Author: Casey Bellew
GitHub username: Psykibear
Date: 5/30/2023
Description: Halfway Progress Report - Writing a class called Othello that allows two people to play text-based Othello

1. Initializing the Player and Othello classes

```
Class Player:
    def __init__(self, name, color)
        self._name = name
        self._color = color
```

Sets up the player class with the initializing calling for the user to have a player defined by their name and color to be used within the game. The game will be calling for the player to be the color black or white.

```
Class Othello:
    def __init__(self)
        self._board = [['*' for marker in range(10)] for marker in range(10)]
        self._board[4][4] = 'O'
        self._board[4][5] = 'X'
        self._board[5][5] = 'O'
        self._board[5][4] = 'X'
        self._players = []
```

Sets up the Othello class for the game to be started and a board to be setup with the initial lists to have a defined asterisk in each of the tile positions except the center of the board. The center of the board will have 'O' or 'X' based on the pictures given in the images from the downloaded file setting up the programing assignment.

2. Determining how to implement create_player method.

```
    def create_player(self, player_name, color):
        player = Player(player_name, color)
        self._players.append(player)
```

The create player method takes the list setup during the initializing method of the Othello class. It then calls for the Player class with the information player_name and color they specified. It then appends the list with the player information provided when the user provides the information as such:

```
    game.create_player("Helen", "white")
```

The game has a requirement that the colors being chosen are white or black so that when a tile is "flipped" it will change to the specified color. The program assumes the player will only choose either white or black in color. Black will be denoted by a "X" on the board while a white will be denoted by "O" when the tile "flips".

3. Determining how to implement print_board method.

Printing the board needs to go row by row through each column. For loop is where I'm aiming to run through each row with a nested for loop for the columns to be printed. The initial values are based off the initialization method of the Othello class. These board tiles do not change until the make_move method is called by the user for a specified location.

```
def print_board(self):
    for row in range(1,9):
        for column in range(1,9):
            print(self._board[row][column], end = " ")
        print()
```

4. Determining how to implement return_available_positions method.

During the game we need to be able to provide the user with the available positions when an invalid move has been made. The invalid move needs to be called/checked during this method so ideally, you'd setup the invalid move method first to check for available positions. These positions would then be housed in a list that would be printed when an invalid move is made by the user. For the position to be a valid position it must be a position that has not already been moved/flipped on. So, since the starting value of the positions is "*" I am using that as a check method. If the value of the column of each row does not equal "*" then it can be considered a valid position. If the valid move returns false then it will provide the play_game method to know whether to print out "Invalid move" "here are the available moves:".

```
def _is_valid_move(self, row, column, color):
    if self._board[row][column] != '*':
        return FALSE
```

Like the print_board method I'm taking the approach of checking each row with a for loop followed by an additional nested for loop for the columns within each row. This will check for the color or "string" at each row/column position to determine if it is available to be used for a make_move. These positions that pass the available position test will be stored in a list that will be printed with the invalid move message.

```
def return_available_positions(self, color):
    available_positions = []
    for row in range(1,9):
        for column in range(1,9):
            if self._board[row][column] == '*':
                available_positions.append((row, column))
    return available_positions
```

5. Determining how to implement return_winner method.

Based on the Othello game from the Wikipedia we need to verify how many either black or white tiles have been moved/flipped. So, like previous methods I have setup the counter variables for black and white tiles that would go through nested for loops to count how many tiles have been flipped to either an “X” or an “O”. Then for each it would increase the counter until completing the check of all board positions. This would be followed up by performing a check of the counters to verify if one is greater than the other determining the winner. If one is not greater than the other, then the program will assume that it is a tie. The print method has to grab the player name that matches the winning color using player for player in self._players if player.color == “winning color”.

```
def return_winner(self):
        black_counter = 0
        white_counter = 0
        for row in range(1,9):
                for column in range(1,9):
                        if self._board[row][column] == ‘X’
                                black_counter +=1
                        elif self._board[row][column == ‘O’
                                white_counter +=1

                if black_counter > white_counter:
                        winner = [player for player in self._players if player.color == “black”[0]
                        return print(“Winner is black player: “ + winner.name)
                elif white_counter > black_counter:
                        winner = [player for player in self._players if player.color == “white”[0]
                        return print(“Winner is white player: “ + winner.name)
                else:
                        return print(“It’s a tie.”)
```

6. Determining how to implement make_move method.

For a player to make a move, the user must input the color specified to be moved/flipped in a specified position. Based on the README, the input values from the user include the position which will be put in in the following format: (row, column). In this method it does not check for valid position because based on putting it in the play_game method it will call for the validate move then. The play_game method will use and call for both the make_move and validate_move methods.

```
def make_move(self, color, piece_position)
        row, column = piece_position
        self._board[row][column] = color[].upper()
```

I’ve added the upper() to make sure it uses the uppercase and lowercase values of x and o for updating the values within the specified spots reqested by the user playing the game.

7. Determining how to implement play_game method; how to validate a move. Determine how to detect the end of the game.

When the user calls for the play_game method it gives the user the ability to make a move based on their input while calling the make_move method, then calls the _is_valid_move method to verify that the user's move is applicable to the available moves. If the move is invalid, it will let the user know that while printing out the available moves the user can attempt next. This will then update the board and print it out. Once a valid move has been made it will then call for an end_of_game method which checks for the board to be filled without an available space to be played by the users. Once this has been triggered then it will run the return_winner method to display which user won the game overall.

```
def play_game(self, player_color, piece_position):
        if not self._is_valid_move(piece_position(row), piece_position(column), player_color):
                print("Invalid move.")
                print("Here are the valid moves: " + self._available_positions(player_color))

        self.make_move(player_color, piece_position)
        self.print_board()

        if self._is_game_ended():
                print("Game has ended.")
                print(self.return._winner())
```