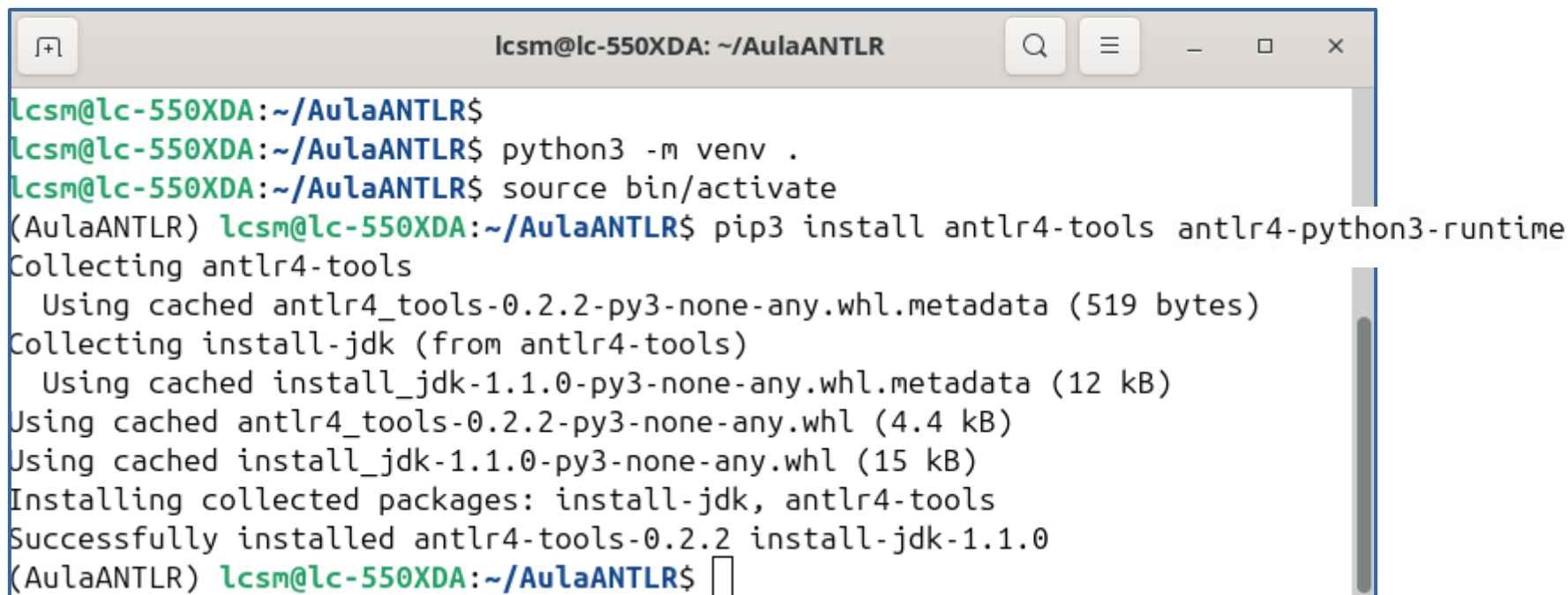


Criando um Avaliador de Expressões  
usando ANTLR.

# I – Instalando ANTLR

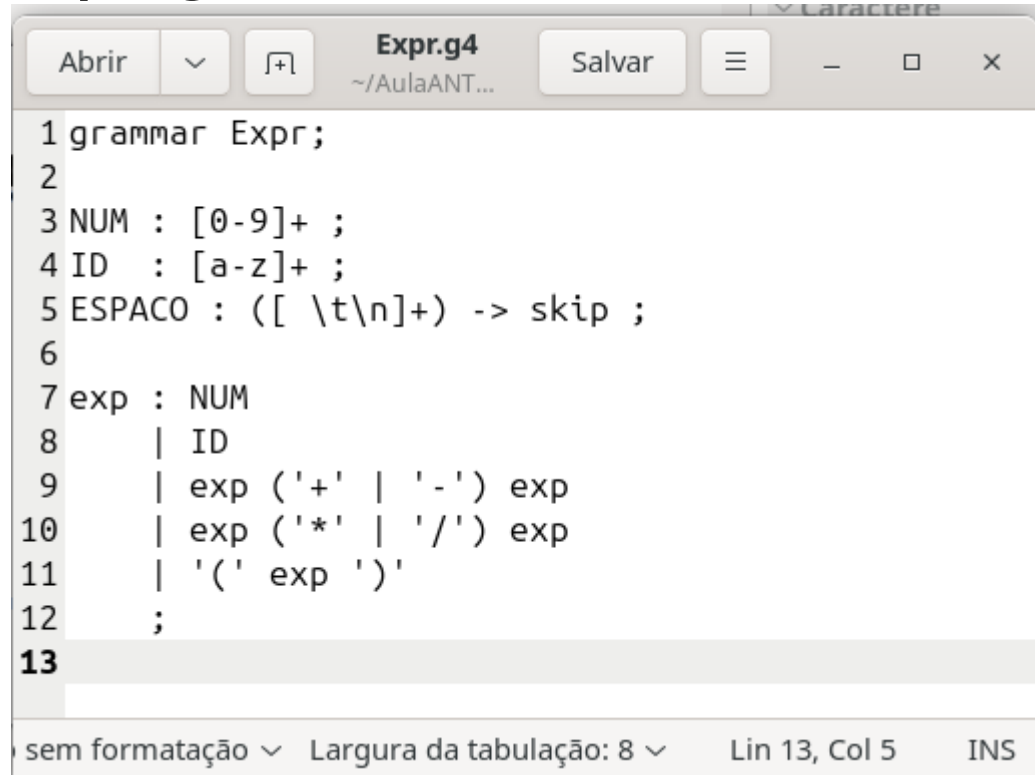
- Usando python e venv



```
lscsm@lc-550XDA: ~/AulaANTLR
lscsm@lc-550XDA:~/AulaANTLR$ python3 -m venv .
lscsm@lc-550XDA:~/AulaANTLR$ source bin/activate
(AulaANTLR) lscsm@lc-550XDA:~/AulaANTLR$ pip3 install antlr4-tools antlr4-python3-runtime
Collecting antlr4-tools
  Using cached antlr4_tools-0.2.2-py3-none-any.whl.metadata (519 bytes)
Collecting install-jdk (from antlr4-tools)
  Using cached install_jdk-1.1.0-py3-none-any.whl.metadata (12 kB)
Using cached antlr4_tools-0.2.2-py3-none-any.whl (4.4 kB)
Using cached install_jdk-1.1.0-py3-none-any.whl (15 kB)
Installing collected packages: install-jdk, antlr4-tools
Successfully installed antlr4-tools-0.2.2 install-jdk-1.1.0
(AulaANTLR) lscsm@lc-550XDA:~/AulaANTLR$
```

# II – Criando Descrição da Linguagem

- Arquivo Expr.g



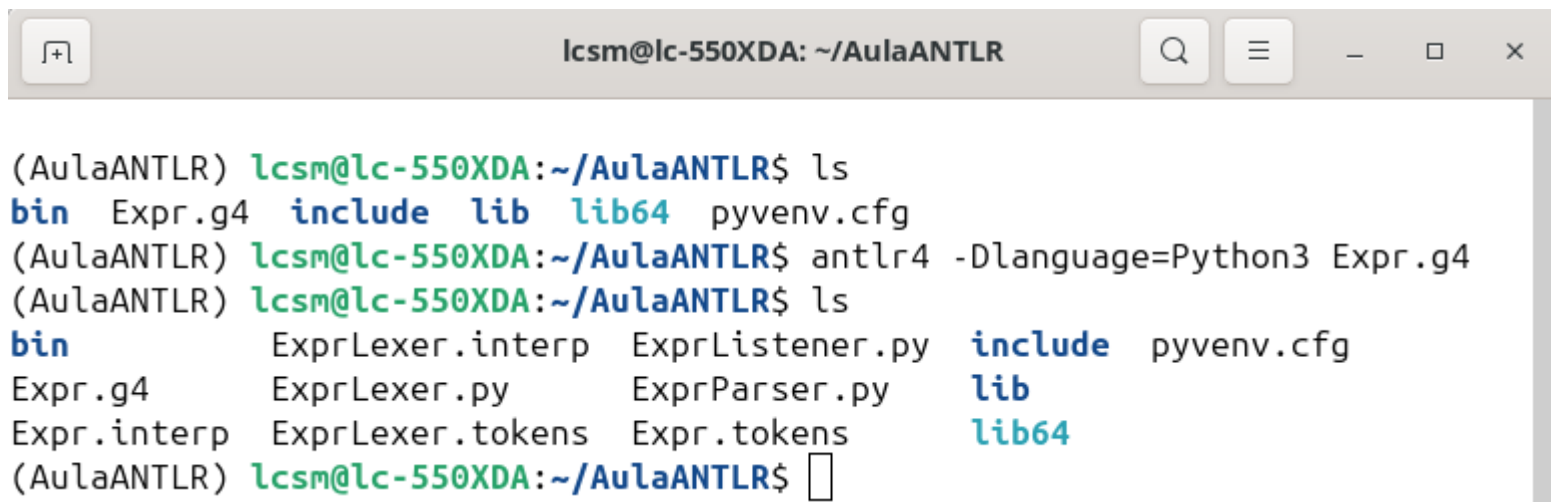
The image shows a screenshot of a text editor window titled "Expr.g4". The window has a menu bar with "Abrir", "Salvar", and a hamburger menu icon. Below the menu bar is a toolbar with icons for opening, saving, and zooming. The main text area contains a grammar definition for "Expr". The grammar is written in a monospaced font and is line-numbered from 1 to 13. The lines are as follows:

```
1 grammar Expr;  
2  
3 NUM : [0-9]+ ;  
4 ID  : [a-z]+ ;  
5 ESPACO : ([ \t\n]+) -> skip ;  
6  
7 exp : NUM  
8     | ID  
9     | exp '+' | '-' exp  
10    | exp '*' | '/' exp  
11    | '(' exp ')'  
12    ;  
13
```

At the bottom of the window, there is a status bar that reads "sem formatação", "Largura da tabulação: 8", "Lin 13, Col 5", and "INS".

# III – Executar ANTLR

- Comando:  
antlr4 -Dlanguage=Python3 Expr.g4

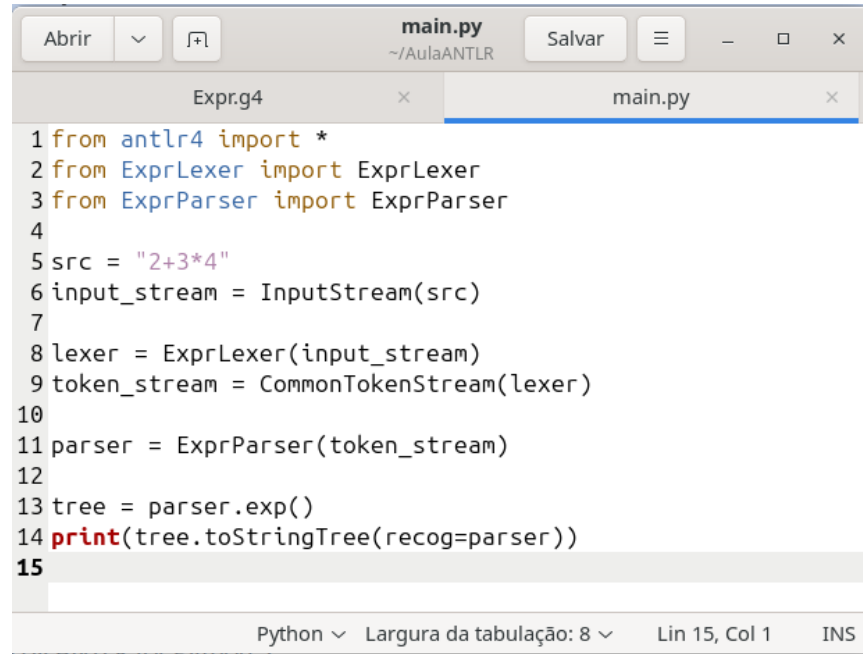


```
lscm@lc-550XDA: ~/AulaANTLR
(AulaANTLR) lscm@lc-550XDA:~/AulaANTLR$ ls
bin  Expr.g4  include  lib  lib64  pyenv.cfg
(AulaANTLR) lscm@lc-550XDA:~/AulaANTLR$ antlr4 -Dlanguage=Python3 Expr.g4
(AulaANTLR) lscm@lc-550XDA:~/AulaANTLR$ ls
bin          ExprLexer.interp  ExprListener.py  include  pyenv.cfg
Expr.g4      ExprLexer.py      ExprParser.py    lib
Expr.interp  ExprLexer.tokens  Expr.tokens      lib64
(AulaANTLR) lscm@lc-550XDA:~/AulaANTLR$
```

# IV – Programa Principal

- Responsável por criar o pipeline do compilador

src --> A. Lex --> A. Sint. --> ....



```
1 from antlr4 import *
2 from ExprLexer import ExprLexer
3 from ExprParser import ExprParser
4
5 src = "2+3*4"
6 input_stream = InputStream(src)
7
8 lexer = ExprLexer(input_stream)
9 token_stream = CommonTokenStream(lexer)
10
11 parser = ExprParser(token_stream)
12
13 tree = parser.exp()
14 print(tree.toStringTree(recog=parser))
15
```

# Árvores Sintáticas

## Principais Métodos

- Classe: `ParserRuleContext`
  - Função: representa um nó interno da árvore (resultado de uma regra da gramática).
- Métodos úteis:
  - `getChildNum()` -> `int` — Retorna o número de filhos.
  - `getChild(i: int)` -> `ParseTree` — pega o filho de índice `i`.
  - `getChildren()` -> `Iterator[ParseTree]` — itera sobre os filhos.
  - `getText()` -> `str` — retorna o texto correspondente à subárvore.
  - `getRuleIndex()` -> `int` — índice da regra da gramática (pode mapear para o nome).
  - `parentCtx` — referência para o pai (se quiser subir na árvore).

# Árvores Sintáticas

## Principais Métodos

- TerminalNodeImpl
  - Função: representa um token do lexer (nó folha).
  - Métodos úteis:
    - `getSymbol()` -> Token — retorna o token associado.
    - `getText()` -> str — retorna o texto do token.
    - `getParent()` -> RuleContext — retorna o nó pai.

.

# Exemplo

- Código Análise de Árvore:

```
tr = parser.expr()
```

```
print("Número de filhos:", tr.getChildCount())
```

```
for i in range(tr.getChildCount()):  
    filho = tr.getChild(i)  
    print(i, filho.getText())
```



# Classe Árvore

- Cada não terminal gera uma classe que representa seus Node.
  - Subclasse de ParseRuleContext

```
exp : NUM  
    | ID  
    | exp ('+' | '-') exp  
    | exp ('*' | '/') exp  
    | '(' exp ')'  
    ;
```

```
class ExpContext(ParserRuleContext):  
  
    def NUM(self):  
    def ID(self):  
    def exp(self, i:int=None):
```

# Classes Árvores Detalhadas

- É possível fazer o ANTLR gerar classes mais detalhadas:
  - Um tipo próprio para cada possível nó
  - Campos para recuperar sub

# Classes Árvores Detalhadas

```
exp : NUM #Const
    | ID #Var
    | e=exp('+' | '-' )d=exp #Soma
    | e=exp('*' | '/' )d=exp #Mult
    | '(' e=exp ')' #Group
    ;
```

```
class SomaContext(ExpContext):
    def __init__(self):
        self.e = None # ExpContext
        self.d = None # ExpContext
```