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
```

```
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)

#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.

---

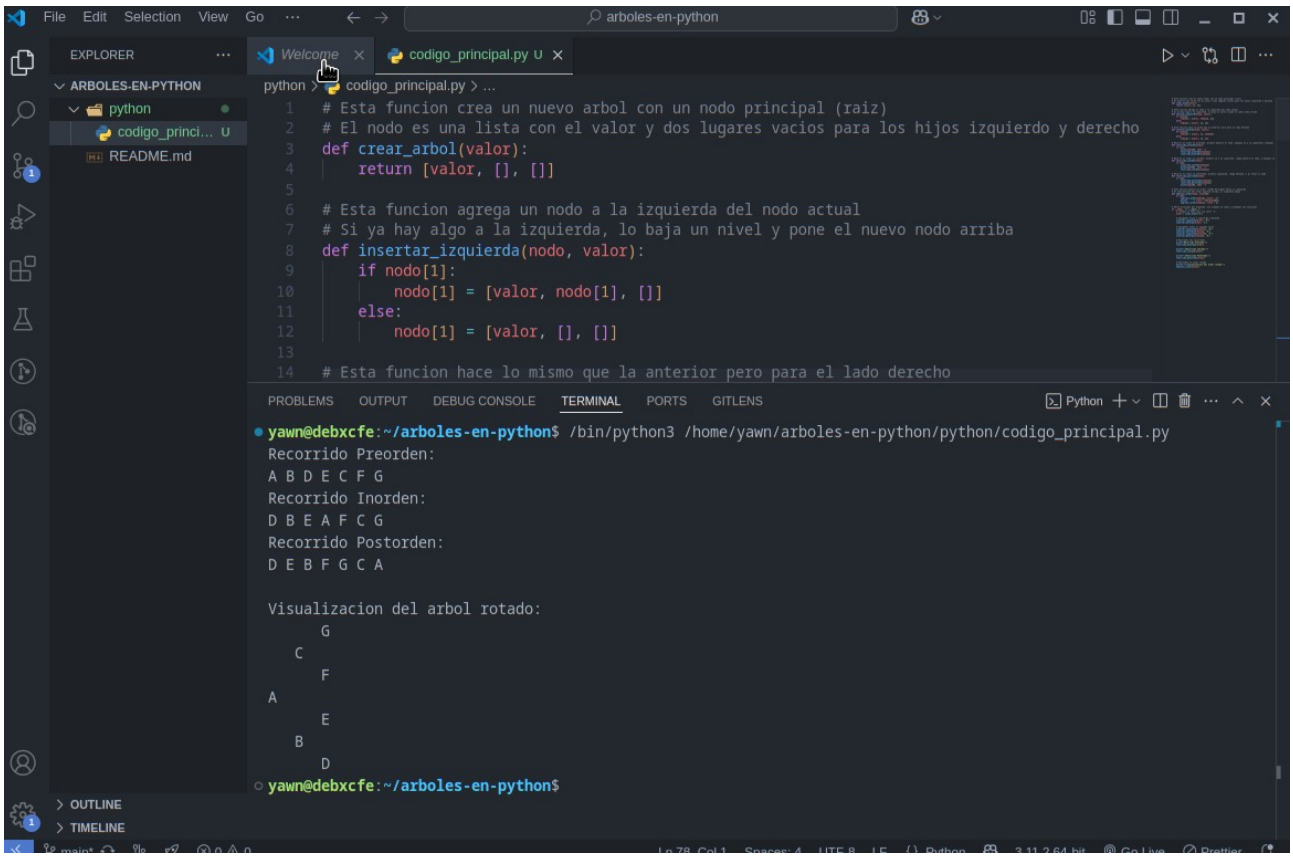
7. Bibliografía

- 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 : https://youtu.be/EYz3A9IDjlg


File Edit Selection View Go
arboles-en-python
EXPLORER
Welcome
codigo_principal.py U
ARBOLES-EN-PYTHON
python
codigo_princi... U
README.md
python > codigo_principal.py > ...
# 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
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
Python
yawn@debxcfe:~/arboles-en-python$ /bin/python3 /home/yawn/arboles-en-python/python/codigo_principal.py
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
Visualizacion del arbol rotado:
G
C
F
A
E
B
D
yawn@debxcfe:~/arboles-en-python$
OUTLINE
TIMELINE