Thomas Herm

Pre-Engineering

CSE 120

Final Project

ROCK PAPER SCISSORS

Table of Contents:

Project Deliverables:

The deliverables are as follows:

Play a certain amount of RPS rounds based on user input. Player should input a case-insensitive (e.g. “rock” == “RoCk”) value of rock, paper, or scissors, and the program should handle improper inputs. Computer should randomly generate a value of rock, paper, or scissors.

Rock should beat scissors, scissors should beat paper, and paper should beat rock. If the player and computer pick the same value, the round should restart.

Game should track points, and declare a winner at the end, with the option to go again.

If it’s a draw, the player should have the option for a tie breaker. Game should exit when player does not want to restart game.

Project Timelines:

The timeline of the project was very different from the projected timeline. In reality, most of the code was finished shortly after receiving the project. I hit a wall when I ran into interdependency issues, and took a break to focus on other courses. I came back in early April and refactored the interdependent code into one file. This is also when the remaining bugs were squashed. On the last week of classes I created the presentation, and I used the last couple days to assemble the pdf.

Received project – 0 w

Most code complete – 1 w

Break - (rest of unaccounted weeks)

Finish code – 1 w

Make presentation – 1 w

Assemble pdf – ½ w

Issues faced:

I usually try to condense my code into smaller files, but I ran into interdependency issues doing this. I had considered putting it all in one file from the onset of this problem, but I forgot that code order did not matter in python, and so I thought the task was a lot more daunting than it actually was.

I also ran into an issue with exception types. Most error handling in my code can be done with a simple _ case in the match statement, but inputting a non-integer for rounds_wanted does need try-except, or it'll crash. I originally tried to use except: TypeError, but this did not fix the problem. This is presumably because all input is string, which is the expected type, and I was trying to convert to an integer, which was expected. The problem was that the **value** entered could not be converted, so I had to use ValueError instead.

Lessons Learned:

1. Interdependent code should be in the same file
2. Code order does not cause issues in Python
3. TypeError is for *unexpected* types
4. ValueError is for invalid or unexpected values – including during type conversion

The first two are fairly important, as they caused a lot of headache early on that could have been easily prevented. Had I remembered that code order is largely unimportant, it would have made the prospect of moving everything to one file much less worrying.

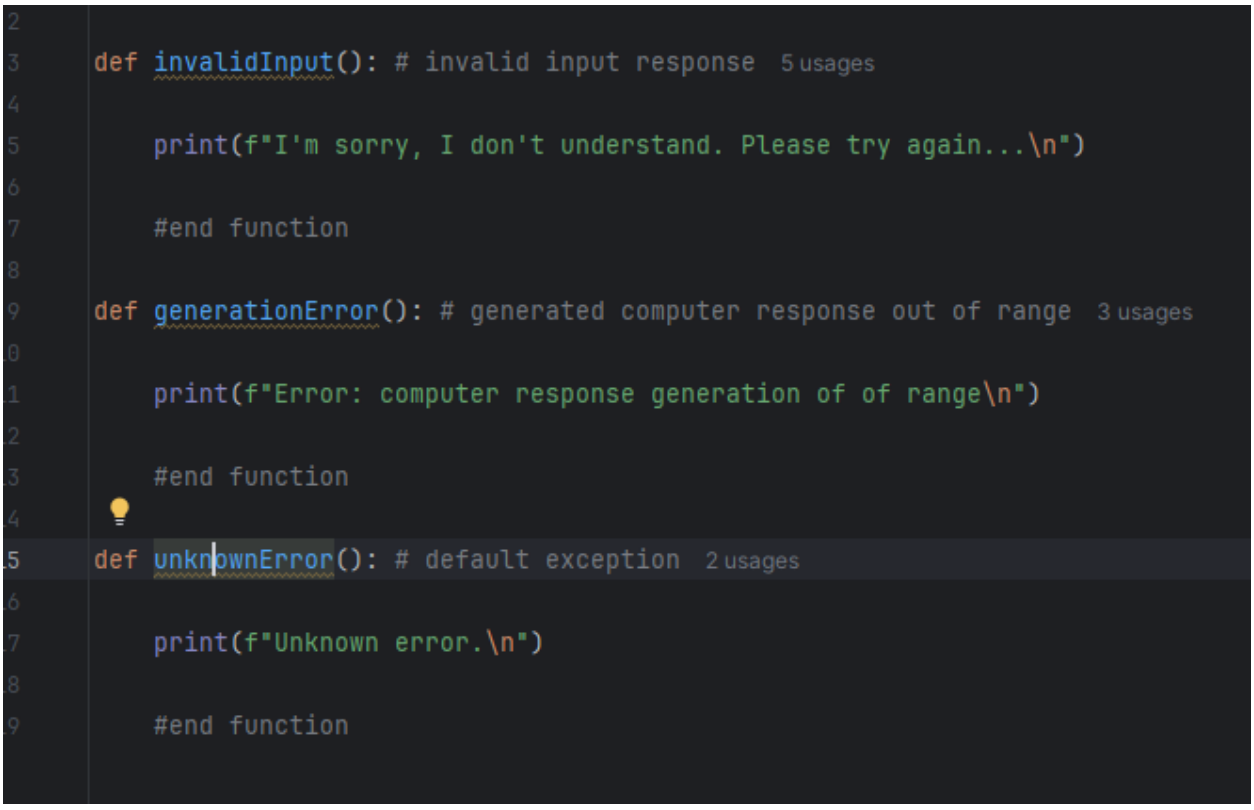
The last couple did not give me too much trouble, but was greatly confusing. It only took me searching for different error types to fix and evaluate. It is a very helpful differentiation moving forward.

Code Breakdown:

Now I'll be breaking down the different files of code.

SEC.py:

SEC.py contains all the security related code, which in this case, is error messages. There are error messages for input errors, generation errors, and unknown errors.



```
def invalidInput(): # invalid input response  5 usages

    print(f"I'm sorry, I don't understand. Please try again...\n")

    #end function

def generationError(): # generated computer response out of range  3 usages

    print(f"Error: computer response generation of of range\n")

    #end function

def unknownError(): # default exception  2 usages

    print(f"Unknown error.\n")

    #end function
```

game_flow.py:

This is where most of the code is housed. It uses what I refer to as arg chains instead of pointers, to ensure that any errors or exploits only affect that reference of the value and that process, rather than the original value and any related process.

```python
def game_round(player_answer, player_points: int = 0, cpu_points: int = 0, current_round: int = 1,  rounds_wanted: int = 1): #Start function  1 usage
                case "scissors": # Scissors vs Rock
                    defeat(player_points, cpu_points, current_round, rounds_wanted)

        case "paper": #on cpu answer paper
            match player_answer:
                case "rock": # Rock vs Paper
                    defeat(player_points, cpu_points, current_round, rounds_wanted)
                case "paper": # Paper vs Paper
                    draw(player_points, cpu_points, current_round, rounds_wanted)
                case "scissors": # Scissors vs Paper
                    victory(player_points, cpu_points, current_round, rounds_wanted)

        case "scissors": # on cpu answer scissors
            match player_answer:
                case "rock": # Rock vs Scissors
                    victory(player_points, cpu_points, current_round, rounds_wanted)
                case "paper": # Paper vs Scissors
                    defeat(player_points, cpu_points, current_round, rounds_wanted)
                case "scissors": # Scissors vs Scissors
                    draw(player_points, cpu_points, current_round, rounds_wanted)

        case _: # On gen error - verification escape
            generationError()


def flow(current_round, rounds_wanted, player_points: int = 0, cpu_points: int = 0): # start function  5 usages

    isValid = False

    if current_round <= rounds_wanted: # on game continue
        while not isValid:

            player_answer = input("ROCK - PAPER - SCISSORS! What do you choose? ").casefold()

            match player_answer:
                    case "rock" | "paper" | "scissors":
                        isValid = True
                        game_round(player_answer, player_points, cpu_points, current_round, rounds_wanted)
                    case _:
                        invalidInput()
```


```python
def game_round(player_answer, player_points: int = 0, cpu_points: int = 0, current_round: int = 1,  rounds_wanted: int = 1): #Start function  1 usage
                case "scissors": # Scissors vs Rock
                    defeat(player_points, cpu_points, current_round, rounds_wanted)

        case "paper": #on cpu answer paper
            match player_answer:
                case "rock": # Rock vs Paper
                    defeat(player_points, cpu_points, current_round, rounds_wanted)
                case "paper": # Paper vs Paper
                    draw(player_points, cpu_points, current_round, rounds_wanted)
                case "scissors": # Scissors vs Paper
                    victory(player_points, cpu_points, current_round, rounds_wanted)

        case "scissors": # on cpu answer scissors
            match player_answer:
                case "rock": # Rock vs Scissors
                    victory(player_points, cpu_points, current_round, rounds_wanted)
                case "paper": # Paper vs Scissors
                    defeat(player_points, cpu_points, current_round, rounds_wanted)
                case "scissors": # Scissors vs Scissors
                    draw(player_points, cpu_points, current_round, rounds_wanted)

        case _: # On gen error - verification escape
            generationError()


def flow(current_round, rounds_wanted, player_points: int = 0, cpu_points: int = 0): # start function  5 usages

    isValid = False

    if current_round <= rounds_wanted: # on game continue
        while not isValid:

            player_answer = input("ROCK - PAPER - SCISSORS! What do you choose? ").casefold()

            match player_answer:
                    case "rock" | "paper" | "scissors":
                        isValid = True
                        game_round(player_answer, player_points, cpu_points, current_round, rounds_wanted)
                    case _:
                        invalidInput()
```

```python
def flow(current_round, rounds_wanted, player_points: int = 0, cpu_points: int = 0): # start function  5 usages

    else: # on game over
        final(player_points, cpu_points, current_round, rounds_wanted)


def start(): # start function  3 usages

    isValid = False # verification bool

    current_round = 1

    while not isValid:  # start verification loop

        try: # on proper integer input

            user_input = input("Welcome to Rock Paper Scissors! How many rounds would you like to play? ")
            rounds_wanted = int(user_input)
            if rounds_wanted <= 0: # on integer 0 or less
                invalidInput()
            else: # on integer 1 or more
                isValid = True

        except ValueError: # on non integer input
            invalidInput()

        except: # on unknown failure
            unknownError()

    # end verification loop

    flow(current_round, rounds_wanted)
    #end function


def restart():  1 usage

    isValid = False

    while not isValid:
        player_restart = input("Do you want to play again?  ").casefold()
```

134:26 CRLF UTF-8 4 spaces Python 3.13 (Code)

```python
def restart():  1 usage
        match player_restart:

            case "yes":
                isValid = True
                print("restarting...")
                start()

            case "no":
                isValid = True
                print("Now closing. Have a great day!")
                sys.exit()

            case _:
                invalidInput()


def final(player_points, cpu_points, current_round, rounds_wanted):  #current round required for tie  2 usages
    print(f"The score is {player_points} to {cpu_points}")
    if player_points > cpu_points:
        print("The player wins!")
    elif player_points == cpu_points:
        tie_response = (input("It's a draw; do you want to go for a tie breaker?  ").casefold())

        match tie_response:

            case "yes":
                current_round -= 1
                flow(current_round, rounds_wanted, player_points, cpu_points)

            case "no":
                print("We'll leave it as a draw")

            case _:
                print("Error code 3: Unknown response. Please enter yes or no.")
                final(player_points, cpu_points, current_round, rounds_wanted)

    else:
        print("The computer wins!")
    restart()
```

134:26 CRLF UTF-8 4 spaces Python 3.13 (Code)

```
def final(player_points, cpu_points, current_round, rounds_wanted):  #current round required for tie  2 usages
        match tie_response:

            case "yes":
                current_round -= 1
                flow(current_round, rounds_wanted, player_points, cpu_points)

            case "no":
                print("We'll leave it as a draw")

            case _:
                print("Error code 3: Unknown response. Please enter yes or no.")
                final(player_points, cpu_points, current_round, rounds_wanted)

    else:
        print("The computer wins!")
    restart()


def victory(player_points, cpu_points, current_round, rounds_wanted):  3 usages
    print("You Won this round!")
    player_points += 1
    current_round += 1
    flow(current_round, rounds_wanted, player_points, cpu_points)


def defeat(player_points, cpu_points, current_round, rounds_wanted):  3 usages
    print("You lost this round")
    cpu_points += 1
    current_round += 1
    flow(current_round, rounds_wanted, player_points, cpu_points)


def draw(player_points, cpu_points, current_round, rounds_wanted):  3 usages
    print("It's a draw; the round will restart.")
    flow(current_round, rounds_wanted, player_points, cpu_points)
```

main.py:

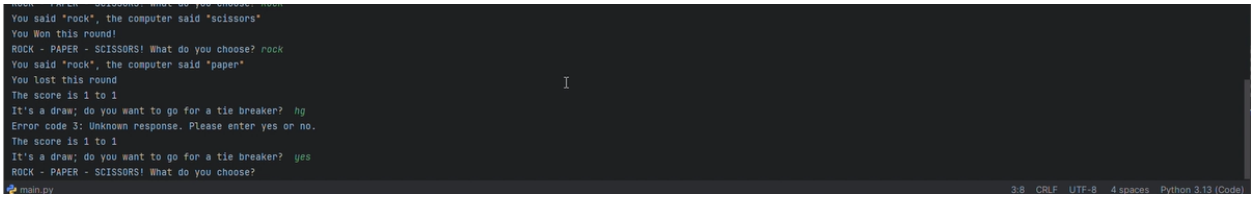
Finally, main.py runs the program using the start() function from game_flow.py.

```
from game_flow import start

start()
```

Final Result:

Here is the result:



Presentation: