Python et le TAL

initiation aux bases de Python

Qu’est-ce que Python ?

- langage de programmation orienté objet
- simple à écrire et facile à lire (pseudo-code : description d’une logique d’un algorithme qui pourrait s’appliquer à n’importe quel langage) :

```python
def tri(tableau):
    if len(tableau) <= 1:
        return tableau
    pivot = tableau[0]
    gauche = [x for x in tableau if x < pivot]
    milieu = [x for x in tableau if x == pivot]
    droite = [x for x in tableau if x > pivot]
    return tri(gauche) + milieu + tri(droite)
    print tri([7,8,3,10,6,2,1])
# imprime "[1, 2, 3, 6, 7, 8, 10]"
```

Algorithme de tri

Tableau de base : `[7,8,3,10,6,2,1]`
On va faire bouger les nombres inférieurs au **pivot** à sa gauche, et supérieurs à sa droite
On crée **deux partitions**, une gauche (x<pivot), et une droite(x>pivot)
on va procéder par déplacement:
ordre après 2 tris : 7,3,8 (3 est inférieur, 8 est supérieur)
ordre après 3 tris : 7,3,8,10
4 tris : 7,3,6,8,10, etc.
Fonction **récursive** : on va refaire la même chose dans les partitions gauches et droites tant que chaque partition contient plus d’un élément.

Les types basiques (dont vous vous servirez)

- les integers : nombres entiers (3)
- les floats : nombres quels qu'ils soient (3.5)
- les booleans : des valeurs qui signifient soit "vrai", soit "faux" :

```
def egalite(num1,num2)
if num1==num2
verification=true
return verification
print egalite(1,2)
# imprime "False"
```

- les strings : des chaînes de caractères

```
hello = 'hello'    # chaîne avec juste guillements simples
world = "world"    # ou doubles guillemets, peu importe
print hello        # imprime "hello"
print len(hello)   # longueur de la chaîne ; imprime "5"
hw = hello + ' ' + world  # concaténation de string
print hw  # imprime "hello world"
```

Les conteneurs (dont vous vous servirez, même s'il y en a d'autres)

- les listes : pourquoi ?
  - parce qu'elles sont facilement modifiables (on peut y ajouter des éléments au fur et à mesure, contrairement aux tableaux, et facilement où on veut)
  - parce qu'en Python (c'est spécifique à Python) elles peuvent être hybrides (de plusieurs types)

```
1  maListe= [3,1,2]
2  maListe.append('autreVal')
3  print maListe # imprime [3,1,2,'autreVal']
4  maListe.insert(1,'encoreUne')
5  print maListe # imprime [3,'encoreUne',1,2,'autreVal']
6  print maListe[2:4]
7  # imprime une tranche depuis indice 2 inclus à indice 4 exclu, donc [1,2]
8  maListe[2:4]=[8,9]
9  print maListe
10 # imprime [3,'encoreUne',8,9,'autreVal']
```

Comment parcourir une liste ?

- le parcours par indice (on spécifie l'indice de position, on y accède)
- les boucles :
  - la plus connue, la boucle "for"

```
animaux = ['chat', 'chien', 'singe']
for animal in animaux:
    print animal
# Prints "chat", "chien", "singe", avec une nouvelle ligne pour chacun
```

  - avec la possibilité d'accéder à l'indice

```
animaux = ['chat', 'chien', 'singe']
for idx, animal in enumerate(animaux):
    print '#%d: %s' % (idx + 1, animal)
# imprime "#1: chat", "#2: chien", "#3: singe", chacun sur sa ligne
```

  - avec concaténation simple

```
nums = [0, 1, 2, 3, 4]
carres = []
for x in nums:
    carres.append(x ** 2)
print carres    # imprime [0, 1, 4, 9, 16]
```

```
nums = [0, 1, 2, 3, 4]
carres = [x ** 2 for x in nums]
print carres    # imprime [0, 1, 4, 9, 16]
```

Les dictionnaires

- ils servent à stocker des clés et leurs valeurs (équivalent d'une Map)

```
1  d = {'chat': 'félin', 'chien': 'canin'}  # nouveau dictionnaire avec données
2  print d['chat']        # va chercher la valeur de la clé "chat"
3  print 'chat' in d      # vérifie si chat est une clé dans d
4  d['poisson'] = 'mouillé'    # crée une clé "poisson" avec valeur "mouillé"
5  print d['poisson']       # imprime "mouillé"
6  # print d['singe']  # KeyError: 'singe' ne fait pas partie du dictionnaire
7  print d.get('singe', 'N/A') # va chercher un élément et donne une valeur défaut
8  print d.get('poisson', 'N/A') # va chercher un élément et donne sa valeur
9  del d['poisson']         # enlève la clé du dictionnaire
10 print d.get('poisson', 'N/A') # "poisson" n'est plus une clé, imprime "N/A"
```

Ils sont faciles à parcourir, dans un sens ou dans l'autre

```
d = {'humain': 2, 'chat': 4, 'crabe': 8}
for animal in d:
    jambes = d[animal]
    print 'Un %s a %d jambes' % (animal, jambes)
# imprime "Un humain a 2 jambes", "Un crabe a 8 jambes", "Un chat a 4 jambes"
```

```
d = {'humain': 2, 'chat': 4, 'crabe': 8}
for animal, jambes in d.iteritems():
    print 'Un %s a %d jambes' % (animal, jambes)
# Imprime "Un humain a 2 jambes", "Un crabe a 8 jambes", "Un chat a 4 jambes"
```

Les Sets et les Tuples

Normalement vous n’en aurez pas besoin tout de suite, mais mieux vaut savoir ce que c’est :

- les sets sont des listes d’éléments uniques (quand un élément apparaît plusieurs fois, il n’est pas réitéré)
- les tuples sont très proches des listes (traditionnellement, les listes sont typées, et mutables, à l’inverse des tuples)
    - exemple : ((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))

Les Fonctions

```
1  def signe(x):
2      if x > 0:
3          return 'positif'
4      elif x < 0:
5          return 'négatif'
6      else:
7          return 'zéro'
8
9  for x in [-1, 0, 1]:
10     print signe(x)
11 # Imprime "négatif", "zéro", "positif"
```

Les Bibliothèques

Une **bibliothèque logicielle** est une collection de routines, qui peuvent être déjà compilées et prêtes à être utilisées par des programmes. Les bibliothèques sont enregistrées dans des fichiers semblables, voire identiques aux fichiers de programmes

Liste des bibliothèques dont vous aurez sûrement besoin :

- un parseur xml (j'utilise lxml) pour Python
- NLTK (vraiment excellent), ou StanfordCoreNLP (plus performant mais plus complexe à utiliser
- des bibliothèques de représentation graphique, comme matplotlib

te

arseur XML

permet de parcourir l'arborescence d'un

```
<TEI>
  <teiHeader>
    <fileDesc>
      <titleStmt>
        <title>Les Misérables</title>
        <author>Hugo, Victor</author>
      </titleStmt>
      <publicationStmt>
        <p>Texte normalisé pour expériences textométriques</p>
      </publicationStmt>
      <sourceDesc>
        <bibl>Gutenberg, tomes
          <ref target="http://www.gutenberg.org/ebooks/17489">1</ref>,
          <ref target="http://www.gutenberg.org/ebooks/17493">2</ref>,
          <ref target="http://www.gutenberg.org/ebooks/17494">3</ref>,
          <ref target="http://www.gutenberg.org/ebooks/17518">4</ref>,
          <ref target="http://www.gutenberg.org/ebooks/17519">5</ref>.
        </bibl>
      </sourceDesc>
    </fileDesc>
    <profileDesc>
      <creation/>
      <langUsage>
        <language ident="fr"/>
      </langUsage>
    </profileDesc>
  </teiHeader>
  <text>
    <body>
      <div xml:id="t1">
        <head>Tome I — Fantine</head>
        <div xml:id="t1-01">
          <head>Livre premier — Un juste</head>
          <div type="chapter" xml:id="t1-01-01">
            <head>Chapitre I.
Monsieur Myriel</head>
            <p>En 1815, M. Charles-François-Bienvenu Myriel était évêque
ans ; il occupait le siège de Digne depuis 1806.</p>
```

```
1 from lxml import etree
2 tree = etree.parse("hugo_miserables.xml")
3 listeHugo=list()
4 listeHugo.extend(user.text for user in tree.xpath("/TEI/text/body/div/head"))
5 hugoIntegral=''
6 for e in listeHugo:
7     if (isinstance(e,str)):
8         hugoIntegral+=e+' '
9 print (hugoIntegral)
```

NLTK ?

Natural Language ToolKit : l'avantage de NLTK, c'est qu'il vous suffit de télécharger ses données intégrées, pour chaque langue ou tout le paquet (attention, c'est long…) : vous avez juste à appeler l'import depuis votre compilateur Python, puis à importer le modèle que vous voulez :

```
1 import nltk
2 nltk.download()
3 # chargement du tokenizer
4 tokenizer = nltk.data.load('tokenizers/punkt/PY3/french.pickle')
```

Etape 1 : Tokéniser

```
1 tokenizer = nltk.data.load('tokenizers/punkt/PY3/french.pickle')
2 tokens = tokenizer.tokenize(hugoIntegral) # retourne une liste de strings
```

ce qui retourne tous les mots du texte entre virgules, à savoir les éléments de la liste :

```
['Chapitre', 'I.', 'Monsieur', 'Myriel', 'En', '1815', ',', 'M.', 'Charles-François-Bienvenu'
, 'Myriel', 'était', 'évêque', 'de', 'Digne.', 'C’était', 'un', 'vieillard', 'd’environ', 'so
ixante-quinze', 'ans', ';', 'il', 'occupait', 'le', 'siège', 'de', 'Digne', 'depuis', '1806.'
, 'Quoique', 'ce', 'détail', 'ne', 'touche', 'en', 'aucune', 'manière', 'au', 'fond', 'même',
'de', 'ce', 'que', 'nous', 'avons', 'à', 'raconter', ',', 'il', 'n’est', 'peut-être', 'pas',
'inutile', ',', 'ne', 'fût-ce', 'que', 'pour', 'être', 'exact', 'en', 'tout', ',', 'd’indique
```

Etape 2 : nettoyer les données

- stopwords (mots-outils) ou pas ? Le problème des SW est qu'ils peuvent fausser les résultats en donnant trop de poids à des mots anodins (mais on peut vouloir, justement, les analyser)

```
1 from nltk.corpus import stopwords
2 stop_words=set(stopwords.words("french"))
3 #texte_sans_SW=[]
4 #for t in tokens :
5 #    if t not in stop_words :
6 #        texte_sans_SW.append(t)
7 texte_sans_SW=[t for t in tokens if not t in stop_words]
```

Etape 3 : raciniser et lemmatiser

- un raciniseur est plus couramment appelé un stemmer : il s'agit de retrouver la forme minimale d'un mot, avec son ***radical***. Cela peut être utile si on cherche les mots qui commencent pareil mais n'ont pas la même nature

```
1 from nltk.stem.snowball import FrenchStemmer
2 stemmed_text=[mot for mot in texte_sans_SW]
```

- un lemmatiseur sert à retrouver la ***forme canonique*** d'un mot, dépouillé de ses flexions.

Deux outils principaux (même s'il y en a d'autres pour NLTK)

- Treetagger :

```
import treetaggerwrapper
# Construction et configuration du wrapper
tagger = treetaggerwrapper.TreeTagger(TAGLANG='fr',TAGDIR='/tmp',
  TAGINENC='utf-8',TAGOUTENC='utf-8')
for tok in texte_sans_SW:
    stringATagger+=tok+' '
tags = tagger.TagText(stringATagger)
print tags
```

- StanfordTagger

```
from nltk.tag import StanfordPOSTagger
jar = 'stanford/stanford-postagger-3.7.0.jar'
model = 'models/french.tagger'
pos_tagger = StanfordPOSTagger(model, jar, encoding='utf8')
pos_tagger.tag(stringATagger.split())
```

Etape à venir : le chunking