Week 2 Homework: Exercises 1-5

Exercise 1

1.0/1.0 point (graded)

For our tic-tac-toe board, we will use a numpy array with dimension 3 by 3.

Write a function `create_board()` that creates such a board with the value of each cell set to the integer 0.

Call `create_board()` and store it.

What is the correct numpy function to initialize our tic-tac-toe board?

`np.empty((3,3), dtype=int)`

`np.zeros((3,3), dtype=int)`
correct

`np.zeros_like(3, dtype=int)`

`np.full((3,3), dtype=int)`

Code =[

```
import numpy as np

def create_board():
    board = np.zeros((3,3), dtype=int)
    return board
]
```

Exercise 2

1/1 point (graded)

Players 1 and 2 will take turns changing values of this array from a 0 to a 1 or 2, indicating the number of the player who places a marker there.

Create a function `place(board, player, position)`, where:

- `player` is the current player (an integer 1 or 2).

- `position` is a tuple of length 2 specifying a desired location to place their marker.

Your function should only allow the current player to place a marker on the board (change the board position to their number) if that position is empty (zero).

Use `create_board()` to store a board as `board`, and use `place` to have Player 1 place a marker on location (0, 0).

What is the correct way to use the `place` function to have Player 1 place a marker on location (0,0)?

```
place((0,0), 1, board)
```

```
place(board, "Player 1", (0,0))
```

```
place(board, 1, 0)
```

```
place(board, 1, (0,0))
```

Correct

Code = [

```
def place(board, player, position):
    if board[position] == 0:
        board[position] = player
        return board

board = create_board()
place(board, 1, (0, 0))
array([[1, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
]
```

Exercise 4

1.0/1.0 point (graded)

The next step is for the current player to place a marker among the available positions. In this exercise, we will select an available board position at random and place a marker there.

Write a function `random_place(board, player)` that places a marker for the current player at random among all the available positions (those currently set to 0).

Find possible placements with `possibilities(board)`.

Select one possible placement at random using `random.choice(selection)`.

Note that `board` is already defined as at the end of Exercise 2. Call `random_place(board, player)` to place a random marker for Player 2, and store this as board to update its value.

Use this code to get you started:

```
import random
random.seed(1)

# write your code here!
```

Enter the first number. Enter the number only.
Answer = [1]
Enter the Second number. Enter the number only.
Answer = [0]

Code = [

```
random.seed(1)

def random_place(board, player):
    selections = possibilities(board)
    if len(selections) > 0:
        selection = random.choice(selections)
        place(board, player, selection)
    return board

random_place(board, 2)
array([[1, 0, 0],
       [2, 0, 0],
       [0, 0, 0]])
]
```

Exercise 5

0/1 point (graded)

We will now have both players place three markers each.

A new board is given by the sample code.
Call `random_place(board, player)` to place three pieces each on board for players 1 and 2.

Print `board` to see your result.

Start with this sample code:

```
random.seed(1)
board = create_board()

# write your code here!
```

At what positions does player 1 have markers after three rounds of random placement?

```
(0,3), (1,1), (2,1)
```

```
(0,2), (1,1), (2,1)
Correct
(0,0), (0,1), (2,2)
```

```
(0,2), (2,1), (2,2)
```

```
(0,0), (1,1), (2,1)
```

```
Code =[
random.seed(1)
board = create_board()

for i in range(3):
    for player in [1, 2]:
        random_place(board, player)

print(board)
[[2 2 1]
 [0 1 0]
 [0 1 2]]
]
```

Week 2 Homework: Exercises 6-9

Exercise 6

1/1 point (graded)

In exercises 6 through 9, we will make functions that check whether either player has won the game.

Make a function `row_win(board, player)` that takes the player (integer) and determines if any row consists of only their marker.

Have it return `True` if this condition is met and `False` otherwise.

Note that `board` is already defined as in Exercise 5. Call `row_win` to check if Player 1 has a complete row.

Does Player 1 have a complete row?

Yes

No
Correct

Code = [

```
def row_win(board, player):
    if np.any(np.all(board==player, axis=1)): # this checks if any row contains all positions equal to player.
        return True
    else:
        return False

row_win(board, 1)
False

]
```

Exercise 7

1/1 point (graded)

In exercises 6 through 9, we will make functions that check whether either player has won the game.

Make a function `col_win(board, player)` that takes the player (integer) and determines if any column consists of only their marker.

Have it return `True` if this condition is met and `False` otherwise.

Note that `board` is already defined as in Exercise 5. Call `col_win` to check if Player 1 has a complete column.

Does Player 1 have a complete column?

Yes

No

Correct

Code = [

```
def col_win(board, player):
    if np.any(np.all(board==player, axis=0)): # this checks if any column contains all positions equal to player
        return True
    else:
        return False

col_win(board, 1)
False
]
```

Exercise 8

0/1 point (graded)

In exercises 6 through 9, we will make functions that check whether either player has won the game.

Finally, create a function `diag_win(board, player)` that takes the player (integer) and determines if any diagonal consists of only their marker.

Have it return `True` if this condition is met
and `False` otherwise.

Note that `board` is modified from Exercise 5.
Call `diag_win` to check if Player 2 has a complete diagonal.

Use this sample code to get started:

```
board[1,1] = 2
```

Does Player 2 have a complete diagonal?

Yes
Correct

No

Code = [

```
def diag_win(board, player):
    if np.all(np.diag(board)==player) or
np.all(np.diag(np.fliplr(board))==player):
        # np.diag returns the diagonal of the array
        # np.fliplr rearranges columns in reverse order
        return True
    else:
        return False

diag_win(board, 2)
True
]
```

Exercise 9

1/1 point (graded)

In exercises 6 through 9, we will make functions that check whether either player has won the game.

Create a function `evaluate(board)` that uses `row_win`, `col_win`, and `diag_win` functions for both players. If one of them has won, return that player's number. If the board is full but no one has won, return -1. Otherwise, return 0.

Note that `board` is defined as in Exercise 8.
Call `evaluate` to see if either player has won the game yet.

Use this sample code to get started:

```
def evaluate(board):
    winner = 0
    for player in [1, 2]:
        # add your code here!
        pass
    if np.all(board != 0) and winner == 0:
```

```
        winner = -1
    return winner
```

Has anyone won the game yet?

Yes, Player 1

Yes, Player 2
Correct

No

Code = [

```
def evaluate(board):
    winner = 0
    for player in [1, 2]:
        if row_win(board, player) or col_win(board, player) or diag_win(board,
player):
            winner = player
    if np.all(board != 0) and winner == 0:
        winner = -1
    return winner

evaluate(board)
2
]
```

Week 2 Homework: Exercises 10-11

Exercise 10

1/1 point (graded)

In this exercise, we will use all the functions we have written to simulate an entire game.

The functions `create_board()`, `random_place(board, player`

`)`, and `evaluate(board)` are all defined as in previous exercises.

Create a function `play_game()` that:

- Creates a board.

- Alternates taking turns between two players (beginning with Player 1), placing a marker during each turn.

- Evaluates the board for a winner after each placement.

- Continues the game until one player wins (returning 1 or 2 to reflect the winning player), or the game is a draw (returning -1).

Call `play_game` 1000 times, and store the results of the game in a list called `results`. Use `random.seed(1)` so we can check your answer!

How many times does Player 1 win out of 1000 games?

Answer = [591]

Code = [

```
random.seed(1)

def play_game():
    board = create_board()
    winner = 0
    while winner == 0:
        for player in [1, 2]:
            random_place(board, player)
            winner = evaluate(board)
            if winner != 0:
                break
    return winner

results = [play_game() for i in range(1000)]
results.count(1)
591
]
```

Exercise 11

1/1 point (graded)

In the previous exercise, we saw that when guessing at random, it's better to go first, as one would expect. Let's see if Player 1 can improve their strategy.

Create a function `play_strategic_game()`, where Player 1 always starts with the middle square, and otherwise both players place their markers randomly.

Call `play_strategic_game` 1000 times.

Set the seed to 1 using `random.seed(1)` again.

How many times does Player 1 win out of 1000 games with this new strategy?

Answer = [716]

Code = [

```
random.seed(1)

def play_strategic_game():
    board, winner = create_board(), 0
    board[1,1] = 1
    while winner == 0:
        for player in [2,1]:
            random_place(board, player)
            winner = evaluate(board)
            if winner != 0:
                break
    return winner

results = [play_strategic_game() for i in range(1000)]
results.count(1)
716
]
```