

ULTIMATE QUIZ CHALLENGE

Introduction:

The Quiz Game application is meticulously crafted to offer an engaging and interactive experience for users aiming to test their knowledge across various subjects. This project implements a quiz game application utilizing Python libraries like tkinter for the user interface and random for question shuffling. Players can choose their difficulty level and answer multiple-choice questions within a time limit (except for hard questions) using the time library. The application calculates the final score and provides a review of answered questions, offering an interactive and engaging knowledge assessment tool.

Key Features:

User-Friendly Interface: The application greets users with a visually appealing interface, ensuring ease of navigation and interaction.

Dynamic Question Generation: Questions are dynamically generated based on the user-selected difficulty level, ensuring a tailored and challenging experience.

Real-time Scoring: Users receive instant feedback on their performance, with scores updated in real-time as questions are answered.

Time-bound Challenges: To enhance the thrill, certain difficulty levels impose time constraints, adding an element of urgency and excitement to the quiz.

Effortless Customization: Administrators enjoy seamless control over question management, including addition, removal, and replacement, through a dedicated admin interface.

Password-Protected Access: Administrators are granted secure access to exclusive functionalities via a password-protected login system, ensuring data integrity and confidentiality.

Technological Insights:

Object-Oriented Design: Leveraging the principles of object-oriented programming, the application embodies encapsulation, abstraction, inheritance, and polymorphism, facilitating code modularity and extensibility.

Resourceful Data Management: Questions and answers are efficiently managed using external files, enabling easy updates and scalability without altering the core application logic.

Responsive GUI Elements: Utilizing tkinter widgets, such as buttons, labels, and frames, ensures a responsive and aesthetically pleasing user interface across different platforms and screen sizes.

OOP Concepts Explained in Detail:

The provided code utilizes several Object-Oriented Programming (OOP) concepts to create a well-structured and interactive quiz game. Here's a breakdown of the key concepts:

1. Classes and Objects:

Core Concept: Classes serve as blueprints for creating objects. Objects are instances of a class that encapsulate data (attributes) and behavior (methods).

Example:

`tk.Tk()` creates an object (the main window) of the `Tk` class from `tkinter`.

`ttk.Frame()` creates multiple frame objects used to organize the layout.

`ttk.Label()` and `ttk.Button()` create label and button objects for displaying information and user interaction.

2. Inheritance:

Core Concept: Inheritance allows a new class (subclass) to inherit properties and functionalities from an existing class (superclass). This promotes code reuse and reduces redundancy.

Example:

The `ttk` module inherits functionalities from the `tk` module. For instance, `ttk.Label` inherits from `tk.Label`, providing additional styling options without rewriting core label behavior.

3. Method Calls:

Core Concept: Objects interact with each other by calling methods (functions defined within a class) of other objects. Methods can access and manipulate the data (attributes) of the object they belong to.

Example:

Clicking the "Start Quiz" button calls the `start_quiz()` method within the main window object. This method creates a new window and triggers further functionalities for the quiz.

4. Polymorphism (Basic Form):

Core Concept: Polymorphism allows objects of different classes to respond to the same method call in different ways. This promotes flexibility and code maintainability.

Example:

The `compliment()` function demonstrates a basic form of polymorphism. Based on the score received (`score`), it provides different congratulatory messages (GREAT!, NOT BAD!, etc.) for the user. The function adapts its behavior based on the input (`score`).

5. Modularity and Reusability:

Core Concept: Breaking down code into smaller, well-defined functions promotes code reusability and maintainability. Each function encapsulates a specific task, making the code easier to understand and modify.

Example:

The code utilizes separate functions like `get_questions()`, `update_timer()`, `display_score()`, and `compliment()` for specific functionalities. This allows for cleaner code organization and potential reuse in future projects.

Libraries used:

1. `tkinter`:

This is the core library for creating graphical user interfaces (GUIs) in Python. It provides basic building blocks like windows, buttons, labels, and frames. Imagine it as a toolbox with various visual elements to build your application.

2. `ttk` (from `tkinter`):

This extension to `tkinter` offers more advanced widgets with improved styling options. It allows you to customize the appearance of your GUI elements, making them look more polished and visually appealing. Think of it as a set of fancier tools within the same toolbox, letting you design your interface with a more modern look.

3. `time`:

This library provides functions for working with time. In the code, it's used for keeping track of the time remaining during a quiz question, creating a sense of urgency for the player. Imagine it as a stopwatch that helps you manage time during the quiz.

4. openpyxl (not used in the current version):

This library (commented out in the code) allows you to read and write data from Excel spreadsheets. It could potentially be used to store and manage quiz questions in an organized format, but it's not currently active in this code. Think of it as a potential filing cabinet to store your quiz questions (not used in this version).

5. random:

This library provides functions for generating random numbers. In the code, it's used to shuffle question options and select random questions from a set, ensuring a different experience for each player. Imagine it as a hat where you shuffle answer choices and questions, adding an element of surprise.

Algorithm

The provided code implements the following algorithm for the quiz game:

1. User Input and Difficulty Selection:

The program starts by creating a graphical user interface (GUI) using tkinter and ttk.

The user enters their name and roll number in designated entry fields.

The user selects their preferred difficulty level (easy, medium, or hard) using radio buttons.

Clicking the "Start Quiz" button triggers the start_quiz() function.

2. Question Selection and Display:

Based on the chosen difficulty, the start_quiz() function calls get_questions(). This function retrieves a set of questions from pre-defined dictionaries (easy_questions, medium_questions, hard_questions).

The function randomly shuffles the questions using functions from the random library.

A new window is created to display the quiz interface.

The ask_question() function is called repeatedly to present each question from the shuffled list.

3. Question Answering and Time Limit:

The ask_question() function displays the current question text and its answer options using labels and buttons (ttk.Label and ttk.Button).

Clicking an answer button triggers a function to check the user's selection.

The time library is used to track the time elapsed since the question was displayed.

If the answer is correct and the elapsed time is less than or equal to 10 seconds (except for hard questions which have a stricter 10-second limit), the score is incremented.

The ask_question() function then retrieves the next question or displays the final score if all questions have been answered.

4. Score Display and Review:

If all questions have been answered, the display_score() function is called. This function displays the user's name, roll number, final score, and a congratulatory message based on the achieved score.

It also shows a list of answered questions along with the chosen answers, allowing the user to review their performance.

Overall, the algorithm follows a clear structure:

User input and difficulty selection.

Question selection and display based on difficulty.

User interaction with questions, answer checking, and time management.

Score calculation, display, and review of answered questions.

This algorithm provides a basic framework for a self-paced quiz game with increasing difficulty levels and time pressure for challenging questions.

Conclusion:

This project demonstrates the creation of an interactive quiz game application using Python. The code leverages the tkinter library for building the user interface and functionalities like random for question shuffling and (if applicable) time for implementing a time limit on questions. The application offers users the opportunity to test their knowledge at different difficulty levels, providing an engaging and self-paced learning experience.

Python code:

```
import tkinter as tk

from tkinter import ttk

import time

import openpyxl

import random


def main():

    root = tk.Tk()

    root.title("Quiz Game")


    style = ttk.Style()

    style.configure('Background.TFrame', background='#f0f0f0') # Set background color for
frames

    style.configure('TButton', foreground='ffffff', background='#4CAF50') # Set button colors

    style.map('TButton', background=[('active', '#45a049')])


def start_quiz():

    name = name_entry.get()

    roll_number = roll_number_entry.get()

    difficulty = difficulty_var.get()

    start_quiz_window(name, roll_number, difficulty)


main_frame = ttk.Frame(root, style='Background.TFrame')

main_frame.pack(expand=True, fill='both')


welcome_label = ttk.Label(main_frame, text="Welcome to the Quiz Game!", font=('Arial',
24))

welcome_label.pack(pady=20)
```

```
name_label = ttk.Label(main_frame, text="Name:")
```

```
name_label.pack()
```

```
name_entry = ttk.Entry(main_frame)
```

```
name_entry.pack()
```

```
roll_number_label = ttk.Label(main_frame, text="Roll Number:")
```

```
roll_number_label.pack()
```

```
roll_number_entry = ttk.Entry(main_frame)
```

```
roll_number_entry.pack()
```

```
difficulty_label = ttk.Label(main_frame, text="Select Difficulty:")
```

```
difficulty_label.pack()
```

```
difficulty_var = tk.IntVar()
```

```
easy_button = ttk.Radiobutton(main_frame, text="Easy 😊", variable=difficulty_var,  
value=1)
```

```
easy_button.pack(anchor='w')
```

```
medium_button = ttk.Radiobutton(main_frame, text="Medium 😊", variable=difficulty_var,  
value=2)
```

```
medium_button.pack(anchor='w')
```

```
hard_button = ttk.Radiobutton(main_frame, text="Hard 😎", variable=difficulty_var,  
value=3)
```

```
hard_button.pack(anchor='w')
```

```
start_button = ttk.Button(main_frame, text="Start Quiz", command=start_quiz)
```

```
start_button.pack(pady=20)
```

```

def start_quiz_window(name, roll_number, difficulty):

    quiz_window = tk.Toplevel(root)
    quiz_window.title("Quiz Game")

    questions = get_questions(difficulty)

    score = {"value": 0}
    correct_answers = []

    def ask_question():
        nonlocal score
        if questions:
            question = questions.pop(0)
            correct_answers.append((question["question"],
question["options"][question["correct_answer"]]))
            question_label = tk.Label(quiz_window, text=question["question"], font=("Arial", 16),
background='#f0f0f0')
            question_label.pack(pady=10)

            shuffled_options = random.sample(question["options"], len(question["options"]))

            for i, option in enumerate(shuffled_options):
                answer_button = ttk.Button(quiz_window, text=f"{i+1}. {option}",
command=lambda i=i: handle_answer(i, question), style='Quiz.TButton')
                answer_button.pack(side="top", pady=5)

            start_time = time.time()

    def handle_answer(answer_index, question):

```



```

nonlocal score

elapsed_time = time.time() - start_time

if elapsed_time <= 10 or difficulty < 3: # Allow only 10 seconds for hard questions
    if shuffled_options[answer_index] ==
question["options"][question["correct_answer"]]:
        score["value"] += 1

        for widget in quiz_window.winfo_children():
            widget.destroy()

        ask_question()
else:
    for widget in quiz_window.winfo_children():
        widget.destroy()

    ask_question()

timer_label = tk.Label(quiz_window, text="", font=("Arial", 14),
background='#f0f0f0')

timer_label.pack(pady=10)

update_timer(timer_label, start_time)

if len(correct_answers) > 1:
    previous_button = ttk.Button(quiz_window, text="Previous",
command=previous_question, style='Quiz.TButton')

    previous_button.pack(side="top", pady=5)

else:
    display_score(name, roll_number, score["value"], correct_answers)

def previous_question():
    correct_answers.pop()

```

```

score["value"] -= 1

for widget in quiz_window.winfo_children():
    widget.destroy()
ask_question()

ask_question()

root.mainloop()

def get_questions(difficulty):
    easy_questions = [
        {"question": "What is the capital of France?",
         "options": ["London", "Berlin", "Paris", "Rome"],
         "correct_answer": 2},
        {"question": "In which year did the first iPhone launch?",
         "options": ["2004", "2007", "2010", "2013"],
         "correct_answer": 1},
        {"question": "What is the capital of Japan?",
         "options": ["Beijing", "Tokyo", "Seoul", "Bangkok"],
         "correct_answer": 1},
        {"question": "Who wrote 'Romeo and Juliet'",
         "options": ["William Shakespeare", "Jane Austen", "Charles Dickens", "Mark Twain"],
         "correct_answer": 0},
        {"question": "What is the largest mammal?",
         "options": ["Elephant", "Whale", "Giraffe", "Kangaroo"],
         "correct_answer": 1},
        {"question": "Which planet is known as the Red Planet?",
         "options": ["Venus", "Jupiter", "Mars", "Saturn"],

```

```
"correct_answer": 2},
{"question": "What is the chemical symbol for water?",
"options": ["Wa", "Wo", "H2O", "O2"],
"correct_answer": 2},
{"question": "Who discovered penicillin?",
"options": ["Alexander Fleming", "Marie Curie", "Louis Pasteur", "Isaac Newton"],
"correct_answer": 0},
{"question": "What is the currency of Germany?",
"options": ["Euro", "Dollar", "Pound", "Yen"],
"correct_answer": 0},
{"question": "What is the tallest mountain in the world?",
"options": ["Mount Kilimanjaro", "Mount Everest", "Mount Fuji", "Mount McKinley"],
"correct_answer": 1}
]
```

```
medium_questions = [
{"question": "What is the square root of 25?",
"options": ["2", "5", "10", "None of these"],
"correct_answer": 0},
{"question": "The largest country in the world by land area is:",
"options": ["Russia", "Canada", "China", "United States"],
"correct_answer": 0},
{"question": "Who painted the Mona Lisa?",
"options": ["Vincent van Gogh", "Leonardo da Vinci", "Pablo Picasso", "Michelangelo"],
"correct_answer": 1},
{"question": "Which gas do plants absorb during photosynthesis?",
"options": ["Oxygen", "Carbon Dioxide", "Nitrogen", "Hydrogen"],
"correct_answer": 1},
```

```
{ "question": "What is the currency of France?",
  "options": ["Euro", "Pound", "Dollar", "Yen"],
  "correct_answer": 0},
{ "question": "What is the capital of Australia?",
  "options": ["Melbourne", "Sydney", "Canberra", "Brisbane"],
  "correct_answer": 2},
{ "question": "Who invented the telephone?",
  "options": ["Thomas Edison", "Alexander Graham Bell", "Guglielmo Marconi", "Nikola Tesla"],
  "correct_answer": 1 },
{ "question": "What is the chemical symbol for gold?",
  "options": ["Au", "Ag", "Fe", "Hg"],
  "correct_answer": 0},
{ "question": "Which planet is known as the Blue Planet?",
  "options": ["Earth", "Neptune", "Uranus", "Pluto"],
  "correct_answer": 0},
{ "question": "Who wrote 'To Kill a Mockingbird'?",
  "options": ["Harper Lee", "Ernest Hemingway", "F. Scott Fitzgerald", "John Steinbeck"],
  "correct_answer": 0}
]
```

```
hard_questions = [
  { "question": "What is the first element in the periodic table?",
    "options": ["Hydrogen", "Helium", "Lithium", "Beryllium"],
    "correct_answer": 0},
  { "question": "What is the capital of Nepal?",
    "options": ["Kathmandu", "Delhi", "Islamabad", "Dhaka"],
    "correct_answer": 0},
  { "question": "Who is the author of 'The Great Gatsby'?",
```

```

"options": ["F. Scott Fitzgerald", "Ernest Hemingway", "John Steinbeck", "Harper Lee"],
"correct_answer": 0},
{"question": "Which is the largest ocean on Earth?",
"options": ["Atlantic Ocean", "Indian Ocean", "Arctic Ocean", "Pacific Ocean"],
"correct_answer": 3},
{"question": "Who discovered the theory of relativity?",
"options": ["Isaac Newton", "Albert Einstein", "Galileo Galilei", "Stephen Hawking"],
"correct_answer": 1},
{"question": "Which country is known as the Land of the Rising Sun?",
"options": ["China", "Japan", "South Korea", "Thailand"],
"correct_answer": 1},
{"question": "What is the chemical formula for table salt?",
"options": ["NaCl", "H2O", "C6H12O6", "CO2"],
"correct_answer": 0},
{"question": "Who painted the famous artwork 'The Starry Night'?",
"options": ["Vincent van Gogh", "Leonardo da Vinci", "Pablo Picasso", "Michelangelo"],
"correct_answer": 0},
{"question": "What is the largest desert in the world?",
"options": ["Sahara Desert", "Arabian Desert", "Gobi Desert", "Antarctica Desert"],
"correct_answer": 0},
{"question": "What is the speed of light in a vacuum?",
"options": ["299,792,458 meters per second", "300,000,000 meters per second",
"100,000,000 meters per second", "500,000,000 meters per second"],
"correct_answer": 0}

```

```
]
```

```
if difficulty == 1:
```

```
    if len(easy_questions) >= 10:
```

```
        return random.sample(easy_questions, 10) # Select 10 random easy questions
```

```

else:
    return easy_questions
elif difficulty == 2:
    if len(medium_questions) >= 10:
        return random.sample(medium_questions, 10) # Select 10 random medium questions
    else:
        return medium_questions
elif difficulty == 3:
    if len(hard_questions) >= 10:
        return random.sample(hard_questions, 10) # Select 10 random hard questions
    else:
        return hard_questions

def update_timer(label, start_time):
    elapsed_time = int(time.time() - start_time)
    remaining_time = max(0, 10 - elapsed_time)
    label.config(text=f"Time left: {remaining_time} seconds", background='#f0f0f0')
    if remaining_time > 0:
        label.after(1000, update_timer, label, start_time)

def display_score(name, roll_number, score, correct_answers):
    score_window = tk.Toplevel()
    score_window.title("Quiz Score")

    score_frame = ttk.Frame(score_window, padding=20, style='Background.TFrame')
    score_frame.pack()

```

```

score_label = ttk.Label(score_frame, text=f"🎉 Congratulations {name} ({roll_number})!
🎉\nYour final score is: {score}/10", font=("Arial", 16))
score_label.pack(pady=10)

compliment_label = ttk.Label(score_frame, text=compliment(score, name), font=("Arial", 12))
compliment_label.pack(pady=10)

for question, answer in correct_answers:
    answer_label = ttk.Label(score_frame, text=f"{question}: {answer}", font=("Arial", 12))
    answer_label.pack()

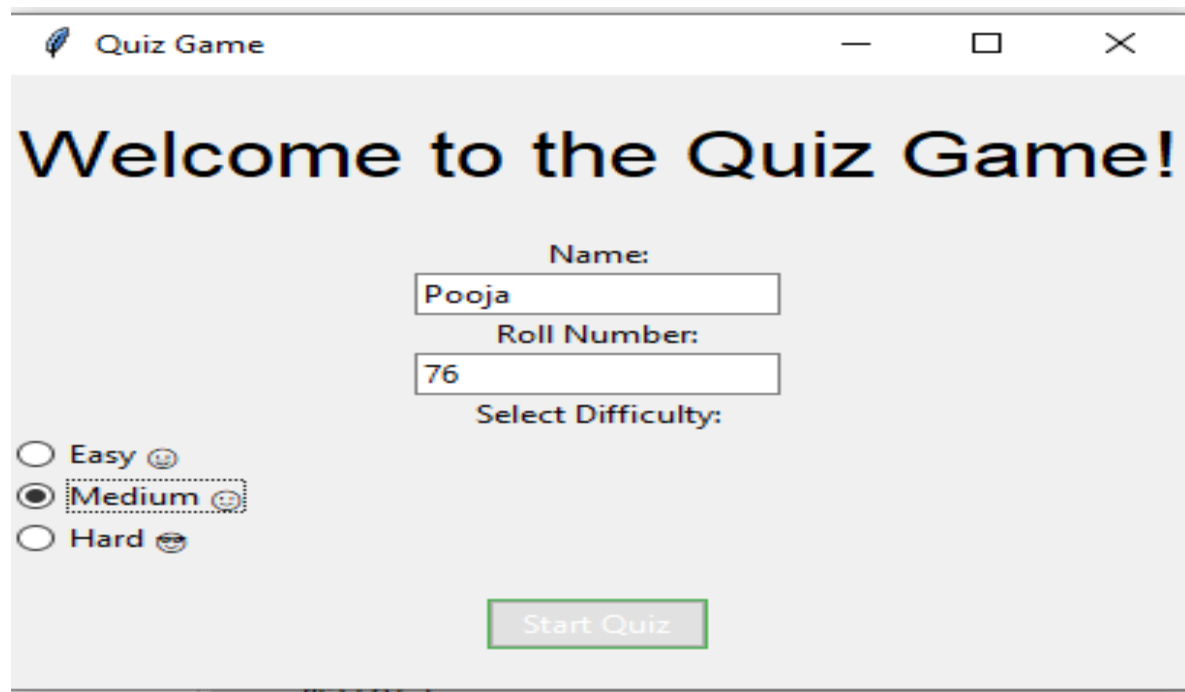
score_window.mainloop()

def compliment(score, name): # Updated compliment function
    if score >= 8:
        return 'That was GREAT!! ' + name
    elif score > 5:
        return 'NOT BAD!! ' + name
    elif score > 3:
        return 'Ohh NO,That was not Good!! ' + name
    elif score <= 3:
        return 'Ohh NO, You need Serious Focus ' + name

if __name__ == "__main__":
    main()

```

Output:



The image shows a window titled "Quiz Game" with a feather icon. The window has a light gray background and contains the following elements:

- A large heading: "Welcome to the Quiz Game!"
- A label "Name:" followed by a text input field containing "Pooja".
- A label "Roll Number:" followed by a text input field containing "76".
- A label "Select Difficulty:" followed by three radio button options:
 - ☐ Easy 😊
 - ☒ Medium 😊
 - ☐ Hard 😊
- A green rectangular button labeled "Start Quiz" at the bottom center.

Viva:

Question: What libraries did you use to build your quiz game and why?

Answer: I primarily used tkinter for creating the graphical user interface (GUI) elements like windows, buttons, and labels. This allowed for a user-friendly experience for interacting with the quiz. Additionally, I used the random library to shuffle question and answer choices, ensuring a dynamic experience for each player. (Optional: Briefly mention time library if used for timers.)

Question: How did you handle different difficulty levels in your quiz?

Answer: I created separate dictionaries containing pre-defined questions categorized by difficulty (easy, medium, hard). Based on the user's chosen difficulty, the program retrieves a set of questions from the corresponding dictionary for the quiz.

Question: How did you ensure fair play and prevent users from cheating by looking up answers during the quiz?

Answer: (If time-based gameplay is implemented) By incorporating a time limit on each question, particularly for harder questions, I encouraged focused answering and discouraged looking up answers. Additionally, randomly shuffling answer choices further discouraged memorizing answer positions.

(If time-based gameplay is not implemented, consider this answer): While a strict time limit wasn't implemented, the focus was on creating a self-paced learning experience. The application provides users with the opportunity to assess their knowledge without undue pressure.

Question: What improvements or additional features would you consider adding to your quiz game?

Answer: This project serves as a solid foundation for a quiz game. Some potential improvements include:

Expanding the question database to cover a wider range of topics.

Implementing a score leaderboard to encourage competition among users.

Adding sound effects or background music for a more engaging experience.

Why Choose tkinter:

Modern Look and Feel: tkinter offers widgets with a more modern and platform-consistent appearance compared to the basic widgets of tkinter. This can enhance the visual appeal of your application.

Improved Styling Options: tkinter provides more control over the styling of your widgets, allowing you to customize fonts, colors, and other visual elements to match your desired aesthetic.

Potential Performance Benefits: In some cases, tkinter widgets might offer slightly better performance, especially on certain platforms.

Why Consider tkinter Widgets:

Simpler and Easier to Use: The basic tkinter widgets are generally simpler to learn and use, especially for beginners. They offer a straightforward approach to creating GUIs.

Wider Range of Widgets: tkinter offers a wider selection of widgets compared to tkinter. This can be helpful if you need specific functionalities like progress bars or menu bars that might not be readily available in tkinter.

Backward Compatibility: tkinter is the core GUI library included with Python by default. This ensures your application will run on any system with Python installed, even if tkinter is not explicitly available.

Here's a recommendation based on your project needs:

For a modern-looking GUI with some customization options: Choose tkinter for its improved aesthetics and styling capabilities.

For a simple, functional GUI or if you need a specific widget not available in tkinter: Opt for tkinter due to its ease of use and wider widget selection.

For maximum compatibility across different systems: Stick with tkinter as it's guaranteed to be available on any Python installation.

In many cases, it's perfectly acceptable to use a combination of both tkinter and ttk widgets within your application. This allows you to leverage the best of both worlds: a modern look from tkinter and the wider functionality or simpler use case of tkinter widgets when needed.