Proyecto: Parser para Fangless Python

Descripción del Proyecto

El proyecto se expande a la siguiente fase del proceso de compilación: el analizador sintáctico o Parser. El objetivo es diseñar e implementar un componente que tome la secuencia de tokens generada por el Lexer y construya una representación intermedia de la estructura del programa, conocida como Árbol de Sintaxis Abstracta (AST).

Este Parser validará la gramática de "Fangless Python" y detectará errores sintácticos. Su correcta implementación es crucial para las fases posteriores del compilador, como la generación de código. El enfoque principal de esta entrega es el Parser, complementando el trabajo realizado en la entrega anterior del Lexer.

Aspectos Administrativos y Técnicos

- **Herramienta de Desarrollo:** La herramienta designada para el desarrollo es **Python**, con el uso obligatorio de la librería **PLY (Python Lex-Yacc)**.
- **Idioma:** El código fuente, los comentarios internos y toda la documentación técnica deberán redactarse en **inglés**.
- **Control de Versiones:** Se requiere el uso del sistema de control de versiones **Git** en conjunto con la plataforma **GitHub**. La integración de cambios al repositorio principal deberá realizarse exclusivamente a través de *Pull Requests*.
- **Nomenclatura de Ramas:** Las ramas de desarrollo deben adherirse estrictamente al formato TASK_#_BriefDescription.
- **Equipos de Trabajo:** El proyecto será desarrollado en equipos de hasta **tres estudiantes**.

- **Fecha de Entrega:** La fecha límite para la entrega del proyecto es el **jueves 2 de Octubre de 2025, a las 23:59 hrs**.
- **Buenas Prácticas de Código:** Se espera que el código producido sea limpio, eficiente, modular y mantenible, facilitando así su futura extensión y revisión.
- El valor de este proyecto es de un 20% de la totalidad de la rúbrica proyectos.

Requerimientos del Lexer (Analizador Léxico)

Entrada y Salida

- **Entrada:** Una secuencia de tokens provista por el Lexer de "Fangless Python".
- **Salida:** Un Árbol de Sintaxis Abstracta (AST) que represente la estructura jerárquica del código fuente.

Características Incluidas en la Sintaxis

El **Parser** debe procesar la secuencia de tokens generada por el **Lexer** y construir un **Árbol de Sintaxis Abstracta (AST)** que represente la estructura jerárquica del código. La gramática de "Fangless Python" se define por las siguientes estructuras sintácticas:

Sentencias (Statements)

El Parser debe reconocer y manejar tanto sentencias simples como compuestas.

- **Sentencias Simples:** Se trata de sentencias que finalizan con un salto de línea (NEWLINE) o un punto y coma (;). Estas incluyen:
    - **Asignación:** Se procesan asignaciones simples (=) y aumentadas (+=, -=, *=, /=, %=, //=, **=).
    - **Llamadas a Funciones:** Se reconocen llamadas a funciones con argumentos posicionales.
    - **return:** Sentencia de retorno de un valor.

- **pass:** Una operación nula, que no hace nada.
- **Control de flujo:** Sentencias break y continue para salir de bucles o avanzar a la siguiente iteración.

- **Sentencias Compuestas:** Estas sentencias definen bloques de código y siempre terminan con dos puntos (COLON), seguidos por una nueva línea y una indentación.
  - **Definición de Clases:** Se manejan las definiciones de clases (class NAME:).
  - **Definición de Funciones:** Se procesan las definiciones de funciones (def NAME(arguments):).
  - **Bloques Condicionales:** La estructura if es la base, que puede extenderse con elif y else. Se procesa la forma completa if expression: block elif expression: block else: block.
  - **Bucles Iterativos:** Se procesa el bucle for (for NAME in expression: block).
  - **Bucles Condicionales:** Se procesa el bucle while (while expression: block).

Expresiones (Expressions)

El Parser debe construir árboles de expresiones respetando la jerarquía y asociatividad de los operadores.

- **Operadores Aritméticos:** Se manejan los siguientes operadores con su respectiva precedencia:
  1. ** (exponenciación)
  2. *, /, //, % (multiplicación, división, división entera y módulo)
  3. +, - (suma y resta)
- **Operadores Relacionales:** Se procesan las comparaciones binarias como ==, !=, <, >, <=, y >=.

- **Operadores Lógicos:** El Parser debe manejar la precedencia y asociatividad de and, or, y not.
    1. not
    2. and
    3. or
- **Literales y Delimitadores:**
    - **Identificadores:** Los nombres de variables, funciones y clases.
    - **Literales:** Se procesan números enteros y flotantes, cadenas de texto (string) y valores booleanos (True, False).
    - **Delimitadores:** El Parser debe reconocer ( ), [ ], { } y , como parte de la sintaxis de listas, tuplas, diccionarios, y llamadas a funciones.

Cualquier token no generado en el análisis léxico requerido en esta especificación deberá ser agregado.

Manejo de Errores Léxicos

El analizador sintáctico debe gestionar los errores de forma robusta, sin interrumpir su ejecución abruptamente e incluyendo los anteriormente programados en el analizador léxico.

- **Caracteres Desconocidos:** Cualquier carácter que no corresponda a un token definido debe generar un error léxico.
- **Secuencias de Escape Inválidas:** El uso de secuencias de escape no reconocidas dentro de literales de cadena debe ser reportado como un error.
- **Indentación Incorrecta:** Deben manejarse y reportarse los errores en la estructura de indentación del código.

- **Gramática Inválida:** Un error sintáctico fundamental, como una sentencia sin la estructura correcta (ej., if sin dos puntos).
- **Expresiones Mal Formadas:** Errores de precedencia o asociatividad, como el uso de un operador binario sin un segundo operando.

Rúbrica de Evaluación

Importante: Los puntos serán evaluados en su totalidad por la defensa del proyecto. La entrega de este es un criterio mínimo de evaluación.

Criterio	Puntos	Descripción
Completitud del Lexer	40	Construcción correcta del AST para todas las estructuras y sentencias especificadas en la gramática, incluyendo precedencia de operadores.
Calidad del Código	30	El código debe ser bien estructurado, modular y estar debidamente documentado, demostrando una implementación eficiente.
Manejo de Errores	15	Capacidad del sistema para detectar y reportar errores léxicos y sintácticos con mensajes claros y precisos que faciliten la depuración.
Documentación	15	Calidad y claridad de la documentación técnica (comentarios) y de usuario (guía de uso), explicando los componentes y su funcionamiento.