جامعة هواري بومدين
للعلوم والتكنولوجيا
U S T H B

Programmation PYTHON

Cours 5

Nassim ZELLAL

2020/2021

Expressions régulières - 1

Les expressions régulières

- Les expressions régulières (en anglais « regular expressions », dont la forme abrégée est regex) représentent une chaîne de caractères servant à décrire de manière abstraite des fragments de texte. Cette chaîne de caractères est appelée motif (en anglais « pattern »). Ce dernier est constitué de deux types de caractères : les caractères dits normaux et les métacaractères, qui respectent une syntaxe particulière.
- On utilise un patron (ou masque, en anglais pattern), afin de trouver une partie d'une chaîne de caractères correspondant à ce patron. Les regex sont présentes dans de nombreux langages de programmation.
- Python utilise les expressions régulières et les rend particulièrement adaptées au traitement des fichiers textuels.

La reconnaissance - search()

- import re
- if re.search("regex", "chaîne"):
- ……..
- if not re.search("regex", "chaîne"):
- ……..
- La méthode search() permet de rechercher une correspondance dans toute la chaîne de caractères. Cette méthode retourne « None » si aucune correspondance n'est trouvée.
- if re.search("le chat", " le chat"):
- print("ok")

La reconnaissance - match()

- import re
- if re.match("regex", "chaîne"):
- ……..
- if not re.match("regex", "chaîne"):
- ……..
- La méthode match() permet de rechercher une correspondance au début de la chaîne de caractères. Cette méthode retourne « None » si aucune correspondance n'est trouvée.
- if re.match("le chat", " le chat"):
- print("ok")

La reconnaissance - findall()

- import re
- if re.findall("regex", "chaîne"):
- ……..
- if not re.findall("regex", "chaîne"):
- ……..
- La méthode findall() renvoie les éléments qui correspondent (toutes les correspondances) dans une liste. Cette méthode retourne une liste vide si aucune correspondance n'est trouvée.
- y= "Je mange une pomme et une tomate"
- x = re.findall("om", y)
- print(x)

Les métacaractères - le point

- Le point . : n'importe quel caractère sauf le saut de ligne.

```
ligne="J'adore la programmation"
if re.search(".",ligne):
    print("ok")
```

Les métacaractères - l'accent circonflexe

- L'accent circonflexe ^ : début de ligne.

- ligne="http::"
- if re.search("^h",ligne):
- print("ok")

Les métacaractères - le dollar

- Le dollar **$** : fin de ligne.
- ligne="http::"
- if re.search("::**$**",ligne):
- print("ok")

Les métacaractères - le pipe

- Le pipe | : alternative.
- ligne="abc"
- if re.search("abc|def",ligne):
- print("ok")

Les métacaractères - les classes (1)

- Le crochets **[…]** : une **classe de caractères**, qui définit un **ensemble de caractères** acceptables en un point particulier de l'expression régulière.
- Une paire de **crochets** [...] contient l'ensemble des caractères acceptables.
- ligne="case et vase"
- x=re.findall("[cv]ase",ligne)
- print(x)

Les métacaractères - les classes (2)

- Le tiret **-** : [0**-**9] => **[0123456789]** (le tiret est un opérateur d'intervalle dans une classe).
- ligne="**2**5"
- if re.search("[0-9]",ligne):
- print("ok")
- ligne="**2**5"
- x=re.search("[1-5]",ligne)
- print(x)

Les métacaractères - les classes (3)

- L’accent circonflexe **^** : le complémentaire d’une classe.
- print(re.search("**[^**a**]**at",ligne))

- **bat**
- aat
- **cat**

Les métacaractères - les classes prédéfinies (1)

- [0-9] => **\d**
- [0-9a-zA-Z_] => **\w**
- [\t\n\r] => **\s**
- **\\bhomme\\b** : frontière de mot (sans préfixe).
- **r"\bhomme\b"** : frontière de mot (ajout du préfixe « r » ou « R » pour désigner une chaîne brute, voir slide 16).

Les métacaractères - les classes prédéfinies (2)

- [^0-9] => **\D**
- [^0-9a-zA-Z_] => **\W**
- [^ \t\n\r] => **\S**

Les métacaractères - les classes prédéfinies (3)

- import sys,re
- var=open(sys.argv[1],'r')
- for i in var:
- if re.search("\\bhomme\\b",i):

  #on peut ajouter le préfixe « r » et écrire (r"\bhomme\b",i):
- print(i,end="")
- var.close()
- ----------
- **>test6.py text-b.txt > result**

Les métacaractères - les classes prédéfinies - « result »

Ils regardaient l'homme debout sur le banc et écoutaient le texte malgré eux.
Dans la routine prodigieusement lassante de la salle d'attente, cet homme qui lisait à voix haute était un événement aussi tro
L'homme, de sa grosse voix de ténor aux accents faussement pathétiques, déversait ces mondes et ces passions dans le brouhaha
ou mornement assise.
Et l'homme qui lisait ainsi, sur un banc vert de la gare, ne leur annonçait rien qui fût indispensable à leur existence.
Lorsqu'elles virent l'homme en train de lire, elles échangèrent un regard et sentirent toutes deux qu'il s'agissait là d'une a
public.
L'homme semblait ne pas avoir entendu la question.
Elle n'avait pas l'intention d'écouter ce que l'homme lisait.
« Il y a un homme qui lit à haute voix dans la salle d'attente ! »
Dans le coin le plus reculé se tenait bel et bien un homme de grande taille, en train de lire un livre à haute voix, le visage
Dans les moments les plus calmes, ses paroles parvenaient même jusqu'aux oreilles du chef de gare.
« Il y a un homme qui lit un livre à haute voix dans la salle d'attente.
« Que faire lorsqu'un homme ne crache pas et ne se bat pas dans la salle d'attente d'une gare, mais lit un livre à haute voix
Après cette entrée en matière claire et directe, il dut fournir à l'homme de loi des explications longues et embrouillées.
Il pensa qu'il pourrait peut-être simplement aller voir l'homme et lui dire :
Au fond, il n'y avait peut-être rien de gênant dans le fait qu'un homme lise dans la salle d'attente, mais tout de même, il va
reste pas là.

Expressions régulières - quantificateurs (gourmands / greedy)

- ***** répète le caractère précédent 0 fois ou plus - gourmand.
- **?** répète le caractère précédent 0 fois ou 1 fois - gourmand.
- **+** répète le caractère précédent 1 fois ou plus - gourmand.
- Exemple : ligne="USTHB"
- if re.search(".",ligne) ➔ U
- if re.search(".+",ligne) ➔ USTHB

Expressions régulières - quantificateurs (non gourmands/non-greedy)

- * → *?
- + → +?
- ? → ??

Expressions régulières - quantificateurs (gourmands/greedy)

- import re
- ligne = 'Voila un <A HREF="index.html">index</A>'
- x=re.search("<**.+**>",ligne)
- print(x)

- > **<re.Match object; span=(9, 39), match='<A HREF="index.html">index</A>'>**

Expressions régulières - quantificateurs (gourmands/greedy)

Expressions régulières - quantificateurs (non gourmands/non-greedy)

- import re
- ligne = 'Voila un <A HREF="index.html">index</A>'
- x=re.search("<**.+?**>",ligne)
- print(x)

- **> <re.Match object; span=(9, 30), match='<A HREF="index.html">'>**

Expressions régulières - quantificateurs (non gourmands/non-greedy)

Expressions régulières - quantificateurs (gourmands/greedy)

- ligne = 'Je vois des voitures'
- x=re.search("voiture**.?**",ligne)
- print(x)

- > **<re.Match object; span=(12, 20), match='voiture*s*'>**

Expressions régulières - quantificateurs (non gourmands/non-greedy)

- ligne = 'Je vois des voitures'
- x=re.search("voiture**.??**",ligne)
- print(x)

- > **<re.Match object; span=(12, 19), match='voiture'>**

Expressions régulières - quantificateurs (intervalles de reconnaissance)

- a{1} répète le caractère précédent 1 fois : a
- a{1,} répète le caractère précédent au moins 1 fois : a, aa, aaa …
- a{2,3} répète le caractère précédent entre m et n fois : aa, aaa

Remarque :

*** == {0,}**

? == {0,1}

+ == {1,}

Exercice 1

- **Faire un script Python, qui demande à l'utilisateur de saisir l'un des mots suivants :**
- **défragmentation, reduplication et colocation.**
- **L'utilisateur doit aussi pouvoir quitter le script en tapant une lettre.**
- **Le script doit :**
- **Compter le nb de caractères de chaque mot.**
- **Séparer les affixes.**
- **Compter le nb de caractères des racines.**
- **Le script doit afficher les mots de la façon suivante :**
- **préfixe+racine+suffixe nb caractères mot, nb caractères racine**

Exercice 2

- **Intégrer dans ce script une expression régulière pour récupérer la racine du verbe, en utilisant un fichier contenant les terminaisons et les pronoms.**

- **import sys**
- **pr_tr = open(sys.argv[1],'r')**
- **print("Saisir un verbe du premier groupe :")**
- **tab=sys.stdin.readline().split("er")**
- **t=pr_tr.readlines()**
- **for i in range(6):**
- **print(t.pop().rstrip()+" "+tab[0]+t.pop(),"\n")**