UNIVERSIDAD TECNOLÓGICA NACIONAL

PROYECTO INTEGRADOR

Programación 1

Título del proyecto: Árboles de expresión

Alumnos: Pablo Molinari y Leonel Mercorelli

Materia: Programación 1

Profesora: Julieta Trapé

Tutor: Ángel David López

Comisión: 17

Fecha: 08/06/2025

Índice

Introducción

El presente trabajo integrador se enmarca en el estudio de **estructuras de datos no lineales** en el lenguaje de programación Python, con foco en los **árboles de expresión**. Esta estructura permite representar operaciones matemáticas de forma jerárquica y evaluarlas recurriendo a métodos recursivos, lo que la convierte en una herramienta fundamental en múltiples áreas de la programación.

La elección de este tema se basó en la necesidad de profundizar en un tipo de estructura que, si bien ha sido mencionada durante la cursada, no había sido explorada en profundidad ni implementada de manera práctica. Los árboles de expresión constituyen una base conceptual clave para comprender cómo se interpretan, analizan y resuelven operaciones matemáticas en sistemas reales como compiladores, motores de evaluación de expresiones y calculadoras avanzadas.

El objetivo principal del trabajo consistió en **desarrollar un programa en Python** capaz de interpretar expresiones matemáticas en notación infija, convertirlas a notación postfija mediante el algoritmo **Shunting Yard**, construir a partir de ellas un árbol binario de expresión, y finalmente evaluar dicho árbol para obtener el resultado final. Asimismo, se buscó afianzar conocimientos previos sobre estructuras dinámicas y lógica de programación, e incorporar nuevas herramientas de análisis, diseño e implementación.

A través de este enfoque, el trabajo aporta una experiencia formativa significativa tanto desde el punto de vista técnico como conceptual, reforzando competencias clave para el desarrollo de software orientado a la manipulación y evaluación de expresiones.

Marco Teórico

Los árboles constituyen estructuras de datos jerárquicas fundamentales en el campo de la informática. Están formados por nodos conectados entre sí, donde cada nodo contiene un valor y puede poseer hasta dos hijos, en el caso de los árboles binarios. En particular, los árboles de expresión son árboles binarios en los que los nodos internos representan operadores aritméticos (como +, -, *, / y ^) y los nodos hoja contienen operandos numéricos. Esta estructura permite representar la prioridad y el orden de las operaciones en expresiones matemáticas complejas de manera clara y organizada.

Clasificación y jerarquía de nodos en un árbol binario

Un árbol binario puede clasificarse según su estructura:

Árbol binario completo: todos los niveles, excepto posiblemente el último, están completamente llenos, y todos los nodos están lo más a la izquierda posible.

Árbol binario lleno: todos los nodos tienen 0 o 2 hijos.

Árbol binario de búsqueda (BST): los valores del subárbol izquierdo son menores al nodo padre, y los del subárbol derecho son mayores.

Los elementos jerárquicos de un árbol incluyen:

Raíz: el nodo principal desde donde parte toda la estructura.

Nodos internos: aquellos con al menos un hijo.

Nodos hoja: aquellos sin hijos.

Altura del árbol: la longitud del camino más largo desde la raíz hasta una hoja.

Notación infija y postfija

La notación **infija** es la forma habitual de escribir expresiones matemáticas, colocando el operador entre los operandos (por ejemplo: 3 + 4 * 2). Esta notación puede resultar ambigua sin el uso de paréntesis que aclaren la prioridad.

La notación **postfija** o **notación polaca inversa** ubica los operadores después de sus operandos (por ejemplo: 3 4 2 * +). En esta forma, la prioridad está implícita en el orden de los elementos, eliminando la necesidad de paréntesis.

Algoritmo Shunting Yard

Para convertir una expresión infija a postfija, se utiliza el algoritmo **Shunting Yard**, desarrollado por **Edsger Dijkstra**. Este algoritmo emplea una pila para los operadores y una lista para la salida, asegurando que la expresión resultante respete la precedencia y asociatividad de los operadores.

Implementación en Python

En el presente trabajo práctico se implementó un conversor de expresiones infijas a postfijas y un evaluador de expresiones mediante árboles binarios. En Python, se utilizó una clase Nodo para representar los elementos del árbol:

```python
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.izquierdo = None
        self.derecho = None
```

Para evaluar la expresión, se recorrió el árbol en orden postorden (primero los hijos, luego el nodo padre) y se utilizaron las funciones del módulo operator de Python,.

Árbol de expresión

Dada la expresión: 3 + 4 * 2, se convierte a notación postfija como 3 4 2 * +. El árbol resultante es:

Aplicaciones de los árboles de expresión

- Compiladores y analizadores sintácticos (parsers)
- Evaluadores de expresiones en calculadoras
- Interpretación de expresiones lógicas y matemáticas

Caso Práctico

El presente proyecto consistió en el desarrollo de una aplicación capaz de interpretar y evaluar expresiones matemáticas mediante estructuras de datos no lineales. Para ello, se abordó la resolución del problema de evaluar operaciones compuestas ingresadas por el usuario, utilizando un árbol binario de expresión como estructura base.

Descripción del problema

A partir de una expresión matemática ingresada en notación infija (por ejemplo: 3 + 4 * 2), el sistema debía:

1. Validar que los caracteres ingresados fueran correctos.
2. Convertir la expresión a notación postfija utilizando el algoritmo Shunting Yard.
3. Construir un árbol binario de expresión a partir de la notación postfija.
4. Evaluar recursivamente el árbol y devolver el resultado final.

Decisiones de diseño

Se optó por utilizar el algoritmo **Shunting Yard** debido a su capacidad para manejar expresiones con múltiples niveles de prioridad y paréntesis, eliminando la necesidad de analizarlos manualmente.

Para la evaluación del árbol, se implementó una **estrategia recursiva**, ya que esta se adapta naturalmente a la estructura jerárquica del árbol y permite recorrerlo en postorden (es decir, evaluando primero los operandos y luego los operadores).

Se recomienda consultar la sección de **Anexos**, donde es posible visualizar el código fuente completo del programa, evidencias de validación del funcionamiento, un video explicativo y un ejemplo gráfico del árbol de expresión.

Metodología Utilizada

El desarrollo del proyecto integrador se estructuró en diferentes etapas, combinando conocimientos previos adquiridos durante la cursada, investigación autodidacta y trabajo colaborativo. La metodología adoptada se basó en un enfoque práctico, orientado a la comprensión profunda del problema y su implementación gradual en el lenguaje de programación Python.

Investigación previa

Si bien los integrantes del grupo contaban con nociones generales sobre estructuras de datos como árboles binarios y el lenguaje Python, se carecía de conocimientos específicos sobre la construcción y evaluación de árboles de expresión, así como sobre la interpretación de expresiones matemáticas en notación infija.

El primer desafío fue comprender cómo transformar una expresión matemática en una estructura de datos que permitiera construir un árbol binario y evaluarlo correctamente. Durante esta etapa se investigó el **algoritmo Shunting Yard**, que permite convertir expresiones infijas en notación postfija de manera eficiente.

Las fuentes consultadas incluyeron:

- Videos explicativos en YouTube sobre árboles de expresión y el algoritmo Shunting Yard.
- Foros como Stack Overflow, donde se analizaron ejemplos de implementación en Python.
- Blogs y artículos técnicos con código de referencia.
- Pruebas y ensayos propios realizados en entornos locales, replicando el algoritmo paso a paso hasta comprender su lógica de funcionamiento.

Una vez comprendido el algoritmo, se diseñó una estrategia de implementación que permitiera traducir la lógica teórica en una solución funcional y eficiente dentro del entorno de desarrollo elegido.

Etapas de diseño e implementación

1. Implementación de una función que convierte expresiones en notación infija a postfija utilizando el algoritmo Shunting Yard.
2. Construcción de un árbol binario de expresión a partir de la notación postfija, con nodos que representan operadores u operandos.
3. Desarrollo de una función recursiva para recorrer el árbol en postorden y obtener el resultado de la operación.
4. Pruebas de funcionamiento con expresiones de distinta complejidad para validar la robustez del sistema.

Herramientas utilizadas

- **Lenguaje de programación:** Python
- **Entorno de desarrollo:** Visual Studio Code
- **Sistema operativo:** Windows
- **Librerías utilizadas:** operator (para operaciones aritméticas)
- **Control de versiones:** GitHub
- **IA:** ChatGPT

Trabajo colaborativo

El proyecto fue desarrollado de forma colaborativa. La coordinación se realizó mediante reuniones virtuales periódicas, y se utilizó un repositorio compartido en GitHub para la sincronización del código.

La división de tareas se realizó equitativamente entre los integrantes del grupo. Un integrante se encargó de investigar y desarrollar la etapa de conversión de la expresión infija a postfija. El otro se enfocó en la construcción del árbol binario a partir de dicha expresión y su posterior evaluación. Ambos participaron activamente en la integración de las partes y en la resolución de problemas lógicos del programa.

Las decisiones se tomaron de forma consensuada, y la redacción del informe se realizó de manera colaborativa, garantizando una comprensión integral del proyecto por parte de ambos integrantes.

Resultados Esperados

Al finalizar el desarrollo del proyecto, se realizaron múltiples pruebas destinadas a verificar el correcto funcionamiento del sistema, validar el comportamiento de cada componente y evaluar la solidez general de la solución implementada. Los resultados obtenidos fueron satisfactorios, cumpliendo con los objetivos establecidos en las etapas iniciales del trabajo.

Funcionamiento logrado

El sistema permitió interpretar expresiones matemáticas ingresadas en notación infija, convertirlas correctamente a notación postfija mediante el algoritmo **Shunting Yard** y construir un árbol binario de expresión a partir de dicha notación. Se validó que el orden de precedencia y agrupación de los operadores fuera respetado en la expresión generada.

A partir del árbol binario, se llevó a cabo una evaluación recursiva en recorrido postorden, la cual devolvió resultados correctos incluso en expresiones anidadas, con paréntesis o con operaciones mixtas.

Casos de prueba realizados

Se realizaron pruebas con expresiones de distinta complejidad. A continuación se detallan algunos ejemplos:

3 + 4 * 2	→	**3 4 2 * +**	→	Resultado esperado: **11**
(3 + 4) * 2	→	**3 4 + 2 ***	→	Resultado esperado: **14**
10 ^ 2 - 142	→	**10 2 ^ 142 -**	→	Resultado esperado: **- 42**
100 * (2 + 12) / (5^2)	→	**100 2 12 + * 5 2 ^ /**	→	Resultado esperado: **56**

En todos los casos, el sistema generó correctamente la notación postfija, construyó el árbol de expresión correspondiente y devolvió el resultado final esperado.

Errores corregidos

Durante el desarrollo se identificaron y solucionaron diversos errores:

En las primeras versiones, los números con más de un dígito (ej: 102) o con decimales (ej: 1,5) eran interpretados como caracteres individuales. Para solucionarlo, se implementó una variable acumuladora que permitiera agrupar correctamente los dígitos antes de agregarlos a la salida.

También hubo dificultades en el manejo de paréntesis anidados. La lógica inicial del algoritmo no contemplaba correctamente estos casos, lo que provocaba errores en el orden de las operaciones. Se realizaron ajustes para asegurar un tratamiento adecuado de los paréntesis anidados y preservar la jerarquía correcta en la expresión

Evaluación del rendimiento

Dado que el alcance del trabajo se centró en la comprensión e implementación funcional del sistema, no se realizaron comparaciones de rendimiento con otras soluciones. Sin embargo, en las pruebas realizadas el sistema respondió de forma inmediata incluso ante expresiones compuestas, lo cual sugiere una eficiencia adecuada para los objetivos propuestos.

Repositorio

El proyecto se encuentra disponible en el siguiente repositorio de GitHub:

https://github.com/LeoMercorelli/proyecto_integrador-p1.git

Conclusiones

El desarrollo de este proyecto permitió al equipo profundizar en el estudio y aplicación de estructuras de datos no lineales, en particular los árboles de expresión, temática que hasta el momento no había sido abordada en profundidad durante la cursada. A través de la implementación práctica, se consolidaron conocimientos teóricos previos y se incorporaron nuevos conceptos fundamentales como el algoritmo **Shunting Yard**, la notación postfija y la **evaluación recursiva** de árboles binarios.

Aprendizajes adquiridos

Uno de los aprendizajes más significativos fue la capacidad de **descomponer un problema complejo** en subproblemas abordables de forma modular, integrando luego sus partes en una solución coherente. Además, se reforzaron habilidades esenciales como el análisis de errores, la manipulación de estructuras dinámicas y el control de flujo en Python. También se desarrolló una mayor autonomía investigativa y capacidad de resolución colaborativa.

Utilidad del proyecto

El trabajo posee una **utilidad práctica concreta** dentro del campo de la programación, ya que permite comprender cómo se interpretan expresiones matemáticas en compiladores, calculadoras o motores de ejecución. Asimismo, ofrece una introducción sólida al uso de árboles en problemas reales y sienta bases para desarrollos más avanzados que involucren lenguajes de programación o análisis simbólico.

Posibles mejoras futuras

A partir de la experiencia adquirida, se identificaron posibles extensiones del proyecto que podrían enriquecer su funcionalidad:

- Incorporar soporte para números negativos y operadores adicionales.
- Implementar una **interfaz gráfica o consola interactiva** que facilite la interacción con el usuario.
- Agregar soporte para **variables simbólicas** (como x, y) evaluables mediante valores asignados por el usuario.
- Incluir una **visualización gráfica** del árbol construido, con fines pedagógicos y explicativos.

- Mejorar las **validaciones sintácticas** para detectar errores como operadores aislados, paréntesis desbalanceados o expresiones mal formadas, evitando así que el programa se “rompa” ante entradas incorrectas.

Dificultades encontradas y resolución

Entre las principales dificultades se destacan la comprensión e implementación del algoritmo **Shunting Yard**, así como el diseño del recorrido recursivo del árbol. Estos obstáculos fueron superados mediante investigación autodidacta, prueba y error, y diálogo constante entre los integrantes del grupo. El proceso no solo permitió resolver problemas técnicos, sino también fortalecer la capacidad de trabajo en equipo y aprendizaje colaborativo.

Reflexión final

En síntesis, el proyecto resultó altamente enriquecedor tanto desde el punto de vista técnico como formativo. Representa un avance significativo en el camino hacia una comprensión más profunda de las estructuras de datos y su aplicación en contextos reales dentro del desarrollo de software.

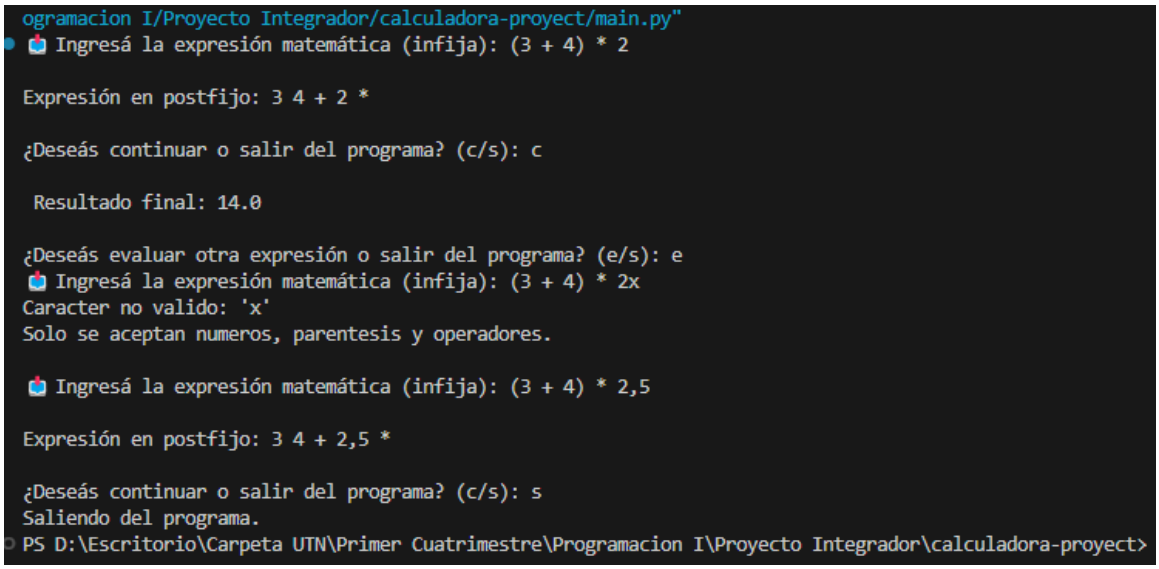
Bibliografía

- Dijkstra, E. W. (1961). *The Shunting Yard Algorithm*. Recuperado de https://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD340.html
- Python Software Foundation. (2024). *Python 3 documentation: operator module*.
- Stack Overflow. (s.f.). *How to implement Shunting Yard Algorithm in Python*. Recuperado el 7 de junio de 2025, de https://stackoverflow.com/
- YouTube. (s.f.). *Shunting Yard Algorithm explained*. Videos consultados entre abril y mayo de 2025.
- Cátedra de Programación I - UTN (2025). *Apuntes, clases teóricas y material audiovisual provisto por el equipo docente durante la cursada.*

ANEXOS

ANEXO 1 – VALIDACION DE FUNCIONAMIENTO

A continuación, se presenta una captura que evidencia el correcto funcionamiento del programa.



En ella se observa:

- La evaluación de una expresión válida: (3 + 4) * 2, que se convierte correctamente a postfija y retorna el resultado 14.0.
- La detección de un carácter no válido (x), con su correspondiente mensaje de advertencia.
- La correcta interpretación de un número decimal en formato con coma: 2,5.
- La opción de finalizar la ejecución del programa mediante la opción de salida.

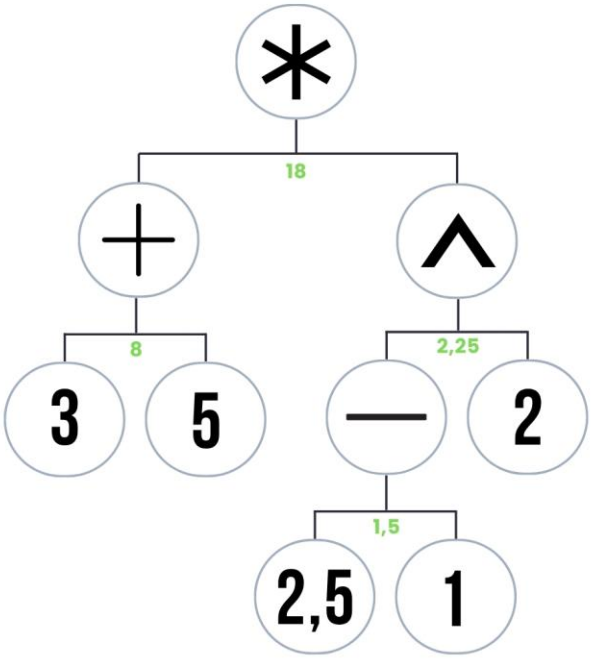
ANEXO 2 – EJEMPLO CON ARBOL GRAFICADO

A continuación se presenta la representación gráfica del árbol de expresión correspondiente a la operación (3 + 5) * (2,5 - 1)^2. En ella se visualiza la jerarquía de operadores, el recorrido postorden y el resultado final obtenido al evaluar la raíz del árbol: 18.

Arbol de expresion

Representación en árbol de la expresión matemática del ejemplo

Expresión infija: (3+5)*(2,5-1)^2
Expresión postfija: 3; 5; +; 2,5; 1; -; 2; ^; *



Valor obtenido al evaluar la raíz del árbol:

18

ANEXO 3 – VIDEO EXPLICATIVO

Como complemento al informe escrito, el equipo realizó un video demostrativo en el cual se presenta el funcionamiento completo del programa desarrollado. En dicho video se explican brevemente las etapas del proyecto, se muestra el ingreso de expresiones por parte del usuario, la conversión a notación postfija, la construcción del árbol de expresión y la evaluación final.

El objetivo de este recurso audiovisual es facilitar la comprensión del proceso y evidenciar el correcto desempeño del sistema en tiempo real.

Enlace al video:

https://www.youtube.com/watch?v=a2J1WSiFHyg&ab_channel=LeonelMercorelli

También les recordamos el enlace al repositorio:

https://github.com/LeoMercorelli/proyecto_integrador-p1.git

ANEXO 4 – CODIGO FUENTE

A continuación, se presenta el código fuente completo del proyecto. El mismo se encuentra comentado y estructurado en secciones para facilitar su lectura, identificación de funciones y comprensión de la lógica implementada.

El código fue escrito en Python, utilizando funciones, clases y estructuras dinámicas, y se ajusta a los objetivos establecidos en el presente trabajo práctico:

```
import operator

# 1) ------------------ CREAMOS LA CLASE PARA LOS NODOS------------------
class Nodo:
    def __init__(self, valor):
        self.valor = valor
        self.izquierdo = None
        self.derecho = None
```

```
# 2) ------------------ DEFINIMOS OPERADORES------------------

operadores = {

    '+': (1, operator.add),

    '-': (1, operator.sub),

    '*': (2, operator.mul),

    '/': (2, operator.truediv),

    '^': (3, operator.pow),

}
```

```
# 3) ------------------ VALIDACION DE LA EXPRESION DEL USUARIO------------------

def es_valida(infija):

    permitidos = [

        '0','1','2','3','4','5','6','7','8','9',

        '+','-','*','/','^','(',')',','

    ]

    for carac in infija:

        if carac not in permitidos:

            print(f"Caracter no valido: '{carac}'")

            print("Solo se aceptan numeros, parentesis y operadores.\n")

            return False

    return True
```

```
# 4) ------------------ ALGORITMO SHUNTING YARD - INFIJA A POSTFIJA ------------------

def convertir_a_postfija(infija):

    postfija = []

    pila = []

    numero = ''

    for carac in infija:  # Recorremos la expresion, caracter por caracter

        if carac.isdigit() or carac == ',':  # Si el caracter es un numero o una coma
decimal

            numero += carac

        else:

            if numero:

                postfija.append(numero)

                numero = ''

            if carac in operadores:

                while (

                    pila and

                    pila[-1] in operadores and

                    operadores[carac][0] <= operadores[pila[-1]][0]

                ):

                    postfija.append(pila.pop())

                pila.append(carac)

            elif carac == '(':

                pila.append(carac)

            elif carac == ')':

                while pila and pila[-1] != '(':

                    postfija.append(pila.pop())

                pila.pop()

    if numero:

        postfija.append(numero)

    while pila:

        postfija.append(pila.pop())

    return postfija
```

```
# 5) ------------- CONSTRUIMOS EL ARBOL USANDO LA EXPRESION POSTFIJA OBTENIDA------------
def construir_arbol(postfija):
    pila = []
    for carac in postfija:
        nodo = Nodo(carac)
        if carac in operadores:
            nodo.derecho = pila.pop()
            nodo.izquierdo = pila.pop()
        pila.append(nodo)
    return pila[0]  # Devolvemos el unico nodo que queda: la raiz
```

```
# 6) ------------------ RESOLVEMOS EL ARBOL RECURSIVAMENTE ------------------
def resolver_nodo(nodo):
    if nodo.valor not in operadores:
        return float(nodo.valor.replace(',', '.'))  # Cambiamos coma por punto
    valor_izq = resolver_nodo(nodo.izquierdo)
    valor_der = resolver_nodo(nodo.derecho)
    return operadores[nodo.valor][1](valor_izq, valor_der)
```

```
##########------------------ PROGRAMA PRINCIPAL ------------------##########

while True:

    while True:

        infija = input("\ud83d\udce5 Ingresá la expresión matemática (infija): ")

        infija = infija.replace(' ', '')

        if es_valida(infija):

            break

    postfija = convertir_a_postfija(infija)

    print(f"\nExpresión en postfijo: {' '.join(postfija)}")

    continuar = input("\n¿Deseás continuar o salir del programa? (c/s): ").lower()

    if continuar == 'c':

        raiz = construir_arbol(postfija)

        resultado = resolver_nodo(raiz)

        print(f"\n Resultado final: {resultado}")

        otra = input("\n¿Deseás evaluar otra expresión o salir del programa? (e/s):
").lower()

        if otra != 'e':

            print("Saliendo del programa.")

            break

    else:

        print("Saliendo del programa.")

        breaka los objetivos establecidos en el presente trabajo práctico.
```