Analyse lexicale

Série 1.2 : Expressions régulières en Python

Exercice 1

1. Filtrer les lignes

Écrire un programme python **`filter.py`** qui fonctionne en ligne de commande et qui :

- prend deux arguments :
  - le premier est une expression régulière,
  - le deuxième est le nom d'un fichier texte.
- affiche les lignes du fichier texte qui correspondent à l'expression régulière.

Exemple : Soit le fichier **`source.txt`** suivant :

```
bonjour
0
1
12, le chiffre est 12
1.6
.6
-1.
-1E-05
-12.23E12
 Commence par un espace
  Commence par deux espaces
      Commence par une Tabulation
            Commence par deux tabulations

A
a
a.a
```

On aura alors :

```
> python filter.py "\d+" source.txt
0
1
12, le chiffre est 12
1.6
.6
-1.
-1E-05
-12.23E12
>
```

Indications : Vous aurez essentiellement besoin de **`re.compile()`** et de la méthode **`search()`** d'un objet Expression Régulière. Si nécessaire, vous pourrez également vous aider de :

- http://docs.python.org/release/3.2/howto/regex.html
- http://docs.python.org/release/3.2/library/re.html

2. Utilisation

Utilisez votre programme ci-dessus pour trouver dans différents fichiers textes les lignes qui :

- contiennent des nombres avec une virgule en notation scientifique.
- commencent par un espace ou une tabulation.
- ne contiennent que des caractères alphanumériques.
- sont non vides.
- contiennent un mot commençant par une majuscule.

Exercice 2

1. Une recherche un peu plus fine

Modifiez votre **`filter.py`** en **`find.py`** pour qu'il n'affiche plus toute la ligne, mais seulement les parties qui correspondent à l'expression régulière.

Exemple : Soit le fichier `source2.txt` suivant :

```
  ligne de code;  //commentaire1 java
//    un autre commentaire2 //
//
toto@gmail.com
to-to@gmail.com
francois.tieche@he-arc.ch
françois.tièche@he-àrc.ch

http://alpha.ch/src/toto.html

a camelCase word
a CamelCase word
not a CAMELCase word
not a CamelcasE word
```

On aura alors :

```
> python filter.py "\d+" source.txt
  ligne de code;  //commentaire1 java

//    un autre commentaire2 //
>
```

Indication : Ici, c'est plutôt la méthode **`findall`** qui va vous intéresser.

2.2 Utilisation

Utilisez votre programme pour extraire :

- Les commentaires "`//`" d'un fichier java.
- Les adresses mail d'un fichier texte.
- Les URLs d'un fichier html

- Les mots en CamelCase d'un fichier texte (Il existe plusieurs définitions du CamelCase. Ici, nous prendrons : mots formés uniquement de lettres et contenant au moins une majuscule précédée et suivie d'une minuscule).

. . . Déja fini ?

Pour les plus rapides ou les plus motivés, quelques suggestions d'approfondissement (pas forcément dans l'ordre !) :

Tout savoir sur les expressions régulières en python

Prenez la peine de lire jusqu'au bout **`http://www.amk.ca/python/howto/regex/`**, vous y apprendrez tout plein de choses très utiles !

Les expressions régulières et le chercher-remplacer

Prenez un bon éditeur de texte avec possibilité de faire du chercher/remplacer par expressions régulières (p.ex. SciTe).

1. Comme échauffement, remplacez le premier caractère de chaque ligne d'un fichier par un "A" (fondamentalement inutile !)
2. Remplacez dans un fichier toutes les dates au format YYYY/MM/DD par leur équivalent au format DD/MM/YYYY (nous sommes francophones, après tout. . .)
3. Vous avez un code utilisant une fonction f qui prend deux arguments. Vous venez de changer la signature de cette fonction et devez systématiquement doubler le premier argument dans les 1'003'542 appels déjà écrits de votre fonction. :-)
   Par exemple, f(x,y) doit devenir f(x,x,y).
   Attention : les appels existants de f avec un nombre différent d'arguments ne doivent pas être modifiés !
4. Imaginez d'autres situations où un chercher-remplacer à l'aide d'expressions régulières vous sera d'un secours inestimable.

À la découverte de `grep`, `sed` et `awk`

`grep`, **`sed`** et **`awk`** sont trois outils indispensables du monde Unix (mais pas seulement...).

1. `grep` est une version évoluée de notre `filter.py` ci-dessus
   - http://fr.wikipedia.org/wiki/Grep
   - http://www.panix.com/~elflord/unix/grep.html
   - http://www.gnu.org/software/grep/manual

2. **`sed`** est une version archi-évoluée du chercher-remplacer avec expressions régulières
   - http://fr.wikipedia.org/wiki/Sed_(logiciel)
   - http://www.gnu.org/software/sed/manual/sed.html
   - http://sed.sourceforge.net/sed1line.txt
   - http://sed.sourceforge.net/#docs

3. **`awk`** est un langage de programmation assez inhabituel, fortement basé sur les expressions régulières, les chaînes de caractères et les tableaux associatifs

- http://en.wikipedia.org/wiki/Awk
- http://www.well.ox.ac.uk/~johnb/comp/awk/awk.html
- http://awk.info/

Vous en demandez encore ? Il ne vous reste plus qu'à vous mettre à **`Perl`**. . .