Master 1 – Compilation
TP n°2
Coloration de code source python

Ce TP sur l'analyse lexicale correspond à un projet plus ambitieux puisqu'il s'agit de coloriser les différents éléments du langage python en produisant directement du HTML.

Contexte

Cette technique est connue comme étant de la coloration syntaxique mais qui correspond en fait à de la coloration lexicale. Nous allons dans notre cas appliquer ce principe à la coloration des programmes python, le but final étant d'obtenir une coloration syntaxique écrite en HTML à partir d'un fichier python :

test.py

```
# fichier de test
valeur = 1

class Toto:
    def __init__(self, x):
        self.ma_valeur = x
        self.ma_valeur = self.ma_valeur*2

    def __repr__(self):
        return repr(self.ma_valeur)

t =  Toto(5)
print(t)
```

ce qui correspond au fichier HTML généré suivant :

```
<html>
<body style="background-color:#2B2B2B;">
<H2> node.py </H2>
<CODE>
<FONT color=808080># fichier de test</FONT>
<BR><FONT color=A9B7C6>valeur</FONT>
<FONT color=FFFFFF>=</FONT>
```

<FONT color=6897BB>1</FONT>

<FONT color=CC7832>class</FONT>
<FONT color=A9B7C6>Toto</FONT>
<FONT color=FFFFFF>:</FONT>

<FONT color=CC7832>def</FONT>
<FONT color=B200B2>__init__</FONT>
<FONT color=FFFFFF>(</FONT>
<FONT color=94558D>self</FONT>
<FONT color=CC7832>,</FONT>
<FONT color=A9B7C6>x</FONT>
<FONT color=FFFFFF>)</FONT>
<FONT color=FFFFFF>:</FONT>

<FONT color=94558D>self</FONT>
<FONT color=FFFFFF>.</FONT>
<FONT color=A9B7C6>ma_valeur</FONT>
<FONT color=FFFFFF>=</FONT>
<FONT color=A9B7C6>x</FONT>

<FONT color=94558D>self</FONT>
<FONT color=FFFFFF>.</FONT>
<FONT color=A9B7C6>ma_valeur</FONT>
<FONT color=FFFFFF>=</FONT>
<FONT color=94558D>self</FONT>
<FONT color=FFFFFF>.</FONT>
<FONT color=A9B7C6>ma_valeur</FONT>
<FONT color=FFFFFF>*</FONT>
<FONT color=6897BB>2</FONT>

<FONT color=CC7832>def</FONT>
<FONT color=B200B2>__repr__</FONT>
<FONT color=FFFFFF>(</FONT>
<FONT color=94558D>self</FONT>
<FONT color=FFFFFF>)</FONT>
<FONT color=FFFFFF>:</FONT>

<FONT color=CC7832>return</FONT>
<FONT color=A9B7C6>repr</FONT>
<FONT color=FFFFFF>(</FONT>
<FONT color=94558D>self</FONT>
<FONT color=FFFFFF>.</FONT>
<FONT color=A9B7C6>ma_valeur</FONT>
<FONT color=FFFFFF>)</FONT>

<FONT color=A9B7C6>t</FONT>
<FONT color=FFFFFF>=</FONT>
<FONT color=A9B7C6>Toto</FONT>
<FONT color=FFFFFF>(</FONT>
<FONT color=6897BB>5</FONT>
<FONT color=FFFFFF>)</FONT>

<FONT color=FFFFFF>print</FONT>
<FONT color=FFFFFF>(</FONT>
<FONT color=A9B7C6>t</FONT>
<FONT color=FFFFFF>)</FONT>
</body>
</html>

ce qui donne visuellement le fichier suivant :

node.py

```
# fichier de test
valeur = 1

class Toto :
def __init__ ( self , x ) :
self . ma_valeur = x
self . ma_valeur = self . ma_valeur * 2

def __repr__ ( self ) :
return repr ( self . ma_valeur )

t = Toto ( 5 )
print ( t )
```

`C'est plutôt pas mal non ?`

Pour le moment, votre analyseur lexical écrit grâce à ply prendra en compte les éléments suivants :

1. les mots-clefs « classiques » qui auront tous la même couleur `if else elif while break for return continue import class in return not unsigned True False def enumerate`… La liste n'est pas exhaustive et il y en aura certainement à rajouter. Nous leur associerons à tous un unique type de token, le type MC (pour Mot-clef). Comme cela est indiqué dans la documentation de ply, il est inutile d'écrire une expression régulière pour chacun de ses éléments, il faut tous les mettre dans une liste de mots-clefs réservés (cf paragraphe 4,3 de la documentation https://www.dabeaz.com/ply/ply.html)
2. les autres mots clefs que nous traiterons séparément print, len, str, self
3. les entiers et les réels ;
4. les importations (les lignes qui commencent par `import` xxx.yyy.zzz) ;
5. les chaînes de caractères entre guillemets ou entre apostrophes ;
6. les identificateurs qui en python commencent par une lettre ou le symbole "_" et se poursuivent par des lettres, le symbole "_" et des chiffres ;
7. les différents opérateurs +,-,<,>=,== et signes [ ] : ,
8. les variables spéciales qui en python commencent par __ et se terminent également par __ ;
9. les commentaires qui commencent par un # et se terminent à la fin de la ligne

A chacune de ces unités lexicales, on lui affectera une couleur propre et chaque couleur correspondra simplement à un affichage du code suivant :
<FONT color=xxx> </FONT>

Par exemple, pour générer la chaîne de caractères en rouge vous devrez produire le code suivant :

<FONT color=7f0000>"création de l'objet"</FONT>

J'ai utilisé les couleurs suivantes pour mon code mais cela peut-être affiné mais cela produit un code

assez joli à regarder

entier, réel	6897BB
virgule	CC7832
mots-clefs	CC7832
identificateur	A9B7C6
mots-clefs spéciaux (__len__)	B200B2
commentaires	808080
chaîne	6A8759
le, str, self	94558D
autre	FFFFFF

Travail à réaliser

Écrire votre fichier `coloration.py` qui permettra se reconnaître ces différents tokens

La façon la plus courante de coloration d'un texte HTML est d'utiliser des codes de couleur hexadécimal par exemple pour un texte en blanc :

Votre fichier `coloration.py` devra sortir tout le code au format HTML par exemple dans le même nom que le fichier passé en paramètre mais cette fois-ci avec l'extention HTML. N'oubliez pas que vous devez également transformer certains caractères par exemple < devra être transformé en HTML en >

Ajout de l'en-tête et du footer

Ces éléments sont indispensables pour visualiser le code HTML que vous générez, j'ai utilisé un fond sombre afin de faire ressortir le texte :

```
<html>
<body style="background-color:#2B2B2B;">
<H2> node.py </H2>
<CODE>

</body>
</html>
```