Lab: Protocols

Part 1: Length protocol

Modify the player class so that it *implements* the length protocol w.r.t. the guess history for that player.

This means that it:

- defines required behaviour for things that can be asked for their length using the len() function

This means we should not be able to write the following code:

```
print('The length of the player history is', len(player1))
```

If you try this at the moment you should get an error message similar to:

```
Traceback (most recent call last):
  File "/number_guess_game.py", line 101, in <module>
    play_game()
  File "/number_guess_game.py", line 87, in play_game
    print('The length of the player history is',
len(player))
TypeError: object of type 'Player' has no len()
```

The length protocol states that to a type must implement one method:

- __len__() returns the length of the type

Add this method to the definition of the class Player. To calculate the length of a player use the size of the history.

You should now be able tun run the following code:

```
player1 = Player('John')
print(player1)

player1.increment_count()
player1.add_guess(4)

player1.increment_count()
player1.add_guess(2)

print(player1)
print(len(player1))
```

The output from this is:

```
Player John - history [] - guess_count 0
Player John - history [4, 2] - guess_count 2
2
```

Note that the above code just uses len(player) to generate the length fo the players history.

Part 2: Iterable and Iterator protocols (Extension Point)

We will now modify the Player class to implement the Iterable and Iterator protocols.

- We will assume that when you iterate over a player – you want to iterate over the history list, Thus we only need to implement the __iter__() method and return the iter object associated with the history list, for example, we can add this to the Player class:

```
# Makes this class Iterable
def __iter__(self):
    return self.history.__iter__()
```

Note the above is a method so make sure the indentation is correct when you add it to your class.

Now modify the number guess game code such that you use the *iterable* player directly to print out the series of guesses the player made. This is in addition to anything you currently have and is mostly to prove what you have done, for example add:

We should now be able to run the following:

```
player1 = Player('John')
print(player1)

player1.increment_count()
player1.add_guess(4)

player1.increment_count()
player1.add_guess(2)

print(player1)
print(len(player1))

for guess in player1:
    print(guess)
```

The output from this is:

```
Player John - history [] - guess_count 0
Player John - history [4, 2] - guess_count 2
2
4
2
```