# Question 1

### Workshops

Workshops: <https://github.com/JayNightmare/Language-Development-Practice>

* Request Access via Outlook email

### Antlr Setup Process

Setting up the environment was hard to begin with as I had not used Antlr before and was not formular with setup process. I spent the first few hours going through the slides and back and forth with ChatGPT in order to troubleshoot and eventually fix a lot of the bugs I was experiencing during the setup process of antlr.

One of the issues I found was how my computer was configured. Since my computer has multiple storage devices inside it with different versions of Java installed on a couple of them, this caused conflicts with the setup as my computer defaulted into using Java 8 instead of Java 21. This meant that when I executed the commands to run antlr normally,

export CLASSPATH=".:~/lib/antlr-4.13.2-complete.jar:$CLASSPATH"

alias antlr4='java -jar ~/lib/antlr-4.13.2-complete.jar'

alias grun='java org.antlr.v4.gui.TestRig'

export would not work as intended. So, I had to modify the setup process, so it used a “.bat” file instead.

antlr4.bat:

@echo off

java -jar "J:\Documents\Programming 3\Coursework\Language Design Coursework\Antlr\antlr-4.13.2-complete.jar" %\*

grun.bat:

@echo off

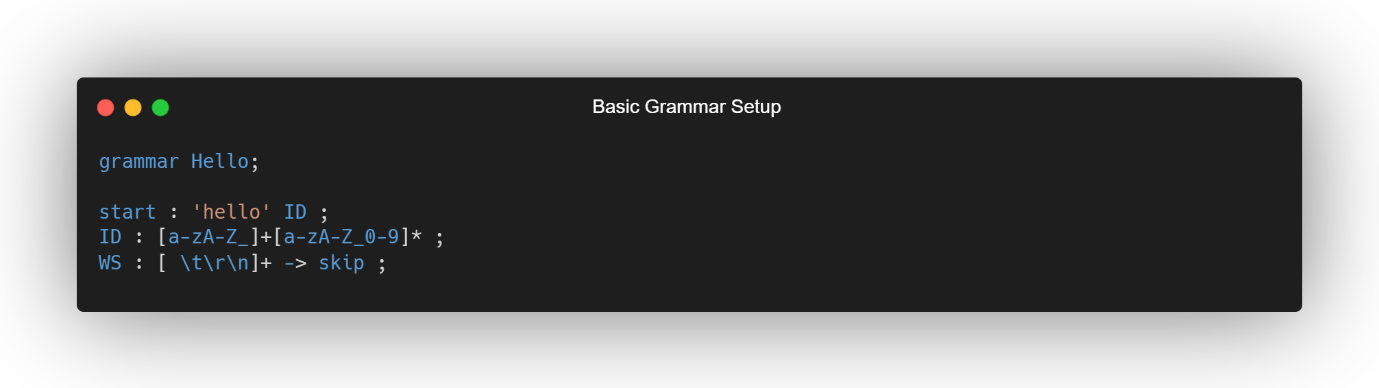
java -cp "J:\Documents\Programming 3\Coursework\Language Design Coursework\Antlr\antlr-4.13.2-complete.jar;." org.antlr.v4.gui.TestRig %\*

Once I made the bat files, I stored them inside the J drive and made a root folder called “bin”. I put the bat files inside and copied the folder path, I then opened the system environment variables and added “J:\bin” as a variable to the PATH variable. Once I did, I went to the command line and ran “antlr4 -Dlanguage=Python3 <file name>.g4” to generate the Antlr python files for the project.

To run grun, I had to modify the javac command from “javac MyLang\*.java” to “javac -cp ".;J:\Documents\Programming 3\Coursework\Language Design Coursework\Antlr\antlr-4.13.2-complete.jar" MyLang\*.java”. The reason for this is because the default javac command defaults to a libs folder in the C Drive that doesn’t exist. This new command calls the java file directly and run all the java files that contain the language I am programming.

### Antlr Learning

While completing the workshops, I learned how to script simple languages together. Such as setting up a Hello.g4 script with the basic grammar for a language:



Through reading the lecture notes and workshops, I was able to create a basic grammar language called Hello. This language would start with hello, followed by an ID. The ID had a regex which allowed for any character (uppercase or lowercase) or number, [a-zA-Z\_]+[a-zA-Z\_0-9]\* (a-z allows lowercase, A-Z allows uppercase, \_ allows for character to contain an underscore, + means it must start with at least once, 0-9 allows for numbers, \* means a character can appear any number of times), which allowed the user to input any ID they wanted. White spaces were skipped and were detected using regex for white spaces such as \t (for tabs), \r (for line separations), and \n (for line breaks).

Once the basic grammar file was created, the antlr4 command was ran (antlr4 -Dlanguage=Python3 <file name>.g4) to generate the files needed for the project. Once they were generated, I created a main.py file which imported sys, antlr4, and the generated Parser, Lexer, and Listener into the file. After that I created a main function which followed the workshop steps.

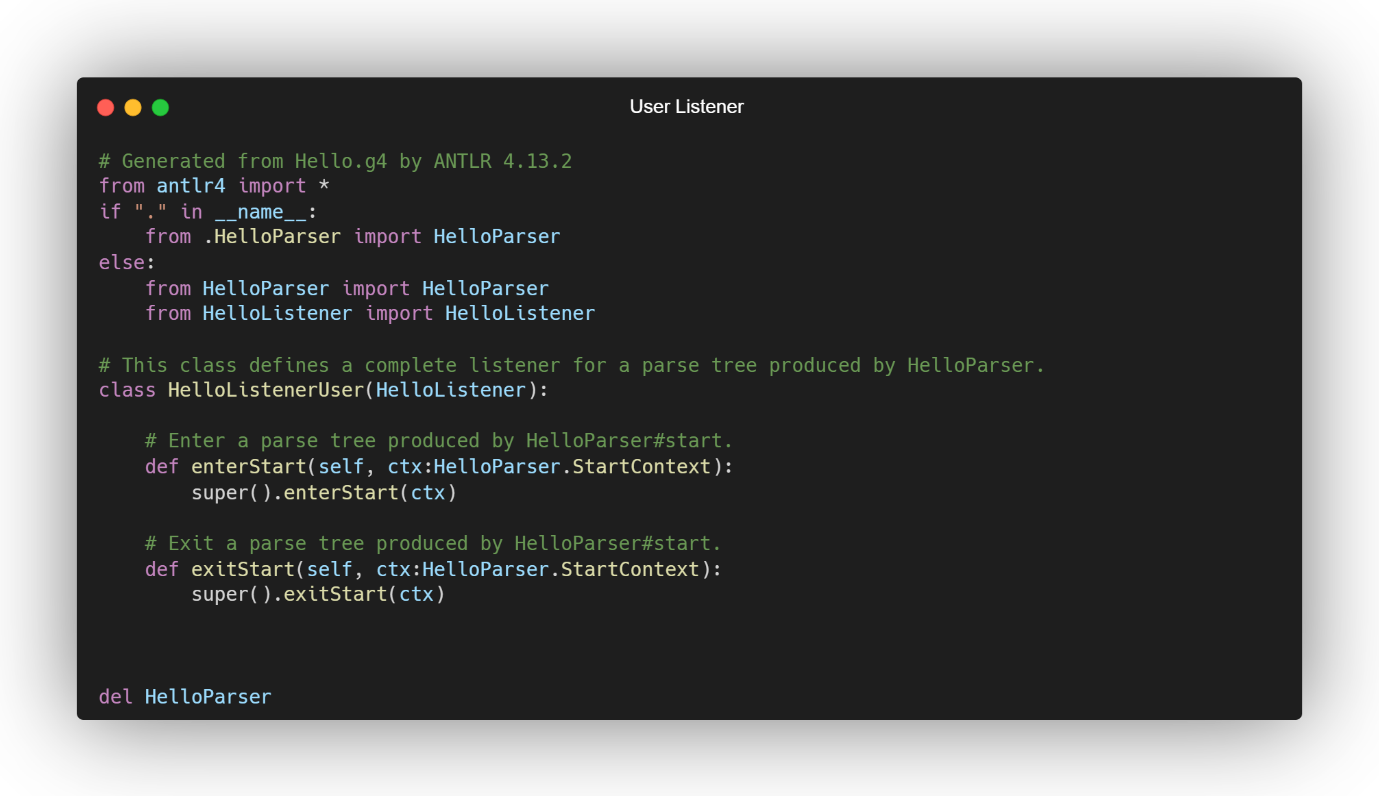
A screen shot of a computer screen

AI-generated content may be incorrect.The main function acted as the entry point of the script. It takes a list of command-line arguments (argv) as an input. If only one argument is provided (the script name itself), the input is read from the standard input (StdinStream). Otherwise, the second argument (argv[1]) is treated as the name of the file, and its contents are read using FileStream. This flexibility allows the script to handle both interactive input and file-based input.

The input stream is passed to a lexer (HelloLexer), which tokenises the input based on the rules defined in the lexer grammar. The tokens are then fed into a parser (HelloParser) via a CommonTokenStream. The parser processes the tokens according to its grammar rules and produces a parse tree, which represents the hierarchical structure of the input.

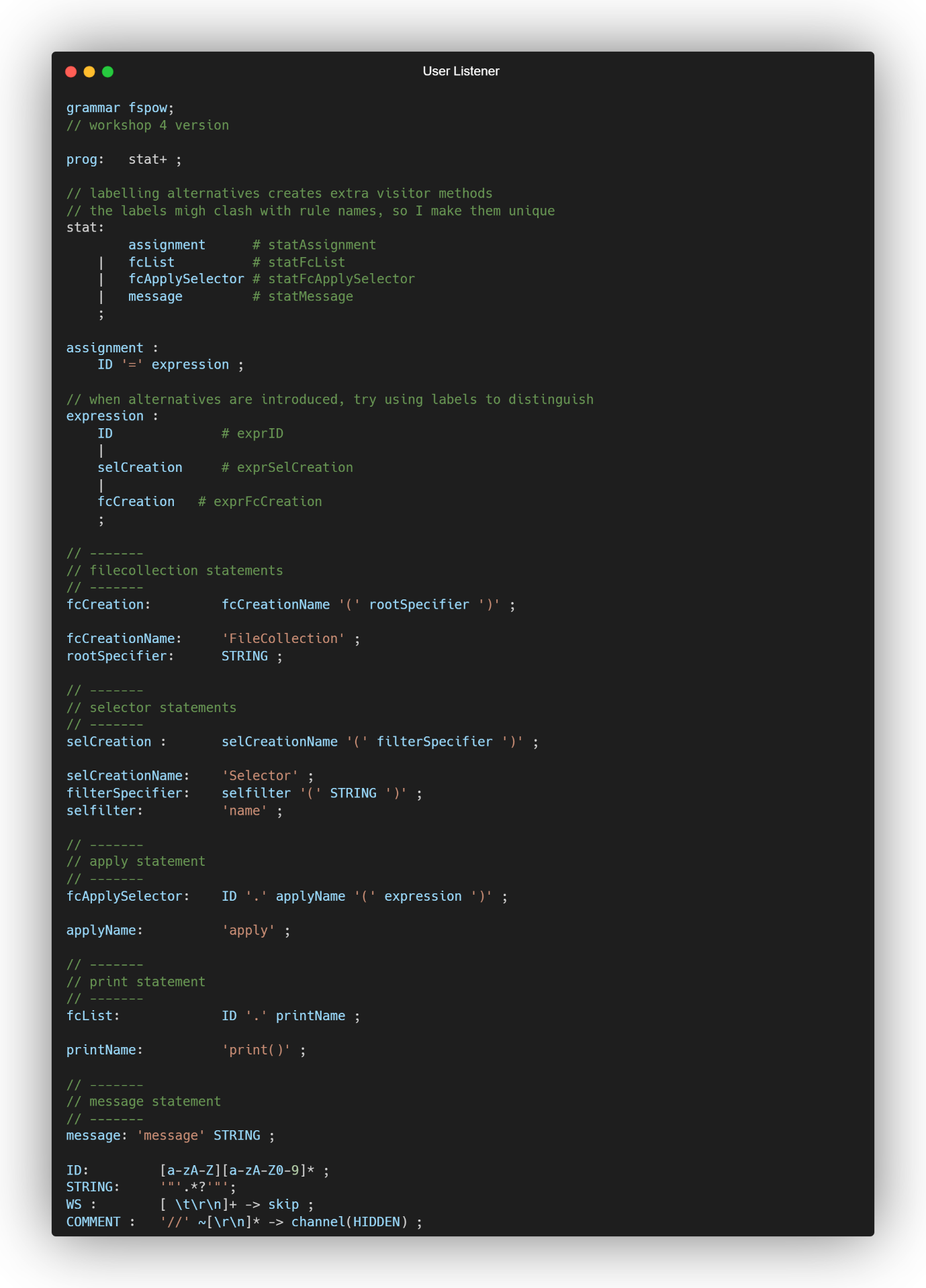
The parse tree is printed to the console using Trees.toStringTree, which provides a textual representation of the tree. This is useful for debugging or understanding how the input was parsed.

Next, the script creates an instance of HelloListenerUser, which is a modifed version of of HelloListener as during the workshop, it was requested to make a modified version of the listen to change the output and not have it be overwritten when running the antlr4 command. This listener contains methods that are triggered as the parse tree is traversed. The ParseTreeWalker is used to walk through the parse tree and invoke the appropriate methods in the listener. This step allows the script to perform additional processing or extract information from the parse tree.



Finally, the “if \_\_name\_\_ == ‘\_\_main\_\_’:” block ensures that the main function is executed only when the script is run directly, not when it is imported as a module. This is a common way when it comes to Python idiom for defining script entry points.

Similar with the following workshops, a basic grammar language was developed but instead used a custom Tree parser called TreeUser which took the trees and formatted the output.

Workshop 4 taught how to use fspow and provided new files like FileCollection, FileCollectionIterator, fspowVistor, Selector, and FSObject. This showed files can be handled. A .txt file was created as an input which provided lines for the fspow to go through. The fspow.g4 grammar file also showed how a more complex setup could be accomplished by showing how an expression can be defined, and how statements are created. 



I chose to use grun to display the output, so I ran the command “grun fspow prog -gui mincommands.txt” to display the tree output. Modification to the code was required as certain parts were not implemented yet. But this was simple to implement as long as I followed the lectures.

### Overall Experience

I liked challenging myself when making a language. It allowed me to play around with how different expressions can affect the output and how strict it is to ensure that proper spelling and syntax are used when creating even a simple grammar language.

Reflecting on the process, I believe that certain parts could have been improved such as how much time I dedicated towards learning antlr and lectures. But due to the pressure of other modules and looming deadlines, the time I wanted to spend on learning antlr went towards other projects. If I were to start over, I believe I would done a better job at managing my time between different projects and ensuring that the time I spent in workshops was spent doing the workshop and not another modules deadline.

Overall, I enjoyed learning Antlr.