Rapport du projet

Par Gabriel Arsenault

Table des matières

[Préparation de l’expérimentation 1](#__RefHeading___Toc170_325362768)

[1. Sujet 1](#__RefHeading___Toc172_325362768)

[2. Objectif 1](#__RefHeading___Toc174_325362768)

[3. Critères d’évaluation 2](#__RefHeading___Toc176_325362768)

[Réalisation de l’expérimentation 3](#__RefHeading___Toc178_325362768)

[4. Expérimentation 3](#__RefHeading___Toc180_325362768)

[1.1. Exigences techniques 3](#__RefHeading___Toc182_325362768)

[1.2. Déroulement 3](#__RefHeading___Toc184_325362768)

[1.3. Résultats 3](#__RefHeading___Toc186_325362768)

[1.1.1 Tutoriel ANTLR 3](#__RefHeading___Toc1900_1301357424)

[1.1.2 Le fichier de grammaire (/src/antlr) 4](#__RefHeading___Toc1902_1301357424)

[Conclusion 5](#__RefHeading___Toc1904_1301357424)

# Préparation de l’expérimentation

## Sujet

J’ai fait mes recherches sur des projets qui requièrent de la connaissance avancé sur la programmation. Un de ces projets est la conception de ton propre langage de programmation. Créer ton langage de programmation est utile pour répondre à tes besoins, surtout quand tu connais les forces et les faiblesses de ton langage mieux que tout le monde. C’est pourquoi je voudrais essayer de créer mon propre langage de programmation.

## Objectif

Mon projet consiste à créer mon propre langage de programmation qui fonctionnera comme les scripts de Python ou les exécutables de C++, dépendamment des efforts nécessaire pour créer un interpréteur versus créer un compiler.

Pour commencer, je vais créer une librairie standard qui contiendra les fonctions de base pour mon langage, puis je vais coder une application qui présentera ces différentes fonctions.

Le contenu de cette ou ces applications dépend entièrement sur ce que je réussi à implémenter dans ma librairie standard. Une librairie standard de n’importe quel langage de programmation contient de nombreuses composantes sur différents sujets. Voici quelques composantes que je planifie intégrer, en ordre de priorité:

1. Fonctions mathématiques de base (abs, sin, cos, tan, racine, min, max, exposent, tronquer/arrondir, etc.)
2. Fonctions sur les manipulations de strings (longueur du string, remplacer A par B, substring, pad left/right, trim, etc.)
3. Fonctions sur les arrays (array, liste, dictionnaire, etc.)
4. Fonctions sur les inputs (lire la ligne de texte, lire une touche du clavier)
5. Fonctions sur l’interaction du terminal (print, changer la couleur du texte et de l’arrière-plan, cls/clear, etc.)
6. Fonctions audio
7. Fonctions de base de données embarquée dans l’application (SQLite, de la même façon que la librairie SQLite de Python)

Ce n’est pas garanti que je vais toutes les intégrer, mais une bonne portion devrait suffire.

## Critères d’évaluation

1. Le langage doit contenir un shell pour exécuter des commandes en temps réel.
2. Le langage doit supporter les arithmétiques, incluant du bit shifting.
3. On doit pouvoir déclarer des nouvelles variables et assigner des variables existantes.
4. Le langage doit supporter les if statements avec des comparaisons (==, ≥, <, !=, etc.) et des opérateurs logiques (&&, || et !).
5. Le langage doit supporter les loops for et/ou les loops while, incluant les mot-clé break et continue.
6. Le langage doit supporter la déclaration et l’appel des fonctions, incluant des paramètres.
7. Le langage doit supporter la création de classes qui incluent des variables et des fonctions.
8. Le langage doit pouvoir importer du code de d’autres fichiers du même langage, donc des modules.

# Réalisation de l’expérimentation

## Expérimentation

## Exigences techniques

1. ANTLR (ANother Tool for Language Recognition)
   1. Générateur de parser qui permet de traduire le code en binaires.
2. Python 3.10
   1. Langage de programmation qui sera ma base pour créer la logique de mon langage

## Déroulement

Pour commencer, je dois comprendre le processus de la création d’un langage de programmation. Je vais suivre des tutoriels et lire la documentation d’ANTLR afin de pouvoir définir les syntaxes de mon langage, puis de les rendre fonctionnelle.

## Résultats

### Tutoriel ANTLR

<https://tomassetti.me/learning-build-languages/>

<https://tomassetti.me/building-a-simple-expression-language/>

Le tutoriel de tomassetti sur la création d’un langage m’a donné la direction que j’avais besoin pour trouver les bons outils. Cependant, afin de ne pas copier le travail de tomassetti, j’ai décidé de ne pas utiliser Gradle, Kotlin ou son fichier de grammaire pour mon projet.

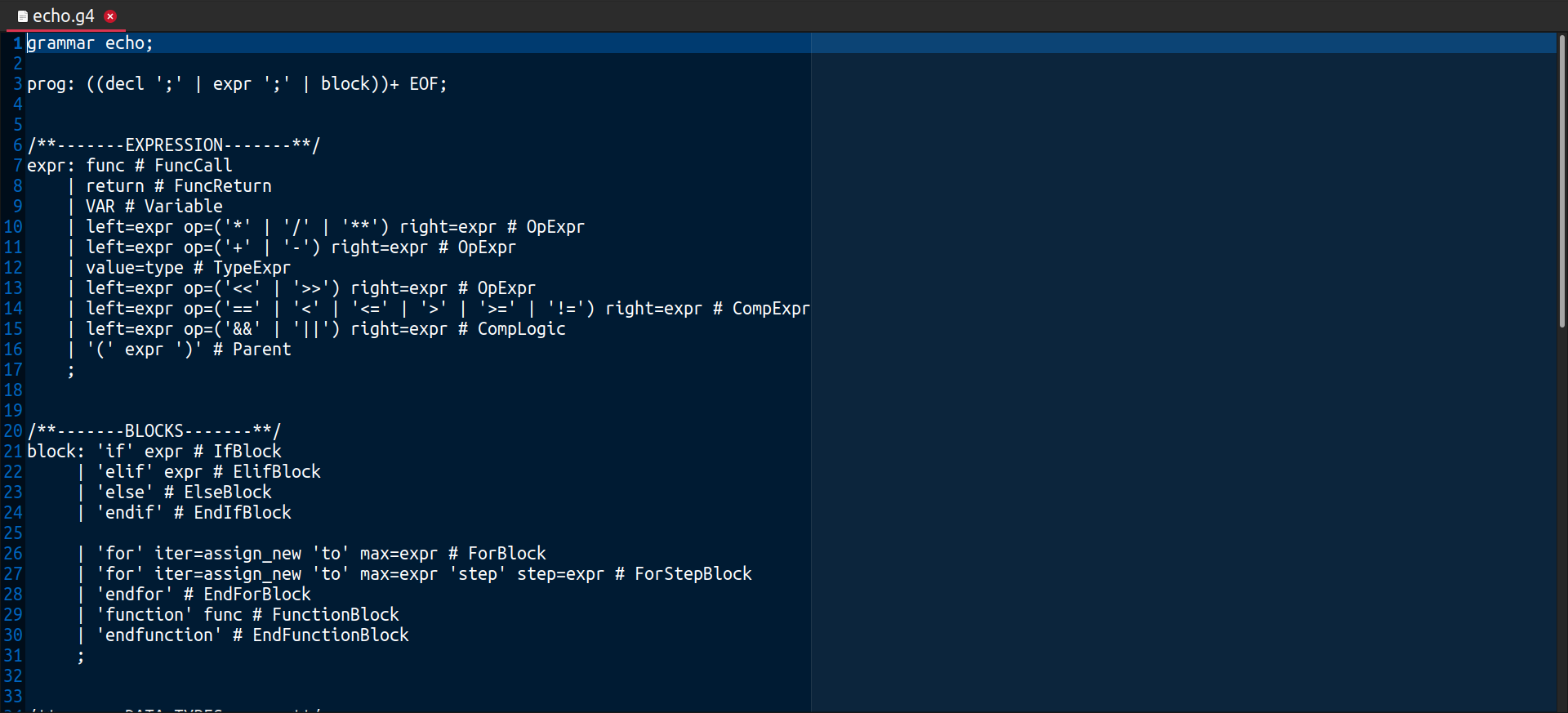
<https://www.youtube.com/playlist?list=PL5dxAmCmjv_4FGYtGzcvBeoS-BobRTJLq>

J’ai suivi un tutoriel sur l’utilisation d’ANTLR. J’ai appris à définir les syntaxes dans le fichier de grammaire et à générer le parser grâce à cette collection de vidéos.

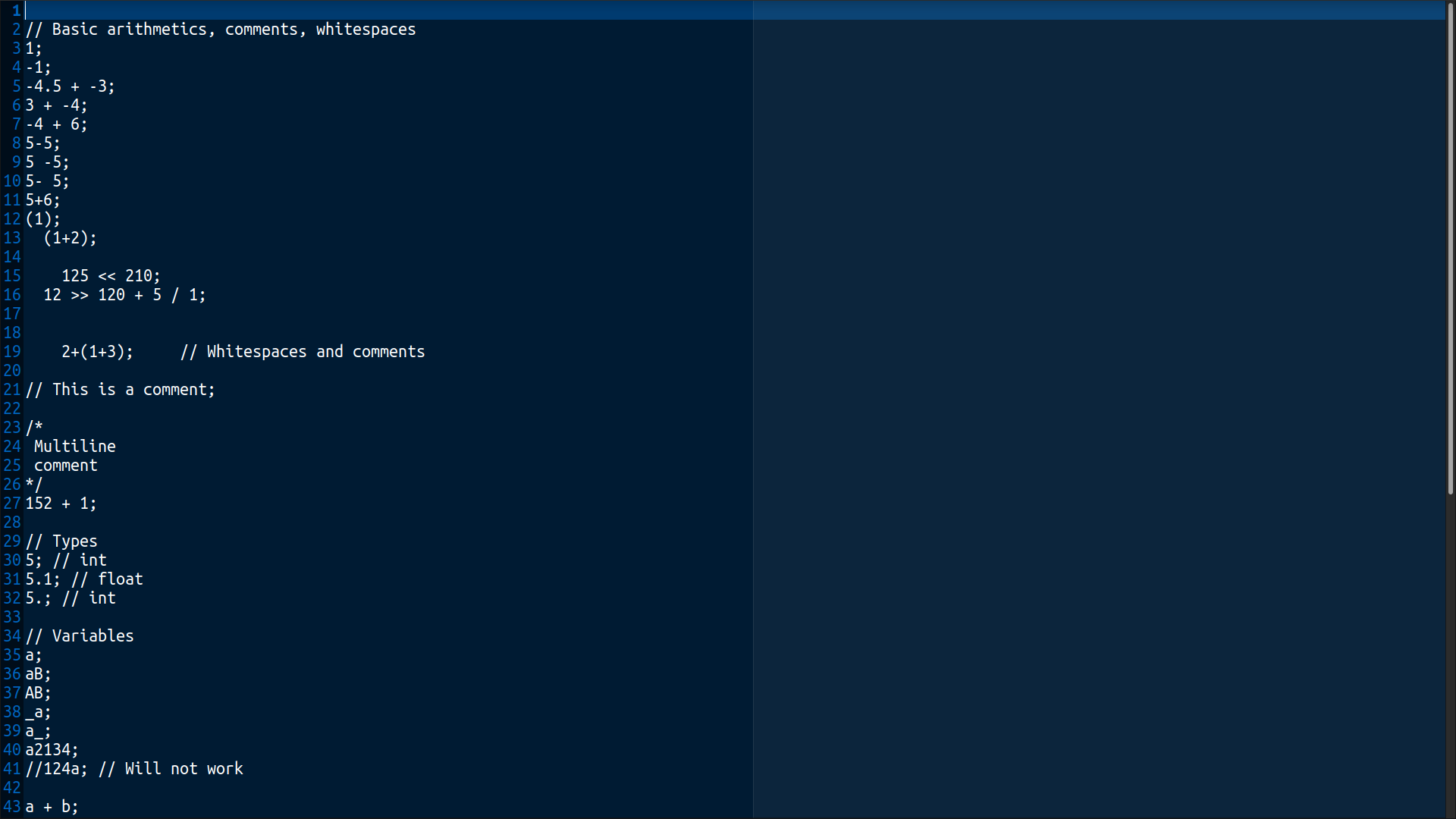
<https://github.com/antlr/antlr4/blob/master/doc/index.md>

Je me suis aussi référé à la documentation officielle d’ANTLR. Par contre, la documentation pour la version Python du parser était limité, donc j’ai dû régler les bogues de mon fichier de grammaire par moi-même la majorité du temps.

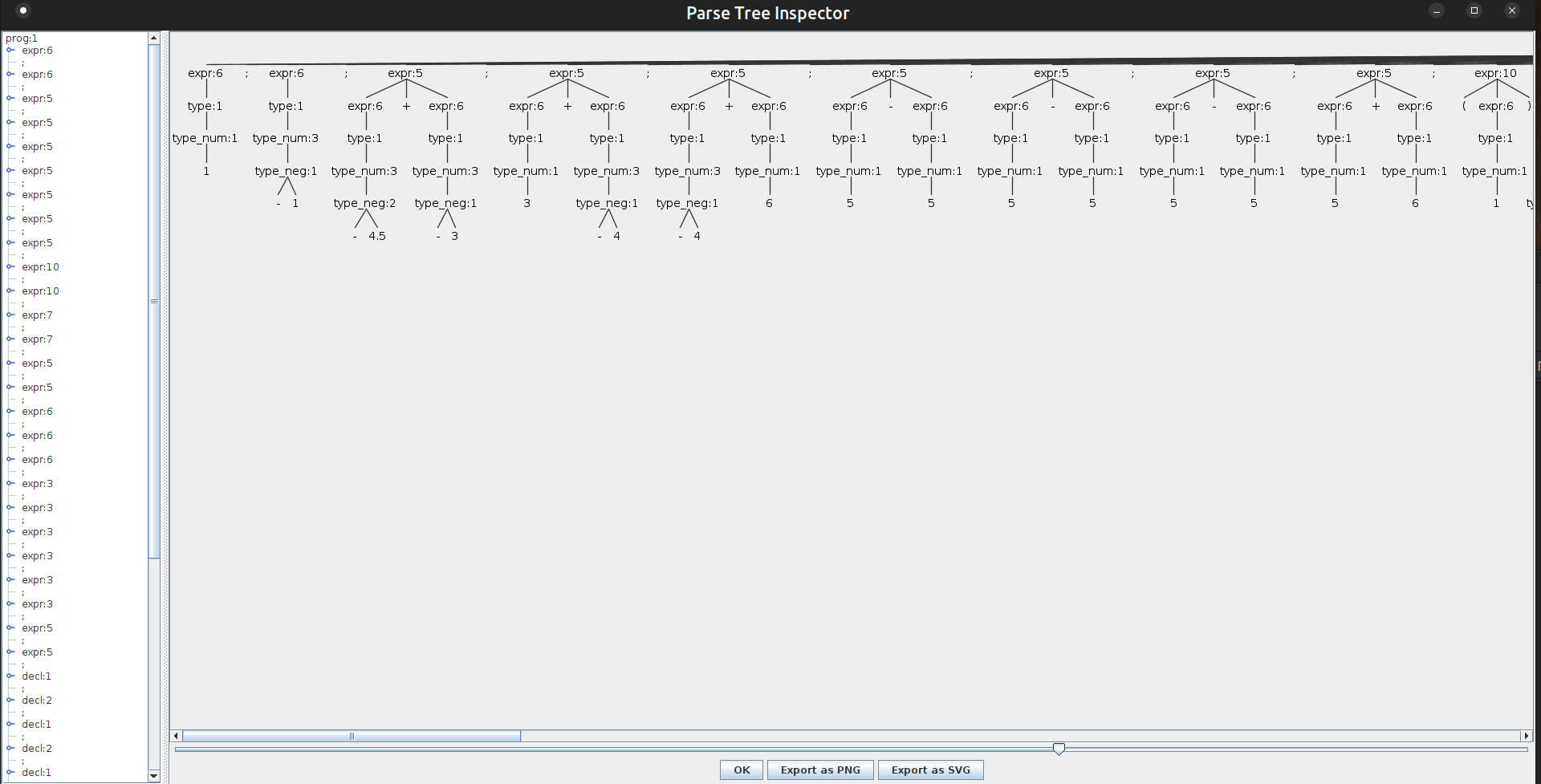
### Le fichier de grammaire (/src/antlr)



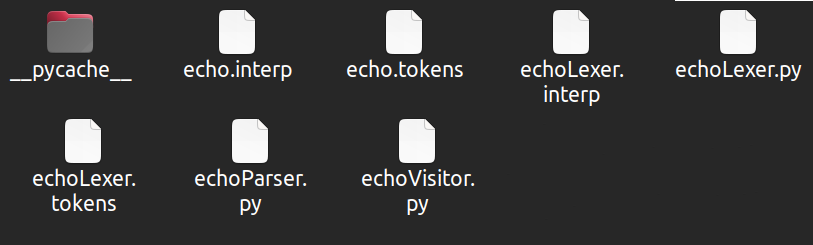
Afin de générer le parser pour mon langage, je dois créer un fichier de grammaire (.g4) qui définit ses syntaxes. Je définis les syntaxes pour les types de données (int, float, bool, string), une expression (*1 + 5, a \* 2*, *(124 – 12) \* (a – abs(-5)*, etc.), un bloc if (if 1 == True, endif), un bloc for (for i = 0 to 10), une déclaration/assignation de variable, les commentaires, etc.



J’ai créé des fichiers tests pour tester les syntaxes définit par Echo.g4. On peut tester chaque fichier en exécutant la commande suivante (Nécessite ANTLR) : **antlr4-parse echo.g4 prog -gui tests/test1.ec**



Et nous avons ce résultat. les syntaxes de test 1 et test 2 fonctionnent correctement depuis la dernière itération du fichier de grammaire (Il se peut qu’un des tests retourne « token recognition error at ‘.’ ». C’est normal puisque je voulais supporter les nombres à virgule n’ayant pas de chiffre après la virgule, donc « 5. ». Malgré qu’il dit que c’est une erreur, le parser continue de fonctionner comme prévu).



Maintenant qu’on a complété les syntaxes (on peut toujours en ajouter plus tard), il faut générer le parser avec la commande suivante : **antlr4 echo.g4 -Dlanguage=Python3 -visitor -no-listener -o ../bin/**

Ceci va générer les fichiers Python ci-dessus dans le dossier « bin ».

# Conclusion