Module **game**

Functions

def Login_Signup(user_profiles: dict, player: int)

Prompts the user to login, signup or view the leaderboard. Used to get the user information for gameplay when logging in, to populate the global user_profiles dictionary with a user name and default player statistics {"elo": 1500, "wins":0} when signing up. Reprompts user to login or signup if an invalid username or action is input.

Args

user_profiles : dict
A dictionary containing the information of users in the following format {"username": {"elo":x,"wins":x}}

Returns

list
upon a succesful login it returns a list containing True and the username inputted. In the format

[True, "username"]

def P1_P2_hangman()

P1_P2_hangman() takes no arguments or inputs; it repeats the hangman() function a total of 6 times for 6 total rounds of hangman. (3 rounds for player 1 and 3 rounds for player 2). Upon completion of the 6 rounds, the function compares the player 1 and player 2's scores and determines a winner or a draw. In the event of a draw, function does not update global lives. If a winner is determined, the loser loses a global life.

def P1_P2_login()

Repeats the Login_Signup function twice for player1 and player 2. uses the information retrieved from the Login_Signup function to populate the global active_users dictionary with the username and default starting lives of 3. In the format: {"player1": {"username":"xxxx","global_lives":3},"player1":{"username":"xxxx",global_lives":3}} also displays the game's title "BRAINBuddy". The function does not take any inputs.

def choice(country_dict)

This function randomly chooses a country with its city and population, from the country

dictionary. To avoid repeated questions, the chosen country would be removed from the country dictionary.

Args

`country_dict` : `dict`
dictionary where country is the key while city and population are the values

Returns

`str`
the randomly chosen country and its city

`int`
population of the randomly chosen country

`def clearScreenAfterDelay(delay=3)`

Clears the screen after a specified delay.

Args

`delay` : `int` , optional
The number of seconds to wait before clearing the screen. Defaults to 3.

`def displayLeaderboard()`

Displays the top 10 user profiles, sorted according to their elo scores.

`def elo_and_wins_update(P1_rating: int, P2_rating: int, outcome: dict)`

Calculates a player's new elo and win after game is concluded. Updates the new calculated elo and wins to the global user_profile dictionary.

Args

`P1_rating` : `int`
The integer value of player1's elo, obtained from the user_profiles dictionary

`P2_rating` : `int`
The integer value of player2's elo, obtained from the user_profiles dictionary

`outcome` : `dict`
dictionary containing who won or lost the game. example of outcome dictionary format: {"P1":1, "P2":0} (1 for won and 0 for lost.)

def gameTimer(time_per_question=8)

This function starts an 8 second timer. After 8 seconds has elapsed it exits. It also exits if it detects that the global 'exit_thread_event' event has been toggled. If it has, then it resets the event before terminating its execution.

Args

time_per_question : `int` , optional
The number of seconds that the timer runs until. Defaults to 8.

def gameplay()

Runs through the game. The function calls on returnPopulatedUserProfiles() to read the 'user_data.csv' file into the global 'user_profiles' variable. After which, it calls on the displayLeaderboard() function and the P1_P2_login() function to prompt for the user to select an action.

The function then continues to call the promptUserForGame() function until the lives of either player1 or player2 reach 0, after which the elo and wins for each player is updated using the elo_and_wins_update() function. Lastly, the function calls on the saveUserProfileToCSV() function output the updated player information to a csv file and displays the leaderboard before finally returning, and thus ending the execution of the program.

def generateMathQuestion()

The function randomly chooses between generating one of three different forms of mathematical equations. It returns the generated equation along with its solution.

For each type of equation, three different mathematical operators are randomly chosen from the 'math_operators' list. The kind of equation to generate is randomly chosen by generating a random integer between 0 and 2 and using match, case statements based on the value of the random integer.

Returns

`str`
The equation that is to be displayed to the user.

`int`
The solution to the equation.

def get_country_dict()

This function opens and reads the csv file as a dictionary. The data is then reassigned to a new dictionary (country_dict), where country is the key while city and population are the values.

Returns

`dict`
: dictionary where country is the key while city and population are the values

def hangman(word: str, player: int, round: int, country: str)

Inititiates the basic hangman game and logic. prompts user for a character input to guess the selected word. prints "correct!" for correct guesses and returns error messages for invalid inputs (such as spaces or empty inputs). user is allowed to enter 'hint' to get a hint (function populates one of the blinded letters).

Args

word : `str`
: A string representing the word or country's capital to be guessed.

player : `int`
: an int representing the player number (1 or 2) used for the interface and as keys in user dictionaries

round : `int`
: an int representing the current round number, used for the interface

country : `str`
: a string of a randomly chosen country name, which will be used as a key for the get_countries dictionaries

def higher_lower()

This function calls the one_player() function for each player to play the game. The returned score is then used to determine who won. The player who lost would have one of their global life deducted and the player's individual global lives left would then be shown.

def one_player()

This function first calls the get_country_dict() function to retrieves a country dictionary containing the population and city of each country. A while loop is used to consecutively launch questions until a player gets a question wrong. In the while loop, it calls the choice() function which randomly chooses a country with its city and population, from

the country dictionary. The question on which given city has a higher or lower population is then prompted and a point is awarded if their given answer matches our calculations.

Returns

`int`
: total points

`def playMathGame()`

This function plays through the math game for both players. It calls the startMathGame function, providing for each player 1 and player 2, then after stores the returned value as 'player_1_score' and 'player_2_score'. It later compares their scores to determine the outcome of the math game - whether any player won or if it's a draw - and decrements the global life of the player accordingly.

`def promptUserForGame()`

This function prompts the user to choose the game that they want to play between 3 different games. Based on the user's input, it goes on to execute the function that starts the game that they've chosen to play.

Returns

`None`
: The return statement executes the function respective to the game that the player wants

to play, and returns it. Hence, as it's executing the function in the return statement, the function returns a None type.

`def returnPopulatedUserProfiles()`

Reads the csv file user_data.csv and processes the items into a dictionary called 'user_profiles' which the function returns.

Returns

`dict`
: 'user_profiles' dictionary with user's username as key and value as a dictionary

with their elo number of wins.

`def saveUserProfileToCSV()`

Overwrites the user_data.csv file with the updated information from the 'user_profiles' dictionary.

def startMathGame(user)

Runs the math game until a the player runs out of attempts, after which the function terminates and returns the score for the player.

The function retrieves the equation and its solution from the generateMathQuestion() function.

The function starts the gameTimer() function on a separate thread for each question that is asked and upon user input, checks if the global 'time_up' flag is set as True, if it is, then the function runs through the loop again.

There is also input validation performed within the function, where the user input is checked to be numeric using the is_digit() method and the '-' is stripped from the start of the input to allow the user to input negative numbers without getting a False value from is_digit() for the validation check.

Comparing the user input with the solution, the function decrements the local lives (attempts) of the player accordingly if they get the answer wrong and awards +10 points if the user input is the correct answer.

Args

user : `str`
The username of the player currently playing the game.

Returns

`int`
The final score of the player for the game.

Index

Functions