Exercise: Rock, Paper, Scissors

Table of Contents

Background

Rock, paper, scissors is a game for two players. Each player chooses one of three hand shapes: rock, paper, or scissors. In the real world, the players show each other their hand shapes at the same time. The winner is determined as follows: rock defeats scissors; scissors defeats paper; paper defeats rock. If both players select the same hand shape, the round ends in a tie.

Instructions

For this and all future exercises the driver does the typing, while the navigator watches for errors and helps to plan ahead for the next steps. Drivers and navigators can switch roles as often as they would like throughout the pair programming session according to the pair programming conventions. Both driver and navigator should be focused and actively engaged on the programming task throughout the session.

Use any code editor to write a Python script called `rps.py`. To make this script testable on GradeScope, please use the template provided within the assignment instructions. Specifically you will be completing the code within the body of the `determine_winner` and `main` functions. Specific functionality and instructions are detailed below.

`determine_winner()`

- Arguments
  - p1 (str) : Player 1s chosen hand sign. This value may be either "rock", "paper" or "scissors" denoted by the shorthand versions of each (ie. 'r', 'p' or 's'). We will assume that these values will always be either 'r', 'p' or 's'.
  - p2 (str) : Player 2s chosen hand sign. This value may be either "rock", "paper" or "scissors" denoted by the shorthand versions of each (ie. 'r', 'p' or 's'). We will assume that these values will always be either 'r', 'p' or 's'.
- Functionality

    - Determine the winner of the round given the values provided for `p1` and `p2` and the conventional rules for rock, paper, scissors. If both players chose the same value, the game is a tie. If:
        - p1 is the winner the function should return 'player1'.
        - p2 is the winner the function should return 'player2'.
        - The game is a tie, the function should return 'tie'.
- Returns
    - A string, either 'tie', 'player1' or 'player2'.

`main()`

- Arguments
    - player1_name (str) : Player 1s name for the round.
    - player2_name (str) : Player 2s name for the round.
- Functionality
    - In your script, ask the user to enter player 1's hand shape using the input function ("r", "p", or "s") and player 2's hand shape (same letters).
    - Call on the determine winner function by passing in the values that were inputted by the user. Save the returned value.
    - Using the returned value, the script should state the winner using a print function.
        - To make this script testable, please use the exact following statements.
            - "Tie!"
            - "{insert player 1s name here} wins!"
            - "{insert player 2s name here} wins!"

NOTE	The output examples demonstrated below assume that the script implemented a solution for clearing out the terminal once input has been entered. To make your output's form match my output exactly, please use the following snippet of code to do so. Its best to clear out the input twice (once after each input).

```
# Cross platform solution for clearing out a termianl/cmd prompt in Python.
os.system('cls||clear')
```

Template

```
"""A template for a python script deliverable for INST326.

Driver: Instructor Gabriel Cruz
Navigator: None
Assignment: Template INST326
```

```
Date: 1_24_21

Challenges Encountered: ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
"""


import sys
import argparse
import os

def determine_winner(p1, p2):
    #insert code and docstrings here
    pass

def main(player1_name, player2_name):
    #insert code and docstrings here
    pass

def parse_args(args_list):
    """Takes a list of strings from the command prompt and passes them through as
arguments

    Args:
        args_list (list) : the list of strings from the command prompt
    Returns:
        args (ArgumentParser)
    """

    #For the sake of readability it is important to insert comments all throughout.
    #Complicated operations get a few lines of comments before the operations
commence.
    #Non-obvious ones get comments at the end of the line.
    #For example:
    #This function uses the argparse module in order to parse command line arguments.

    parser = argparse.ArgumentParser() #Create an ArgumentParser object.

    #Then we will add arguments to this parser object.
    #In this case, we have a required positional argument.
    #Followed by an optional keyword argument which contains a default value.

    parser.add_argument('p1_name', type=str, help="Please enter Player1's Name")
    parser.add_argument('p2_name', type=str, help="Please enter Player2's Name")

    args = parser.parse_args(args_list) #We need to parse the list of command line
arguments using this object.

    return args

if __name__ == "__main__":
    #If name == main statements are statements that basically ask:
    #Is the current script being run natively or as a module?
```

```
    #It the script is being run as a module, the block of code under this will not be
executed.
    #If the script is being run natively, the block of code below this will be
executed.

    arguments = parse_args(sys.argv[1:]) #Pass in the list of command line arguments
to the parse_args function.

    #The returned object is an object with those command line arguments as attributes
of an object.
    #We will pass both of these arguments into the main function.
    #Note that you do not need a main function, but you might find it helpfull.
    #You do want to make sure to have minimal code under the 'if __name__ ==
"__main__":' statement.

    main(arguments.p1_name, arguments.p2_name)
```

Running your program

Your program is designed to run from the terminal. To run it, open a terminal and ensure you are in the directory where your script is saved.

The program takes two required command-line arguments: Player 1s Name, and Player 2s Name.

Below are some examples of how to use the program. The examples assume you are using macOS and your program is called `rps.py`. If you are using Windows, replace `python3` with `python`.

Basic usage

```
python3 rps.py Lorena Vicky
```

Output:

```
Enter player 1's hand shape ('r', 'p', or 's'):
```

> **NOTE** The program will not proceed until you provide inputs.

Basic Usage with Multiple Inputs

Inputs:

```
python3 rps.py Lorena Vicky
r
p
```

Output:

```
Vicky wins!
```

Submitting your script

Whether or not the instructional team member has had a chance to look over your script, at the end of the session, the driver should email a copy of the script to the navigator. If you did not complete the script during pair programming time, you may complete the script together at a later time, or each of you may complete the script individually. **It is not okay for one team member to complete the script for the other person without that person's participation.**

Both driver and navigator should upload the script to GradeScope, even if the instructional team member has signed off on the script.