Lesson 02

Introduction to ANTLR

The Basics of ANTLR

Parts of ANTLR

- The tool, used to generate the lexer and parser, is always the same
- The runtime, needed to run the the generated lexer and parser, changes for each language

Example for Python Software

- The Java command line tool
- The Python (2 or 3) runtime

Where to Find ANTLR

https://www.antlr.org/download.html

Instructions

1. Copy the downloaded tool where you usually put third-party java libraries (e.g. `/usr/local/lib` or `C:\Program Files\Java\libs`)
2. Add the tool to your `CLASSPATH`. Add it to your startup script (e.g., `.bash_profile`)
3. (optional) Add also aliases to your startup script to simplify the usage of ANTLR

Linux/Mac Os Instructions

```
// 1.
sudo cp antlr-4.9.2-complete.jar /usr/local/lib/
// 2. and 3.
// add this to your .bash_profile
export CLASSPATH = ".:/usr/local/lib/antlr-4.9.2-complete.jar:$CLASSPATH"
// simplify the use of the tool to generate lexer and parser
antlr4='java -Xmx500M -cp "/usr/local/lib/antlr-4.9.2-complete.jar:$CLASSPATH"
org.antlr.v4.Tool'
// simplify the use of the tool to test the generated code
alias grun='java org.antlr.v4.gui.TestRig'
```

Windows Instructions

```
// 1. Copy antlr-4.9.2-complete.jar in C:\Program Files\Java\libs (or wherever you prefer)

// 2. Create or append to the CLASSPATH variable the location of antlr

// you can do to that by pressing WIN + R and typing sysdm.cpl, then selecting Advanced (tab) > Environment variables > System Variables

// CLASSPATH -> .;C:\Program Files\Java\libs\antlr-4.9.2-complete.jar;%CLASSPATH%

// 3. Add aliases

// create antlr4.bat

java org.antlr.v4.Tool %*

// create grun.bat

java org.antlr.v4.gui.TestRig %*

// put them in the system PATH or any of the directories included in your PATH
```

ANTLR Workflow

1. Write the grammar (a .g4 file)
2. Generate the parser with the antlr4 tool
   - `antlr4 <options> <grammar-file-g4>`
3. Develop as usual

ANTLR Command Line Options

The most important options:

- `-Dlanguage` to specify the target language
- `-visitor, -listener, -no-visitor, -no-listener` to regulate the generation of visitor and listener

Example to Generate a C# parser

```
antlr4 -Dlanguage=CSharp <grammar-file-g4>
```

Target Languages Supported

- Java
- C# (using the option Dlanguage=CSharp)
- Python (2 and 3) (using the option Dlanguage=Python2 and Dlanguage=Python3)
- JavaScript (using the option Dlanguage=JavaScript)
- Go (using the option Dlanguage=Go)
- C++ (using the option Dlanguage=Cpp)
- Swift (using the option Dlanguage=Swift)
- PHP (using the option Dlanguage=PHP)
- Dart (using the option Dlanguage=Dart)

Error: Language Not Supported

```
$ antlr4 -Dlanguage=csharp Hello.g4
error(31):  ANTLR cannot generate csharp code as of version 4.9.1
```

Options values are case-sensitive!

Unofficial Target Languages

Anybody can make a target language.

- There is one for TypeScript, created by Sam Harwell
  - https://github.com/tunnelvisionlabs/antlr4ts
- One for Kotlin, created by us
  - https://github.com/strumenta/antlr-kotlin

ANTLR Command Line Options

The most important options:

- `-Dlanguage` to specify the target language
- `-visitor, -listener, -no-visitor, -no-listener` to regulate the generation of visitor and listener

Visitor/Listener Options

```
$ antlr4 Hello.g4 –visitor –no-listener
// generates the visitor, but not the listener
$ antlr4 Hello.g4 –no-visitor -listener
// generates the listener, but not the visitor (this is the default)
$ antlr4 Hello.g4 –visitor
// generates both the listener and the visitor
$ antlr4 Hello.g4 –visitor -listener
// generates both the listener and the visitor
```

Encoding Option

`-encoding <name-of-the-encoding>`

- The encoding option must be used if the input is written in an encoding different from the default Unicode
- It does not alter the input, it just informs ANTLR that the encoding is not Unicode

Encoding Option

```
$ antlr4 -Dlanguage=Python3 HelloISO.g4 -encoding ISO-8859-15
// generates the lexer and parser for Python3 to handle ISO-8859-15 encoding
```

Encoding Option

If you select the wrong encoding, you might get errors right when you generate the grammar.

```
$ antlr4 -Dlanguage=Python3 HelloUTF8.g4 -encoding ISO-8859-15
// this grammar file does not uses the encoding UTF-8
error(50): Lesson02/HelloUTF8.g4:1:0: syntax error: 'ï' came as a complete surprise to me
error(50): Lesson02/HelloUTF8.g4:1:1: syntax error: '»' came as a complete surprise to me
error(50): Lesson02/HelloUTF8.g4:1:2: syntax error: '¿' came as a complete surprise to me
error(50): Lesson02/HelloUTF8.g4:1:3: syntax error: mismatched input 'grammar' expecting SEMI
```

Other Options

- `-package <name-of-the-package>`

package is Java terminology, but many languages support similar concepts

- `-lib / -o <location>`

-lib is to set the location of the grammars, -o to set the location of the generated parser and lexer

Other Options

- `-atn`

Used to generated DOT files representing the Augmented Transition Network (ATN) used by ANTLR. Use it for debugging and if you are curious to see how ANTLR works.

Generate DOT Files

```
$ antlr4 Hello.g4 -atn
```

This generates a DOT file for every rule in the Lexer in the form

- HelloLexer.NAME-OF-THE-RULE.dot

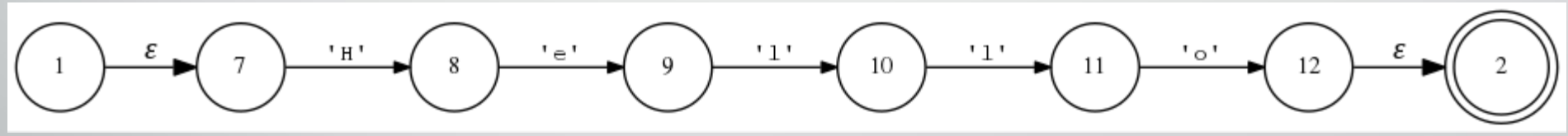
and a DOT file for every rule in the Parser in the form

- Hello.name-of-the-rule.dot

Generate DOT Files

```
// download and install GraphViz to get the dot program
$ dot HelloLexer.HELLO.dot -Tpng -o HelloLexer.HELLO.png
```

This generates a PNG image for the HELLO rule of the Lexer, like this one



Grun

```
grun <grammar-name> <rule-to-test> <input-filename(s)>
```

ANTLR comes with a little utility named TestRig that you can use to test your grammar.

As we have seen in the setup section, the utility is usually aliased to grun.

Grun can work only on a parser generated in Java.

When to Use Grun

Grun is useful at the beginning of your development.

When you are testing manually the first draft of your grammar and you still don’t have your software.

Grun

```
grammar       Hello;

HELLO:        'Hello';
NAME:         [a-zA-Z]+;
WHITESPACE:   ' ';
greeting:     HELLO WHITESPACE NAME;
```

Grun

```
$ antlr4 Hello.g4
// after we have generated the lexer and parser in Java we compile the code
// For this command to work javac(.exe), available in the JDK, must be in your PATH
$ javac Hello*.java
$ grun Hello greeting
// we then digit
Hello John
// and CTRL+D (Linux/Mac) or CTRL+Z (Windows) to indicate the end of the input
```

Grun

If we didn't make any mistake and the input is valid for the rule we are not going to receive any feedback.

Grun

```
$ grun Hello greeting
// we then digit
hello John
// and CTRL+D (Linux/Mac) or CTRL+Z (Windows) to indicate the end of the input
// we should receive the following error
line 1: 0 mismatched input 'hello' expecting 'Hello'
```

Grun

Grun ha three useful options:

- `-tokens`, that output the tokens found by the parser
- `-gui`, that output an image of the parse tree generated by the parser
- `-diagnostics`, that notifies you about potential issues in your grammar

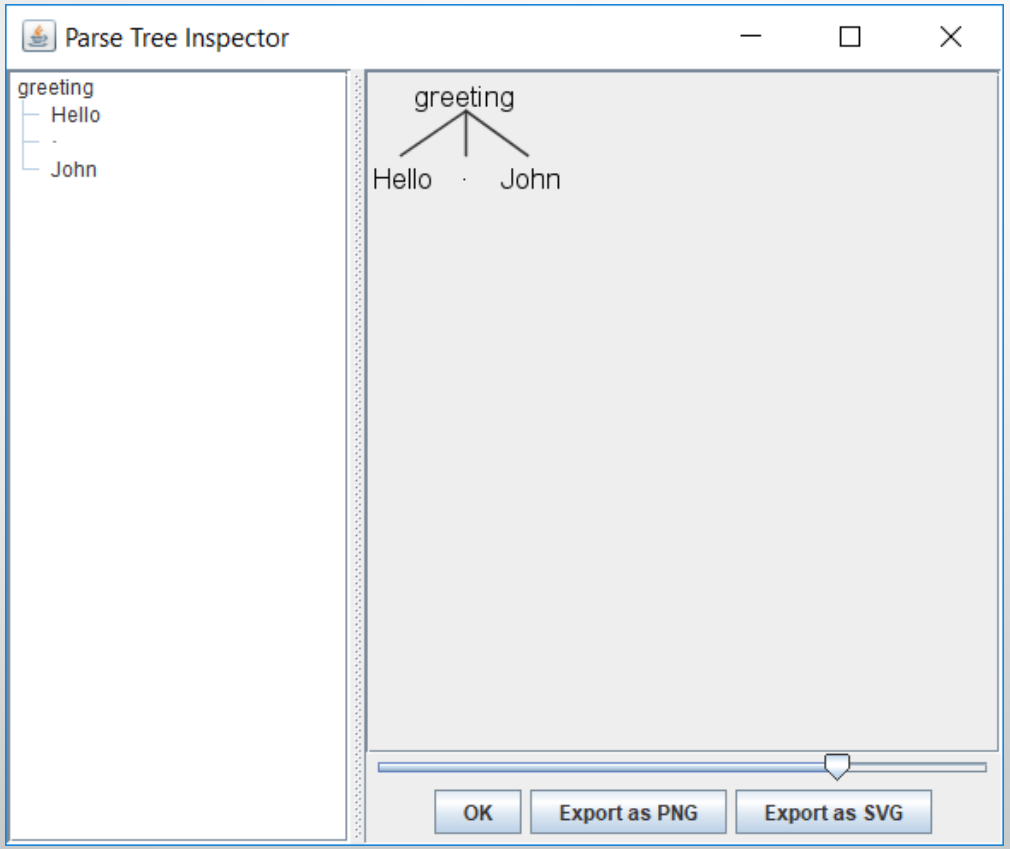
Grun

```
$ grun Hello greeting -tokens
// we then digit
Hello John
// and CTRL+D (Linux/Mac) or CTRL+Z (Windows) to indicate the end of the input
// we should see
[@0,0:4='Hello',<'Hello'>,1:0]
[@1,5:5=' ',<' '>,1:5]
[@2,6:9='John',<NAME>,1:6]
[@3,10:9='<EOF>',<EOF>,1:10]
```

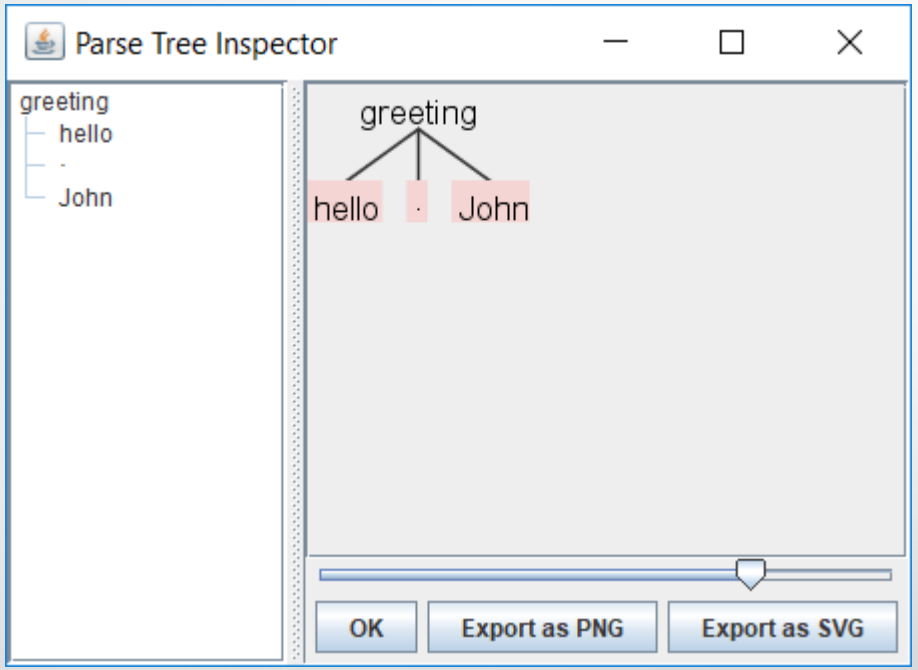
Grun

```
$ grun Hello greeting -gui
// we then digit
Hello John
// and CTRL+D (Linux/Mac) or CTRL+Z (Windows) to indicate the end of the input
// we should see
```

Grun


Parse Tree Inspector
greeting
Hello
,
John
greeting
Hello , John
OK
Export as PNG
Export as SVG

Grun



Grun with errors in the input

ANTLR and Python

ANTLR and Python

```
$ pip install antlr4-python2-runtime
$ pip3 install antlr4-python3-runtime
```

The ANTLR runtime is available from PyPi, so it's easy to install it

ANTLR and Python

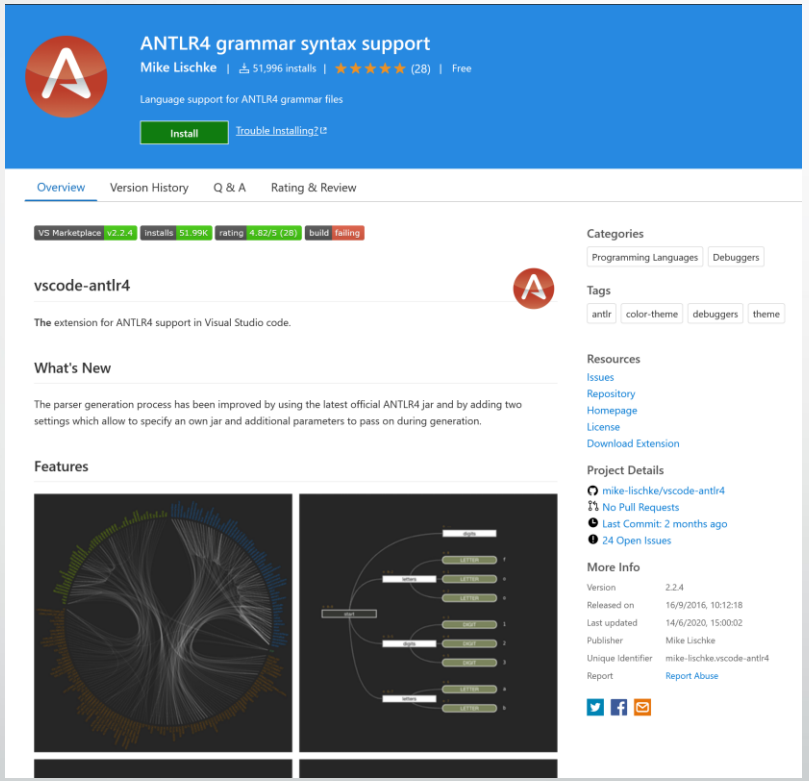
```
// typical ANTLR workflow with Python
// you generate the Python parser and lexer from the grammar
$ antlr4 -Dlanguage=Python3 Hello.g4
// then you use Python as usual
python3 your_software.py
```

Visual Studio Code

Why Visual Studio Code

- It is cross-platform and available for Linux, MacOs and Windows
- It has an extension for Python
- It has a great extension for ANTLR
  - https://marketplace.visualstudio.com/items?itemName=mike-lischke.vscode-antlr4

Visual Studio Code Extension



Visual Studio Code

- Example user settings
  - "antlr4.generation.mode": "external"
  - "antlr4.generation.language": "Python3"
- Example project settings
  - "antlr4.generation.listeners": true
  - "antlr4.generation.visitors": true