# COMP1811 – Python Project Report

| **Name** | **Cristian Dumbravanu** | **Student ID** | **001167783** |
| --- | --- | --- | --- |
| **Partner’s name** | **Alina Cricovan** | **Partner SIDs** | 001198342 |

# Brief statement of features you have completed

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

| 1.1 Circle the parts of the coursework you have **fully completed and are fully working**. Please be accurate. | **Features** | **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  F1 ii) The GUI displays the button widgets of “Adding a new topic “, “Deleting a new topic “and “Editing a topic” on the Additional page. But they only serve a visual role but don’t operate or allow the user to interact with it.  F1 iii) Similar to F1 ii) The user can see the button of “Deleting a record “ but it has no function to produce a output. | | |

# 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

*List each bug plus a brief description*

When taking a quiz, the user is able to go back to the previous question but that would automatically unselect the answer when going back and then pressing next. The user is able capable of moving on to the next question without selecting an answer, but the code will automatically consider it as wrong.

When you press back on the first question you come to the question 5.

After finishing a quiz, you cannot start another one without exiting Tkinter and running the code again. However, you can return to the main menu.

## Weaknesses

*List each weakness plus a brief description*

Our program lacks the many functions that it displays to have. We have not fulfilled all the necessary

functions required.

The program extracts the questions and the answers from 4 different databases that are ran on json

and not Sqlite 3. We assume that if we merged the databases into one then the code would be more

optimized and efficient.

# 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 gone for a simpler design that was inspired by the physical multiple-choice tests we had taken in GCSEs or A level. We have used plain colors to give a more professional look and modern look while still having certain buttons a red color to attract the user’s attention to the feature. The rest of the features were all around the simple design to make it adaptable for any user to understand no matter the age or experience with technology. The administration features that have been completed work well and compliment the lay out of the GUI to once again make it simple for the users.

# 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

Each frame / page has been given a separate class that is unique for its widgets or design but are all centralized around the first page / frame.

Our classes are :

Page

Page1

Page2

Page2\_1

Page4

Page5

Page6

Quiz

## Brief Explanation of Class Design and OOP Features Used

We have utilized each class as a blueprint for each page and creating objects based on widgets or output. All the attributes for each frame are defined in the class of that specific page. After we had defined a class, a new class object is created with the same name. This class object allowed us to access the different attributes as well as to instantiate new objects of that class. Some classes also consist of methods that are basically specific functions for that class. E.g. counting the total after each quiz.

For OOP we have used the classes and objects to create inheritance, polymorphism and encapsulation between the pages and its functions. This has made our code more efficient by saving a lot of space that could’ve been repeated if not for inheritance. This optimized our code making it simpler for us to understand and requires less storage.

# 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.)*

We both contributed equally when it comes to creating the classes. We both came up with a structure and that same structure was followed and replicated throughout the code.

## Class …

1. class Page2(Page):
2. def \_\_init\_\_(self, \*args, \*\*kwargs):
3. Page.\_\_init\_\_(self, \*args, \*\*kwargs)
4. label = tk.Label(self, text="Topics", font=('Georgia', 18))
5. label.pack(side="top",padx=5, pady=1)

## Class …

1. class Page2\_1(Page):
2. def \_\_init\_\_(self, \*args, \*\*kwargs):
3. Page.\_\_init\_\_(self, \*args, \*\*kwargs)
4. label = tk.Label(self, text="Search for a topic:", font=('Georgia', 18))
5. label.pack(side="top",padx=5, pady=1)

## Class …

class Page3(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

label = tk.Label(self, text="Do you want to start?", font=('Georgia', 30))

label.pack(padx=5, pady=2)

## Class …

1. class Page4(Page):
2. def \_\_init\_\_(self, \*args, \*\*kwargs):
3. Page.\_\_init\_\_(self, \*args, \*\*kwargs)
4. label = tk.Label(self, text="Do you want to start?", font=('Georgia', 30))
5. label.pack(padx=5, pady=2)

# 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)*

The main method of testing used was using the integrational strategy and white box. Since we were both at a distance, we would send each other’s progress and overwrite additional code to it and so on. Any new lines of coded added were shared on the Teams chat and we would continue working on the latest version that was uploaded.

The other testing was done through the terminal. If any errors occur, then the PROBLEMS section of the VScode studio would display the exact line of code explaining what the issue is.

And of course, we would just experiment with the GUI ourselves and any bugs we had we would change them with each other and create a list.

# 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

## Sub-feature i- screenshots …

Text

Description automatically generatedGraphical user interface, text

Description automatically generatedGraphical user interface, website

Description automatically generated

## Sub-feature ii- screenshots …

Text

Description automatically generatedA picture containing graphical user interface

Description automatically generated

## Sub-feature iii- screenshots …

## Feature F2

## Sub-feature i- screenshots …

Text

Description automatically generated

## Sub-feature ii- screenshots …

Text

Description automatically generated

## Sub-feature iii- screenshots …

# 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

We haven’t met all the requirements and I wouldn’t consider our implementation and design to be flawless. I still however consider it to be a good foundation to what could be a fully functional quiz.

## Evaluate you own and your group’s performance

The general group performance was great. We didn’t lack communication even at a distance and we would always keep each other up with anything new and potential ideas and certain solutions were always discussed to resolve any trouble we came across.

Personally, since I haven’t started working on the course work a lot earlier and did everything within a month I wouldn’t consider my performance as a “peak “of my potential.

## What went well?

The general basic foundation of a quiz and how the GUI is presented for a user.

The practice of OOP and the work using classes followed by the other programing strategies used to minimize any repeat any code.

We have both shower knowledge sufficient knowledge of Python and at least the basic understating of Tkinter,

## What went less well?

The database was the main factor holding us back. Since we didn’t manage to complete it as required, we weren’t able to include all of the features that we wanted and were demanded.

## What was learnt?

We learned a lot about communication and pear working. I’ve personally adapted to my partner’s work ethic and style.

Time management and organization was one of the biggest factors I was missing so I will be focusing more on that for the future projects and prioritize planning and starting the projects as quick as possible and not to then be overwhelmed by the work at the end.

## How would a similar task be completed differently?

We would develop a plan and stages to our development. The plan would include dates, deadlines, and milestone for us to have a better understanding of our progress. We would immediately begin with the course work and would have different ideas on how to approach the development and factor in the ups and down to decide on an ideal one.

## How could the module be improved?

Creating a database suitable for this project so we could accomplish the required features of the Quiz.

## 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: 13 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: 19 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: 12 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: 3 out of 10**

**I think my overall mark would be: 57 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** |
| --- | --- | --- | --- | --- |
| **001167783** | Json databse , half of the classes and half of the methodologies. | 50% |  |  |
| 001198342 | GUI design half of the classes and half of the methodologies and implementation of inheritance. | 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)*

#import tkinter with the name tk

import tkinter as tk

#import font for fonts(text)

import tkinter.font as font

#this import everything from tkinter

from tkinter import \*

# and import messagebox as mb from tkinter

from tkinter import messagebox as mb

#import json to use json file for data

import json

#data for 1st quiz

with open('Data1.json') as f:

data = json.load(f)

# set the question, options, and answer

question = (data['question'])

options = (data['options'])

answer = (data[ 'answer'])

#data for 2nd quiz

with open('Data2.json') as f:

data1 = json.load(f)

# set the question, options, and answer

question1 = (data1['question1'])

options1 = (data1['options1'])

answer1 = (data1[ 'answer1'])

#data for 3rd quiz

with open('Data3.json') as f:

data2 = json.load(f)

# set the question, options, and answer

question2 = (data2['question2'])

options2 = (data2['options2'])

answer2 = (data2[ 'answer2'])

#data for 4th quiz

with open('Data4.json') as f:

data3 = json.load(f)

# set the question, options, and answer

question3 = (data3['question3'])

options3 = (data3['options3'])

answer3 = (data3[ 'answer3'])

#create the frame page which will serve for all pages

class Page(tk.Frame):

def \_\_init\_\_(self, \*args, \*\*kwargs):

tk.Frame.\_\_init\_\_(self, \*args, \*\*kwargs)

def show(self):

self.lift()

#the frame for the page1

class Page1(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

#the frame for the page2

class Page2(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

label = tk.Label(self, text="Topics", font=('Georgia', 18))

label.pack(side="top",padx=5, pady=1)

class Page2\_1(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

label = tk.Label(self, text="Search for a topic:", font=('Georgia', 18))

label.pack(side="top",padx=5, pady=1)

#the frame for the page3

class Page3(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

label = tk.Label(self, text="Do you want to start?", font=('Georgia', 30))

label.pack(padx=5, pady=2)

#the frame for the page4

class Page4(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

label = tk.Label(self, text="Do you want to start?", font=('Georgia', 30))

label.pack(padx=5, pady=2)

#the frame for the page5

class Page5(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

label = tk.Label(self, text="Do you want to start?", font=('Georgia', 30))

label.pack(padx=5, pady=2)

#the frame for the page6

class Page6(Page):

def \_\_init\_\_(self, \*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

label = tk.Label(self, text="Do you want to start?", font=('Georgia', 30))

label.pack(padx=5, pady=2)

#the frame for the quiz(structure)

class Quiz(Page):

def \_\_init\_\_(self,\*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

#set the questions number to 0(start)

self.q\_no=0

self.display\_title()

self.display\_question()

#opt\_selected holds an integer value which is used for selected option in a question.

self.opt\_selected=IntVar()

#displaying radio button for the current question and used to display options for the current question

self.opts=self.radio\_buttons()

#this is to display options for the current question

self.display\_options()

#this is to display the button for next and exit in the actual quiz

self.buttons()

self.data\_size=len(question)

#set the correct answered questions to 0

self.correct=0

def display\_result(self):

wrong\_count = self.data\_size - self.correct

correct = f"Correct: {self.correct}"

wrong = f"Wrong: {wrong\_count}"

score = int(self.correct / self.data\_size \* 100)

result = f"Score: {score}%"

#this shows in a separate window the results from the quiz

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

def check\_ans(self, q\_no):

# checks for if the selected option is correct

if self.opt\_selected.get() == answer[q\_no]:

# if the option is correct it return the value true

return True

def next\_btn(self):

# Check if the answer is correct

if self.check\_ans(self.q\_no):

# if the answer is correct it increments the correct by 1

self.correct += 1

# Moves to next Question by incrementing the q\_no counter

self.q\_no += 1

# checks if the q\_no size is equal to the data size

if self.q\_no==self.data\_size:

# if it is correct then it displays the score

self.display\_result()

# destroys the GUI

self.destroy()

else:

# shows the next question

self.display\_question()

self.display\_options()

def back\_btn(self):

# Check if the answer is correct

if self.check\_ans(self.q\_no):

# if the answer is correct it increments the correct by 1

self.correct -= 1

# back to the Question

self.q\_no -= 1

# checks if the q\_no size is equal to the data size

if self.q\_no==self.data\_size:

# if it is correct then it displays the score

self.display\_result()

self.destroy()

else:

# shows the next question

self.display\_question()

self.display\_options()

def buttons(self):

# The first button is the Next button to move to the

# next Question

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

width=10,font=("Georgia",16,"bold"))

# palcing the button on the screen

back\_button=Button(self, text="Back", command=self.back\_btn,

width=10,bg="blue",font=("Georgia",16,"bold"))

next\_button.place(x=400,y=380)

back\_button.place(x=270,y=380)

# This is the second button which is used to Quit the GUI

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

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

# placing the Quit button on the screen

# quit\_button.place(x=700,y=50)

def display\_options(self):

val=0

# deselecting the options

self.opt\_selected.set(0)

# looping over the options to be displayed for the

# text of the radio buttons.

for option in options[self.q\_no]:

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

val+=1

# This method shows the current Question on the screen

def display\_question(self):

# setting the Question properties

q\_no = Label(self, text=question[self.q\_no], width=60,

font=( 'Georgia' ,16, 'bold' ), anchor= 'w' )

#placing the option on the screen

q\_no.place(x=210, y=100)

# This method is used to Display Title

def display\_title(self):

# The title to be shown

title = Label(self, text="QUIZ 1",

width=50, font=("Georgia", 20, "bold"))

# place of the title

title.place(x=20, y=1)

def radio\_buttons(self):

# initialize the list with an empty list of options

q\_list = []

# position of the first option

y\_pos = 150

# adding the options to the list

while len(q\_list) < 4:

# setting the radio button properties

radio\_btn = Radiobutton(self,text=" ",variable=self.opt\_selected,

value = len(q\_list)+1,font = ("Georgia",14))

# adding the button to the list

q\_list.append(radio\_btn)

# placing the button

radio\_btn.place(x = 100, y = y\_pos)

# incrementing the y-axis position by 40

y\_pos += 40

# return the radio buttons

return q\_list

class Quiz2(Page):

def \_\_init\_\_(self,\*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

self.q\_no=0

self.display\_title()

self.display\_question()

self.opt\_selected=IntVar()

self.opts=self.radio\_buttons()

self.display\_options()

self.buttons()

self.data\_size=len(question1)

self.correct=0

def display\_result(self):

wrong\_count = self.data\_size - self.correct

correct = f"Correct: {self.correct}"

wrong = f"Wrong: {wrong\_count}"

score = int(self.correct / self.data\_size \* 100)

result = f"Score: {score}%"

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

def check\_ans(self, q\_no):

if self.opt\_selected.get() == answer1[q\_no]:

return True

def next\_btn(self):

if self.check\_ans(self.q\_no):

self.correct += 1

self.q\_no += 1

if self.q\_no==self.data\_size:

self.display\_result()

self.destroy()

else:

self.display\_question()

self.display\_options()

def back\_btn(self):

if self.check\_ans(self.q\_no):

self.correct -= 1

self.q\_no -= 1

if self.q\_no==self.data\_size:

self.display\_result()

self.destroy()

else:

self.display\_question()

self.display\_options()

def buttons(self):

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

width=10,font=("Georgia",16,"bold"))

back\_button=Button(self, text="Back", command=self.back\_btn,

width=10,bg="blue",font=("Georgia",16,"bold"))

next\_button.place(x=400,y=380)

back\_button.place(x=270,y=380)

def display\_options(self):

val=0

self.opt\_selected.set(0)

for option in options1[self.q\_no]:

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

val+=1

def display\_question(self):

q\_no = Label(self, text=question1[self.q\_no], width=60,

font=( 'Georgia' ,16, 'bold' ), anchor= 'w' )

q\_no.place(x=210, y=100)

def display\_title(self):

title = Label(self, text="QUIZ 2",

width=50, font=("Georgia", 20, "bold"))

title.place(x=20, y=1)

def radio\_buttons(self):

q\_list = []

y\_pos = 150

while len(q\_list) < 4:

radio\_btn = Radiobutton(self,text=" ",variable=self.opt\_selected,

value = len(q\_list)+1,font = ("Georgia",14))

q\_list.append(radio\_btn)

radio\_btn.place(x = 100, y = y\_pos)

y\_pos += 40

return q\_list

class Quiz3(Page):

def \_\_init\_\_(self,\*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

self.q\_no=0

self.display\_title()

self.display\_question()

self.opt\_selected=IntVar()

self.opts=self.radio\_buttons()

self.display\_options()

self.buttons()

self.data\_size=len(question2)

self.correct=0

def display\_result(self):

wrong\_count = self.data\_size - self.correct

correct = f"Correct: {self.correct}"

wrong = f"Wrong: {wrong\_count}"

score = int(self.correct / self.data\_size \* 100)

result = f"Score: {score}%"

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

def check\_ans(self, q\_no):

if self.opt\_selected.get() == answer2[q\_no]:

return True

def next\_btn(self):

if self.check\_ans(self.q\_no):

self.correct += 1

self.q\_no += 1

if self.q\_no==self.data\_size:

self.display\_result()

self.destroy()

else:

self.display\_question()

self.display\_options()

def back\_btn(self):

if self.check\_ans(self.q\_no):

self.correct -= 1

self.q\_no -= 1

if self.q\_no==self.data\_size:

self.display\_result()

self.destroy()

else:

self.display\_question()

self.display\_options()

def buttons(self):

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

width=10,font=("Georgia",16,"bold"))

back\_button=Button(self, text="Back", command=self.back\_btn,

width=10,bg="blue",font=("Georgia",16,"bold"))

next\_button.place(x=400,y=380)

back\_button.place(x=270,y=380)

def display\_options(self):

val=0

self.opt\_selected.set(0)

for option in options2[self.q\_no]:

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

val+=1

def display\_question(self):

q\_no = Label(self, text=question2[self.q\_no], width=60,

font=( 'Georgia' ,16, 'bold' ), anchor= 'w' )

q\_no.place(x=210, y=100)

def display\_title(self):

title = Label(self, text="QUIZ 3",

width=50, font=("Georgia", 20, "bold"))

title.place(x=20, y=1)

def radio\_buttons(self):

q\_list = []

y\_pos = 150

while len(q\_list) < 4:

radio\_btn = Radiobutton(self,text=" ",variable=self.opt\_selected,

value = len(q\_list)+1,font = ("Georgia",14))

q\_list.append(radio\_btn)

radio\_btn.place(x = 100, y = y\_pos)

y\_pos += 40

return q\_list

class Quiz4(Page):

def \_\_init\_\_(self,\*args, \*\*kwargs):

Page.\_\_init\_\_(self, \*args, \*\*kwargs)

self.q\_no=0

self.display\_title()

self.display\_question()

self.opt\_selected=IntVar()

self.opts=self.radio\_buttons()

self.display\_options()

self.buttons()

self.data\_size=len(question3)

self.correct=0

def display\_result(self):

wrong\_count = self.data\_size - self.correct

correct = f"Correct: {self.correct}"

wrong = f"Wrong: {wrong\_count}"

score = int(self.correct / self.data\_size \* 100)

result = f"Score: {score}%"

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

def check\_ans(self, q\_no):

if self.opt\_selected.get() == answer3[q\_no]:

return True

def next\_btn(self):

if self.check\_ans(self.q\_no):

self.correct += 1

self.q\_no += 1

if self.q\_no==self.data\_size:

self.display\_result()

self.destroy()

else:

self.display\_question()

self.display\_options()

def back\_btn(self):

if self.check\_ans(self.q\_no):

self.correct -= 1

self.q\_no -= 1

if self.q\_no==self.data\_size:

self.display\_result()

self.destroy()

else:

self.display\_question()

self.display\_options()

def buttons(self):

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

width=10,font=("Georgia",16,"bold"))

back\_button=Button(self, text="Back", command=self.back\_btn,

width=10,bg="blue",font=("Georgia",16,"bold"))

next\_button.place(x=400,y=380)

back\_button.place(x=270,y=380)

def display\_options(self):

val=0

self.opt\_selected.set(0)

for option in options3[self.q\_no]:

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

val+=1

def display\_question(self):

q\_no = Label(self, text=question3[self.q\_no], width=60,

font=( 'Georgia' ,16, 'bold' ), anchor= 'w' )

q\_no.place(x=210, y=100)

def display\_title(self):

title = Label(self, text="QUIZ 4",

width=50, font=("Georgia", 20, "bold"))

title.place(x=20, y=1)

def radio\_buttons(self):

q\_list = []

y\_pos = 150

while len(q\_list) < 4:

radio\_btn = Radiobutton(self,text=" ",variable=self.opt\_selected,

value = len(q\_list)+1,font = ("Georgia",14))

q\_list.append(radio\_btn)

radio\_btn.place(x = 100, y = y\_pos)

y\_pos += 40

return q\_list

#this is the main frame, where all the pages, buttons, and other functions will be displayed

class MainView(tk.Frame):

def \_\_init\_\_(self, \*args, \*\*kwargs):

tk.Frame.\_\_init\_\_(self, \*args, \*\*kwargs)

#we note every page with a new name to work easily with them

p1 = Page1(self)

p2 = Page2(self)

p2\_1 = Page2\_1(self)

p3 = Page3(self)

p4 = Page4(self)

p5 = Page5(self)

p6 = Page6(self)

p7 = Quiz(self)

p8 = Quiz2(self)

p9 = Quiz3(self)

p10 = Quiz4(self)

#this will be the frame for the buttons from the pages

buttonframe = tk.Frame(self)

#this is where the pages are stored

container = tk.Frame(self)

buttonframe.pack(side="top", fill="x", expand=False)

container.pack(side="top", fill="both", expand=True)

#here, every page is placed in the container, in thier own space

p1.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p2.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p2\_1.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p3.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p4.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p5.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p6.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p7.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p8.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p9.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

p10.place(in\_=container, x=0, y=0, relwidth=1, relheight=1)

#page1

#we created a label in the first page and organised it with "pack"

label = tk.Label(p1, text="Main Menu", font=('Georgia', 30))

label.pack(pady=10)

#this is an additional feature, where users can write thier names, and the system will greet them

e = tk.Entry(p1, width=50)

e.insert(0, "")

def myCLick(a):

hello = "Hello " + e.get() + "!"

mylabel = tk.Label(p1, text=hello, font="Georgia" ,)

mylabel.pack()

#this is an additional button for entering the name

# myButton = tk.Button(p1, text="Enter", command=myCLick)

# myButton.pack()

tk.Label(p1, text="Enter your name:", font="Georgia" ).pack()

e.pack()

root.bind('<Return>', myCLick)

#the next butt which move to the next page

b1 = tk.Button(p1, text="Next",font="Georgia" , command=lambda: p2.show())

b1.pack()

p1.show()

#page2

#the exit button will stay in the whole time on the window. We can quit the page whenever we want

button\_exit = tk.Button(root, text="exit program", font="Georgia" ,command=root.quit)

#every topic we have is a button, and when is clicked it opens a new page

b3=tk.Button(p2,text="Mathematics", font="Georgia" ,command=lambda:p3.show())

b4=tk.Button(p2,text="Computer Sience",font="Georgia" , command=lambda:p4.show())

b5=tk.Button(p2,text="Software Engineering",font="Georgia" , command=lambda:p5.show())

b6=tk.Button(p2,text="History",font="Georgia" , command=lambda:p6.show())

b7=tk.Button(p2,text="Additional page",font="Georgia" , command=lambda:p2\_1.show())

#this is how we pack the buttons

b3.pack(side="top",fill="x")

b4.pack(side="top",fill="x")

b5.pack(side="top",fill="x")

b6.pack(side="top",fill="x")

b7.pack(side="top",fill="x")

#this button will switch pages

b0=tk.Button(p2, text='Go back!', font=('Georgia', 15),fg='red',command=lambda:p1.show())

b0.pack(pady=10)

#page2\_1

#this additional page offers the opportunity to check whether a topic exists or not

s = tk.Entry(p2\_1, width=30)

s.insert(0,"")

first="Mathematics"

second="Computer Sience"

third="Software Engineering"

forth="History"

b=[first, second,third, forth]

def myt():

if s.get() in b:

text1 = s.get() + " already exists!"

newlabel = tk.Label(p2\_1, text=text1, font="Georgia" )

newlabel.pack()

else:

label2=tk.Label(p2\_1, text="No such topic!", font="Georgia" )

label2.pack()

s.pack()

enter1=tk.Button(p2\_1, text="enter",font="Georgia" , command=myt)

enter1.pack()

#this are the buttons if someone wants to add, edit, or delete a topic

add1=tk.Button(p2\_1, text="Add a new topic",font="Georgia" )

add1.pack()

edit1=tk.Button(p2\_1, text="Edit a topic",font="Georgia" )

edit1.pack()

delete1=tk.Button(p2\_1, text="Delete a topic",font="Georgia" )

delete1.pack()

#this button allows the user to go back to the previous page

b2=tk.Button(p2\_1, text='Go back!', font=('Georgia', 15),fg='red',command=lambda:p2.show())

b2.pack(pady=10)

#page3

#the buttons will repeat for all the pages

b3\_2=tk.Button(p3, text="Start the quiz!", font="Georgia" ,command=lambda:p7.show())

b3\_2.pack(side='top', pady=20)

add\_b= tk.Button(p3, text="Add a new question",font="Georgia" )

add\_b.pack()

edit\_b= tk.Button(p3, text="Edit a question",font="Georgia" )

edit\_b.pack()

delete\_b=tk.Button(p3, text="Detele a question",font="Georgia" )

delete\_b.pack()

b3\_1=tk.Button(p3,text="Back",font="Georgia" , command=lambda:p2.show())

b3\_1.pack(side="top", pady=10)

#page4

b3\_2\_a=tk.Button(p4, text="Start the quiz!", font="Georgia" ,command=lambda:p8.show())

b3\_2\_a.pack(side='top', pady=20)

add\_b4= tk.Button(p4, text="Add a new question",font="Georgia" )

add\_b4.pack()

edit\_b4= tk.Button(p4, text="Edit a question",font="Georgia" )

edit\_b4.pack()

delete\_b4=tk.Button(p4, text="Detele a question",font="Georgia" )

delete\_b4.pack()

b3\_1a=tk.Button(p4,text="Back",font="Georgia" , command=lambda:p2.show())

b3\_1a.pack(side="top", pady=10)

#page5

b3\_2b=tk.Button(p5, text="Start the quiz!",font="Georgia" , command=lambda:p9.show())

b3\_2b.pack(side='top', pady=20)

add\_b5= tk.Button(p5, text="Add a new question",font="Georgia" )

add\_b5.pack()

edit\_b5= tk.Button(p5, text="Edit a question",font="Georgia" )

edit\_b5.pack()

delete\_b5=tk.Button(p5, text="Detele a question",font="Georgia" )

delete\_b5.pack()

b3\_1b=tk.Button(p5,text="Back",font="Georgia" , command=lambda:p2.show())

b3\_1b.pack(side="top", pady=10)

#page6

b3\_2c=tk.Button(p6, text="Start the quiz!",font="Georgia" , command=lambda:p10.show())

b3\_2c.pack(side='top', pady=20)

add\_b6= tk.Button(p6, text="Add a new question",font="Georgia" )

add\_b6.pack()

edit\_b6= tk.Button(p6, text="Edit a question",font="Georgia" )

edit\_b6.pack()

delete\_b6=tk.Button(p6, text="Detele a question",font="Georgia" )

delete\_b6.pack()

b3\_1c=tk.Button(p6,text="Back",font="Georgia" , command=lambda:p2.show())

b3\_1c.pack(side="top", pady=10)

button\_exit.pack(side="bottom",fill="x")

#this is how we form the window in tkinter, how we call the class, and set the title

root = tk.Tk()

main = MainView(root)

root.title("Quiz Generator")

main.pack(side="top", fill="both", expand=True)

#here we set the size of the page

root.wm\_geometry("800x450")

root.mainloop()

# END OF THE PROGRAM