Rock Paper Scissors

▾ python_code_2.py 7

```
# install colorama using pip
# install termcolor using pip
# [INFORMATION] [python_code_2.py]--------
# Player 1 : Human player,
# Player 2 : Random move,
# Player 3 : Mimic Human player's previous move,
# Player 4 : Cycles through 'rock, paper, scissors'.

```

AWESOME

Code quality check

Good job making sure `pycodestyle` returns no errors when checking the quality of your code. A tool such as this is es a team so everyone adheres to the same rules and all code is clean and consistent. Depending on who you work with different, but generally speaking `pycodestyle` is a great tool to use.

```
import random
import time
from colorama import init
```

AWESOME

A beautify game experience

You have made an excellent and beautiful presentation of the game in the console with the `colorama` library 🙂. Well

```
init()

moves = ['rock', 'paper', 'scissors']


# function to print text with a delay
def print_pause(message):
    print(message)
    time.sleep(1)


# Parent class
class Player:
```

AWESOME

Player Strategies

Very well done implementing all the player strategies 🖐⬆.

```
    def move(self):
        return RandomPlayer.random_move(self)
```

```
    # 'Learn function', to memorize opponent's move and return the same
    def learn(self, my_move, their_move):
        self.my_move = my_move
        self.their_move = their_move
        return self.their_move


# 'Human class', which allows the users to play the game
class HumanPlayer(Player):
    def __init__(self):
        self.human_move = ""

    def move(self):
        # return input from the user
        self.human_move = input("rock, paper, scissors ? > ").lower()
        # loops itself in case player inputs unrecognized text/move
        while self.human_move not in moves:
```

AWESOME

Validation

Well done handling the cases where the user enters an invalid move 👍.

```
            self.human_move = input("rock, paper, scissors ? > ").lower()
        return self.human_move


# Reflect class, which mimics the opponent's move in the next round
class ReflectPlayer(Player):
    def move(self):
        # returns the "HumanPlayer's" move from previous session
        return game.p1.my_move


# 'CyclePlayer' class, which cycles items from the 'moves' list
class CyclePlayer(Player):
    def __init__(self):
        self.round = 0

    def move(self):
        # Round 1 -> returns a random move
        while self.round == 0:
            self.round += 1
            return RandomPlayer.random_move(self)
        # Round 2 returns the adjacent/next item from the 'moves' list
        try:
            return moves[moves.index(self.my_move) + 1]
        # If index exceeds the length of the list,
        # It circles back to its fist item
        except IndexError:
            return moves[0]


# 'RandomPlayer' class, which returns a random move from the 'moves' list
class RandomPlayer(Player):
    def random_move(self):
        return random.choice(["rock", "paper", "scissors"])


# 'Constant player' class, which returns a constant move -> 'rock'
class ConstantPlayer(Player):
    def move(self):
        return 'rock'

    def learn(self, my_move, their_move):
        pass


# 'Game class', which decides the length of the game, announce winner
# and controls game flow
class Game():
    def __init__(self, p1, p2, p3, p4):
        self.p1 = p1
        self.p2 = p2
        self.p3 = p3
        self.p4 = p4
        self.p1.name = "Player 1"
        self.p2.name = "Player 2"
        self.p3.name = "Player 3"
        self.p4.name = "Player 4"
        # player 1 session wins count
        self.p1.win = 0
        # player 1 round wins count
        self.p1.won = 0
        self.p2.win = 0
        self.p2.won = 0
        self.p3.win = 0
```

```
109         self.p3.won = 0
110         self.p4.win = 0
111         self.p4.won = 0
112         self.round = 0
113         self.game = "initialized"
114 
115     # function to announce the winner scenario
116     def announce_winner(self):
117         # A list of lists with player's name and player's win count
118         self.list = [[self.p1.name, self.p1.won],
119                      [self.p2.name, self.p2.won],
120                      [self.p3.name, self.p3.won],
121                      [self.p4.name, self.p4.won]]
122         self.winner = "none"
123         self.wins = 0
124         self.wins_count = 0
125         print_pause(f"\nTotal Score\t: {self.p1.name} - {self.p1.won}, "
```

AWESOME

String Interpolation

Good job using `f-string` interpolation to format your output 🖐.

You can learn more about other python interpolation methods from this article:
https://www.programiz.com/python-programming/string-interpolation

```
126                      f"{self.p2.name} - {self.p2.won}, "
127                      f"{self.p3.name} - {self.p3.won}, "
128                      f"{self.p4.name} - {self.p4.won} ")
129         #  a loop to find the biggest number/ most wins in the list
130         for self.sublists_1 in self.list[:1]:
131             self.wins = self.sublists_1[1]
132             self.winner = self.sublists_1[0]
133             for self.sublists_2 in self.list[1:]:
134                 if self.sublists_2[1] > self.wins:
135                     self.wins = self.sublists_2[1]
136                     self.winner = self.sublists_2[0]
137         # loop to find the tie scenario
138         for self.sublists_1 in self.list:
139             if self.sublists_1[1] == self.wins:
140                 self.wins_count += 1
```

AWESOME

Increment shorthand operator

Nice work using the shorthand increment operator, that makes your code looks elegant, concise, and familiar to fellov

```
141         # condition to deal with tie scenario
142         if self.wins_count > 1:
143             # blink's the string 4 times
144             self.blink(f"Game Result     : ** This game ended in a Tie b/w "
145                        "players ** ", 4)
146         else:
147             # blink's the string 4 times
148             self.blink(f"Game Result     : ** {self.winner} wins the "
149                        "Game ** ", 4)
150 
151     # 'beats' function, which returns a boolean vaule using game rules
152     def beats(self, one, two):
153         return ((one == 'rock' and two == 'scissors') or
154                 (one == 'scissors' and two == 'paper') or
155                 (one == 'paper' and two == 'rock'))
156 
157     def play_round(self):
158         print_pause("\033[0;30;41m[Session 1]------------\033[1;31;40m")
159         self.winner1 = self.play_sub_round(self.p1, self.p2)
160         # loop to deal with the 'tie' scenario
161         while self.winner1 == "tie":
162             self.winner1 = self.play_sub_round(self.p1, self.p2)
163         print_pause("\033[0;30;46m[Session 2]------------\033[1;36;40m")
164         self.winner2 = self.play_sub_round(self.p3, self.p4)
165         # loop to deal with the 'tie' scenario
166         while self.winner2 == "tie":
167             self.winner2 = self.play_sub_round(self.p3, self.p4)
168         print_pause("\033[0;30;43m[Session 3]------------\033[1;33;40m")
169         self.winner3 = self.play_sub_round(self.winner1, self.winner2)
170         # loop to deal with the 'tie' scenario
171         while self.winner3 == "tie":
172             self.winner3 = self.play_sub_round(self.winner1, self.winner2)
173         # conditional statements to diaplay the round winner
174         if self.winner3 == self.p1:
```

```
            self.p1.won += 1
            # blink's the string 4 times
            self.blink(f"\033[1;32;40mRound Result\t: ** {self.p1.name} "
                       "wins the round **", 4)
        elif self.winner3 == self.p2:
            self.p2.won += 1
            self.blink(f"\033[1;32;40mRound Result\t: ** {self.p2.name} "
                       "wins the round **", 4)
        elif self.winner3 == self.p3:
            self.p3.won += 1
            self.blink(f"\033[1;32;40mRound Result\t: ** {self.p3.name} "
                       "wins the round **", 4)
        elif self.winner3 == self.p4:
            self.p4.won += 1
            self.blink(f"\033[1;32;40mRound Result\t: ** {self.p4.name} "
                       "wins the round **", 4)
        # prints individual wins of all 4 players in the round
        print_pause(f"\n\033[1;34;40mIndividual "
                    f"wins\t: {self.p1.name} - {self.p1.win}, "
                    f"{self.p2.name} - {self.p2.win}, "
                    f"{self.p3.name} - {self.p3.win}, "
                    f"{self.p4.name} - {self.p4.win}")
        # prints total/round wins of all 4 players in the game
        print_pause(f"\033[1;34;40mTotal Score\t: {self.p1.name} -"
                    f" {self.p1.won}, "
                    f"{self.p2.name} - {self.p2.won}, "
                    f"{self.p3.name} - {self.p3.won}, "
                    f"{self.p4.name} - {self.p4.won}\033[1;37;40m")

    def play_sub_round(self, c1, c2):
        c1.session_win = 0
        c2.session_win = 0
        # calls the first player move
        move1 = c1.move()
        # calls the second player move
        move2 = c2.move()
        # prints both players moves
        print_pause(f"{c1.name} played\t: {move1}  \n{c2.name} Played"
                    f" : {move2}")
        # Learn opponent's move
        c1.learn(move1, move2)
        c2.learn(move2, move1)
        # condition to determine first Player's victory scenario
        if self.beats(move1, move2):
            # increment individual wins
            c1.win += 1
            # increment session's wins
            c1.session_win += 1
            self.blink(f"Play_off Result\t: ** {c1.name} Wins **", 4)
            # statement to display score every session
            print_pause(f"Session Score \t: {c1.name} "
                        f"- {c1.session_win}, {c2.name} - {c2.session_win}")
            # reset session's score to zero
            c1.session_win = 0
            return c1
        # condition to determine 'game-tie' scenario
        elif move1 == move2:
            self.blink("Play_off Result\t: ** Game Tie **", 4)
            print_pause(f"Session Score \t: {c1.name} "
                        f"- {c1.session_win}, {c2.name} - {c2.session_win}")
            return "tie"
        # condition to determine second Player's victory scenario
        else:
            c2.win += 1
            c2.session_win += 1
            self.blink(f"Play_off Result\t: ** {c2.name} Wins **", 4)
            print_pause(f"Session Score \t: {c1.name} "
                        f"- {c1.session_win}, {c2.name} - {c2.session_win}")
            c2.session_win = 0
            return c2

    def play_game(self):
```

AWESOME

Multi-round match implementation

Good job implementing a complete match game.

```
        self.spin(" GAME START ", 4)
        # Case: if player does not want to quit the game
        while self.game != "quit" and self.game != "no":
            # increment round count and display it
            self.round += 1
            print_pause(f"\n\033[0;30;47m-----------[ ROUND {self.round} ]-"
                        "----------\033[0;37;40m")
            # play another round
            self.play_round()
```

```
            self.game = input("\nPlay again? Type 'play' or 'quit' > ").lower()
            # Condition to handle unrecognized input on 'self.game'
            while (self.game != "play" and self.game != "yes") and \
                  (self.game != "quit" and self.game != "no"):
                self.game = input("Play again? Type"
                                  " 'play' or 'quit' > ").lower()
        # function method to announce winner
        self.announce_winner()
        print_pause("")
        self.spin(" GAME OVER ", 4)

    # function which limits game play to one round
    def play_game_once(self):
        self.spin(" GAME START ", 4)
        self.play_round()
        print_pause("")
        self.spin(" GAME OVER ", 4)

    # function to blink the text
    def blink(self, string, num):
        self.blank_list = []
        for letter in string:
            self.blank_list.append(" ")
            self.blank_string = "".join(self.blank_list)
        for _ in range(num):
            self.clear = "\b" * (len(string))
            print(string, end='', flush=True)
            time.sleep(0.2)
            print(self.clear, end='', flush=True)
            print(self.blank_string, end='', flush=True)
            time.sleep(0.2)
            print(self.clear, end='', flush=True)
        print(string)

    # funtion to display the text between spinning lines
    def spin(self, string, num):
        self.clear = "\b"*(4 + len(string))
        for _ in range(num):
            for ch in '-\\|/':
                print(ch + ch + string + ch + ch, end='', flush=True)
                time.sleep(0.1)
                print(self.clear, end='', flush=True)

    # funtion to display information related to the game
    def intro(self):
        print_pause("\n\033[0;30;45m[INFORMATION]--------\033[1;35;40m")
        print_pause("Player 1 : Yourself")
        print_pause("Player 2 : Random move")
        print_pause("Player 3 : Mimic your previous move")
        print_pause("Player 4 : Cycles through 'rock, paper, scissors'\n")
        print_pause("Each round has 3 sessions")
        print_pause("Session 1 : Play_off b/w Player 1 and Player 2")
        print_pause("Session 2 : Play_off b/w Player 3 and Player 4")
        print_pause("Session 3 : Play_off b/w session 1 and session 2"
                    " winners\033[0;37;40m\n")


# condition to run the code only if executed directly
if __name__ == '__main__':
    game = Game(HumanPlayer(), Player(), ReflectPlayer(), CyclePlayer())
    game.intro()
    game.play_game()

```

▸ python_code.py

▸ README.md

▸ Note to reviewer.txt

RETURN TO PATH