**Outline**

Access the Python Development environment and continue the tutorial to gain an additional exposure to the Python programming language. Begin to develop an familiarity with intermediate programming concepts.

**Objectives**

* Use correct terminology to describe programming concepts;
* Describe the types of data that computers can process and store (e.g., numbers, text);
* Explain the difference between constants and variables used in programming;
* Use variables, expressions, and assignment statements to store and manipulate numbers and text in a program

**Materials**

* Python3 Development Environment at: //repl.it/
* Python Tutorial at: <http://www.letslearnpython.com/learn/>

**Accessing the Tutorial**

Accessing the Tutorial

* Go to: <http://www.letslearnpython.com/learn/>
* Read up to “Lesson 12: Input”

**Level 1: Input & Output**

1. Read through “Lesson 12: Input – What Is Input?” and “Lesson 12: Input – Example” and “Lesson 12: Input – Shortcut”.
2. Type the following code into the white area of the IDE and run the program. Explain what you see in the black area of the IDE.

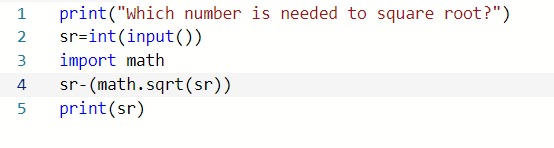
print("Type your name:")

name = input()

print("Hi", name, "how are you?")

Whatever the term that you input, it will output as”Hi” input” How are you”. If you input a name like Bob, it will output “Hi John, how are you”.

1. Create a short program that reads numerical input from the console and does the following:
   1. Uses the input() function to read a numerical value from the console.
   2. Calculates the square root of the number
   3. Prints the result to the console output
   4. Provides appropriate prompt and message strings to go with the input and output.
   5. Provide your complete program below.



**Level 2: Tic-Tac-Toe Game**

1. Write a Python program to play a game of Toc-Tac-Toe. (You may modify a program that you found on-line to meet the expectations of this module.)
   1. The program may be either player v. computer or player 1 v. player 2.
   2. The program does not need to determine a winner
   3. The program just needs to keep track of moves and spaces in the game board
2. import random
3. #GamePlay Board
4. Board = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
5. #3x3 board Positions 1-9
6. def showBoard():
7. print(Board[1], "│", Board[2], "│", Board[3])
8. print("━━━━━━━━━━")
9. print(Board[4], "│", Board[5], "│", Board[6])
10. print("━━━━━━━━━━")
11. print(Board[7], "│", Board[8], "│", Board[9])
12. #Checks each line for winning combination
13. def checkLine(player, spot1, spot2, spot3):
14. if Board[spot1] == player and Board[spot2] == player and Board[
15. spot3] == player:
16. return True
17. #Checks all posible combinations for winner
18. def checkWinner(player):
19. if checkLine(player, 1, 2, 3):
20. return True
21. if checkLine(player, 4, 5, 6):
22. return True
23. if checkLine(player, 7, 8, 9):
24. return True
25. if checkLine(player, 1, 4, 7):
26. return True
27. if checkLine(player, 2, 5, 8):
28. return True
29. if checkLine(player, 3, 6, 9):
30. return True
31. if checkLine(player, 1, 5, 9):
32. return True
33. if checkLine(player, 3, 5, 7):
34. return True
35. #Game function for moves and gameplay
36. while True:
37. showBoard()
38. #user input (1-9) User= X
39. X = input("Choose a spot (1-9):")
40. X = int(X)
41. if Board[X] != 'O' and Board[X] != 'X':
42. Board[X] = 'X'
43. else:
44. print("This spot is already being used by the computer, sorry :(")
45. #Checks if X won the game or not
46. if checkWinner('X') == True:
47. print("!!You have managed to beat the Computer, Congratulations!!")
48. break;
49. #Checks if game is DRAW
50. if checkWinner('O') == False and checkWinner('X') == False:
51. print('!!DRAW!!')
52. break;
53. #comuter Move Computer= O
54. while True:
55. O = random.randint(1, 9)
56. if Board[O] != 'O' and Board[O] != 'X':
57. Board[O] = 'O'
58. break;
59. #Checks if X won the game or not
60. if checkWinner('O') == True:
61. print('!!Pathetic, the Computer has beated you, You Lose!!')
62. break;

Provide a complete listing of your program.

* 1. Your listing **MUST** include line numbers .

3. Explain how your program keeps track of the game board.   
(Provide specific code references by line number.)

* 1. What python types and data structures are used?
  2. How are moves by player X and player O recorded?
  3. How are free spaces recorded?

My program keeps track of my game mode is by having the list titled “Board” which is located on line 4. The list begun with indexes which begins from 0-9. However, I used the indexes 1-9 for my tic tac toe board. I have done this by making a function called “showBoard” in which it places the indexes 1-3 on the top row, 4-6 on the middle row and 7-9 on the bottom row in order to make the area for where the player or the computer place their slots. This is located on line (7-12). Nextly, I have also made dark lines which will pass through the board to help formulate it to become the proper tic tac toe board. I had many vertical and horizontal lines to separate each of the rows and columns. These lines are shown as strings and to do this, I had to use the print command print() which is in the showBoard function. The player controls the X term in tic tac toe. For it to move, the human whom is playing X will have to move where ever they desire as long as it’s between (1-9). These indexes will be replaced by X. (Look at lines 42-47). The computer controls the O term in Tic Tac Toe. For it to move, the number is generated by a random number generator which means that the O which is controlled by the computer will move completely random. (Lines 62-65). If the spaces are already taken by the indexes X or O it will display a message in which you cannot put it in that particular spot. (Lines 45-48 and 63-65). The spots which are open will display regular numbers and the tic tac toe board is updated with every single computer or player move.

4.Explain how moves and commands are input from the console.  
(Provide specific code references by line number.)

* 1. How does the player tell the program about the move location (row, column)?
  2. How does the program verify that the move location is valid?
  3. How does the program verify that the space is free?
  4. What does the program do if there is something wrong with the move?

The player who is (X) is given the choice of placing their index from 1-9 on the grid. Once the player chooses the number, the number is then changed into an integer with the int(input)(). This can be identified as an index of the Board. The number gets checked only if the number is a valid move. Due to this, the program checks if the index is either X or O and if the index is not selected the automatic selected index will automatically become X using the “if” term. If there is a number or letter which is invalid it will display a message in which it tells the player that they should put a valid number. (Line 44-48). I used “continue;” in order to let the player try again if they accidently typed an invalid number or letter (Line 49). This means that the program will continue to work even if the player has put an invalid number.

5. Explain how the program keeps track of gameplay.  
(Provide specific code references by line number.)

* 1. How does the program switch between player X and player O moves?
  2. How does the program keep asking for moves?
  3. How does the program decide when to stop asking for moves?

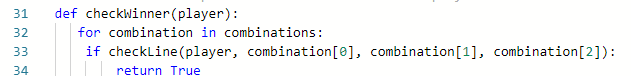
The player switches between both player X and player O which means the loop will always become true unless it ends with the “break;” command. This means that the moves will continue between both player X and player O until one of the players eventually wins the game. Once the loop, both the player X is asked to move and the player O (computer) moves and this cycle will repeat until one of these players eventually gets 3 in a row. The game automatically stops when the board is completely full due to their being no moves left or when there is a clear winner. The winner combination is located at (line 56-75). The many combinations that the player could place can be located at (lines 22-38). After a winner is determined, the loop will stop due to the “break” and if the player (who is X) will receive this indication, they will earn the indication of “("!!You have managed to beat the Computer, Congratulations!!") (Line 86). If the computer (O) wins then the player will receive this message, '!!Pathetic, the Computer has beated you, You Lose!!”. (Line 105-this may change due to change of code). The entire program will end either if the player loses, wins or if there are no moves left. In order to keep track of the gameplay for Tic Tac Toe, the function “showBoard” is placed in the loop. (Line 71).

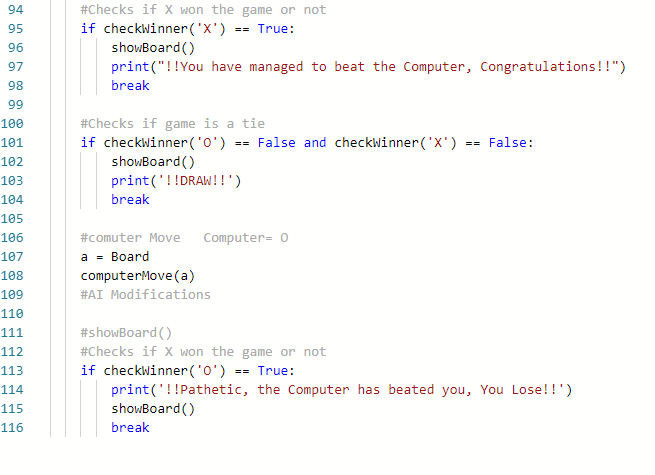
**Level 3: Basic Enhancements**

1. Explain, in plain words, a strategy for determining if player “x” or player “O” has won the game after a move is made.

The strategy that was being made for determining if player “X” and player “O” has won the game after a move is made is by having combinations. These combinations mean that if the player or computer will have 3 integers in a row, they will win the game. (Line 31-34) If the player has won the game, they will earn the “!!You have managed to beat the Computer, Congratulations!!”. (Line 93-98). To determine the draw, I had to implement the code “If checkWinner (‘O’) == False and checkWinner (‘X’) == False” in order to determine a draw. This code means that if both of the players will have not been able to get 3 in a row and filled up the board, it will lead to a draw. (Line 101-104). If the algorithm of the computer has beated you, that means you will receive this message “Pathetic, The Computer has beated you, You Lose!”.

1. Provide a function called “checkWinForX” that returns the Boolean value of “True” if player “x” won the game.
2. Modify your program to check and print a message, and stop the game of player “x” or player “O” wins the game.





1. Demonstrate your enhanced game to Mr. Nestor for credit for this level.

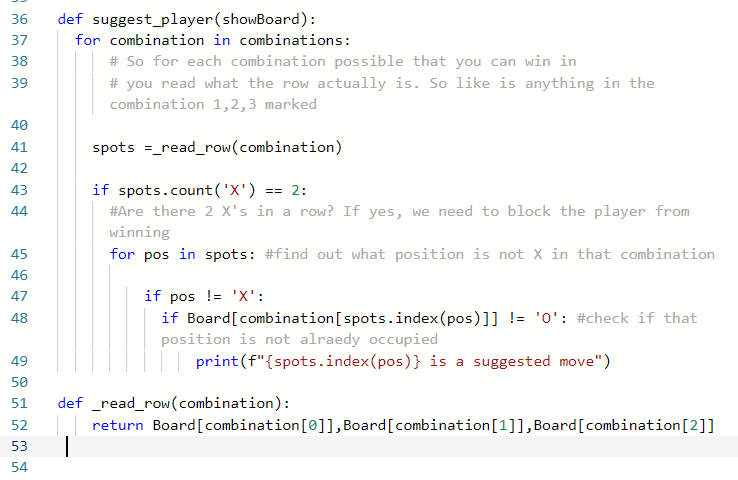
**Level 4: AI Enhancements**

1. Explain, in plain words, a strategy for suggesting the best move for player “x” or player “O” to make when it is their turn to move.

To suggest the best move for player X is by implementing spots =\_read\_row(combination). This combination suggests that the player should move their sign to this particular spot. If X has filled atleast 2 spots, a message would appear saying “(integer) is a suggested move”. This code can be possible thanks to the code located at (Lines 47-49). This code with “If Board [combination[spots.index(pos)]]!= ‘0’ helps to bring the suggestions to life as I can print the index of suggesting a move depending on which the player puts X in 2 particular spots.

However, the system that I have implemented does not suggest the right number placement. With more coding and potential editing. Once this is accomplished, the suggesting system will work more effectively and suggest the right areas instead of already occupied ones.

1. Create a function to implement your strategy for suggesting the best move.
2. Modify your program to print a suggested move when it is each player’s turn to move.
3. Demonstrate your AI enhanced game to Mr. Nestor for credit for this level.



**Code for Smarter AI**

To make a smarter AI, I had to change the suggestion material and replace each of the combinations with simpler code. In order to bring more of a challenging player, I had to implement cleaned\_list = ['X' if 'X' == s else 'O' if 'O' == s else ' ' for s in board] and cleaned\_list.pop(0). This is a way to clear out each of the spots which will be taken from either player X or O. If the player manages to get 2 spots in a row, the AI will then begin to be aggressive in which it will begin to block off player X. On the updated AI code, Lines 56-69, there will be a code named Board[combination[spots.index(pos)]] = 'O'. This code is the code to stop X (the player) from winning. This brings more challenge in which X has to think of other combinations if they get blocked off by O. In lines 73-76, this is the code which activates in which the AI or O player does not have to block X as it did not fill 2 spots in a row. This means O will place its index on another random spot.

