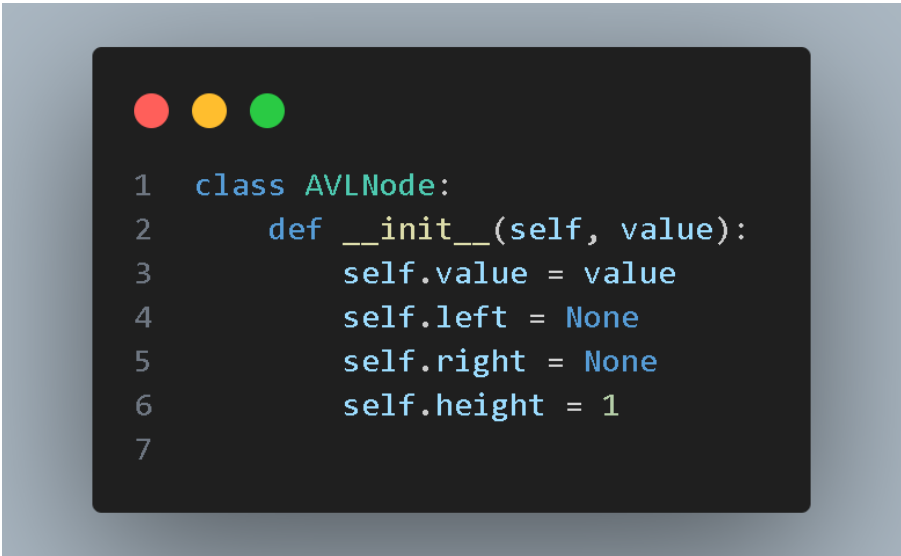


PRACTICA 2. TAD ÁRBOL (AVL)

Estructura de Datos

Para la realización de esta práctica e y definir el TAD Árbol en Python(3.10.8), implementé dos clases, "AVLNode" y "AVLTree".

La clase **AVLNode** contiene un constructor al cual le pasamos un parámetro, que se trata del objeto "espectador", del cual obtendremos el valor para equilibrar el árbol. Además de este parámetro, se definen una serie de atributos que corresponden: "height" a la altura que se encuentra el nodo, "left" a su hijo izquierdo, que es otro objeto de clase "AVLNode" pero que esta inicializado a "None", y "right" que será los mismo que el anterior, pero para definir su hijo derecho.



```
1 class AVLNode:
2     def __init__(self, value):
3         self.value = value
4         self.left = None
5         self.right = None
6         self.height = 1
7
```

Para la clase **AVLTree** definimos los siguientes métodos:

-**constructor __init__(self)**: este método se encarga de instanciar el objeto AVLtree, el cual no necesita parámetros que definimos el atributo root dentro de este, que será el elemento raíz del árbol.

-**height(self,node)**: este método devuelve la altura a la que se encuentra un nodo en el árbol, se le pasa como parámetro un nodo del árbol.

-**isEmpty(self)**: devuelve True o Falso en caso de que el árbol no tenga nodos.

- balance(self,node)**: calcula la diferencia de altura entre nodos hermanos y devuelve el resultado.
- rotate_right(self,nodo)**: realiza el intercambio del nodo desequilibrado con su nodo hijo izquierdo, también modifica las alturas de los nodos involucrados en el intercambio.
- rotate_left(self,nodo)**: realiza el intercambio del nodo desequilibrado con su nodo hijo derecho, también modifica las alturas de los nodos involucrados en el intercambio.
- insert(self,nodo)**: inserta el nodo en el árbol y una vez insertado comprueba la altura del árbol para equilibrarlo si es necesario mediante las funciones rotate_right y rotate_left.
- get_max(self,nodo)**: devuelve el nodo de la rama más a la derecha del árbol. Sirve para encontrar el nodo el valor más alto.
- borrar_nodo(self,value)**: busca el valor pasado dentro de los nodos del árbol, y si lo encuentra lo elimina de este, y una vez eliminado comprueba el balance del árbol, y si es necesario lo equilibra.
- search(self,value)**: recorre el árbol de forma recursiva comparando el valor pasado con el valor de los nodos de este, si lo encuentra un nodo con el mismo valor, devuelve el nodo en el que está, y si no, devuelve "None"
- get_random_node(self)**: guarda en una lista los nodos pasados por el método get_node_list(self,nodo), al cual se le pasa el nodo raíz, y luego de eso devuelve forma aleatoria un nodo de la lista.
- get_node_list(self,nodo)**: recorre un árbol desde la raíz guardando en una lista los nodos por los que pasa, y una vez finalizado el recorrido devuelve esta lista.
- inorder(self,curr_node,ids)**: recorre el árbol en orden de forma recursiva, y guarda cada valor del nodo (por ejemplo: node.espectador.id) por el que pasa en una lista, que devuelve una vez finalice el recorrido
- preorder(self,curr_node,ids)**: recorrd el árbol pre-orden de forma recursiva, y guarda cada valor del nodo (por ejemplo: node.espectador.id) por el que pasa en una lista, que devuelve una vez finalice el recorrido
- postorder(self,curr_node,ids)**: recorrd el árbol post-orden de forma recursiva, y guarda cada valor del nodo (por ejemplo: node.espectador.id) por el que pasa en una lista, que devuelve una vez finalice el recorrido
- bfs_trasversal(self)** (recorrido por niveles): recorrd el árbol por niveles de forma recursiva, y guarda cada valor del nodo (por ejemplo: node.espectador.id) por el que pasa en una lista, una vez finalizado imprime la lista de valores de nodos recorridos
- mostrarArbol(self,tipo_recorrido)**: contiene una llamada a los distintos métodos de recorrido del árbol según la opción que elija el usuario. Se encarga de imprimir las listas de valores de los métodos inorder,preorder y postorder.