

# TD 11

## Traitement de données linguistiques avec Python et la logithèque Unix

### Modules et packages

Télécharger test\_package.zip .

- Un module, c'est juste un fichier .py dont on peut importer les fonctions.
- Un package, c'est juste un dossier dont on peut importer des modules.
  - Le dossier du package contient un fichier `__init.py__` **facultatif depuis Python 3.2** ; si il n'en contient pas, c'est un *namespace package*.

Cf. <https://stackoverflow.com/questions/51481273/difference-between-library-vs-module-vs-package-vs-object-in-python>

### Préparer un script Python pour une utilisation en tant que module *et* en tant que programme

Templates :

- Lecture des arguments :
  - Méthode basique avec `sys.argv` : testarg1.py
    - testarg1.py
    - testarg1.py -c 5 -v toto
  - Méthode enrichie avec la [lib argparse](#) : testarg2.py
    - testarg2.py
    - testarg2.py -c 5 -v toto

### Parallélisme et pipes

Note sur le parallélisme en Python (*dans* Python, donc pas avec des pipes qui sont utilisés *en dehors* de Python) :

- `multithreading` → mono-cœur (soumis au GIL – Global Interpreter Lock), utile pour un programme qui passe du temps à *attendre* (un téléchargement, une lecture/écriture de fichier, une grosse requête en base de données, etc.), ou bien qui doit rester *réactif* (GUI)
- `multiprocessing` → vrai multi-cœur, plus compliqué à utiliser

<https://stackoverflow.com/questions/3044580/multiprocessing-vs-threading-python>

<https://realpython.com/python-concurrency/>



L'option -o permet de restreindre la sortie à la partie de la ligne qui a matché (on n'affiche pas le reste). Essayez d'attraper toutes les occurrences de aïe, sans leur contexte.

Option -i pour insensible à la casse. Utilisez la pour votre recherche.

Il y a du bruit (*paie*)... Comment afficher seulement les occurrences du mot aïe, et pas tous les mots qui contiennent aïe ?

<https://www.regular-expressions.info/unicode.html>

<https://stackoverflow.com/questions/2973436/regex-lookahead-lookbehind-and-atomic-groups>

## Chercher/remplacer dans un flux avec sed et perl

sed est installé partout, option -r pour passer en regex étendues, pas de PCRE ;-(.

- `cat 88mSMS.xml | sed -re 's/</[/'`
  - seulement la première occurrence de chaque ligne (comme `regex.sub(pattern, repl, string)` en Python)
- `cat 88mSMS.xml | sed -re 's/</[/'g'`
  - toutes les occurrences de la ligne (comme `regex.subn(pattern, repl, string)` en Python)
- `cat 88mSMS.xml | sed -re 's/\p{L}+/A'`
  - sed ne sait pas ce que c'est un `\p{L}`
- `cat '88mSMS.xml' | sed -re 's/.*/g'`
- `cat '88mSMS.xml' | sed -re 's/.+//g'`

### Exercices

- Enlever tous les tags du document :
- Tokéniser avec sed

perl n'est pas installé partout, mais est PCRE ; on peut l'utiliser en remplacement de sed avec l'argument -pe .

- `cat mul.py | perl -C -pe 's/\n//g'`

- Supprimer les retours à la ligne dans mul.py :
- cat 88mSMS.xml | perl -pe 's/\p{L}+/A/'
  - seulement la première occurrence de chaque ligne (comme regex.sub(pattern, repl, string) en Python)
- cat 88mSMS.xml | perl -pe 's/\p{L}+/A/g'
  - toutes les occurrences de la ligne (comme regex.subn(pattern, repl, string) en Python)
- cat 88mSMS.xml | perl -pe 's/(\p{L}+)/uc(\$1)/ge'
  - avec l'option **e**, on peut écrire du Perl en partant de la droite, ici pour passer les caractères en majuscule... mais ça ne marche pas pour les caractères non-ASCII (= non-latins)
- cat 88mSMS.xml | perl -C -pe 's/(\p{L}+)/uc(\$1)/ge'
  - il faut préciser à Perl que **STD est en UTF-8**, et ça marche !
- Tokéniser avec perl
  - cat '88mSMS.xml' | sed -re 's/<[^>]+>//g' | perl -C -pe 's/\P{L}\*(\p{L}+)\P{L}\*/\$1\n/g'
- Comment compter les mots (avec *sort* et *uniq*) ?
  - Comment faire pour compter les mots sans tenir compte de la casse ?

## Édition de flux avec awk

<https://fr.wikipedia.org/wiki/Awk>

<https://www.malekal.com/comment-utiliser-la-commande-awk-avec-des-exemples/>

```
echo 'une ligne' | awk 'pattern { action }' FS="\t"
```

```
echo 'une ligne' | awk 'pattern { action } pattern { action }' FS="\t"
```

Si pas de pattern, *action* se déclenche pour toutes les lignes.

Dans *action* comme dans *pattern*, \$0 désigne toute la ligne, \$1 le premier champ, \$2 le deuxième, etc. NR désigne le numéro de ligne courante.

Pour afficher le fichier lexPresto\_20151027.csv :

```
cat lexPresto_20151027.csv | awk '{print $0}' FS="\t"
```

Seulement la seconde colonne :

```
cat lexPresto_20151027.csv | awk '{print $2}' FS="\t"
```

Seulement la seconde colonne des lignes dont la première colonne finit par un *e* :

```
cat lexPresto_20151027.csv | awk '$1 ~ /e$/ {print $2}' FS="\t"
```

Dans *pattern*, les patterns spéciaux BEGIN et END désignent le début et la fin du fichier.

## Exercices

Afficher seulement la colonne 4 avec awk.

Afficher toutes les lignes dont la colonne 4 commence par DMF.

Afficher toutes les lignes dont la colonne 4 ne commence *pas* par DMF.

Afficher toutes les lignes dont la 5<sup>e</sup> colonne est > 0.

Ajouter une première colonne avec le numéro de ligne.

Supprimer les 10 premières lignes

Encadrer le fichier mul.py par des \*\*\*\*\*.

Additionner les valeurs de la colonne 5.

Afficher le lexique entre les formes faire (inclus) et finir (exclu) [c'est le cas le + simple]

Afficher le lexique