Javascript Interpreter with Python

Arghyadeep Giri
Jiahua You

What is an Interpreter? (Interpreter vs Compiler)

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

JavaScript is an interpreted language, not a compiled **language**. A program such as C++ or Java needs to be compiled before it is run. The source code is passed through a program called a compiler, which translates it into bytecode that the machine understands and can execute.

Figure: Compiler

Figure: Interpreter

Parts of an interpreter

Lexer

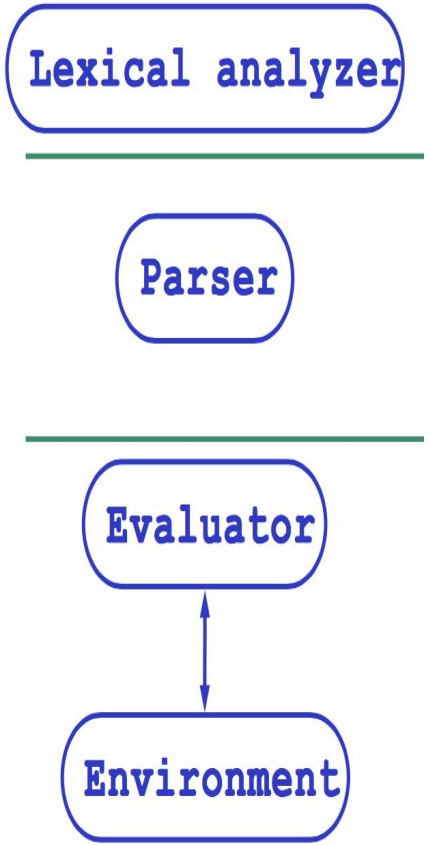
- Break up input string into “words” called tokens

Parser

- Convert the linear sequence of tokens to a tree
- Like diagramming sentences in elementary school
- Also convert self-evaluating tokens to their internal values

Evaluator

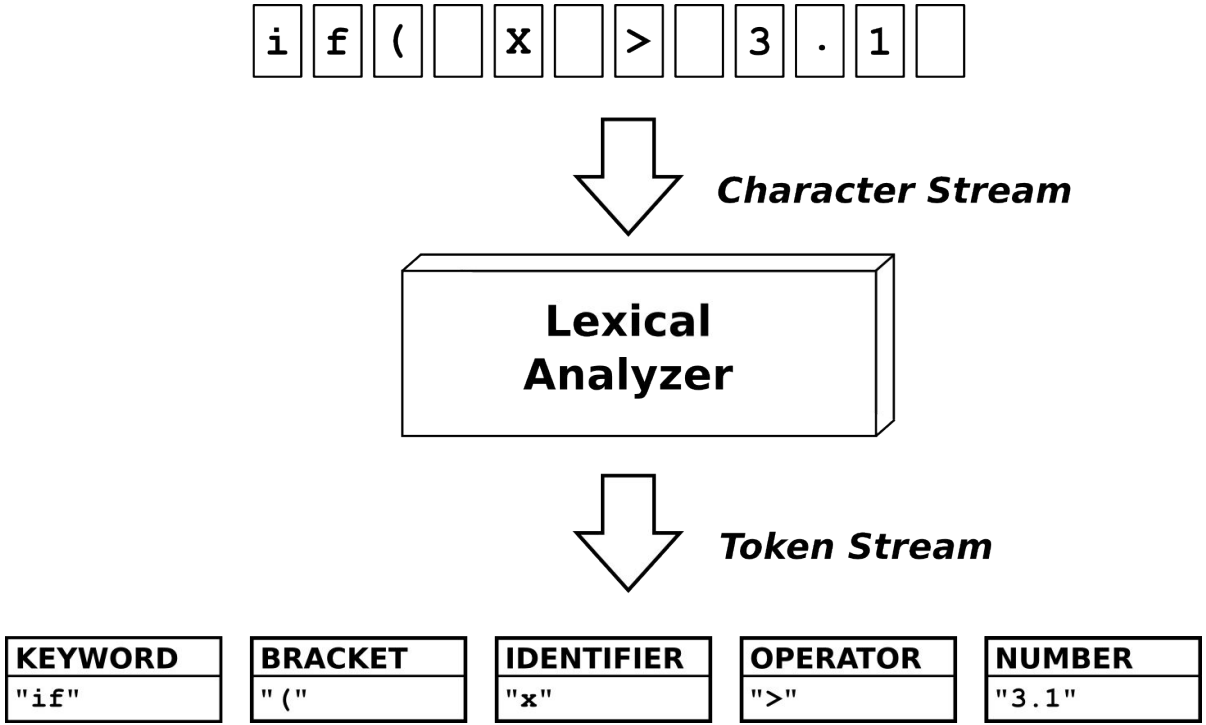
- Follow the language rules to convert parse tree to a value
- Read and modify the environment as needed



How does the lexer work?

It calls the scanner to get characters one at a time and organizes them into tokens and token types.

Thus, the **lexer** calls the scanner to pass it one character at a time and groups them together and identifies them as tokens for the language parser (which is the next stage).

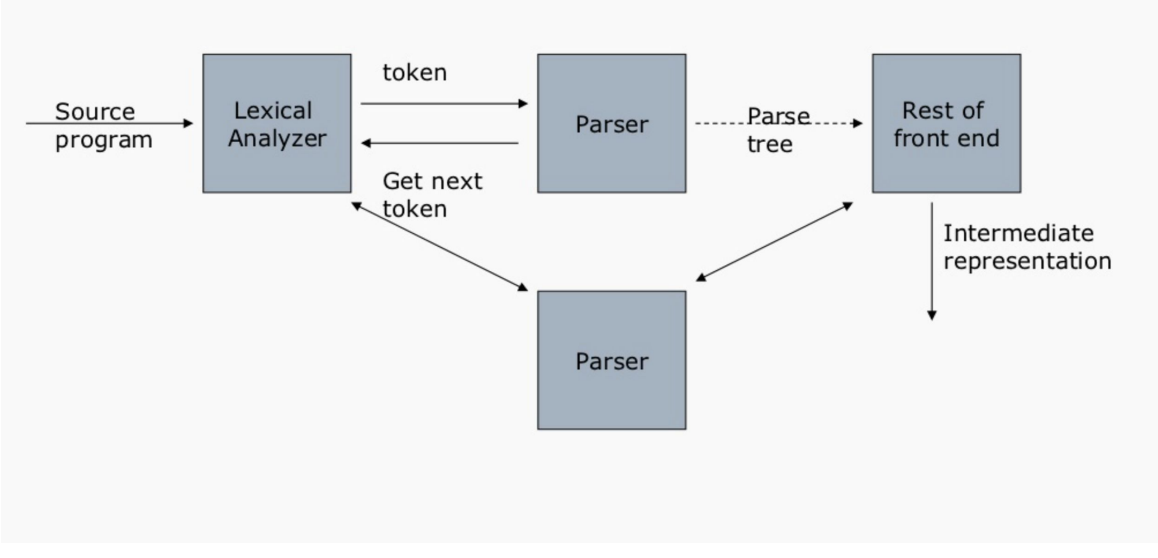


The function of the parser

Parser functions: reads input from a text stream. When it succeeds, it returns a result value (e.g. a parsed number or an AST node); when it fails, it returns error messages describing what went wrong.

The role of the Parser: - Where the lexical analysis splits the input into tokens, the purpose of syntax analysis (also known as parsing) is to recombine these tokens to reflect the data structure of the text.

The parse must also reject invalid texts by reporting syntax errors, and recover from commonly occurring errors so that it can continue processing the remainder of its input.



Lexer implementation in Python

Removing comments and white spaces

'/*(.|\n)*?*/' *multi line*

'//.*\n' *single line*

‘\s+’ *white space*

Identifier and keywords

'[A-Za-z_][A-Za-z0-9_]*'

Strings

"""([^\\\n]|(\\.))*?"|'([^\\\n]|(\\.))*?'"""

Example:

var a;

a = 23;

Produces the following tokens:

```
========== RESTART: C:\Users\Arghyadeep\Desktop\interpreter demo.py
Token('NAME', 'var')
Token('NAME', 'a')
Token('SEMICOLON', ';')
Token('NAME', 'a')
Token('ASSIGN', '=')
Token('NUMBER', '23')
>>>
```

Challenges

Invalid characters or keywords:

E.g. “variable” is not a keyword and should not occur before any variable.

E.g. Making sure that variable names does not start with numbers or does not contain invalid characters.

Handling reserved keywords:

E.g. var getting tokenized as “name” token

Parser implementation in python

Implemented with an AST class

The tokens from the lexer are streamed and hence the blocks of the program are created.

Production rules example:

main:statements
statements: statements statement
statement: expr SEMICOLON
expr: NUMBER
expr: expr PLUS expr

Challenges

Parsing regular expressions

Operations and variables mismatch

Handling different types of statements(variable declaration, assigned statements compound statements, etc)

Evaluator implementation in Python

Key challenges:

Scopes: creating a scoped symbol table

Handling nested scopes

function symbols and return types

semantic analysis

Final evaluation

References

1. https://eloquentjavascript.net/12_language.html
2. https://ruslanspivak.com/lsbasi-part1/
3. http://adrianton3.github.io/blog/art/jinter/jinter