Analyser du texte avec python

Manipuler des chaines de caractères

Introduction

Les données externe à un programme, dés l'or que l'on s'intéresse à leurs traitement avec un programme python, on est contraint de manipuler régulièrement des chaînes de caractères.

Il est bon de voir quel outils nous avons à disposition pour traiter ces données textuelles.

<u>Au programme de ce cours :</u>

- Conversion de chaines
- Rechercher et remplacer
- Expression rationnelles
- Exercice apéritifs
- Projet 1 : un moteur de recherche
- Projet 2 : appli de réservation de salle

Conversion de chaines

Les premiers outils d'analyse sur les strings sont ceux qui nous permettent de revenir à des structures peut-être plus facilement manipulable, **les listes**.

- La liste l1 est construite tout simplement en énumérant les éléments de la chaîne et en utilisant cette énumération dans les élément de la liste.

- La liste l2 quant à elle introduit une nouvelle méthode **split()** dont le rôle est de diviser la chaîne en une liste de sous-chaîne, délimité par un séparateur passé en argument.

- En absence d'arguments, comme pour la liste l3, python utilise l'espace comme séparateur.

```
s1 = "Du passe faisons table rase"
l1 = [i for i in s1]
l2 = s1.split("as")
l3 = s1.split()
```

```
print(l1[0], l1[1], l1[2], l1[3],)
print(l2)
print(l3)
```

```
D u   p
['Du p', 'se faisons table r', 'e']
['Du', 'passe', 'faisons', 'table', 'rase']
```

Conversion de chaines

Les opérations réciproque qui permettent e récupérer une chaîne depuis une liste sont assez similaire.

- La méthode qui permet de combiner les chaînes d'une liste en en une unique chaîne s'appelle **join()**.
- Le séparateur est placé à gauche tandis que la liste est passé en argument.
- Il s'agit d'une méthode sur les chaînes de caractères à la quel on passe une liste en argument.

```
l1 = ['H', 'e', 'l', 'l', 'o', ' ', 'W']
l2 = ['Du', 'passe', 'faisons', 'table', 'rase']
s1 = "".join(l1)
s2 = " ".join(l2)
```

```
print(s1)
print(s2)
```

```
Hello W
Du passe faisons table rase
```

Rechercher et remplacer

L'extraction de sous-chaine peut très bien être opéré grâce au **slicing**.

```
s1 = "Faisons table rase"
s2 = s1[:8].lower() + s1[8:13].upper()
print(s2)
```

```
faisons TABLE
```

- La méthode upper() transforme les lettres en majuscules
- La méthode() les passe en minuscules.

Rechercher et remplacer

Le plus intéressant est d'extraire une sous-chaîne par rapport à son contenus :

```
s1 = "Ce texte contient deux fois le mot texte."
print(s1.find('texte', 0))
```

```
3
```

```
print(s1.find('texte', 4))
```

```
35
```

```
print(s1.find('texte', 36))
```

```
-1
```

```
s1 = "Ce texte contient deux fois le mot texte."
max = 0
while max > -1:
    max = s1.find('texte', max+1)
    print(max)
```

```
3
35
-1
```

- Ici on a utiliser la méthode **find(x, début, fin)** qui permet de rechercher un terme x dans la chaîne à la quel on applique cette méthode.
- On spécifiant l'index de début et l'index de fin qui définisse une plage de recherche dans cette chaîne.

Rechercher et remplacer

L'intérêt de récupérer l'indice est d'effectuer une substitution ou d'éliminer des sous chaînes indésirables.

Exemple :

- A deux reprise on localise un bloc de texte qui nous intéresse : 'euros' et 'deux'.
- On utilise l'indice récupéré pour remplacé ces termes via une combinaison de slicing.

```
s1 = "Le prix est de deux euros."
if 'euros' in s1:
    i = s1.index('euros')
    s2 = s1[:i-5] + '$' + s1[i-5:i]
    j  = s2.find('deux')
    s3 = s2[:j] + '2.15' + s2[j+4:]
    print(s3)
```

```
Le prix est de $2.15
```

Rechercher et remplacer

On peut obtenir le même résultat en utilisant la méthode replace().

<u>Exemple :</u>

Dans cette exemple on obtient les deux substitution simultanément.

On à un code plus concis, mais une souplesse d'utilisation moindre.

```
s1 = 'Le prix est de deux euros'
s2 = s1.replace('deux_euros', '$2.15')
print(s2)
```

```
Le prix est de deux euros
```

Rechercher et remplacer

On peut aussi chercher à détecter des sous-chaîne vérifiant certaines sous-caractéristiques.

Exemple :

Ici on utilise trois méthodes successives :

- isalpha() qui teste si une chaîne est composé uniquement de lettres.
- isnumeric() qui teste si une chaîne est composé uniquement de chiffres.
- isalnum() qui teste si une chaîne est composé d'un mélange des deux.

```
s = '300 seulement ?'
l = s.split()
for mot in l:
    if mot.isalpha():
        print('mot')
    if mot.isnumeric():
        print('nombre')
    if not mot.isalnum():
        print('?')
```

```
nombre
mot
?
```

Expressions rationnelles

Nous allons maintenant introduire un outil utilisé par tout les langage qui on vocation à traiter de l'information textuelle tel que Perl ou PHP.

<u>Rechercher :</u>

L'idée fondamental est d'identifier, non pas un bloc spécifique ou un type de caractères, mais un motif élaborer pouvant combiner des informations :

- de type
- d'alphabet
- de positionnement
- de répétition.

Commençons avec quelque exemples.

Expressions rationnelles

Exemple :

Ici on utilise la fonction findall() de la librairie « re » qui prend pour premier argument le motif à recherché et pour second argument la chaîne à parcourir. Elle renvoie la liste des éléments correspondant au motif recherché.

- La ligne 3 identifie les mots commençant par exactement une majuscule.
- La ligne 4 énumère ceux contenant au moins un l majuscule ou i minuscule.

```
from re import findall
s = "Le PIB de l'Argentine baisse depuis 3 ans"
l1 = findall('[A-Z][a-z]+', s)
l2  = findall('[a-zA-Z]*[iI][a-zA-Z]*',s)
print(l1)
print(l2)
```

```
['Le', 'Argentine']
['PIB', 'Argentine', 'baisse', 'depuis']
```

Expressions rationnelles

Ligne 3 :

- [A-Z] signifie un caractère dont le code ascii est compris entre celui de A et celui de Z et donc qui est une majuscule.
- [a-z] signifie un caractère dont le code ascii est compris entre celui de a et celui de z et donc qui est une minuscule.
- Le + signifie que le motif qui le précède peut être répété plusieurs fois.

La combinaison **‘[A-Z][a-z]+’** signifie donc « **une majuscule suivit d’une ou plusieurs minuscules** ».

Expressions rationnelles

Ligne 4 :

- [A-Za-z] signifie une majuscule ou une minuscule.
- [iI] signifie un I majuscule ou un i minuscule.
- L'étoile * implique que le motif qui la précède peut être présent une fois, plusieurs fois ou pas du tout.

La combinaison **'[A-Za-z]*[iI][A-Za-z]*'** signifie donc **« un nombre quelconque de lettres, puis un i, puis un nombre quelconque de lettres, indépendamment de la casse d'aucun des caractères »**.

Expressions rationnelles

Quelques éléments de base pour la fonction findall() :

Symbole	Signification
.	N'importe quel caractère
[x,y]	X ou y
[x-y]	Un caractère compris entre x ou y
[^x]	N'importe quel caractère sauf x
\s	Un caractère blanc (espace ou tabulation)
^	Début de ligne
$	Fin de ligne
xy	X puis y
{x}	Répèter x fois
{x,y}	Répèter entre x et y fois

Expressions rationnelles

Méta- caractères importants :

- Le backslash(\) pour échapper les caractères spéciaux.
- La combinaison (?:) qui sert à délimiter des blocs en particulier pour faire porter des quantificateurs. Par exemple : (?:abc){1, 3} signifie que abc peut être présent 1 ou 3 fois successives.
- Raccourcis pratiques :

Symbole	équivalent	signification
?	{,1}	Répéter 0 ou 1 fois
+	{1,}	Répéter au moins 1 fois
*	{0,}	Répéter un nombre quelconque de fois

Expressions rationnelles

Exemple :

Nous avons là une expression qui repère tout les numéros de téléphone français dans un bloc de texte.

```
from re import findall
motif = '0[1-9](?:[\s\.]?[0-9]{2}){4}'
n1 = "0678828383"
n2 = "09.34.67.12.11"
n3 = "03 11 23 20 38,"
n4 = "03 11 23 20,"
n5 = "03.11 23 2038,"
n6 = "03-23-20-20-38"
s = n1+n2+n3+n4+n5+n6
print(findall(motif, s))
```

```
['0678828383', '09.34.67.12.11', '03 11 23 20 38', '03.11 23 2038']
```

Expressions rationnelles

<u>Remplacement :</u>

La fonction de recherche est utile pour :

- Extraire de l’information d’un texte
- Effectuer des remplacements automatiques.
- Traduire du texte
- Chiffrer du texte
- Désactiver du contenu sensible (blocs de code)

Expressions rationnelles

- Ici on introduit la fonction sub() qui prend trois arguments :
  - Un motif de recherche
  - Une valeur de remplacement
  - La chaîne dans la quel rechercher

```
from re import sub
s = 'Un texte <strong>HTML<strong/>avec des balises'
s += ' et même<script type="text/javascript">'
s += 'var i = 5 ;</script> du javascript dedans.'
s1 = sub('<[a-z]*>', '', s)
print(s1)
```

```
Un texte HTML<strong/>avec des balises et même<script typ
e="text/javascript">var i = 5 ;</script> du javascript de
dans.
```

Expressions rationnelles

Amélioration du code précédant :
On ajoute la détection de caractère spéciaux entre les balises afin de prendre en compte les balises fermantes.

```
from re import sub
s = 'Un texte <strong>HTML<strong/> avec des balises'
s += ' et même<script type="text/javascript">'
s += ' var i = 5 ;</script> du javascript dedans.'
s1 = sub('<.*>', '', s)
s2 = sub('<[a-z\\/"=\s]*>', '', s)
s3 = sub('<[^>]*>', '', s)
print(s1)
print(s2)
print(s3)
```

```
Un texte  du javascript dedans.
Un texte HTML avec des balises et même var i = 5 ; du javascript dedans.
Un texte HTML avec des balises et même var i = 5 ; du javascript dedans.
```

Expressions rationnelles

- La première expression trouve la plus grande solution possible correspondant au motif, et va donc partir du premier chevron ouvrant et s'arrêter au dernier chevron fermant, englobant presque tout notre texte.
- La second fonctionne mieux mais nous oblige à déclarer de façon extensive tout ce qu'on ne garde pas, au risque d'oublier certains caractères.
- La troisième nous dit simplement de nous arrêter au premier chevron fermant.

Exercice :

1. Ecrire une fonction hascap(s) qui renvoie tout les mots de la chaîne s commençant par une majuscule.
   - Pour ce faire utiliser la fonction ord() pour obtenir le code ASCII des lettres (Les lettres majuscule ont un code allant de 65 à 90).
2. Proposer une fonction inflation(s) qui va doubler la valeur de tout les nombre dans la chaine s. Exemple : « Le prix est de 27 euros » devient « Le prix est de 54 euros ».
   - Utiliser la fonction enumerate() pour lancer une boucle for (Taper dans Google « enumerate boucle for ».)

Exercice :

3. Proposer une fonction lignes qui à partir d'une long chaîne s (>100 caractères) renvoie une liste de chaîne de caractères contenant chacun 24 caractères maximum et terminant par un espace.

   Voici un exemple de chaîne sur le quel vous pouvez travailler :

   - s = "Onze ans déjà que cela passe vite Vous "
     s += "vous étiez servis simplement de vos armes la "
     s += "mort n'éblouit pas les yeux des partisans Vous «
     s += "aviez vos portraits sur les murs de nos villes«

Exercice :

4. Proposer un programme qui renvoie la liste de tout les nombres (y compris décimaux et négatifs) d'une chaîne de caractères.
   A tester sur la chaîne : « Les 2 maquereaux valent 6.50 euros ».

5. Proposer une fonction arrondi(s) qui dans la chaîne s troncature tout les nombre décimaux. On autorise les nombres négatifs.

   Pour ce faire, vous avez la possibilité d'utiliser :

   - des () pour désigner des blocs de données dans l'expression rationnelle.
   - pour remplacer chacun des blocs l'expression est r'\1_\2_'.