We want to show the user what the game looks like before they start and then after they play. Before they start, the board is going to be empty, so instead of showing them empty spaces, we would show them numbers correlating to the index of the space. To do this we define a method inside our Tic-Tac-Toe class like so.

def print\_board\_nums():

        # 0 | 1 | 2 etc (tells us what number corresponds to what box)

        number\_board = [[str(i) for i in range(j\*3, (j+1)\*3)] for j in range(3)]

        for row in number\_board:

            print("| " + " | ".join(row) + " |")

The goal here is to make a list that would contain three lists, so we can say for each row in the list print the row. To get this list, we say for j in range(3) i.e. 0, 1, 2, and then for i in range(j \* 3, (j + 1)\*3), convert the i to a string and put in the list. So for example, in the first instance of the j for loop, j is zero (0) so our i for loop becomes for i in range (0, 3). At the end of the day, we would have a list containing three lists and then we can loop through it to print out the values. <br />

After each player plays, we want to print out the board with the updated play. To do this, we are also going to create a list of lists but this time, we are going to do for i in range(3), give me a sliced version of our board which would return an array with the appropriate values.

def print\_board(self):

        #this is just getting the rows

        for row in [self.board[i\*3:(i+1)\*3] for i in range(3)]:

            print("| " + " | ".join(row) + " |")

The next thing we want to do is to check if there are empty squares, if there are then return true, else return false. This would be useful for when we play the game.

def empty\_squares(self):

        return " " in self.board #this would return a boolean of whether there is a space or not

<code>” “ in self.board</code> is checking whether there is an empty space in the board and if there is it would return true so we are passing that to the return statement so the method either returns true or false. We can also define a method to tell us what the number of empty squares currently are.

def num\_empty\_squares(self):

        #the two lines of code below do the same thing

        # return len(self.available\_moves)

        return self.board.count(" ") #this would count the number of spaces in the board because board is a list.

At this point the Tic-Tac-Toe class looks like this.

class TicTacToe:

    def \_\_init\_\_(self):

        self.board = [" " for \_ in range(9)] #use a single list to replicate 3x3 board.

        self.current\_winner = None #keep track of winner

    def print\_board(self):

        #this is just getting the rows

        for row in [self.board[i\*3:(i+1)\*3] for i in range(3)]:

            print("| " + " | ".join(row) + " |")

    @staticmethod

    def print\_board\_nums():

        # 0 | 1 | 2 etc (tells us what number corresponds to what box)

        number\_board = [[str(i) for i in range(j\*3, (j+1)\*3)] for j in range(3)]

        for row in number\_board:

            print("| " + " | ".join(row) + " |")

    def available\_moves(self):

        #the line below is how it would look using list comprehension

        return [i for i, spot in enumerate(self.board) if spot == " "]

        #the code below does the exact same thing

        # moves = []

        # for (i, spot) in enumerate(self.board):

        #     #["x", "x", "o"] --> [(0, "x"), (1, "x"), (2, "o")]

        #     if spot == " ":

        #         moves.append(i)

        # return moves

    def empty\_squares(self):

        return " " in self.board #this would return a boolean of whether there is a space or not

    def num\_empty\_squares(self):

        #the two lines of code below do the same thing

        # return len(self.available\_moves)

        return self.board.count(" ") #this would count the number of spaces in the board because board is a list.

    def make\_move(self, square, letter):

        #if valid move, then make the move (assign square to letter)

        #then return true. if invalid, return false

        if self.board[square] == " ":

            self.board[square] = letter

            if self.winner(square, letter):

                self.current\_winner = letter

            return True

        return False

    def winner(self, square, letter):

        # winner if 3 in a row anywhere (row, column or diagonal)

        # first check the row

        row\_ind = square // 3 #divide whatever square is by 3 and then round it down so its always going to be either 0, 1 or 2

        row = self.board[row\_ind \* 3 : (row\_ind + 1) \* 3]

        if all([spot == letter for spot in row]): #returns true if everything is true else returns false

            return True

        #check the column next

        col\_ind = square % 3

        column = [self.board[col\_ind+i\*3] for i in range(3)]

        if all([spot == letter for spot in column]):

            return True

        #check diagonals

        #but only if the square is an even number (0, 2, 4, 6, 8)

        #these are the only moves possible to win a diagonal

        if square % 2 == 0: #if it's an even number

            diagonal1 = [self.board[i] for i in [0, 4, 8]] #left to right diagonal

            if all([spot == letter for spot in diagonal1]):

                return True

            diagonal2 = [self.board[i] for i in [2, 4, 6]] #right to left diagonal

            if all([spot == letter for spot in diagonal2]):

                return True

        # if all of these fail

        return False