PYTHON PROJECT

Title : Rock, Paper and scissor game

NAME : NUPUR HEDAU

REGISTRATION NUMBER : 25MIP10123

COURSE: PYTHON ESSENTIALS

FACULTY: G.PRABHU KANNA

Introduction

The rock, paper and scissor is a very old and traditional game that has been engaging people around the world for so many years now. Traditionally it is a two-player game where rock is indicated by fist, scissor with fist with index and middle finger extended and paper with open hand. The outcome of this game is determined by the rules as follows rock crushes scissors, scissors cut paper, and paper covers rock.

In the context of this python project, I was aiming to make similar kind of game with computer instead of two-player game. One player will be the user itself and other play will be the computer. As a computer-based application to demonstrate fundamental programming concepts such as conditional logic, random number generation, user input handling, and result evaluation. The project also emphasizes the importance of user-friendly interfaces and interactive gameplay.

Problem Statement

Design and implement a console-based Rock, Paper, Scissors game in Python where a user can play against the computer. The game should allow the user to input their name and choice (rock, paper, or scissors) while the computer randomly selects its choice. The program must validate user inputs, determine the winner of each round based on standard rules, maintain and display a running scoreboard tracking the user and computer wins, and provide a menu interface for the user to choose to play, view rules, check the scoreboard, or exit the game. The implementation should be modular, using functions for input handling, game logic, scoring, and user interaction, ensuring a user-friendly and robust gaming experience without reliance on advanced string formatting features.

Functional Requirements

Three Major Functional Modules

1. **User Interaction Module**
    - Handles all user inputs including player name, game choice (rock, paper, scissors), and menu selections.
    - Validates input to ensure it matches allowed options.
    - Displays menus, instructions, game results, and scoreboard information.
    - Example functions: input_from_user(), main_menu(), rules(), dis play_score().
2. **Game Logic Module**
    - Contains the core game rules used to determine the winner of each round.
    - Compares user and computer choices and assesses round outcomes.
    - Updates scores based on results.
    - Example function: the_game().
3. **Computer Choice Module**

- Generates a random choice from the available options to simulate the computer's play.
- Uses the Python standard library's random module.
- Example function: computer_choice().

Clear Input/Output Structure

- **Inputs:**
  - User name (string) via console prompt.
  - User game choice (string) validated against allowed moves.
  - Menu choice (integer) for selecting play, rules, scoreboard, or exit actions.
- **Outputs:**
  - Console messages displaying prompts, invalid input warnings, and gameplay results.
  - Scoreboard showing ongoing user and computer scores.
  - Printed game rules and menu options.

Logical Workflow of User Interaction with the System

1. **Startup**
   - Application starts and displays a welcome message and menu options.
2. **Menu Selection**

- User inputs a choice:
    - Press 1 to play a game round.
    - Press 2 to view rules.
    - Press 3 to view scoreboard.
    - Press 4 to exit.

3. **Playing a Round**
    - User inputs name and their move.
    - Computer randomly selects a move.
    - Game logic module evaluates winner and updates scores.
    - Displays both players' choices and the round result.

Non- Functional Requirements

1. Performance

The game must respond instantly to user inputs and display results immediately to provide a smooth user experience.

The random choice generation for the computer must be efficient and not introduce perceptible delays.

2. Usability

The user interface must be intuitive and easy to navigate via the console menu.

Clear prompts, instructions, and error messages must assist users in making valid inputs.

The scoreboard and game rules must be displayed in a readable format.

3. Reliability

The system must consistently validate user inputs to prevent invalid selections or errors during gameplay.

The application must handle unexpected inputs gracefully without crashing or terminating abruptly.

4. Maintainability

The code is modular, with separate functions for input, game logic, display, and menu control, making it easy to update or extend.

The use of simple and widely supported string formatting (% operator) ensures broad compatibility and easier debugging.

5. Error Handling Strategy

The program validates user choices and menu inputs, repeatedly prompting until correct input is provided.

Invalid menu options and game choices trigger informative messages without breaking application flow.

System Architecture

The system follows a simple modular architecture where distinct components handle user interaction, game logic, score management, and display. It employs a procedural programming approach structured with functions as building blocks.

Design Diagrams

Use Cases:

- Start Game (User starts or restarts the game)
- Input Player Name and Choice
- Generate Computer Choice
- Evaluate Winner
- Update & Display Scoreboard
- Show Game Rules
- Exit Game

Workflow Diagram

1. Start application → Show Main Menu
2. User chooses:
    - Play → Input Player Name → Input Player Choice → Generate Computer Choice → Evaluate Winner → Update Scores → Display Scores → Return to Main Menu
    - Rules → Display Rules → Return to Main Menu
    - Scoreboard → Display Scores → Return to Main Menu
    - Exit → End application

Sequence Diagram

- User → Main Menu: select option
- Main Menu → input_from_user(): get player name and choice
- Main Menu → computer_choice(): generate random choice
- Main Menu → the_game(): compare choices and update scores
- Main Menu → display_score(): show current scoreboard
- Main Menu ← User: next input or exit

Design Decisions & Rationale

- Modular Functions: The program is split into small focused functions (input_from_user(), computer_choice(), etc.) for readability and maintainability.
- Procedural Approach: Since the project is simple, functions provide sufficient modularization without class overhead.
- Menu-Driven Flow: The main loop uses a menu system for easy user interaction and extending the UI.
- Random Choice for Computer: Uses Python's random.choice to simulate an unpredictable opponent.
- Scores Handled In-Memory: Scores are kept in variables during execution as persistence is not a project requirement.
- User Input Validation: Ensures robustness and prevents invalid game states.
- No Advanced String Formatting: Uses % operator for compatibility and simplicity.
- Expandable Architecture: The design allows future enhancements such as extending choices, saving scores, or GUI integration.

Implementation details

Implementation Details

1. **Module Imports**
    - Imports the built-in random module to enable the computer to make a random choice from the valid game options.
2. **Function: input_from_user()**
    - Presents a prompt for the player to enter their name and their game choice (rock, paper, or scissors).
    - The input is converted to lowercase to standardize comparisons.
    - Uses a validation loop to ensure the user inputs one of the permitted options; if invalid, it prompts again until valid input is received.
    - Returns a tuple containing the user's choice and their name.
3. **Function: computer_choice()**
    - Defines the three valid options in a list.
    - Returns a randomly selected choice from the list, simulating the computer's move.
4. **Function: rules()**

- Prints the game rules to the console, explaining how the game choices interact and how winners are determined.

5. **Function: the_game(user, computer, user_score, computer_score)**
    - Takes the user's and computer's choices along with current scores as parameters.
    - Compares the choices according to game rules:
        - If choices match, declares a tie.
        - Otherwise, determines if the user wins or the computer wins and updates the respective score counters.
    - Prints the outcome message.
    - Returns the updated scores for further tracking.
6. **Function: display_score(user_score, computer_score)**
    - Prints the current scoreboard state displaying both the user's and computer's scores using formatted output.
7. **Function: main_menu()**
    - Initializes the scores to zero.
    - Presents a menu in a loop with options:
        - Play the game (choice 1),
        - Show rules (choice 2),
        - Show scoreboard (choice 3),

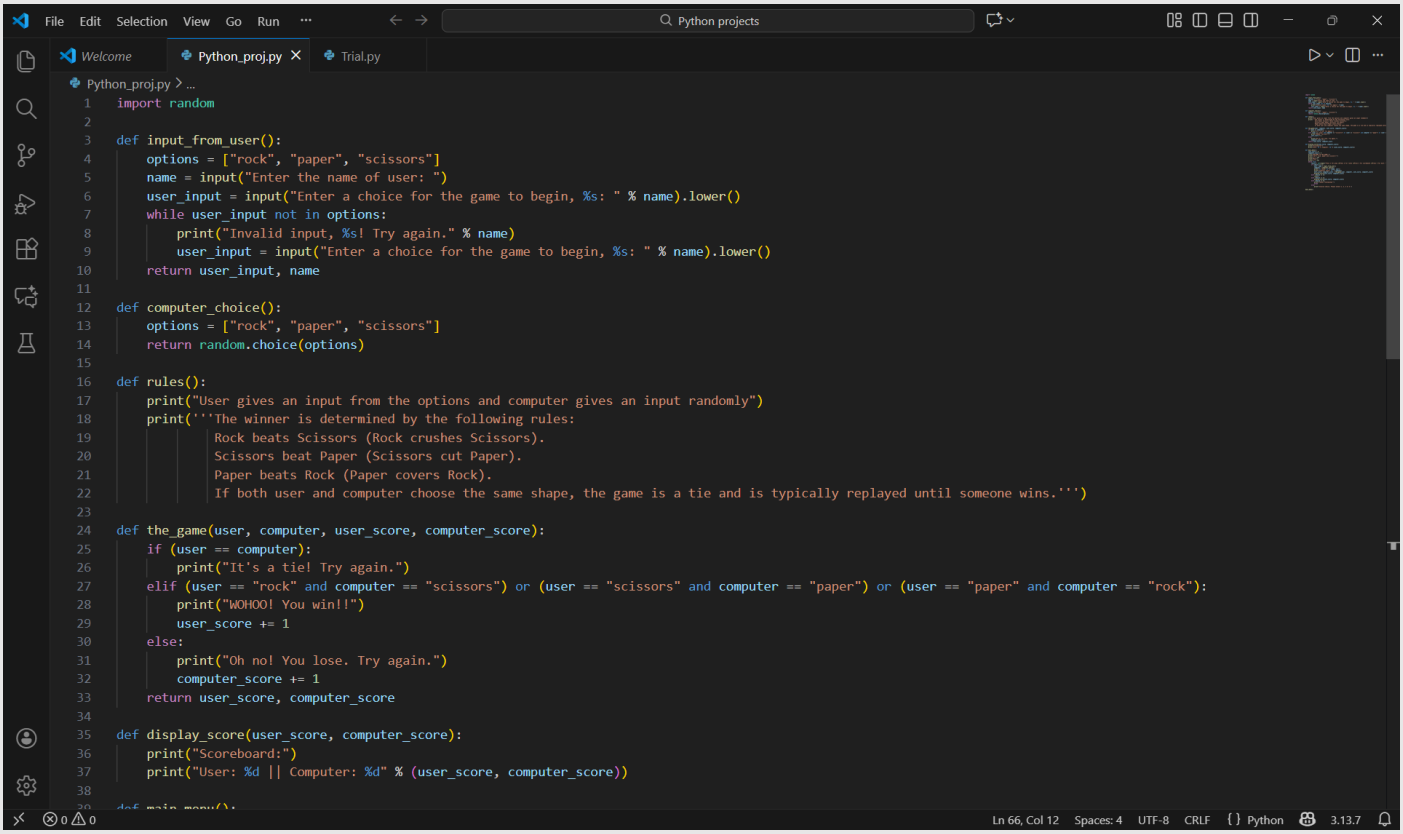
- Exit program (choice 4).

- Based on user menu selection, the program routes to the corresponding function:
  - On playing, collects inputs, generates computer choice, processes the game round, and updates scores.
  - Displays rules or scoreboard as requested.
  - Exits the loop and program on exit selection.
- Provides input validation on menu selection, prompting for valid inputs when necessary.

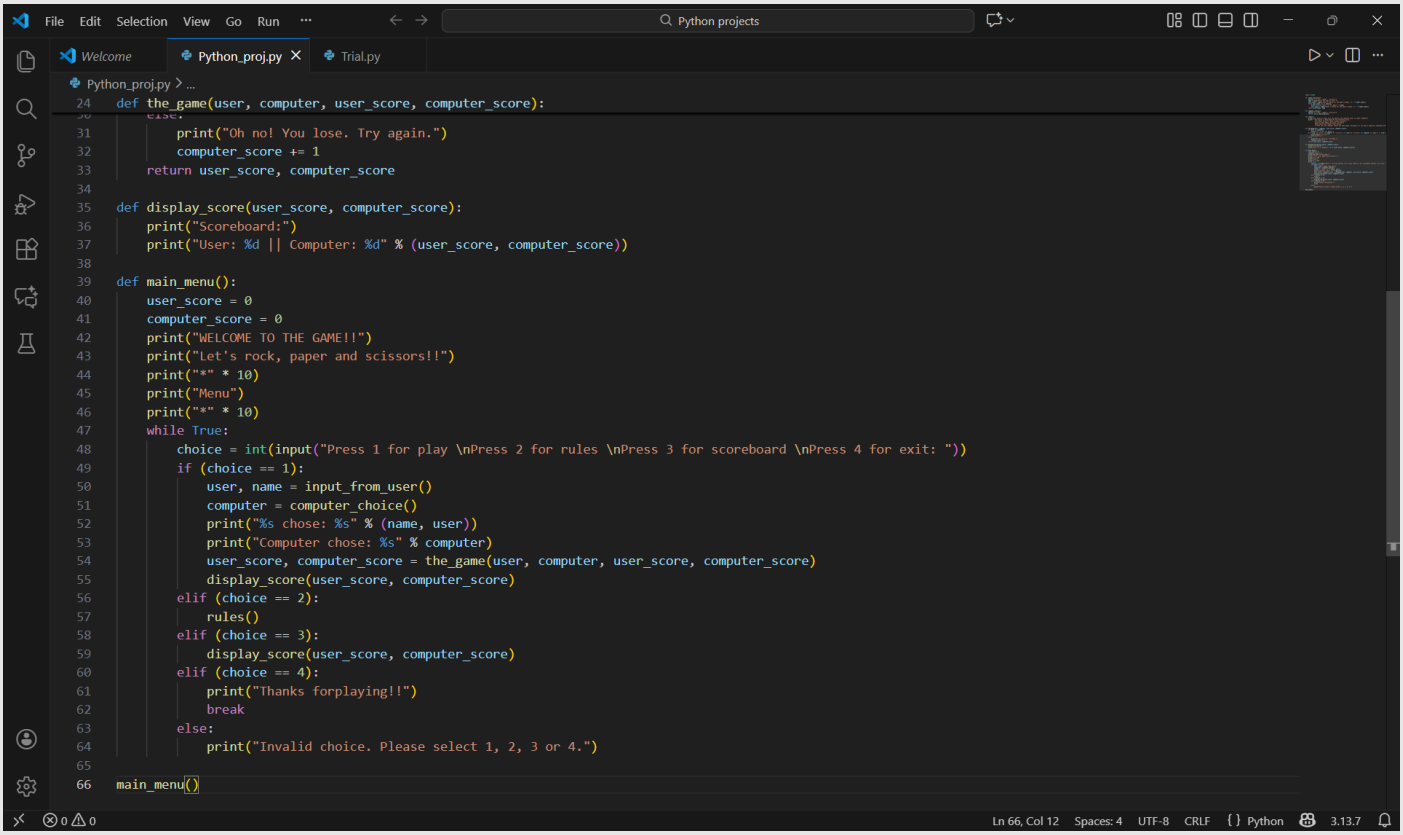
8. **Program Execution**

- The game starts with the call to main_menu(), which drives the interactive console-based game experience.

Screenshots/Results



```python
import random

def input_from_user():
    options = ["rock", "paper", "scissors"]
    name = input("Enter the name of user: ")
    user_input = input("Enter a choice for the game to begin, %s: " % name).lower()
    while user_input not in options:
        print("Invalid input, %s! Try again." % name)
        user_input = input("Enter a choice for the game to begin, %s: " % name).lower()
    return user_input, name

def computer_choice():
    options = ["rock", "paper", "scissors"]
    return random.choice(options)

def rules():
    print("User gives an input from the options and computer gives an input randomly")
    print('''The winner is determined by the following rules:
            Rock beats Scissors (Rock crushes Scissors).
            Scissors beat Paper (Scissors cut Paper).
            Paper beats Rock (Paper covers Rock).
            If both user and computer choose the same shape, the game is a tie and is typically replayed until someone wins.''')

def the_game(user, computer, user_score, computer_score):
    if (user == computer):
        print("It's a tie! Try again.")
    elif (user == "rock" and computer == "scissors") or (user == "scissors" and computer == "paper") or (user == "paper" and computer == "rock"):
        print("WOHOO! You win!!")
        user_score += 1
    else:
        print("Oh no! You lose. Try again.")
        computer_score += 1
    return user_score, computer_score

def display_score(user_score, computer_score):
    print("Scoreboard:")
    print("User: %d || Computer: %d" % (user_score, computer_score))
```



```python
def the_game(user, computer, user_score, computer_score):
    else:
        print("Oh no! You lose. Try again.")
        computer_score += 1
    return user_score, computer_score

def display_score(user_score, computer_score):
    print("Scoreboard:")
    print("User: %d || Computer: %d" % (user_score, computer_score))

def main_menu():
    user_score = 0
    computer_score = 0
    print("WELCOME TO THE GAME!!")
    print("Let's rock, paper and scissors!!")
    print("*" * 10)
    print("Menu")
    print("*" * 10)
    while True:
        choice = int(input("Press 1 for play \nPress 2 for rules \nPress 3 for scoreboard \nPress 4 for exit: "))
        if (choice == 1):
            user, name = input_from_user()
            computer = computer_choice()
            print("%s chose: %s" % (name, user))
            print("Computer chose: %s" % computer)
            user_score, computer_score = the_game(user, computer, user_score, computer_score)
            display_score(user_score, computer_score)
        elif (choice == 2):
            rules()
        elif (choice == 3):
            display_score(user_score, computer_score)
        elif (choice == 4):
            print("Thanks forplaying!!")
            break
        else:
            print("Invalid choice. Please select 1, 2, 3 or 4.")

main_menu()
```

Testing Approach

Testing different cases.

PS C:\Users\nupur\Python prog\Python projects> & C:/Users/nupur/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nupur/Python prog/Python projects/Python_proj.py"

WELCOME TO THE GAME!!

Let's rock, paper and scissors!!

****

Menu

****

Press 1 for play

Press 2 for rules

Press 3 for scoreboard

Press 4 for exit: 1

Enter the name of user: Nupur

Enter a choice for the game to begin, Nupur: rock

Nupur chose: rock

Computer chose: scissors

WOHOO! You win!!

Scoreboard:

User: 1 || Computer: 0

Press 1 for play

Press 2 for rules

Press 3 for scoreboard

Press 4 for exit: 1

Enter the name of user: Nupur

Enter a choice for the game to begin, Nupur: paper

Nupur chose: paper

Computer chose: paper

It's a tie! Try again.

Scoreboard:

User: 1 || Computer: 0

Press 1 for play

Press 2 for rules

Press 3 for scoreboard

Press 4 for exit: 1

Enter the name of user: Nupur

Enter a choice for the game to begin, Nupur : Scissor

Invalid input, Nupur ! Try again.

Enter a choice for the game to begin, Nupur : 1

Invalid input, Nupur ! Try again.

Enter a choice for the game to begin, Nupur : scissors

Nupur chose: scissors

Computer chose: paper

WOHOO! You win!!

Scoreboard:

User: 2 || Computer: 0

Press 1 for play

Press 2 for rules

Press 3 for scoreboard

Press 4 for exit: 1

Enter the name of user: Nupur

Enter a choice for the game to begin, Nupur: paper

Nupur chose: paper

Computer chose: scissors

Oh no! You lose. Try again.

Scoreboard:

User: 2 || Computer: 1

Press 1 for play

Press 2 for rules

Press 3 for scoreboard

Press 4 for exit: 2

User gives an input from the options and computer gives an input randomly

The winner is determined by the following rules:

Rock beats Scissors (Rock crushes Scissors).

Scissors beat Paper (Scissors cut Paper).

Paper beats Rock (Paper covers Rock).

If both user and computer choose the same shape, the game is a tie and is typically replayed until someone wins.

Press 1 for play

Press 2 for rules

Press 3 for scoreboard

Press 4 for exit: 3

Scoreboard:

User: 2 || Computer: 1

Press 1 for play

Press 2 for rules

Press 3 for scoreboard

Press 4 for exit: 4

Thanks forplaying!!

PS C:\Users\nupur\Python prog\Python projects>

Challenges Faced

I faced some of the minor challenges which later were solved using simple solutions by dividing the problem into sub-parts and working in that. Then gathering all the solved sub-parts to one to solve the major problems.

Like using elif statement instead of using several if statements. Using elif statement for multiple choice and else for teminating. Conversion of all input to lower case for consistency.

Learnings & Key Takeaways

I have learned something new and interesting while doing this python project.

LEARNING OUTCOMES

- Basic python-syntax : Learning how to use loops, condition statement, basic functions and module.
- Randomization : Using random module to generate random choices for the computer's choice.
- String formatting : Using the string formation to store the necessary output and variables.
- Score tracking : Tracking the score of the winner and displaying for the user to understand easily and learning to update it properly.
- Menu System : Created a simple menu-driven program to display the menu for the user and giving the user to select the choice.

KEY TAKEAWAYS

- Game logic : Understanding game rules and translating them into code is a core skill in game development.
- **User Experience**: A clear menu and feedback help make the game enjoyable and easy to use.

- **Modularity**: Breaking code into functions makes it easier to debug, test, and extend.
- **Input Validation**: Always validate user input to prevent errors and improve user experience.
- **String Formatting**: Different formatting methods (%, .format(), f-strings) let you display dynamic information clearly.

Future Enhancements

There’s a few future enhancements we can make to make this project to next level advance project :-

- By using Artifical Intelligence (AI). By using animation and sound effects using libraries like pygame.
- Allow two users to play against each other instead of just user vs computer.
- Use a graphical interface (e.g., Tkinter) for a more interactive experience.

Reference

1. **Real Python - Make Your First Python Game: Rock, Paper, Scissors**
   https://realpython.com/python-rock-paper-scissors/
   A comprehensive Python tutorial demonstrating game creation including user input, randomization, and game logic.
2. **GeeksforGeeks - Python Program to Implement Rock Paper Scissor Game**
   https://www.geeksforgeeks.org/python/python-program-implement-rock-paper-scissor-game/
   Step-by-step guide and code for designing Rock-Paper-Scissors using Python with explanations.
3. **Codingal - Rock, Paper, Scissors game in Python**
   https://www.codingal.com/coding-for-kids/blog/rock-paper-scissors-game-in-python/
   Beginner-friendly project outline showing foundational programming concepts via the RPS game.