**Outline**

Develop a functioning Tic Tac Toe game using basic Python concepts. Extend the game to verify user input and incorporate modularity. Use random number generator to implement a basic AI.

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

**Level 1: Initial Board Output and User Input**

The game board may be implemented may be implemented using the Python List construct. A single row may be represented as follows.

# Definitions of the Game Board Setup

x\_mark = " X "

o\_mark = " O "

blank = " "

# Setup and display of a single row

boardRow = [blank, x\_mark, o\_mark]

print (boardRow)

1. Create a three row representation of the game board by using / modifying the above code.   
   (Hint: Think about lists of lists.)

User input may be collected using the Python code as follows.

# Note: Numbers must be converted using the int() function

Print (“Make a move…”)

xORy = input("X or Y =”)

rowMove = int(input("Row = "))

colMove = int(input("Col = "))

# The move can be added to a row as follows:

boardRow [colMove] = xORy

print (boardRow)

1. Extend your program to collect user input and add the move to your three row game board.   
   Use / modify the sample code.
2. Define (in words) the types of user input that would be an invalid move. List each input example as a separate point.
3. Extend your program to add code that checks for valid user input. Use / modify the sample code provided below.

# If statements can be used to check that the range of input values are correct

if (colMove > 2 ) :

print (“Column value must be between 0 to 2. Please try again “)

if ((xORy != x\_mark ) or (xORy != y\_mark)) :

print (“Mark must be either X or Y. Please try again “)

**Level 2: Looping For 9 Moves**

The use of for loops in Python is illustrated by the following code:   
(Note: Proper indentation is IMPORTANT!)

x\_mark = " X "

o\_mark = " O "

blank = " "

# A Row on the board is a list of three spaces

# The Game board is a list of lists!

# i.e. The Board is a list of three rows.

boardRow = [blank, blank, blank]

boardGame = [boardRow,boardRow,boardRow]

# Printing the Game Board by looping for the three rows

for row in range(0,3) :

print(boardGame[row])

print(" ")

# Filling the Game Board with X's by using nested loops

for row in range(0,3) :

for col in range(0,3) :

boardGame[row][col] = x\_mark

# Printing the Game Board by looping for the three rows

for row in range(0,3) :

print(boardGame[row])

print(" ")

# A simple for loop that loops 9 times

for move in range(1,10) :

print("move = ", move)

1. Extend your program from Level 1 to loop over 9 game moves (i.e. a maximum length game).   
   Your loop should include the following elements:
   * NOTE: Your loop does not have to check for winning conditions
   * You should alternate between player X and player O moves
   * You should collect move input form the keyboard for each player
   * Print a copy of the game board after each player move
2. Extend your code to add checks and re-tries for invalid player moves. Use the checks you developed in Level 1 and add the following elements:
   * Use a loop to tell the player to “Try Again” until they get the move right. (Hint: Find out how “while” loops with condition checks work.)
   * Implement these checks as sub-loops within your main loop (question 1 loop).

**Level 3: Using Functions**

The use of Functions in Python is important for grouping a series of statements that perform a specific task or operation that will be repeated many times in your main program. The use of a function is illustrated by the following code: (Note: Proper indentation is IMPORTANT!)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# This Function checks to make sure that the user

# move is valid.

#

# Input Parameters:

# row - row of the move

# col - column of the move

#

# Output Result:

# true - if the move is good

# false - if the move is bad

#

def isMoveValid(row, col) :

if ((row < 0) or (row > 2)) :

# row value is out of range

return False

if ((col < 0) or (col > 2)) :

# column value is out of range

return False

# If all the above checks are ok then the move is good

return True

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#

# Main Program. Collect Usewr Input

# and check that the move is valid

#

rowMove = int(input("Row = "))

colMove = int(input("Col = "))

if (isMoveValid(rowMove, colMove) == False) :

print ("The move is NOT valid. Try Again!")

else :

print ("Good Work! The move is valid.")

1. Extend your program from Level 2 use functions to group frequently used code. You should have functions for the following actions:
   * Collecting user move information
   * Checking user move information
   * Adding moves to your game board