

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



ANTLR with Python



Sumeet · [Follow](#)

8 min read · Nov 29, 2023



17



2



If you don't know what ANTLR is, here is what the [official site](#) says about it:

ANTLR (ANother Tool for Language Recognition) is a powerful parser generator for reading, processing, executing, or translating structured text or binary files. It's widely used to build languages, tools, and frameworks. From a grammar, ANTLR generates a parser that can build and walk parse trees.



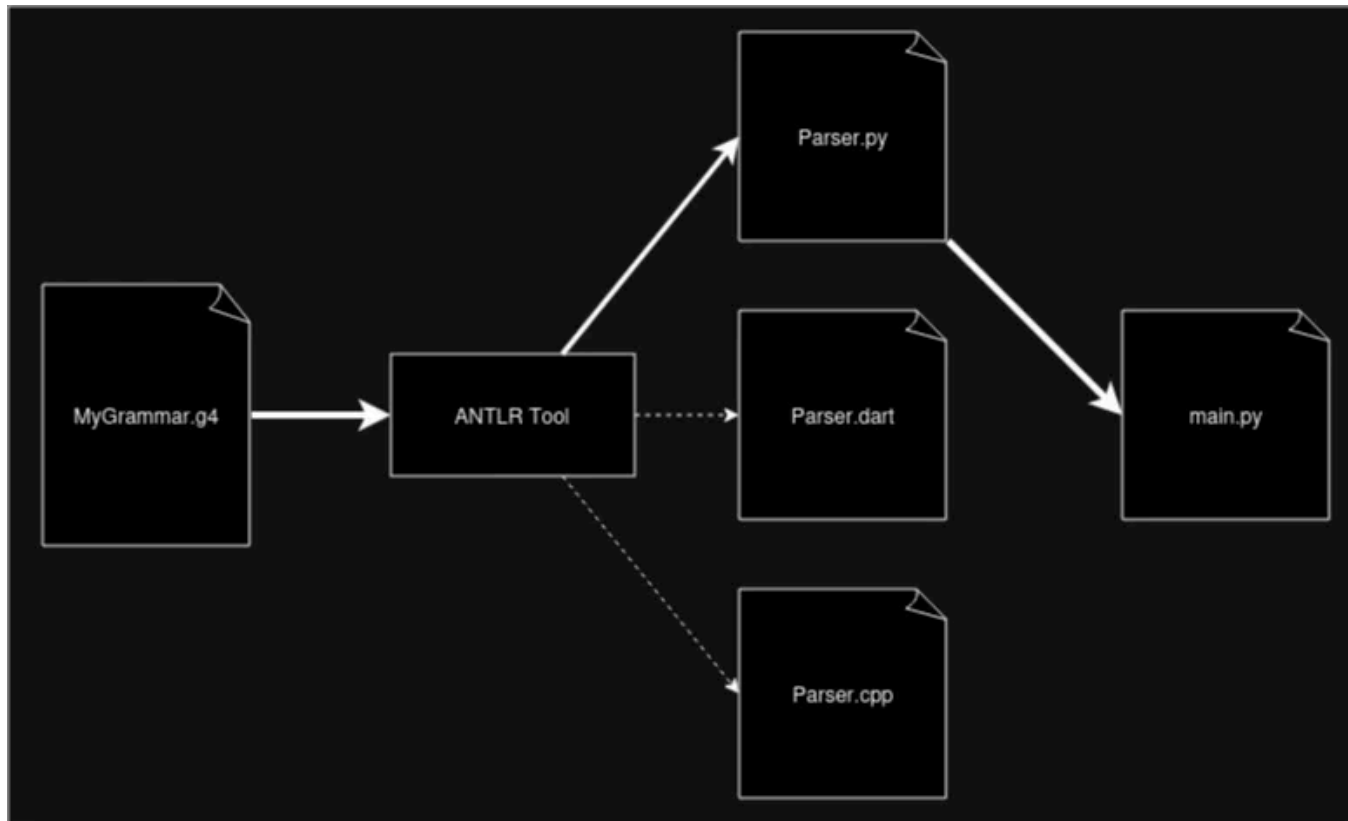
I'm writing this article because I couldn't find any good article/blog explaining how to use it with Python. The book for ANTLR 4, called "[The Definitive ANTLR 4 Reference](#)" is written by the ANTLR 4's creator Terence Parr, but it's written for Java, and I don't like Java.

Examples in this article are from Debian GNU/Linux. The example shown here can be followed for macOS and other Linux distributions as well, with no or minor changes.

ANTLR and Python Setup

ANTLR has two parts

1. **ANTLR tool**, which will generate a parser from grammar. This is written in JAVA, and hence, you need to have java installed on your machine.
2. **ANTLR runtime library**, the ANTLR tool can generate parser in many languages (Python in this case). To use the parser in our programs, we need a runtime library.



ANTLR usage

Installing ANTLR tool

To get ANTLR tool, we just need to download the `.jar` file.

As we can see in above diagram, we pass our grammar as an input to the ANTLR tool, and it generated parser for many different languages.

Execute the following two command, this will download the `antlr-4.13.1-complete.jar` into `/usr/local/lib` directory.

You can store the `.jar` anywhere you want, but then you need to adjust the next commands.

```
$ cd /usr/local/lib
$ curl -O http://www.antlr.org/download/antlr-4.13.1-complete.jar
```

Add that path in the `CLASSPATH`, so that Java can find that `.jar` i.e. ANTLR tool.

```
$ export CLASSPATH="/usr/local/lib/antlr-4.13.1-complete.jar:$CLASSPATH"
```

Check if everything executed correctly. If you get the same response from both the following commands, then it means that ANTLR tool installed correctly.

```
$ java -jar /usr/local/lib/antlr-4.13.1-complete.jar
ANTLR Parser Generator Version 4.13.1
-o ___          specify output directory where all output is generated
-lib ___        specify location of grammars, tokens files
```

```
-atn                generate rule augmented transition network diagrams
-encoding ___       specify grammar file encoding; e.g., euc-jp
...
...

$ java org.antlr.v4.Tool
ANTLR Parser Generator  Version 4.13.1
-o ___              specify output directory where all output is generated
-lib ___            specify location of grammars, tokens files
-atn                generate rule augmented transition network diagrams
-encoding ___       specify grammar file encoding; e.g., euc-jp
...
...
```

To use ANTLR tool, we have to use any of the above commas, and none of them is easy to type. So, let's create an alias for that.

```
$ alias antlr4=' java -jar /usr/local/lib/antlr-4.13.1-complete.jar'
```

After creating alias, the command `antlr4` should give the same output as above.

Installing ANTLR4 Python Runtime

Installing runtime is easy, just download python package from PyPI

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

That's all we have to do here.

Hello ANTLR4

Let's write a simple grammar, which recognizes the language “Hello <any_string>”, and create generate parser for that language in Python3.

The example is taken from the book [The Definitive ANTLR 4 Reference](#).

The grammars for ANTLR are written in the file with extension `.g4`

Create a file names `Hello.g4` , and write following grammar in it.

```
grammar Hello;  
r : 'hello' ID ;  
ID : [a-z]+ ;  
WS : [ \t\r\n]+ -> skip;
```

Language Explanation

1. Parsing rules are written in small alphabets. Example `r` which says that any string which is in the form of `hello <ID>` belongs to language.
2. Lexical rules are written in capital alphabets. Example, `ID` , which says `ID` is any string which composed of 1 or more letters between a to z .
3. Lexical rules are represented using regular expressions.
4. The lexical rule, `ws` says that, skip one or more spaces, tabs, returns, and newlines.

Using Language

Let's generate a parser for our small `Hello` language in Python3.

The parameter `-Dlanguage=Python3` tells ANTLR4 that we want our parser in Python3.

```
$ antlr4 -Dlanguage=Python3 Hello.g4
```

Now, if you list the files, you should have your parser in Python3.


```
$ ls
Hello.g4  Hello.interp  HelloLexer.interp  HelloLexer.py  HelloLexer.tokens  Hel
```

We shall understand how to use all the files later, for now, let's write a python program which parses the `Hello` language.

Create `main.py` file in the same directory.

```
from antlr4 import *
from HelloLexer import HelloLexer
from HelloParser import HelloParser

input_text = input("> ")
lexer = HelloLexer(InputStream(input_text))
stream = CommonTokenStream(lexer)
parser = HelloParser(stream)

tree = parser.r()

print(tree.toStringTree(recog=parser))
```

In the line `tree = parser.r()`, the `r` is there because we have used the `r` as initial rule in our language.

Let's try to run it.

```
$ python3 main.py  
> hello world  
(r hello world)  
  
$ python3 main.py  
> hello abc  
(r hello abc)
```

Our languages are getting recognized.

Now let's see what happens if we add an invalid string. If we remember correctly, only small alphabets are allowed in place of ID. So, if we any capital alphabet, our parser gives error.

```
$ python3 main.py  
> hello World  
line 1:6 token recognition error at: 'W'  
(r hello orld)
```

Even if we misspell `hello` we shall get error.

```
$ python3 main.py
> Hello world
line 1:0 token recognition error at: 'H'
line 1:1 missing 'hello' at 'ello'
(r <missing 'hello'> ello)
```

If we look closely, the parser is able to know where exactly the error is. This is how our compilers/linters give us line number where there is an error in our program.

Creating Morse Code Translator

Now let's try to create a translator using ANTLR. This translator will convert Morse code to the string.

Create `Morese.g4` file and paste the following grammar:

```
grammar Morse;

// Parser rules
morse_code : (letter | digit | WS)* ;

letter : A | B | C | D | E | F | G | H | I | J | K | L | M |
        N | O | P | Q | R | S | T | U | V | W | X | Y | Z ;
```

```
digit : ZERO | ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN | EIGHT | NINE ;
```

```
// Lexer rules
```

```
A : '.' ;
```

```
B : '-...' ;
```

```
C : '-.-.' ;
```

```
D : '-..' ;
```

```
E : '.' ;
```

```
F : '..-.' ;
```

```
G : '---' ;
```

```
H : '....' ;
```

```
I : '..' ;
```

```
J : '.---' ;
```

```
K : '-.-' ;
```

```
L : '.....' ;
```

```
M : '---' ;
```

```
N : '-.' ;
```

```
O : '----' ;
```

```
P : '.---' ;
```

```
Q : '---.' ;
```

```
R : '-..' ;
```

```
S : '....' ;
```

```
T : '-' ;
```

```
U : '..-' ;
```

```
V : '...-' ;
```

```
W : '.---' ;
```

```
X : '-.-.' ;
```

```
Y : '-.--' ;
```

```
Z : '---.' ;
```

```
ZERO : '-----' ;
```

```
ONE : '.-----' ;
```

```
TWO : '..-----' ;
```

```
THREE : '.....' ;
```

```
FOUR : '.....' ;
```

```
FIVE : '.....' ;
```

```
SIX : '-....' ;  
SEVEN : '-...-' ;  
EIGHT : '-...-' ;  
NINE : '-...-' ;  
  
WS : [ \t\r\n]+ -> skip ;
```

Understanding grammar

Grammar is pretty easy to understand.

We have the following parser rules.

1. `morse_code` which recognizes any number, or letters or digits.
2. `letter` can be any one of 26 characters.
3. `digit` can be any one of 10 digits.

We have defines Morse code rules for all characters and digits.

Generate Parser

Generate parser for Python3, using following command.

```
$ antlr4 -Dlanguage=Python3 Morse.g4
```

Use Parser

Now that we have generated parser, let's use it.

```
from antlr4 import *  
from MorseLexer import MorseLexer  
from MorseParser import MorseParser  
from MorseListener import MorseListener  
  
input_text = input('> ')  
lexer = MorseLexer(InputStream(input_text))  
stream = CommonTokenStream(lexer)  
parser = MorseParser(stream)  
  
tree = parser.morse_code()  
  
print(tree.toStringTree(recog=parser))
```

Let's try some input.

```
$ python3 main.py  
> .... . .-... .-... ---
```

```
(morse_code (letter ....) (letter .) (letter .-..) (letter .-..) (letter ---))
```

We shall explore more about ANTLR `in` Python `in` some other articles.

Ok, it's recognizing our language.

Write Listener

If you observe carefully, there is another file generated along with lexer and parser Python file.

If we see the `MorseListener.py` file, there is class `MorseListener`. In this class, there are different methods prefixed with `enter` and `exit`.

For example, `enterMorse_code` is executed when the `morse_code` rule in the grammar starts executing, and `exitMorse_code` function will be executed once that `morse_code` rule finished. This works similarly for all other rules like `enterLetter`, `enterDigit`.

We can override these methods and implement whatever we want in there.

Create a class inheriting `MorseListener`, and override `enterMorse_code`, `exitMorse_code`, `enterLetter`, `enterDigit`.

```
class MorseToPythonString(MorseListener):

    def enterMorse_code(self, ctx:MorseParser.Morse_codeContext):
        print(' ', end='')

    def exitMorse_code(self, ctx:MorseParser.Morse_codeContext):
        print(' ', end='')

    def enterLetter(self, ctx:MorseParser.LetterContext):
        for child in ctx.getChildren():
            if child.symbol.type == MorseParser.A:
                print("a", end='')
            if child.symbol.type == MorseParser.B:
                print("b", end='')
            if child.symbol.type == MorseParser.C:
                print("c", end='')
            if child.symbol.type == MorseParser.D:
                print("d", end='')
            if child.symbol.type == MorseParser.E:
                print("e", end='')
            if child.symbol.type == MorseParser.F:
                print("f", end='')
            if child.symbol.type == MorseParser.G:
                print("g", end='')
            if child.symbol.type == MorseParser.H:
                print("h", end='')
            if child.symbol.type == MorseParser.I:
                print("i", end='')
            if child.symbol.type == MorseParser.J:
                print("j", end='')
            if child.symbol.type == MorseParser.K:
                print("k", end='')
            if child.symbol.type == MorseParser.L:
                print("l", end='')
            if child.symbol.type == MorseParser.M:
                print("m", end='')
```



```

        if child.symbol.type == MorseParser.N:
            print("n", end="")
        if child.symbol.type == MorseParser.O:
            print("o", end="")
        if child.symbol.type == MorseParser.P:
            print("p", end="")
        if child.symbol.type == MorseParser.Q:
            print("q", end="")
        if child.symbol.type == MorseParser.R:
            print("r", end="")
        if child.symbol.type == MorseParser.S:
            print("s", end="")
        if child.symbol.type == MorseParser.T:
            print("t", end="")
        if child.symbol.type == MorseParser.U:
            print("u", end="")
        if child.symbol.type == MorseParser.V:
            print("v", end="")
        if child.symbol.type == MorseParser.W:
            print("w", end="")
        if child.symbol.type == MorseParser.X:
            print("x", end="")
        if child.symbol.type == MorseParser.Y:
            print("y", end="")
        if child.symbol.type == MorseParser.Z:
            print("z", end="")

def enterDigit(self, ctx:MorseParser.LetterContext):
    for child in ctx.getChildren():
        if child.symbol.type == MorseParser.ZERO:
            print("0", end="")
        if child.symbol.type == MorseParser.ONE:
            print("1", end="")
        if child.symbol.type == MorseParser.TWO:
            print("2", end="")
        if child.symbol.type == MorseParser.THREE:
            print("3", end="")

```

```
if child.symbol.type == MorseParser.FOUR:  
    print("4", end="")  
if child.symbol.type == MorseParser.FIVE:  
    print("5", end="")  
if child.symbol.type == MorseParser.SIX:  
    print("6", end="")  
if child.symbol.type == MorseParser.SEVEN:  
    print("7", end="")  
if child.symbol.type == MorseParser.EIGHT:  
    print("8", end="")  
if child.symbol.type == MorseParser.NINE:  
    print("9", end="")
```

1. `enterMorse_code` will be executed at the start. It will print the double quote " character.
2. `exitMorse_code` will be executed at the end, and it will print the double quote " .
3. The `enterLetter` will be executed once any letter is encountered. In this function, we're checking which token it has encountered, and according to that, we're printing the character.
4. The `enterDigit` will be executed once any digit is encountered. Similar to the `enterLetter`, we're printing the digit according to the token.

Update the code

Use Walker

Let's now create an object of `ParseTreeWalker`. The `walk` method is used to walk the tree, to that method, pass the listener class and the tree. Listener class will be out new listener class `MorseToPythonString`.

```
input_text = input('> ')
lexer = MorseLexer(InputStream(input_text))
stream = CommonTokenStream(lexer)
parser = MorseParser(stream)

tree = parser.morse_code()

walker = ParseTreeWalker()
walker.walk(MorseToPythonString(), tree)
```

Try Out

Now that we have walker in place, let's try executing it.

```
$ python3 main.py
> .... . .-.. .-.. ---
"hello"
```

As we can see, our program translating the input correctly.

Antlr4

Parser

Parser Generator

Grammar

Python



Written by Sumeet

9 Followers

Software Engineer, love Linux FOSS, Golang, Python.

Follow



More from Sumeet



Sumeet

Using JSON Config in Go

In this article, we're going to see how we can use json files an config for out Go...

Apr 7



Sumeet

RegEx in Golang


Apr 6



See all from Sumeet

Recommended from Medium

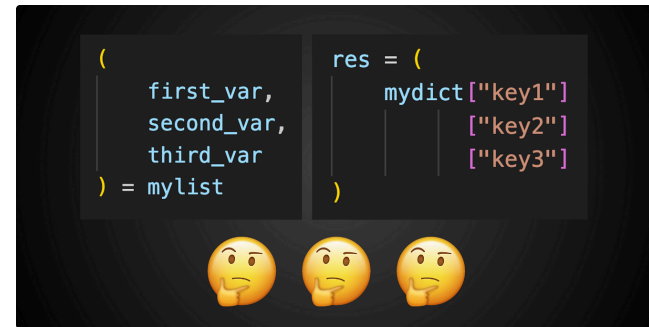


 Abdur Rahman in Stackademic

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

★ Oct 22 🖱 4.6K 💬 23 📖 ⋮



 Liu Zuo Lin in Level Up Coding

12 Production-Grade Python Code Styles I've Picked Up From Work

Read Free...

★ Nov 2 🖱 1.9K 💬 19 📖 ⋮

Lists



Coding & Development

11 stories · 894 saves



Predictive Modeling w/ Python

20 stories · 1649 saves



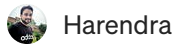
Practical Guides to Machine Learning

10 stories · 2011 saves



ChatGPT

21 stories · 864 saves

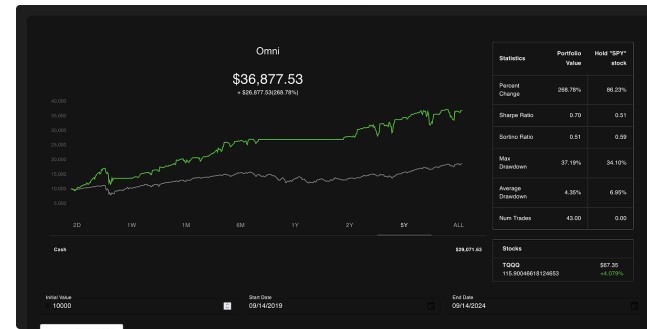


Harendra

How I Am Using a Lifetime 100% Free Server

Get a server with 24 GB RAM + 4 CPU + 200 GB Storage + Always Free

★ Oct 25 🖱️ 3.8K 💬 47 📌 ⋮

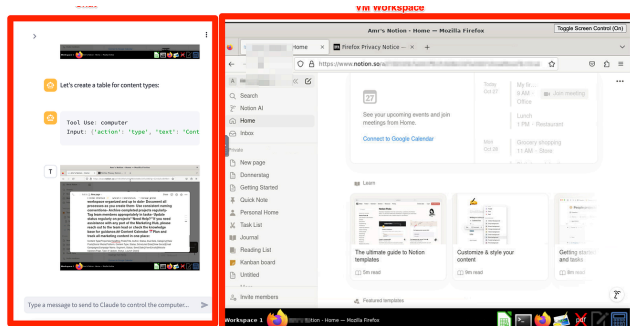


Austin Starks in DataDrivenInvestor

I used OpenAI's o1 model to develop a trading strategy. It is...

It literally took one try. I was shocked.

★ Sep 15 🖱️ 5.9K 💬 148 📌 ⋮



AI Rabbit in CodeX

Has Anthropic Claude just wiped out an entire industry?



Dan in PromptLayer

Prompt Templates with Jinja2

Jinja2 is a powerful templating engine that can take your prompts to the next level. See...

If you have been following the news, you may have read about a new feature (or should I c...

Aug 16



Oct 27



1K



18



See more recommendations

[Help](#)

[Status](#)

[About](#)

[Careers](#)

[Press](#)

[Blog](#)

[Privacy](#)

[Terms](#)

[Text to speech](#)

[Teams](#)