer\_description = self.answer\_description.get("1.0", 'end')

        answer\_is\_correct = self.answer\_is\_correct.get()

        if answer\_is\_correct == "Correct":

            insert\_answer(answer\_description, "Correct", self.question\_id)

        elif answer\_is\_correct == "Wrong":

            insert\_answer(answer\_description, "Wrong", self.question\_id)

        self.master.refresh()

        self.destroy()

#functions to see the results of the quiz

class Results(Frame):

    def \_\_init\_\_(self, master):

        Frame.\_\_init\_\_(self, master)

        self.master = master

        self.result\_list = {'result': 'Result',

            'date': 'Date'

            }

        self.rslt = get\_result()

        self.tree = GenericTree(self, self.result\_list, self.rslt)

        self.tree.pack()

        self.bttn = GenericButtons(self, lambda:None, self.dlt\_result, lambda:None, self.back)

        self.bttn.add\_bttn.destroy()

        self.bttn.add\_an\_answer\_button.destroy()

        self.bttn.q\_button.destroy()

        self.bttn.pack()

        #the button to save the result

        print\_button = Button(self, text="Save the result", command=self.print)

        print\_button.pack(pady=10)

    #the function for saving the result

    def print(self):

        files = [('All Files', '\*.\*'),

                        ('Python Files', '\*.py'),

                        ('Text Document', '\*.txt')]

        file = filedialog.asksaveasfile(filetypes=files, defaultextension=files)

    def back(self):

        main.pack()

        self.pack\_forget()

    def dlt\_result(self):

        item = self.tree.selected\_item()[0]

        itm = item

        itm = self.tree.tree.item(itm)['values']

        self.rslt = itm[0]

        self.result\_date = itm[1]

        delete\_result(self.rslt, self.result\_date)

        self.refresh()

    def refresh(self):

        self.tree.refresh(get\_result())

#functions to see the answers of a question

class Answers(Frame):

    def \_\_init\_\_(self, master, id):

        Frame.\_\_init\_\_(self, master)

        self.master = master

        answer\_dictionary = {'description' : 'Description',

                        'is\_correct' : 'Correct or Wrong',

                        'question\_id': 'Question Number'}

        self.id = id

        self.answers = get\_answers(id)

        self.tree = GenericTree(self, answer\_dictionary, self.answers)

        self.tree.pack()

        self.button = GenericButtons(self, self.add\_answer, self.dlt\_answer, self.add\_an\_answer, self.back)

        self.button.pack()

    def back(self):

        question.pack()

        question.refresh()

        self.pack\_forget()

    def add\_answer(self):

        AddAnswers(self, ['Correct', 'Wrong'], self.id)

    def dlt\_answer(self):

        item = self.tree.selected\_item()[0]

        itm = item

        itm = self.tree.tree.item(itm)['values']

        answer\_description = itm[0]

        answer\_is\_correct = itm[1]

        delete\_answer(answer\_description, answer\_is\_correct)

        self.refresh()

    def add\_an\_answer(self):

        pass

    def refresh(self):

        self.answers = get\_answers(self.id)

        print(self.answers)

        self.tree.refresh(self.answers)

#functions to see the questions

class Question(Frame):

    def \_\_init\_\_(self, master):

        Frame.\_\_init\_\_(self, master)

        self.master = master

        self.dictnry = {'question': 'Questions',

            'category': 'Category',

            'type': 'Type'

            }

        self.result = get\_question()

        self.tree = GenericTree(self, self.dictnry, self.result)

        self.tree.pack()

        self.button = GenericButtons(self, self.add, self.delete, self.add\_the\_answer, self.back)

        self.button.pack()

    def back(self):

        main.pack()

        self.pack\_forget()

    def add(self):

        self.category = [

            'Science: Computers',

            'Science: Mathematics',

            'General Knowledge']

        self.type = [

            'Multiple',

            'True/False'

        ]

        AddQuestion(self, self.category, self.type)

    def delete(self):

        item = self.tree.selected\_item()[0]

        itm = item

        itm = self.tree.tree.item(itm)['values']

        self.question = itm[0]

        self.category = itm[1]

        self.type = itm[2]

        delete\_question(self.question, self.category, self.type)

        self.refresh()

    def add\_the\_answer(self):

        item = self.tree.selected\_item()[0]

        itm = item

        itm = self.tree.tree.item(itm)['values']

        question = itm[0]

        category = itm[1]

        type = itm[2]

        id = get\_id(question, category, type)

        answer = Answers(root, id)

        answer.pack()

        self.pack\_forget()

    def refresh(self):

        self.tree.refresh(get\_question())

question = Question(root)

root.mainloop()

ii- This is the “quiz\_ui.py” in which the functions of UI of the quiz are written.

#this is the file where the UI is created for the app. this file includes the design and the function of the app

from doctest import master

from random import choice

from re import X

from tkinter import BOTTOM, TOP, Y, Entry, Tk, Canvas, StringVar, Label, Radiobutton, Button, messagebox, Frame

from tkinter.constants import S

from unittest.main import main

from quiz\_brain import QuizBrain

from time import time, strftime

from question\_bank\_db import \*

THEME\_COLOR = "#375362"

class QuizInterface(Frame):

    def \_\_init\_\_(self, quiz\_brain, master, main):

        Frame.\_\_init\_\_(self, master)

        self.main = main

        self.master = master

        self.quiz = quiz\_brain

        #Display title

        self.display\_title()

        #Creating a canvas for question text, and display question

        self.canvas = Canvas(self, width=800, height=500)

        self.question\_text = self.canvas.create\_text(400, 80,

                                                     text="Question here",

                                                     width=680, fill=THEME\_COLOR,

                                                     font=('Century Gothic', 15, 'italic')

                                                     )

        self.canvas.pack(side=TOP)

        self.display\_question()

        #Declare a StringVar to store the answers

        self.user\_answer = StringVar(self)

        #Display four options

        self.opts = self.radio\_buttons()

        self.display\_options()

        #To show if the answer is correct

        self.feedback = Label(self, font=("ariel", 15, "bold"))

        self.feedback.place(x=300, y=20)

        #Next and Quit Buttons

        self.buttons()

    def display\_title(self):

        """To display title"""

        #Title

        title = Label(self, text="Quiz Application for the Coursework",

                      width=55, bg="blue", fg="white", font=("Bahnschrift", 20, "bold"))

        title.place(x=0, y=2)

    def display\_question(self):

        """To display the question"""

        q\_text = self.quiz.next\_question()

        self.canvas.itemconfig(self.question\_text, text=q\_text)

    def radio\_buttons(self):

        """To create the options"""

        #initialize the list with an empty list

        choice\_list = []

        #position of the first option

        y\_pos = 115

        #adding the options to the list

        while len(choice\_list) < 4:

            #setting the radiobutton properties

            radio\_btn = Radiobutton(self, text="", variable=self.user\_answer,

                                    value='', font=("Century Gothic", 14))

            radio\_btn.place(x=200, y=y\_pos)

            #adding the button to the list

            choice\_list.append(radio\_btn)

            #incrementing the y-axis by 40

            y\_pos += 40

        return choice\_list

    def display\_options(self):

        """To display four options"""

        val = 0

        #deselecting the options

        self.user\_answer.set(None)

        #looping over the options to be displayed

        for option in self.quiz.current\_question.choices:

            self.opts[val] ['text'] = option

            self.opts[val] ['value'] = option

            val += 1

    def next\_btn(self):

        """To show feedback and check for more questions"""

        #Check if the answer is correct

        if self.quiz.check\_answer(self.user\_answer.get()):

            self.feedback["fg"] = "green"

            self.feedback["text"] = 'Great, correct answer! \U0001F44D'

        else:

            self.feedback['fg'] = 'red'

            self.feedback['text'] = ('\u274E Oops!'

                                     f'The right answer is: {self.quiz.current\_question.correct\_answer}')

        if self.quiz.has\_more\_questions():

            #Moves to the next question

            self.display\_question()

            self.display\_options()

        else:

            #if no more questions, display the score

            self.display\_result()

            #exits the quiz and goes back to the main screen

            self.main.pack()

            self.pack\_forget()

    def buttons(self):

        """To show the next and quit buttons"""

        #next question

        next\_button = Button(self, text="Next", command=self.next\_btn,

                             width=10, bg="green", fg="white", font=("ariel", 16, "bold"))

        next\_button.place(x=50, y=10)

        #quit button

        quit\_button = Button(self, text="Quit", command=self.back,

                             width=5, bg="red", fg="white", font=("ariel", 16, "bold"))

        quit\_button.place(x=700, y=10)

    def back(self):

        self.main.pack()

        self.pack\_forget()

    def display\_result(self):

        """To display the result"""

        correct, wrong, score\_percent = self.quiz.get\_score()

        #saves the result in previus results page

        store\_result = "{}/{}".format(correct, correct + wrong)

        date = strftime('%d %m %y --%H: %M: %S:')

        insert\_result(store\_result, date)

        correct = f"Correct: {correct}"

        wrong = f"Wrong: {wrong}"

        #calculates the percentage of the answers

        result = f"Score: {score\_percent}%"

        #shows a message box to display the result

        messagebox.showinfo("Result", f"{result}\n{correct}\n{wrong}")

        #creates a text file where the results are stored

        filename = 'results.txt'

        data = 'Correct Answer: '+correct+'\nWrong Answer: '+wrong+'\nScore: '+str(score\_percent)+'%'+'\nCompleted quiz at: '+strftime('%d/%m/%y --%H:/%M:/%S:')

        with open(filename, 'a') as file:

            file.write(data)

iii- This is the “question\_model.py”.

#this file creates the question model for the quiz, such as the questions themselves and the choices

class Question:

    def \_\_init\_\_(self, question: str, correct\_answer: str, choices: list):

        self.question\_text = question

        self.correct\_answer = correct\_answer

        self.choices = choices

iv – This is the page “quiz\_brain.py”.

#this is, as the name implies, the brain of the quiz. this file contains what to do for each question.

class QuizBrain:

    def \_\_init\_\_(self, questions):

        self.question\_no = 0

        self.score = 0

        self.questions = questions

        self.current\_question = None

    def has\_more\_questions(self):

        """To check if the quiz has more questions"""

        return self.question\_no < len(self.questions)

    def next\_question(self):

        """Get the next question by incrementing the question number"""

        self.current\_question = self.questions[self.question\_no]

        self.question\_no += 1

        q\_text = self.current\_question.question\_text

        return f"Q.{self.question\_no}: {q\_text}"

    def check\_answer(self, user\_answer):

        """Check the user anwer and get the score"""

        correct\_answer = self.current\_question.correct\_answer

        if user\_answer.lower() == correct\_answer.lower():

            self.score += 1

            return True

        else:

            return False

    def get\_score(self):

        """Get the number of correct and wrong answers as well as the percentage"""

        wrong = self.question\_no - self.score

        score\_percent = int(self.score / self.question\_no \* 100)

        return(self.score, wrong, score\_percent)

v – This is the “quiz\_data.py”.

#this file is for storing the questions and the options for the quiz

import requests

parameters = {

    "amount" : 5,

    "type" : "multiple"

}

#the program will take the questions from the php page created

response = requests.get(url="https://opentdb.com/api.php?amount=5&category=18&difficulty=easy&type=multiple", params=parameters)

question\_data = response.json() ["results"]

#These are the sample questions in the quiz.

"""

{"response\_code":0,"results":[

    {"category":"Science: Computers","type":"multiple","difficulty":"easy","question":"What amount of bits commonly equals one byte?","correct\_answer":"8","incorrect\_answers":["1","2","64"]},

    {"category":"Science: Computers","type":"multiple","difficulty":"easy","question":"What is the domain name for the country Tuvalu?","correct\_answer":".tv","incorrect\_answers":[".tu",".tt",".tl"]},

    {"category":"Science: Computers","type":"multiple","difficulty":"easy","question":"In &quot;Hexadecimal&quot;, what color would be displayed from the color code? &quot;#00FF00&quot;?","correct\_answer":"Green","incorrect\_answers":["Red","Blue","Yellow"]},

    {"category":"Science: Computers","type":"multiple","difficulty":"easy","question":"How many values can a single byte represent?","correct\_answer":"256","incorrect\_answers":["8","1","1024"]},{"category":"Science: Computers","type":"multiple","difficulty":"easy","question":"Which programming language shares its name with an island in Indonesia?","correct\_answer":"Java","incorrect\_answers":["Python","C","Jakarta"]}

    ]}

"""