



USCS23

Assignment 1 Report

Project Name:

Trivial Triumph

Group:

Khilfi Bin Khairul Amin (23029065)
Nur Yasmin Suraya Binti Sapar (23038422)

Lecturer:

Ms. Hanin Binti Abdul Rahman

Submission Date:

February 20th, 2024

Table of Content

Introduction.....	3
README File.....	4
Selected Codes.....	5
1. quiz.py.....	5
2. menus.py.....	11
3. db.py.....	20
4. app.py.....	25
5. auth.py.....	27
6. ui.py.....	30
Conclusion.....	37

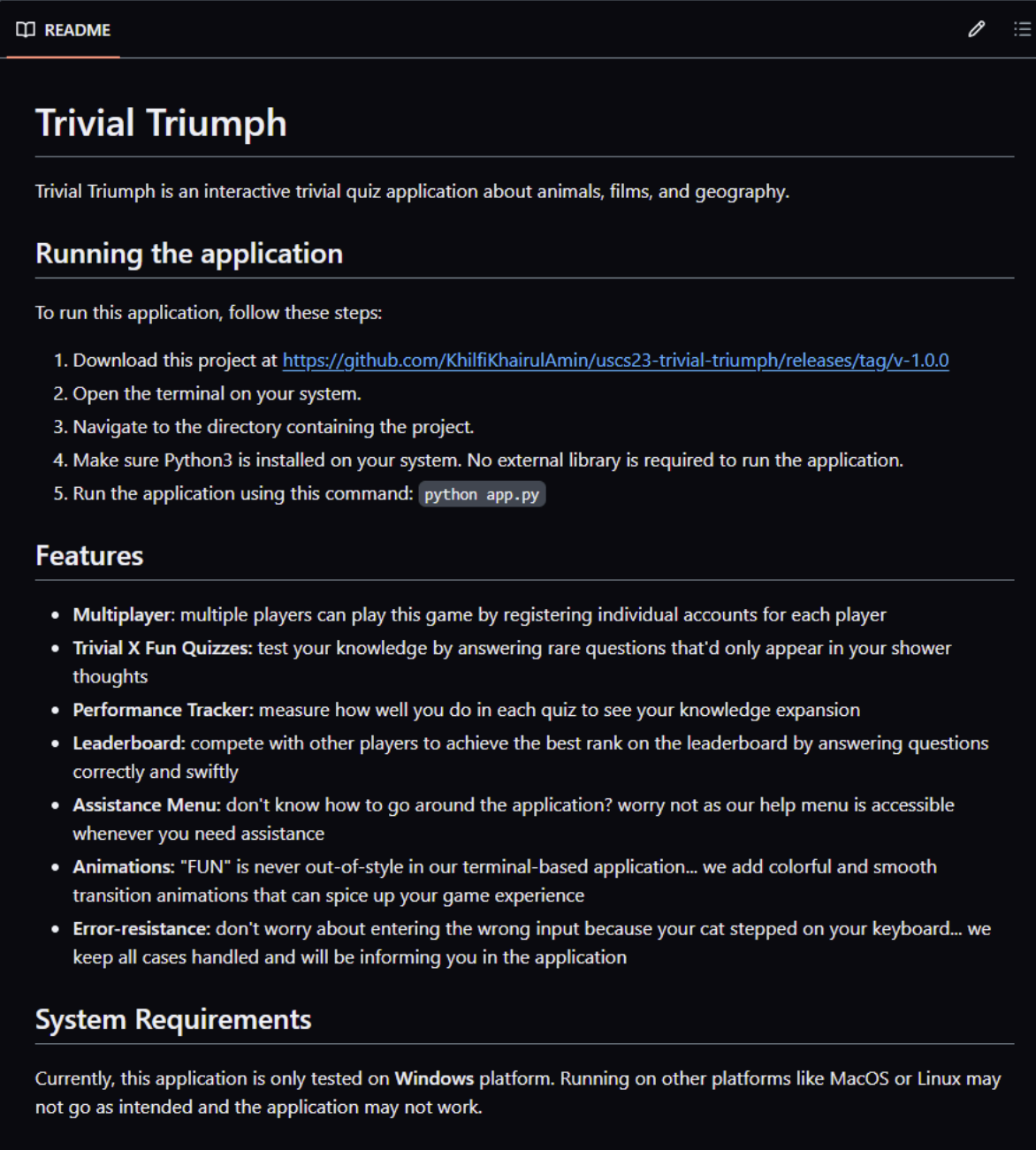
Introduction

In today's fast-paced digital landscape, engaging individuals in meaningful learning experiences presents a significant challenge. Traditional methods often struggle to compete with modern distractions, leaving a gap in entertaining educational resources. This report explores the development of Trivial Triumph, a terminal-based trivial quiz Python application as a solution to this problem.

The proliferation of digital distractions to broad educational resources have led to a decline in knowledge retention and engagement. By offering a convenient and gamified learning platform accessible via the old-school terminal interfaces, Trivial Triumph quiz application seeks to bridge this gap. In this report, we'll explore the guides towards using the application, and the technical implementations behind the application.

README File

The README file of this project contains the necessary information about this application, how to run the application on the user's system, the features included inside this application, and the system requirements for this application to run properly on the user's machine.

A screenshot of a README file for a project named 'Trivial Triumph'. The interface has a dark theme. At the top, there's a header bar with 'README' on the left and edit/expand icons on the right. The main content starts with the title 'Trivial Triumph' in a large, bold font. Below it, a paragraph describes the application as an interactive trivial quiz. The next section is 'Running the application', which lists five steps to get the project and run it. This is followed by a 'Features' section with a bulleted list of game features like multiplayer, quizzes, performance tracking, and a leaderboard. The final section is 'System Requirements', which notes that the application is currently only tested on Windows. The entire screenshot is set against a dark background with light-colored text.

README

Trivial Triumph

Trivial Triumph is an interactive trivial quiz application about animals, films, and geography.

Running the application

To run this application, follow these steps:

1. Download this project at <https://github.com/KhilfiKhairulAmin/uscs23-trivial-triumph/releases/tag/v-1.0.0>
2. Open the terminal on your system.
3. Navigate to the directory containing the project.
4. Make sure Python3 is installed on your system. No external library is required to run the application.
5. Run the application using this command: `python app.py`

Features

- **Multiplayer:** multiple players can play this game by registering individual accounts for each player
- **Trivial X Fun Quizzes:** test your knowledge by answering rare questions that'd only appear in your shower thoughts
- **Performance Tracker:** measure how well you do in each quiz to see your knowledge expansion
- **Leaderboard:** compete with other players to achieve the best rank on the leaderboard by answering questions correctly and swiftly
- **Assistance Menu:** don't know how to go around the application? worry not as our help menu is accessible whenever you need assistance
- **Animations:** "FUN" is never out-of-style in our terminal-based application... we add colorful and smooth transition animations that can spice up your game experience
- **Error-resistance:** don't worry about entering the wrong input because your cat stepped on your keyboard... we keep all cases handled and will be informing you in the application

System Requirements

Currently, this application is only tested on **Windows** platform. Running on other platforms like MacOS or Linux may not go as intended and the application may not work.

The full project can be downloaded at our [GitHub repository of Trivial Triumph](#).

Selected Codes

This section will cover important parts of the application source code. Each piece of significant code sections will be explained thoroughly to explain the flow and magic behind this application. We'll only examine some parts of the application codes, thus, the trivial code sections aren't explained. You can examine the whole codebase at our [GitHub repository of Trivial Triumph](#).

1. quiz.py

This Python script, named quiz.py, manages quiz logics and scoring.

Imports

The script imports various functions from external modules db and ui that contain functions for loading questions from a database and handling user interface interactions.

```
"""
Program: quiz.py
Author: Trivial Triumph Devs
Provide quiz algorithms for handling quiz logics and scoring
"""

from db import load_mcq_questions, load_tf_questions,
load_matching_questions, load_FIB_questions, load_sub_questions
from ui import center, error, fill, prompt, prompt_choice, success

import random
```

Function Definition

1. quizEasy:

This function is an easy mode quiz. It loads 3 types of questions (MCQ, True/False, Matching) and calls respective functions to conduct the quiz. If the total score of the user more than 9, user will have the chance to play the hard mode, which includes Fill in the Blanks and Subjective questions.

```
def quizEasy():

    questionsMCQ = load_mcq_questions()
```

```

questionsTF      = load_tf_questions()
questionsMatch   = load_matching_questions()
questionsFIB     = load_FIB_questions()
questionsSub     = load_sub_questions()

score = 0
score += quizEasy_MCQ(questionsMCQ, number_of_questions=3)
score += quizEasy_TF(questionsTF, number_of_questions=3)
score += quizEasy_Match(questionsMatch, number_of_questions=3)

fill("*")
center()

if score > 9:
    center("HARD MODE   (5marks)", col="\033[91m")
    score += quizHard_FIB(questionsFIB, number_of_questions=3)
    score += quizHard_Sub(questionsSub, number_of_questions=3)
else:
    center("EASY MODE   (2marks)", col="\033[36m")
    score += quizEasy_MCQ(questionsMCQ, number_of_questions=2)
    score += quizEasy_TF(questionsTF, number_of_questions=2)
    score += quizEasy_Match(questionsMatch, number_of_questions=2)

return score

```

2. quizEasy_MCQ, quizEasy_TF, quizEasy_Match, quizHard_FIB, quizHard_Sub:

These functions handle specific types of questions (Multiple Choice, True/False, Matching, Fill in the Blanks, Subjective). They present questions to the user, accept their answers, and calculate the score.

```

def quizEasy_MCQ(questionsMCQ: list, number_of_questions=3):
    center("MULTIPLE CHOICE QUESTIONS", end="\n\n\n", col="\033[33m")

    score = 0
    for count in range(1, number_of_questions+1):
        questionsNo = random.randint(0, len(questionsMCQ)-1)

        question, options, answer = questionsMCQ[questionsNo]
        center(f"{count}. {question}")
        for option in options:

```

```

        center(option)
    center()

    while True:
        try:
            userAnswer = prompt_choice(prompt_message="Enter your
answer <A, B, C, D>: ", choices=["A", "a", "B", "b", "C", "c", "D",
"d"], input_width=2).lower()
            center()
            break
        except ValueError as err:
            error(err)

    if userAnswer == answer:
        score += 2
        success("Correct!\n")
    else:
        error("Incorrect.\n")

    # Remove picked questions so it doesn't repeat
    questionsMCQ.pop(questionsNo)

return score

def quizEasy_TF(questionsTF: list, number_of_questions=3):
    center("TRUE OR FALSE QUESTIONS", end="\n\n\n", col="\033[33m")

    score = 0
    for count in range(1, number_of_questions+1):
        questionsNo = random.randint(0, len(questionsTF)-1)

        question, answer = questionsTF[questionsNo]
        center(f"{count}. {question}\n")

        userAnswer = prompt_choice(prompt_message="Enter your answer
<True/False>: ", choices=["True", "true", "False", "false"],
input_width=6).lower()
        center()

        if userAnswer == answer:
            score += 2
            success("Correct!\n")

```

```

        else:
            error("Incorrect.\n")

        # Remove picked questions so it doesn't repeat
        questionsTF.pop(questionsNo)

    return score

def quizEasy_Match(questionsMatch: list, number_of_questions=3):
    center("MATCHING QUESTIONS", end="\n\n\n", col="\033[33m")

    score = 0
    for count in range(1, number_of_questions+1):
        center(f"{count}. Match the statements <A,B,C> correctly to  
their answers <1,2,3>.\n")

        questionsNo = random.randint(0, len(questionsMatch)-1)

        match = questionsMatch[questionsNo]
        correct_answers = [1, 2, 3]
        random.shuffle(correct_answers)

        question_map = {
            correct_answers[0]: match[0][1],
            correct_answers[1]: match[1][1],
            correct_answers[2]: match[2][1],
        }

        # Display matching boxes
        for i in range(3):
            center(f"%80s ({chr(65+i)})\t({i+1}) %-80s\n" %
(match[i][0], question_map[i+1]))
            center()

        correct = 0
        for j in range(3):

            while True:
                try:
                    answer = prompt_choice(f"({chr(65+j)}) -> ",
choices=[1, 2, 3])
                    center()

```



```

        break
    except ValueError as err:
        error(err)

    if answer == str(correct_answers[j]):
        success("Correct!\n")
        correct += 1
    else:
        error("Incorrect!\n")

    if correct == 3:
        score += 2
    elif correct == 2:
        score += 1

    # Remove picked question so it doesn't repeat
    questionsMatch.pop(questionsNo)

return score

def quizHard_FIB(questionsFIB, number_of_questions=3):
    center("FILL IN THE BLANKS QUESTIONS", end="\n\n\n",
col="\033[33m")

    score = 0
    for count in range (1, number_of_questions+1):
        questionsNo = random.randint(0, len(questionsFIB)-1)

        question, answer = questionsFIB[questionsNo]
        center(f"{count}. {question}\n")
        userAnswer = prompt("Answer: ", 10).lower()
        center()
        if userAnswer == answer:
            score += 5
            success("Correct!!\n")
        else:
            error("Incorrect.\n")

        questionsFIB.pop(questionsNo)

    return score

```

```
def quizHard_Sub(questionsSub, number_of_questions=3):
    center("SUBJECTIVE QUESTIONS", end="\n\n\n", col="\033[33m")

    score = 0
    for count in range (1, number_of_questions+1):
        questionsNo = random.randint(0, len(questionsSub)-1)

        question, answer = questionsSub[questionsNo]
        center(f"{count}. {question}\n")
        userAnswer = prompt("Answer: ", 10).lower()
        center()
        if userAnswer == answer:
            score += 5
            success("Correct!!\n")
        else:
            error("Incorrect.\n")

        questionsSub.pop(questionsNo)

    return score
```

2. menus.py

This file named menus.py provides the menus for the quiz application. It integrates the validation process, file operations, quiz algorithms, and user interfaces to make smoothly working menus.

Imports

These modules are imported inside this file. The `time` module is a built-in module in Python. The other modules (`auth`, `db`, `quiz`, `ui`) are modules we made for the applications. No external online modules are used by the application.

```
import time
from auth import sign_up, log_in
from db import save_users_data, get_users_data
from quiz import quizEasy
from ui import clear, countdown, display_header, center, fill,
good_game, prompt, prompt_choice, display_header_cinematic
```

Function Definitions

1. `sign_up_menu`, `log_in_menu`:

These menus serve as interfaces and validators to authorize users based on username and password input. A set of rules are applied to input validation to make sure the application can display and store them. If the user is authorized successfully, a new account will be created and stored (for `sign_up_menu` only) and then the current username will be stored inside a global variable as a passkey of the user for the current session.

```
def sign_up_menu():
    """
    Sign up menu
    """
    # Display header of login menu
    display_header()
    center("Sign Up\n")

    global Users, CurUser
    while True:
        username = prompt("Username: ")
        password = prompt("Password: ", hidden=True)
```

```

        repeat_password = prompt("Repeat password: ",
hidden=True, input_width=24)

        CurUser = sign_up(username, password, repeat_password,
Users)

        if not CurUser: continue

        # Save the updated data
        save_users_data(Users)
        return HOME_MENU # Go to home menu

def login_menu():
    """
    Log in menu
    """
    # Display header of login menu
    display_header()
    center("Log In\n")

    while True:
        username = prompt("Username: ")
        password = prompt("Password: ", hidden=True)

        global CurUser, Users
        CurUser = log_in(username, password, Users)

        if not CurUser: continue

    return HOME_MENU # Go to home menu

```

Output



2. quiz_menu:

This menu is the heart of the application. It handles quiz execution and manages scores and other data while the quiz is being played by the user. It also incorporates colorful and smooth animations to improve the gaming experience of the player. Lastly, it calculates, displays, and stores the overall score, time taken, highest personal score, and leaderboard rank.

```
def quiz_menu():
    """
    Handles Quiz display and flow
    """
    clear()
    center("Trivial Triumph", col="\033[33m")
    fill("*")
    center()
    center("You will answer 15 fun trivial questions.")
    center("If you answer some of the first 9 questions
correctly, you stand a chance on doubling marks in HARD
MODE.")
    center("There are MCQ, True/False, Matching, Fill The
Blanks, and Subjective questions.", end="\n\n")
    prompt("Press Enter when you're ready!", input_width=0,
hidden=True)
    countdown()
    display_header(subtitle="Happy Answering!")

    start = time.time() # Record starting time
    score = quizEasy() # Run the quiz
    end = time.time() # Record finishing time

    # Result processing
    time_taken = int(end - start)
    if Users[CurUser][1] == -1:
        is_new_high_score = True
    else:
        is_new_high_score = score > max([x[0] for x in
Users[CurUser][1:]])

    # Store new score
```

```

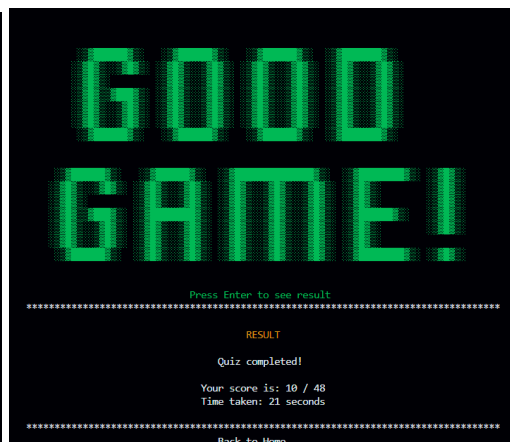
if Users[CurUser][1] != -1:
    Users[CurUser].append((score, time_taken))
else:
    Users[CurUser][1] = (score, time_taken)
save_users_data(Users)

fill("*")
center("END OF QUIZ", col="\033[33m")
time.sleep(1)
good_game()

prompt("Press Enter to see result\n", hidden=True,
input_width=0)
fill("*")
center()
center("RESULT", col="\033[33m", end="\n\n")
center(f"Quiz completed!", end="\n\n")
if is_new_high_score: center(f"NEW PERSONAL HIGH SCORE!",
col="\033[32m")
center(f"Your score is: {score} / 48")
center(f"Time taken: {time_taken} seconds", end="\n\n")
fill("*")
prompt("Back to Home...\n", hidden=True, input_width=0)
return HOME_MENU # Go to home menu

```

Output



3. leaderboard_menu, player_statistics_menu:

These menus are what make the quiz more purposeful. Leaderboard menus display the current ranking of the best players; it makes the quiz more

competitive among players. The statistics menu allows the player to keep track of their performances and stores all the quizzes that they have played. It can make sure players improve overtime through this application, and their knowledge can be expanded effectively.

```
def leaderboard_menu():
    """
    Leaderboard menu
    """
    users = get_users_data()
    display_header(subtitle="Top 10 Players' Highest Marks")

    # Filter from player that hasn't played any matches
    remove_list = []
    for k in users.keys():
        if users[k][1] == -1:
            remove_list.append(k)

    for k in remove_list:
        users.pop(k)

    # Sort users by their highest score
    sorted_users = sorted(users.items(), key=lambda x:
(max([score[0] for score in x[1][1:]]), min([-time[1] for
time in x[1][1:]])), reverse=True)

    center("Leaderboard", col="\033[33m", end="\n\n")
    center("Rank      Name                      Highest Score      Time
Taken")

    for rank, (username, scores) in
enumerate(sorted_users[:10], 1):
        sorted_scores = sorted(scores[1:], key=lambda x: (x[0],
-x[1]), reverse=True)
        highest_score, lowest_time = sorted_scores[0]
        center("%4d    %-12s    %-2d / 48          %-3ds    "
% (rank, username, highest_score, lowest_time))

    center(end="\n\n\n")
    fill("*")
```

```

    prompt("Press Enter to return to Home...\n", hidden=True,
input_width=0)
    return HOME_MENU # Go to home menu

def player_statistics_menu():
    """
    Generate and display statistics about the user performance
    based on matches, scores, and time.
    """
    # Display header
    clear()
    center("Player Stats", col="\033[33m")
    center()
    fill("*")
    center()

    matches_played = 0 if Users[CurUser][1] == -1 else
len(Users[CurUser][1:])

    center(f"Username: {CurUser}", col="\033[33m")
    center(f"Matches Played: {matches_played}", end="\n\n")

    if matches_played == 0:
        center("No data available. Play at least one match to see
statistics.")
    else:
        center("+-----+-----+-----+-----+")
        center("| Match | Score | Time taken | Speed |")
        center("+-----+-----+-----+-----+")

        count = 0
        total_score = 0
        total_time = 0
        highest_score = 0

        for score, time in Users[CurUser][1:]:
            center("%7d|%4d/48|%10d s| %5.2f |" % (count+1, score,
time, time / 15))
            center("+-----+-----+-----+-----+")
            count += 1

```



```

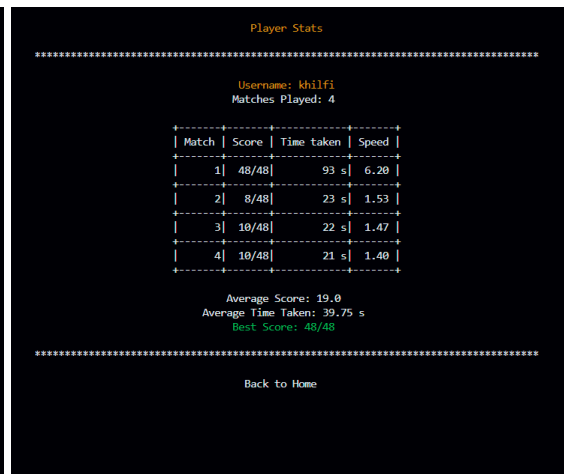
        total_score += score
        total_time += time
        if score > highest_score:
            highest_score = score

    center()
    center(f"Average Score: {round(total_score / count, 2)}")
    center(f"Average Time Taken: {round(total_time / count,
2)} s")
    center(f"Best Score: {highest_score}/48", col="\033[32m")

    center()
    fill("*")
    center()
    prompt("Back to Home\n", input_width=1, hidden=True)
    return HOME_MENU

```

Output



4. help_menu, exit_menu, exit_modal:

These menus serve as assistance that makes the user experience top-notch. The help menu displays the associated controls in this application to help them navigate and use the application smoothly. The exit menu and modal serve as a way for the user to quit the application. Exit modal will ask for confirmation before the user quits to avoid accidental quits by the user. We also exploit the `KeyboardInterrupt` error (when the user presses Ctrl+C) as a feature in our application such that when the user presses Ctrl+C keys, they can go back to the previous menu or quit the application instead of encountering the error.

```

def help_menu():
    """
    Show helpful controls to navigate through the app
    """
    display_header("")
    center("Controls", end="\n\n")
    center("Navigate      -   [1, 2, 3, 4, 5]   ")
    center("Answer        -   [A..Z, a..z, 0..9]")
    center("Proceed          -   [Enter]           ")
    center("Back/Previous     -   [Ctrl + C]          ")
    center()
    prompt("Back to Home...\n", hidden=True, input_width=1)
    return HOME_MENU # Go to home menu


def exit_menu():
    """
    Clear the whole application texts when exit application
    """
    clear()


def exit_modal(message="Are you sure you want to quit?"):
    """
    Prompt confirmation from user whether to exit the
    application or not.
    """
    try:
        clear()
        center(message, end="\n\n")
        center("[1] No      [2] Yes")
        center()

        choice = int(prompt_choice(">", choices=[1, 2]))
        if choice == 1:
            return False
        elif choice == 2:
            return True
    except KeyboardInterrupt:
        return False

```

Output



3. db.py

This Python program, named db.py, serves as an interface for database operations.

Constant

1. **CWD:** Stores the current working directory path.
2. **USER_DATA_FILE:** Specifies the path to the file where user data is stored (users.txt).
3. **QUESTIONS_FOLDER:** Specifies the path to the folder containing all quiz questions.

```
from os import getcwd

# Store current working directory for file reference
CWD = getcwd()

# Path to file where the users data are stored
USER_DATA_FILE = "users.txt"

# Path to folder containing all questions
QUESTIONS_FOLDER = f"{CWD}\questions"
```

Functions

1. **get_users_data():** Reads user data from the users.txt file. If the file is not found, it creates a new one with a header and returns an empty dictionary. It then parses the data into a dictionary format and returns it.
2. **save_users_data(users: dict):** Writes user data from a dictionary back to the users.txt file. It formats the data and writes it line by line.

```
def get_users_data():
    """
    Read users data from users.txt
    """
    try:
        # Open the database file
        f = open(USER_DATA_FILE, "r")

    except FileNotFoundError:

        # When file is not found, create new file
```

```

new_f = open(USER_DATA_FILE, "w")
new_f.write("username,password,scores-time\n")
new_f.close()
f = open(USER_DATA_FILE, "r")

next(f) # Skip the 1st line, which is the data header (see users.txt
file for reference)

users = {}
# Parse data into dictionary
for line in f:
    raw = line.strip()
    username, password, *scores_time = raw.split(",")
    if scores_time[0] == "-1":
        users[username] = [password, -1]
    else:
        scores_time = [s.split("-") for s in scores_time]
        scores_time = [tuple([int(s[0]),int(s[1])]) for s in
scores_time]
        users[username] = [password] + scores_time
return users

def save_users_data(users: dict):
    """
    Write users data into users.txt file
    """
    f = open(USER_DATA_FILE, "w")

    raw = "username,password,scores-time\n"

    # Parse dictionary data into string
    for username, data in users.items():
        if data[1] == -1:
            raw += f"{username},{data[0]},-1\n"
        else:
            raw += f"{username},{data[0]},{'',''.join([f'{d[0]}-{d[1]}' for d
in data[1:]])}\n"

    f.write(raw)

```

3. Functions to load different types of quiz questions:

- a. **load_mcq_questions():** Reads Multiple Choice Questions (MCQ) from the mcq.txt file, parses them, and returns a list of tuples containing the questions, options, and correct answers.
- b. **load_tf_questions():** Reads True/False questions from the tf.txt file, parses them, and returns a list of tuples containing the questions and correct answers.
- c. **load_matching_questions():** Reads Matching questions from the matching.txt file, parses them, and returns a list of lists. Each inner list contains tuples representing question-answer pairs.
- d. **load_FIB_questions():** Reads Fill-in-the-Blanks questions from the FIB.txt file, parses them, and returns a list of tuples containing the questions and correct answers.
- e. **load_sub_questions():** Reads Subjective questions from the sub.txt file, parses them, and returns a list of tuples containing the questions and correct answers.

```
def load_mcq_questions():
    """
    Read MCQ questions and answers in mcq.txt
    """
    f = open(f"{QUESTIONS_FOLDER}\\mcq.txt")

    # Skip the first 6 lines which are the sample questions of the file
    for _ in range(6):
        next(f) # Skip the line

    mcq = []
    for _ in f:
        question = next(f).strip()
        answers = []
        answers.append(f"A. {next(f).strip()}")
        answers.append(f"B. {next(f).strip()}")
        answers.append(f"C. {next(f).strip()}")
        answers.append(f"D. {next(f).strip()}")
        correct_answer = next(f).strip().lower()
        mcq.append((question, answers, correct_answer))

    return mcq

def load_tf_questions():
```

```

"""
Read True/False questions and answers in tf.txt
"""

f = open(f"{QUESTIONS_FOLDER}\\tf.txt")

# Skip the first 2 lines which are the sample questions of the file
for _ in range(2):
    next(f)  # Skip the line

tf = []
for _ in f:
    question = next(f).strip()
    correct_answer = next(f).strip().lower()
    tf.append((question, correct_answer))

return tf

def load_matching_questions():
    """
    Read matching questions in matching.txt
    """

    f = open(f"{QUESTIONS_FOLDER}\\matching.txt")
    for _ in range(3):
        next(f)

    matchings = []
    for _ in f:  # Read three questions and answers
        m = []
        for _ in range(3):
            question, correct_answer = next(f).strip().split(" -> ")
            m.append((question, correct_answer))
        matchings.append(m)

    return matchings

def load_FIB_questions():
    """
    Read fill in the blank questions in FIB.txt
    """

    f = open(f"{QUESTIONS_FOLDER}\\FIB.txt")

```

```

        for _ in range (2):
            next(f)

    FIB = []
    for _ in f:
        question = next(f).strip()
        correct_answer = next(f).strip().lower()
        FIB.append((question, correct_answer))

    return FIB

def load_sub_questions():
    """
    Read subjective questions in sub.txt
    """
    f = open(f"{QUESTIONS_FOLDER}\sub.txt")

    for _ in range (2):
        next(f)

    sub = []
    for _ in f:
        question = next(f).strip()
        correct_answer = next(f).strip().lower()
        sub.append((question, correct_answer))

    return sub

```

Main Block

A main block is included at the end of the file for testing purposes. It saves sample user data, retrieves it, and prints it along with sample quiz questions of different types.

```

if __name__ == "__main__":
    save_users_data({'yasmin': ['yasmin', (67,100), (100,60)], 'khilfi':
['khilfi', -1], 'irfan': ['izerith', -1]})
    print(get_users_data(), end="\n\n\n")
    print(load_mcq_questions(), end="\n\n\n")
    print(load_tf_questions(), end="\n\n\n")
    print(load_matching_questions(), end="\n\n\n")
    print(load_FIB_questions(), end="\n\n\n")
    print(load_sub_questions(), end="\n\n\n")

```


4. app.py

This code represents the main control flow of the quiz application.

Import

The code imports various menu-related functions and constants from the `menus` module.

```
from menus import STATISTIC, exit_menu, help_menu, main_menu,
player_statistics_menu, sign_up_menu, login_menu, home_menu,
quiz_menu, leaderboard_menu, \
                                MAIN_MENU, SIGN_UP, LOGIN, HOME_MENU,
QUIZ, LEADERBOARD, HELP, EXIT
```

Function statement

1. main()

This function initializes variables `parent_menu` and `cur_menu` to `EXIT` and `MAIN_MENU`. The program enters a loop where it continuously checks the current menu (`cur_menu`) and executes the corresponding menu function. Depending on the current menu, it sets the `cur_menu` variable to the next menu to be displayed. If the user presses Ctrl+C, indicating an interruption, the program catches the `KeyboardInterrupt` exception and resets the `cur_menu` to the `parent_menu`.

```
def main():
    """
    Starting point of the application. This function manages the
    state of the application.
    """
    parent_menu = EXIT
    cur_menu = MAIN_MENU

    while True:
        try:
            if cur_menu == MAIN_MENU:
                cur_menu = main_menu()    # Parent menu 1

            elif cur_menu == SIGN_UP:
                parent_menu = MAIN_MENU
```

```
    cur_menu = sign_up_menu()

elif cur_menu == LOGIN:
    parent_menu = MAIN_MENU
    cur_menu = login_menu()

elif cur_menu == HOME_MENU:
    cur_menu = home_menu()    # Parent menu 2

elif cur_menu == QUIZ:
    parent_menu = HOME_MENU
    cur_menu = quiz_menu()

elif cur_menu == LEADERBOARD:
    parent_menu = HOME_MENU
    cur_menu = leaderboard_menu()

elif cur_menu == STATISTIC:
    parent_menu = HOME_MENU
    cur_menu = player_statistics_menu()

elif cur_menu == HELP:
    parent_menu = HOME_MENU
    cur_menu = help_menu()

elif cur_menu == EXIT:
    exit_menu()
    break

# If user press Ctrl+C, go back to parent menu
except KeyboardInterrupt:
    cur_menu = parent_menu
```

5. auth.py

This Python script, auth.py, provides functionality for user authentication.

Constants

1. **MIN_CHAR_USERNAME** and **MAX_CHAR_USERNAME**: Define the minimum and maximum lengths allowed for a username.
2. **MIN_CHAR_PASSWORD** and **MAX_CHAR_PASSWORD**: Define the minimum and maximum lengths allowed for a password.

```
MIN_CHAR_USERNAME = 3
MAX_CHAR_USERNAME = 12

MIN_CHAR_PASSWORD = 3
MAX_CHAR_PASSWORD = 20
```

Functions

1. **sign_up(new_username, new_password, repeat_new_password, users):**
Registers a new user by verifying the provided username and password. It checks if the username meets length and uniqueness criteria, if the password matches the repeated password, and then adds the new user to the users dictionary.

```
def sign_up(new_username: str, new_password: str, repeat_new_password: str, users: dict) -> str:
    """
    Register a new user. Returns username if authorized, else return empty string. This function automatically assigns the new user data into `users` variable if authorization successful.
    """

    # Make sure username is at least 3 characters long and maximum 12 characters long
    if len(new_username) < MIN_CHAR_USERNAME or len(new_username) > MAX_CHAR_USERNAME:
        error(f"Username must contain at least {MIN_CHAR_USERNAME} characters and maximum of {MAX_CHAR_USERNAME} characters")
        return ''

    # Make sure username is unique
    if new_username in users.keys():
```

```

    error("Username already exists")
    return ''

# Allow underscores
temp = new_username.replace("_", "")

# Make sure username only contains alphanumeric characters without
space
if not temp.isalnum():
    error("Username must consists of only alphanumeric characters")
    return ''

# Make sure password is at least 3 characters long and maximum 20
characters long
if len(new_password) < MIN_CHAR_PASSWORD or len(new_password) >
MAX_CHAR_PASSWORD:
    error(f"Password must contain at least {MIN_CHAR_PASSWORD}
characters and maximum of {MAX_CHAR_PASSWORD} characters")
    return ''

if repeat_new_password != new_password:
    error("Password does not match the repeated password")
    return ''

# Add the new user
users[new_username] = [new_password, -1]

return new_username

```

2. **log_in(username, password, users):** Authenticates an existing user by verifying the provided username and password against the stored user data.

```

def log_in(username: str, password: str, users: dict) -> str:
    """
    Authenticate existing user. Returns username if authorized, else
    return empty string.
    """
    # Check username existence
    if username not in users.keys():
        error("Username does not exist")
        return ''

```

```
# Verify password
if users[username][0] != password:
    error("Username or password is false")
    return ''

return username
```

6. ui.py

This Python script, ui.py, provides custom functions to enhance the display of text in the terminal for the "Trivial Triumph" quiz application. These constants are used to visually display the title in the terminal. It enhances the user interface of the quiz application by providing visually appealing text display, user input handling, and animation effects.

[illegible]

[illegible]

```

    sleep(0.1)

def fill(char):
    """
    Fill the line with the character
    """
    center(char * ASCII_ART_WIDTH)

def prompt(prompt_message="", input_width=18, hidden=False) -> str:
    """
    Prompt user input at the center of the terminal
    """
    if hidden: return getpass.getpass(prompt_message.center(WIDTH -
input_width).rstrip()+" ") # Password input
    else: return input(prompt_message.center(WIDTH -
input_width).rstrip()+" ") # Normal input

def prompt_choice(prompt_message="", choices = [0, 1], input_width=2)
-> str:
    """
    Validate choice input
    """

    while True:
        choice = prompt(prompt_message, input_width)

        if choice not in [str(choice) for choice in choices]:
            error("Invalid choice")
            continue

        return choice

def display_header(subtitle="Welcome to Trivial Triumph!"):
    """
    Display Trivial Triumph remarkable header
    """
    clear()
    print(TRIVIAL_ASCII_ART, end="")
    sleep(0.1)

```



```

print(TRIUMPH_ASCII_ART)
sleep(0.1)
center(f"{subtitle}\n")
fill("*")
print()

def display_header_cinematic(subtitle="Welcome to Trivial Triumph!"):
    """
    Display Trivial Triumph remarkable header cinematically
    """
    clear()
    sleep(1)
    print(TRIVIAL_ASCII_ART, end="")
    sleep(1.5)
    print(TRIUMPH_ASCII_ART)
    sleep(1.5)
    center(f"{subtitle}\n")
    sleep(1.7)
    fill("*")
    print()

def error(text=""):
    """
    Print error message in red colour
    """
    center(f"{text}", end="", col="\033[91m")
    print("\033[00m\n")

def success(text=""):
    """
    Print success message in green colour
    """
    center(f"{text}", end="", col="\033[32m")
    print("\033[00m\n")

def countdown():
    global WIDTH
    LEFT_PADDING = " " * ((WIDTH - 38) // 2)

```

```

clear()
print("\033[33m")
SKIP_ROW = "\n" * 5
print(f"{SKIP_ROW}
{LEFT_PADDING}      .--,-`-.-.
{LEFT_PADDING}      /  /      '.
{LEFT_PADDING}      /  ./      ;
{LEFT_PADDING}      \  ``\  .`-  '
{LEFT_PADDING}      \___\ /  \  :
{LEFT_PADDING}           \  :  |
{LEFT_PADDING}          /  /  /
{LEFT_PADDING}          \  \  \
{LEFT_PADDING}         ____ /  :  |
{LEFT_PADDING}        /    /\  /  :
{LEFT_PADDING}       /  , /  ',-  .
{LEFT_PADDING}      \  ' '\      ;
{LEFT_PADDING}      \    \      .'
{LEFT_PADDING}      `--`-,-,-'
""")

```

```

sleep(1)
clear()
print(f"{SKIP_ROW}
{LEFT_PADDING}          ,-----,
{LEFT_PADDING}          . '      . ' \
{LEFT_PADDING}          ,-----, '    |
{LEFT_PADDING}          |      :    .  ;
{LEFT_PADDING}          ;      | . '  /
{LEFT_PADDING}          `-----' /  ;
{LEFT_PADDING}          /  ;  /
{LEFT_PADDING}          ;  /  /-,
{LEFT_PADDING}          /  /  /.`|
{LEFT_PADDING}        ./___;      :
{LEFT_PADDING}        |      :    . '
{LEFT_PADDING}        ;      | . '
{LEFT_PADDING}        `---'
""")

```

```

sleep(1.2)
clear()
print(f"{SKIP_ROW}
{LEFT_PADDING}          ,---,
{LEFT_PADDING}          ,`--.' |
{LEFT_PADDING}          /      /  :
{LEFT_PADDING}          :      | . ' '

```

```

{LEFT_PADDING}      \----': |
{LEFT_PADDING}      ' ' ;
{LEFT_PADDING}      | | |
{LEFT_PADDING}      ' : ;
{LEFT_PADDING}      | | '
{LEFT_PADDING}      ' : |
{LEFT_PADDING}      ; |.'
{LEFT_PADDING}      '---'

    """)
    sleep(1.2)
    clear()
    print(f""{SKIP_ROW}
{LEFT_PADDING}                                     ,---,
{LEFT_PADDING}                                     ,-----.. ,`--.' |
{LEFT_PADDING}      ,-----..      / / \ | : :
{LEFT_PADDING}/ / \ / \ / . : ' ' ;
{LEFT_PADDING}| : : . / ;. \ | | |
{LEFT_PADDING}. | ;. / . ; / ` ; ' : ;
{LEFT_PADDING}. ; /--` ; | ; \ ; | | | '
{LEFT_PADDING}; | ; _ | : | ; | ' ' : |
{LEFT_PADDING}| : |.' .' . | ' ' ' : ; | ;
{LEFT_PADDING}. | '_. ' : ' ; \ ; / | `---'. |
{LEFT_PADDING}' ; : \ | \ \ ' , / `---..` ;
{LEFT_PADDING}' | '/ .' ; : / .--,_
{LEFT_PADDING}| : / \ \ \ .' | | `
{LEFT_PADDING}\ \ \ .' `---` `---` , ;
{LEFT_PADDING} `---` `---`

    """)
    sleep(1)
    clear()
    print("\033[00m")

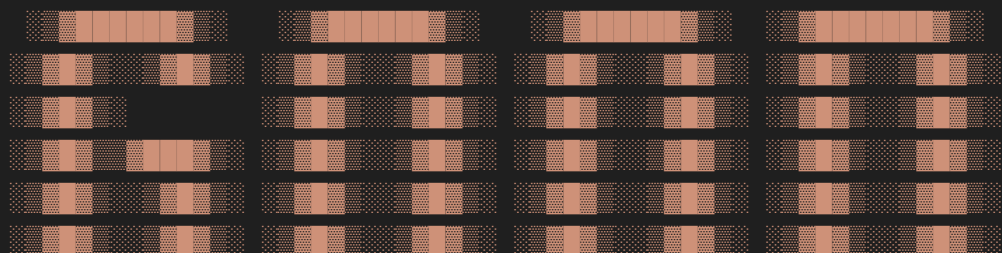
```

```
def good_game():
```

```
    clear()
```

```
    SKIP = "\n" * 5
```

```
    GG = f""{SKIP}
```



Conclusion

In conclusion, the Python program Trivial Triumph offers a robust and user-friendly platform for creating and conducting quizzes. The program's design allows for easy customization and scalability, making it suitable for various educational and recreational purposes. Trivial Triumph stands as a versatile and effective tool for creating and administering quizzes in various contexts, from classroom assessments to online trivia games. Its flexibility, accessibility, and robust functionality make it a valuable asset for educators, quiz enthusiasts, and anyone seeking to engage and challenge their audience. With further development and refinement, Trivial Triumph has the potential to become a leading platform for interactive learning and entertainment.