# Evidence 7

class Node:

def \_\_init\_\_(self, Data):

self.\_\_LeftP = int(0)

self.\_\_Data = str(Data)

self.\_\_RightP = int(0)

def GetLeftP(self):

return self.\_\_LeftP

def GetRightP(self):

return self.\_\_RightP

def GetData(self):

return self.\_\_Data

def SetLeftP(self, LeftP):

self.\_\_LeftP = int(LeftP)

def SetRightP(self, RightP):

self.\_\_RightP = int(RightP)

def SetData(self, Data):

self.\_\_Data = str(Data)

class BinaryTree:

def \_\_init\_\_(self):

self.\_\_ThisTree = [Node("") for i in range(20 + 1)]

self.\_\_Root = 0

self.\_\_NextFreePosition = 1

for i in range(1, 20):

self.\_\_ThisTree.SetLeftP(i + 1)

# Evidence 8

def AddItemToBinaryTree(self, NewTreeItem):

if self.\_\_NextFreePosition == 0: # binary tree is full

print("No space available, cannot add to binary tree!")

return

else:

temp = self.\_\_ThisTree[self.\_\_NextFreePosition].GetLeftP()

self.\_\_ThisTree[self.\_\_NextFreePosition].SetData(NewTreeItem) # assign NewTreeItem to next free position to be inserted

self.\_\_ThisTree[self.\_\_NextFreePosition].SetLeftP(0)

self.\_\_ThisTree[self.\_\_NextFreePosition].SetRightP(0)# no effect because RightP already 0

LastMove = 'X'

if self.\_\_Root == 0:

self.\_\_Root = self.\_\_NextFreePosition

else:

CurrentPosition = self.\_\_Root

LastMove = 'X'

while CurrentPosition != 0:

PreviousPosition = CurrentPosition

if NewTreeItem < self.\_\_ThisTree[CurrentPosition].GetData():

# move left

LastMove = 'L'

CurrentPosition = self.\_\_ThisTree[CurrentPosition].GetLeftP()

else:

# move right

LastMove = 'R'

CurrentPosition = self.\_\_ThisTree[CurrentPosition].GetRightP()

if LastMove == 'R':

self.\_\_ThisTree[PreviousPosition].SetRightP(self.\_\_NextFreePosition)

elif LastMove == 'L':

self.\_\_ThisTree[PreviousPosition].SetLeftP(self.\_\_NextFreePosition)

self.\_\_NextFreePosition = temp

# Evidence 9

def OutputData(self):

print("Value of Root: {0}".format(self.\_\_Root))

print("Value of NextFreePosition: {0}".format(self.\_\_NextFreePosition))

print()

print("-" \* 73)

print("| {0:^10} | {1:^15} | {2:^20} | {3:^15} |".format("Index", "Left Pointer", "Data", "Right Pointer"))

print("-" \* 73)

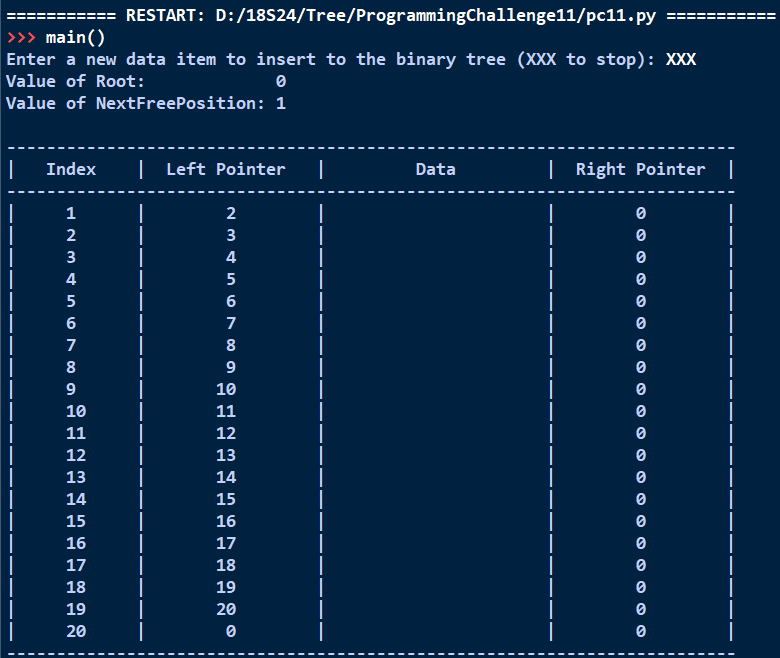
for i in range(1, 21):

CurrentNode = self.\_\_ThisTree[i]

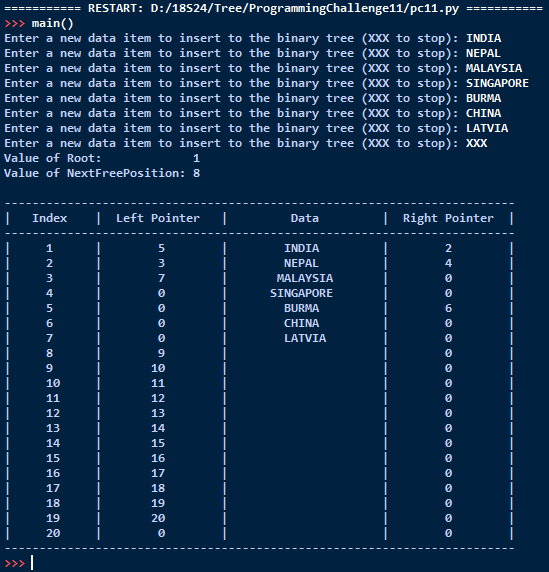
print("| {0:^10} | {1:^15} | {2:^20} | {3:^15} |".format(i, CurrentNode.GetLeftP(), CurrentNode.GetData(), CurrentNode.GetRightP()))

print("-" \* 73)

# Evidence 10



# Evidence 11



# Evidence 12

def TraverseInOrderFromRoot(self): # function to traverse in-order from root

self.TraverseInOrder(self.\_\_Root)

def TraverseInOrder(self, index):

if index != 0:

self.TraverseInOrder(self.\_\_ThisTree[index].GetLeftP())

print(self.\_\_ThisTree[index].GetData())

self.TraverseInOrder(self.\_\_ThisTree[index].GetRightP())

# Evidence 13

