# Python Team Project

Large development projects are rarely done by one person alone. Instead most developers work in teams. Thus, understanding how to divide up work and collaborate are essential to computer programming.

In this project you will work in teams of two. To get started talk over the following points with your teammate. It is also recommended that you put your division of work in writing as you will need to turn in a summary of your team experiences.

- Introduce yourself.
- Share your computer background.
- Discuss if you have common time to get together outside of class.
- Review the materials for this assignment.
- Determine your strategy for completing the game. Example strategies are:
    1. Assign different functions of the game to different team members, then integrate and test.
    2. Decide if you want to work on the same file, or work in separate files, combining them as you go.
    3. Have one team member type the program, while working and discussing the implementation together.
    4. Have one team member type the program, while working and discussing the implementation together. Halfway through switch roles.
    5. Have one team member implement, while another team member tests
    6. Be sure sure strategy includes how you will test the code, don't wait until it is all done to test.
    7. Other ideas

To start, go to the assignment page and left click on the link titled `rps.tar`. Select the radio button titled `Save File` then execute the following commands.

```
denver:~$ cd python_fun
denver:~/python_fun$ mkdir team_project
denver:~/python_fun$ cd team_project
denver:~/python_fun/team_project$ mv ~/Downloads/rps.tar ./
denver:~/python_fun/team_project$ tar xvf rps.tar
denver:~/python_fun/team_project$ chmod +x rps.py
denver:~/python_fun/team_project$ gedit rps_back.py &
denver:~/python_fun/team_project$ ./rps.py
```

When you click `Start Game` in the window that appeared, the game should close and you should get a message on you console that says "Not implemented yet." This is OK, you will add the implementation.

Some things to note, the ‘x’ in the top right corner of the game window does not always work, instead use the “Quit” button in the bottom left corner to quit the game. Also we are working with Python version 2.7 in this assignment because the libraries used were written in this version. The only notable difference here is that the print function does not use parentheses any more.

## Project Description

For this project you will be implementing the back end for a Rock, Paper, Scissors game. This means that you will write the part that actually handles the game. The front end (presentation part) has already been implemented and you will use it as a library.

The file `rps.py` is the front end for the game. This is also the file that you will run from the command line. This program will call your functions and depends on them working for a fully functional game. You will not make any changes to the `rps.py` file. All changes you make will be made to `rps_back.py`. You will also be using the library `matplotlib`, which we used in the last lab.

## Getting Started

The first function we will start with is `instructions`. The current code in this function is shown below.

```python
def instructions(response):
    # Add to instruction_string so that it contains information on
    # how to play rock-paper-scissors
    instruction_string = ""

    # Use a string method to make response all one case

    # Use an if statement to check if the response is "yes"

    # If the user does want instructions pass instruction_string to
    # rps.print_instructions
```

To start, we need to fill `instruction_string` with the correct instructions for Rock, Paper, Scissors. Add the following code to this function.

```python
instruction_string = "Choose rock, paper, or scissors from the buttons. "
instruction_string += "The computer will then choose a move. "
instruction_string += "Rock beats scissors, scissors beats paper, and paper beats
    rock."
```

Now that we have the instructions ready, we need to check the response. The response can be in any case, but we should only display the instructions if they respond “yes.” Without having to check all the combinations of casings, let's just make the string all lower case and match that with "yes." Add the following line below the comment line that says: "Use a string . . . "

```
response = response.lower()
```

Finally we use an if statement and call `rps.print_instructions` if the response was yes. Add the following line below the comment line that says "Use an if statement . . . "

```
if response == "yes":
    rps.print_instructions(instruction_string)
```

Now you need to test it to make sure it works! The `play_match` function is where the program execution starts, so lets go to that function and add the following line to the beginning. Note that this should be the first statement in the function.

```
rps.ask_instructions()
```

When any form of "yes" is entered you should see a screen with your instructions, any other input should skip this screen and the game should close (because nothing else is implemented yet). Test other cases to make sure that it is working well.

Now that you have a feel on how to make changes and test, continue to add implementation for your game based on the strategy that you and your partner have decided on. Pay attention to the order in which you add code to make sure you can test as you go along.

If you get lost, brainstorm with your partner, look at the `rps_hints.py` file on the assignment page, look at previous python labs for this class, consult google, ask one of the instructors/class helpers, come to office hours, look at the optional functions descriptions below. i.e. there are lots of strategies for help!

# Function Descriptions (optional clarifications)

## 0.1 `play_match`

Your "main" function will be `play_match`, which will be called almost immediately when you run the `rps.py` program. Here you will need to call three separate functions from `rps`.

– `rps.ask_instructions`

  This functions will prompt the user if they need instructions. Their response will be passed along to your `instructions` function. There is not return value from this call.

– `rps.get_name`

  This function prompts the user for their name, then calls your function titled `check_name`. The return value is what you return from `check_name`.

- `rps.get_num_play`

  This function prompts the user for how many times they would like to play. The response is then passed to your `check_times_to_play` function. The return value is what you return from `check_times_to_play`.

It is recommended you implement the following functions, in the order presented, before finishing with `play_match`.

## 0.2 `instructions`

This function takes one parameter, `response`. This variable contains the users response to the prompt "Would you like instructions (yes/no): ". If this response is "yes," in any combination of case, call `rps.print_instructions`, passing it a string variable which has instructions for how to play Rock, Paper, Scissors.

## 0.3 `check_name`

For this function you will need to check that the parameter `name` starts with an uppercase letter, is of length greater than one and less than ten, and that it is only one name (i.e. no spaces). If any of these requirements are not met call `rps.quit_game`, passing a string with an appropriate error message. If the name is OK, then return it.

## 0.4 `check_times_to_play`

This function is similar to `check_name`. You are passed a variable `num` which you must check that it is greater than two and less than twenty one. If these requirements are not met call `rps.quit_game`, passing a string with the error message. If the number is OK, then return the integer version of it (it comes in as a string).

## 0.5 `play_game`

This function takes one parameter, the name of the player. You must get the players move by calling `rps.get_player_move`, which will return a string of what the player chose. You then must generate a move for the computer using the `random.random` function (note this returns a decimal number from zero, inclusive, to one, exclusive).

Once you have the computers move and the players move, determine who one. Call `rps.display_results` with a string that spans three lines, and describes the players move, the computers move, and who won (or tied).

Finally, return a string describing who won.

## 0.6 `play_match`

Once the other functions are working (and you can test them by calling them from this function) you will need to add the rest of the implementation for `play_match`. This function will call `play_game` the number of times that was received from `rps.get_num_play`. Think about how you can use a while loop to do this (or look up “for loop python” on google and use that). You must keep a running tally of the number of times the player won, the computer won, and ties.

At the end of the loop call `make_graph`, with parameters for the name of the player, number of wins for the player, number of wins for the computer, and number of ties (in that exact order).

## 0.7 `make_graph`

Using the parameters you must create a graph to appropriately display the results of the Rock, Paper, Scissors game. The type of graph is up to you. Look to the last lab, or the link to the documentation to get a template of how to make the graph.

# Submission

When your code is working correctly refer to the assignment page for submission instructions.