Simple Python-like Language Design for (FOR LOOP) Using PLY

Overview

This project aims to design a simple programming language with syntax similar to Python. The language supports basic control flow structures such as loops and function calls.

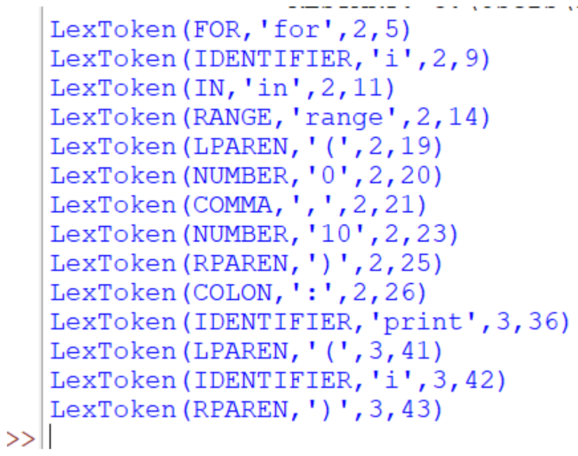
Features

- Syntax Similarity to Python: The language syntax closely resembles Python, making it easy for Python developers to understand and use.
- Support for 'for' Loops: The language includes support for 'for' loops with the ability to iterate over a range of numbers.
- Basic Error Handling: The lexer and parser include basic error handling to catch syntax errors in the code.
- Intermediate Code Generation: The parser generates intermediate representation (IR) of the code, making it easier to perform further optimizations or translate into other languages.
- Code Generation: The project includes a code generation module that translates the intermediate representation into executable Python code.

LANGUAGE COMPONENTS

➢ Lexer

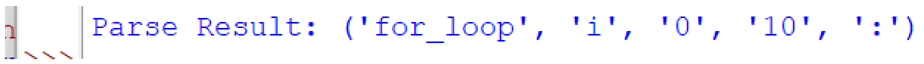
The lexer defines token rules to tokenize input code. It recognizes keywords, identifiers, numbers, and punctuation symbols such as parentheses and commas.



```
LexToken(FOR,'for',2,5)
LexToken(IDENTIFIER,'i',2,9)
LexToken(IN,'in',2,11)
LexToken(RANGE,'range',2,14)
LexToken(LPAREN,'(',2,19)
LexToken(NUMBER,'0',2,20)
LexToken(COMMA,',',2,21)
LexToken(NUMBER,'10',2,23)
LexToken(RPAREN,')',2,25)
LexToken(COLON,':',2,26)
LexToken(IDENTIFIER,'print',3,36)
LexToken(LPAREN,'(',3,41)
LexToken(IDENTIFIER,'i',3,42)
LexToken(RPAREN,')',3,43)
>>
```

➢ Parser

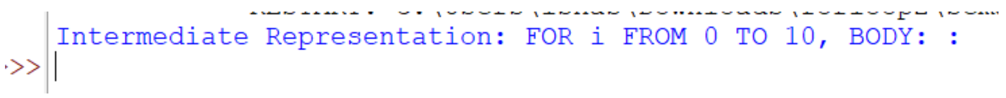
The parser defines syntactical rules to parse the tokenized code and generate an abstract syntax tree (AST). It recognizes for loop structures and function call statements within the loop body.



```
Parse Result: ('for_loop', 'i', '0', '10', ':')
```

➢ Semantic Analysis and Intermediate Code Generation

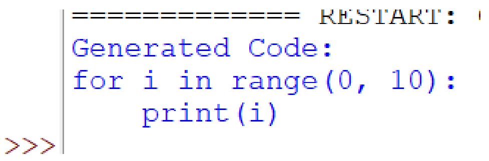
Semantic analysis checks the correctness of the parsed code and generates intermediate representations. In this implementation, the semantic analysis includes verifying for loop structures and generating intermediate representations of loops.



```
Intermediate Representation: FOR i FROM 0 TO 10, BODY: :
>>>
```

➢ Code Generation

Code generation translates the intermediate representations into executable Python code. It generates Python code equivalent to the input loop structure, allowing execution of the defined loop behavior.



```
============= RESTART:
Generated Code:
for i in range(0, 10):
    print(i)
>>>
```

Usage

To use the custom language, follow these steps:

Define Input Code: Write the desired loop structure and function calls in the custom language syntax.

Run Lexer and Parser: Execute the lexer and parser to tokenize and parse the input code, respectively.

Semantic Analysis and Intermediate Code Generation: Perform semantic analysis and generate intermediate representations of the parsed code.

Code Generation: Translate the intermediate representations into executable Python code using the provided code generation function.

Execute Generated Code: Execute the generated Python code to observe the behavior of the defined loop structure and function calls.

Dependencies

- Python 3.x
- PLY (Python Lex-Yacc) library