Gerald Fleming

Project Title: Pokemon Types Game (Python)

1. Introduction

A Beginner Level Python project to outline the flow of a project as well as use of modules and functions. Pokemon battle simulator is based on real up-to-date strength and weakness charts from gaming companies, Nintendo's Pokemon franchise. We use the Random() module as well as multiple custom functions to make difficult tasks easier by breaking them into individual parts. Player selects a pokemon type and is placed against a computer in rock, paper, scissors like duel but with more choices and type rules and a coin flip special rule. Also have a battle record keeper to display Win/Loss data. README file included as well.

2. Design and Implementation

For the Pokemon Types game, I used a module(POKEMON_PKG) and two files, one file(pokemon_game.py) for the game and where the game is played from and another file(pokemon_effect.py) containing all used custom functions to make the code as clean and organized as possible. I wanted it to be a bit easy to understand at a glance but a bit more complex than Rock, Paper, Scissors since I have some prior experience in other languages. I worked to make sure that no matter what the play inputs the game will not crash whether it be invalid entry or capitalization using the String upper() method.

For the functions I made my life easier by creating a function for each type's win and loss circumstances and when to coin flip for the winner. I was going to try to implement a class but I didn't feel like it was needed as I had to clearly define the strengths and weaknesses for each function, so it would have just been 18 different methods versus 18 different functions.

```
# Bug

def poke_bug(mypoke, opppoke):
    strength = ["GRASS", "DARK", "PSYCHIC"]
    weakness = ["FIRE", "FLYING", "ROCK"]
    if opppoke in strength:
        result = poke_win(mypoke, opppoke)
    elif opppoke in weakness:
        result = poke_lose(mypoke, opppoke)
    else:
        result = poke_neutral(mypoke, opppoke)
    return result

# Dark


def poke_dark(mypoke, opppoke):
    strength = ["GHOST", "PSYCHIC"]
    weakness = ["BUG", "FAIRY", "FIGHTING"]
    if opppoke in strength:
        result = poke_win(mypoke, opppoke)
    elif opppoke in weakness:
        result = poke_lose(mypoke, opppoke)
    else:
        result = poke_neutral(mypoke, opppoke)
    return result

# Dragon - Special case Dragon would be neutral


def poke_dragon(mypoke, opppoke):
    strength = ["DRAGON"]
    weakness = ["DRAGON", "FAIRY", "ICE"]
    if opppoke in strength:
        result = poke_neutral(mypoke, opppoke)
    elif opppoke in weakness:
        result = poke_lose(mypoke, opppoke)
    else:
        result = poke_neutral(mypoke, opppoke)
    return result
```

Dragon had one strength which was also it's weakness so it did not have a clear 100% win scenario so poke_win() was not needed. I used an input analyzer to determine which type function to use with poke_battle(). I was grateful to be able to have the functions do an action and return the outcome data for later.

```
def poke_battle(mypoke, opppoke):

    if mypoke == "BUG":
        result = poke_bug(mypoke, opppoke)
    elif mypoke == "DARK":
        result = poke_dark(mypoke, opppoke)
    elif mypoke == "DRAGON":
        result = poke_dragon(mypoke, opppoke)
    elif mypoke == "ELECTRIC":
        result = poke_electric(mypoke, opppoke)
    elif mypoke == "FAIRY":
        result = poke_fairy(mypoke, opppoke)
    elif mypoke == "FIGHTING":
        result = poke_fighting(mypoke, opppoke)
    elif mypoke == "FIRE":
        result = poke_fire(mypoke, opppoke)
    elif mypoke == "FLYING":
        result = poke_flying(mypoke, opppoke)
    elif mypoke == "GHOST":
        result = poke_ghost(mypoke, opppoke)
    elif mypoke == "GRASS":
        result = poke_grass(mypoke, opppoke)
    elif mypoke == "GROUND":
        result = poke_ground(mypoke, opppoke)
    elif mypoke == "ICE":
        result = poke_ice(mypoke, opppoke)
    elif mypoke == "NORMAL":
        result = poke_normal(mypoke, opppoke)
    elif mypoke == "POISON":
        result = poke_poison(mypoke, opppoke)
    elif mypoke == "PSYCHIC":
        result = poke_psychic(mypoke, opppoke)
    elif mypoke == "ROCK":
        result = poke_rock(mypoke, opppoke)
    elif mypoke == "STEEL":
        result = poke_steel(mypoke, opppoke)
    elif mypoke == "WATER":
        result = poke_water(mypoke, opppoke)
    else:
        print("An Error has occured with ", opppoke)

    return result
```

I had 3 possible outcomes. poke_win(), a clear winner based on strengths, poke_lose(), a clear loser based on weaknesses, and poke_neutral() which enacts the coin flip using the built-in Random() function to determine winner or loser.

```
def poke_win(mypoke, opppoke):
    print("Your", mypoke, "type pokemon is strong against your opponent's",
          opppoke, "type Pokemon.\n")
    print("\nYOU WIN!!!!!!!\n")
    result = " WIN"
    return result


def poke_lose(mypoke, opppoke):
    print("Your", mypoke, "type pokemon is weak against your opponent's",
          opppoke, "type Pokemon.\n")
    print("\nYOU LOSE!!!!!!\n")
    result = "LOSE"
    return result


def poke_neutral(mypoke, opppoke):
    print(mypoke, "type and", opppoke,
          "type are neutral to each other. \nLet's flip a coin. Heads you WIN, Tails you LOSE...\n")
    coin = random.randint(1, 2)
    if coin == 1:
        print("That's heads...WOW Lucky day\nYOU WIN!!!!!!\n")
        result = " WIN"
    else:
        print("That's tails...BUMMER\nYOU LOSE!!!!!!\n")
        result = "LOSE"
    return result
```

The last item I added based on one of the workshops was a Battle data recorder and a function to display it that is visually appealing and letting players replay or quit when they needed to. Had to play around with the ASCII Art as it's not something I actively include in my projects but it came out beautifully and I got to learn about string padding with the String center() method.

```python
# String padding for display art
def string_padding(input):
    if len(input) == 8:
        return input
    else:
        return input.center(8)


# Battle data
def battle_da(function) string_padding(input: Any) -> Any
    mypoke = string_padding(mypoke)
    opppoke = string_padding(opppoke)

    line_string = "| " + mypoke + " |  " + opppoke + " | " + result + " |"
    battle_data.append(line_string)


def battle_data_print(battle_data):
    print("-------------------------------")
    print("| Your Pick|  Opp Pick| Result|")
    print("-------------------------------")
    for x in battle_data:
        print(x)
    print("-------------------------------")
```

3. Conclusions

I learned about python function use and the utility of designing modularly. I also learned through some self-experimentation how to use github, create a README.md file, and the markdown language. The best feature of the project is how easy a big task can be if you break it down to small functions to answer any trouble you come up with. The string_padding() function was 1000% useful because I wanted something I could copy paste and the separate file for the functions made the code so much cleaner.

If I had learned about the class object I would have used it throughout the project maybe but it would be purely to show I understood it , rather than improving the functionality of the code. If I could improve this project, I would make it a web app so it doesn't just run from the terminal and I would add visuals for the types to make the game more flashy.