Name:______________________________ **Partner:** ______________________________

Python Activity 41: Tic Tac Toe - Game

It's best to think through your program design prior to coding!

Learning Objectives
Students will be able to:
Content:
- Describe how `TTTcube` objects are implemented
- Summarize how to test newly developed code *in isolation* and why we might do so
- Explain the game logic for Tic-Tac-Toe in computational terms
- Consider common cases and **edge cases** in a given problem/solution

Process:
- Write code that implements `TTTcube` objects
- Write code that connects all our `TTT` classes together with game logic

Prior Knowledge
- Python concepts: user-defined classes, inheritance, tic-tac-toe

Critical Thinking Questions:

1. *Follow along in the class lecture*, and match the following methods/concepts on the left to their purpose (or output) on the right:

`TTTcube`	Ex: `'X'`
`TTTcube()`	Creates a new tic-tac-toe cube instance with `'-'`
`ttt_cb.get_letter()`	Public method to change value of calling cube object
`ttt_cb.set_letter(s)`	Public method to return str of calling cube object
`str(ttt_cb)`	`Cube` is its parent class

2. *Follow along in class lecture,* and fill out the *Class Object Models* below for `Cube` and `TTTcube`: (*Hint:* Be sure to include each method's return types, and the names of any parameters!)

`class Cube`	`class TTTcube`
Attributes:	Attributes:
Methods:	Methods:

3. a. Why might it be a good idea to test our class & methods in *isolation*? How might we do that for the `TTTcube` class?

b. Write a few lines of code to create a new `TTTcube`, print one of the letters, and change the value of a letter:

4. Consider the *game logic* for Tic-Tac-Toe.

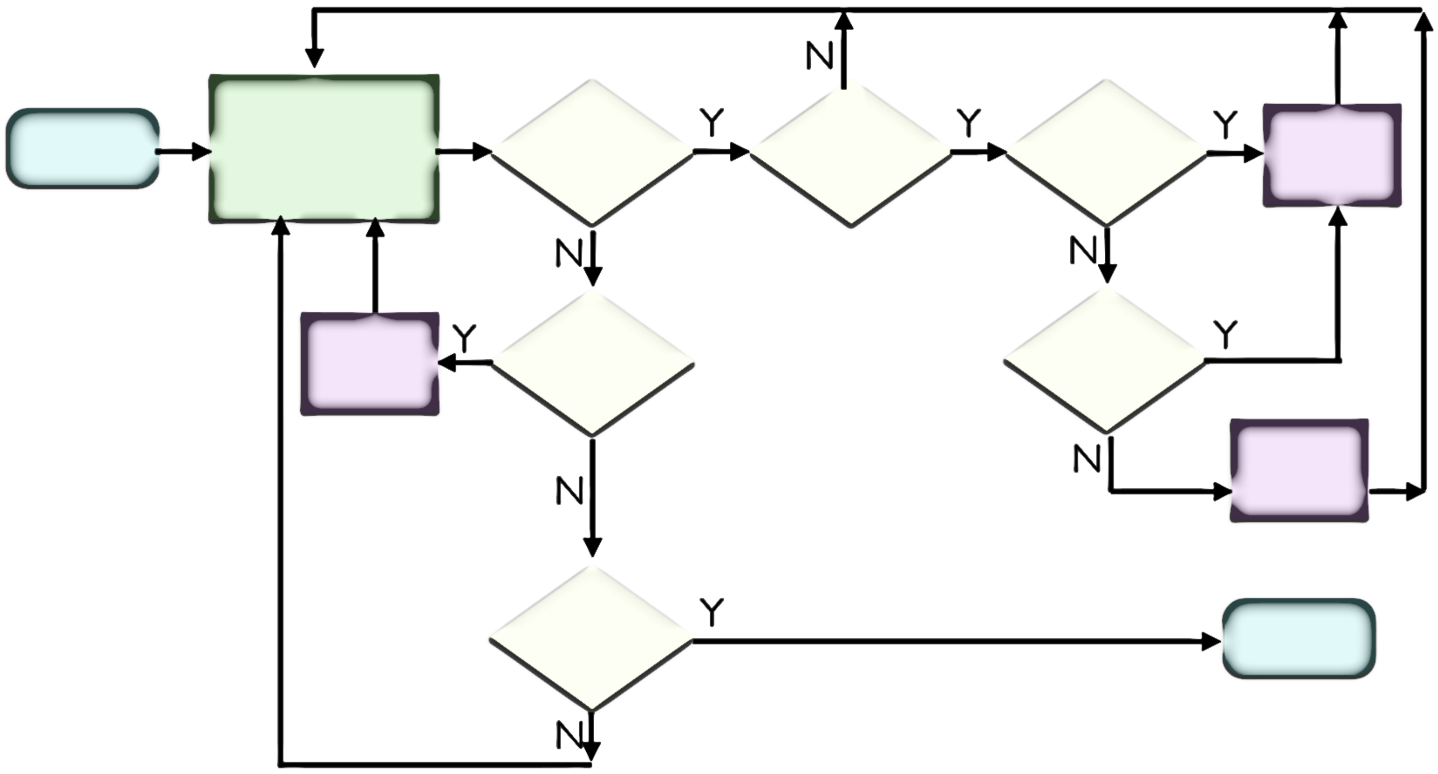
a. When designing a computational solution, it's a good idea to consider a *common case* initially. What is an example of a common case in our Tic-Tac-Toe game?

b. What might be examples of *less common cases* in our game?

c. What must the game do when the user types `'r'` or `'q'` when asked for input?

d. When designing a computational solution, it's also a good idea to increase the robustness of our solution by handling *edge cases* (unexpected, or very uncommon cases). What might be an *edge case* in our Tic-Tac-Toe game?

5. *Follow along in the class lecture* and fill in the following tic-tac-toe game logic decision map:



Application Questions: Use Python to check your work

(The Boggle Lab is a really good application of these concepts!)