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/

Utiliser les *pipes* : lire depuis STDIN, et écrire sur STDOUT

- Fonctionnement des *pipes* (remplissage des tuyaux, mise en pause, buffer), comment bien utiliser les pipes : /!\ document complets XML, Json, sort, etc.
- Exemples :
  - `echo 2 | ./mul.py 2`
  - ```
    for i in $(seq 0 1000); do echo $i | ./mul.py 100 | ./mul.py 100  |
    ./mul.py 100 | ./mul.py 100 | ./mul.py 100 | ./mul.py 100 | ./mul.py
    100 | ./mul.py 100 | ./mul.py 100 | ./mul.py 100  | ./mul.py 100 |
    ./mul.py 100 | ./mul.py 100 | ./mul.py 100 | ./mul.py 100 | ./mul.py
    100; done
    ```
    - essayer avec un `mul.py 1000000000`
  - Dans Python :

    /!\ Appelle le shell du système, pas forcément le même suivant la machine

    ```
    import subprocess
    result = subprocess.check_output("for i in $(seq 0 1000); do echo
    $i | ./mul.py 100 | ./mul.py 100  | ./mul.py 100 | ./mul.py 100 |
    ./mul.py 100 | ./mul.py 100 | ./mul.py 100 | ./mul.py 100 | ./mul.py
    100 | ./mul.py 100  | ./mul.py 100 | ./mul.py 100 | ./mul.py 100 |
    ./mul.py 100 | ./mul.py 100 | ./mul.py 100; done", shell=True,
    text=True)
    ```

Flux et regex

Rappel :

- les regex de base supportent .* mais pas .+
- les regex étendues (POSIX) supportent .+ mais pas .+?
- les regex Perl-compatibles (PCRE) supportent .+? et \p{L}

Matcher dans un flux avec grep

Exercices

Afficher seulement les lignes de type SMS.

```
cat 88mSMS.xml | grep 'type="sms"'
```

L'option -v inverse la sélection. Afficher seulement les lignes qui ne sont *pas* de type SMS.

```
cat 88mSMS.xml | grep -v 'type="sms"'
```

Supprimer les lignes vides du fichier mul.py

```
cat mul.py | grep -v '^$'
```

L'option -P passe en mode compatible PCRE (pas dispo sur toutes les plateformes...). Essayez d'attraper toutes les lignes qui contiennent une occurrences de *aïe*, quelle soit sa graphie (p. ex. *aïiiie*). On attrapera aussi les mots qui contiennent des graphies comme *aie* (p.ex. *paie*), mais pour l'instant ce n'est pas grave.
```

```
cat 88mSMS.xml | grep -P A[iï]+e
```

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.

```
cat 88mSMS.xml | grep -Po A[iï]+e
```

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

```
cat 88mSMS.xml | grep -Pi A[iï]+e
```

```
cat 88mSMS.xml | grep -Pio A[iï]+e
```

Il y a du bruit (p*aie*)… Comment afficher seulement les occurrences du mot aïe, et pas tous les mots qui contienne *aïe* ?

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

cat '88mSMS.xml' | grep -Pio '\bA[iï]+e\b'

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

cat '88mSMS.xml' | grep -Pio '(?<=\P{L})A[iï]+e(?=\P{L})'

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.sub**n**(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 :
  - `cat '88mSMS.xml' | sed -re 's/<[^>]+>//g'`
- Tokéniser avec sed
  - `cat '88mSMS.xml' | sed -re 's/<[^>]+>//g' | sed -re 's!\b!\n!g'`
  - `cat '88mSMS.xml' | sed -re 's/<[^>]+>//g' | sed -re 's!\s*\b(\w+)\b!\1\n!g'`

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.sub**n**(pattern, repl, string) en Python)
- cat 88mSMS.xml | perl -pe 's/**(**\p{L}+)/uc($1)/g**e**'
    - avec l'option **e**, on peut écrire du Perl en partir 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*) ?
    - ```
      cat '88mSMS.xml' | sed -re 's/<[^>]+>//g' |
      perl -C -pe 's/\P{L}*(\p{L}+)\P{L}*/$1\n/g' |
      sort | uniq -c | sort -n
      ```
    - Comment faire pour compter les mots sans tenir compte de la casse ?
        - ```
          cat '88mSMS.xml' | sed -re 's/<[^>]+>//g' |
          perl -C -pe 's/(\p{L}+)/uc($1)/ge' |
          perl -C -pe 's/\P{L}*(\p{L}+)\P{L}*/$1\n/g' |
          sort | uniq -c | sort -n
          ```

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.

```
cat lexPresto_20151027.csv | awk '{print $4}' FS="\t"
```

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

```
cat lexPresto_20151027.csv | awk '$4 ~ /DMF.+/ {print $0}' FS="\t"
```

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

```
cat lexPresto_20151027.csv | awk '$4 !~ /DMF.+/ {print $0}' FS="\t"
```

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

```
cat lexPresto_20151027.csv | awk '$5 > 0 {print $0}' FS="\t"
```

Ajouter une première colonne avec le numéro de ligne.

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

Supprimer les 10 premières lignes

```
cat lexPresto_20151027.csv | awk 'NR > 10 {print $0}' FS="\t"
```

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

```
cat mul.py | awk 'BEGIN { print "**********" } {print $0} END { print
"**********" }' FS="\t"
```

Additionner les valeurs de la colonne 5.

```
cat lexPresto_20151027.csv | awk '{MYSUM=MYSUM+$5} END {print MYSUM}' FS="\t"
```

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

```
cat lexPresto_20151027.csv |
awk '$1~/^faire$/ {flag=1} $1~/^finir$/ {flag=0} flag {print $0}' FS="\t"
```

Afficher le lexique

```
cat lexPresto_20151027.csv |
awk '$1~/^faire$/ {flag=1; next} $1~/^finir$/ {flag=0; print $0} flag {print
$0}' FS="\t"
```