# Tic Tac Toe Mini Project

*55 pts, Due May 21*

## Contents



## Classes and Objects:

This project is designed to give you some familiarity with the concept of designing and using classes and objects.

Object Oriented Programming is the concept that in the real world we think in terms of objects and generalized concepts of objects. We all know what a car is – cars all have some general characteristics in common (4 wheels, a steering wheel, brakes, etc.) and cars do certain things – drive, stop, turn, etc. That is a class definition of a car – it's general properties and its functions. In computer science,that concept of grouping properties and functions together into one definition is what would be known as a class definition. Then your car, the car down the street, that strange purple car with the one headlight out- each individual car is an object of type car.

In programming so far, when we think of ints. Ints can be multiplied, added, subtracted, compared, etc. Those are all functions that are associated with ints. Ints are a primitive type, and just hold a value.

A somewhat more complex type that we've seen are strings. Strings hold as values a set of characters. They also have functions associated with them – string.index(), string.lower(), string.count(), string.find(), etc. All these are functions associated with strings specifically.

**NOTE:** when functions are defined inside of a class definition, they are known as ***methods.***

So now you are being introduced to the ability to write your own types. A lot of what you're doing in this lab is quite similar to what you did for Minesweeper (on an extremely reduced level!) – the big difference is that now we're writing a class definition for the game TicTacToe. In other words, and this sounds weird, but it's really what we're doing – we're making TicTacToe a type, and we're then writing all the methods associated with the type TicTacToe.

In Minesweeper the functions makeBoard, printBoard, addBombs, checkwin, etc. would all be considered methods if we'd made the game Minesweeper a class. Equally, boardSize, mat, mat2, and for the GUI squaresize, etc. would all be properties of the class Minesweeper.

Below, in the class definition of TicTacToe, the methods printBoard, boardFull, personPlay, playGame, and checkWin are all methods that are part of the TicTacToe class. Equally, the TicTacToe class has properties like the board (a 3x3 matrix), p1, and p2.

## Some clues for writing the methods below:

- Everything defined within the class can be accessed by anything else in the class.

- We did not make minesweeper a class, and thus we had to make mat and mat2 global variables. By making TicTacToe a class, with properties that include the board, now all the methods inside the TicTacToe class can just use the board variable, and if one method makes changes to it, the other methods will be accessing the changed method.

- You don't have to pass the properties from method to method within a class definition. See above

- The constructor (__init__) usually gives initial values to the properties belonging to the class. It may also call the methods that set up the class object.
  - You NEVER explicitly call __init__. It happens automatically when you say:
    g=TicTacToe(board,'A','B')

  - That line there makes the __init__ constructor happen.

- Self: Techically self is a self-referential pointer. It is a way of referencing an object when it is in the process of being created.
  - For now, from a pragmatic standpoint, just put it as the first parameter in each method inside a class because that is how all methods can access all the properties belonging to the class.
  - Once you've put it as the first parameter, ignore it.
  - In other words, the board matrix goes into the b parameter, 'A' goes into the p1 parameter, and 'B' goes into the p2 parameter.

***Okay, give it a shot! Welcome to object oriented programming!!!***

```
class TicTacToe(object):
    def __init__(self,b,p1,p2): #5 pts
        ##   constructor: initializes a field called board to b, player1
        ##   to p1, and player2 to p2 (player1 and player2 are just single
        ##   characters that will be placed on the board, so normally would
        ##   be X and O

    def  printBoard(self): #4 pts
        ##   prints out the tic tac toe board as it stands so far, so the first
        ##   time it will look like this:
        ##      _  _          _
        ##      _  _          _
        ##      _  _          _
        ##   After the game has been played for a few turns, if this function is
        ##   it might print out something like this:
        ##      _  _          O
        ##      _  X          _
        ##      X  _          O
        ##   (This method just prints out the board, it doesn't put the x's and o's
        ##    in the board)

    def boardFull(self): #5 pts
        ##      checks every space on the entire board to see whether there
        ##      are any spaces that are still '_'.  If so, it returns False,
        ##      otherwise it returns True (boolean true and false, not the words)

    def personPlay(self,c):
        print(c+"'s turn:")
        x = int(input(c+": Enter x: "))
        y = int(input(c+": Enter y: "))
        while (not self.boardFull() and self.board[y][x] != '_'):
```

```
        x = int(input(c+": Enter x: "))
        y = int(input(c+": Enter y: "))
    self.board[y][x] = c
    self.printBoard()
```

## def playGame(self):

```
    win = False
    winner = '_'
    pchar = self.player1
    turn = 0
    while not win and not self.boardFull():
        self.personPlay(pchar)
        win = self.checkwin(pchar)

        if win:
            winner = pchar
        elif not self.boardFull():
            turn +=1
            if turn%2 == 1:
                pchar = self.player2
            else:
                pchar = self.player1
    if winner == '_':
        print("It's a tie")
    else:
        print(winner + " won")
    return
```

## def checkwin(self,c):  #8 pts

```
        ##   this function (method) checks the tic tac toe board to see if the character
        ##   (held by the parameter c) is present 3 in a row on the board (diagonally,
        ##   up and down, or left to right.  If the character held by the parameter c
        ##   is found 3 in a row, the method returns True. Otherwise the method returns
        ##   False.
```

## def __str__(self):  #8 pts

```
        ##   returns a string representing the tic tac toe board as it stands so far, so the first
        ##   time the returned string will hold this:
        ##      _  _          _
        ##      _  _          _
        ##      _  _          _
        ##   After the game has been played for a few turns, if this function is
        ##   it might return a string that looks something like this:
        ##      _  _          O
        ##      _  X          _
        ##      X  _          O
        ##   (This method returns a string representing the board – it will be used in print
        ##  commands)
```

```
board = [['_','_','_'], ['_','_','_'],['_','_','_']]
g=TicTacToe(board,'A','B')
g.playGame()
```

#20 pts Getting everything to work:

#5 pts for surviving this crazy semester!!!