# ****Trabajo Integrador – Programación I****

**Alumno:** Kenyi Meza,   
**Correo electrónico institucional:** mezakenyi@gmail.com  
**Materia:** Programación I  
**Profesor:** Julieta Trapé  
**Fecha de entrega:** 09/06/2025  
**Título del trabajo:** Implementación de Árboles Binarios en Python Usando Listas

## 1. Introducción

Las estructuras de datos son fundamentales en el desarrollo de software, ya que permiten organizar y procesar información de manera eficiente. En este trabajo se aborda la implementación de árboles binarios utilizando exclusivamente listas en Python, sin recurrir a clases ni objetos. Esta elección facilita la comprensión estructural del árbol binario.

El objetivo principal es demostrar que, con herramientas simples como listas anidadas, es posible construir, recorrer y visualizar un árbol binario funcional. A su vez, este enfoque permite ejercitar conceptos claves como la recursividad, el diseño de funciones y la representación jerárquica de datos.

## 2. Marco Teórico

### ¿Qué es un árbol?

Un árbol es una estructura de datos **no lineal** compuesta por nodos conectados jerárquicamente. El nodo superior se denomina **raíz**, y cada nodo puede tener cero o más **hijos**. En el caso del **árbol binario**, cada nodo puede tener a lo sumo dos hijos: uno izquierdo y uno derecho.

**Elementos importantes:**

* **Raíz:** nodo principal.
* **Hojas:** nodos sin hijos.
* **Altura:** número de niveles del árbol.
* **Subárbol:** cualquier nodo con sus descendientes.

### Recorridos clásicos de árboles binarios:

* **Inorden:** izquierda → raíz → derecha.
* **Preorden:** raíz → izquierda → derecha.
* **Postorden:** izquierda → derecha → raíz.

### Representación con listas en Python

En vez de usar clases, se representa cada nodo como una lista de tres elementos:

“[nodo, subárbol\_izquierdo, subárbol\_derecho]”

Esto permite trabajar con estructuras jerárquicas sin necesidad de orientación a objetos, centrando la atención en la lógica de construcción y navegación del árbol.

## 3. Caso Práctico

A continuación se presenta el código fuente del árbol binario, acompañado de comentarios explicativos.

# Esta funcion crea un nuevo arbol con un nodo principal (raiz)

# El nodo es una lista con el valor y dos lugares vacios para los hijos izquierdo y derecho

def crear\_arbol(valor):

return [valor, [], []]

# Esta funcion agrega un nodo a la izquierda del nodo actual

# Si ya hay algo a la izquierda, lo baja un nivel y pone el nuevo nodo arriba

def insertar\_izquierda(nodo, valor):

if nodo[1]:

nodo[1] = [valor, nodo[1], []]

else:

nodo[1] = [valor, [], []]

# Esta funcion hace lo mismo que la anterior pero para el lado derecho

def insertar\_derecha(nodo, valor):

if nodo[2]:

nodo[2] = [valor, [], nodo[2]]

else:

nodo[2] = [valor, [], []]

# Recorre el arbol en preorden: primero muestra el nodo, despues va a la izquierda y despues a la derecha

def recorrido\_preorden(nodo):

if nodo:

print(nodo[0], end=' ')

recorrido\_preorden(nodo[1])

recorrido\_preorden(nodo[2])

# Recorre el arbol en inorden: primero va a la izquierda, luego muestra el nodo, y despues va a la derecha

def recorrido\_inorden(nodo):

if nodo:

recorrido\_inorden(nodo[1])

print(nodo[0], end=' ')

recorrido\_inorden(nodo[2])

# Recorre el arbol en postorden: primero izquierda, luego derecha, y al final el nodo

def recorrido\_postorden(nodo):

if nodo:

recorrido\_postorden(nodo[1])

recorrido\_postorden(nodo[2])

print(nodo[0], end=' ')

# Esta funcion muestra el arbol rotado 90 grados hacia la izquierda

# El lado derecho del arbol aparece arriba, el izquierdo abajo

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

if nodo:

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

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

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

# Parte principal del programa: aca armamos el arbol y probamos las funciones

if \_\_name\_\_ == "\_\_main\_\_":

# Creamos el nodo raiz con valor 'A'

arbol = crear\_arbol('A')

# Agregamos hijos a izquierda y derecha

insertar\_izquierda(arbol, 'B')

insertar\_derecha(arbol, 'C')

# Agregamos nodos al segundo nivel

insertar\_izquierda(arbol[1], 'D')

insertar\_derecha(arbol[1], 'E')

insertar\_izquierda(arbol[2], 'F')

insertar\_derecha(arbol[2], 'G')

# Mostramos los recorridos

print("Recorrido Preorden:")

recorrido\_preorden(arbol)

print("\nRecorrido Inorden:")

recorrido\_inorden(arbol)

print("\nRecorrido Postorden:")

recorrido\_postorden(arbol)

# Mostramos el arbol rotado

print("\n\nVisualizacion del arbol rotado:")

imprimir\_arbol(arbol)

### Ejecución esperada:

Recorrido Preorden:

A B D E C F G

Recorrido Inorden:

D B E A F C G

Recorrido Postorden:

D E B F G C A

Visualización del árbol rotado:

G

C

F

A

E

B

D

## 4. Metodología Utilizada

* Estudio del material de clase provisto por la cátedra (presentaciones y rúbricas).
* Implementación paso a paso en Python 3.11 bajo debian12.
* Desarrollo de funciones recursivas para construcción y recorridos.
* Prueba del código en vscode.
* Publicación del código en GitHub.

## 5. Resultados Obtenidos

* El árbol binario fue construido correctamente usando listas.
* Se ejecutaron con éxito los recorridos inorden, preorden y postorden.
* La visualización rotada permitió observar la jerarquía del árbol de forma clara.
* El código funciona correctamente y está documentado con comentarios en cada función.

## 6. Conclusiones

El desarrollo de este trabajo permitió aplicar la lógica de estructuras de datos en un entorno práctico, reforzando la comprensión de la recursividad y las jerarquías. El uso de listas en lugar de clases facilitó el enfoque didáctico del problema.

Este enfoque es ideal para quienes se inician en programación, ya que reduce la complejidad sintáctica y permite concentrarse en la lógica estructural. Como extensión futura, se podrían agregar funciones de búsqueda, eliminación o balanceo.

## 7. Bibliografía

* Material de clase.
* YouTube: https://www.youtube.com/watch?v=d0ibZK\_6Q7g

## 8. Anexos

* Enlace al repositorio de GitHub: <https://github.com/ya-awn/arboles-en-python>
* Video explicativo :

