**Algoritmos y Estructuras de Datos**

**LABORATORIO N° 09**

**Árboles Genéricos**

**CODIGO DEL CURSO:**

|  |  |  |
| --- | --- | --- |
| ***Alumno(s)*** | | ***Nota*** |
| ***Esteban Blas Aguilar Arosquipa*** | |  |
|  | |  |
|  | |  |
| ***Grupo*** | ***A*** | |
| ***Ciclo*** | ***III*** | |
| ***Fecha de entrega*** | ***13/05/2025*** | |

**I.- OBJETIVOS:**

* Comprender la estructura y lógica de los árboles.
* Construir algoritmos implementando la estructura de árboles utilizando clases y objetos.

**II.- SEGURIDAD:**

|  |  |
| --- | --- |
|  | **Advertencia:**  **En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red; así como la ingestión de alimentos o bebidas.** |

**III.- FUNDAMENTO TEÓRICO:**

* Revisar el texto guía que está en el campus Virtual.

**IV.- NORMAS EMPLEADAS:**

* No aplica

**V.- RECURSOS:**

* En este laboratorio cada alumno trabajará con un equipo con Windows 10.

**VI.- METODOLOGÍA PARA EL DESARROLLO DE LA TAREA:**

* El desarrollo del laboratorio es individual.

**VII.- PROCEDIMIENTO:**

**EJERCICIO DE APLICACIÓN**

1.- Representar el siguiente árbol genérico

class NodoArbol:

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

self.valor = valor

self.hijos = [] # Lista de hijos, que son otros NodoArbol

def agregar\_hijo(self, nuevo\_hijo):

"""Agrega un hijo al nodo actual"""

self.hijos.append(nuevo\_hijo)

def eliminar\_hijo\_por\_valor(self, valor\_hijo):

"""Elimina el hijo con el valor indicado"""

for i, hijo in enumerate(self.hijos):

if hijo.valor == valor\_hijo:

del self.hijos[i]

return True

return False

def buscar(self, valor):

"""Busca un nodo por valor usando recorrido DFS"""

if self.valor == valor:

return self

for hijo in self.hijos:

resultado = hijo.buscar(valor)

if resultado:

return resultado

return None

def imprimir(self, nivel=0):

"""Imprime el árbol con indentación para visualizar estructura"""

print(" " \* nivel + f"- {self.valor}")

for hijo in self.hijos:

hijo.imprimir(nivel + 1)

Ejecución

# Crear raíz

raiz = NodoArbol("A")

# Agregar hijos a la raíz

raiz.agregar\_hijo(NodoArbol("B"))

raiz.agregar\_hijo(NodoArbol("C"))

raiz.agregar\_hijo(NodoArbol("D"))

# Agregar nietos al nodo B

nodo\_b = raiz.buscar("B")

nodo\_b.agregar\_hijo(NodoArbol("E"))

nodo\_b.agregar\_hijo(NodoArbol("F"))

# Mostrar árbol

print("Árbol antes de eliminar:")

raiz.imprimir()

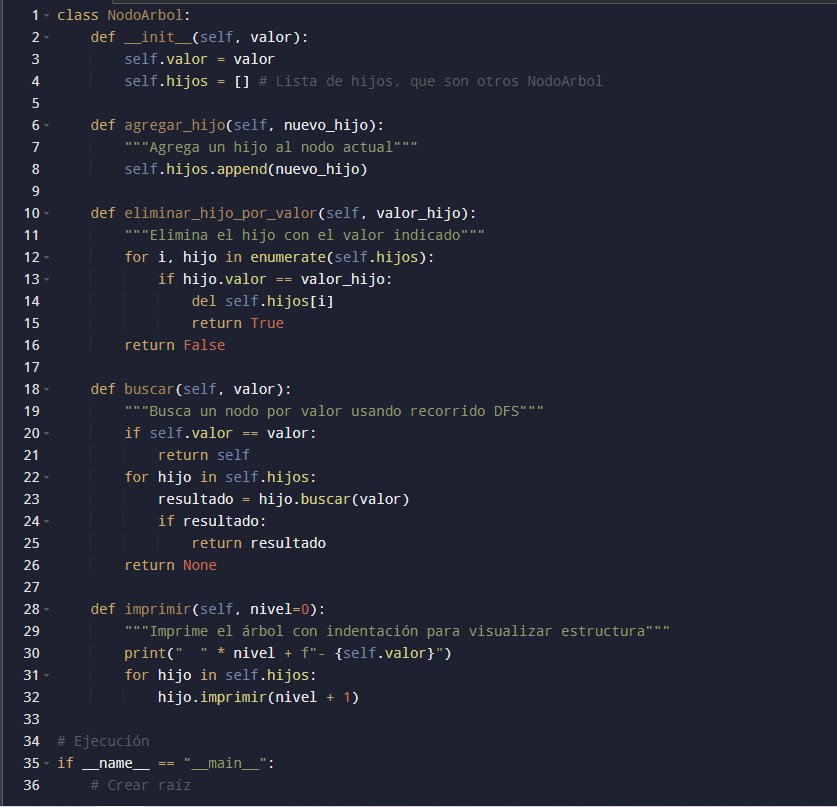
# Eliminar hijo "F" de "B"

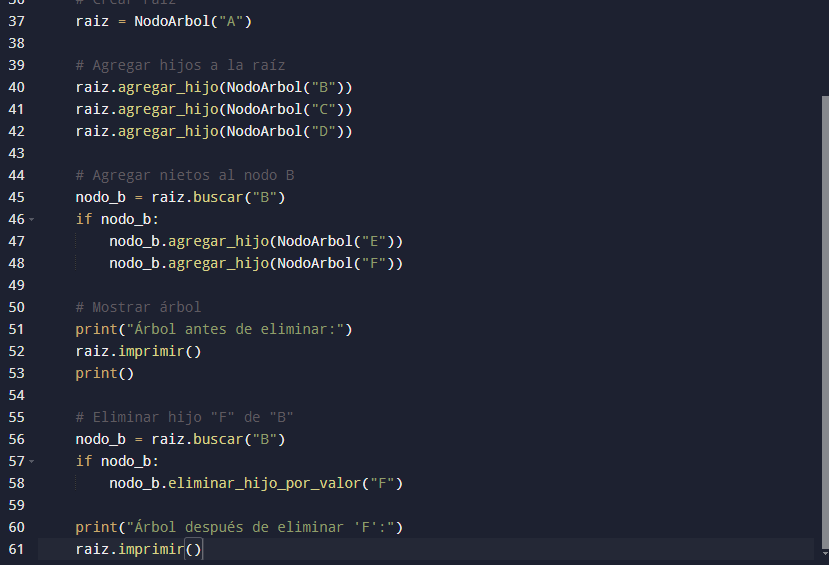
nodo\_b.eliminar\_hijo\_por\_valor("F")

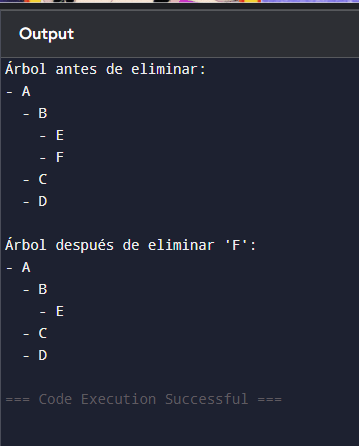
print("\nÁrbol después de eliminar 'F':")

raiz.imprimir()

EJECUCION







Ejercicios

- Definición de un árbol genérico con listas enlazadas.

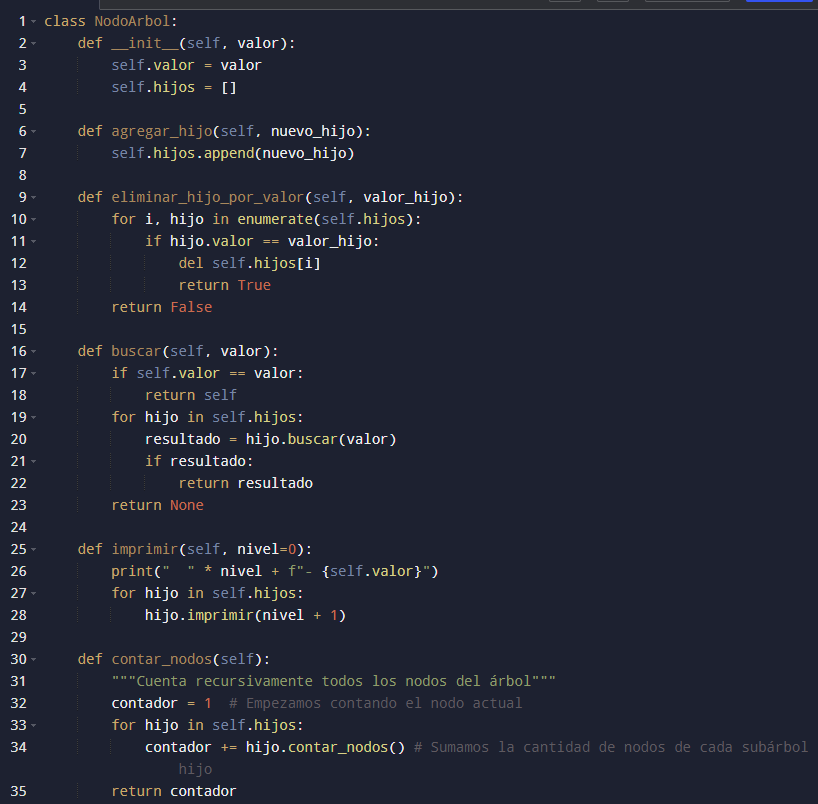
- Implementar una función de eliminar para evitar que se eliminen nodos si estos tienen hijos.

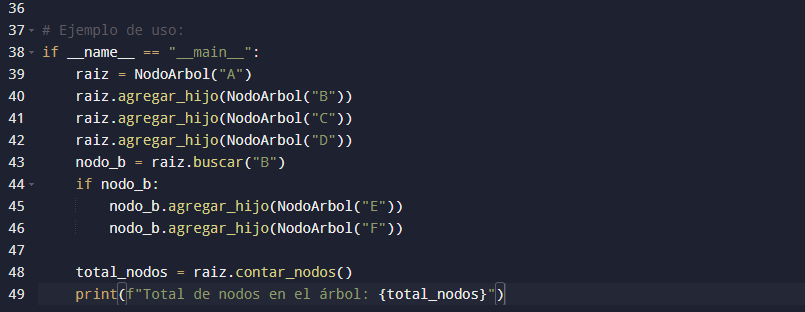
Desafios:

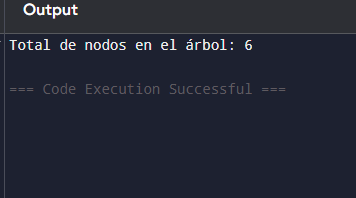
1. Contar todos los nodos del árbol

Objetivo: Crear una función que cuente cuántos nodos hay en total (incluyendo la raíz y todos los hijos).

Pista: Usar recursividad y sumar 1 + total de los hijos.



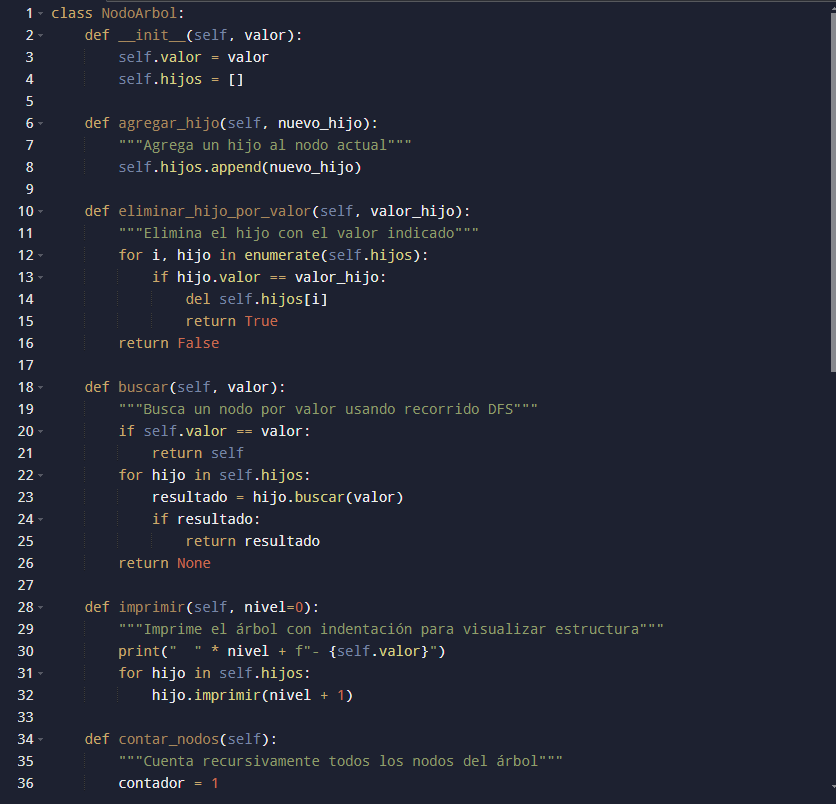


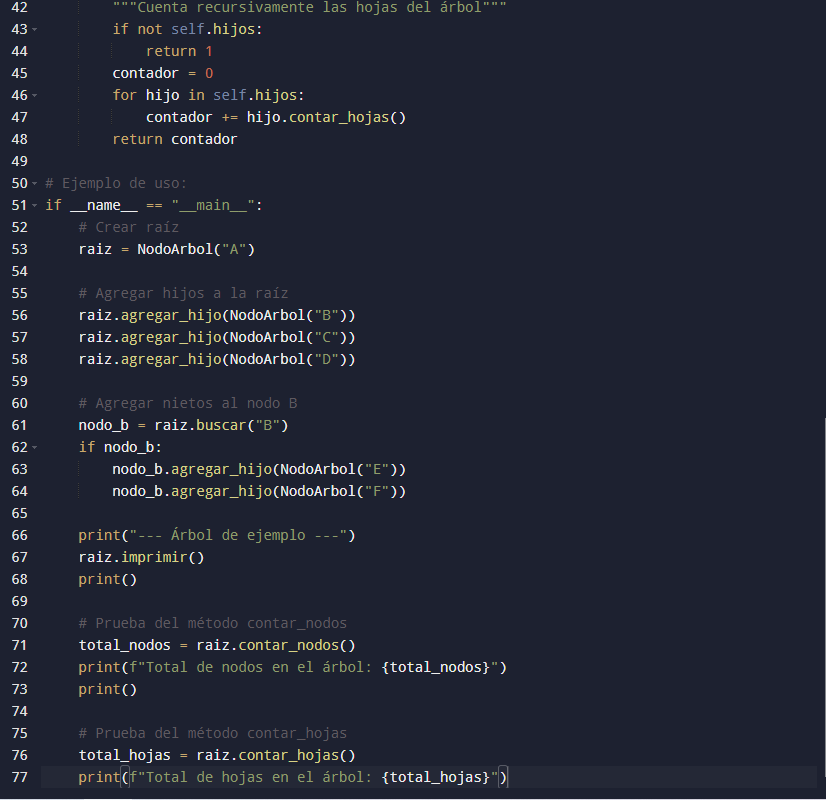


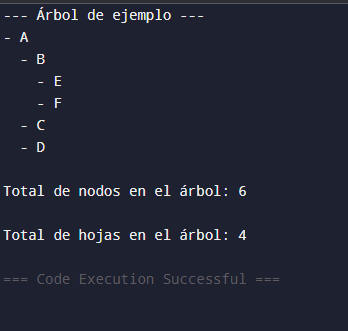
2. Contar todas las hojas del árbol

Objetivo: Crear una función que cuente solo los nodos que no tienen hijos (hojas).

Pista: Si el nodo no tiene hijos, cuenta como 1.



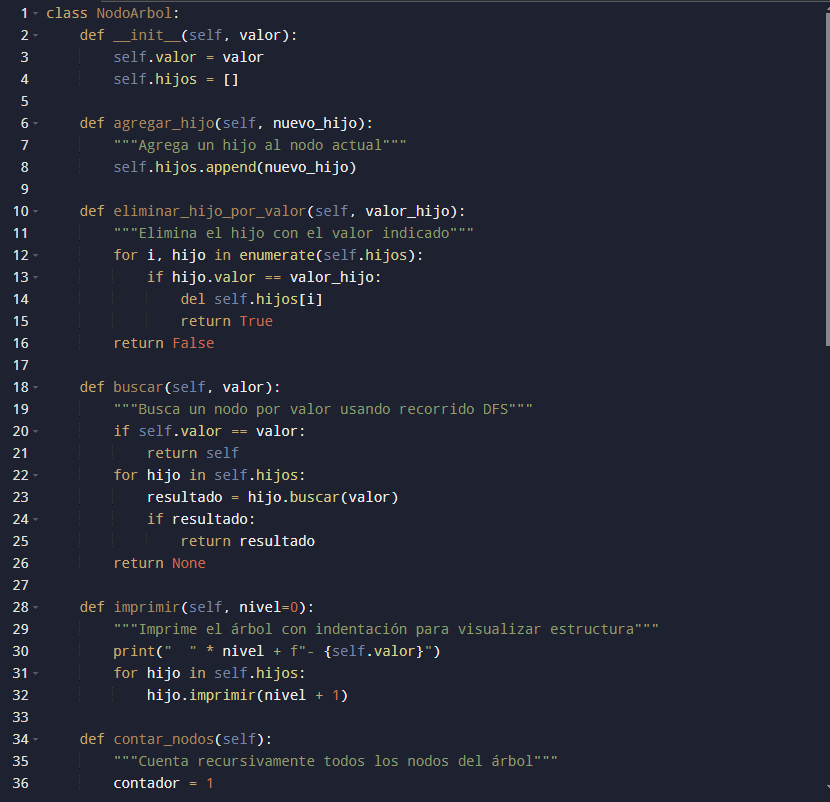


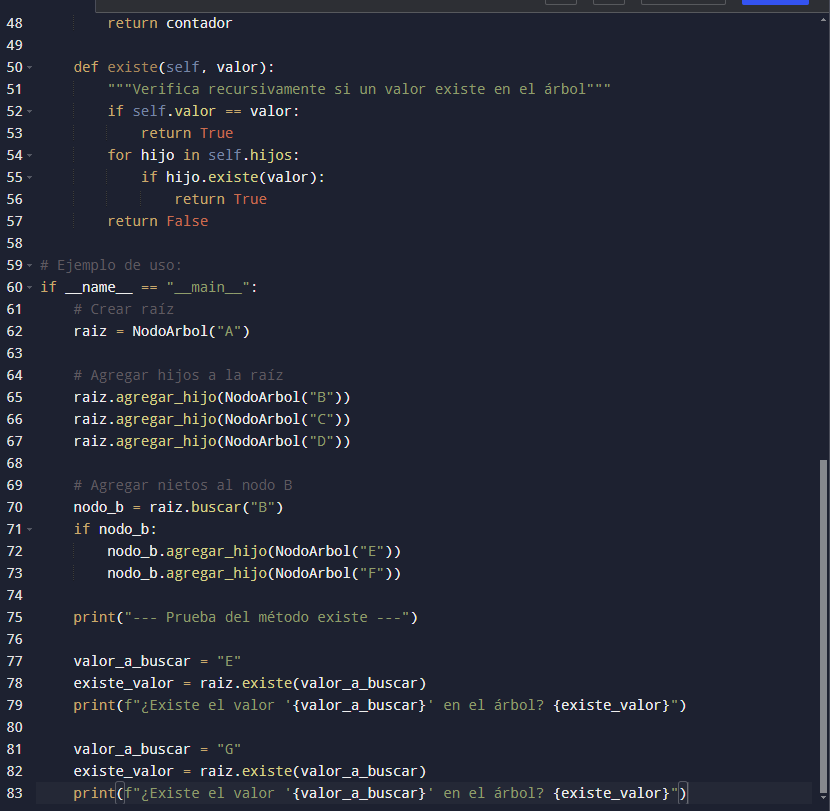


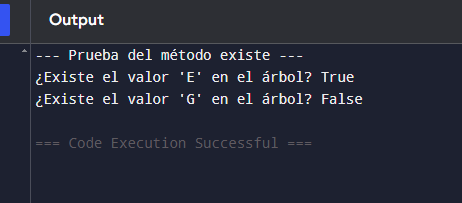
3. Verificar si un valor existe en el árbol

Objetivo: Retornar True o False si un nodo con cierto valor existe.

Pista: Similar a la función buscar, pero devuelve solo un booleano.



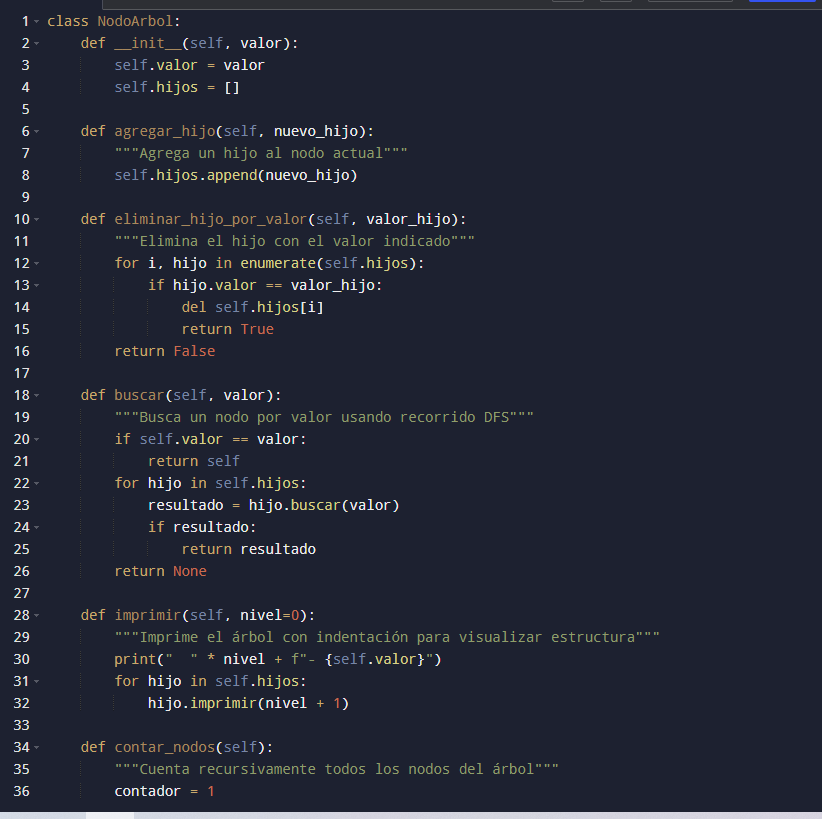


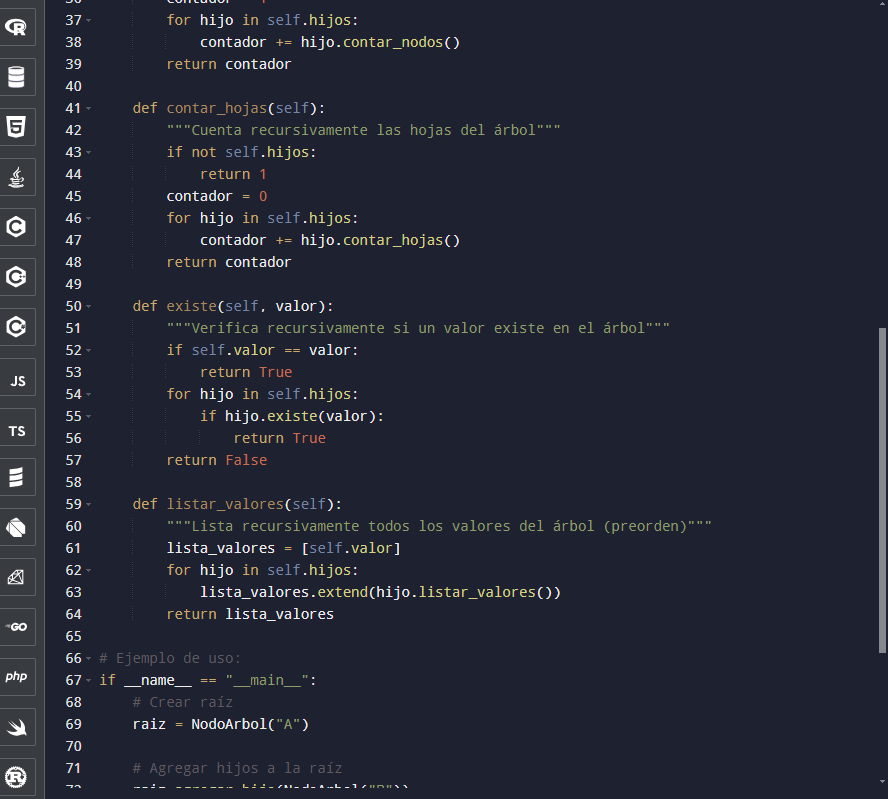


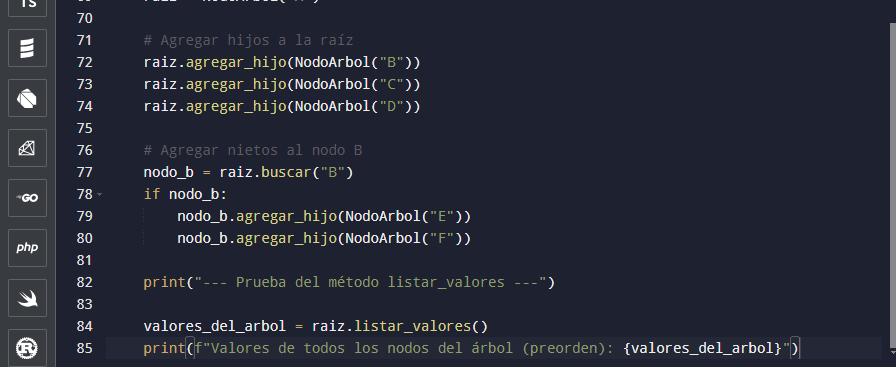
4. Listar todos los valores del árbol

Objetivo: Generar una lista con todos los valores del árbol (preorden, postorden o cualquier forma sencilla).

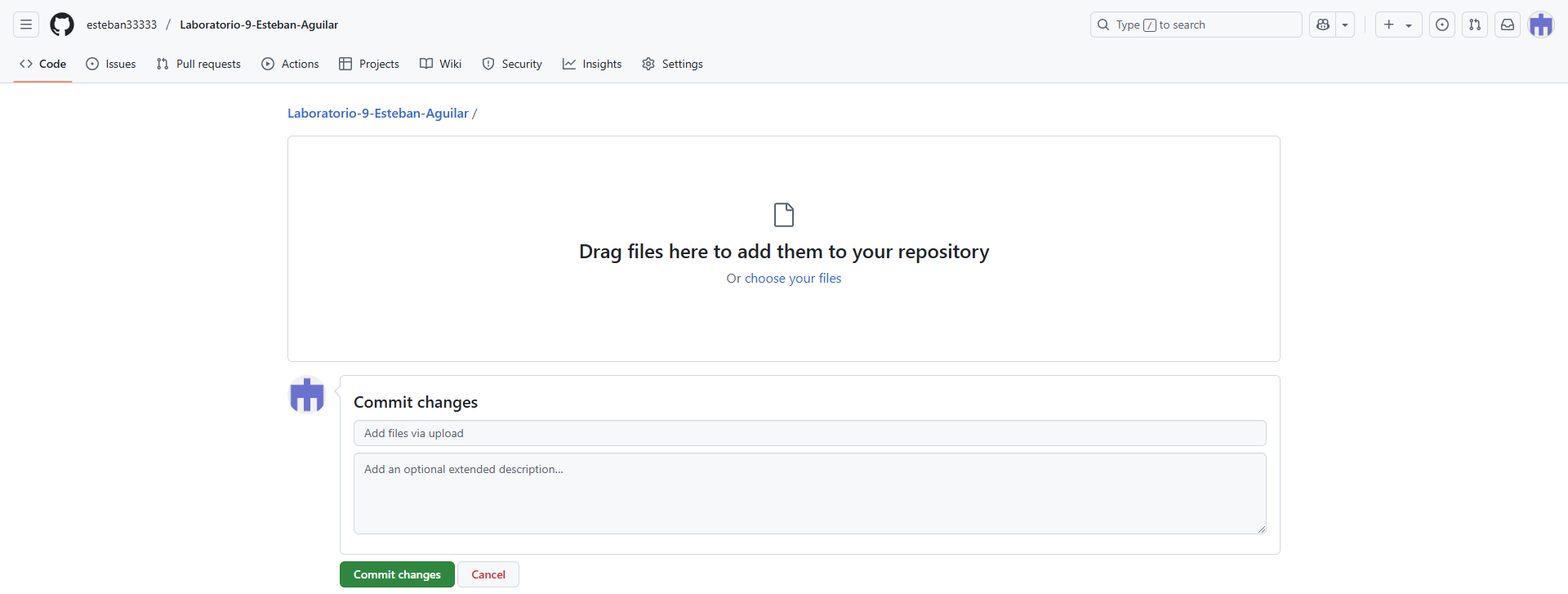
Pista: Usar acumuladores o listas y recursividad.







5. Subir mi código a github



**CONCLUSIONES:**

**La recursividad es una herramienta poderosa para trabajar con estructuras de datos jerárquicas como los árboles.** Muchos de los métodos implementados (contar nodos, contar hojas, buscar, verificar existencia, listar valores) se benefician enormemente de la recursión, permitiendo una solución elegante y concisa para recorrer y operar sobre todos los elementos del árbol.

**La estructura de nodos enlazados es fundamental para representar árboles genéricos.** Cada NodoArbol contiene un valor y una lista de referencias a sus hijos, lo que permite una flexibilidad significativa en la cantidad de hijos que cada nodo puede tener, diferenciándolo de estructuras más rígidas como los árboles binarios.

**La implementación de operaciones básicas en un árbol genérico sienta las bases para funcionalidades más complejas.** Con la capacidad de agregar, eliminar, buscar y recorrer nodos, se pueden construir algoritmos más avanzados para tareas como la búsqueda de caminos, la representación de jerarquías organizacionales, o la manipulación de datos estructurados.

**El manejo de errores y la depuración son partes esenciales del desarrollo de software.** Los errores que surgieron durante el proceso (como el AttributeError) resaltan la importancia de verificar cuidadosamente el código, asegurarse de estar ejecutando la versión correcta y comprender los mensajes de error para identificar y corregir los problemas de manera efectiva.

**La representación visual de la estructura del árbol (a través del método imprimir) es crucial para comprender su organización y verificar la correcta implementación de las operaciones.** La indentación ayuda a entender las relaciones padre-hijo y facilita la detección de posibles errores en la construcción o manipulación del árbol.