Trabajo Integrador – Programación I

**Datos Generales**

Título del trabajo: Árboles Binarios con Listas en Python: Simulador Interactivo de Menú Jerárquico

Alumno: Diego Pavón – diego.pavon@tupad.utn.edu.ar

Materia: Programación I

Profesora: Julieta Trapé

Fecha de Entrega: 9 de Junio del 2025

**Índice**

1. Introducción

2. Marco Teórico

3. Caso Práctico

4. Metodología Utilizada

5. Resultados Obtenidos

6. Conclusiones

7. Bibliografía

8. Anexos

**1. Introducción**

El presente trabajo se centra en la implementación de árboles binarios utilizando listas en el lenguaje Python. Se eligió este tema por su importancia en la organización jerárquica de datos, especialmente útil en búsquedas, navegación de menús, y estructuras dinámicas.

El objetivo principal es comprender cómo se puede modelar y manipular un árbol utilizando listas anidadas. Se desarrolló además una versión interactiva que simula un menú jerárquico, lo cual permite visualizar y modificar el árbol en tiempo real desde consola.

**2. Marco Teórico**

Un árbol es una estructura de datos no lineal compuesta por nodos conectados entre sí. El nodo principal se llama raíz, y cada nodo puede tener cero o más hijos. Un árbol binario es un tipo particular donde cada nodo tiene como máximo dos hijos: uno izquierdo y uno derecho.

Características fundamentales:

- Raíz: nodo principal del árbol.

- Hojas: nodos sin hijos.

- Altura: cantidad de niveles desde la raíz hasta la hoja más profunda.

- Subárboles: cada nodo con sus descendientes forma un subárbol.

Representación en Python:

[nombre\_opcion, subarbol\_izquierdo, subarbol\_derecho]

Esta forma de modelado es simple y adecuada para introducir conceptos como recursividad y estructuras dinámicas.

**3. Caso Práctico**

Se desarrolló un sistema interactivo que permite:

- Crear un árbol con raíz "Menú Principal"

- Agregar nodos a izquierda o derecha

- Buscar nodos por nombre

- Imprimir el árbol visualmente rotado 90°

- Recorrer el árbol (preorden, inorden, postorden)

Ejemplo de código base:

import unicodedata

def crear\_opcion(nombre):

return [nombre, [], []]

def insertar\_izquierda(nodo, nueva\_opcion):

nodo[1] = [nueva\_opcion, nodo[1], []] if nodo[1] else [nueva\_opcion, [], []]

def insertar\_derecha(nodo, nueva\_opcion):

nodo[2] = [nueva\_opcion, [], nodo[2]] if nodo[2] else [nueva\_opcion, [], []]

def imprimir\_arbol(arbol, nivel=0):

if arbol:

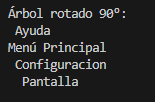
imprimir\_arbol(arbol[2], nivel + 1)

print(" " \* nivel + arbol[0])

imprimir\_arbol(arbol[1], nivel + 1)

El sistema incluye un menú por consola donde el usuario puede interactuar con el árbol dinámicamente.

Captura de ejemplo (estructura del árbol):



**4. Metodología Utilizada**

- Se investigaron implementaciones de árboles en Python sin clases.

- Se siguió un enfoque incremental: primero las funciones básicas (crear, insertar, recorrer), luego la interfaz interactiva.

- Se utilizó Python 3.x en entorno local (VS Code) para pruebas.

- Las pruebas fueron realizadas directamente desde consola.

**5. Resultados Obtenidos**

- Se logró representar un árbol binario jerárquico utilizando listas anidadas.

- El menú interactivo permite recorrer el árbol y modificarlo dinámicamente.

- El árbol se visualiza correctamente rotado, y los recorridos funcionan como se espera.

**6. Conclusiones**

El trabajo permitió comprender de manera práctica cómo funcionan los árboles binarios, cómo recorrerlos y cómo modelarlos sin necesidad de clases. Se valoró especialmente la visualización y la navegación dinámica.

Este enfoque es ideal para estudiantes como nosotros que están comenzando con estructuras de datos. Las mejoras futuras pueden incluir validación de nombres y exportación del árbol a archivo

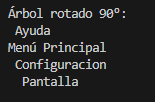
**7. Bibliografía**

- UTN Material Didáctico (2025). Módulos 5, 6 y 7.

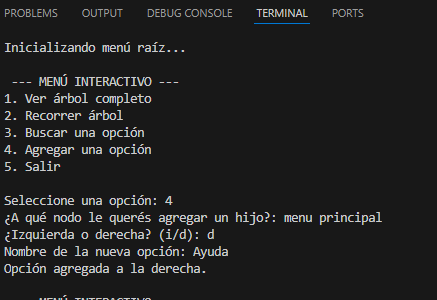
- Python Software Foundation. (2024). Python 3 Documentation. https://docs.python.org/3/

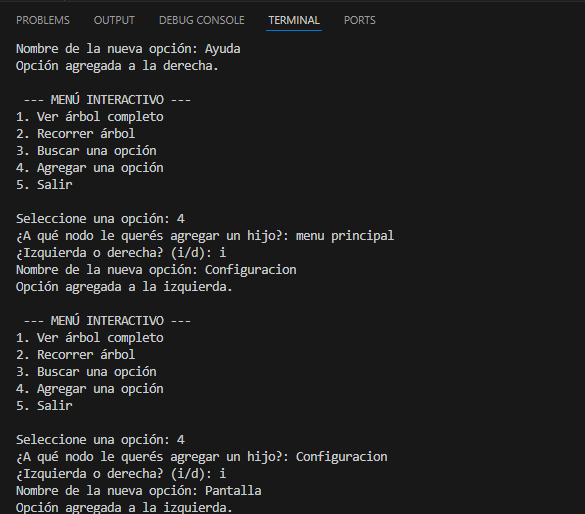
**8. Anexos**

- Captura del árbol rotado.



- Fragmentos de ejecución en consola.





- Código fuente completo en archivo adjunto.

# Importamos la libreria necesaria para normalizar los strings

import unicodedata

# Funciones para manejo del arbol con listas

def crear\_opcion(nombre):

    return [nombre, [], []]

def insertar\_izquierda(nodo, nueva\_opcion):

    subarbol\_izq = nodo[1]

    if subarbol\_izq:

        nodo[1] = [nueva\_opcion, subarbol\_izq, []]

    else:

        nodo[1] = [nueva\_opcion, [], []]

def insertar\_derecha(nodo, nueva\_opcion):

    subarbol\_der = nodo[2]

    if subarbol\_der:

        nodo[2] = [nueva\_opcion, [], subarbol\_der]

    else:

        nodo[2] = [nueva\_opcion, [], []]

def imprimir\_arbol(arbol, nivel=0):

    if arbol:

        imprimir\_arbol(arbol[2], nivel + 1)

        print(' ' \* nivel + str(arbol[0]))

        imprimir\_arbol(arbol[1], nivel + 1)

def preorden(arbol):

    if arbol:

        print(arbol[0])

        preorden(arbol[1])

        preorden(arbol[2])

def inorden(arbol):

    if arbol:

        inorden(arbol[1])

        print(arbol[0])

        inorden(arbol[2])

def postorden(arbol):

    if arbol:

        postorden(arbol[1])

        postorden(arbol[2])

        print(arbol[0])

def normalizar(texto):  # Convierte el texto a minúsculas, elimina tildes y espacios innecesarios.

    texto = texto.strip().lower()

    texto = unicodedata.normalize('NFD', texto)

    texto = ''.join(c for c in texto if unicodedata.category(c) != 'Mn')

    return texto

def buscar\_opcion(arbol, nombre):

    if not arbol:

        return None

    if normalizar(arbol[0]) == normalizar(nombre):

        return arbol

    encontrado = buscar\_opcion(arbol[1], nombre)

    if encontrado:

        return encontrado

    return buscar\_opcion(arbol[2], nombre)

# Interfaz de usuario

def menu\_interactico():

    print('Inicializando menú raíz...')

    menu = crear\_opcion('Menú Principal')

    while True:

        print('\n --- MENÚ INTERACTIVO --- ')

        print('1. Ver árbol completo')

        print('2. Recorrer árbol')

        print('3. Buscar una opción')

        print('4. Agregar una opción')

        print('5. Salir')

        opcion = input('\nSeleccione una opción: ')

        if opcion == '1':

            print('\nÁrbol rotado 90°:')

            imprimir\_arbol(menu)

        elif opcion == '2':

            print('\nTipos de recorrido:')

            print('a. Preorden')

            print('b. Inorden')

            print('c. Postorden')

            tipo = input('\nElige el tipo de recorrido: (a/b/c): ')

            print('\nResultado:')

            if tipo == 'a':

                preorden(menu)

            elif tipo == 'b':

                inorden(menu)

            elif tipo == 'c':

                postorden(menu)

            else:

                print('Opción no válida.')

        elif opcion == '3':

            nombre = input('Ingrese el nombre de la opción a buscar: ')

            resultado = buscar\_opcion(menu, nombre)

            if resultado:

                print('Opción encontrada:', resultado[0], '!')

            else:

                print('Opción no encontrada.')

        elif opcion == '4':

            padre = input('¿A qué nodo le querés agregar un hijo?: ')

            lado = input('¿Izquierda o derecha? (i/d): ').lower()

            nuevo\_nombre = input('Nombre de la nueva opción: ')

            nodo\_padre = buscar\_opcion(menu, padre)

            if nodo\_padre:

                if lado == 'i':

                    insertar\_izquierda(nodo\_padre, nuevo\_nombre)

                    print('Opción agregada a la izquierda.')

                elif lado == 'd':

                    insertar\_derecha(nodo\_padre, nuevo\_nombre)

                    print('Opción agregada a la derecha.')

                else:

                    print('Dirección inválida.')

            else:

                print('No se encontró el nodo padre.')

        elif opcion == '5':

            print('Saliendo del programa...')

            break

        else:

            print('Opción no válida. Intente nuevamente.')

# Ejecutar el menú

menu\_interactico()