



Proyecto Final – Programación I (Tecnicatura Universitaria en Programación a Distancia)

Este trabajo integrador es una **propuesta de investigación aplicada** que busca combinar teoría, desarrollo práctico en Python y difusión del conocimiento.



Objetivo General


Realizar una investigación **teórico-práctica** sobre temas fundamentales y avanzados en Python. Deben combinarse:

- **Marco teórico**
 - **Caso práctico en Python**
 - **Análisis y conclusiones**
 - **Difusión del trabajo (Git y video)**
-



Temas posibles para investigar

Los grupos pueden elegir uno o más de estos ejes temáticos:

-  **Algoritmos de búsqueda y ordenamiento**
 - **Análisis de algoritmos (eficiencia y optimización)**
 - **Estructuras de datos avanzadas (ej. árboles)**
-



Entregables obligatorios

Cada grupo de **dos personas** debe presentar:

1. Carpeta de investigación (formato digital)

- **Marco teórico** con citas y fuentes.
- **Caso práctico en Python** (funcional y relacionado con el tema).

- **Conclusiones** del equipo.

2. Repositorio Git (público o accesible para docentes)

- **Código fuente funcional y documentado**
- **README** con:
 - Descripción del proyecto
 - Instrucciones de uso
 - Reflexiones del equipo

3. Video tutorial (10 a 15 minutos)

- Presentación del tema y la experiencia
- Demostración del desarrollo en Python
- Explicación del proceso de trabajo
- Reflexiones finales

Requisitos mínimos

- **Marco teórico** con referencias.
- **Caso práctico** que funcione de verdad (ej. código que ordene datos y mida su eficiencia).
- **Repositorio en GitHub** o similar, accesible.
- **Video** con participación activa de ambos integrantes.

Metodología sugerida (paso a paso)

1. Elegir el tema.
2. Investigar fuentes confiables.

3. Redactar marco teórico.
 4. Diseñar y programar el caso práctico.
 5. Hacer pruebas y documentar.
 6. Subir a GitHub.
 7. Grabar el video.
 8. Reflexionar y escribir conclusiones.
-

Criterios de evaluación

- Profundidad y claridad del marco teórico.
- Creatividad y relevancia del proyecto en Python.
- Calidad del código (organización, limpieza, buenas prácticas).
- Buen uso de Git.
- Calidad y claridad del video.

Tema elegido: Estructuras de datos avanzadas (Ej. árboles)

Ideas para el marco teórico

Incluí conceptos como:

- ¿Qué es una estructura de datos?
- ¿Qué es un árbol? Tipos más comunes:
 - Árbol binario
 - Árbol binario de búsqueda (BST)
 - Árbol AVL (auto-balanceado)
 - Árbol Trie

- Ventajas y casos de uso en el mundo real (bases de datos, compresión, inteligencia artificial, etc.)
- Comparación de árboles con otras estructuras (listas, pilas, colas)

Fuentes sugeridas:

- Documentación oficial de Python
- Libros como "Estructuras de Datos y Algoritmos en Python"
- Artículos académicos o de Medium, RealPython, GeeksForGeeks

Ideas para el caso práctico en Python

Algunas ideas posibles:

- Implementar un **árbol binario de búsqueda (BST)** con inserción, búsqueda y recorrido (inorden, preorden, postorden)
- Medir el rendimiento en comparación con una lista
- Visualizar cómo se estructura el árbol (con texto o usando librerías como [graphviz](#))
- Opcional: Implementar AVL para mostrar balanceo automático

Ideas para el repositorio Git

- Estructura clara de carpetas
- README completo:
 - Descripción del árbol implementado
 - Instrucciones para correr el código
 - Reflexión: ¿Qué aprendieron sobre estructuras de datos?
- Buen uso de [commits](#), ramas si quieren experimentar

Ideas para el video

- Breve explicación teórica con ejemplos visuales
- Demostración en vivo: cómo insertan, buscan, recorren el árbol
- Qué aprendieron, dificultades, cómo las resolvieron
- Recomendación: usen gráficos o dibujos para explicar la estructura del árbol

MARCO TEÓRICO — *Estructuras de Datos* *Avanzadas: Árboles*

Te dejo una versión base que podés modificar y ampliar según lo que vos y tu compañero/a quieran incluir:

♦ ¿Qué es una estructura de datos?

Una **estructura de datos** es una forma organizada de almacenar, acceder y modificar información en un programa. Algunas estructuras simples son listas, pilas y colas. Las estructuras avanzadas, como los **árboles**, permiten representar relaciones jerárquicas de forma eficiente.

♦ ¿Qué es un árbol?

Un **árbol** es una estructura de datos no lineal que consiste en **nodos conectados jerárquicamente**. Cada nodo puede tener hijos, y hay un único nodo raíz desde donde parte toda la estructura.

Un árbol típico tiene:

- **Raíz:** el nodo principal.
- **Padre e hijos:** cada nodo puede tener subnodos (hijos).
- **Hojas:** nodos sin hijos.
- **Altura:** número de niveles del árbol.

♦ Tipos comunes de árboles

1. **Árbol Binario**

Cada nodo tiene como máximo dos hijos (izquierdo y derecho).

2. **Árbol Binario de Búsqueda (BST)**

Un árbol binario en el que:

- Los valores menores al nodo actual van a la izquierda.
- Los valores mayores, a la derecha.
Esto permite realizar búsquedas, inserciones y eliminaciones con eficiencia logarítmica.

3. **Árbol AVL**

Un tipo de BST **auto-balanceado**, que ajusta su estructura para que la diferencia de altura entre subárboles sea como máximo 1.

4. **Árbol Trie (o prefijo)**

Utilizado para almacenar cadenas, como palabras en un diccionario, de forma eficiente.

♦ Casos de uso en el mundo real

- **Sistemas de archivos:** carpetas y subcarpetas funcionan como un árbol.
- **Búsqueda en bases de datos:** los índices suelen estar basados en árboles B o B+.
- **Inteligencia artificial:** los árboles de decisión son esenciales para machine learning.
- **Compiladores:** el análisis sintáctico se realiza usando árboles de derivación.

♦ Ventajas de usar árboles

- Permiten organizar datos jerárquicamente.
- Búsquedas, inserciones y eliminaciones son eficientes (en BST o AVL: $O(\log n)$).
- Son esenciales para algoritmos complejos y estructuras de alto rendimiento.

♦ Referencias sugeridas

- Cormen, T. H. et al. *Introduction to Algorithms*.
- Python Docs: <https://docs.python.org>

July
17

PLAN DE TRABAJO INTENSIVO (10 DÍAS)

Día	Tarea Principal	Detalles y Reparto
Día 1	Organización y elección final del enfoque	<ul style="list-style-type: none">- Confirmar roles y división de tareas.- Crear repositorio Git y carpeta compartida (Drive o Notion).- Decidir qué funcionalidades básicas tendrá el árbol (insertar, buscar, recorridos, etc).
Día 2	Redacción del Marco Teórico (1ra parte)	<ul style="list-style-type: none">- Una persona redacta definición, tipos de árboles, usos y ventajas.- La otra busca y anota fuentes confiables (libros, documentación oficial, etc).
Día 3	Marco Teórico (2da parte) + Revisión	<ul style="list-style-type: none">- Completar teoría sobre Árbol Binario de Búsqueda (BST).- Citar correctamente.- El otro miembro revisa y mejora redacción.
Día 4	Desarrollo del Árbol en Python (versión 1)	<ul style="list-style-type: none">- Uno desarrolla las clases y métodos básicos (insertar, buscar, recorridos).- El otro crea un script de pruebas e identifica bugs.
Día 5	Mejoras al código + medición de eficiencia (opcional)	<ul style="list-style-type: none">- Refinar el código y agregar mediciones (tiempo de búsqueda vs lista).- Comentar y documentar el código.
Día 6	README + estructura del repositorio Git	<ul style="list-style-type: none">- Redactar README.md con: descripción, instrucciones, autores, aprendizajes.- Subir todo bien organizado a Git.
Día 7	Grabar el video (guion + grabación)	<ul style="list-style-type: none">- Guionar la presentación: breve teoría, demo, metodología, reflexión.- Dividir partes y grabar cada sección.

Día 8	Edición del video	<ul style="list-style-type: none"> - Unir las partes, agregar texto si es necesario. - Subirlo a YouTube (oculto) o Google Drive.
Día 9	Redacción de conclusiones	<ul style="list-style-type: none"> - Reflexión grupal: ¿qué aprendimos?, ¿qué dificultades hubo?, ¿qué mejoramos? - Incluir en README y carpeta digital.
Día 10	Revisión general + entrega	<ul style="list-style-type: none"> - Verificar que esté todo: <ul style="list-style-type: none"> ✓ Marco teórico ✓ Código funcional ✓ Git actualizado ✓ Video listo ✓ Conclusiones - Enviar o subir según indique la cátedra.

Herramientas sugeridas

Necesidad	Herramienta recomendada
Código y control de versiones	Git + GitHub
Redacción colaborativa	Google Docs
Video grabación	Zoom, OBS Studio, Clipchamp
Video edición rápida	CapCut, Canva, Clipchamp
Cronómetro y eficiencia	<code>time</code> en Python (<code>timeit</code> o <code>perf_counter</code>)
Comunicación	WhatsApp / Discord / Meet