Week 2: Introduction to Python (Part 2)

Introduction

In this worksheet we will use our functions from the previous week to build our Tic-TacToe game! If you did not complete the exercises last week, then you can find a solution in this zip folder named 'sheet1solution.py'.
A couple of Python tips and tricks have been given, these may help you with producing your solutions. If you are unsure of how to get Python up and running then please refer to worksheet 1!

Python Tips & Tricks

1. You can concatenate strings using '+', i.e 'A' + 'B' = 'AB'
2. As easy way to check if an element is in a list is by using 'in', i.e '5 in [1,4,5,7] = True'
3. You can use the above to iterate through a loop, e.g:

```
    List = [3,4]
    for Element in List:
        print(Element)

```

```
    3
    4
```

Getting Started: Tic-Tac-Toe

Hopefully you are aware of the simple Tic-Tac-Toe game as this is what you will be building from scratch within this worksheet! If not, it is a simple 2 player game where whichever player ('X' or 'O') can get 3 in a row first.

If you open the 'sheet2.py' file within VSCode (or your chosen IDE) you will see a template of code for a game of TicTacToe. A class has been created for you, with the properties and constructor already created for you - your job is to create the methods/functions that will be needed for game play. The class and properties will be initialised for you when a new version of the class is created, i.e:

```
Game = TicTacToe()
```

Properties

The properties and how they are initialised are explained below:

1. 'self.Board' is where the current state/board of the game will be stored. It is initialised as a 1x9 list of spaces - when a turn is played the corresponding space will be replaced by a 'X' or a 'O'. This is explained in more detail further on.
2. 'self.EmptyCells' is a list which contains the indices of the cells of the board which are empty. As the board is initialised as empty, this is initialised as a list containing 0-8.
3. 'self.Turn' is a string which represents whose turn it currently is (either 'X' or 'O'). It is initialised as 'X' so player X will play first.
4. 'self.Winner' will store a string of the winner once the game has ended (either 'X', 'O' or 'Tie'). It is initialised as None as the game has not been won yet.
5. 'self.StatesX' and 'self.StatesO' will store a list of the state of the board after player X or O (respectively) have had their turn. This will be explained in more detail later on.

The Board

While the board is stored as a list of length 9 (this can be seen in the properties), the board should be displayed as 2-dimensional.

If you run the 'sheet2.py' file you have been given you will see an empty board displayed (and not much more). A function has already been provided for you (__str__()) which inputs the board (in list form) and will print it to the terminal for you to see. If you wish to print the board inside the class definition you can use:

```
print(self)
```

As an example, a board as a list and how it is printed to the terminal can be seen below:

```
self.Board = [' ',' ','X',' ','O',' ',' ',' ','X']
print(self)
```

```
- - - - - - -
|   |   | X |
- - - - - - -
|   | O |   |
- - - - - - -
|   |   | X |
- - - - - - -
```

To convert to and from list/board format, the cells in the board have been given the following numbering:

0	1	2
3	4	5
6	7	8

Tasks

The following tasks are to start building the game. You are welcome to complete these all together with one person screen sharing or separately. Don't forget to add examples to your cheat sheet as you go along!

A lot of this work has already been done for you when you wrote the functions in the previous worksheet. In line1 of the template you will notice the following code:

```
import sheet1 as s1
```

This is done so we can use our previous functions inside our new code without having to rewrite them. To call on a function within this separate file we can use the call 's1.FunctionName()'.

Copy and paste your previous work into the SAME folder as the 'sheet2.py' file (make sure it has the name of 'sheet1.py'). If you did not complete sheet1, then just change the name of 'sheet1solution.py' to 'sheet1.py' and then everything should work for you!

Task 1: RandomTurn

Complete the 'RandomTurn(self)' function. This function will look at which cells in the board are currently empty and choose a random position to fill in with either an 'X' or 'O', updating the 'self.Board' property. Instead of adding whose turn it is as an input to the function, don't forget you can use 'self.Turn' inside any function to get which player's turn it is!

Hint: you can re-use previous functions.

You can uncomment the testing call for 'Task1' at the bottom of code to check your function works (it may help to temporarily add print statements inside the function to check that the function is doing what you expect it to!).

Note that all functions inside a class definitions have to have 'self' as the first input, however you are welcome to add more inputs after if you wish!

Task 2: HumanTurn

Complete the 'HumanTurn(self)' function - some of this function has been done for you. This function asks the user (you) to input a number into the terminal and then, if that is a valid number and the cell is empty, it will place a marker on the board for you.

Steps already completed for you:

1. Ask the user (you) to input a number
2. Check that they have input a number between 0 and 8
3. If so, save the number as a variable named 'Position'
4. If not, ask again

Steps for you to complete:

1. If the position is empty, place a marker in that cell (update self.Board)
2. If not:
    (a) Print a statement to the user saying that the cell is already filled
    (b) Recursively call self.HumanTurn so that the user is asked for another input

Task 3: Update

Complete the function 'Update()'. This function will be called after every time a move has been played. This function should do the following things:

1. Update the 'EmptyCells' property to match the current board (note that self.Board will be updated in other functions) - Hint: sheet1.

2. Depending on whose turn it just was, save the current board into either the StatesX or StatesO property. If player X has just made a move (so 'self.Turn' should still be set to player X), then the board is added to the StatesX list and vice versa for player O. Instead of saving the board as a list, we will convert it to a singular string (Hint: sheet1). Hence, StatesX/StatesO will be lists of strings where each string has length 9 (containing spaces, X's and O's).

3. Swap turns (either from 'X' to 'O' or vice versa) - be careful using if/elif statements here as you do not want to accidentally swap turns twice.

4. Check to see if the game is over.

A function 'IsGameOver' has been already done for you. This updates the self.Winner property to either 'X' or 'O', or updates it to 'Tie' if the board gets full.

An example of the before and after of the properties (when a random move is taken) can be seen below.

```
BEFORE
Board:  [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ']
Empty Cells:  [0, 1, 2, 3, 4, 5, 6, 7, 8]
Turn:  X
Winner:  None
StatesX:  []
StatesO:  []

AFTER
Board:  [' ', ' ', ' ', 'X', ' ', ' ', ' ', ' ', ' ']
Empty Cells:  [0, 1, 2, 4, 5, 6, 7, 8]
Turn:  O
Winner:  None
StatesX:  ['    X      ']
StatesO:  []
```

You can uncomment the testing at the bottom of the template code to test your function yourself (make sure you recomment this after you are done however).

Task 4: HumanVsRandom

Here we will build the main function for our game play - 'HumanVsRandom'. The function will continually swap between calling 'HumanTurn' and 'RandomTurn' until the game has been won, which will create a program in which we play against the computer. We will play as player X and the computer will play as player O.

DISCUSSION Point: What kind of programming technique should we use so that the game is continually in play until somebody has won?

For each turn the function should:

1. Print the board

2. Print whose turn it currently is

3. Call either HumanTurn() or RandomTurn()
4. Update the game

After the game has ended, the function should:

1. Print the board
2. Print out the winner

EXTRA: IsGameOver

If you have finished all the tasks above and are wanting to test your Python skills, try implementing your own 'IsGameOver' function (without looking at the one given to you!).

This function should check all the rows, columns and diagonals checking to see if a player has won and update self.Winner correspondingly. After that, it should also check to see if the board is full (in case of a tie).

Discussion

Play around with your completed game - why is it so easy to win?

How could we improve upon our code to make it harder?

What are some benefits of Python that you have seen in this worksheet that you might not get in other languages?

Next Week

In the next worksheet we will work on improving our computer so that it makes smarter game play choices.

Topic Suggestions

If you have any topics which you would like covered in these sessions then feel free to let me know by clicking here.
