# Practice Project: Rock-Paper-Scissors-Lizard-Spock

**Practice with Lab Sandbox**

This project is the first required programming project in our Python 2 programming class "[An Introduction to Interactive Programming in Python](#)" (IIPP). You are encouraged to complete this optional practice project to get more hands-on experience working in Python before you attempt the required project for this week.

## Practice Project: Rock-paper-scissors-lizard-Spock

Rock-paper-scissors is a hand game that is played by two people. The players count to three in unison and simultaneously "throw" one of three hand signals that correspond to rock, paper or scissors. The winner is determined by the rules:

- Rock smashes scissors
- Scissors cuts paper
- Paper covers rock

Rock-paper-scissors is a surprisingly popular game that many people play seriously (see the [Wikipedia article](#) for details). Due to the fact that a tie happens around 1/3 of the time, several variants of Rock-Paper-Scissors exist that include more choices to make ties less likely.

Rock-paper-scissors-lizard-Spock (RPSLS) is a variant of Rock-paper-scissors that allows five choices. Each choice wins against two other choices, loses against two other choices and ties against itself. Much of RPSLS's popularity is that it has been featured in 3 episodes of the TV series "The Big Bang Theory". The [Wikipedia entry](#) for RPSLS gives the complete description of the details of the game.

In this practice project, we will build a Python function $rpsls(name)$ that takes as input the string $name$, which is one of $"rock"$, $"paper"$, $"scissors"$, $"lizard"$, or $"Spock"$. The function then simulates playing a round of Rock-paper-scissors-lizard-Spock by generating its own random choice from these alternatives and then determining the winner using a simple rule that we will next describe.

While Rock-paper-scissor-lizard-Spock has a set of ten rules that logically determine who wins a round of RPSLS, coding up these rules would require a large number (5x5=25) of $if/elif/else$ clauses in your project code. A simpler method for determining the winner is to assign each of the five choices a number:

- 0 — rock
- 1 — Spock
- 2 — paper
- 3 — lizard
- 4 — scissors

In this expanded list, each choice wins against the preceding two choices and loses against the following two choices (if rock and scissors are thought of as being adjacent using modular arithmetic).

For this practice project, we will provide a walk-through of the steps involved in building your project to aid its development. A template for the project is **[available here](#)**. Please work from this template.

# Project development process

1. Build a helper function name_to_number(name) that converts the string name into a number between 0 and 4 as described above. This function should use a sequence of if/elif/else clauses. You can use conditions of the form name == 'paper', etc. to distinguish the cases. To make debugging your code easier, we suggest including a final else clause that catches cases when name does not match any of the five correct input strings and prints an appropriate error message. You can test your implementation of name_to_number() using this name_to_number testing template. (Also available in the Tips for RPSLS page).

2. Next, you should build a second helper function number_to_name(number) that converts a number in the range 0 to 4 into its corresponding name as a string. Again, we suggest including a final elseelse clause that catches cases when number is not in the correct range. You can test your implementation of number_to_name() using this number_to_name testing template.

3. Implement the first part of the main function rpsls(player_choice). Print out a blank line (to separate consecutive games) followed by a line with an appropriate message describing the player's choice. Then compute the number player_number between 0 and 4 corresponding to the player's choice by calling the helper function name_to_number() using player_choice.

4. Implement the second part of rpsls() that generates the computer's guess and prints out an appropriate message for that guess. In particular, compute a random number comp_number between 0 and 4 that corresponds to the computer's guess using the function random.randrange(). We suggest experimenting with randrange in a separate CodeSkulptor window before deciding on how to call it to make sure that you do not accidently generate numbers in the wrong range. Then compute the name comp_choice corresponding to the computer's number using the function number_to_name() and print an appropriate message with the computer's choice to the console.

5. Implement the last part of rpsls() that determines and prints out the winner. Specifically, compute the difference between comp_number and player_number taken modulo five. Then write an if/elif/elseif/elif/else statement whose conditions test the various possible values of this difference and then prints an appropriate message concerning the winner. If you have trouble deriving the conditions for the clauses of this if/elif/else statement, we suggest reviewing the "RPSLS" video which describes a simple test for determine the winner of RPSLS.

To test your project, you will call your rpslsfunction repeatedly in the program with different player choices. You will see that we have provided five such calls at the bottom of the template. Running your program repeatedly should generate different computer guesses and different winners each time.