Université de Lille

FACULTÉ DES SCIENCES ET TECHNOLOGIES
Département Informatique

TDM 4 - Automates et Langages Réguliers

octobre 2021

Analyse lexicale - coloration syntaxique

Le but de cet exercice est de produire un analyseur lexical capable de découper une programme source JAVA en une suite de tokens, puis de l'utiliser en vue de produire une version colorisée du programme. Les tokens composant les textes sources JAVA seront les suivants :

1. `IDENT` : **identificateurs**
   Une identificateur Java peut comporter des lettres (majuscules ou minuscules), chiffres décimaux, ainsi que `$` et `_`. Le premier caractère ne peut pas être un chiffre.
2. `RESERVED` : **mots réservés**
   Ces mots ne peuvent pas être utilisés comme identificateurs ordinaires. Par exemple `while`, `for`, `new`, `class`, `and`... Une liste complète des mots réservés figure dans les fichiers fournis.
3. `STRING` : **chaînes**
   Une chaîne est délimitée par des guillemets `"`. Elle ne peut **contenir** le caractère `"` ni `\` , sauf s'ils constituent l'une des séquences `\\ \" \' \b \t \n \f \r`
   Par exemple `"x\yc"` n'est pas une chaîne correcte, ni `"x"c"`, ni `"xy\"`, alors que `"a\nb\\c\"d"` l'est.
4. `INT` : **entiers**
   Java admet 4 manières d'écrire les entiers (attention, il y a des différences avec la syntaxe Python) :
   - **écriture décimale** : les chiffres autorisés sont compris entre `0` et `9`. **Le premier chiffre doit être différent de** `0`. Exemples : `45` `19` `4058`
   - **écriture octale** : les chiffres autorisés sont compris entre `0` et `7`. **Le premier chiffre doit être égal à** `0`. Exemples : `045` `01` `0`
   - **écriture binaire** : les chiffres autorisés sont `0` et `1`. Le nombre est précédé du préfixe `0b` ou `0B` . Exemples : `0b10110` `0b0101` `0b000101` `0b0` `0b1`
   - **écriture hexadécimale** : sont autorisés les chiffres compris entre `0` et `9`, ou les lettres entre `A` et `F` ou entre `a` et `f`. Le nombre est précédé du préfixe `0x` ou `0X`. Exemples : `0x1ea7` `0x01ea7`

   De plus, chacune des 4 formes admet la présence de un ou plusieurs caractères underscore (`_`), **entre** les chiffres (pas au début ni à la fin, ni dans le préfixe). Exemples :
   `0b10___1_10` `1_2_3` `012_456` `0_50` sont autorisés,
   mais `0b_1110` `0b1110_` `0_b1110` `_123` sont interdits
5. `FLOAT` : **nombres avec partie décimale**
   Ces nombres sont constitués d'une mantisse éventuellement suivie d'un exposant.
   - la **mantisse** comporte un point encadré de deux « séquences décimales ». L'une **seule** des séquences décimales peut être omise : soit celle qui précède le point, soit celle qui le suit (voir exercice du TD1). Une « séquence décimale » est une suite quelconque de chiffres entre `0` et `9`. Le premier chiffre peut cette fois être un `0`. Chaque séquence décimale peut comporter des underscore dans les mêmes conditions que pour les entiers.
   - l'**exposant** est un `e` ou `E` éventuellement suivi d'un signe et obligatoirement suivi d'une séquence décimale.
6. `COMMENT` : **commentaires**
   Deux formes de commentaires sont autorisés :
   - commençant par `//`, ils se terminent à la fin de la ligne (et donc ne peuvent contenir le caractère fin de ligne)
   - encadrés par `/*` et `*/` Ils peuvent cette fois s'étendre sur plusieurs lignes (donc comporter des fins de ligne).
7. `ASSIGN` : **affectations**
   Il s'agit du signe égal, qui peut être précédé d'un opérateur parmi `- + * / % & | ^`
   Exemples : `=` `+=` `*=` `&=` `|=`

8. `OP` : **opérateurs**
   `- + * / % & | ^ >>> >> << < <= > >= != == && ||`
9. `SPACES` : **espaces**
   Toute suite de caractères d'espacement (hors fin de ligne, donc suite de caractères espace ou tabulation) constituera un token `SPACES`.
10. `EOL` : **fin de ligne**
    Chaque fin de ligne sera traduite en un token `EOL`. Vous assurerez la gestion du compteur de lignes `self.lineno`
11. `OTHER` : **autres**
    Un caractère qui n'entre pas dans l'un des tokens ci-dessus constituera un token `OTHER`.

NB : quelques simplifications mineures ont été apportées par rapport à la véritable syntaxe Java.

Exercice 1 :
Q 1 . Réaliser un analyseur lexicale permettant de décomposer un texte en une séquence des tokens définis ci-dessus.
Un squelette de solution `java_lexer_skel.py` vous est fourni.
Renommez ce fichier en `java_lexer.py` puis complétez son développement.
Q 2 . Application : coloration syntaxique
Construire un programme python permettant de produire un texte HTML traduisant un fichier source java en un élément HTML avec un balisage des token permettant de colorer les différentes parties du programme selon leur token (exemple : les identificateurs en bleu, les entiers en jaune , etc)
Les textes des tokens (sauf SPACES, EOL, OTHER) seront encapsulés dans un élément
`<span class=" ...">.. </span>` dont le nom de classe sera celui du token (en minuscules).

Exemple :

```
<span class="ident">quantity</span> <span class="assign">=</span> <span class="int">0</span>;
```

Un squelette de programme pour cette question vous est également fourni (à compléter).
Un fichier de style CSS est également donné (vous pouvez le compléter ou l'aménager).