

from typing import List, Tuple

import random

import requests

import html

import time

import threading

import os

def main():

quiz = Quiz()

quiz.play()

if \_\_name\_\_ == "\_\_main\_\_":

from quiz import initialize

initialize()

scoresheet: List[Tuple[str, int]] = []

class Quiz:

# jeryk

def \_\_init\_\_(self, num\_questions=5, difficulty=1, mode="classic"):

self.scores = []

self.correct\_answers = []

self.incorrect\_answers = []

self.num\_questions = num\_questions

self.difficulty = difficulty

self.mode = mode

self.questions = self.load\_questions()

self.lives = 5 # For survival mode

def load\_questions(self):

# Fetch questions from Open Trivia Database (opentdb.com)

difficulty\_map = {1: "easy", 2: "medium", 3: "hard"}

url = f"https://opentdb.com/api.php?amount={self.num\_questions}&difficulty={difficulty\_map.get(self.difficulty, 'medium')}&type=multiple&category=18"

response = requests.get(url)

questions\_data = response.json()

questions = []

if not questions\_data.get('results'):

print("Failed to fetch questions. Please check your internet connection or try again later.")

return []

for item in questions\_data['results']:

question = html.unescape(item['question'])

correct\_answer = html.unescape(item['correct\_answer'])

incorrect\_answers = [html.unescape(answer) for answer in item['incorrect\_answers']]

choices = [correct\_answer] + incorrect\_answers

random.shuffle(choices)

correct\_index = choices.index(correct\_answer)

formatted\_question = {"question": question, "choices": choices, "correct": correct\_index}

questions.append(formatted\_question)

return questions

# ryu

def get\_user\_input(self, prompt, accept\_only\_ABCD=True):

user\_input = input(prompt)

if not accept\_only\_ABCD or user\_input.upper() in ['A', 'B', 'C', 'D']:

return user\_input.upper() if accept\_only\_ABCD else user\_input

else:

print("Please enter a valid choice (A, B, C, D)." if accept\_only\_ABCD else "Invalid input, please try again.")

return self.get\_user\_input(prompt, accept\_only\_ABCD)

def play(self):

if self.mode == "1":

self.play\_classic()

elif self.mode == "2":

self.play\_timed()

elif self.mode == "3":

self.play\_survival()

else:

print(f"Unknown mode: {self.mode}")

# ardy

def play\_classic(self):

for question in self.questions:

print(question["question"])

for idx, choice in enumerate(question["choices"]):

print(f"{chr(idx+65)}. {choice}")

answer = self.get\_user\_input("Your answer: ").upper()

if ord(answer) - 65 == question["correct"]:

print("Correct!")

self.scores.append(1)

self.correct\_answers.append(question["question"])

else:

print("Wrong!")

self.scores.append(0)

self.incorrect\_answers.append((question["question"], question["choices"], question["correct"]))

self.show\_scoresheet()

def play\_timed(self):

def wait\_for\_input(question, choices):

timer = 15

print(question)

for idx, choice in enumerate(choices):

print(f"{chr(idx+65)}. {choice}")

answer = [None]

def get\_user\_answer():

answer[0] = self.get\_user\_input("Your answer: \n", accept\_only\_ABCD=True).upper()

input\_thread = threading.Thread(target=get\_user\_answer)

input\_thread.start()

start\_time = time.time()

while timer > 0 and input\_thread.is\_alive():

print(f"Time left: {timer}s", end="\r")

time.sleep(1)

timer = 15 - int(time.time() - start\_time)

if timer <= 0:

print("\nTime's up!")

if not input\_thread.is\_alive():

input\_thread.join()

else:

input\_thread.join(timeout=1)

break

return answer[0]

for question in self.questions:

user\_answer = wait\_for\_input(question["question"], question["choices"])

if user\_answer is None or ord(user\_answer) - 65 != question["correct"]:

print("Wrong!")

self.scores.append(0)

self.incorrect\_answers.append((question["question"], question["choices"], question["correct"]))

else:

print("Correct!")

self.scores.append(1)

self.correct\_answers.append(question["question"])

self.show\_scoresheet()

def play\_survival(self):

for question in self.questions:

print(f"Lives: {self.lives}")

print(question["question"])

for idx, choice in enumerate(question["choices"]):

print(f"{chr(idx+65)}. {choice}")

answer = self.get\_user\_input("Your answer: ", accept\_only\_ABCD=True)

if ord(answer) - 65 == question["correct"]:

print("Correct!")

self.scores.append(1)

self.correct\_answers.append(question["question"])

else:

print("Wrong!")

self.scores.append(0)

self.incorrect\_answers.append((question["question"], question["choices"], question["correct"]))

self.lives -= 1

if self.lives == 0:

print("Game Over!")

self.scores.append(0)

self.show\_scoresheet()

return

self.show\_scoresheet()

# rojan

def show\_scoresheet(self):

if (len(scoresheet) > 0):

print("Scoresheet:")

total\_score = sum(self.scores)

print(f"Your score: {total\_score}/{len(self.questions)}")

player\_name = input("Enter your name for the scoresheet: ")

mode\_text = "(Classic)" if self.mode == "1" else "(Timed)" if self.mode == "2" else "(Survival)"

self.update\_scoresheet(f"{player\_name} {mode\_text} - {total\_score}/{len(self.questions)}")

print(f"Thank you, {player\_name}. Your score has been recorded.")

# Placeholder for displaying scoresheet. This should be replaced with actual scoresheet display logic.

while True:

print("Enter 'N' to start a new quiz, 'Q' to view quiz summary, 'S' to view the scoresheet, or enter any other key to exit.")

action = self.get\_user\_input("Choose an option: ", accept\_only\_ABCD=False)

if action.lower() == 'n':

initialize()

elif action.lower() == 'q':

self.quiz\_summary()

elif action.lower() == 's':

self.display\_scoresheet()

else:

print("Thank you for playing!")

break

def update\_scoresheet(self, player\_name\_with\_mode):

global scoresheet

scoresheet.append((player\_name\_with\_mode))

# scoresheet = sorted(scoresheet, key=lambda x: x[1], reverse=True) # Sort by score in descending order

# parcia

def display\_scoresheet(self):

if (len(scoresheet) > 0):

print("Scoresheet:")

for idx, name in enumerate(scoresheet): # Display top 10 scores

print(f"{idx+1}. {name}")

input("Press enter to return to the main menu.")

initialize()

def quiz\_summary(self):

print("\nQuiz Summary:")

print("Correct Answers:")

for question in self.correct\_answers:

print(f"\n- {question}")

print("\nIncorrect Answers:")

for question, choices, correct in self.incorrect\_answers:

print(f"\n- Question: {question}")

print(" Choices:")

for idx, choice in enumerate(choices):

print(f" {chr(idx+65)}. {choice}")

print(f" Correct Answer: {choices[correct]}")

print("\n")

# lim

def initialize():

os.system('cls')

print("Welcome to the Team Allstar Computer Trivia Quiz!")

while True:

mode = input("Choose a game mode (1. Classic, 2. Timed, 3. Survival): ")

if mode in ["1", "2", "3"]:

break

else:

print("Invalid mode selected. Please choose from '1. Classic', '2. Timed', or '3. Survival'.")

num\_questions = 30 if mode == "3" else None

while num\_questions is None:

try:

num\_questions = int(input("How many questions would you like? (5-30): "))

if 5 <= num\_questions <= 30:

break

else:

print("Number of questions must be between 5 and 30.")

except ValueError:

print("Please enter a valid number.")

while True:

try:

difficulty\_input = input("Select difficulty (1: Easy, 2: Medium, 3: Hard): ")

if difficulty\_input.lower() in ["1", "easy"]:

difficulty = 1

break

elif difficulty\_input.lower() in ["2", "medium"]:

difficulty = 2

break

elif difficulty\_input.lower() in ["3", "hard"]:

difficulty = 3

break

else:

print("Please select a valid difficulty level (1: Easy, 2: Medium, 3: Hard).")

except ValueError:

print("Please enter a valid number.")

quiz = Quiz(num\_questions=num\_questions, difficulty=difficulty, mode=mode)

quiz.play()