Projet TP - Module Compilation

# Mini-Compilateur Python avec Foreach

**Nom :** Hocina

**Prenom:**Sarah

**Groupe**:A4

**Date :** 08/12/2025

**Année Universitaire**:2025/2026

# 1. Introduction

L'objectif de ce travail est de développer un **analyseur lexical** (ou lexer) en Java, capable de lire un fichier source Python et de produire une liste de **tokens.**Un token est une unité lexicale qui représente un élément significatif du langage : mot-clé, identifiant, opérateur, nombre, chaîne, etc.

## 2. Fonctionnalités du Programme

**Lecture du fichier source:Le programme lit entièrement le fichier test.py à laide d'un BufferedReader.**

**Découpage du texte en tokens:Chaque caractère est analysé selon une machine à états finis comportant les états :**

- IDENTIFIANT
- NOMBRE
- CHAINE
- CARACTERE
- OPERATEUR
- COMMENTAIRE_LIGNE
- COMMENTAIRE_BLOC

### Classification des tokens

Les catégories reconnues sont :

- MOT_CLE : mots clés en Python (ex: if, for, class, …)
- IDENTIFICATEUR : variables ou noms d'identifiants
- NOMBRE : entiers, flottants, notations scientifique et hexadécimale
- OPERATEUR : +, -, ==, //, etc.
- SEPARATEUR : parenthèses, accolades, etc.
- CHAINE : chaîne délimitée par (" ")
- CARACTERE : caractère délimité par (‘’)
- ERREUR LEXICALE : tout élément non reconnu

L'analyseur utilise une machine à états finis (Finite State Machine) pour gérer les différents types de lexèmes.

### 3.1. État DEBUT

Décide du type d'élément rencontré :

lettre ou _ → IDENTIFIANT

chiffre → NOMBRE

guillemet → CHAINE

apostrophe → CARACTERE

→ commentaire ligne

/* */ → commentaire bloc

opérateur ou séparateur

sinon → erreur lexicale

### 3.2. État IDENTIFIANT

Accepte lettres, chiffres, _.

À la fin, on vérifie s'il s'agit de :

un mot clé

un opérateur littéral

un identificateur valide

une erreur

### 3.3. État NOMBRE

Accepte :

chiffres

point .

notation exponentielle e E

format hexadécimal x

format binaire b

### 3.4. États CHAINE et CARACTERE

Identifient les éléments textuels entre guillemets.

### 3.5. États des Commentaires

commentaire ligne : s'arrête au retour à la ligne

commentaire bloc : s'arrête à */

**Détection des Erreurs Lexicales**

Les erreurs détectées sont :

identifiant commençant par un caractère interdit

caractère non reconnu

opérateur ou mot inconnu

fin de fichier inattendue dans une chaîne ou caractère

# Résultat de l'Analyse

À la fin de l'exécution, chaque token extrait est affiché dans ce format :

valeur : TYPE ou ERREUR LEXICALE : valeur

Ce qui permet :

la détection d'anomalies

une préparation à l'analyse syntaxique

Analyseur Syntaxique:

Règle de production:

Programme→Instruction Programme|ε

Instruction→Affectation|Condition|Boucle|Affichage|Bloc

Affectation→IDENTIFICATEUR = Expression

Condition→if Expression : Bloc ElsePartie

ElsePartie→else : Bloc|ε

Boucle→BoucleFor|BoucleWhile|BoucleForeach

BoucleFor→for IDENTIFICATEUR in range ( Arguments ) : Bloc

BoucleWhile→while Expression : Bloc

BoucleForeach→foreach IDENTIFICATEUR in Expression : Bloc

Bloc→{ Programme }

Affichage→print ( Arguments )

Arguments→Expression Arguments'|ε

Arguments'→, Expression Arguments'|ε

Expression→Terme Expression'

Expression'→+ Terme Expression'|- Terme Expression'|ε

Terme→Facteur Terme'

Terme'→* Facteur Terme'|/ Facteur Terme'|ε

Facteur→NOMBRE|IDENTIFICATEUR|CHAINE|CARACTERE|( Expression )|Appel

Appel→IDENTIFICATEUR ( Arguments )

Symbole termineau:

IDENTIFICATEUR→Variables commençant par $ (ex: $x, $total)

NOMBRE→Entiers, décimaux, hex, binaire

CHAINE→Chaînes entre " "

CARACTERE→Caractères entre ' '

MOT_CLE→if, else, for, while, foreach, in, range, print

OPERATEUR→+, -, *, /, =, etc.

SEPARATEUR→( ) { } [ ] , : ;

Exemples de codes:

Foreach simple

```
foreach $item in $liste :
{
   print($item)
}
```

### Foreach avec condition

```
foreach $num in $nombres :
{
   if $num > 10 :
   {
      print($num)
   }
}
```

### Boucle for classique

```
for $i in range(0, 10) :
{
   $total = $total + $i
}
```

## Ensembles des débuts:

### Programme

FIRST = {$, if, for, while, foreach, print, {, ε}

FOLLOW = {$, }}

### Instruction

FIRST = {$, if, for, while, foreach, print, {}

FOLLOW = {$, if, for, while, foreach, print, {, }}

Expression

FIRST = {NOMBRE, $, ", ', (}

FOLLOW = {), ,, :, +, -, *, /}

Boucle

FIRST = {for, while, foreach}

FOLLOW = {$, if, for, while, foreach, print, {, }}

Grammaire ↔ Code

| Règle de Grammaire | Méthode dans le Code |
|---|---|
| Programme → Instruction Programme \| ε | programme() |
| Instruction → Affectation \| Condition \| Boucle \| Affichage \| Bloc | instruction() |
| Affectation → IDENTIFICATEUR = Expression | affectation() |
| Condition → if Expression : Bloc ElsePartie | condition() |
| Boucle → BoucleFor \| BoucleWhile \| BoucleForeach | boucle() |
| BoucleFor → for IDENTIFICATEUR in range ( Arguments ) : Bloc | boucleFor() |
| BoucleWhile → while Expression : Bloc | boucleWhile() |
| BoucleForeach → foreach IDENTIFICATEUR in Expression : Bloc | boucleForeach() |
| Bloc → { Programme } | bloc() |
| Expression → Terme Expression' | expression() |
| Expression' → + Terme Expression' \| - Terme Expression' \| ε | expressionPrime() |
| Terme → Facteur Terme' | terme() |
| Terme' → * Facteur Terme' \| / Facteur Terme' \| ε | termePrime() |
| Facteur → NOMBRE \| IDENTIFICATEUR \| CHAINE \| ... | facteur() |

## Exemple de Fonctionnement

**Code source :**

python

foreach item in liste :{  print(item)}

**Déroulement de l'analyse :**

programme() → instruction() → boucle() → boucleForeach() → match("foreach") → match("IDENTIFICATEUR") → "item" → match("in") → expression() → "liste" → match(":") → bloc() → match("{") → programme() → instruction() → affichage() → match("print") → match("(") → arguments() → match(")") → match("}")

---

# 5. Gestion des Erreurs

## 5.1 Types d'Erreurs Détectées

**Erreurs Lexicales :**

- Caractères invalides : @, $, etc.
- Chaînes non fermées
- Nombres mal formés

**Erreurs Syntaxiques :**

- Mots-clés manquants : foreach item liste (manque in)
- Séparateurs manquants : if x > 10 { ... } (manque :)
- Parenthèses non appariées

## 5.2 Stratégie de Récupération

Le compilateur **ne s'arrête pas** à la première erreur :

1. **Détecte l'erreur** et l'affiche
2. **Cherche un point de synchronisation** (;, }, mot-clé)
3. **Continue l'analyse** pour détecter d'autres erreurs

---

# 7. Cas de Test

## Test 1 : Code Correct

**Fichier :** test1_correct.py

python

```
foreach item in liste :{    if item > 10 :    {        print(item)    }}
```

**Résultat attendu :** ✓ Analyse réussie

---

## Test 2 : Erreur Lexicale

**Fichier :** test2_erreur_lexicale.py

python

```
foreach item in liste :{    print(@item)}
```

**Résultat attendu :**

```
ERREUR LEXICALE : @
```

---

## Test 3 : Erreur Syntaxique

**Fichier :** test3_erreur_syntaxique.py

python

```
foreach item liste :{    print(item)}
```

**Résultat attendu :**

```
ERREUR SYNTAXIQUE [Position 2] : Attendu mot-cle: in, Trouve: liste
```

---

## Test 4 : Plusieurs Erreurs

**Fichier :** test4_plusieurs_erreurs.py

python

```
foreach item liste :{    if item > :    {        print(@item)    }}
```

**Résultat attendu :**

```
ERREUR LEXICALE : @ERREUR SYNTAXIQUE [Position 2] : Attendu mot-cle: inERREUR SYNTAXIQUE [Position 5] : Expression invalide
```

---

# 8. Utilisation

## 8.1 Compilation

bash

```
mvn clean package
```

## 8.2 Exécution

bash

```
java -jar target/Analyseur_Lexical-1.0-SNAPSHOT.jar
```

## 8.3 Tests Automatiques

Le programme exécute automatiquement tous les fichiers de test et affiche les résultats.

---

# 9. Technologies Utilisées

- **Langage :** Java 21
- **Build Tool :** Maven
- **IDE :** NetBeans 25
- **Gestion de versions :** Git / GitHub

---

# 10. Améliorations Possibles

1. **Analyse sémantique** : Vérification des types, portée des variables
2. **Génération de code** : Production de bytecode ou code machine
3. **Optimisations** : Élimination de code mort, propagation de constantes
4. **Meilleurs messages d'erreur** : Suggestions de correction
5. **Support de plus d'instructions** : classes, fonctions, exceptions

---

# 11. Conclusion

Ce projet m'a permis de comprendre les bases de la construction d'un compilateur :

- **Analyse lexicale** : Décomposition en tokens
- **Analyse syntaxique** : Vérification de la structure
- **Gestion d'erreurs** : Détection et récupération