Data Driven Application

# Development Document

……………………..

Adhenz Miranda

Bath Spa University RAK

BSC Creative Computing Year 2

Code Lab II

# Table of Contents

[**Development Document 0**](#_30j0zll)

[**Table of Contents 1**](#_1fob9te)

[**TechBlaizer 2**](#_3znysh7)

[Abstract 2](#_2et92p0)

[Project Plan 3](#_3dy6vkm)

[Goals 3](#_1t3h5sf)

[Original Timeline 3](#_4d34og8)

[Issues Faced 4](#_2s8eyo1)

[Linking the arguments and classes 4](#_17dp8vu)

[Converting HTML special characters into appropriate Python strings 5](#_3rdcrjn)

[Numbers next to the listed text in the for loop 6](#_26in1rg)

[Functions not having appropriate arguments when going to the next class 7](#_lnxbz9)

[Disabling the button states to prevent score system abuse and element overflow in fixed windows. 7](#_35nkun2)

[Actual Timeline 8](#_1ksv4uv)

[Evidence of Design 9](#_44sinio)

[Wireframe 9](#_2jxsxqh)

[Flowchart 10](#_z337ya)

[UML Diagram 11](#_3j2qqm3)

[Technical Description & Walkthrough 12](#_1y810tw)

[Start Menu Page 12](#_4i7ojhp)

[Difficulty and Topics Page 13](#_1ci93xb)

[Proceeding and Initializing the Questions class 15](#_3whwml4)

[Updating score and displaying the correct answer 17](#_2bn6wsx)

[Displaying the final score and replaying the game/app 18](#_qsh70q)

[Libraries Used 20](#_3as4poj)

[Testing 21](#_1pxezwc)

[Final Testing 21](#_49x2ik5)

[Results 21](#_2p2csry)

[User Testing 22](#_147n2zr)

[User 1 22](#_3o7alnk)

[User 2 22](#_23ckvvd)

[User 3 22](#_ihv636)

[Critical Reflection 23](#_32hioqz)

[**Appendix 24**](#_1hmsyys)

# TechBlaizer

## Abstract

TechBlaizer is a Tech and Science Trivia based off the OpenTriviaDB API and it has 3 difficulties the user can choose and 5 different Topics relating to Technology and Science, and it always generates 5 random and different questions every time. It is fun to play, and it helps educate user on certain things they might as well know, This app can play/restarted until the player is satisfied

|  |  |
| --- | --- |
| Executable Project Folders | |
| Google Drive | <https://drive.google.com/drive/folders/1ORrkr6jVTR5BTJGwqe55wMt2Mt3KaDM-?usp=drive_link> |
| GitHub Link | [adhenzmiranda/TechBlaizers (github.com)](https://github.com/adhenzmiranda/TechBlaizers) |
| Other Links | |
| Project Resources | Figma:  <https://www.figma.com/file/AbIhJKDSmrVzFTSKXLIFSd/Code-Lab?type=design&node-id=0%3A1&mode=design&t=nHKFZINoM66C3aEf-1> |

## Project Plan

### Goals

These are the simple goals I have set for the Project:

* Understand the JSON API of OpenTrivia
* Implement and call the data through the Python file
* Simple UI Design
* Displaying and reading proper data from the API
* Scoring System and showing if the answer is correct or incorrect.
* Debugging the code and Visual Errors

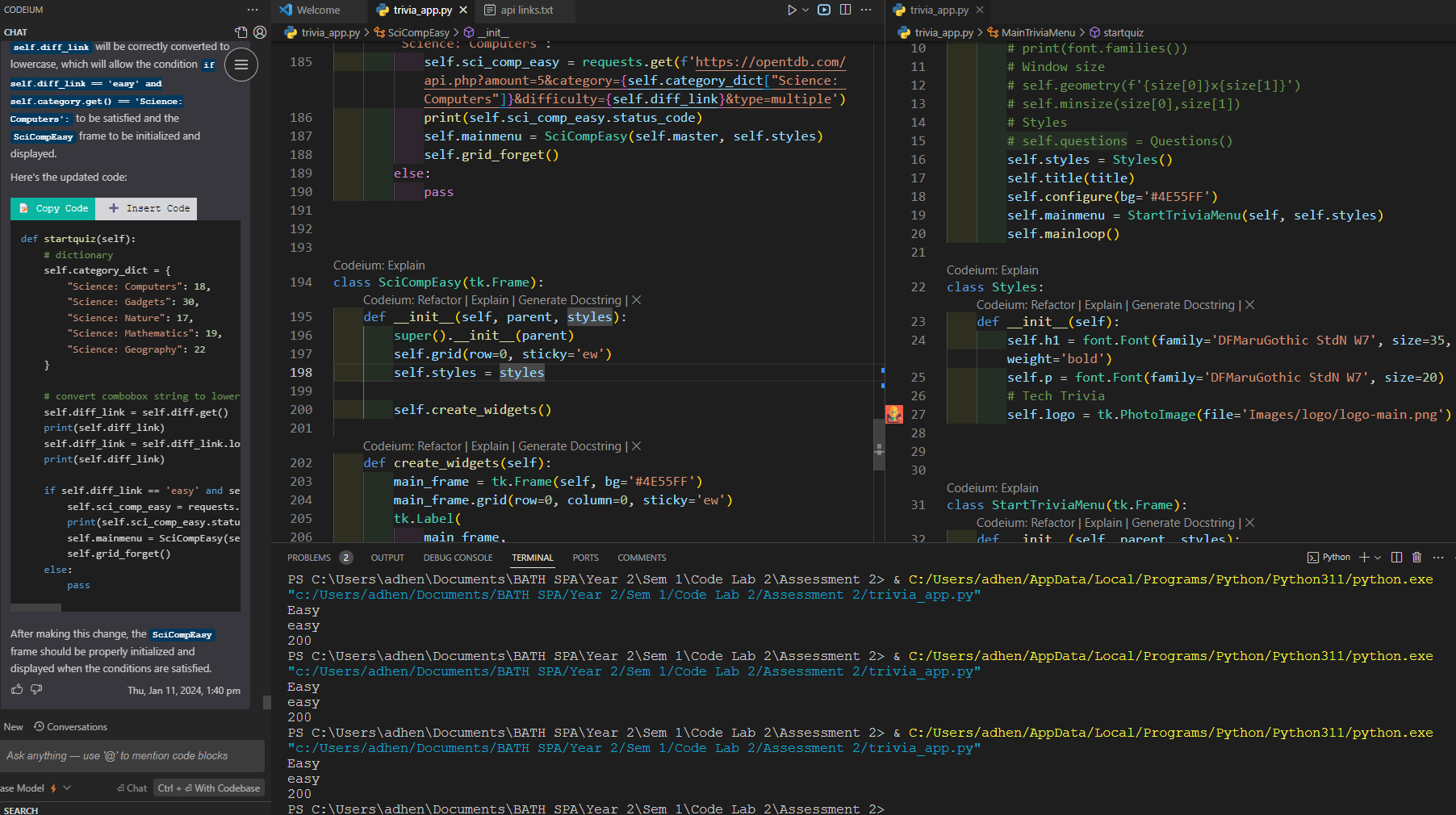
### Original Timeline

The original estimated completion time of the app was five working days. Each day was planned out as the following:

|  |  |
| --- | --- |
| Day | Plan |
| 1 | Understanding the JSON API and calling it from the Python file and implementing it with basic functions and classes, along with the scoring system |
| 2 | Understanding the UI/UX Flow of the user by establishing the main and basic classes and necessary frames |
| 3 | Designing and implementing the different pages of the App |
| 4 | Creating the wireframes and design of the app thru Figma |
| 5 | Debugging and polishing some visual errors within the app |

### Issues Faced

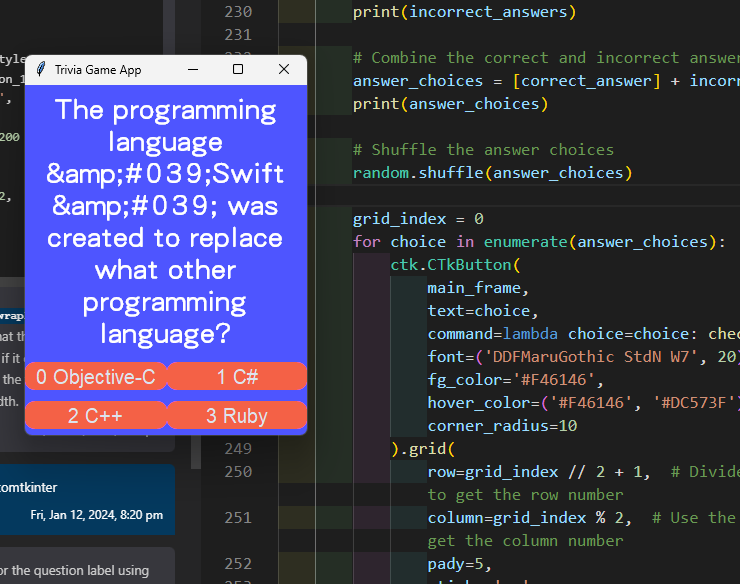
#### Linking the arguments and classes



This problem shows the case of why when I call a function to go to the next Frame class, I had to understand how to call the needed arguments and functions and transfer it over to the next class, and also because in the prior class it contains the URLs of the JSON file and I wanted it to be linked into the next frame when they initialize the class as it is the frame that I want to show to the user, in this case showing the first question in the API and the given answers.

This problem was simply solved by changing the function by adding the necessary arguments and order of the Boolean and if-else statement

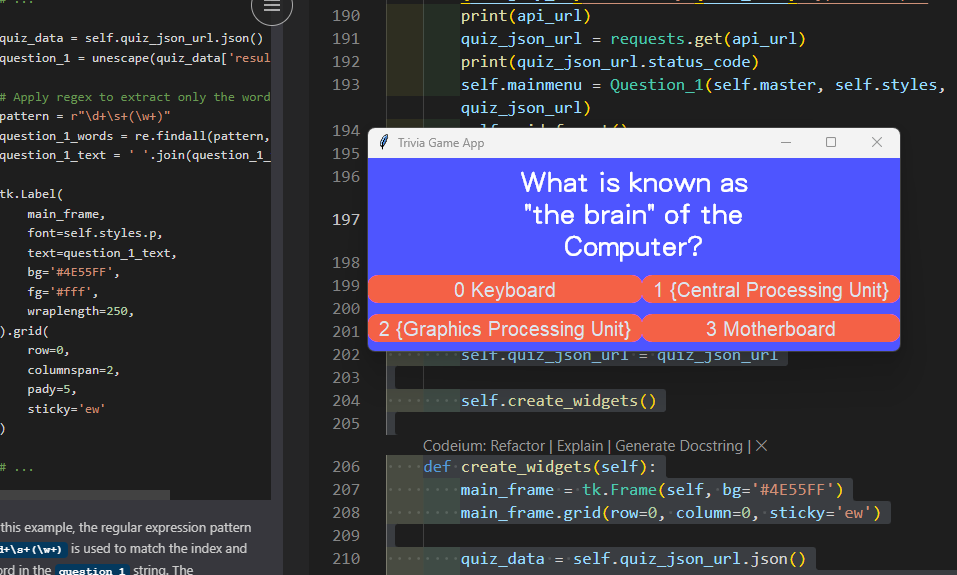
#### Converting HTML special characters into appropriate Python strings



As I was making and displaying the questions, some questions would come up and content HTML special characters which in this image as an example is “&amp;#039” which is supposedly an ampersand in normal strings rather than HTML.

This was simply solved by importing the HTML library in Python and calling the unescape function, which turns or converts the contents in the JSON file that contains the special characters and converts it into normal Python strings.

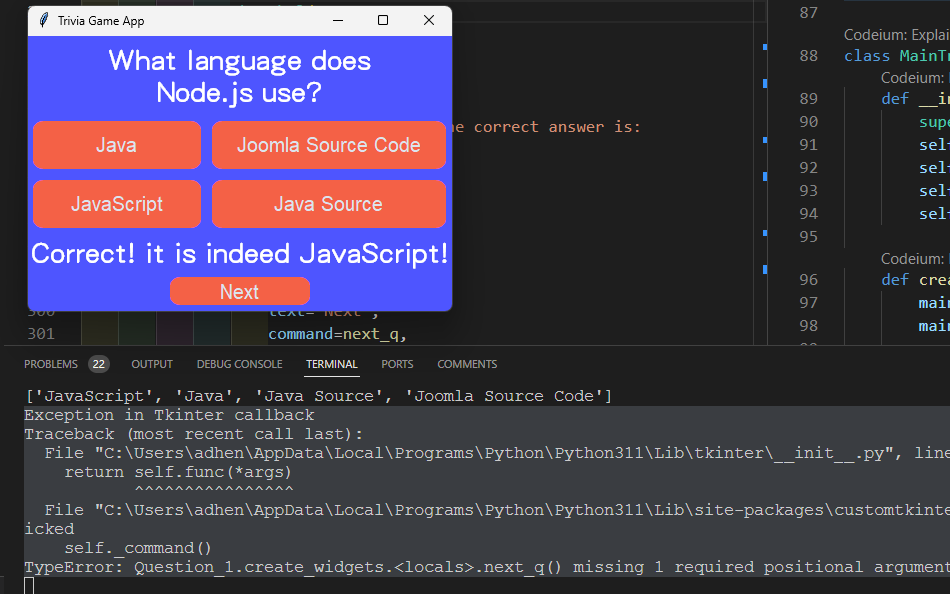
#### Numbers next to the listed text in the for loop



There was a case as I was coding the for loops for the buttons to be displayed, in the case where in OpenTrivia’s API where it lists the correct and incorrect answers as separate, I combined both of those lists and made into one custom lists, and shuffled the contents of that list so that it is not always in a fixed position when it gets displayed.

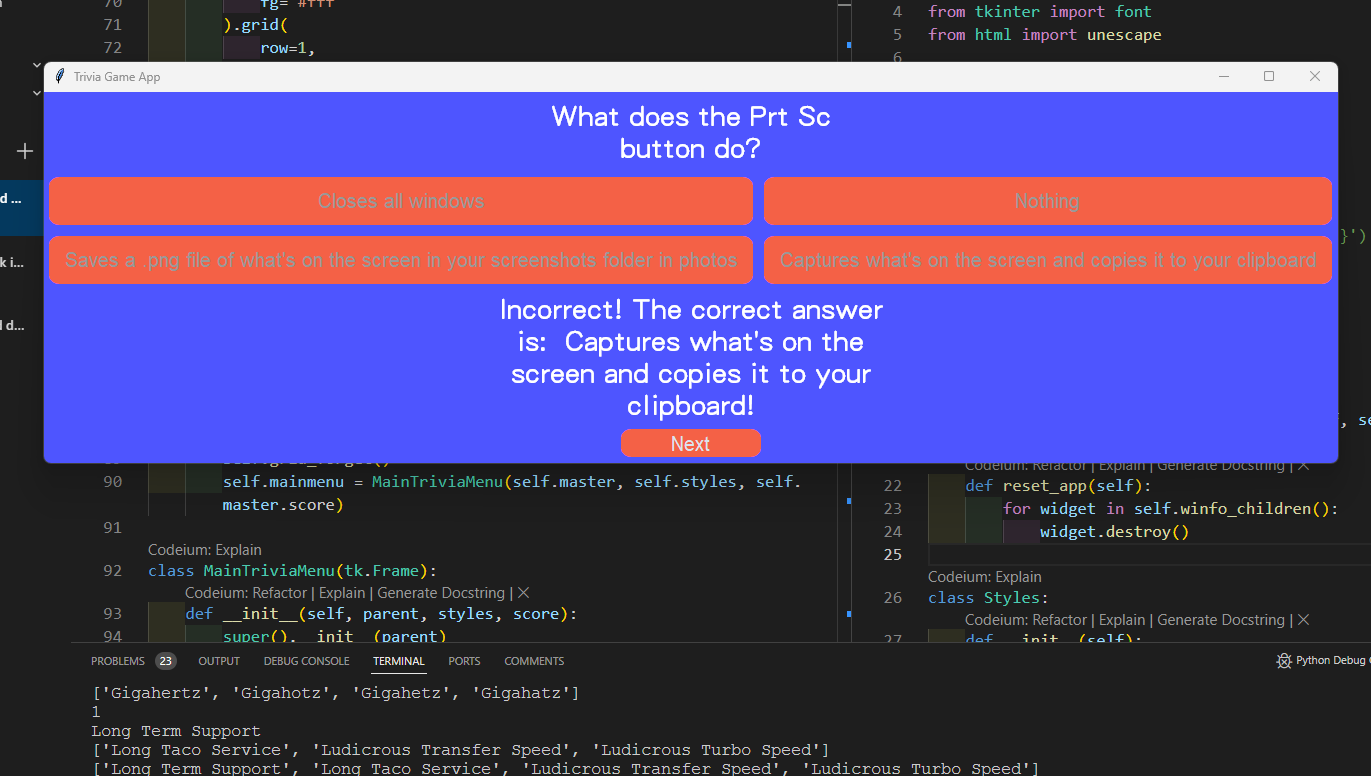
The solution for that was use the enumerate function in the list to just the combined list instead and normal for loop in turn displaying the buttons without the numbers.

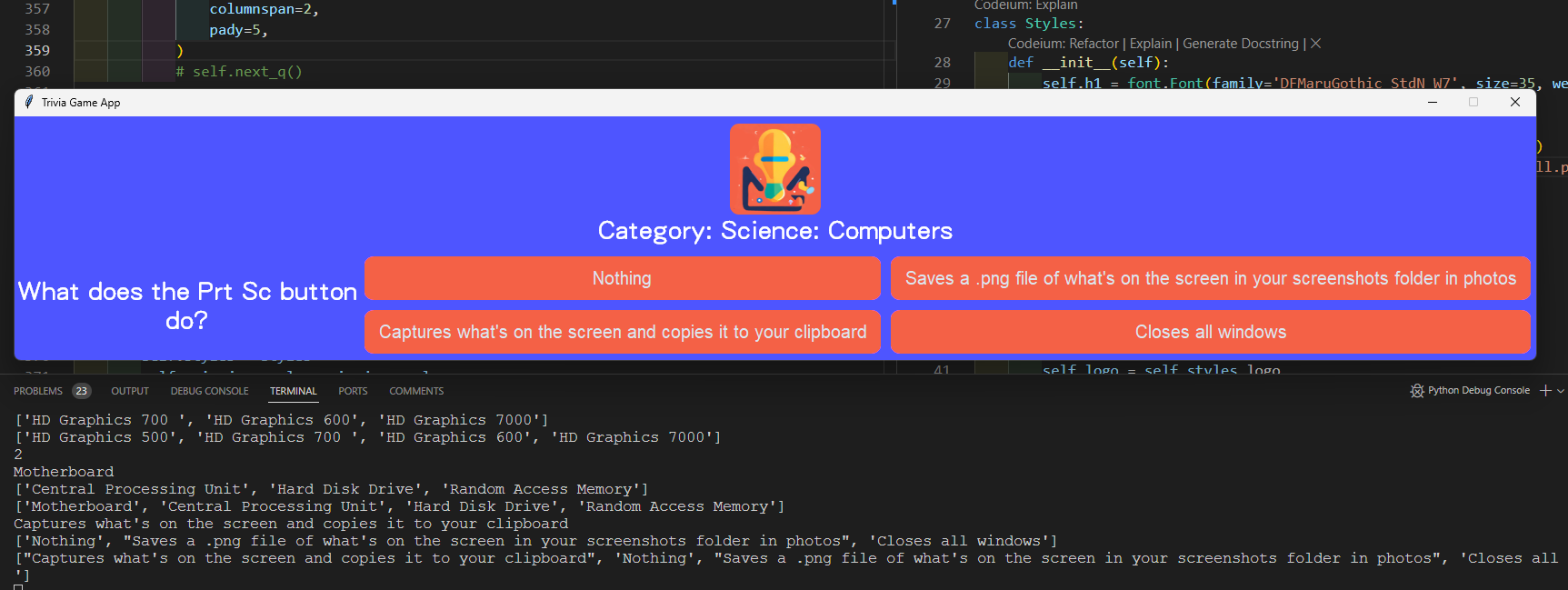
#### Functions not having appropriate arguments when going to the next class



Similar to the first problem, but in this case, it was missing some arguments that I overlooked in the code. It was solved by adding the necessary arguments to call the function to the next frame.

#### Disabling the button states to prevent score system abuse and element overflow in fixed windows.



This is so that the player doesn’t abuse the built-in scoring system within the app, It was solved by making it so the state of the button changes whenever they get the correct or incorrect answer, and as also presented in the image, it wasn’t always just one word for the buttons of the given answers in the API, sometimes it would contain long answers, in turn causing it to elongate or make it bigger width, which causes inconsistencies with the question, and of their sizes.

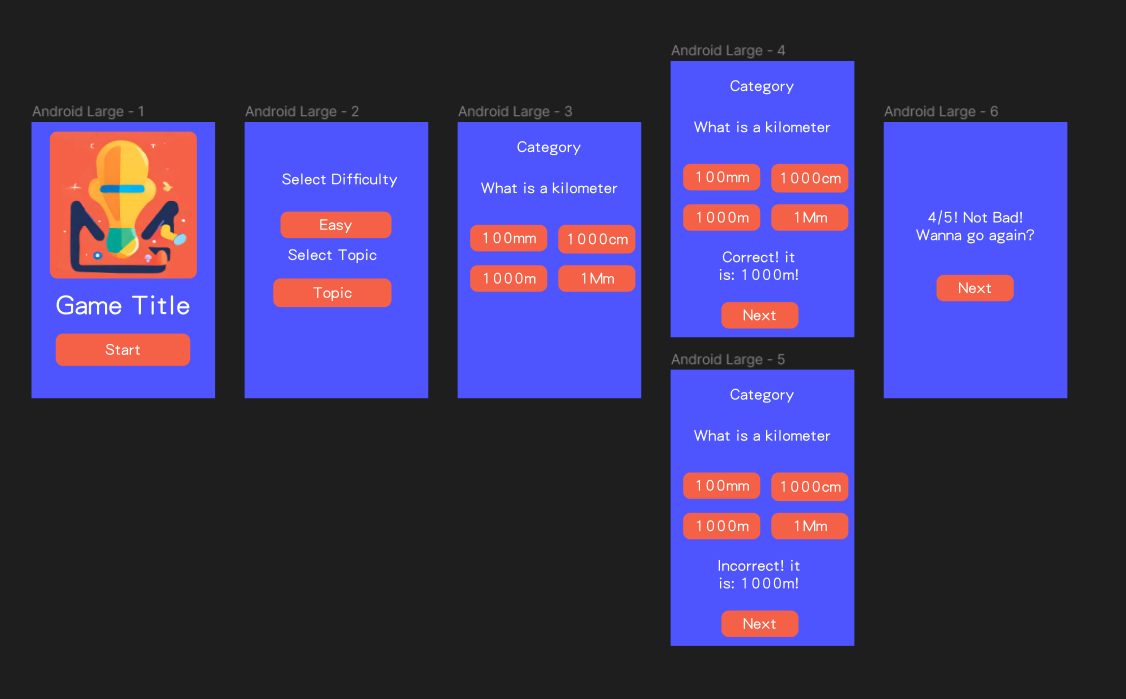
This problem I wasn’t able to fix as unlike the normaltk class where you can set the text wrap to be fixed, you cannot do it unfortunately with customtkinter, the getaway around of how I deal with this problem is simply by just using the grid\_forget functions and calling the new frame every time, as using a fixed size would cause as seen on the image, overflow and inconsistencies. Trying a different layout such as this makes it worse and looks unfavorable.

### Actual Timeline

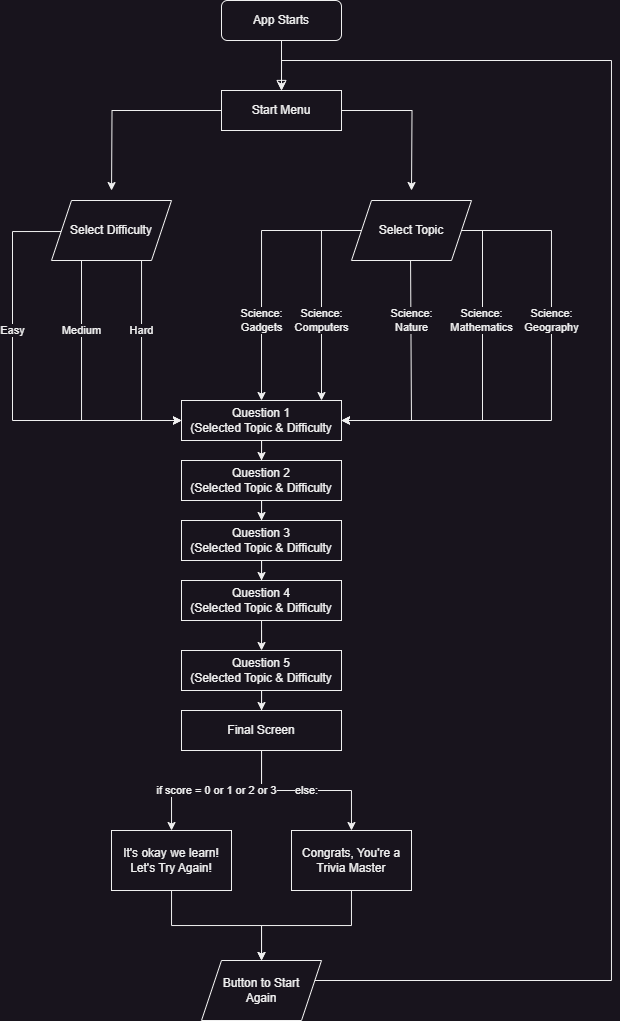
|  |  |
| --- | --- |
| Day | Events |
| 1 | Understanding the OpenTrivia API and building a simple base, and designing a logo, name, and theme of the Trivia App |
| 2 | Creating different classes and essential frames, along with implementing and tinkering with the online JSON API |
| 3 | Making the visual elements and using the customtkinter module |
| 4 | Simple visual elements and functionality implemented |
| 5 | Finalizing the structure and classes of the app; Designed a wireframe after finishing the layout of the app |
| 6 | Debugging the score system |
| 7 | Debugging the code and making the development document |

## Evidence of Design

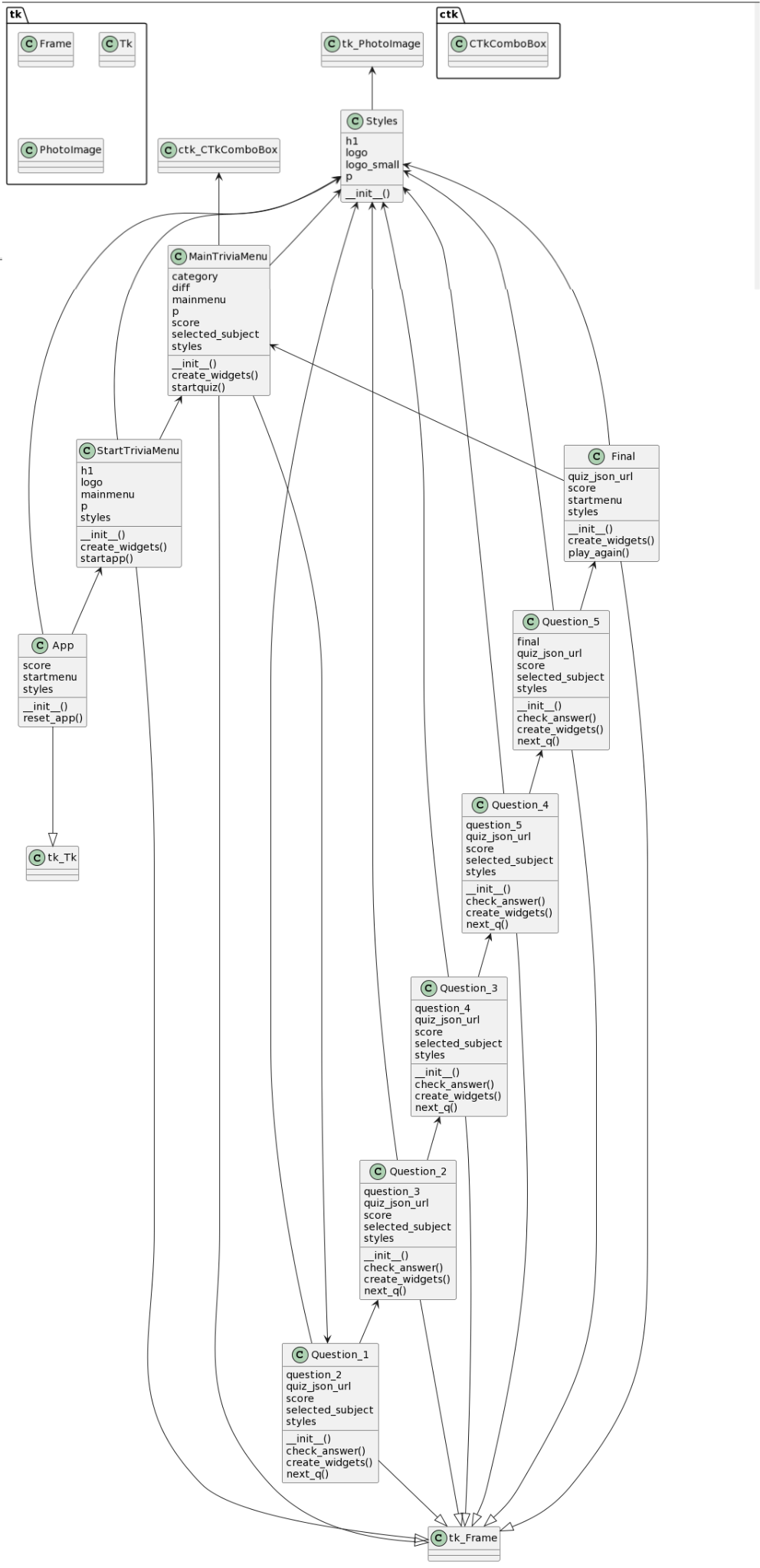
### Wireframe



### Flowchart



### UML Diagram



## Technical Description & Walkthrough

|  |  |
| --- | --- |
| YouTube Link (Walkthrough) | <https://youtu.be/8uGaB0jImR0> |
| YouTube Link  (Technical) | <https://youtu.be/yRluzk0EXDU> |
| Google Drive Link | <https://drive.google.com/drive/folders/1ORrkr6jVTR5BTJGwqe55wMt2Mt3KaDM-?usp=sharing> |

### Start Menu Page

This part of the app shows the main page of the app when you start it.

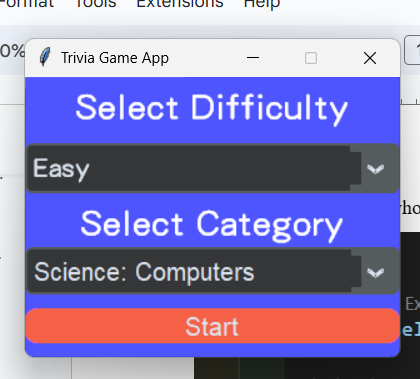


This code initializes the whole app and calls the next frame

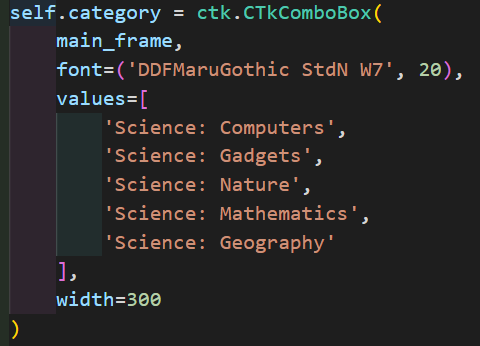
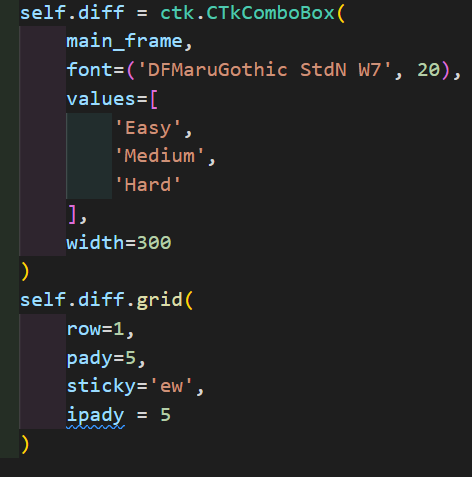
### 

### Difficulty and Topics Page

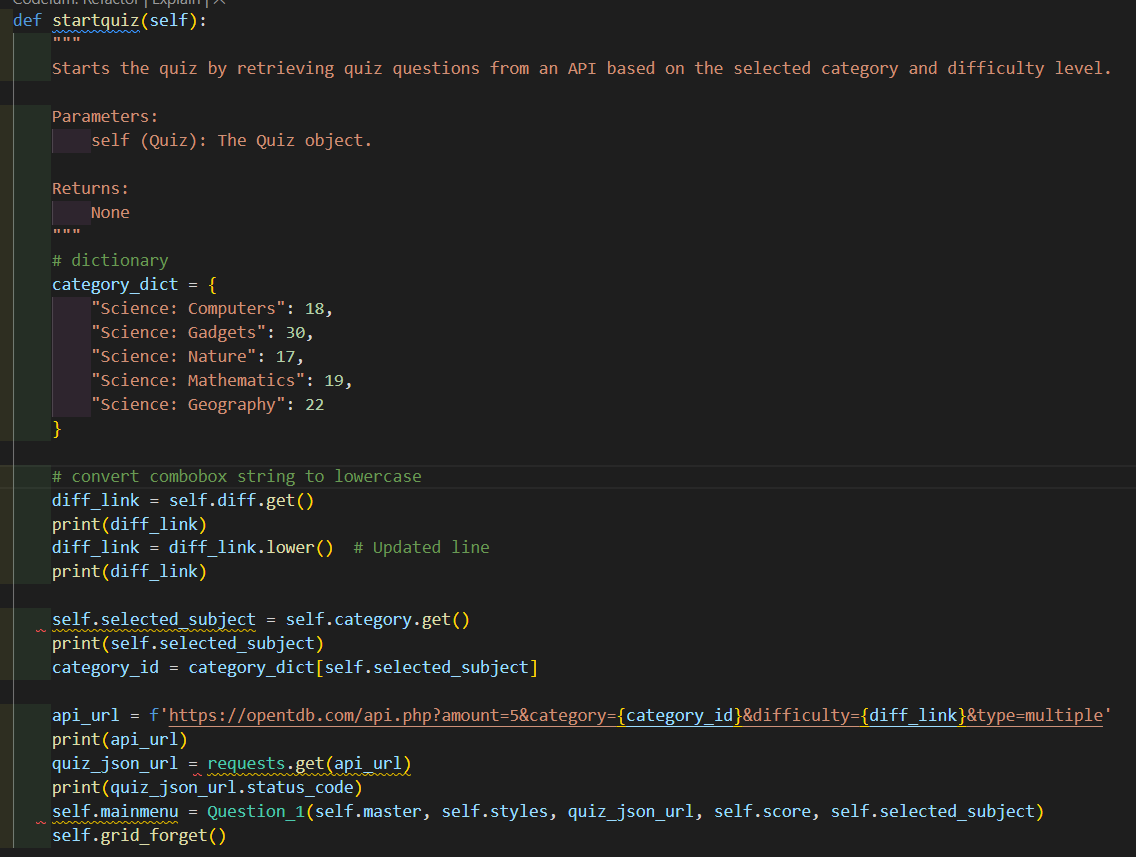
After clicking start, it will send the user to the page and choose their desired difficulty and category of topic they desire.



These 2 particular comboboxes are what makes and sets it, it has two variables, a category variable, and a difficulty variable, having these set defines and alters the API link based on what the user chooses



Calling the start\_quiz function calls and initializes the following



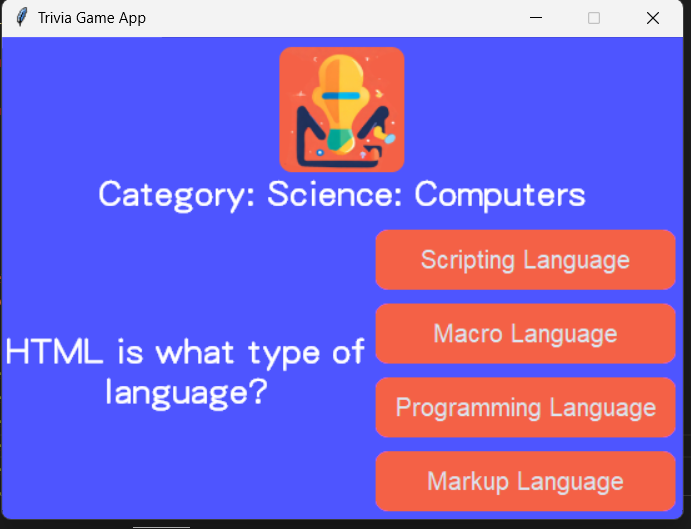
Since the API links follow a pattern of how the difficulty and categories are set, I first made a dictionary that defines the exact string as the combobox and declare it an integer value set to that particular topic in the API.

Next, I get the difficulty of String Value sin the combobox and since it is only a string that is needed to declare the difficulty for the API link, all i needed to do was just make a dedicated variable, get the string value in the combobox and convert it all into a lowercase string.

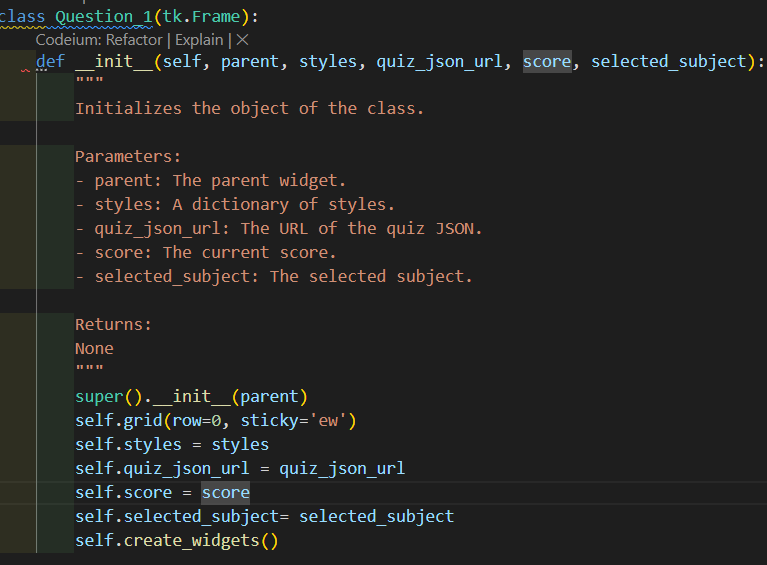
Next, I made another variable the gets the string values into combobox for the categories, and if that string matches the key of the values in the combobox, it will call that key and use that integer value in my category\_dict class, and made a new variable for category\_id which contains the value of the number.

Finally, I made a api\_url variable that contains the original API link, and I used a format string to alter the text of the API link and made a variable that calls and will contain the original content it gets in the by doing the requests.get function, and so that the initial variable I set for the for the api\_link, i can make it an argument to proceed to the following classes to display in each frame. And I call the initial class I want to call and display the first Question class and implement all the necessary variable argument needed for the next frame.

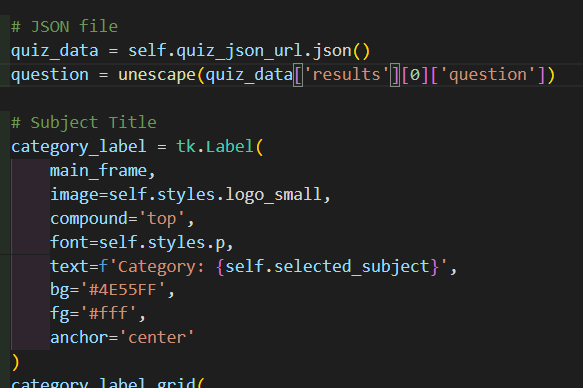
### Proceeding and Initializing the Questions class

****

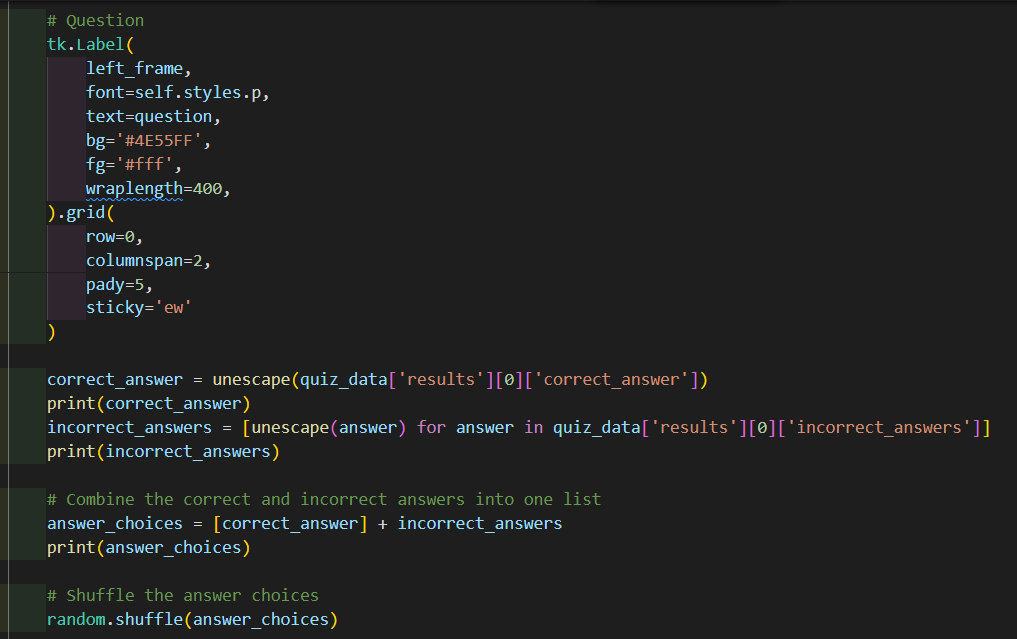
This is the page for each question, same layout, but each question and choices are different



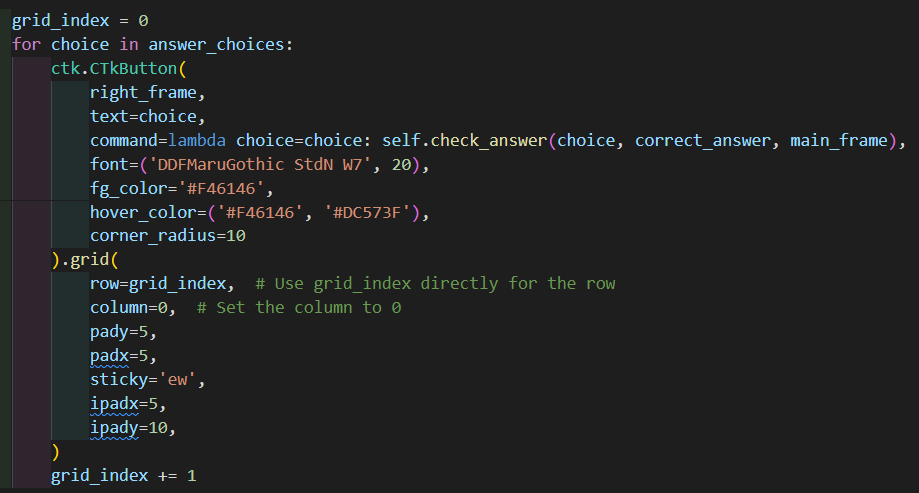
And these are the variables and arguments I used for me to call in the \_\_init\_\_ function, the same structure and format applies up until the last question where the final function is different, we’ll get to that later.



As previously stated, I made it so that when I call the quiz\_json\_url, i can just call it by making variable and initialize by using .json() function, and so when it reads the JSON it contains in, i can just call the contents like nested dictionaries, and use unescape to remove the HTML special characters.

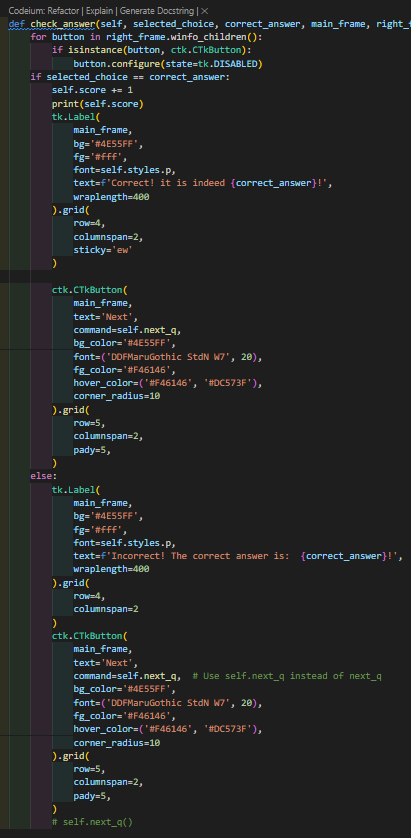


This part of the code makes it so that displays the question as seen on the previous Image and similar to the question variable, I made two variables containing the correct\_answers and incorrect\_answers and compile both of them into a list using list comprehension and shuffled the list using the random.shuffle function so it’s not always displayed in a fixed position when outputting the choices.

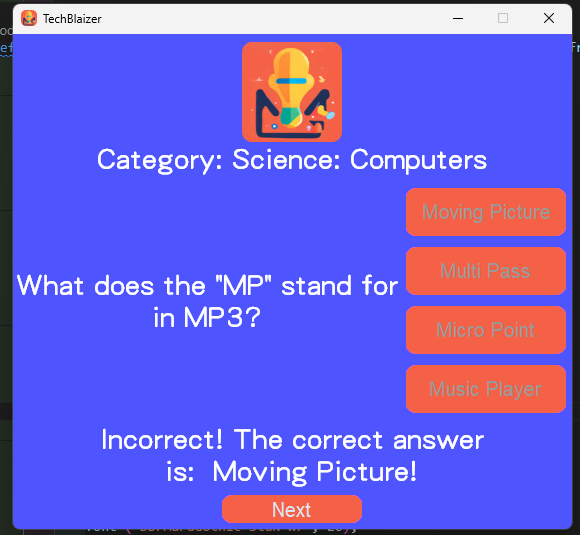
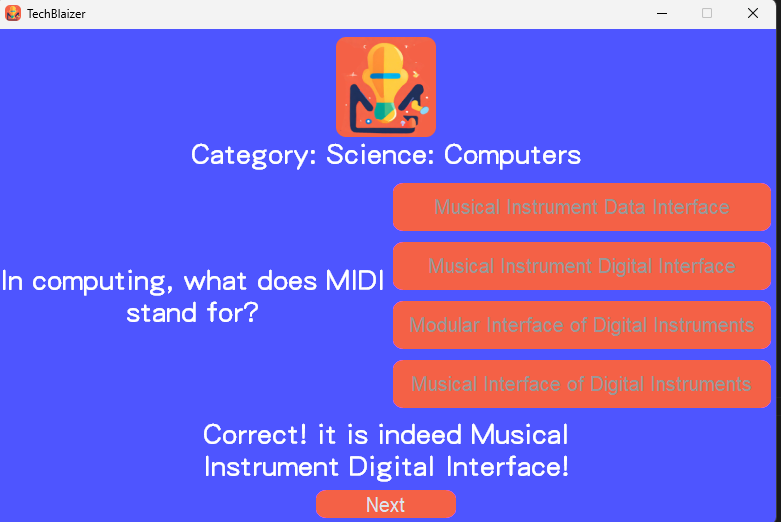


This code makes a loop of the 4 choices contained in the list that I made, so for every items that’s in the list, that’ how much it will loop for, and it will create 4 buttons in a single file column

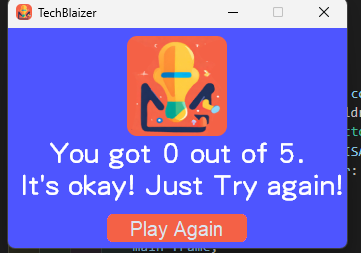
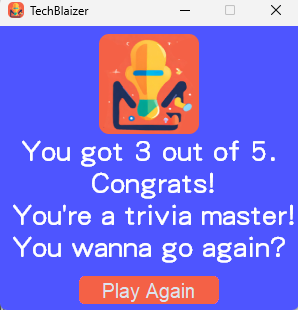
### Updating score and displaying the correct answer

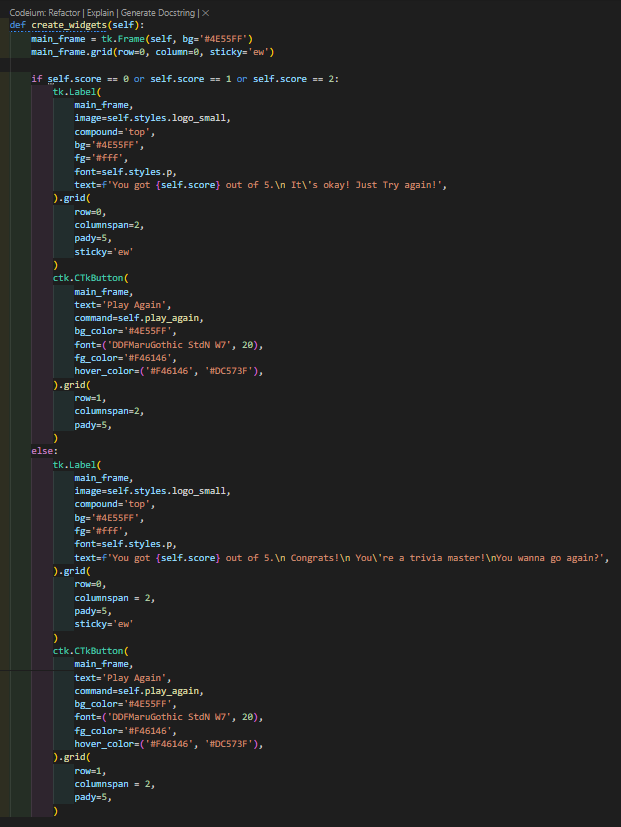


This particular function checks if their answer is correct, the scores variable and updates in increments of 1 and display that they are correct along with the correct answer, else it would just say they were incorrect, but it will still display the incorrect answer



### Displaying the final score and replaying the game/app



This takes us to the final page of the app which displays the final scores and a button for them to play again  


This code shows a condition if they either get 0, 1 or 2 as their score, it will display a message, and in the else it will display something different

### Libraries Used

|  |  |
| --- | --- |
| Making the Python GUI |  |
| Accessing the API |  |
| Shuffling the choices |  |
| Formatting/Converting HTML special characters into normal Python String |  |

## Testing

### Final Testing

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Page** | **Name** | **Steps** | **Expectation** | **Result** |
| N/A | Initializing the App | Running the Python File | No errors should load | Pass |
| Start Menu | Main Starting Frame of the App | Click Start Button | Directs to the Difficulty and Topic Page/ Main Trivia Menu class | Pass |
| Main Trivia Menu | Prompting the Difficulty and Topics | Clicking the Arrow down on the UI/Combobox and Clicking Start after choosing | Being able to Prompt the Difficulty and Topics they choose, and clicking start directs to the next question frames | Pass |
| Question Page Frames (1-5) | Displaying Appropriate Questions and prompts | Clicking any choice disables the choices and shows the user the correct answer and show a button that goes to the next question | Capable of displaying the initial question from the API along with the button choices and the initial functionality and a button that will proceed to the next Question until the Final frame | Pass |
| Final/Scores Page | Displaying the Score | Displays the total scores, and a button to play again | Shows their final score and a button to proceed to the difficulty and topics page | Pass |

#### Results

\* Pass - The process or function works as intended without problems.

\* Unfavourable - The process or function works as intended but faces some issues.

\* Fail - The process or function failed to function at all.

|  |  |
| --- | --- |
| Test Result Resources | |
| Google Drive Link | <https://drive.google.com/drive/folders/1PIHj1OrLrqdRk2h39KxobiHnUhJNpnt7?usp=sharing> |
| Github Link | [TechBlaizers/Testing Folder at main · adhenzmiranda/TechBlaizers (github.com)](https://github.com/adhenzmiranda/TechBlaizers/tree/main/Testing%20Folder) |

### User Testing

#### User 1

Issues:

Once the user comes to the second page there are some minor visual bugs that can be seen from the combo box

|  |
| --- |
| Feedback:   1. The starting screen is a bit small compared to the other screens of the app 2. I think the idea of the quiz game is very fun and it gives the user a bit of joy rather than making it seem like a chore 3. The app design is well thought but i feel like the color choice could have been better |

#### User 2

Issues:

N/A

|  |
| --- |
| Feedback  I like that it looks simple, yet it also implements the app for its own design layout, and the quiz is actually quite fun to play. |

#### User 3

Issues:

N/A

|  |
| --- |
| Feedback:  The app was well made and there were no errors or issues that occurred. It was really fun and interesting. |

## Critical Reflection

Creating a Python GUI is honestly quite a fun challenge to do, and it is also the first time I have ever handled APIs and a GUI Library at that, and honestly if I had more time developing this app, it could be better, and there are definitely lots of improvements I could list.

Number one, dealing with the classes, I feel as though there could have been a better way, I could have written classes and I could have also shortened some parts of the code I probably overlooked as I developed the app.

And I felt I could’ve added more features such as adding a user prompt on if they want to do True or False which the OpenTrivia API offers, or mix-match both True and False and Multiple choice which is the only limited choice of choosing the choices in the App.

And there were definitely some parts of the code I could have structured better rather than the current code I have written and there were also parts of the GUI where rather than switching to the next class by forgetting and calling the frames, I could have used tkraise function but I feel even though with that it might no help with the overflowing of elements depending on the choices from the API which I tried solving but I did what I could with the time I had and It is definitely better than nothing

With all these aside, if I were to compare myself a few months ago where I didn’t know anything about the Tkinter Python GUI or how APIs work, and even forgetting some basics from the refresher assessment, I feel as though and can’t help but feel proud with what I can make, and I am willing to take these learning that I have made from making this data-driven app and apply it to my future projects and endeavors and I will keep note and track of how I am definitely going to deal and encounter with such problems later on.

# Appendix

import tkinter as tk

from html import unescape

import random

from tkinter import font

import requests

import customtkinter as ctk

class App(tk.Tk):

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

"""

Initializes the object with a given title.

Parameters:

title (str): The title of the object.

Returns:

None

"""

super().\_\_init\_\_()

self.styles = Styles()

self.title(title)

self.configure(bg='#4E55FF')

self.score = 0

self.startmenu = StartTriviaMenu(self, self.styles)

self.resizable(False, False)

self.iconbitmap('Images/logo/logo-ico.ico')

self.mainloop()

def reset\_app(self):

"""

Reset the app by destroying all widgets.

Parameters:

None

Returns:

None

Pseudocode:

1. Loop through each widget in the app's children

2. Destroy the widget

"""

for widget in self.winfo\_children():

widget.destroy()

class Styles:

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

"""

Initializes the class and sets the initial values for the instance variables.

Parameters:

self (object): The instance of the class.

Returns:

None

"""

self.h1 = font.Font(family='DFMaruGothic StdN W7', size=35, weight='bold')

self.p = font.Font(family='DFMaruGothic StdN W7', size=20)

# Tech Trivia

self.logo = tk.PhotoImage(file='Images/logo/logo-main.png')

self.logo\_small = tk.PhotoImage(file='Images/logo/logo-small.png')

class StartTriviaMenu(tk.Frame):

def \_\_init\_\_(self, parent, styles):

"""

Initializes the object with the given parent widget and styles.

Parameters:

parent (Widget): The parent widget.

styles (Styles): The styles object.

Returns:

None

"""

super().\_\_init\_\_(parent)

self.styles = styles

self.h1 = self.styles.h1

self.p = self.styles.p

self.logo = self.styles.logo

self.configure(bg='#4E55FF')

self.grid(row=0, sticky='ew')

self.create\_widgets()

def create\_widgets(self):

"""

Creates and configures the widgets for the GUI.

Returns:

None

"""

# Frame

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.columnconfigure(0, weight=1)

main\_frame.grid(row=0)

# Logo Image

# Label

tk.Label(

main\_frame,

font=self.h1,

compound='top',

image=self.logo,

text='TechBlaizer',

bg='#4E55FF',

fg='#fff'

).grid(

row=0,

pady=5,

)

tk.Label(

main\_frame,

font=self.p,

text='A Tech and Science\nTrivia Game',

bg='#4E55FF',

fg='#fff'

).grid(

row=1,

)

ctk.CTkButton(

main\_frame,

text='Start',

corner\_radius=10,

fg\_color='#F46146',

font=('DFMaruGothic StdN W7', 20),

hover\_color=('#F46146', '#DC573F'),

command=self.startapp

).grid(

pady=10,

row=2

)

def startapp(self):

"""

Start the app by hiding the current grid and creating a new instance of the MainTriviaMenu class.

Parameters:

None

Returns:

None

"""

self.grid\_forget()

self.mainmenu = MainTriviaMenu(self.master, self.styles, self.master.score)

class MainTriviaMenu(tk.Frame):

def \_\_init\_\_(self, parent, styles, score):

"""

Initializes an instance of the class.

Args:

parent: The parent widget.

styles: The styles used for the widgets.

score: The score value.

Returns:

None.

"""

super().\_\_init\_\_(parent)

self.styles = styles

self.p = self.styles.p

self.score = score

self.grid(row=0, sticky='ew')

self.create\_widgets()

def create\_widgets(self):

"""

Creates and configures the widgets for the user interface of the application.

Parameters:

- None

Returns:

- None

"""

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.columnconfigure(0, weight=1)

main\_frame.grid(row=0, column=0, sticky='ew')

tk.Label(

main\_frame,

font=self.styles.p,

text='Select Difficulty',

bg='#4E55FF',

fg='#fff'

).grid(

row=0,

pady=5,

sticky='ew'

)

self.diff = ctk.CTkComboBox(

main\_frame,

font=('DFMaruGothic StdN W7', 20),

values=[

'Easy',

'Medium',

'Hard'

],

width=300

)

self.diff.grid(

row=1,

pady=5,

sticky='ew',

ipady = 5

)

tk.Label(

main\_frame,

font=self.styles.p,

text='Select Category',

bg='#4E55FF',

fg='#fff'

).grid(

row=2,

sticky='ew'

)

self.category = ctk.CTkComboBox(

main\_frame,

font=('DDFMaruGothic StdN W7', 20),

values=[

'Science: Computers',

'Science: Gadgets',

'Science: Nature',

'Science: Mathematics',

'Science: Geography'

],

width=300

)

self.category.grid(

row=3,

sticky='ew',

ipady = 5

)

ctk.CTkButton(

main\_frame,

text='Start',

corner\_radius=10,

fg\_color='#F46146',

font=('DDFMaruGothic StdN W7', 20),

hover\_color=('#F46146', '#DC573F'),

command=self.startquiz

).grid(

row=4,

pady=10,

)

def startquiz(self):

"""

Starts the quiz by retrieving quiz questions from an API based on the selected category and difficulty level.

Parameters:

self (Quiz): The Quiz object.

Returns:

None

"""

# dictionary

# convert combobox string to lowercase

diff\_link = self.diff.get()

print(diff\_link)

diff\_link = diff\_link.lower() # Updated line

print(diff\_link)

category\_dict = {

"Science: Computers": 18,

"Science: Gadgets": 30,

"Science: Nature": 17,

"Science: Mathematics": 19,

"Science: Geography": 22

}

self.selected\_subject = self.category.get()

print(self.selected\_subject)

category\_id = category\_dict[self.selected\_subject]

api\_url = f'https://opentdb.com/api.php?amount=5&category={category\_id}&difficulty={diff\_link}&type=multiple'

print(api\_url)

quiz\_json\_url = requests.get(api\_url)

print(quiz\_json\_url.status\_code)

self.mainmenu = Question\_1(self.master, self.styles, quiz\_json\_url, self.score, self.selected\_subject)

self.grid\_forget()

class Question\_1(tk.Frame):

def \_\_init\_\_(self, parent, styles, quiz\_json\_url, score, selected\_subject):

"""

Initializes the object of the class.

Parameters:

- parent: The parent widget.

- styles: A dictionary of styles.

- quiz\_json\_url: The URL of the quiz JSON.

- score: The current score.

- selected\_subject: The selected subject.

Returns:

None

"""

super().\_\_init\_\_(parent)

self.grid(row=0, sticky='ew')

self.styles = styles

self.quiz\_json\_url = quiz\_json\_url

self.score = score

self.selected\_subject= selected\_subject

self.create\_widgets()

def create\_widgets(self):

# Frames

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.grid(row=0, column=0, sticky='ew')

left\_frame = tk.Frame(main\_frame, bg='#4E55FF')

left\_frame.grid(row=1, column=0, sticky='ew')

right\_frame = tk.Frame(main\_frame, bg='#4E55FF')

right\_frame.grid(row=1, column=1, sticky='ew')

bottom\_frame = tk.Frame(main\_frame, bg='#4E55FF')

bottom\_frame.grid(row=2, column=0, columnspan=2, sticky='ew')

# JSON file

quiz\_data = self.quiz\_json\_url.json()

question = unescape(quiz\_data['results'][0]['question'])

# Subject Title

category\_label = tk.Label(

main\_frame,

image=self.styles.logo\_small,

compound='top',

font=self.styles.p,

text=f'Category: {self.selected\_subject}',

bg='#4E55FF',

fg='#fff',

anchor='center'

)

category\_label.grid(

row=0,

column=0,

columnspan=2,

pady=5,

sticky='ew'

)

# Question

tk.Label(

left\_frame,

font=self.styles.p,

text=question,

bg='#4E55FF',

fg='#fff',

wraplength=400,

anchor='w'

).grid(

row=0,

columnspan=2,

pady=5,

sticky='ew'

)

correct\_answer = unescape(quiz\_data['results'][0]['correct\_answer'])

print(correct\_answer)

incorrect\_answers = [unescape(answer) for answer in quiz\_data['results'][0]['incorrect\_answers']]

print(incorrect\_answers)

# Combine the correct and incorrect answers into one list

answer\_choices = [correct\_answer] + incorrect\_answers

print(answer\_choices)

# Shuffle the answer choices

random.shuffle(answer\_choices)

grid\_index = 0

for choice in answer\_choices:

ctk.CTkButton(

right\_frame,

text=choice,

command=lambda choice=choice: self.check\_answer(choice, correct\_answer, main\_frame, right\_frame),

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=grid\_index, # Use grid\_index directly for the row

column=0, # Set the column to 0

pady=5,

padx=5,

sticky='ew',

ipadx=5,

ipady=10,

)

grid\_index += 1

def check\_answer(self, selected\_choice, correct\_answer, main\_frame, right\_frame):

for button in right\_frame.winfo\_children():

if isinstance(button, ctk.CTkButton):

button.configure(state=tk.DISABLED)

if selected\_choice == correct\_answer:

self.score += 1

print(self.score)

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Correct! it is indeed {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2,

sticky='ew'

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

else:

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Incorrect! The correct answer is: {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

# self.next\_q()

def next\_q(self):

self.question\_2 = Question\_2(self.master, self.styles, self.quiz\_json\_url, self.score, self.selected\_subject)

self.grid\_forget()

class Question\_2(tk.Frame):

def \_\_init\_\_(self, parent, styles, quiz\_json\_url, score, selected\_subject):

super().\_\_init\_\_(parent)

self.grid(row=0, sticky='ew')

self.styles = styles

self.quiz\_json\_url = quiz\_json\_url

self.score = score

self.selected\_subject= selected\_subject

self.create\_widgets()

def create\_widgets(self):

# Frames

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.grid(row=0, column=0, sticky='ew')

left\_frame = tk.Frame(main\_frame, bg='#4E55FF')

left\_frame.grid(row=1, column=0, sticky='ew')

right\_frame = tk.Frame(main\_frame, bg='#4E55FF')

right\_frame.grid(row=1, column=1, sticky='ew')

bottom\_frame = tk.Frame(main\_frame, bg='#4E55FF')

bottom\_frame.grid(row=2, column=0, columnspan=2, sticky='ew')

# JSON file

quiz\_data = self.quiz\_json\_url.json()

question = unescape(quiz\_data['results'][1]['question'])

# Subject Title

category\_label = tk.Label(

main\_frame,

image=self.styles.logo\_small,

compound='top',

font=self.styles.p,

text=f'Category: {self.selected\_subject}',

bg='#4E55FF',

fg='#fff',

anchor='center'

)

category\_label.grid(

row=0,

column=0,

columnspan=2,

pady=5,

sticky='ew'

)

# Question

tk.Label(

left\_frame,

font=self.styles.p,

text=question,

bg='#4E55FF',

fg='#fff',

wraplength=400,

anchor='w'

).grid(

row=0,

columnspan=2,

pady=5,

sticky='ew'

)

correct\_answer = unescape(quiz\_data['results'][1]['correct\_answer'])

print(correct\_answer)

incorrect\_answers = [unescape(answer) for answer in quiz\_data['results'][1]['incorrect\_answers']]

print(incorrect\_answers)

# Combine the correct and incorrect answers into one list

answer\_choices = [correct\_answer] + incorrect\_answers

print(answer\_choices)

# Shuffle the answer choices

random.shuffle(answer\_choices)

grid\_index = 0

for choice in answer\_choices:

ctk.CTkButton(

right\_frame,

text=choice,

command=lambda choice=choice: self.check\_answer(choice, correct\_answer, main\_frame, right\_frame),

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=grid\_index, # Use grid\_index directly for the row

column=0, # Set the column to 0

pady=5,

padx=5,

sticky='ew',

ipadx=5,

ipady=10,

)

grid\_index += 1

def check\_answer(self, selected\_choice, correct\_answer, main\_frame, right\_frame):

for button in right\_frame.winfo\_children():

if isinstance(button, ctk.CTkButton):

button.configure(state=tk.DISABLED)

if selected\_choice == correct\_answer:

self.score += 1

print(self.score)

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Correct! it is indeed {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2,

sticky='ew'

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

else:

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Incorrect! The correct answer is: {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q, # Use self.next\_q instead of next\_q

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

# self.next\_q()

def next\_q(self):

self.question\_3 = Question\_3(self.master, self.styles, self.quiz\_json\_url, self.score, self.selected\_subject)

self.grid\_forget()

class Question\_3(tk.Frame):

def \_\_init\_\_(self, parent, styles, quiz\_json\_url, score, selected\_subject):

super().\_\_init\_\_(parent)

self.grid(row=0, sticky='ew')

self.styles = styles

self.quiz\_json\_url = quiz\_json\_url

self.score = score

self.selected\_subject= selected\_subject

self.create\_widgets()

def create\_widgets(self):

# Frames

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.grid(row=0, column=0, sticky='ew')

left\_frame = tk.Frame(main\_frame, bg='#4E55FF')

left\_frame.grid(row=1, column=0, sticky='ew')

right\_frame = tk.Frame(main\_frame, bg='#4E55FF')

right\_frame.grid(row=1, column=1, sticky='ew')

bottom\_frame = tk.Frame(main\_frame, bg='#4E55FF')

bottom\_frame.grid(row=2, column=0, columnspan=2, sticky='ew')

# JSON file

quiz\_data = self.quiz\_json\_url.json()

question = unescape(quiz\_data['results'][2]['question'])

# Subject Title

category\_label = tk.Label(

main\_frame,

image=self.styles.logo\_small,

compound='top',

font=self.styles.p,

text=f'Category: {self.selected\_subject}',

bg='#4E55FF',

fg='#fff',

anchor='center'

)

category\_label.grid(

row=0,

column=0,

columnspan=2,

pady=5,

sticky='ew'

)

# Question

tk.Label(

left\_frame,

font=self.styles.p,

text=question,

bg='#4E55FF',

fg='#fff',

wraplength=400,

anchor='w'

).grid(

row=0,

columnspan=2,

pady=5,

sticky='ew'

)

correct\_answer = unescape(quiz\_data['results'][2]['correct\_answer'])

print(correct\_answer)

incorrect\_answers = [unescape(answer) for answer in quiz\_data['results'][2]['incorrect\_answers']]

print(incorrect\_answers)

# Combine the correct and incorrect answers into one list

answer\_choices = [correct\_answer] + incorrect\_answers

print(answer\_choices)

# Shuffle the answer choices

random.shuffle(answer\_choices)

grid\_index = 0

for choice in answer\_choices:

ctk.CTkButton(

right\_frame,

text=choice,

command=lambda choice=choice: self.check\_answer(choice, correct\_answer, main\_frame, right\_frame),

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=grid\_index, # Use grid\_index directly for the row

column=0, # Set the column to 0

pady=5,

padx=5,

sticky='ew',

ipadx=5,

ipady=10,

)

grid\_index += 1

def check\_answer(self, selected\_choice, correct\_answer, main\_frame, right\_frame):

for button in right\_frame.winfo\_children():

if isinstance(button, ctk.CTkButton):

button.configure(state=tk.DISABLED)

if selected\_choice == correct\_answer:

self.score += 1

print(self.score)

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Correct! it is indeed {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2,

sticky='ew'

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

else:

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Incorrect! The correct answer is: {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q, # Use self.next\_q instead of next\_q

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

# self.next\_q()

def next\_q(self):

self.question\_4 = Question\_4(self.master, self.styles, self.quiz\_json\_url, self.score, self.selected\_subject)

self.grid\_forget()

class Question\_4(tk.Frame):

def \_\_init\_\_(self, parent, styles, quiz\_json\_url, score, selected\_subject):

super().\_\_init\_\_(parent)

self.grid(row=0, sticky='ew')

self.styles = styles

self.quiz\_json\_url = quiz\_json\_url

self.score = score

self.selected\_subject= selected\_subject

self.create\_widgets()

def create\_widgets(self):

# Frames

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.grid(row=0, column=0, sticky='ew')

left\_frame = tk.Frame(main\_frame, bg='#4E55FF')

left\_frame.grid(row=1, column=0, sticky='ew')

right\_frame = tk.Frame(main\_frame, bg='#4E55FF')

right\_frame.grid(row=1, column=1, sticky='ew')

bottom\_frame = tk.Frame(main\_frame, bg='#4E55FF')

bottom\_frame.grid(row=2, column=0, columnspan=2, sticky='ew')

# JSON file

quiz\_data = self.quiz\_json\_url.json()

question = unescape(quiz\_data['results'][3]['question'])

# Subject Title

category\_label = tk.Label(

main\_frame,

image=self.styles.logo\_small,

compound='top',

font=self.styles.p,

text=f'Category: {self.selected\_subject}',

bg='#4E55FF',

fg='#fff',

anchor='center'

)

category\_label.grid(

row=0,

column=0,

columnspan=2,

pady=5,

sticky='ew'

)

# Question

tk.Label(

left\_frame,

font=self.styles.p,

text=question,

bg='#4E55FF',

fg='#fff',

wraplength=400,

anchor='w'

).grid(

row=0,

columnspan=2,

pady=5,

sticky='ew'

)

correct\_answer = unescape(quiz\_data['results'][3]['correct\_answer'])

print(correct\_answer)

incorrect\_answers = [unescape(answer) for answer in quiz\_data['results'][3]['incorrect\_answers']]

print(incorrect\_answers)

# Combine the correct and incorrect answers into one list

answer\_choices = [correct\_answer] + incorrect\_answers

print(answer\_choices)

# Shuffle the answer choices

random.shuffle(answer\_choices)

grid\_index = 0

for choice in answer\_choices:

ctk.CTkButton(

right\_frame,

text=choice,

command=lambda choice=choice: self.check\_answer(choice, correct\_answer, main\_frame, right\_frame),

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=grid\_index, # Use grid\_index directly for the row

column=0, # Set the column to 0

pady=5,

padx=5,

sticky='ew',

ipadx=5,

ipady=10,

)

grid\_index += 1

def check\_answer(self, selected\_choice, correct\_answer, main\_frame, right\_frame):

for button in right\_frame.winfo\_children():

if isinstance(button, ctk.CTkButton):

button.configure(state=tk.DISABLED)

if selected\_choice == correct\_answer:

self.score += 1

print(self.score)

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Correct! it is indeed {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2,

sticky='ew'

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

else:

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Incorrect! The correct answer is: {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q, # Use self.next\_q instead of next\_q

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

# self.next\_q()

def next\_q(self):

self.question\_5 = Question\_5(self.master, self.styles, self.quiz\_json\_url, self.score, self.selected\_subject)

self.grid\_forget()

class Question\_5(tk.Frame):

def \_\_init\_\_(self, parent, styles, quiz\_json\_url, score, selected\_subject):

super().\_\_init\_\_(parent)

self.grid(row=0, sticky='ew')

self.styles = styles

self.quiz\_json\_url = quiz\_json\_url

self.score = score

self.selected\_subject= selected\_subject

self.create\_widgets()

def create\_widgets(self):

# Frames

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.grid(row=0, column=0, sticky='ew')

left\_frame = tk.Frame(main\_frame, bg='#4E55FF')

left\_frame.grid(row=1, column=0, sticky='ew')

right\_frame = tk.Frame(main\_frame, bg='#4E55FF')

right\_frame.grid(row=1, column=1, sticky='ew')

bottom\_frame = tk.Frame(main\_frame, bg='#4E55FF')

bottom\_frame.grid(row=2, column=0, columnspan=2, sticky='ew')

# JSON file

quiz\_data = self.quiz\_json\_url.json()

question = unescape(quiz\_data['results'][4]['question'])

# Subject Title

category\_label = tk.Label(

main\_frame,

image=self.styles.logo\_small,

compound='top',

font=self.styles.p,

text=f'Category: {self.selected\_subject}',

bg='#4E55FF',

fg='#fff',

anchor='center'

)

category\_label.grid(

row=0,

column=0,

columnspan=2,

pady=5,

sticky='ew'

)

# Question

tk.Label(

left\_frame,

font=self.styles.p,

text=question,

bg='#4E55FF',

fg='#fff',

wraplength=400,

anchor='w'

).grid(

row=0,

columnspan=2,

pady=5,

sticky='ew'

)

correct\_answer = unescape(quiz\_data['results'][4]['correct\_answer'])

print(correct\_answer)

incorrect\_answers = [unescape(answer) for answer in quiz\_data['results'][4]['incorrect\_answers']]

print(incorrect\_answers)

# Combine the correct and incorrect answers into one list

answer\_choices = [correct\_answer] + incorrect\_answers

print(answer\_choices)

# Shuffle the answer choices

random.shuffle(answer\_choices)

grid\_index = 0

for choice in answer\_choices:

ctk.CTkButton(

right\_frame,

text=choice,

command=lambda choice=choice: self.check\_answer(choice, correct\_answer, main\_frame, right\_frame),

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=grid\_index, # Use grid\_index directly for the row

column=0, # Set the column to 0

pady=5,

padx=5,

sticky='ew',

ipadx=5,

ipady=10,

)

grid\_index += 1

def check\_answer(self, selected\_choice, correct\_answer, main\_frame, right\_frame):

for button in right\_frame.winfo\_children():

if isinstance(button, ctk.CTkButton):

button.configure(state=tk.DISABLED)

if selected\_choice == correct\_answer:

self.score += 1

print(self.score)

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Correct! it is indeed {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2,

sticky='ew'

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

else:

tk.Label(

main\_frame,

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'Incorrect! The correct answer is: {correct\_answer}!',

wraplength=400

).grid(

row=4,

columnspan=2

)

ctk.CTkButton(

main\_frame,

text='Next',

command=self.next\_q, # Use self.next\_q instead of next\_q

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

corner\_radius=10

).grid(

row=5,

columnspan=2,

pady=5,

)

# self.next\_q()

def next\_q(self):

self.final = Final(self.master, self.styles, self.quiz\_json\_url, self.score)

self.grid\_forget()

class Final(tk.Frame):

def \_\_init\_\_(self, parent, styles, quiz\_json\_url, score):

super().\_\_init\_\_(parent)

self.grid(row=0, sticky='ew')

self.styles = styles

self.quiz\_json\_url = quiz\_json\_url

self.score = score

self.create\_widgets()

def create\_widgets(self):

main\_frame = tk.Frame(self, bg='#4E55FF')

main\_frame.grid(row=0, column=0, sticky='ew')

if self.score == 0 or self.score == 1 or self.score == 2:

tk.Label(

main\_frame,

image=self.styles.logo\_small,

compound='top',

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'You got {self.score} out of 5.\n It\'s okay! Just Try again!',

).grid(

row=0,

columnspan=2,

pady=5,

sticky='ew'

)

ctk.CTkButton(

main\_frame,

text='Play Again',

command=self.play\_again,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

).grid(

row=1,

columnspan=2,

pady=5,

)

else:

tk.Label(

main\_frame,

image=self.styles.logo\_small,

compound='top',

bg='#4E55FF',

fg='#fff',

font=self.styles.p,

text=f'You got {self.score} out of 5.\n Congrats!\n You\'re a trivia master!\nYou wanna go again?',

).grid(

row=0,

columnspan = 2,

pady=5,

sticky='ew'

)

ctk.CTkButton(

main\_frame,

text='Play Again',

command=self.play\_again,

bg\_color='#4E55FF',

font=('DDFMaruGothic StdN W7', 20),

fg\_color='#F46146',

hover\_color=('#F46146', '#DC573F'),

).grid(

row=1,

columnspan = 2,

pady=5,

)

def play\_again(self):

self.grid\_forget()

self.startmenu = MainTriviaMenu(self.master, self.styles, self.master.score)

App('Trivia Game App')