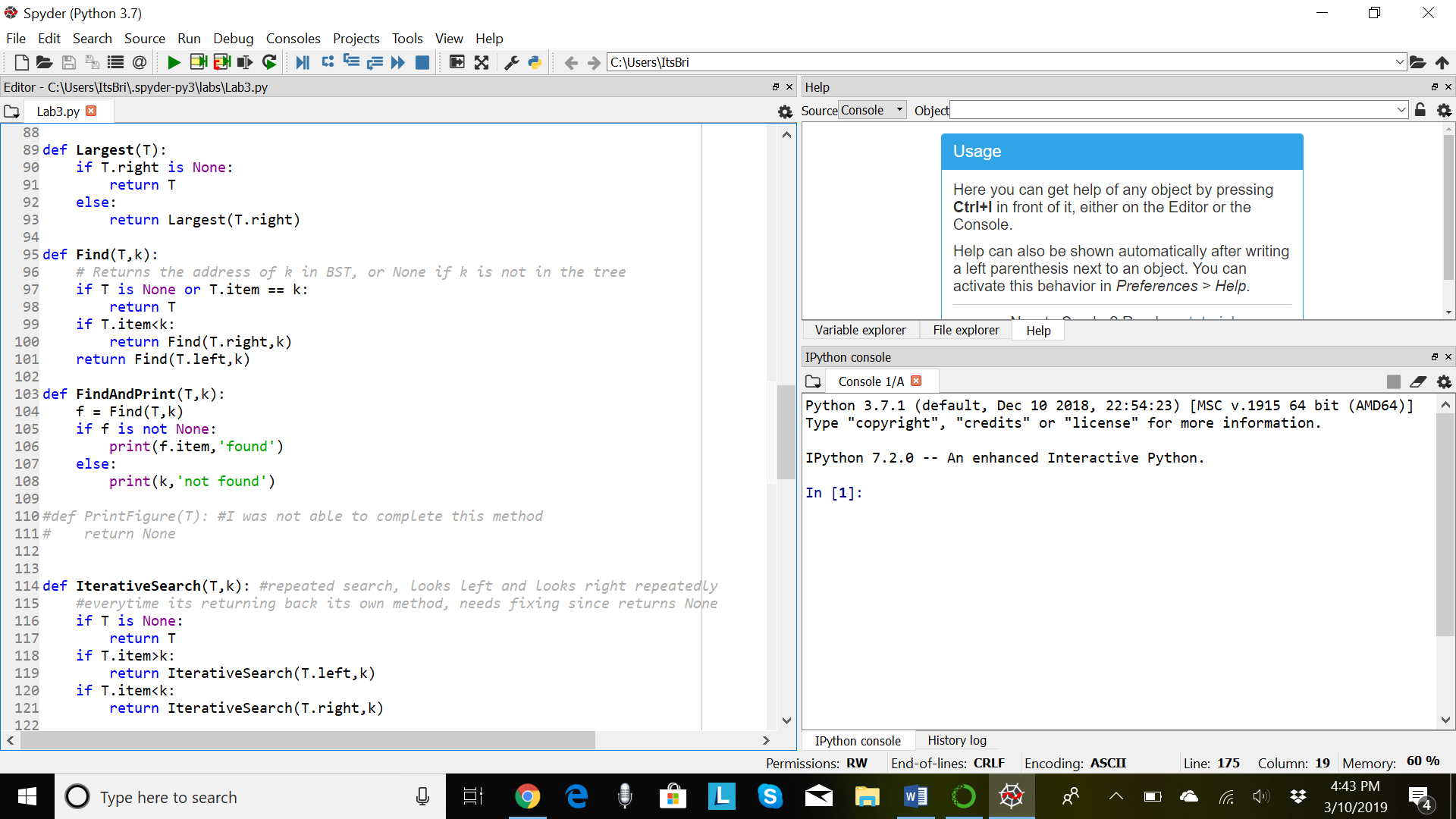
Tovar, Brianna

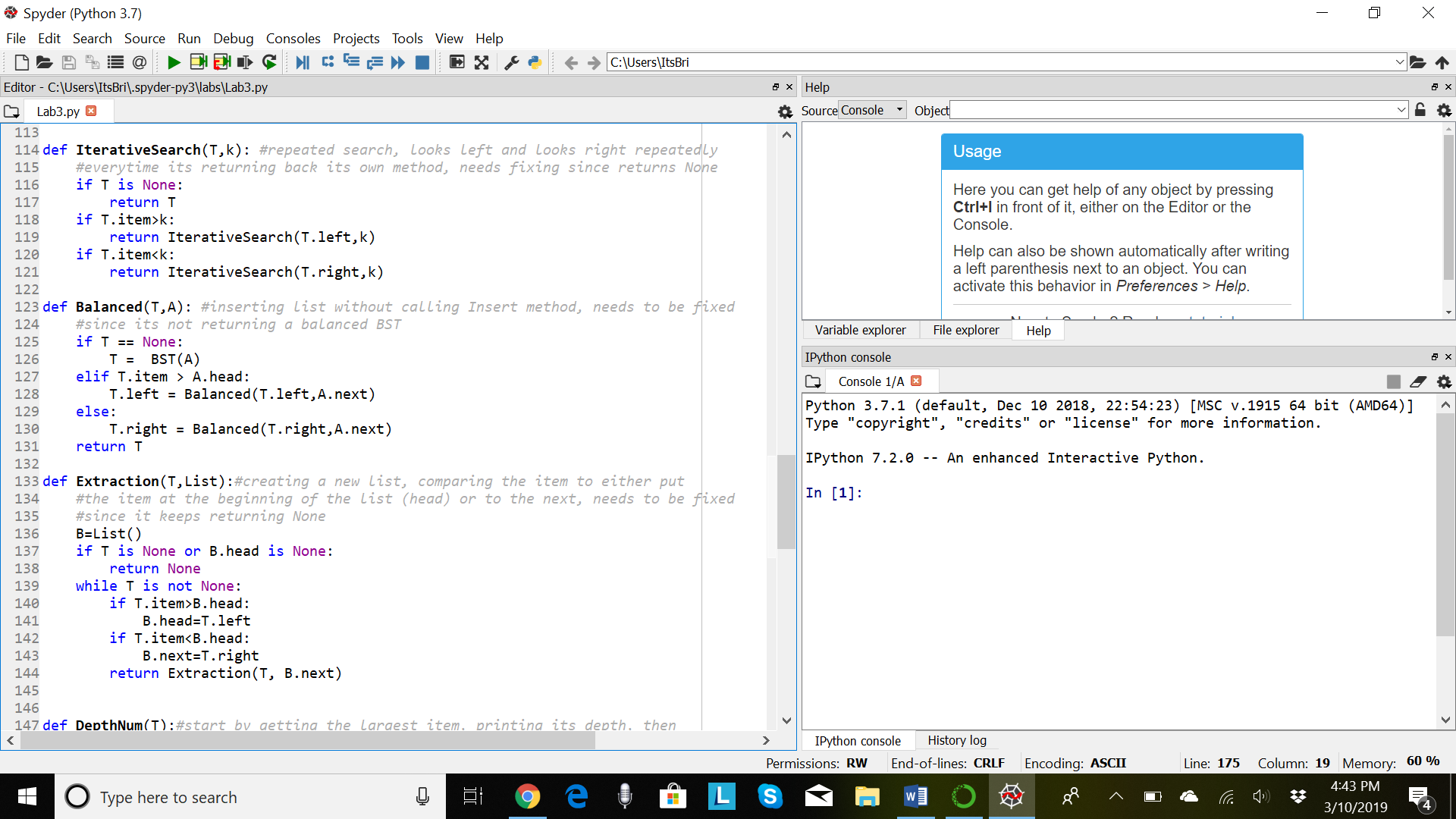
MW 1:30-2:50pm

Lab 3: BST

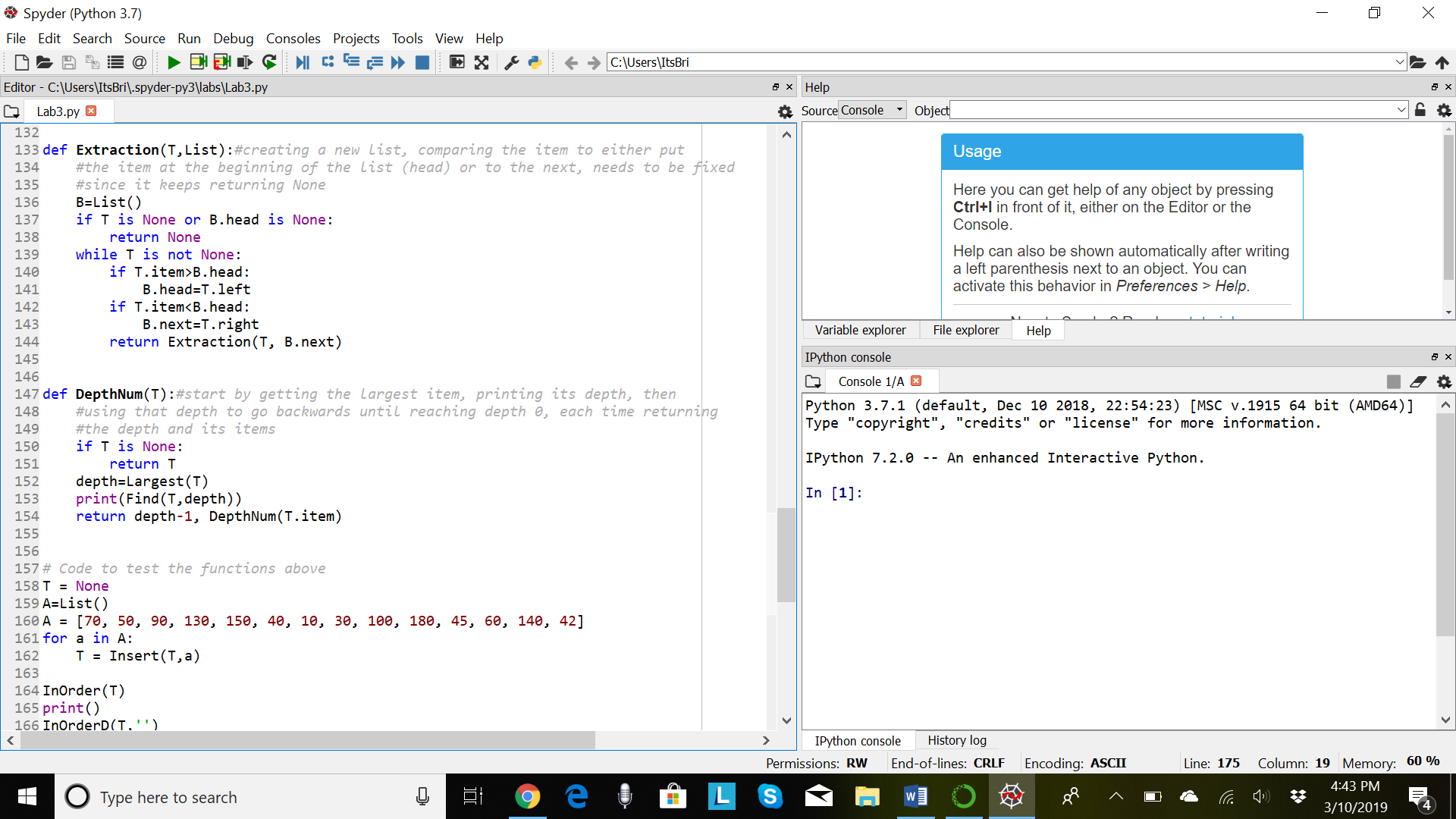
This lab was by far the hardest I’ve had to complete. Binary search trees on their own are easier for me to complete, but creating and exporting lists from them was no simple task. For the beginning of my code, my first method under “PrintFigure”, I was not able to complete since besides not fully understanding how to draw bubbles around each node of the binary tree and the stems connecting them all.

I did, however, was able to work on my next four methods. My second method named “IterativeSearch”, I was trying to compute the find method but instead repeating similar to how a for loop would be implemented. Unfortunately, I could not work this method out since it kept returning None instead of checking the binary tree left children first and then right.

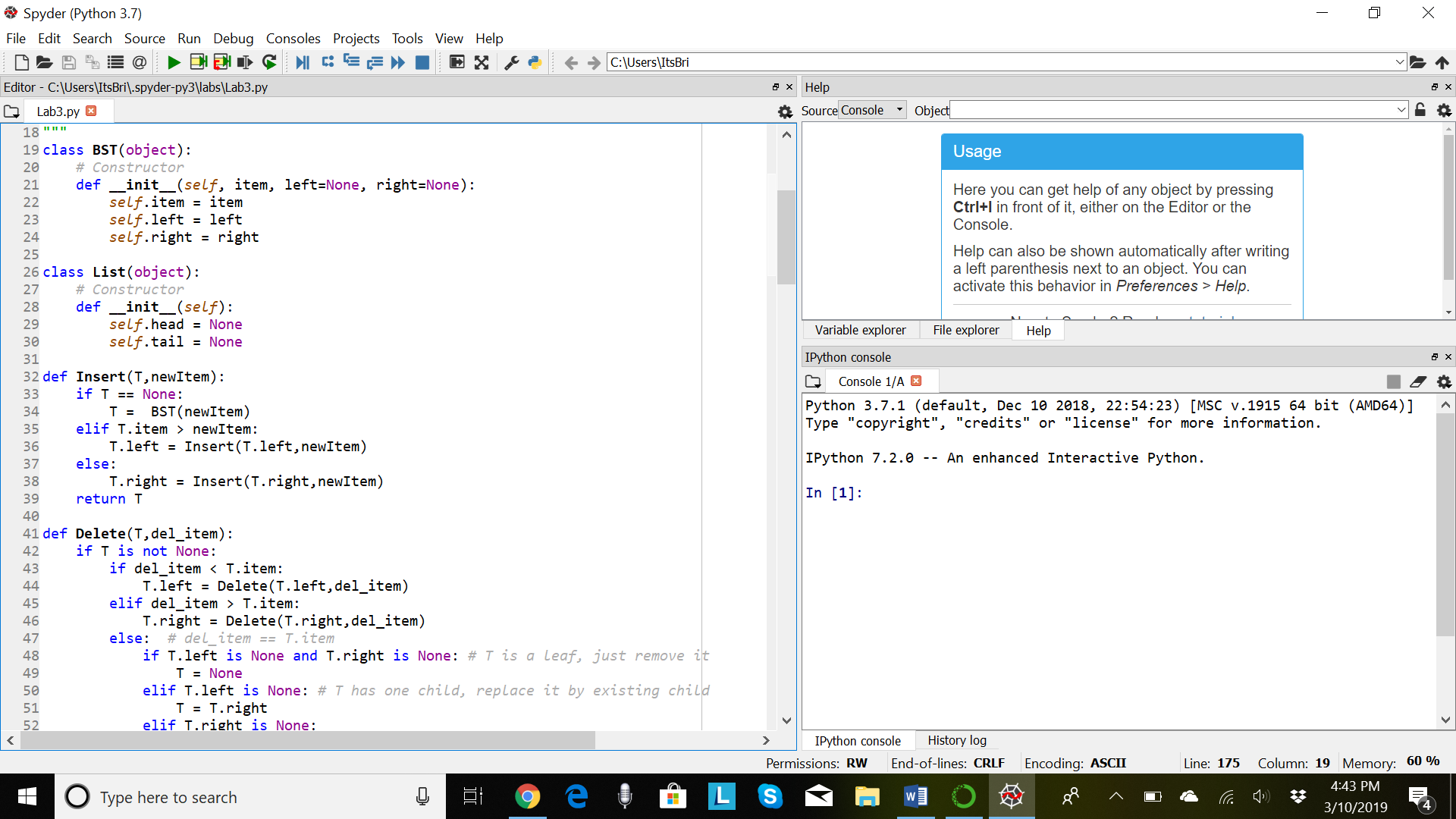
For my third method, named “Balanced”, I tried implementing the insert method given but changing the conditions so it follows a balanced tree instead. For this, I followed what the insert method uses to insert the list into the tree, but add conditions of “if Full”. In the end, I wasn’t able to finish this method since this was the last method I worked on, and I wasn’t able to include the conditions to create a balanced binary tree instead of a regular insert method for a binary search tree.



As for my fourth method, named “Extraction”, I was not able to finish but was able to compile it. This method was supposed to create a new list I named “B”, and go throughout the binary search tree to grab each node and “Extract” that node into its place in the list. It first checks if T is None or if the head is none, it’ll return None. Then, I implemented a while loop to help keep track of where in the list I was adding to from the binary tree, taking each one by one so my method can have a running time of O(n). In the end, this method keep returning None.



As for my last method, my fifth method named “DepthNum”, this was supposed to return the number of items in each depth. It starts by taking the depth of the Largest method for the given binary tree, then going backwards, would take each of those depths, going down, and print the items given. This was another method of mine that returned None or an error when given its conditions.



To implement my “B” list created, I had to create the object itself using a constructor.

Lastly, to conclude why my program wasn’t finished on time nor completed, work was endless this past week, and I didn’t fully understand the methods to create for Binary Search Trees. For this, I wasn’t able to find my running times for my methods.

Appendix:

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

class List(object):

# Constructor

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

self.head = None

self.tail = None

def Insert(T,newItem):

if T == None:

T = BST(newItem)

elif T.item > newItem:

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

def Delete(T,del\_item):

if T is not None:

if del\_item < T.item:

T.left = Delete(T.left,del\_item)

elif del\_item > T.item:

T.right = Delete(T.right,del\_item)

else: # del\_item == T.item

if T.left is None and T.right is None: # T is a leaf, just remove it

T = None

elif T.left is None: # T has one child, replace it by existing child

T = T.right

elif T.right is None:

T = T.left

else: # T has two chldren. Replace T by its successor, delete successor

m = Smallest(T.right)

T.item = m.item

T.right = Delete(T.right,m.item)

return T

def InOrder(T):

# Prints items in BST in ascending order

if T is not None:

InOrder(T.left)

print(T.item,end = ' ')

InOrder(T.right)

def InOrderD(T,space):

# Prints items and structure of BST

if T is not None:

InOrderD(T.right,space+' ')

print(space,T.item)

InOrderD(T.left,space+' ')

def SmallestL(T):

# Returns smallest item in BST. Returns None if T is None

if T is None:

return None

while T.left is not None:

T = T.left

return T

def Smallest(T):

# Returns smallest item in BST. Error if T is None

if T.left is None:

return T

else:

return Smallest(T.left)

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

def Find(T,k):

# Returns the address of k in BST, or None if k is not in the tree

if T is None or T.item == k:

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

#def PrintFigure(T): #I was not able to complete this method

# return None

def IterativeSearch(T,k): #repeated search, looks left and looks right repeatedly

#everytime its returning back its own method, needs fixing since returns None

if T is None:

return T

if T.item>k:

return IterativeSearch(T.left,k)

if T.item<k:

return IterativeSearch(T.right,k)

def Balanced(T,A): #inserting list without calling Insert method, needs to be fixed

#since its not returning a balanced BST

if T == None:

T = BST(A)

elif T.item > A.head:

T.left = Balanced(T.left,A.next)

else:

T.right = Balanced(T.right,A.next)

return T

def Extraction(T,List):#creating a new list, comparing the item to either put

#the item at the beginning of the list (head) or to the next, needs to be fixed

#since it keeps returning None

B=List()

if T is None or B.head is None:

return None

while T is not None:

if T.item>B.head:

B.head=T.left

if T.item<B.head:

B.next=T.right

return Extraction(T, B.next)

def DepthNum(T):#start by getting the largest item, printing its depth, then

#using that depth to go backwards until reaching depth 0, each time returning

#the depth and its items

if T is None:

return T

depth=Largest(T)

print(Find(T,depth))

return depth-1, DepthNum(T.item)

# Code to test the functions above

T = None

A=List()

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

T = Insert(T,a)

InOrder(T)

print()

InOrderD(T,'')

print()

print(SmallestL(T).item)

print(Smallest(T).item)

print(IterativeSearch(T,90))

print(Balanced(T,A))

print(Extraction(T,List))

print(DepthNum(T))