

Manual para ejecutar un programa Antlr4 con Python.

Alumno:

Hernandez Arturo Eliseo

Carrera:

Ingeniería en Sistemas Computacionales

CHETUMAL Q.ROO, DICIEMBRE DE 2024

Requisitos previos

Herramientas que necesitas:

1. Java Development Kit (JDK):
 - ANTLR es una herramienta basada en Java, por lo que necesitarás tener Java instalado.
 - Descarga JDK. En mi caso instale el JDK-23
2. Python (3.x):
 - Instala Python desde python.org. Usa el Python mas reciente para no tener problemas con Antlr4 cuando se ejecuta.
3. pip:
 - Administrador de paquetes de Python, generalmente incluido en las instalaciones de Python.
4. antlr4 Herramienta:
 - Archivo jar de ANTLR para generar analizadores.

Instala JAVA y verifica que está funcionando correctamente. Usa el siguiente comando en el terminal.

- `java -version`

Instalación manual

1. Descargue el archivo jar de ANTLR:
 - Visite la página de versiones de ANTLR.
 - Descargue la última versión de antlr-4.x-complete.jar.


```
grammar Beginners;

// Main Rules
programa    : 'inicio:' bloque;
bloque      : instruccion+;
instruccion : mostrar | asignar | si | darvueltas | hacerFuncion | mientras;

// Mostrar texto
mostrar     : 'escribe' '(' contenido ')' ';' ;

// Contenido flexible que combina texto y expresiones
contenido   : elemento (OPERADOR elemento)*;
elemento    : texto | expresion;

// Operators for concatenation
OPERADOR    : '+'; // Add more operators if needed

// Asignación de variables
asignar     : tipo ID '=' expresion ';';
tipo        : 'numero' | 'decimal' | 'palabra' | 'si_no';

// Condicionales
si          : 'si' '(' condicion ')' '{' bloque '}' ('sino' '{' bloque '}')?;

// Ciclos
darvueltas  : 'repite' '(' expresion ')' '{' bloque '}';
mientras    : 'mientras' '(' condicion ')' '{' bloque '}';

// Funciones
hacerFuncion : 'funcion' ID '(' parametros? ')' '{' bloque '}';
parametros  : ID (',' ID)*;
```

Crear un intérprete de Python

```
interpreter.py > BeginnersExecutor > visitMientras

1  from antlr4 import *
2  from BeginnersLexer import BeginnersLexer
3  from BeginnersParser import BeginnersParser
4  from BeginnersVisitor import ParseTreeVisitor
5
6  class BeginnersExecutor(ParseTreeVisitor):
7      def __init__(self):
8          self.variables = {}
9
10     def visitPrograma(self, ctx):
11         print("Visiting the main program block...")
12         self.visit(ctx.bloque())
13
14     def visitBloque(self, ctx):
15         for instruccion in ctx.instruccion():
16             self.visit(instruccion)
17
18     def visitAsignar(self, ctx):
19         nombre = ctx.ID().getText()
20         valor = self.visit(ctx.expresion())
21         self.variables[nombre] = valor
22         print(f"Assigned {nombre} = {valor}")
23
24
25     def visitMostrar(self, ctx):
26
27         result = ""
28         for elemento_ctx in ctx.contenido().elemento():
29             if elemento_ctx.texto():
30                 raw_text = elemento_ctx.texto().getText()
31                 clean_text = raw_text[1:-1]
32                 result += clean_text
```

Generar archivos de analizador léxico y analizador utilizando este comando:

- `antlr4 -Dlanguage=Python3 (nombre_de_archivo)`

o si no funciona ese comando o te da error utiliza este para generarlo.

- `java -jar antlr-4.x-complete.jar(la versión que utilizas) -Dlanguage=Python3 (nombre_de_archivo) -visitor -no-listener`

Ejecutar el intérprete:

- `python interpreter.py (nombre_de_archivo)`

El lenguaje que estoy utilizando es este:

grammar Beginners;

// Main Rules

programa : 'inicio:' bloque;

bloque : instruccion+;

instruccion : mostrar | asignar | si | darvueltas | hacerFuncion | mientras;

// Mostrar texto

mostrar : 'escribe' '(' contenido ')' ';;';

// Contenido flexible que combina texto y expresiones

contenido : elemento (OPERADOR elemento)*;

elemento : texto | expresion;

// Operadoras de concatenación

OPERADOR : '+'; // Add more operators if needed

// Asignación de variables

asignar : tipo ID '=' expresion ';';

tipo : 'numero' | 'decimal' | 'palabra' | 'si_no';

// Condicionales

si : 'si' '(' condicion ')' '{' bloque '}' ('sino' '{' bloque '}')?;

// Ciclos

darvueltas : 'repite' '(' expresion ')' '{' bloque '}';

mientras : 'mientras' '(' condicion ')' '{' bloque '}';

// Funciones

hacerFuncion: 'funcion' ID '(' parametros? ')' '{' bloque '}';

parametros : ID (',' ID)*;

// Expresiones

expresion : termino (('+'|'-') termino)*;

termino : factor (('*'|'/') factor)*;

factor : '(' expresion ')' | NUMERO | ID | DECIMAL;

condicion : expresion ('=='|'!='|'<'|'>'|'<='|'>=') expresion;

// Terminales

NUMERO : [0-9]+; // Match digits

DECIMAL : [0-9]+ '.' [0-9]+;

ID : [a-zA-Z_][a-zA-Z_0-9]*; // Valid identifiers

// Definir la regla 'texto' para que coincida con el texto citado

texto : ('"' ~'"'* '"') | ('\'' ~\'"* \'');

// Ignorar espacios y comentarios

WS : [\t\r\n]+ -> skip;

COMENTARIO : '#' ~[\r\n]* -> skip;