Cumulative Project: Rock Paper Scissors x99

Write functions to power a 3-round variation on the classic Rock Paper Scissors game

Rock Paper Scissors x99

Project Overview

In this project, you will build all of the logic needed for a more intense version of Rock Paper Scissors (RPS). Rather than selecting just one of Rock, Paper, or Scissors - each player will select three moves. Each move will consist of a type (Rock, Paper, or Scissors) as well as a strength value. Each player will have 99 total points to use as strength between all three of their moves. For example, an example set of moves might be:

- Move 1: Rock - 30 Strength Points
- Move 2: Rock - 60 Strength Points
- Move 3: Paper - 9 Strength Points

The strength for each move must be at least 1.

After each player's moves are chosen, they will compare moves in the order they were selected. If two moves have different types (for example, Rock vs Scissors), then normal RPS rules will apply (in this case, Rock beats Scissors). However, if two types are the same, then the move with more strength will win. If both strength values are equal, then a tie is declared.

The player that wins the majority of the three rounds will be the winner of the game.

To demo all of this functionality, try out a final version of this project, located here.

How To Begin

To start, download the starting code for this project here. After downloading the zip folder, double click it to uncompress it and access the contents of this project.

Implementation Details

All of your code should be written in the file at the following path: **js/game-logic.js**. Use the descriptions and testing suite discussed below to guide implementation of all necessary functionality.

To complete this project, your code will need to contain the following:

- Twelve global variables representing each player's move types and values (3 move types and 3 move values for each player). These variable names should be in the form of `playerOneMoveOneType`, `playerOneMoveOneValue`, etc.
- A function called `setPlayerMoves`, which will take a string representing a player (in the form of `'Player One'` or `'Player Two'`), three move types, and three move values, and set the correct global move variables. The method signature for this function should be as follows: `setPlayerMoves(player, moveOneType, moveOneValue, moveTwoType, moveTwoValue, moveThreeType, moveThreeValue)`.
- A function called `getRoundWinner`, which takes a round number (`1`, `2`, or `3`), compares both player's move types and values for that round, and returns the appropriate winner (`'Player One'`, `'Player Two'`, or `'Tie'`)
- A function called `getGameWinner`, which compares both player's move types and values for the whole game and returns the appropriate winner (`'Player One'`, `'Player Two'`, or `'Tie'`)
- Bonus: A function called `setComputerMoves`, which chooses three random moves for player two. The move type for each move should be completely random, and the move values should be random but add up to 99.

To demo your version of the game, open **index.html** in your browser (by double clicking **index.html** in a file browser or dragging it into your Internet browser). You will be writing JavaScript code that uses new syntax (you will learn more about this later), so you will need to use the most up-to-date version of Chrome to ensure your code runs correctly. If your version of Chrome is too old, correctly-written code may still not run as expected.

Disclaimer: If you have prior JavaScript or programming experience, you might be able to come up with an implementation that uses JavaScript features such as Arrays or best practices that we haven't yet covered. The implementation details and the tests for this project require a specific implementation based upon the material covered so far: JS types, variables, functions, and scope, but not topics that will be covered in later units.

As you continue, the project specifications and tests will become less granular, allowing you to implement the required functionality on your own.

Testing

A testing suite has been provided for you, checking for all essential functionality and edge cases.

To run these tests, first open the root project directory in your terminal. Then run `npm install` to install all necessary testing dependencies (you will only need to do this step once). Finally, run `npm run test`. You will see a list of tests that ran with information about whether or not each test passed. After this list, you will see more specific output about why each failing test failed.

As you implement functionality, run the tests to ensure you are creating correctly named variables and functions that return the proper values. The tests will additionally help you identify edge cases that you may not have anticipated when first writing the functions.

Debugging

Watch the video below to learn how to use Chrome DevTools to help you debug as you complete the RPS project.