ANTLR (v4) With Python3

A (hopefully useful) introduction to meta programming.

Who am I ?

Viresh Gupta | Just another programmer :)

virresh.github.io

What is meta programming ?

It’s a program which takes another program as it’s input.

That is, you write a program which can modify the source of another program !

Example ?

A Compiler !
Converts <insert any compiled language here> source into machine code !

There are infinite possibilities !
We’ll see some of them later on.

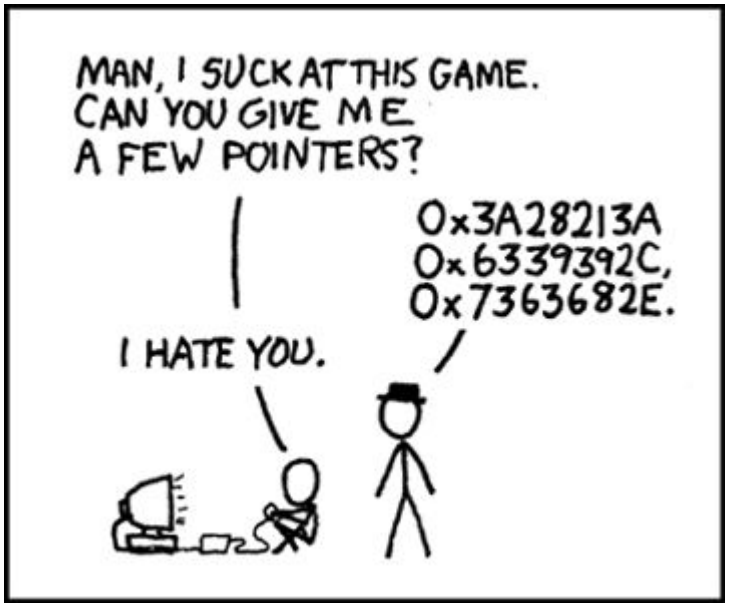
How is any programming language made ?

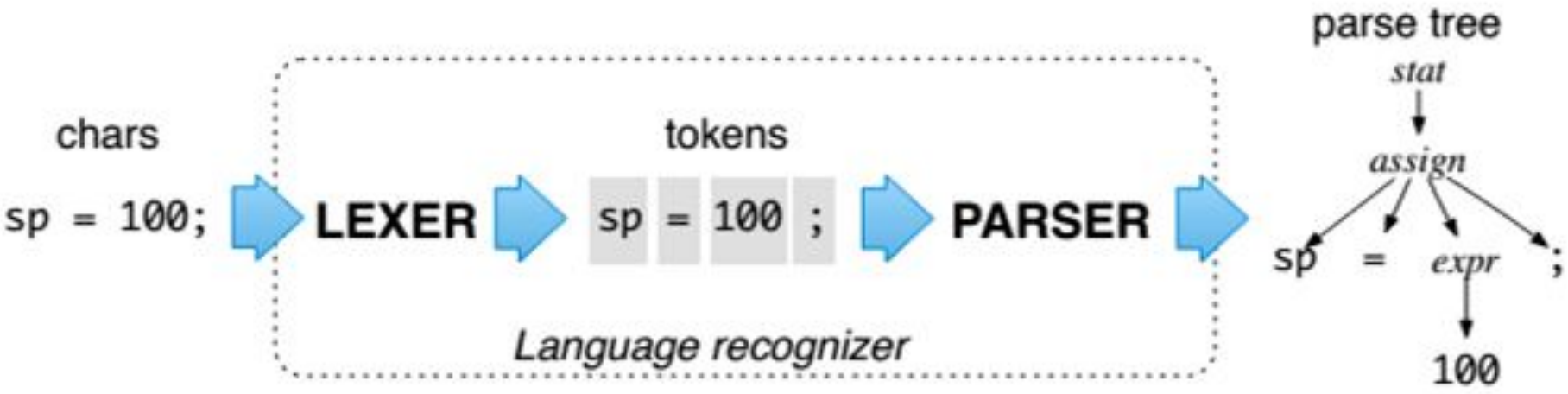
Define some rules and syntax.

Recognize the input program as valid according to above rules or not.

Parse the valid programs and generate output.

Voila !
We just created our own programming language




chars
sp = 100;
LEXER
tokens
sp = 100 ;
PARSER
Language recognizer
parse tree
stat
assign
sp
=
expr
;
100

What is a grammar ?

The rules of a language !
The same holds true for a programming language.

We can define our own set of rules called a grammar. Any input string following those rules is a valid and acceptable input.

Sounds like Finite State Machines ?
(Don’t worry if you don’t know what they are, But do take a ToC course at university ;)).

Example grammar - The Integer expressions Grammar

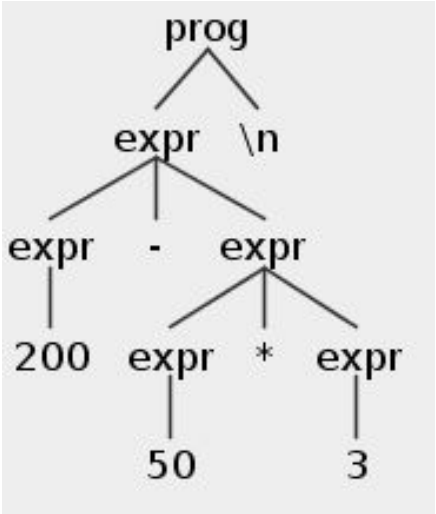
```
grammar Expr;                          <- Grammar Name
prog:     (expr NEWLINE)* ;            <- Grammar Root
expr:     expr ('*'|'/') expr          <- Giving precedence to * and /
    |     expr ('+'|'-') expr          <- telling how an expr can contain an expr
    |     INT                          <- Finally it has to end with an integer
    |     '(' expr ')'                 <- Brackets are acceptable
    ;
NEWLINE : [\r\n]+ ;                    <- Delimiters of our program
INT     : [0-9]+ ;                     <- Acceptable list of “integers”
```

What’s a parse tree ?

Parse tree is a verbatim conversion of input into a tree according to the rules defined by a grammar.

For example, let’s see the parse tree of the following example generated by our expressions grammar:

```
200-50*3
```



Visitors and Listeners

Visitors are a very flexible mechanism for traversing through the parse trees. Although we are responsible for invoking all function calls, we are in full control of which all nodes are traversed and what all events occur in which order.

It's highly flexible and useful for understanding how the traversal is working.

Listeners are another mechanism by which we can traverse through the parse trees. It's not very flexible, but is great for generic functionalities. We need to define an event, and tell the parser to start parsing.

It will traverse through the entire tree and as soon as the required event is triggered, our code is called, executed and then the tree traversal continues.

What is ANTLR ?

ANTLR is a tool for Language Recognition ! (Recall the second step for building a language)

It provides useful APIs for building parsers from grammars quickly and provides a runtime for a variety of target languages like C#, Python, JavaScript etc.
(Although it's mainly written in Java)

It has following components

- A generator for the recognizer and a set of sample parser that "does nothing".
  This is a JAR file, and you need java for this to work.
- Recognizer and a runtime, to validate the input program according to a given set of rules.
  This can be done in any supported language. We will use Python.

Setting up ANTLR

Follow official instructions at https://www.antlr.org/

Steps for setting up ANTLR

```
cd /usr/local/lib
sudo wget https://www.antlr.org/download/antlr-4.7.2-complete.jar
export CLASSPATH=".:/usr/local/lib/antlr-4.7.2-complete.jar:$CLASSPATH"
alias antlr4='java -jar /usr/local/lib/antlr-4.7.2-complete.jar'
alias grun='java org.antlr.v4.gui.TestRig'
cd -
```

Create Expr.g4 file now and put in the grammar rules from our example grammar. Also install the python runtime using:

```
pip install antlr4-python3-runtime
```

Let’s build a calculator !

Generate sample recognizers and parsers

First we need a Lexer to break down our input into tokens and a parser to convert these tokens into a parse tree.
ANTLR can generate this code for us fortunately !

```
git clone https://github.com/virresh/antlr_integer_calc.git
cd antlr_integer_calc
antlr4 -Dlanguage=Python3 grammar/Expr.g4 -visitor -Xexact-output-dir
-o ./python_files/
```

Note: Steps after cloning are already performed for you in the git repository, but still in case you need to, you can reproduce using the antlr's jar file downloaded while setting up ANTLR.

Visualize your expression (optional)

If you have java installed and set-up as shown previously, it's easy to visualise the parse tree of any input for a given grammar.

For this you'll have to generate the JAVA target files, i.e:

```
antlr4 Expr.g4
javac Expr*.java
grun Expr prog -gui
```

Enter your input at STDIN and press CTRL D (to signal end of file) so that ANTLR can start the process for rendering the parse tree.

Some Additional Use Cases

Linting | Transpiling

What is Linting ?

```
if my_long_variable == 5:

        print(“A very very very
long sentence which gets hard to
read plus 8 spaces in the
beginning :(”)
```

```
if my_long_variable == 5:

    print(“A very very very”,

          “long sentence which”,

          “is now easy to read”,

          “with 4 spaces in the”

          )
```

What is Linting ?

A program that can analyse your code and report errors / inconsistent coding styles inside that code.

Examples -
JSHint
AutoPEP8
PyLint
CPPLint

They are extensively used at almost all big companies for enforcing coding standards and making code maintainable in the long run.

What is Transpiling ?

Converting source code from <language A> to <language B> . This is a subset of compilers in the sense that the code so generated still needs a compiler of <language B> in order to be executed by a machine.

Examples -

Babel
Emscripten

Thank You !

References

Page 4 Figure: https://prateekvjoshi.com/2014/04/05/what-is-metaprogramming-part-22/

Page 5 Figure: http://blog.ptsecurity.com/2016/06/theory-and-practice-of-source-code.html

Feedback

https://bit.ly/antlrpyjul20feedback