UDACITY Logout

‹ Return to "Intro to Programming Nanodegree" in the classroom

Rock Paper Scissors

REVIEW | CODE REVIEW 7 | HISTORY

Meets Specifications

Very well done in your submission 👍. **Congratulations,** 🎉 you met all the required specifications 👏.
Keep up the great work 📈🏆.

> Yes, your code satisfies all the `pycodestyle` from the first submission. I guess the previous reviewer made a mistake informing you about the actual problem which was the input validation instead of the `pycodestyle` issues. So my apologies on behalf of the previous reviewer.

Gameplay

✓ Paper beats rock; rock beats scissors; scissors beat paper.

Well done the game is playing perfectly, that is so much fun 👍.

✓ 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!

Nice work displaying players scores after each round.

✓ 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

You have implemented all player types properly 👏.

✓ 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

✓ 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".

Good job, the `Game` class has all the required functionality.

✓ Each computer player strategy should be a subclass of the `Player` base class, as should the `Human` player.

Code Style

✓ 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`.

Excellent! Your code passes the `pycodestyle` check without any errors or warnings 👋.

✓ The code should be thoroughly tested.

Invalid moves should not make the program crash. *(See the next item!)*

The program handles invalid input very well 👍.

✓ 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".

DOWNLOAD PROJECT

7 CODE REVIEW COMMENTS ›

RETURN TO PATH