LS4_200314

Expressions régulières (= rationelles)

Suite de caracteres qui définit un ensemble de mots. La plus part des symboles (caractères) ne représentent qu'eux memes. Les autres symboles sont appelés metacaractères ou caractères spéciaux.

Les caracteres spéciaux

- '.'représente n'importe quel caractère excepté le caractère retour à la ligne ('\n') *Exemple : '.a' : représente les suites de deux caractères, formées d'un caractère quelconque (excepté \n) suivi d'un 'a'.*
- '\d' qui représente les caractères numériques *Exemple : 'a\d' : {'a0','a1','a2',...,'a9'}*
- '\w' qui représente les caractères alphanumériques ('a'...'z','A'...'Z','0'...'9')
- '\s' qui représente les caractères d'espacements
- '\D', '\W', '\S' sont les complémentaires respectifs de '\d','\w' et '\s'
- '[ ]' : represente l'un des symboles situés entre les crochets Remarque : Entre [ ] les spécificités perte leur rôles, sauf '^' *Exemple : [abc] = {'a','b','c'} [\db] = {' \','d','b'}*

Exemple : On veut tous les mots formés de 'a' ou de 'b' : {a,b}* {a,b}* = [ab]*

- '[^]' : représente la négation d'un des symboles *Exemple : [^ab] : n'importe quel caractère qui soit ni 'a', ni 'b'*
- [a-t]: n'importe quel caractere de 'a' à 't' {'a','b',...,'t'}
- [a-t0-3]: n'importe quel caractere de 'a' à 't' ou de '0' à '3'
- ' | ' symbolise le 'ou' *Exemple : '\d | b' : {'O','1',...,'9','b'}*
- '$' signifie la fin de la chaîne *Exemple : 'a+$': cherche une chaine de caractère qui termine par n'importe quelle suite de a*
- '^' : signifie le début de chaine *Exemple : '^bb' : chaine qui commencent par 'bb'*
- '\b' : signifie le début ou la fin d'un mot *Exemple : '\bc\w\c\b'* : les mots commençant et terminant par c '\bc\w*c' : les mots commençant par c*

-Caracteres de répétiton

- '*' représente la répétition de **0** à "l'infini" du caractère ou du groupe qui le **précède** Exemple : 'a*' : {ø, 'a','aa','aaa',...}
- '+' représente la répétition de **1** à "l'inifini" du caractère ou du groupe qui le **précède**
- '?' répétition de 0 à 1 du caractère ou groupe qui le représente (absence ou présence du caractère qui le précède)
- {m,n} : placé après un caractère signifie que le caractère est répété au moins m fois et au plus n fois. Remarque : {0,} = correspond à '*' *{1,} = correspond à '+' {,3} = répétition au plus de 3 fois* Exemple : 'a{2,4} : {'aa','aaa','aaaa'}

Groupes

( ) : définit un groupe, sont numérotés dans l'ordre d'apparation (lecture gauche droite). Et peuvent être appelés par leur numéro précédé du symbole '\'. Exemple : '(\d)\w*\1' : chaînes qui comment par un chiffre c, suivi de n'importe quelle suite de caractères alphanumériques, suivi de c.

On peut échapper un caractère spécial en mettant un '\' devant.

Exemple : '\\d' = suitre de 2 caracteres : '\' puis 'd'

Les méthodes

Elles se trouvent dans le module 're'.

```
import re

re.match(expr_reg, chaine [,flag]) :
```

- si il existe un **préfixe** de la chaîne contenu dans l'expression reguliere passée en paramètre retourne objet de type expr_reg
- sinon return 'None'

Exemple :

```
>>>re.match('ba','banane')
retourne un objet car 'ba' prefixe de banane

>>>re.match('na','banane')
None

>>>re.match('ba*c+','baaacca')
le prefixe reteNU est 'baaacc'
```

Remarque : Les caractères de répétitions induisent un comportement 'glouton' Pour ne pas avoir

ce comportement il faut alors ajouter '?' à la suite du caractères de répétition

```
>>>re.match('ba*c+?','baaacc')
le préfixe reteNU est 'baaac'
```

Pour obtenir le préfixe retenu, on a la méthode group() qui s'applique à l'objet retourné par la méthode match

```
>>>o = re.match('ba*c+','baaacca')
>>>o.group()
'baaacc'
```

S'il y a des groupes dans l'expression :

```
>>> o = re.match('a(\d\d)[A-Z](\d\d)', 'a71C98ttt')
>>> o.groupe(0)
'a71C98'
>>> o.group(1)
71
>>> o.group(2)
98
```

Flags

re.IGNORECASE == re.I ignore la casse, ignore la casse du deuxieme parametre

```
>>>re.match('ab','Abc', re.I).group()
'Ab'
```

re.MULTILINE = re.M permet que le '^' signifie début de ligne et le '$' signifie lafin de ligne (utile pour le search())

re.DOTALL = re.s : permet que le '.' signifie n'importe quel caractère (y compris '\n')

re.search(expr_reg, chaine, [,flags]) : recherche la première sous chainemaximale qui est inclue dans l'expression réguliere. Retourne objet de type expression reguliere si succès, sinon None

raw

```
>>>re.match(r'a\n')
lis litteralement a \ n ne prend pas en compte le retour de ligne
```