**Python Concepts**

**Importing additional classes:**

To allow for more complex code work without the chore of writing all the algorithms from scratch, we can import additional premade libraries and use their functions to simplify certain tasks. Python comes with several useful libraries built into it while outside developers have created additional classes. In this example we will use a standard library included with Python however to use outside classes you just need to include the .py file in either your working directory or the project directory. To see a list of standard libraries included with python go here: <https://docs.python.org/3/library/>

**To Import:**

To import, use the keyword *import* followed by the class you want to import.

Example:

import random

The above line will import the class library random. This will not be used in our walk-through example however to generate a random number between 0-8 will allow you to generate a random number to place and X or 0 for the computer’s turn in the challenge assignment.

**Functions:**

Functions are just blocks of reusable code. To define a function we use the keyword *def* followed by the function name and parenthesis and then a semicolon. If we need to use data inside of the function we can pass in variables which are known as parameters for the function.

Example:

def main():

The above line defines a function and any code inside of it (tabbed) will then be called every time the function is called. To call this function you would simply add the following code wherever you would like to call it:

main()

**Lists:**

Lists are simply a more complex version of a variable. In fact they consist of many different values. For instance, looking at a single month on a calendar you will notice many squares each representing a different day, but they are all part of the month. Each day on a calendar can have notes associated with it but at a bare minimum would have a number representing the day of the month that square represents. While a calendar starts on day 1 a List actually starts in position 0. If we break this idea down to a week instead of a month.

Declare an DList with values example:

daysOfWeek = [‘Sunday’, ‘Monday’, ’Tuesday’, ’Wednesday’, ’Thursday’, ’Friday’, ’Saturday’]

To then call a specific spot of the array we simply reference it with a numerical value representing the position of the spot in the array we are trying to print/set/use:

print(daysOfWeek[0])

This would print to the screen:

Sunday

**Activity: TIC-TAC-TOE**

In this activity we will design and build a simple game in python called Tic-Tac-Toe. You should be familiar with the idea of this game but just in case, we will quickly summarize it here. This game will consist of a 3 by 3 grid creating a total of 9 squares. The game will either be 1-player with a random computer choice happening each round or a 2-player game where each side will take a turn placing an X or a O. When either player (or computer) has 3 of its marks in a row, the game will end, and the winner will be displayed.

To build this game we will use four (4) different functions to separate out the game concepts. Everything will be then run from with a main function which will contain our main game loop. Notes will be designated as red comments inside the code block below explaining each step.

The main function:

def main():

#Board consists of 9 squares, this will be done all within a single dictionary names boardState. To begin, the values will all be empty strings

boardState = [‘’, ‘’, ‘’, ‘’, ‘’, ‘’, ‘’, ‘’, ‘’]

#next we need a Boolean variable set to True until the end of the game is reached.

playGame = True

#next we need to set the number of players, this allows up to modify the code later if we want to allow for the computer to pay one of the turns.

numPlayers = 2

#to have a computer option you can use a line like the one below to prompt the user to decide how many players

#prompt the player

#main game While Loop.

while(playGame):

#run function to display the board at the beginning of each turn. pass in the boardState Dictionary

displayBoard(boardState)

#player turn should also set Boolean value for playGame

playGame = playerTurn(boardState, 1, ‘X’)

#if the player did not win, game should be played and check to see who player 2 is.

If(playGame):

# If numPlayers = 3 then Computer, otherwise its player 2 turns of course at this point it is only player2.

if(numPlayers == 2):

player = 2

else:

player = 3

#then call the playerTurn function again.

playGame = playerTurn(boardState, player, ‘O’)

#once game is done display the board state one last time

displayBoard(boardState)

#game over

print(“Game Over”)

With the main function completed, you can see 2 additional functions were called from within the main function. These functions are playerTurn() and displayBoard().

The next function we will build will be the displayBoard() function which just as its name states, will be responsible for printing the board visually to the screen after each round showing the all the X’s and O’s that have been placed. This function requires a parameter to be passed to it containing the dictionary for the current boardState.

def displayBoard(boardState):

#to simplify gameplay we will number each column. Each column will be 3 spaces wide separated by the pipe character ( | ).

print(“ | 1 | 2 | 3 |”)

print(“-“\*16) #this prints a 16 dashes.

#initialize the board spot variable

spot = 0

#for loop to create each row

For row in range(3):

#print left side containing the row number

print(“ “+str(row+1)+” |”, end=””)

#now a for loop to go through each of the columns in this row

for col in range(3):

#if spot is empty, print a blank, else place the O or X.

If(boardState[spot]==’’):

print(‘ |’, end=””)

else:

print(‘ ‘+boardState[spot]+’ |’, end=””)

#increment spot by 1

spot+=1

print(‘’)

print(“-“\*16)

So now we have the main game function and the function to display the board. Next we need to handle the individual player turns. This function will require 3 parameters. These parameters will be the boardState dictionary, the number of the player as well as the letter to be placed on the board (X/O)

def playerTurn(boardState, ply, letter):

#display the board initially

displayBoard(boardState)

#wrap inside of a loop to keep going until valid entries are given

while(True):

if(ply != 3): #not the computer

print(‘Player ‘+str(ply)+’: ‘)

column = int(input(“Enter Column Number > “))

row = int(input(“Enter Row Number > “))

#else:

#code here for Computer turn

#Check to make sure the numbers entered are valid

If( (row < 4) and (row > 0) and (column < 4) and (column > 0)):

# as the rows and columns are all within on Dictionary we need to adjust some numbers to correctly select the right spot.

If( row == 1):

rowv = -1

elif(row == 2):

rowv = 2

elif(row == 3):

rowv = 5

#check to see if spot is taken, if not, place the letter and advance to next players turn

If(boardState[rowv+column] == ‘’):

boardState[rowv+column] = letter

#after letter placement, check on if the game is over running the gameOver() method that we have not made yet.

return (not gameOver(boardState))

else:

# if not the computer, let the player know they entered invalid positions

if(ply != 3):

print(‘Spot Taken, Try Again!’)

else:

print(‘Invalid coordinate, try again’)

#display the board one more time to allow the player to pick different coordinates

displayBoard(boardState)

The final function needed is the gameOver function which takes the boardState as a parameter. This function will check to see if there is 3 X’s in a row or 3 O’s in a row or if the game ended in a stalemate. This may be one of the more confusing methods to run through.

def gameOver(boardState):

#set gameDone variable which determines if the game is over

gameDone = True

#set the winning player

win = ‘?’

#first lets check if the board is full. If it is not full, we will set gameDone to false.

for count in range(len(boardState)):

if(boardState[count]==’’):

gameDone = False

#now we check to see if the board contains a win.

If(not gameDone):

#check each row

for count in range(3):

#check to see if all values in that row are the same, and also not blank!

If(boardState[count\*3]== boardState[count\*3+1]== boardState[count\*3+2] and boardState[count\*3] != ‘’):

gameDone = True

#set winner

win = boardState[count\*3]

# check each column in a similar way

for count in range(3):

#check to see if all values in that column are the same, and also not blank!

if(boardState[count]==boardState[count+3]==boardState[count+6] and boardState[count] != ''):

gameDone = True

#set winner

win = boardState[count]

#check diagonals

if(boardState[0]==boardState[4]==boardState[8] and boardState[4] != ''):

gameDone = True

win = boardState[4]

if(boardState[2]==boardState[4]==boardState[6] and boardState[4] != ''):

gameDone = True

win = boardState[4]

#if game is over check the winner and display

if(gameDone):

if(win == "?"):

print("Game ends in stalemate!")

else:

print("Winner is : "+win)

#return true or false back to calling function

return gameDone