Expressions rationnelles

Laurent Signac – CC-BY-SA – 01-02-22 1915 020a1d0ca27dfd81e3ce

Une expression rationnelle, ou expression régulière ou motif (en anglais *regular expression* ou *regex*) est une chaîne de caractères qui décrit, grâce à une syntaxe particulière un ensemble de chaînes possibles.

Par exemple `a..` désigne toutes les chaînes de 3 caractères exactement dont la première est un `a`.

On rencontre les expressions rationnelles dans de nombreux contextes : langages de programmation, mais aussi outils de traitement de texte.

Python propose le module `re`, dans sa bibliothèque standard, pour manipuler les expressions rationnelles. En règle générale, les fonctions de ce module prennent un motif (éventuellement précompilé) et une chaîne de caractères, dans laquelle on recherche le motif.

1 Principe des regex

Une regex permet donc de décrire dans un langages particulier un ensemble de chaînes de caractères. Par exemple :

- les chaînes qui commencent par `A` : `A.*`
- les chaîne qui commencent par `A`, suivi d'un certain nombre de `B` (éventuellement 0), puis un nouveau `A` : `AB*A`
- les chaînes d'au moins deux lettres, qui commencent par un `a` se terminent par un `b` et contiennent n'importe quel caractère sauf un `c` entre les deux extrémités : `a[^c]*b`
- …

Les caractères spéciaux les plus utiles sont probablement :

- `.` remplace n'importe quel caractère
- `*` indique que le caractère précédent peut être répété plusieurs fois (éventuellement 0)
- `+` indique que le caractère précédent peut être répété plusieurs fois (mais au moins 1)
- `$` indique une fin de ligne
- `^` indique un début de ligne

La syntaxe complète serait (très) longue à décrire. Voici un point de départ pour une vue plutôt synthétique : Regex Quick Start[1]

Les regex sont disponibles dans la plupart des langages (dont Python avec le module `re`) mais aussi dans les logiciels d'édition de texte (chercher/remplacer).

- L'outil en ligne https://regexr.com/ permet de tester des regex.
- L'outil en ligne https://ihateregex.io contient de nombreux exemples.

2 Utilisation avec Python

- `re.finditer(<motif>, <string>)` itère sur toutes les occurrences du motif dans la chaîne (renvoie des objets de type `re.Match`)
- `re.findall(<motif>, <string>)` renvoie la liste des chaînes trouvées
- `re.match(<motif>, <string>)` renvoie un objet de type `re.Match` si `<motif>` est présent au **début** de `<string>` et `None` sinon
- `re.search(<motif>, <string>)` renvoie un objet de type `re.Match` si `<motif>` est présent quelque part dans `<string>` et `None` sinon
- `re.sub(<motif>, <replacement>, <string>)` renvoie une chaîne dans laquelle les occurrences de `<motif>` dans `<string>` ont été remplacées par `<replacement>`.

[1] https://www.rexegg.com/regex-quickstart.html

Il existe d'autres fonctions dans ce module, et la plupart acceptent d'autres paramètres (des flags) permettant de modifier leur comportement. Pour avoir une vision exhaustive des possibilités, il faut se reporter à la page de documentation du module `re`[2]

Lorsqu'on a des besoins particuliers en performance, il faut songer à utiliser des motifs qu'on aura *précompilés* avec `re.compile(<motif>)`.

3 Exemples d'utilisation des fonctions

```
>>> for x in re.finditer("o.", "sorbet citron"):
        print(x)

<re.Match object; span=(1, 3), match='or'>,
<re.Match object; span=(11, 13), match='on'>

>>> re.findall("o.", "sorbet citron")
['or', 'on']

>>> re.match("sor", "sorbet citron")
<re.Match object; span=(0, 3), match='sor'>

>>> re.match("cit", "sorbet citron")

>>> re.search("cit", "sorbet citron")
<re.Match object; span=(7, 10), match='cit'>

>>>re.sub("cit", "poti", "sorbet citron")
'sorbet potiron'

>>>re.sub("o", "ou", "sorbet citron")
'sourbet citroun'
```

Par défaut les opérateurs `+` et `*` sont dits *gloutons* (*greedy* en anglais). C'est à dire que le motif `.*` va correspondre à la *plus longue chaîne possible* :

```
>>> re.findall("<.*>", "<3 + 5> * <4 - 2>")
['<3 + 5> * <4 - 2>']
```

On peut contraindre `+` et `*` à ne pas être gloutons, de manière à ce qu'ils correspondent à la *plus courte chaîne possible*, en les suffixant par `?` :

```
>> re.findall("<.*?>", "<3 + 5> * <4 - 2>")
['<3 + 5>', '<4 - 2>']
```

4 Exemples d'expressions rationnelles

```
import re
texte = """\
Les sanglots longs des violons de l'automne
blessent mon cœur d'une langueur monotone.
Tout suffocant et blême, quand sonne l'heure,
je me souviens des jours anciens et je pleure
"""
```

[2] https://docs.python.org/fr/3/library/re.html

■ Capturer tous les `o` :

```
for r in re.finditer("o", texte):
    print(r, "==>", r.group(0))
```

```
<re.Match object; span=(9, 10), match='o'> ==> o
<re.Match object; span=(14, 15), match='o'> ==> o
<re.Match object; span=(25, 26), match='o'> ==> o
<re.Match object; span=(27, 28), match='o'> ==> o
<re.Match object; span=(39, 40), match='o'> ==> o
<re.Match object; span=(55, 56), match='o'> ==> o
<re.Match object; span=(79, 80), match='o'> ==> o
<re.Match object; span=(81, 82), match='o'> ==> o
<re.Match object; span=(83, 84), match='o'> ==> o
<re.Match object; span=(90, 91), match='o'> ==> o
<re.Match object; span=(98, 99), match='o'> ==> o
<re.Match object; span=(121, 122), match='o'> ==> o
<re.Match object; span=(143, 144), match='o'> ==> o
<re.Match object; span=(156, 157), match='o'> ==> o
```

■ Capturer tous les `o` suivis de 3 caractères :

```
for r in re.finditer("o...", texte):
    print(r, "==>", r.group(0))
```

```
<re.Match object; span=(9, 13), match='ots '> ==> ots
<re.Match object; span=(14, 18), match='ongs'> ==> ongs
<re.Match object; span=(25, 29), match='olon'> ==> olon
<re.Match object; span=(39, 43), match='omne'> ==> omne
<re.Match object; span=(55, 59), match='on c'> ==> on c
<re.Match object; span=(79, 83), match='onot'> ==> onot
<re.Match object; span=(83, 87), match='one.'> ==> one.
<re.Match object; span=(90, 94), match='out '> ==> out
<re.Match object; span=(98, 102), match='ocan'> ==> ocan
<re.Match object; span=(121, 125), match='onne'> ==> onne
<re.Match object; span=(143, 147), match='ouvi'> ==> ouvi
<re.Match object; span=(156, 160), match='ours'> ==> ours
```

■ Capturer toutes les lignes contenant un `o` :

```
# re.MULTILINE est nécessaire pour que ^ matche un début de ligne
# et non juste le début de la chaîne.
for r in re.finditer("^.*o.*$", texte, re.MULTILINE):
    print(r.group(0))
```

```
Les sanglots longs des violons de l'automne
blessent mon cœur d'une langueur monotone.
Tout suffocant et blême, quand sonne l'heure,
je me souviens des jours anciens et je pleure
```

■ Capturer les chaînes composées d'un `o`, puis d'au moins 1 caractère, jusqu'à un `n` :

```
# Notez que c'est la capture la plus longue possible qui est faite
for r in re.finditer("o.+n", texte):
    print(r.group(0))
```

```
ots longs des violons de l'automn
on cœur d'une langueur monoton
out suffocant et blême, quand sonn
ouviens des jours ancien
```

■ Capturer les chaînes composées d'un `o`, puis d'une série de lettres (sauf un `a` ou un passage à la ligne), puis d'un `n` :

Version gloutonne (la chaîne la plus longue est capturée, c'est le comportement par défaut) :

```
for r in re.finditer("o[^a\n]*n", texte):
    print(r.group(0))
```

```
ots longs des violon
omn
on cœur d'un
onoton
onn
ouvien
```

Version **non** gloutonne (la chaîne la plus courte est capturée) :

```
for r in re.finditer("o[^a\n]*?n", texte):
    print(r.group(0))
```

```
ots lon
olon
omn
on
on
oton
on
ouvien
```

■ Capturer les chaînes composées d'un `o`, puis d'une série d'au moins 3 lettres, puis un `a`. N'afficher **que** ce qui est entre le `o` et le `a`

```
# On peut créer des groupes dans la chaîne capturée et les extraire
for r in re.finditer("o(....*)a", texte):
    print(r.group(0), "/", r.group(1))
```

```
ots longs des violons de l'a / ts longs des violons de l'
on cœur d'une la / n cœur d'une l
out suffocant et blême, qua / ut suffocant et blême, qu
ouviens des jours a / uviens des jours
```

■ Utilisation de `findall`

La fonction `re.findall` permet de renvoyer une liste contenant toutes les chaînes capturées. Le motif complet est renvoyé s'il n'y a pas de de groupe. S'il y a des groupes, ce sont des tuples, formés par ces groupes qui sont renvoyés.

```
texte2 = "ABCDEFGAEIOU"

print(re.findall(".E.", texte2))
['DEF', 'AEI']

print(re.findall(".(E.)", texte2))
['EF', 'EI']

print(re.findall("(.)E(.)", texte2))
[('D', 'F'), ('A', 'I')]
```

5 Exemple d'utilisation

Certaines pages du site *allociné* référencent les dernières séries Netflix. On souhaite récupérer automatiquement les titres des séries sur les 5 premières pages.

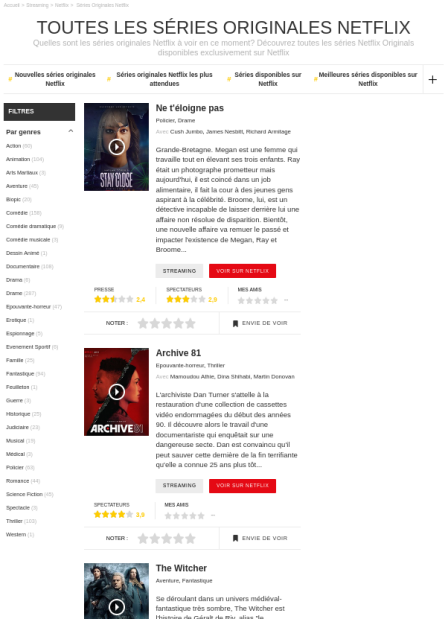

Figure 1: Exemple de page allociné

Ce type de page (ainsi que le code source de la page) est visible en suivant ce lien :
https://www.allocine.fr/netflix/series/?page=1

Le code ci-dessous consulte les pages 1 à 5, extrait les titres (on se rend compte en observant le source de la page que tous les titres sont dans des balises hyperlien, ayant comme attribut `class="meta-title-link"`), et en construit la liste avant de l'afficher.

```
import re
import requests
resultats = []
for i in range(1, 6):
    url = f"https://www.allocine.fr/netflix/series/?page={i}"
    r = requests.get(url)
    for a in re.finditer('<a class="meta-title-link.*>(.*)</a>', r.text, re.M):
        resultats.append(a.group(1))

for k, titre in enumerate(resultats, 1):
    print(f"{k:02d} : {titre}")
```

Et nous obtenons (février 2022) :

```
01 : Ne t&#039;éloigne pas
02 : Archive 81
03 : The Witcher
04 : Ozark
05 : Emily in Paris
06 : Kitz
07 : The Silent Sea
08 : Plan coeur
09 : Arcane
10 : La Femme qui habitait en face de la fille à la fenêtre
11 : After Life
12 : La Casa de Papel
...
```

6 Liens

- Regular Expressions for Regular Folk[3]

[3] https://refrf.shreyasminocha.me/