UDACITY

PROJECT SPECIFICATION

Rock Paper Scissors

Gameplay

CRITERIA	MEETS SPECIFICATIONS
The program plays a game of Rock Paper Scissors, following the conventional rules.	Paper beats rock; rock beats scissors; scissors beat paper.
The program plays a match consisting of multiple rounds, and tracks players' total score.	The game displays the results after each round, including each player's score. At the end, the final score is displayed. The number of rounds per game, as well as when to stop, are up to you!
There are at least *four* different computer player classes, each implementing a different strategy.	The game should have (at least) four computer player strategies: • A player that always plays 'rock' • A player that chooses its moves randomly. • A player that remembers and imitates what the human player did in the previous round. • A player that cycles through the three moves

CRITERIA	MEETS SPECIFICATIONS
Each player class has a method that returns that player's move, and a method for remembering information about the round.	The game should call each player's move method once in each round, to get that player's move. After each round, it should call the remembering method to tell each player what the other player's move was. Some computer players don't need to remember anything, so their remembering method should do nothing.

Object-Oriented Programming

CRITERIA	MEETS SPECIFICATIONS
The code uses classes and objects to store game data, rather than global variables.	The `Game` class should include a method to play a single round, and a method to play a match of several rounds. Facts about the current match, such as the players' score, or the number of rounds played, should be stored as instance variables. They shouldn't be stored as global variables. It's okay to use global variables for the game moves "rock", "paper", and "scissors".
The code uses subclasses appropriately.	Each computer player strategy should be a subclass of the `Player` base class, as should the `Human` player.

Code Style

CRITERIA	MEETS SPECIFICATIONS
The code style follows the standard Python style guide.	The `pycodestyle` tool should report zero errors and zero warnings. If the program is called `rps.py`, the command to test it is `pycodestyle rps.py`.
The program does not crash or display any error messages.	The code should be thoroughly tested. Invalid moves should not make the program crash. *(See the next item!)*
The program checks the validity of user input.	If the player enters a move that is not valid, the game should give them the chance to retry that move until they enter a valid move. The game should not crash, and it should not treat invalid input as a valid move. **Example:** If the player enters "roxk" instead of "rock", the game should let them try again; it should not crash, and it should not assume they meant "rock".

Suggestions to Make Your Project Stand Out!

- Adapting the game to play **Rock Paper Scissors Spock Lizard** or another expanded version of the game.
- Running a **tournament** of many computer players, with different strategies, and discovering which strategies win the most!

- Adding **color** or other features to the game display. (Look up "terminal color codes" in your favorite search engine.)