DOCUMENTACIÓN FINAL

Proyecto #1 - MiniLang

Docente:	Ing. Lucas Cueva
Integrantes:	
• Jorge Alejandro Discua	22311096
• Alejandro Jose Reynaud	22211074
• Douglas Omar Rubi	12011121
• Jorge Luis Aguirre	22341189

Fecha de entrega: 15 de diciembre de 2025

Tabla de contenido

Contenido:

- Resumen ejecutivo
- 1. Introducción
- 2. Objetivos del proyecto
- 3. Alcance y características implementadas
- 4. Especificación del lenguaje MiniLang
- 4.1 Reglas generales
- 4.2 Tipos de datos
- 4.3 Operadores
- 4.4 Sintaxis principal (gramática simplificada)
- 4.5 Ejemplo de programa
- 5. Arquitectura del sistema
- 5.1 Componentes
- 6. Detalles de implementación
- 6.1 Lexer (Análisis léxico)
- 6.2 Parser y AST
- 6.3 Interpreter
- 6.4 Manejo de errores
- 7. Pruebas y validación
- 8. Manual de usuario
- 8.1 Requisitos
- 8.2 Ejecución
- 8.3 Recomendaciones de escritura
- 9. Limitaciones actuales
- 10. Próximos pasos y mejoras propuestas
- 11. Conclusiones
- 12. Anexos
- 12.1 Glosario breve

Resumen ejecutivo

MiniLang es un lenguaje de programación educativo e intencionalmente simple, orientado a enseñar los conceptos fundamentales detrás de un intérprete: análisis léxico, parsing, construcción de un árbol de sintaxis abstracta (AST) y evaluación de instrucciones. En esta entrega final se documenta el alcance implementado, la sintaxis soportada, la arquitectura y el uso del intérprete.

El sistema permite declarar y asignar variables enteras, ejecutar expresiones aritméticas con paréntesis, realizar comparaciones y ejecutar estructuras condicionales tipo if usando la sintaxis de MiniLang. Además, se inició la incorporación de strings para una manipulación básica, dejando preparado el camino para extensiones futuras.

1. Introducción

MiniLang es un lenguaje de programación simple diseñado para enseñar conceptos básicos de interpretación de código. La meta del proyecto es comprender y aplicar las etapas principales que siguen los lenguajes interpretados: tokenización, construcción de gramática y evaluación de expresiones e instrucciones.

El intérprete se ejecuta desde un programa principal (por ejemplo, un archivo main.py) donde el usuario define un fragmento de código (sample) escrito en MiniLang para su ejecución. Este enfoque facilita realizar pruebas rápidas del lenguaje conforme se agregan nuevas características.

2. Objetivos del proyecto

Objetivo general:

Diseñar e implementar un intérprete funcional para MiniLang, documentando su sintaxis, alcance y arquitectura, con énfasis en la comprensión de los componentes internos de un lenguaje.

Objetivos específicos:

- Interpretar y almacenar variables enteras.
- Resolver expresiones aritméticas con suma, resta, multiplicación, división y paréntesis.
- Comparar valores usando operadores relacionales (>, <, >=, <=).

- Ejecutar estructuras condicionales (si ... entonces ... fin_si).
- Implementar un manejo básico de errores de sintaxis con mensajes descriptivos.
- Iniciar el soporte para strings y su ejecución básica en el intérprete.

3. Alcance y características implementadas

A continuación se listan las capacidades que forman parte del alcance de la entrega final:

- Variables y asignación: identificadores (por ejemplo, a, b) y operador '='.
- Expresiones aritméticas: +, -, *, /, con soporte de paréntesis.
- Comparaciones: >, <, >=, <=
- Estructura condicional: si <condición> entonces ... fin_si.
- Salida estándar: imprimir(<expresión>);
- Manejo sencillo de strings (soporte inicial / en progreso).

Palabras reservadas principales: inicio, fin, imprimir, si, entonces, fin_si. El lenguaje utiliza ';' como terminador de instrucciones para facilitar la detección de errores.

4. Especificación del lenguaje MiniLang

4.1 Reglas generales

- Todo programa debe comenzar con la palabra reservada 'inicio' y terminar con 'fin'.
- Las instrucciones (por ejemplo, asignaciones e imprimir) terminan con ';'.
- Los espacios y saltos de línea no afectan el significado del programa, excepto para separar tokens.
- Los identificadores representan nombres de variables y se componen de letras (y, opcionalmente, números).

4.2 Tipos de datos

- Enteros: usados para cálculos y comparaciones.
- Strings: soporte básico en desarrollo (por ejemplo, tokenización y ejecución inicial).

4.3 Operadores

Categoría	Operadores	Descripción
Aritméticos	+, -, *, /	Operaciones sobre enteros, con precedencia estándar y paréntesis.
Relacionales	>, <, >=, <=	Comparaciones usadas en condiciones del 'si'.
Asignación	=	Asigna el resultado de una expresión a una variable.

4.4 Sintaxis principal (gramática simplificada)

```
programa      -> 'inicio' { sentencia } 'fin'
sentencia     -> asignacion | imprimir | condicional
asignacion    -> IDENT '=' expr ';'
imprimir      -> 'imprimir' '(' expr ')' ';'
condicional   -> 'si' condicion 'entonces' { sentencia } 'fin_si'
condicion     -> expr relop expr
relop         -> '>' | '<' | '>=' | '<='
expr          -> termino { ('+' | '-') termino }
termino       -> factor  { ('*' | '/') factor  }
factor        -> NUMERO | IDENT | '(' expr ')' | STRING
```

4.5 Ejemplo de programa

```
inicio
    a = 5;
    b = a + 3 * (2 + 1);
    imprimir(a);
    imprimir(b);
    si b > 10 entonces
        imprimir(b);
    fin_si
    b = b - 1;
    imprimir(b);
fin
```

5. Arquitectura del sistema

MiniLang se implementa como un intérprete con un flujo clásico de procesamiento. El código fuente es transformado a tokens, luego a una estructura sintáctica (AST) y finalmente se evalúa para producir resultados.

Flujo general:

Código Fuente → Lexer → Parser → AST → Interpreter → Resultado

5.1 Componentes

- Lexer: convierte el texto en tokens (números, identificadores, operadores y palabras reservadas).
- Parser: valida la secuencia de tokens según la gramática y construye el AST.
- AST: estructura intermedia que representa el programa (expresiones, asignaciones, imprimir y condicionales).
- Interpreter: recorre el AST y ejecuta las operaciones, manteniendo un entorno de variables.
- Error Handler: genera errores de sintaxis con mensajes claros para el usuario.

Durante el avance más reciente se inició la ampliación del lexer y la gramática para futuras características (por ejemplo, bucles y funciones), aunque el enfoque funcional principal de esta entrega se mantiene en expresiones, condicionales y strings básicos.

6. Detalles de implementación

6.1 Lexer (Análisis léxico)

El lexer recorre el código fuente carácter por carácter, agrupando los símbolos en tokens. Cada token incluye un tipo (por ejemplo NUMERO, IDENT, OPERADOR) y su valor asociado. Se reconocen palabras reservadas como inicio, fin, imprimir, si, entonces y fin_si.

Tokens principales

Token	Ejemplo	Uso
NUMERO	5	Literales enteros.
IDENT	a, b	Nombres de variables.
ASIGNACION	=	Asignar expresiones a variables.
ARITMETICO	+ - * /	Operadores aritméticos.
PARENTESIS	( )	Agrupar expresiones y controlar precedencia.
RELACIONAL	>, <, >=, <=	Comparaciones dentro de condiciones.
PUNTOYCOMA	;	Fin de instrucción.
KW_INICIO	inicio	Marca el inicio del programa.
KW_FIN	fin	Marca el final del programa.

KW_IMPRIMIR	imprimir	Salida estándar.
KW_SI	si	Inicio de condicional.
KW_ENTONCES	entonces	Separador de la condición y el bloque.
KW_FIN_SI	fin_si	Cierre de condicional.
STRING	"Hola"	Literales de texto (soporte básico).

6.2 Parser y AST

El parser consume la lista de tokens y aplica reglas de gramática para construir un AST. El uso de un AST permite separar la validación sintáctica de la ejecución, simplificando la extensión del lenguaje. Para expresiones, se respeta la precedencia estándar (* y / antes que + y -) y el agrupamiento por paréntesis.

Las instrucciones del programa se modelan como nodos del AST (por ejemplo: Assign, Print, If, BinOp, Number, Var). El bloque de un 'si' contiene una lista de sentencias que solo se ejecutan cuando la condición es verdadera.

6.3 Interpreter

El intérprete recorre el AST y evalúa cada nodo. Se mantiene una tabla de símbolos (entorno) donde se guardan las variables y sus valores. Al ejecutar una asignación, se evalúa la expresión del lado derecho y se almacena en el identificador del lado izquierdo. Para 'imprimir', se evalúa la expresión y se muestra el resultado.

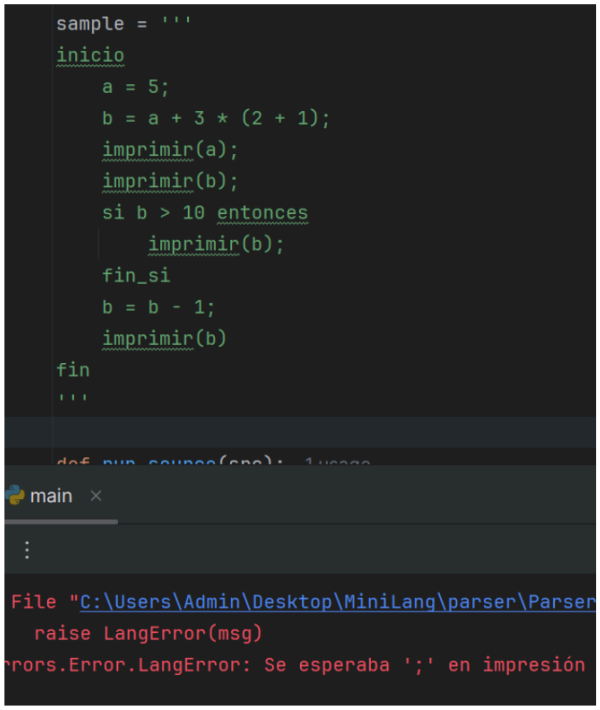
En el caso de la estructura 'si', se evalúa la condición relacional y, si el resultado es verdadero, se ejecuta el bloque interno. Este diseño permite ampliar fácilmente el lenguaje con nuevas sentencias (por ejemplo, while/for o funciones) sin reescribir el flujo principal.

6.4 Manejo de errores

El sistema incluye un manejo básico de errores de sintaxis. Por ejemplo, si el usuario omite el ';' al final de una instrucción de imprimir, el parser genera un error con un mensaje descriptivo indicando lo que se esperaba.

Ejemplo de error:

```
imprimir(b)
# Error: Se esperaba ';' en impresión
```



Manejo básico de error de sintaxis (no se colocó ; al final de imprimir(b))

7. Pruebas y validación

Las pruebas se realizaron ejecutando diferentes samples en MiniLang, verificando la correcta evaluación de expresiones, asignaciones, impresión y condicionales. Se validó también el manejo de errores básicos ante entradas incorrectas.

- Prueba de asignación y aritmética con paréntesis.
- Prueba de comparación y ejecución condicional (si).
- Prueba de impresión de variables.
- Prueba de error por omisión del punto y coma en imprimir.

8. Manual de usuario

8.1 Requisitos

- Python 3.x instalado.
- Estructura del proyecto con un archivo principal (por ejemplo main.py) que cargue el sample de MiniLang.

8.2 Ejecución

1. Abrir el archivo principal del proyecto (main.py).
2. Editar el string sample e introducir el programa en MiniLang.
3. Ejecutar el programa desde consola: python main.py
4. Observar la salida del intérprete en la terminal.

8.3 Recomendaciones de escritura

- Asegurarse de cerrar el programa con 'fin'.
- Colocar ';' al final de cada asignación e instrucción imprimir.
- Usar 'fin_si' para cerrar los condicionales.
- Para expresiones complejas, utilizar paréntesis para evitar ambigüedades.

9. Limitaciones actuales

- No hay estructuras de repetición (for/while) completamente disponibles en la entrega.

- Sistema de funciones limitado a funciones de tipo int (con definición, parámetros y retorno).
- El manejo de strings básico pero sólido, no tiene funciones que involucren manipulación avanzada de strings.
- No existe un sistema profundo de tipos ni verificación semántica completa.
- El rendimiento no ha sido optimizado; el enfoque es educativo.

10. Próximos pasos y futuras mejoras propuestas

- Concatenación avanzada de strings y operaciones de texto.
- Funciones con múltiples parámetros y soporte de retorno.
- Implementación de bucles (mientras/para) y estructuras de repetición.
- Sistema de módulos o includes para reutilización de código.
- Operaciones de entrada del usuario (input).
- Documentación extendida del lenguaje (más ejemplos y casos borde).

11. Conclusiones

Con MiniLang se logró construir un intérprete funcional que demuestra el flujo interno de ejecución de un lenguaje: del código fuente a tokens, de tokens a un AST, y del AST a resultados ejecutables. El alcance implementado cubre los elementos esenciales para comprender la base de un lenguaje interpretado, y deja establecido un punto de partida claro para futuras extensiones como bucles, funciones y manejo más completo de strings.

La modularidad de la arquitectura (lexer, parser, AST e intérprete) facilita el mantenimiento del proyecto y su crecimiento. La experiencia adquirida en esta implementación fortalece la comprensión práctica de conceptos vistos en el curso relacionados con gramáticas, parsing y ejecución de instrucciones.

12. Anexos

12.1 Glosario breve

- Token: unidad mínima reconocida por el lexer (número, palabra reservada, operador).

- Parser: componente que valida la estructura del programa y construye el AST.
- AST (Árbol de Sintaxis Abstracta): representación estructurada del programa.
- Interpreter: componente que evalúa el AST y ejecuta el programa.

12.2 Imagenes de finales de funcionamiento de MiniLang:

```
sample = '''
inicio
    imprimir("================================");
    imprimir("        DEMOSTRACION MINILANG       ");
    imprimir("================================");
    imprimir("");
    imprimir("prueba de variables y expresiones");
    imprimir("");

    a = 5;
    b = a + 3 * (2 + 1);
    imprimir("Resultado a:");
    imprimir(a);
    imprimir("Resultado b:");
    imprimir(b);

    imprimir("");
    imprimir(">>Manejo de condicionales");

    si b > 10 entonces
        imprimir("b es mayor que 10");
    fin_si

    imprimir("");
    imprimir(">>Prueba de self-assignment with update");
    imprimir("Imprimiendo b");
    imprimir(b);
    imprimir("");
    b = b - 1;
    imprimir("Resultado de b - 1:");
    imprimir(b);

    imprimir("");
    imprimir(">>Prueba de strings y concatenacion");
    c = "ya manejo variables tipo string";
    imprimir(c);
    imprimir("y puedo imprimir tanto strings simples");
    imprimir("asi como" + " concatenacion de strings");

    imprimir("");
    imprimir(">>Prueba de manejo de funciones");
    imprimir("Definimos suma(x,y), luego hacemos suma(1,2)");
    func suma(x, y) {
        return x + y;
    }
    resultado = suma(1,2);
    imprimir(resultado);
fin
'''
```

Imagen del sample para evaluación final.

Imagen de salida en consola para el sample de la evaluación final.

12.3 Enlace al repositorio de GitHub

https://github.com/Aledisc/MiniLang---Lenguajes-de-programaci-n