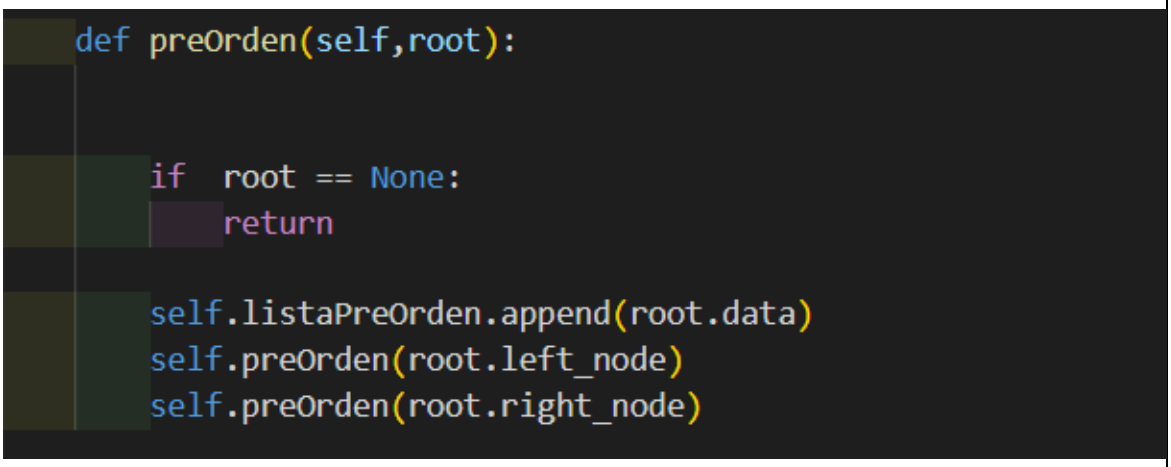
Taller 1
Arboles Binarios
Cristian David Gonzalez Castaño - 22582

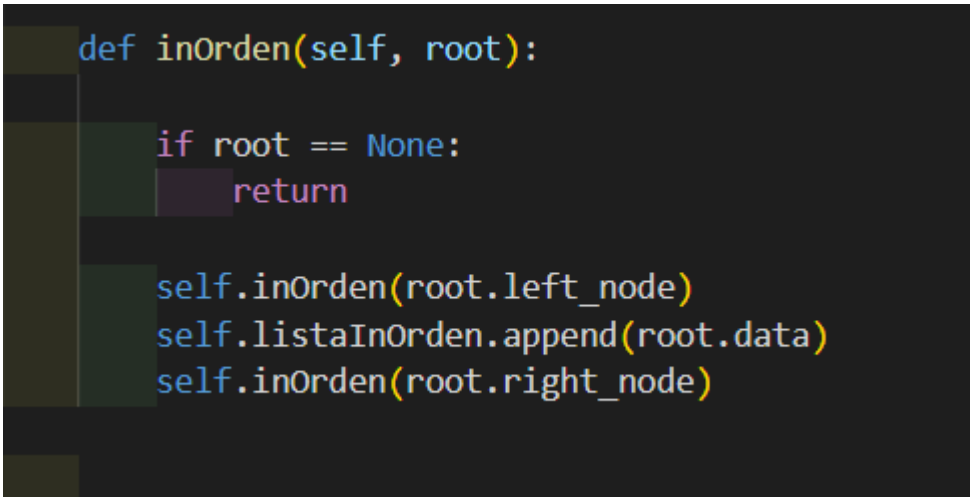
Problema 1
Escriba el método isComplete que verifique si el árbol binario es completo. Un árbol binario es completo si todos los niveles excepto el último están completamente llenos y en el último nivel todos los nodos están lo más a la izquierda posible.
Captura de pantalla de la solución (Código fuente)

```
def isComplete(self,root, level = 0):

    Altura = self.n_Altura(root)

    #si el arbol lleno viene un con true por lo tanto es completo
    if self.isFull(root) == True:
        return True


    else:

        #Me compueba que tenga hijo izquiero primero que todo
        if root.left_node != None and root.right_node == None:
            return True

        #Comprueba que esten en el ultimo level
        if root.left_node == None and root.right_node == None:
            return level == Altura - 1


        #Verifico que tenga ambos hijos , llamando nuevamente a la funcion
        #El level + 1  suma cada vez que se abra otra nueva ventana
        if (root.left_node != None and root.right_node != None):
            return (self.isComplete(root.left_node, level +1) and self.isComplete(root.right_node, level + 1))

        return False
```

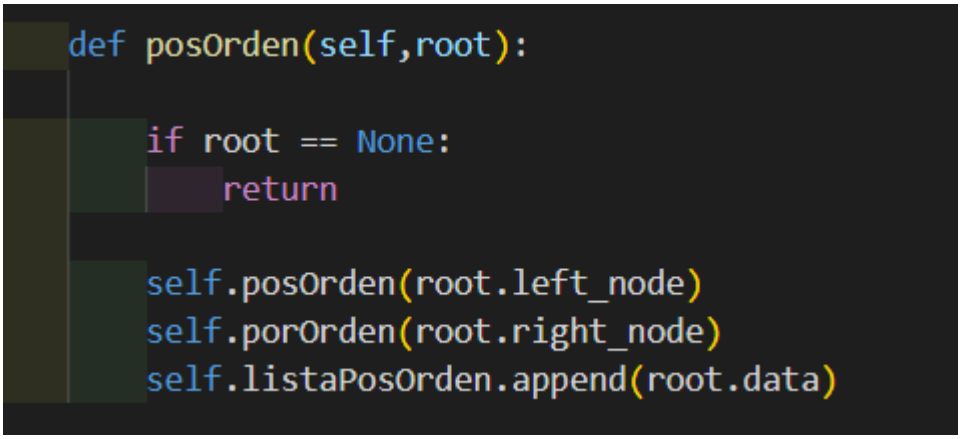
Problema 2
Escriba el método isFull que verifique si el árbol binario esta lleno. Un árbol binario se dice lleno si todo nodo (excepto las hojas) tiene dos hijos.
Captura de pantalla de la solución (Código fuente)

```
#Funcion la cual nos devuelve el numero de los nodos
def num_Nodos(self,root):

    if not root:
        return 0
    return 1 + self.num_Nodos(root.left_node) + self.num_Nodos(root.right_node)


#Funcion la cual nos retorna la altura del arbol
def n_Altura(self, root):
    if not root:
        return 0
    return 1 + max(self.n_Altura(root.left_node) , self.n_Altura(root.right_node))

# Un arbol es lleno si el numero de nodos es igual 2^H - 1 , donde H es altura , esta es la condicion para devolver un boolean
def isFull(self,root):
    if not root:
        return True

    numero_nodos = self.num_Nodos(root) #recursivo donde utilizamos las funciones para calcular altura y num de nodos
    altura_arbol = self.n_Altura(root)
    return numero_nodos == 2**altura_arbol - 1
```

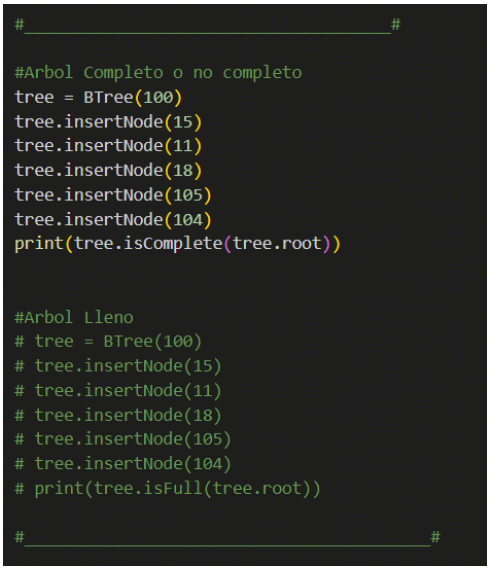
Problema 3
Implemente una método que realice el recorrido en profundidad de un árbol binario de búsqueda en preorden (raíz, izquierda, derecha).
Captura de pantalla de la solución (Código fuente)
1-Primero creamos una lista vacía, donde se va a guardar correctamente la data de los nodos  ``` def preOrden(self,root):   if root == None: return   self.listaPreOrden.append(root.data)   self.preOrden(root.left_node)   self.preOrden(root.right_node) ```

Problema 4
Implemente una método que realice el recorrido en profundidad de un árbol binario de búsqueda en inorden (izquierda, raíz, derecha).
Captura de pantalla de la solución (Código fuente)
1-Primero creamos una lista vacía, donde se va a guardar correctamente la data de los nodos 

```
def inOrden(self, root):

    if root == None:
        return

    self.inOrden(root.left_node)
    self.listaInOrden.append(root.data)
    self.inOrden(root.right_node)
```

Problema 5
Implemente una método que realice el recorrido en profundidad de un árbol binario de búsqueda en postorden (izquierda, derecha, raíz).
Captura de pantalla de la solución (Código fuente)
1-Primero creamos una lista vacía, donde se va a guardar correctamente la data de los nodos 

Comprobación en main



```
#_____________________________________#

#Arbol Completo o no completo
tree = BTree(100)
tree.insertNode(15)
tree.insertNode(11)
tree.insertNode(18)
tree.insertNode(105)
tree.insertNode(104)
print(tree.isComplete(tree.root))


#Arbol Lleno
# tree = BTree(100)
# tree.insertNode(15)
# tree.insertNode(11)
# tree.insertNode(18)
# tree.insertNode(105)
# tree.insertNode(104)
# print(tree.isFull(tree.root))

#_____________________________________#
```

```
#Comprobando recorridos

#Recorrido preOrden
# arbol= BTree(100)
# arbol.insertNode(15)
# arbol.insertNode(11)
# arbol.insertNode(18)
# arbol.insertNode(105)
# arbol.preOrden(arbol.root)
# print(arbol.listaPreOrden)

#Recorrido inOrden
# arbol= BTree(100)
# arbol.insertNode(15)
# arbol.insertNode(11)
# arbol.insertNode(18)
# arbol.insertNode(105)
# arbol.preOrden(arbol.root)
# print(arbol.listaInOrden)

#Recorrido posOrden
# arbol= BTree(100)
# arbol.insertNode(15)
# arbol.insertNode(11)
# arbol.insertNode(18)
# arbol.insertNode(105)
# arbol.preOrden(arbol.root)
# print(arbol.listaPosOrden)
```