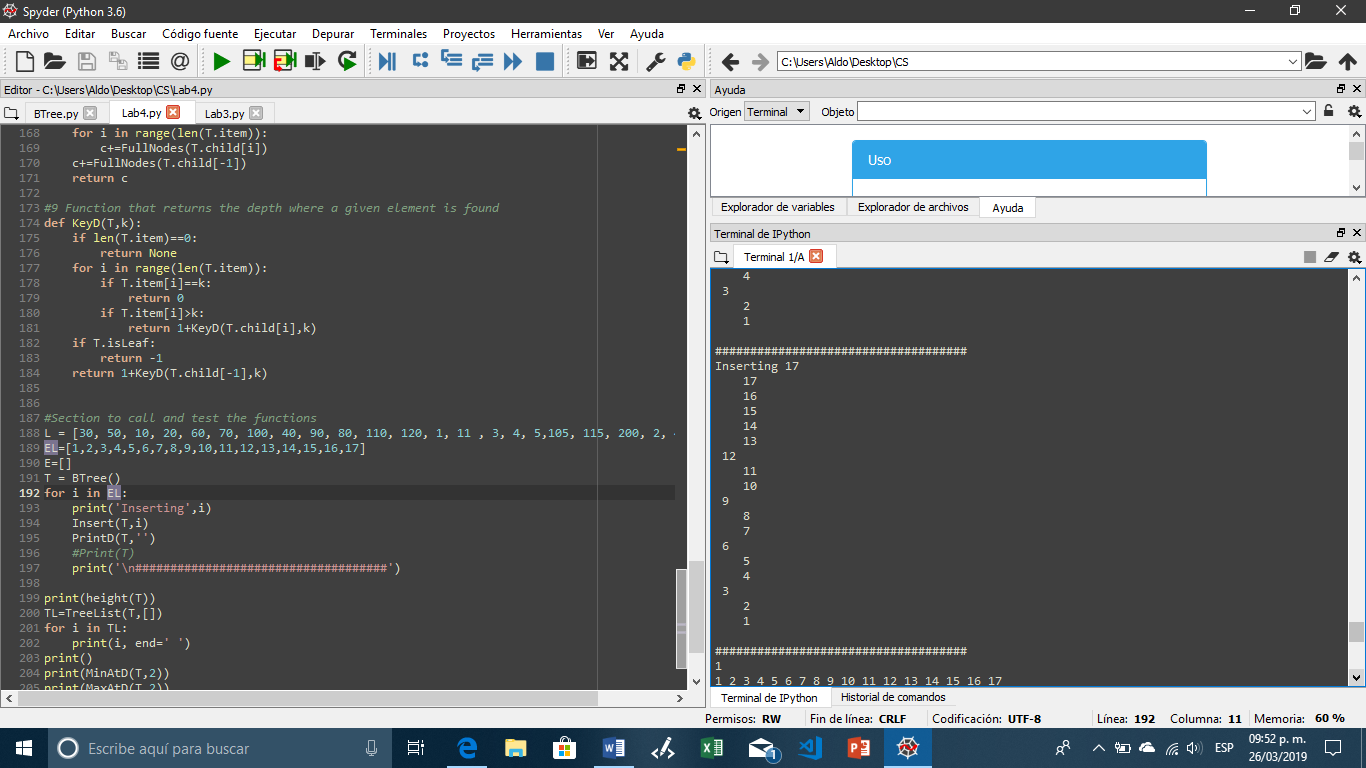
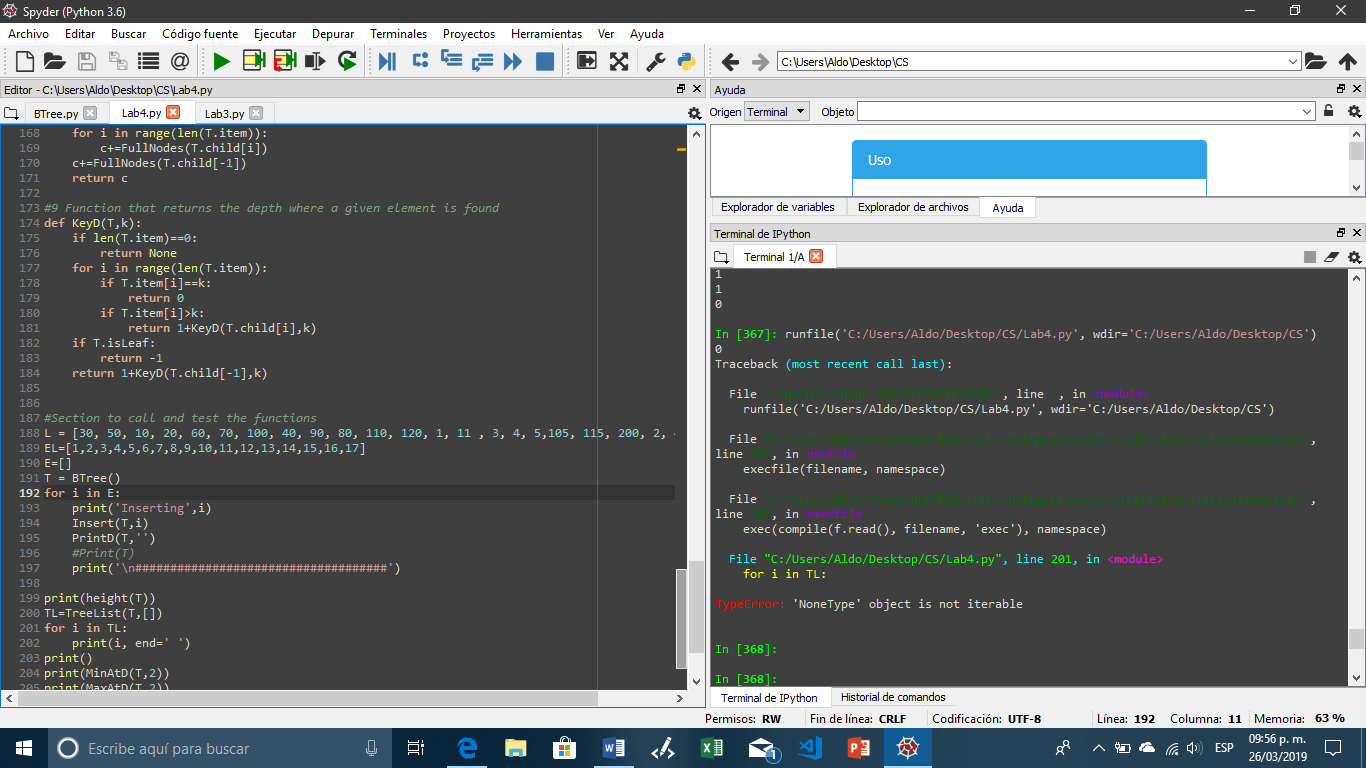
For this lab, the problem we were approaching was related with b-trees. Basically, our task was to execute some functions to perform operation to modify and handle b-trees.

**Height:** For this function we where supposed to obtain the height of the given tree. In this case this function was given by professor Fuentes, but the basic idea is to add one recursively until we reach the end of the tree.

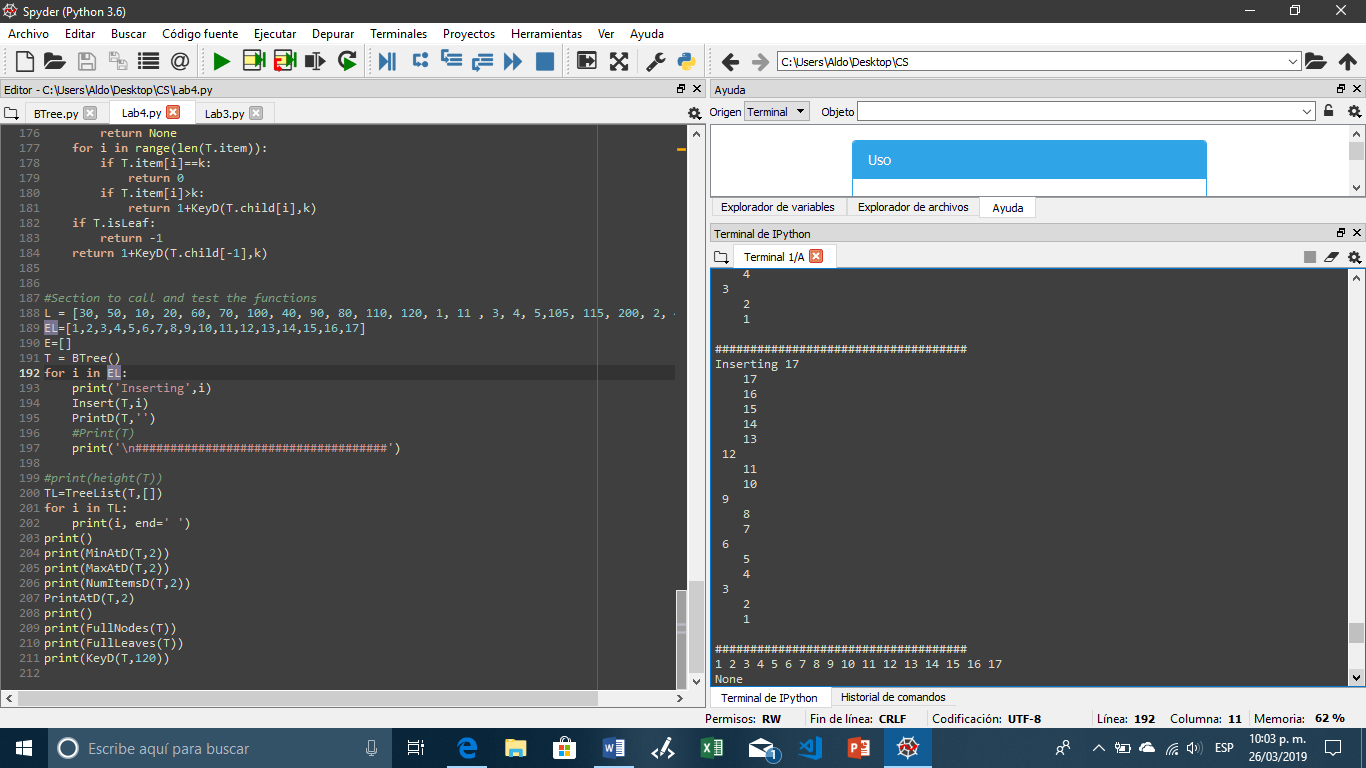
Input: Normal tree Input: Empty tree

**Sorted list from tree:** For this function we need to store every element in the tree into a sorted list. Here we first check if the current node is a leaf in case, we have reached the end of the list or in case we are dealing with a list with only one node. If this is the case, we append all the elements in this node to the list. If this not true, we need to reach the last node, and the best way to reach every node in order is with a for loop, so we recursively call every node inside the for loop appending at the same time the elements in each node to the list. At the end the sorted list is returned.

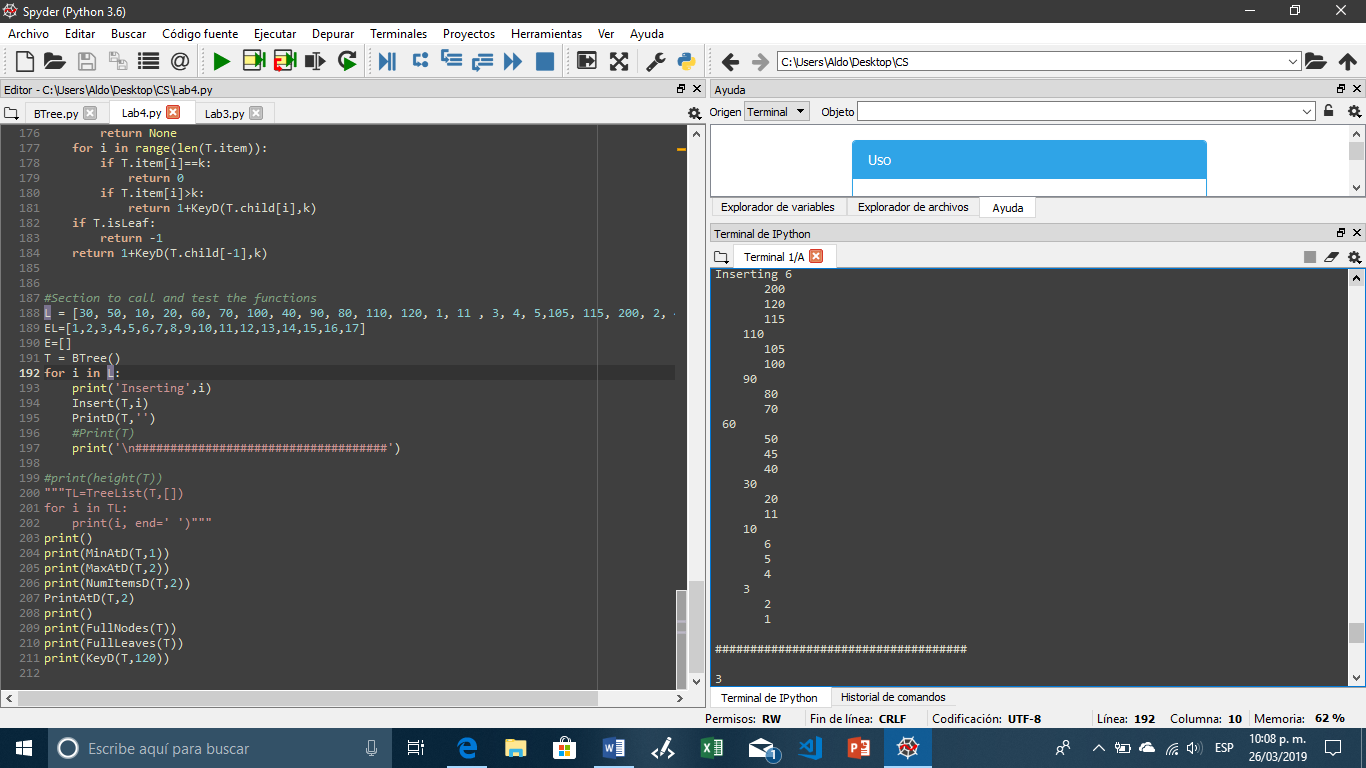
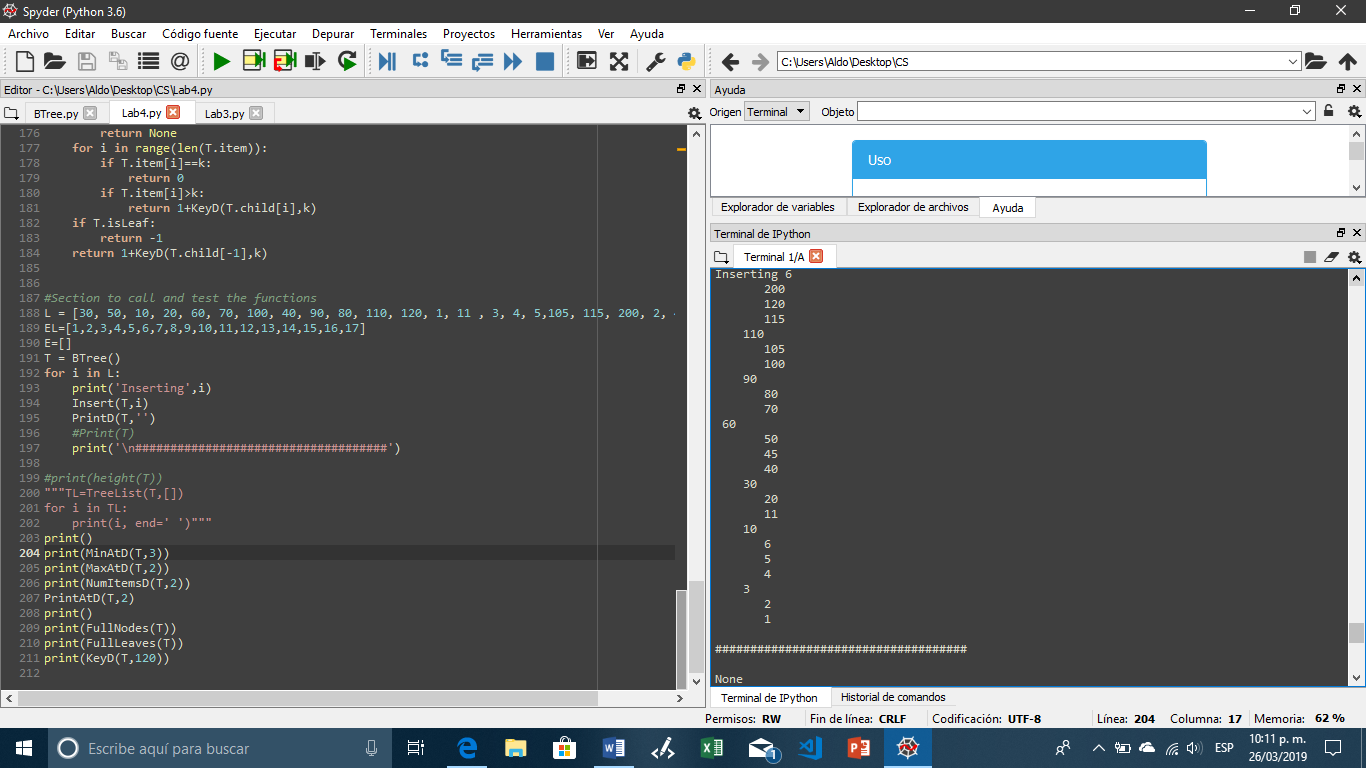
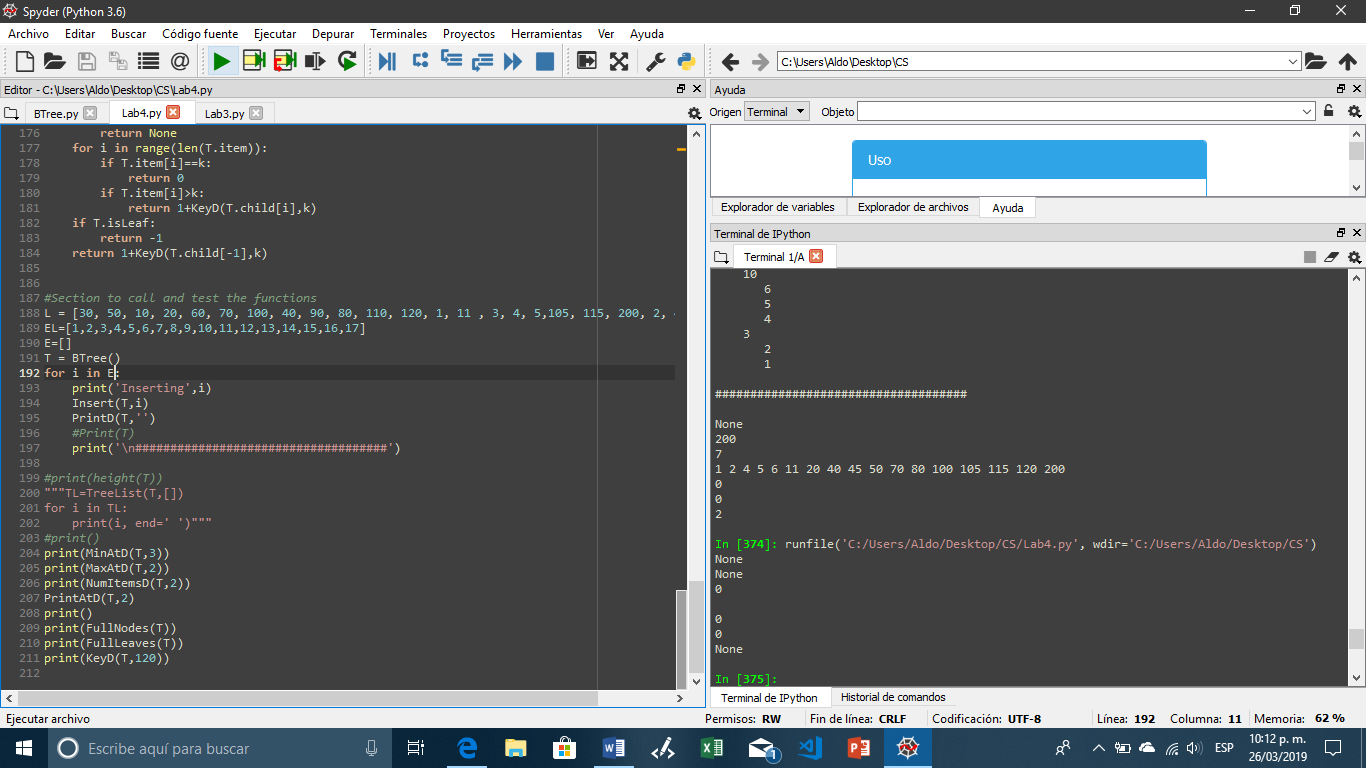
Input: Normal tree Input: Empty tree

If we have an empty tree nothing is printed



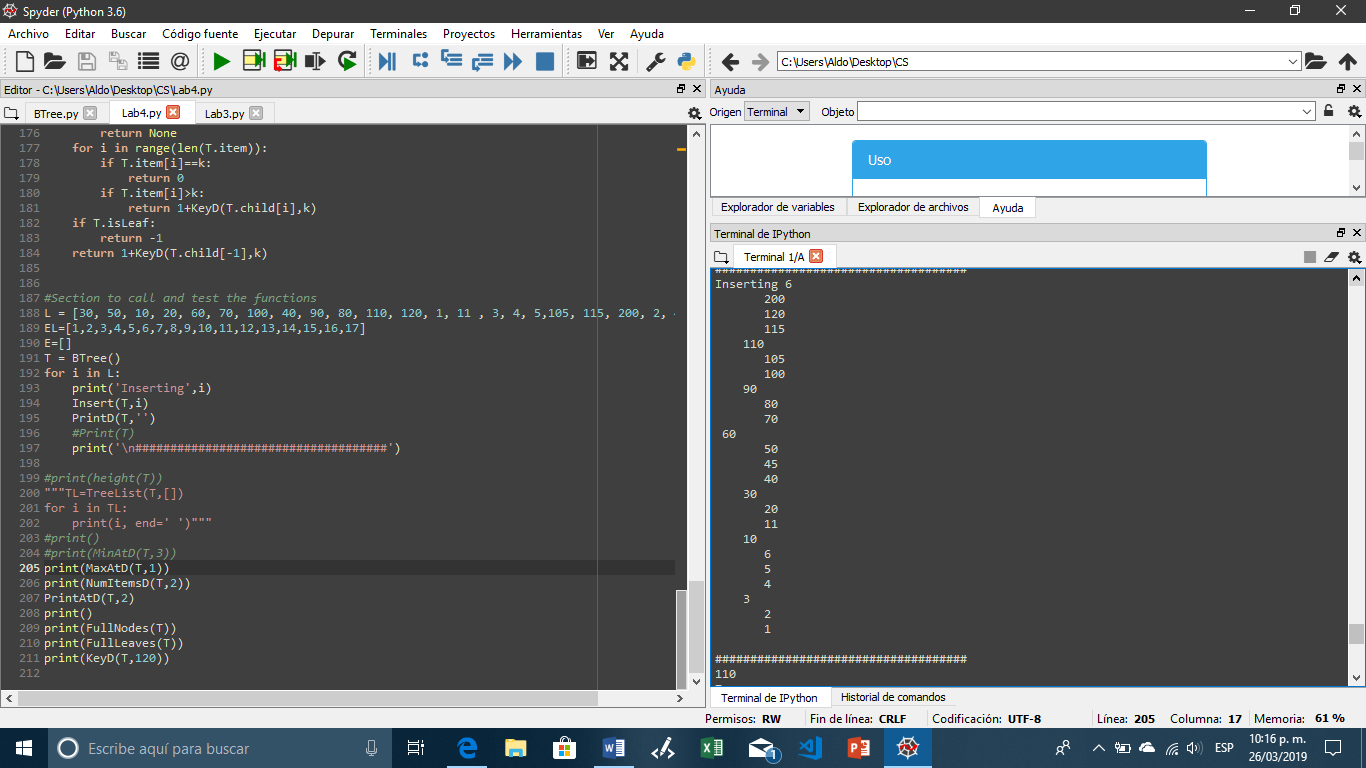
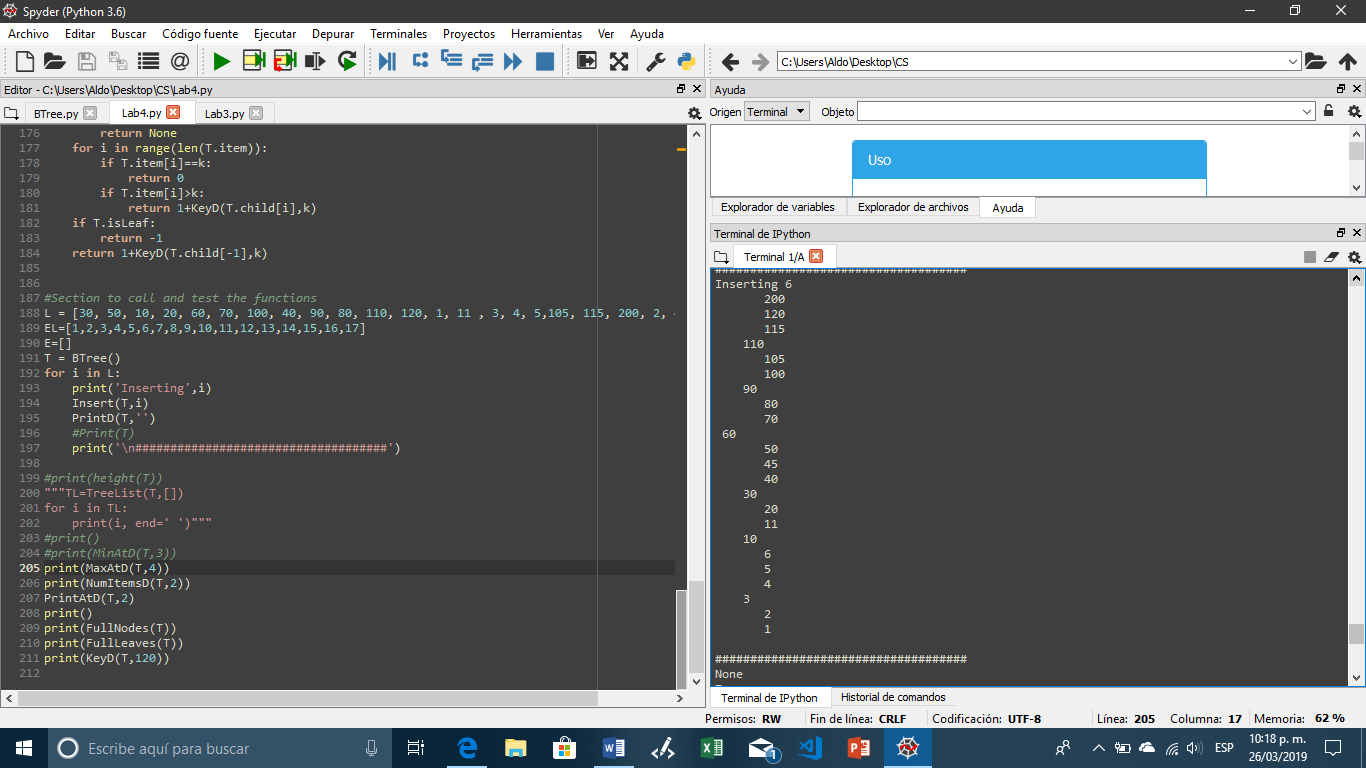
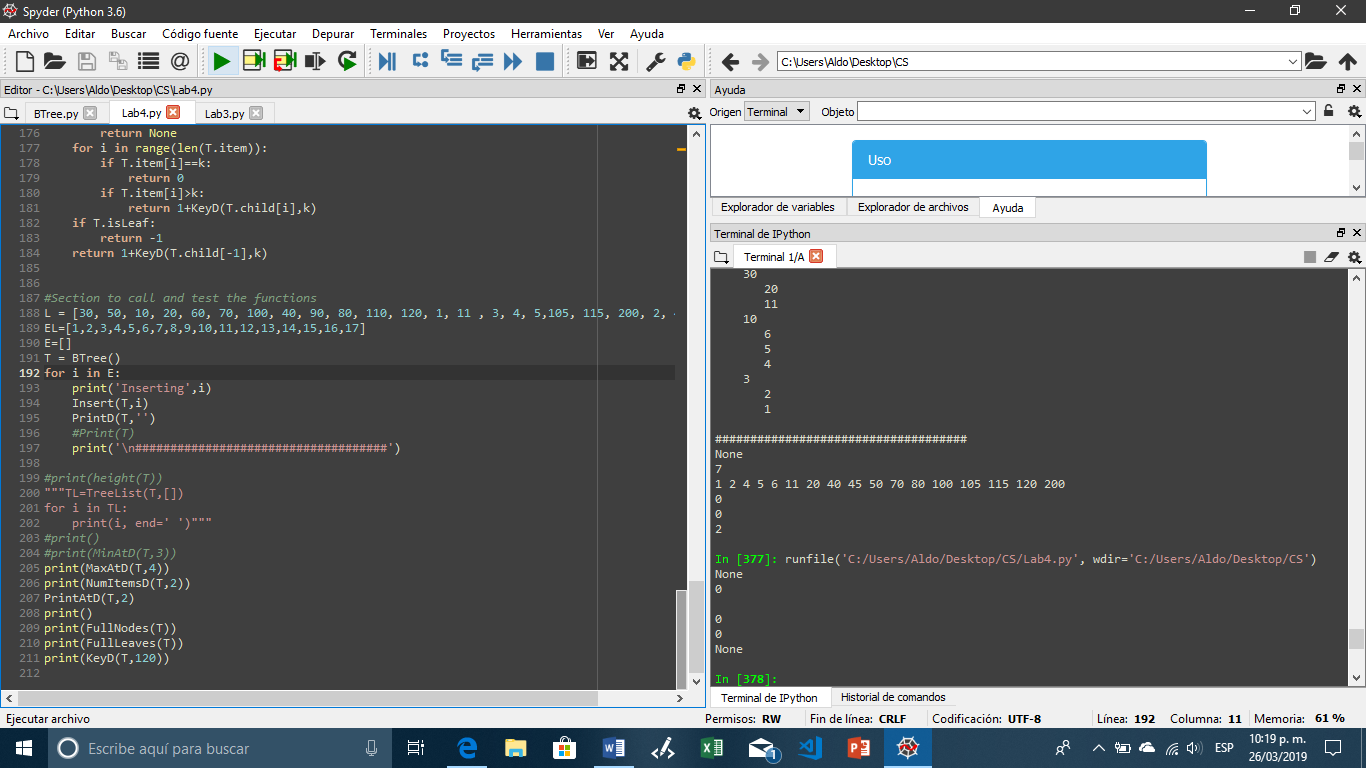
**Smallest at depth:** For this function we needed to find the smallest elements at a given depth and return it when found. Recursively we will reduce the depth until we find the level we are looking for. When the given depth is found we return the left most elements, because it is the smallest in that depth. If the depth we are looking for if bigger than the length of the tree it will return none.

Input: Normal tree(d=1) Input: Index out of bounds (d=3) Input: Empty tree

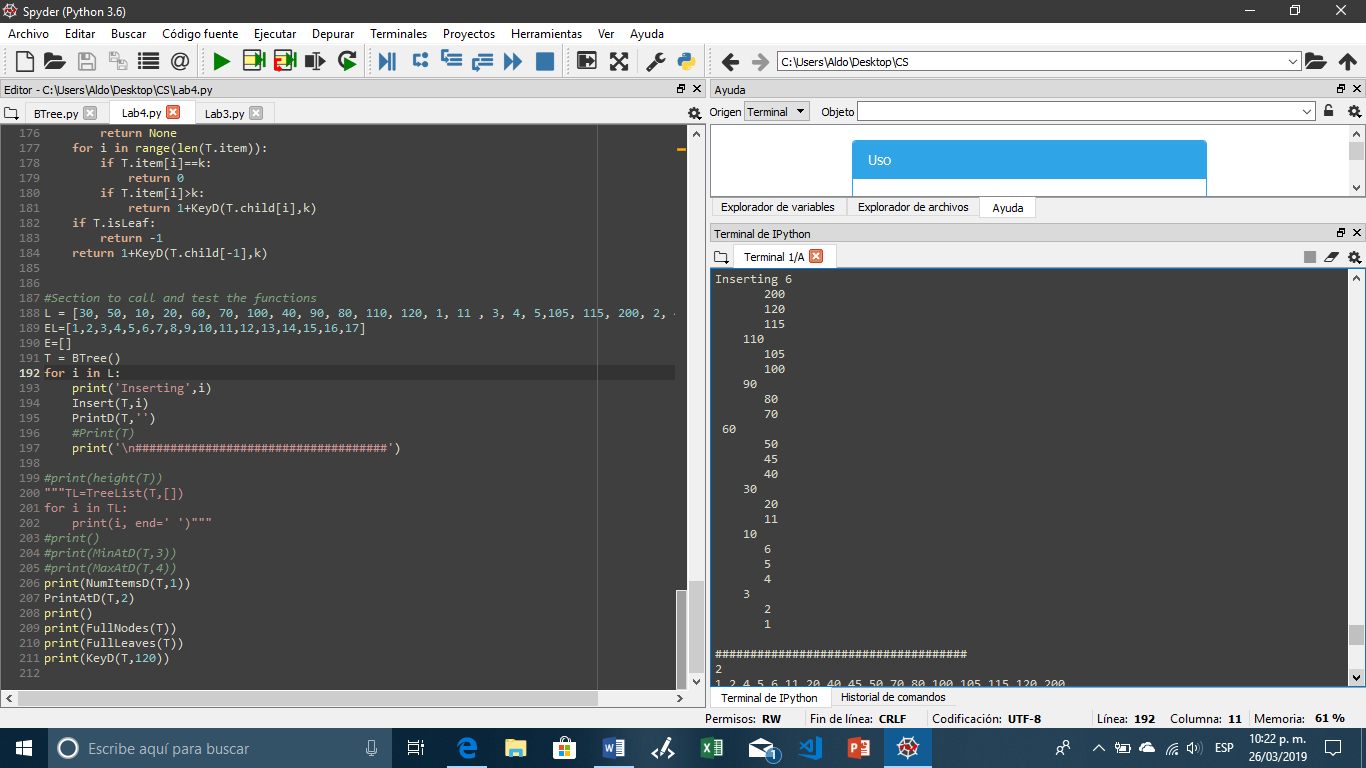
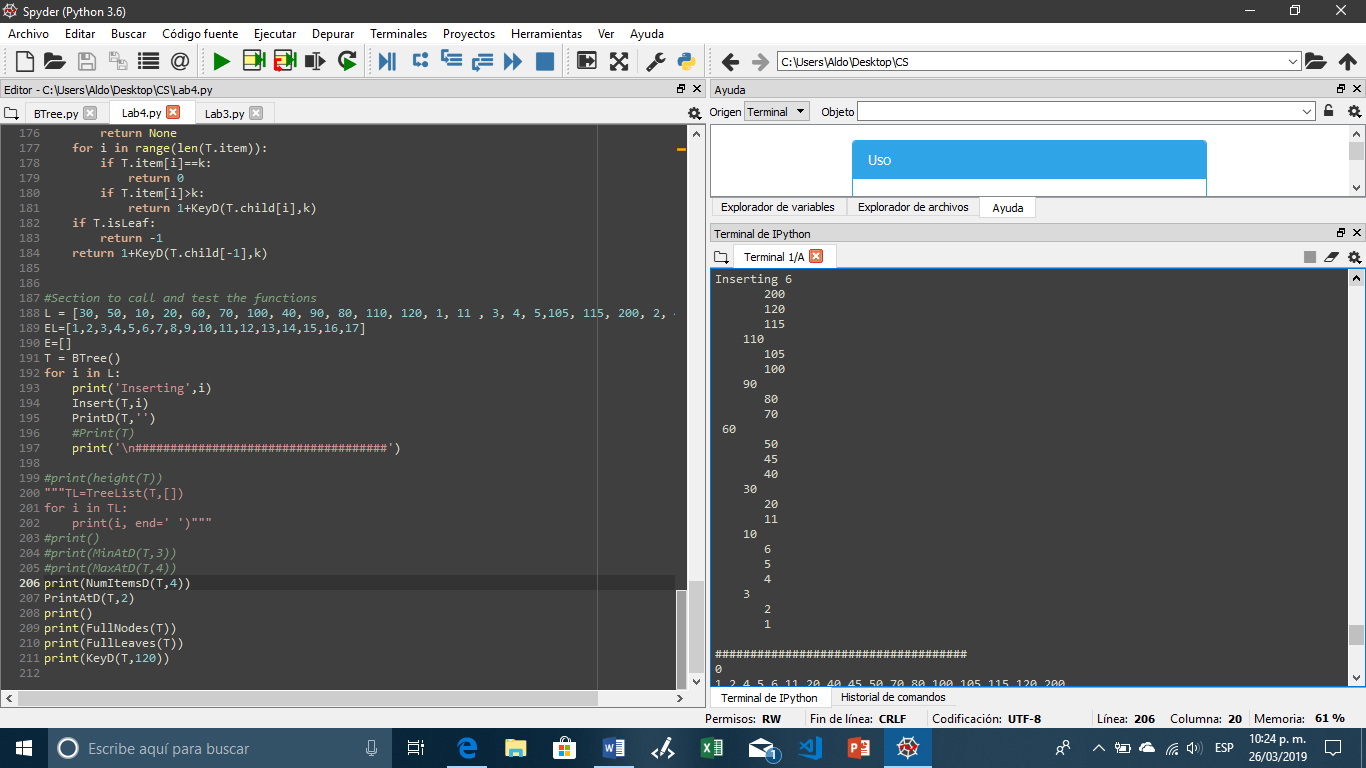
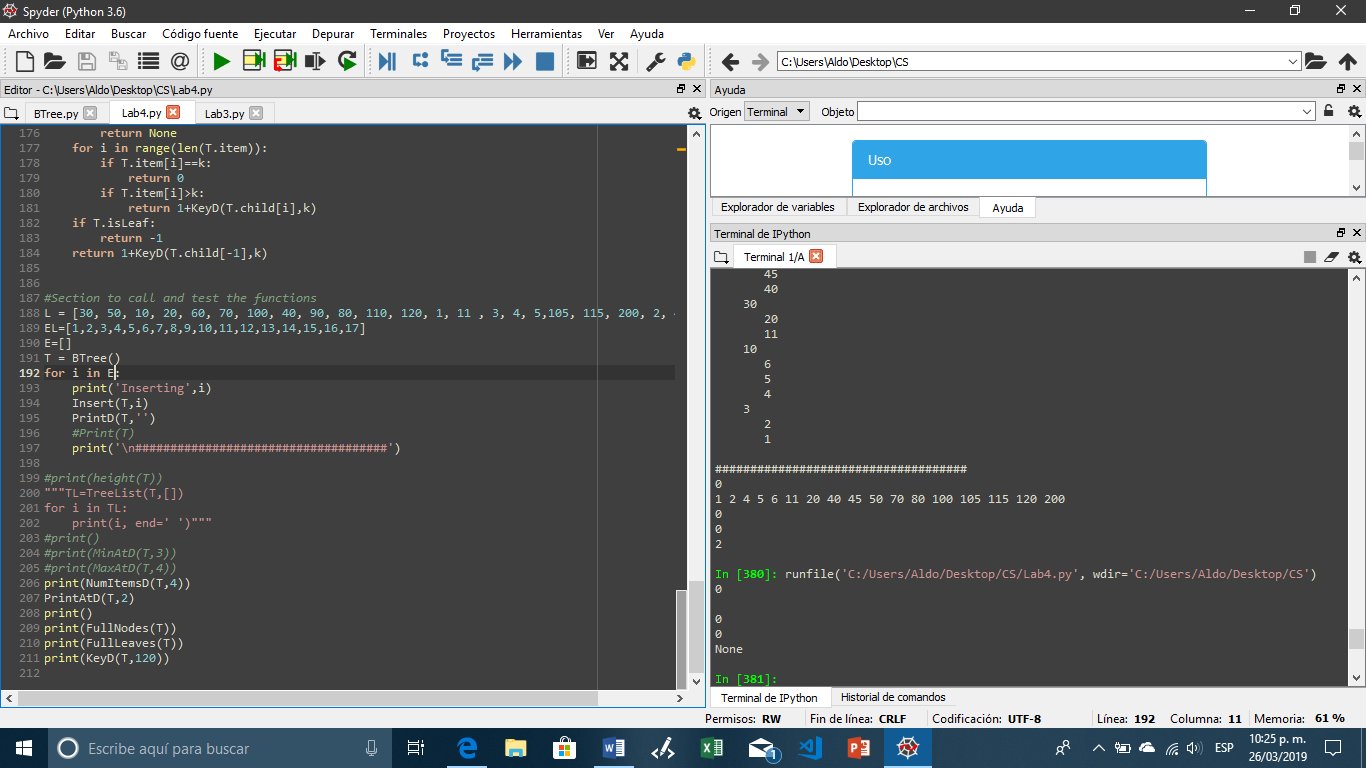
**Biggest at depth:** This function is kind of similar to the previous one. We need to find the level we are looking for with recursion, and when we reach it, we return the right most element because it will be the biggest one at that level. And again, if the depth we are given is bigger than the height of the tree it will return none.

Input: Normal tree (d=1) Input: Index out of bounds (d=4) Input: Empty tree

**Elements at depth:** This function returns the number of nodes at a given depth. As I did previously, I will use a for loop to access every element in the tree decreasing the depth until I reach the level I am looking for, once I get there, I add one every time I see a node at that level.

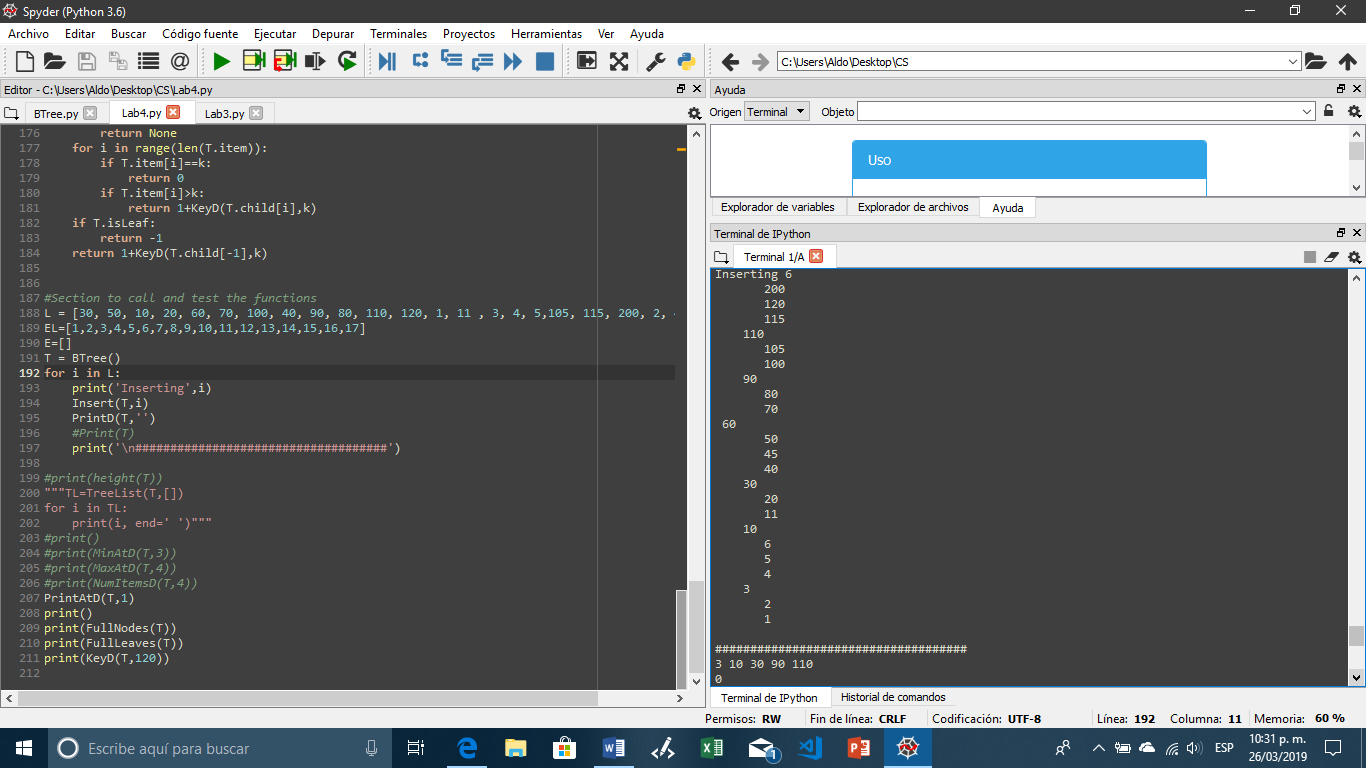
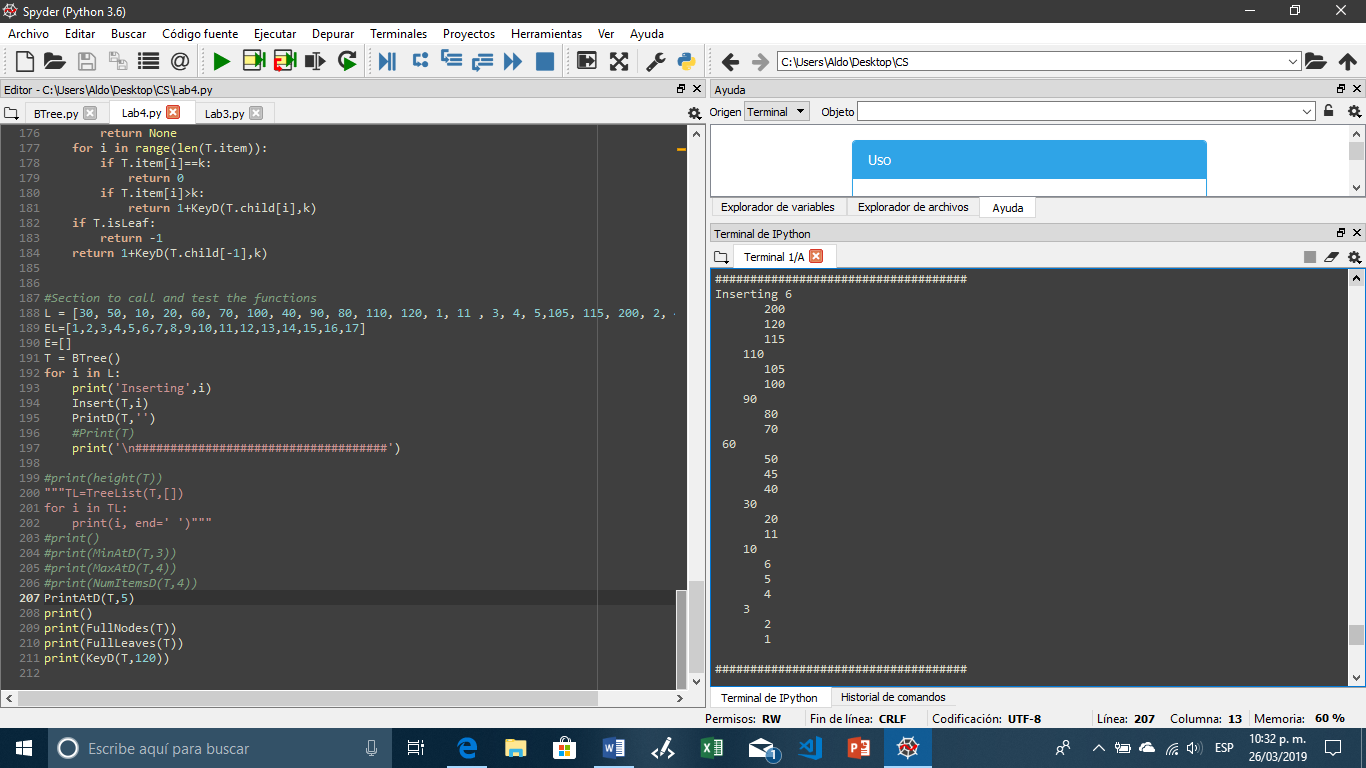
Input: Normal tree(d=1) Input: Index out of bounds(d=4) Input: Empty tree

**Print at depth:** For this function we need to print every element at a given depth. Again, a for loop is required to access every node in the tree using recursion, and once we reach the level we are looking for, we print all the elements at that level using a loop.

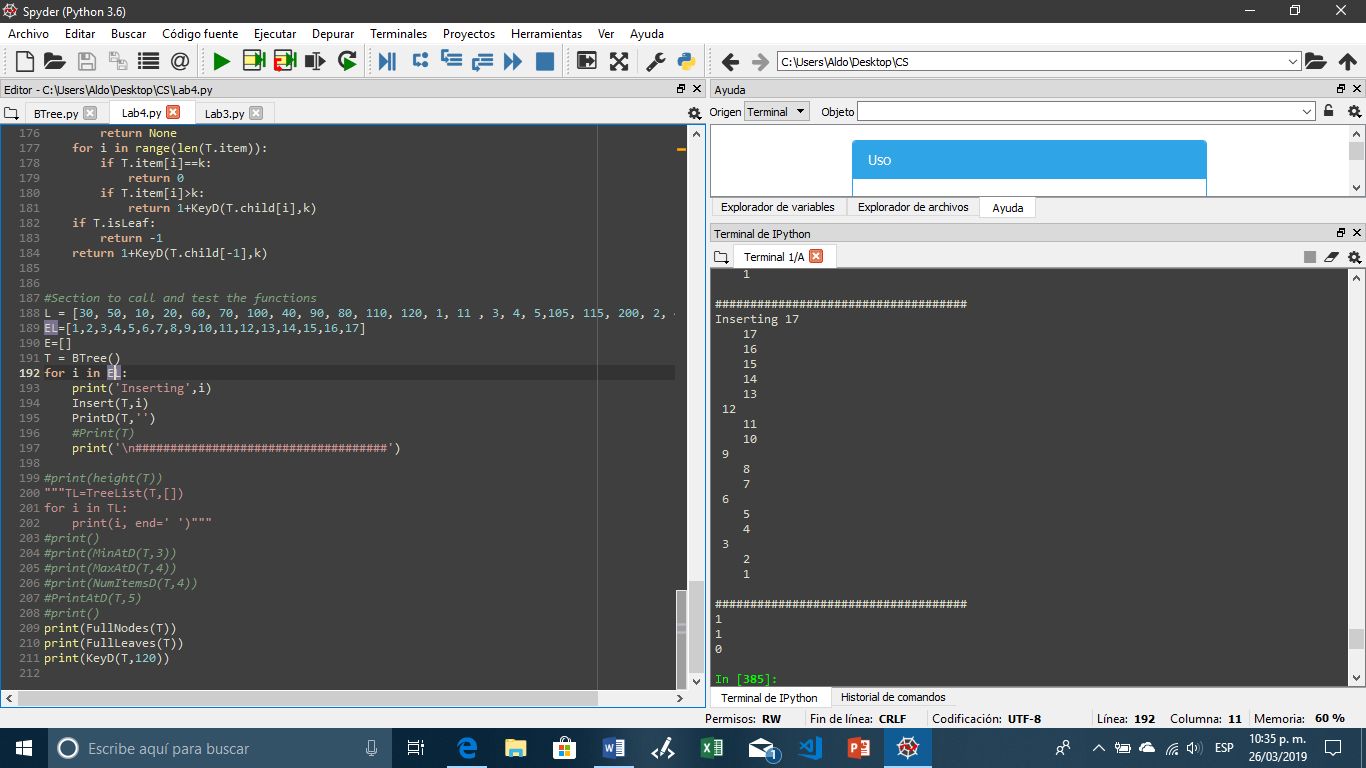
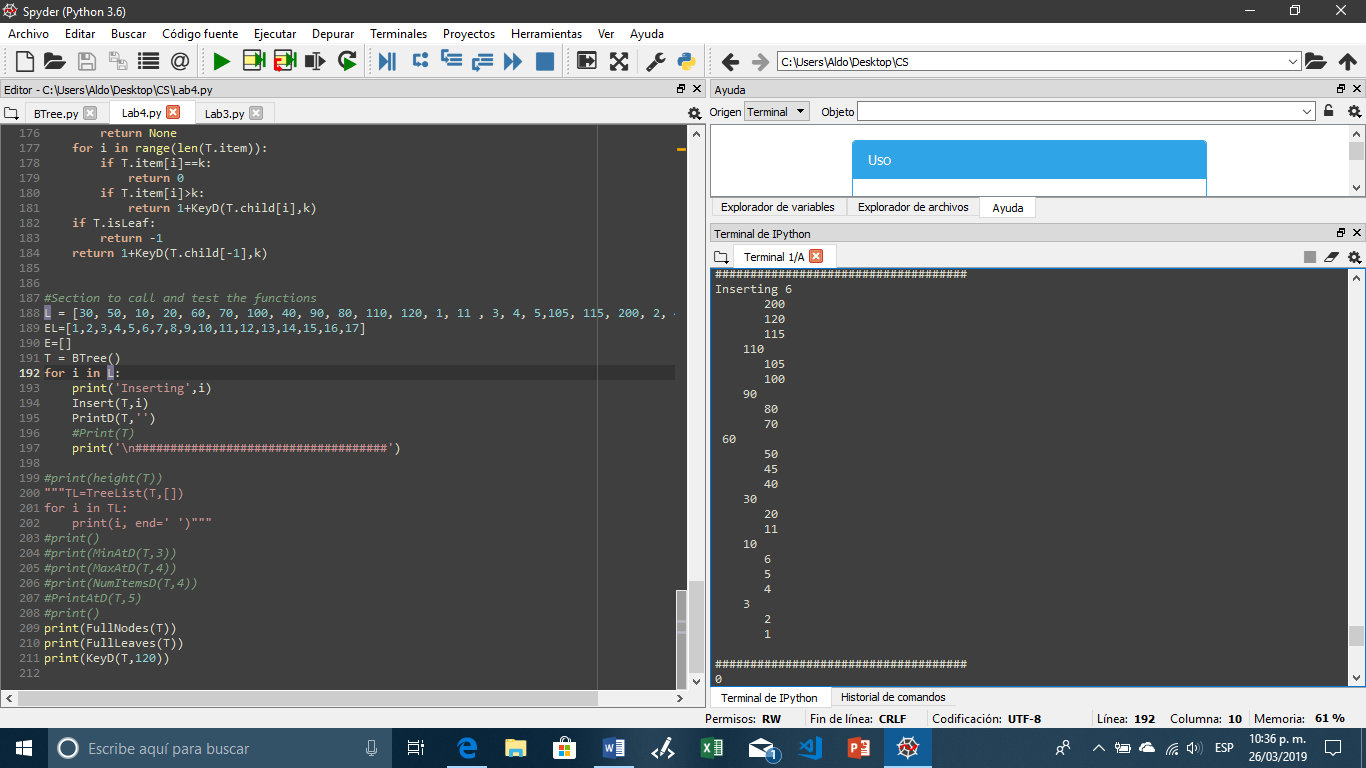
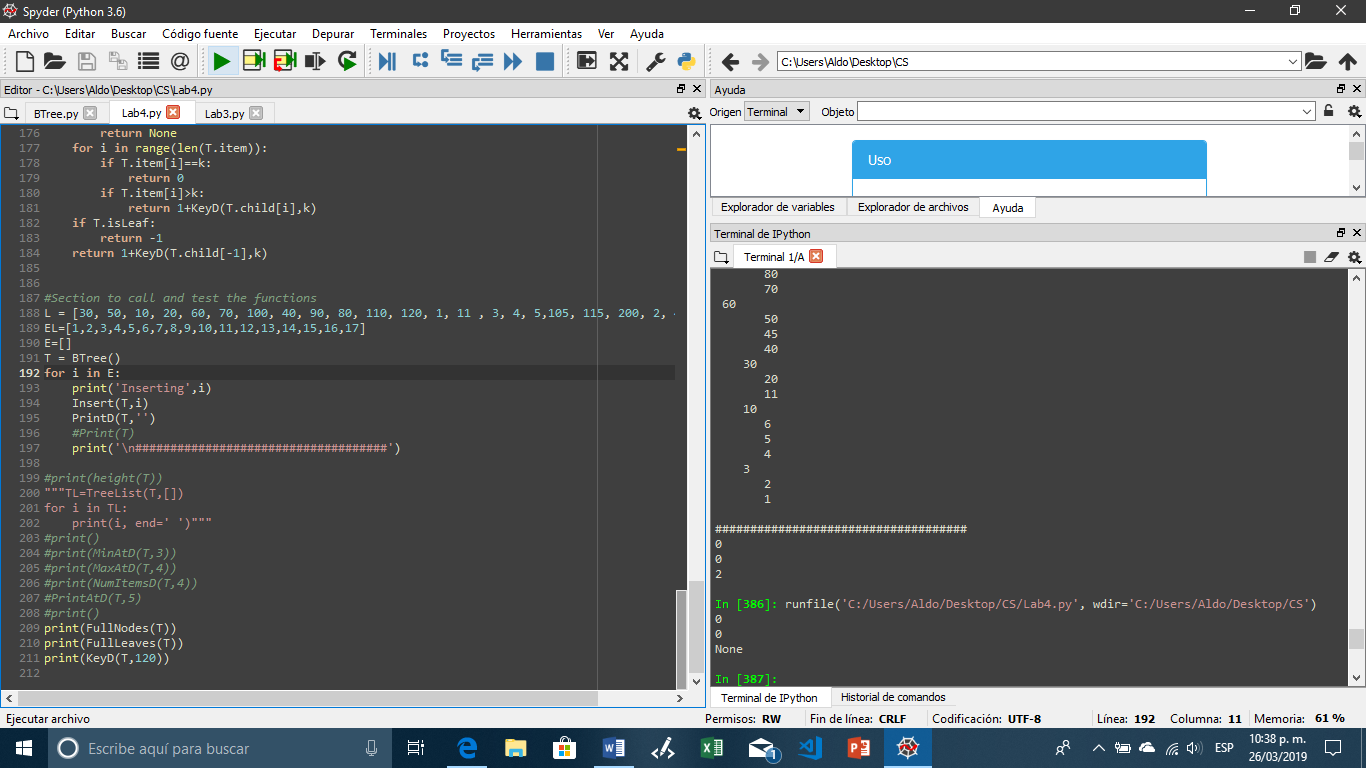
Input: Normal tree(d=1) Input: Index out of bounds(d=5) Input: Empty tree

Nothing is printed

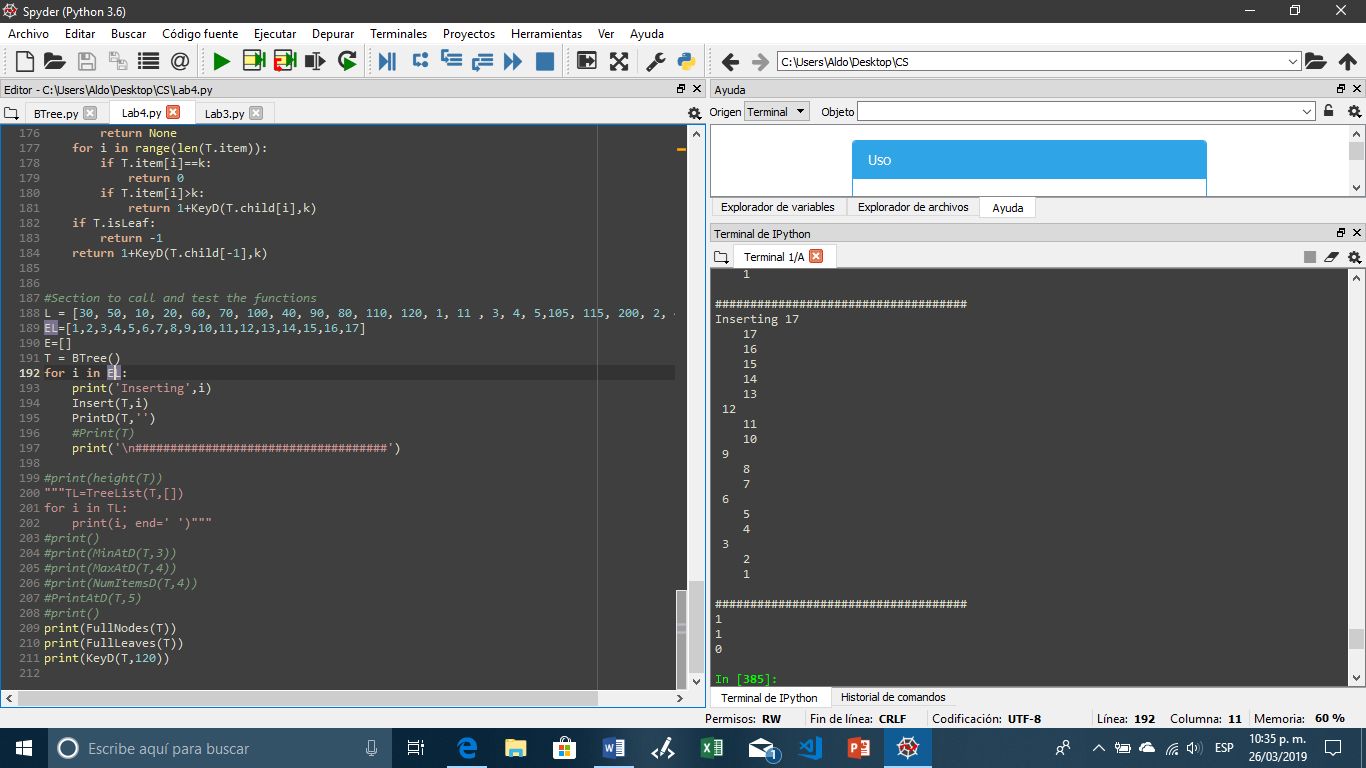
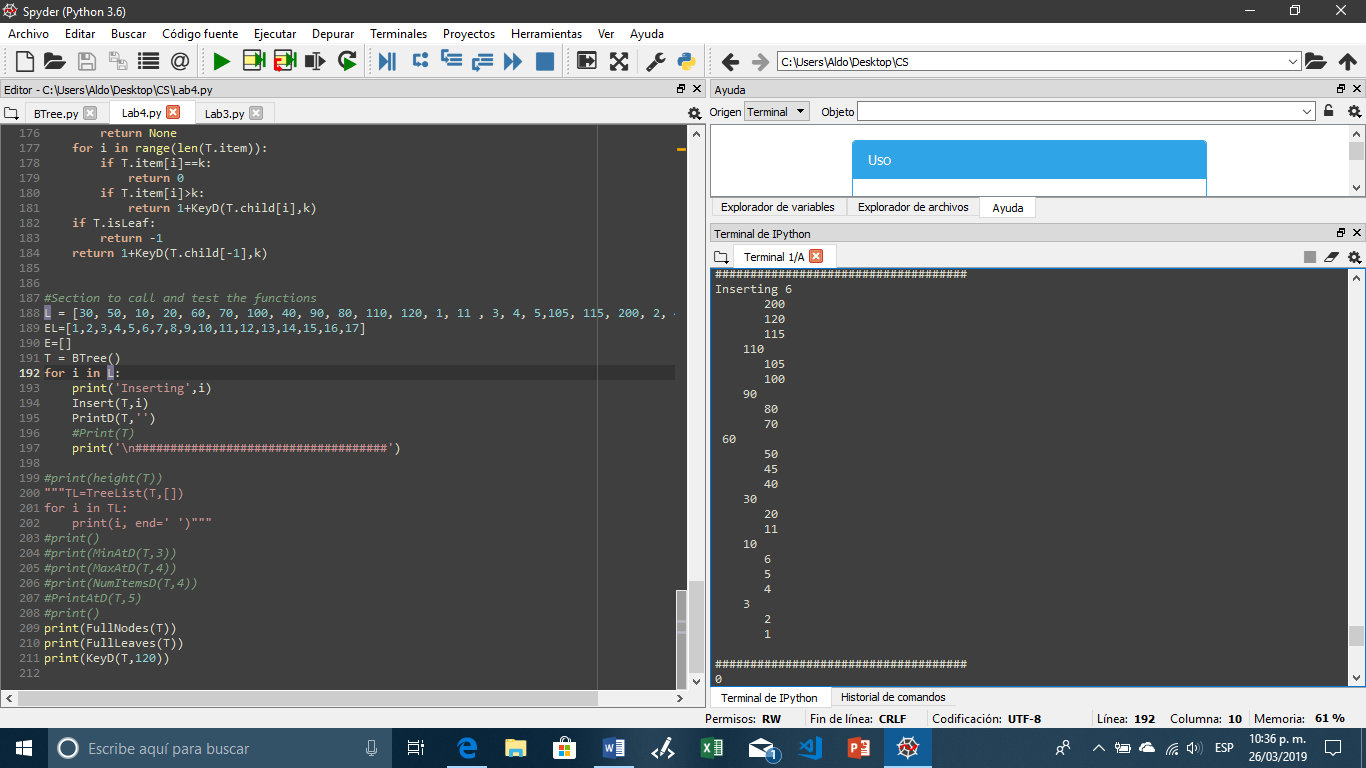
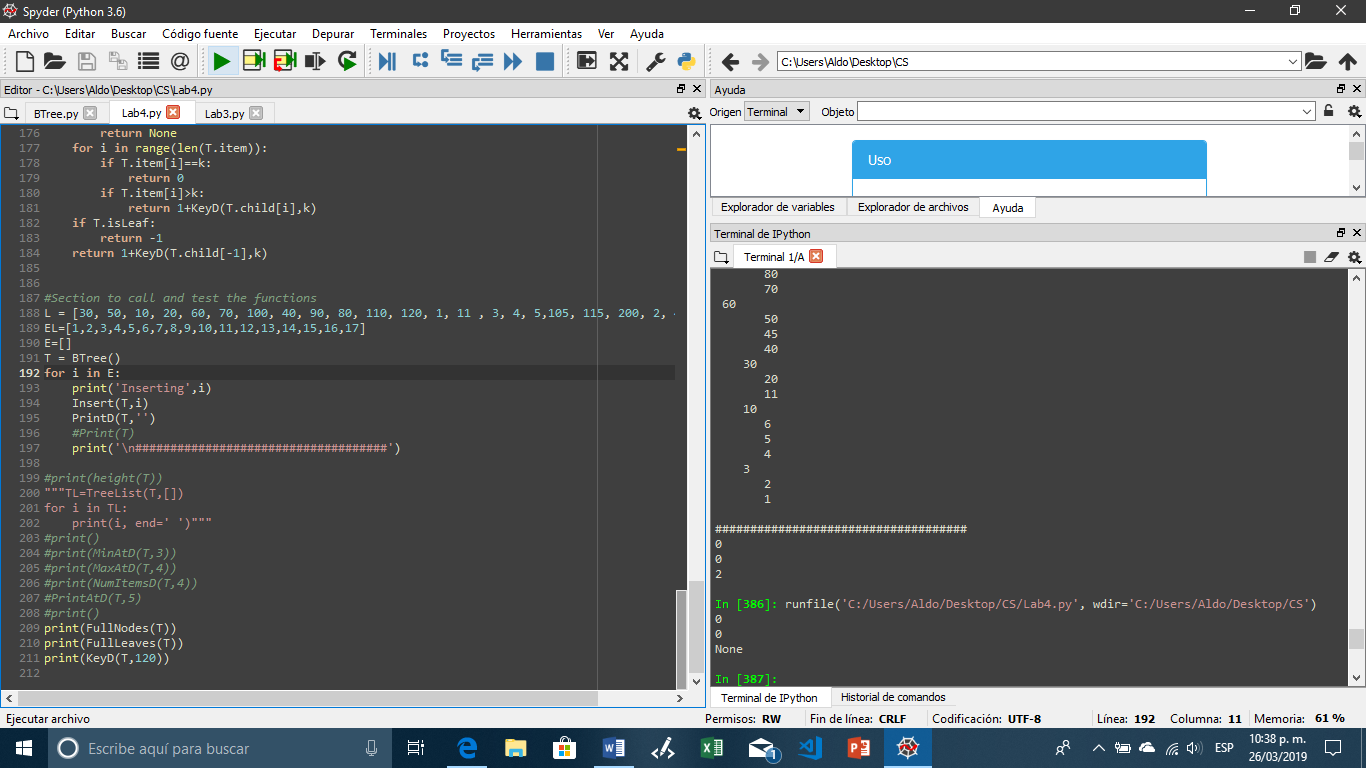
**Full nodes:** For this function we need to return the number of nodes that are full. One more time I am using a for loop to access every node and I am adding one to the final result. If the tree is empty, or if there are no full nodes inside the three, the value returned will be 0.

Input: Tree with full node Input: Tree without full nodes Input: Empty tree

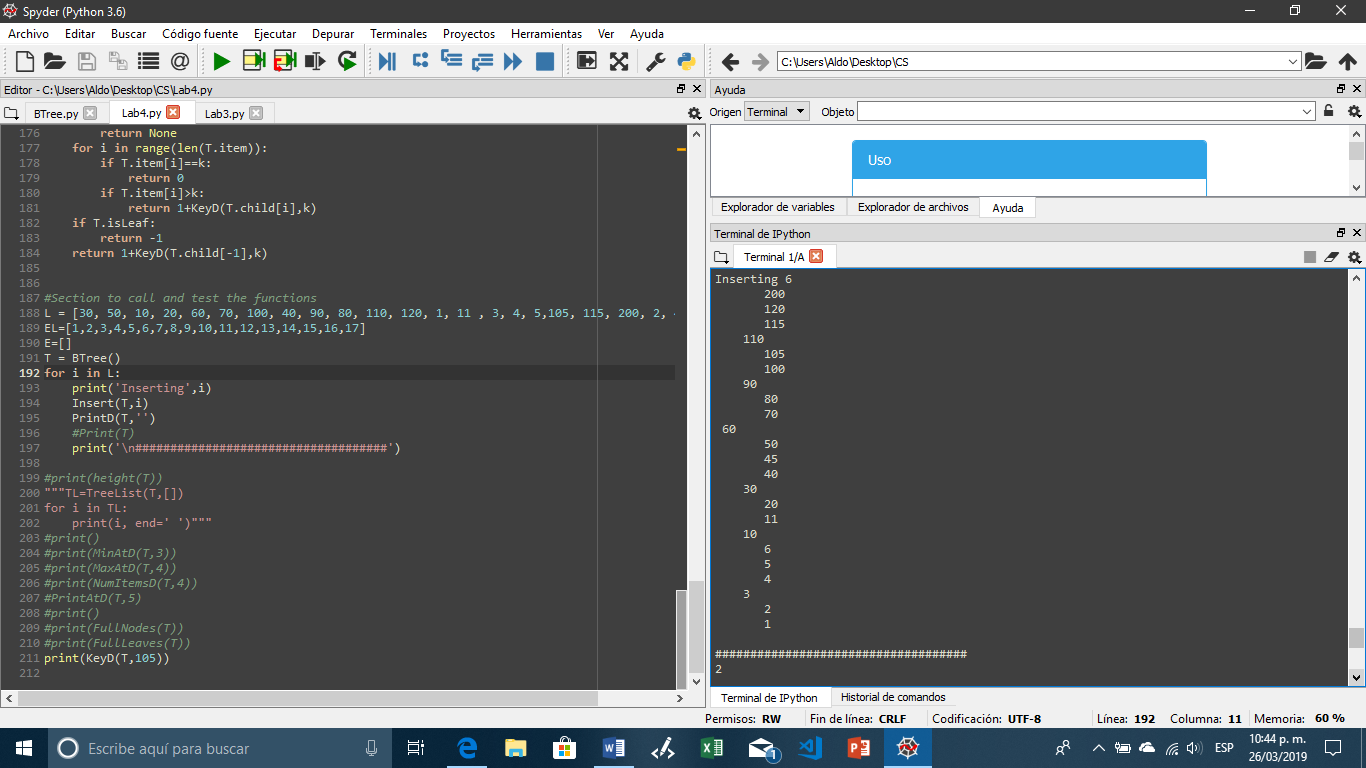
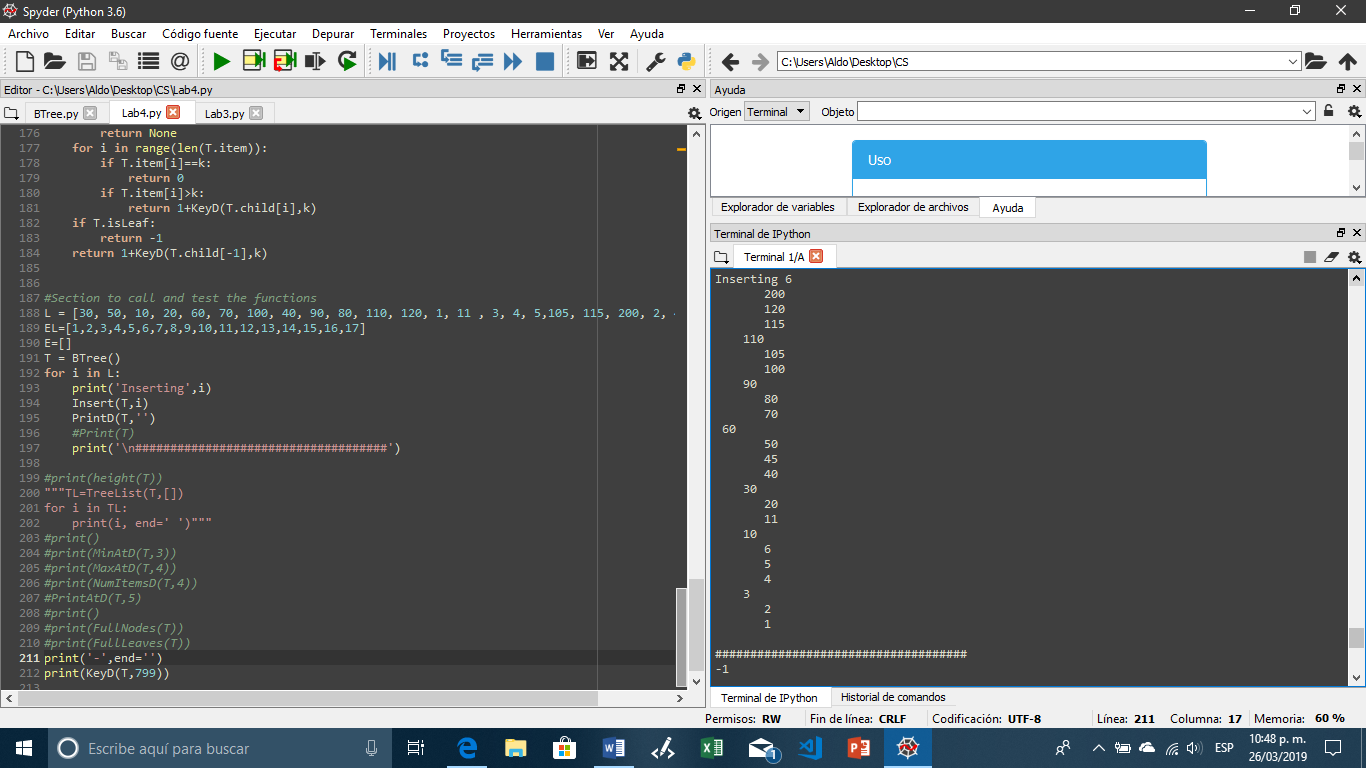
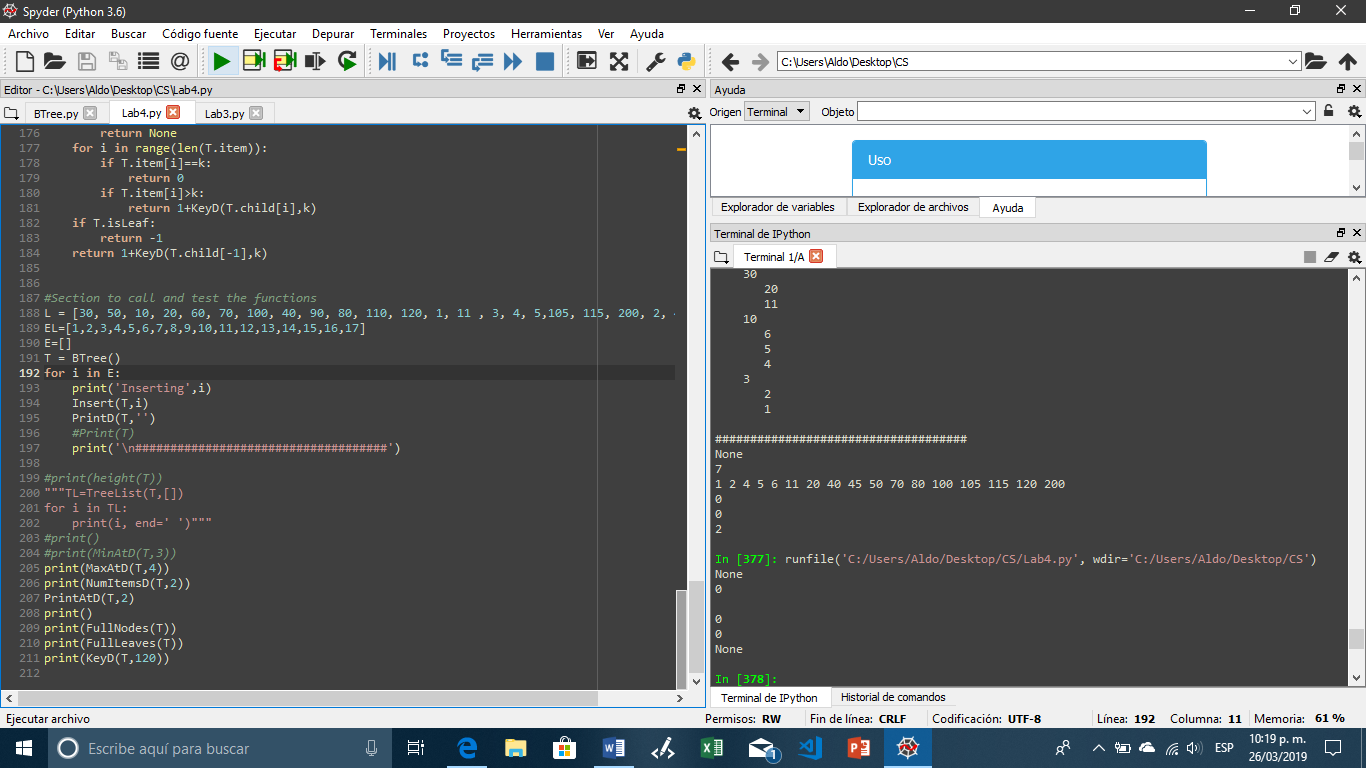
**Full Leaves:** For this function we need to return the number of leaves that are full. For this, we use a for loop, and the first thing that is checked is if the current node is a leaf. If it is a leaf we then check if it is full, if it is, we add one to the final result, if it is not, we move on. If the tree is empty, or if there are no full leaves the value returned will be 0.

Input: Tree with full leaf Input: Tree without full leaves Input: Empty tree

**Depth of key:** For this function we are given a key, and our task is to loop through the tree until we find it, and to return the depth at where it is found, or -1 if it is not found. Here we are going to check if the elements we are looking for is in the current node, if it is present, we return, but if it is not, we need to check if the elements we are looking for is bigger or smaller that the current element, so we can access and look in the right sub tree.

Input: K in the tree(k=105) Input: K not in tree(k=799) Input: Empty tree

After completing this lab, I have a better understanding of how a b-tree works, how to modify it and execute operations using them. And I can say I am able now to implement them if I need to do it in a future project.

#1 Function to obtain the height of the tree

def height(T):

if T.isLeaf:

return 0

return 1 + height(T.child[0])

#2 Function to store the items in the tree into a sorted list

def TreeList(T, A):

if T.isLeaf:

for i in range(len(T.item)):

A.append(T.item[i])

return

for i in range(len(T.item)):

TreeList(T.child[i],A)

A.append(T.item[i])

TreeList(T.child[len(T.item)],A)

return A

#3 Function to look for the smallest element at a given depth, returns none if the

#given depth does not exist

def MinAtD(T,d):

if len(T.item)<=0:

return None

if d==0:

return T.item[0]

if T.isLeaf:

return None

return MinAtD(T.child[0],d-1)

#4 Function to look for the biggest element at a given depth, returns none if the

#given depth does not exist

def MaxAtD(T,d):

if len(T.item)==0:

return None

if d==0:

return T.item[-1]

if T.isLeaf:

return None

return MaxAtD(T.child[-1],d-1)

#5 Function to return the number of elements at a given depth

def NumItemsD(T, d):

if d==0:

return 1

if T.isLeaf:

return 0

c=0

for i in range(len(T.child)):

c+=NumItemsD(T.child[i], d-1)

return c

#6 Function to print all the elements at a given depth

def PrintAtD(T,d):

if d==0:

for i in T.item:

print(i, end=' ')

for i in range(len(T.child)):

PrintAtD(T.child[i],d-1)

return

#7 Function to return the number of full nodes in a tree

def FullNodes(T):

if len(T.item)==0:

return 0

if len(T.item)==T.max\_items:

return 1

if T.isLeaf:

return 0

c=0

for i in range(len(T.item)):

c+=FullNodes(T.child[i])

c+=FullNodes(T.child[-1])

return c

#8 Function to return the number of full leaves in a tree

def FullLeaves(T):

if len(T.item)==0:

return 0

if len(T.item)==T.max\_items and T.isLeaf:

return 1

if T.isLeaf:

return 0

c=0

for i in range(len(T.item)):

c+=FullNodes(T.child[i])

c+=FullNodes(T.child[-1])

return c

#9 Function that returns the depth where a given element is found

def KeyD(T,k):

if len(T.item)==0:

return None

for i in range(len(T.item)):

if T.item[i]==k:

return 0

if T.item[i]>k:

return 1+KeyD(T.child[i],k)

if T.isLeaf:

return -1

return 1+KeyD(T.child[-1],k)

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

Aldo A. Venzor