

B216A Développement informatique

# Expressions régulières

*Sébastien Combéfis, Quentin Lurkin*



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

# Validation de données

- Important de **valider** les inputs

*Données entrées par le user, provenant de fichiers, du réseau...*

- **Formatage** des données

*Via un parser qui produit une erreur en cas de format invalide*

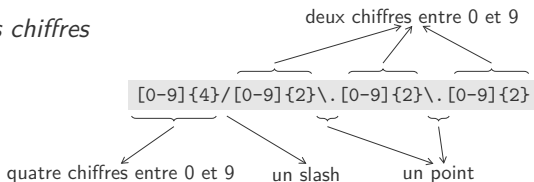
- Vérifier les données avec une **expression régulière** (module `re`)

*La donnée suit-elle un motif prédéfini ?*

# Exemple : Numéro de téléphone

- **Numéro de téléphone** de la forme `xxxx/xx.xx.xx`

Où les *x* sont des chiffres



```
1 pattern = r'[0-9]{4}/[0-9]{2}\.[0-9]{2}\.[0-9]{2}'
2 p = re.compile(pattern)
3
4 print(p.match('0394/83.31.41'))
5 print(p.match('0394/83-31-41'))
```

```
<_sre.SRE_Match object; span=(0, 13), match='0394/83.31.41'>
None
```

# Vérifier une chaîne (1)

- Définition d'un **motif** représentant les chaînes valides

*En utilisant une expression régulière*

- **Compilation** du motif avec la fonction `re.compile`

*Renvoie un objet de type `regex`*

- **Vérification** d'une chaîne de caractères avec la méthode `match`

*Renvoie `None` si invalide, et un objet décrivant le match sinon*

# Description de motif (1)

- Précéder les **méta-caractères** avec un backslash

`. ^ $ * + ? { } [ ] \ | ( )`

- **Classes de caractères** définies avec les `[ ]`

`[abc]` : a, b ou c

`[0-9]` : n'importe quoi entre 0 et 9

`[a-zA-Z]` : n'importe quoi entre a et z ou entre A et Z

`[^aeiou]` : n'importe quoi sauf a, e, i, o ou u

`[a-z&&[^b]]` : intersection entre deux ensembles

- **Classes prédéfinies** de caractères

`.` : n'importe quel caractère (sauf retour à la ligne)

`\d` : un chiffre (équivalent à `[0-9]`)

`\s` : un caractère blanc (équivalent à `[ \t\n\r\f\v]`)

`\w` : un caractère alpha-numérique (équivalent à `[a-zA-Z0-9_]`)

# Description de motif (2)

## ■ Répétitions d'occurrences avec un quantificateur

$\{n\}$  : exactement  $n$  occurrences

$\{m,n\}$  : au moins  $m$  et au plus  $n$  occurrences

$\{m,\}$  : au moins  $m$  occurrences

$\{,n\}$  : au plus  $n$  occurrences

## ■ Quantificateurs prédéfinis

$? = \{0,1\}$

$* = \{0,\}$

$+ = \{1,\}$

## ■ Frontières de recherche

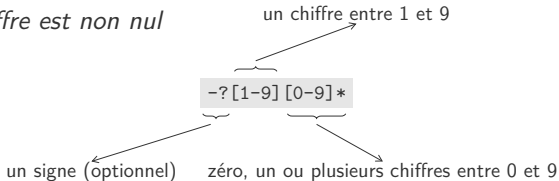
$\wedge$  : début de la ligne

$\$$  : fin de la ligne

# Exemple : Nombre entier

- **Nombre entier** avec éventuellement un signe —

*Le premier chiffre est non nul*



```
1 pattern = r'-?[1-9][0-9]*'
2 p = re.compile(pattern)
3
4 print(p.match('15') is not None)           # True
5 print(p.match('03') is not None)           # False
6 print(p.match('-7') is not None)            # True
7 print(p.match('-42') is not None)           # True
8
9 print(p.match('8 enfants !') is not None)   # True
```



# Vérifier une chaîne (2)

- Vérifier le contenu d'une chaîne avec les frontières

*La méthode `match` cherche à partir du début de la chaîne*

- `^` matche le début de la chaîne et `$` la fin

*S'assure qu'une chaîne complète suit un motif*

```
1 pattern = r'-?[1-9][0-9]*'  
2 p = re.compile(pattern)  
3  
4 print(p.match('8 enfants !') is not None)    # False
```

# Options de compilation

- **Flags** à passer à la méthode `compile`

*Constantes définies dans le module `re`, à combiner avec `/`*

Flag	Description
<code>re.IGNORECASE</code>	Recherche insensible à la casse
<code>re.DOTALL</code>	Le point matche également les retours à la ligne
<code>re.MULTILINE</code>	Les matches peuvent se faire sur plusieurs lignes

```
1 pattern = r'^[a-z]+$'
2 p = re.compile(pattern, re.IGNORECASE)
3
4 print(p.match('bogoss') is not None) # True
5 print(p.match('LaTeX') is not None) # True
```

# La peste du backslash

- Utilisation recommandée des chaînes brutes

*Sinon, de nombreux soucis avec le backslash*

- Nom de commande L<sup>A</sup>T<sub>E</sub>X

Regex	Description
<code>\[a-zA-Z]+</code>	Nom d'une commande
<code>\\[a-zA-Z]+</code>	Échappement du <code>\</code> pour compile
<code>\\\\[a-zA-Z]+</code>	Construction d'un littéral <code>str</code>

```
1 p = re.compile('\\\\([a-zA-Z]+)')      # re.compile(r'\\([a-zA-Z]+)')
2 m = p.match('\\LaTeX')
3 if m is not None:
4     print(m.group(1))
```

# Rechercher un motif (1)

- La méthode `match` renvoie un **objet du match**

*Plusieurs méthodes pour obtenir des informations sur le match*

Méthode	Description
<code>group</code>	Renvoie la sous-chaine matchée
<code>start</code>	Renvoie la position du début du match
<code>end</code>	Renvoie la position de la fin du match
<code>span</code>	Renvoie un tuple avec les positions de début et de fin du match

```
1 pattern = r'\d+'
2 p = re.compile(pattern)
3
4 m = p.match('72 est égal à 70 + 2')
5 if m is not None:
6     print(m.group())      # 72
```

## Rechercher un motif (2)

- **search** cherche un motif dans une chaîne

*Recherche d'une sous-chaîne qui matche le motif*

- **findall** recherche toutes les sous-chaînes qui matchent

*Renvoie une liste des sous-chaînes matchées*

```
1 pattern = r'\d+'
2 p = re.compile(pattern)
3
4 m = p.search('on sait que 72 est égal à 70 + 2')
5 if m is not None:
6     print(m.group())      # 72
7
8 print(p.findall('72 est égal à 70 + 2'))  # ['72', '70', '2']
```

# Rechercher un motif (3)

- `finditer` renvoie un itérateur d'objets de match

*On peut ainsi parcourir les matches avec une boucle `for`*

```
1 pattern = r'\d+'
2 p = re.compile(pattern)
3
4 for m in p.finditer('on sait que 72 est égal à 70 + 2'):
5     print(m.span(), ': ', m.group())
```

```
(12, 14) : 72
(26, 28) : 70
(31, 32) : 2
```

- **Alternative** entre plusieurs motifs avec |

*Représente l'opérateur « ou »*

```
1 pattern = r'^[0-9]+|[a-z]+$'  
2 p = re.compile(pattern)  
3  
4 print(p.match('283') is not None)  
5 print(p.match('boat') is not None)  
6 print(p.match('boat283') is not None)
```

```
True  
True  
False
```

# Capturer des groupes

- Disséquer une chaîne en sous-chaînes matchantes

*Pouvoir extraire plusieurs parties d'une chaîne donnée*

- Définition de groupes dans le motif avec des ( )

*Chaque groupe représente un sous-motif*

```
1 pattern = r'^([a-z]+)@([a-z]+)\.([a-z]{2,3})$'
2 p = re.compile(pattern)
3
4 m = p.match('email@example.net')
5 if m is not None:
6     print(m.groups())      # ('email', 'example', 'net')
7     print(m.group(1))      # email
```



# Option de capture des groupes

- Option avec **point d'interrogation** après la parenthèse ouvrante

*Ce qui suit le ? décrit l'option*

- Option **spécifique à Python** avec (?P...)

*Indique une option pas présente dans la syntaxe PERL*

- **Groupe non capturant** avec l'option :

*Permet d'appliquer un quantificateur sur un sous-motif*

```
1 pattern = r'^(?:[0-9]+)$'
2 p = re.compile(pattern)
3
4 m = p.match('283')
5 if m is not None:
6     print(m.groups())          # ()
```

# Groupe nommé

- Possibilité de **nommer un groupe** avec les options Python

*Le groupe est récupéré avec son nom plutôt que sa position*

```
1 pattern = r'^(?P<pseudo>[a-z]+)@(?P<domain>[a-z]+)\.(?P<extension>[a-z]{2,3})$'
2 p = re.compile(pattern)
3
4 m = p.match('email@example.net')
5 if m is not None:
6     print(m.groups())
7     print(m.group(1))
8     print(m.group('domain'))
```

```
('email', 'example', 'net')
email
example
```

# Exemple : Extraction de liens (1)

## ■ Extraction des adresses de liens dans une page HTML

*Retrouver la valeur de l'attribut href des éléments a*

*Utilisation du motif <a.\*href="(.\*)."\*>*

```
1 pattern = r'<a.*href="(.*)."*>'
2 p = re.compile(pattern, re.MULTILINE)
3
4 for m in p.finditer(''<a href="http://www.google.be" rel="popup"
5 class="extlink">
6 <a target="_blank" href="http://www.yahoo.be" class="extlink" rel="
7 popup">
8 <a href="http://www.alltheweb.com">''):
```

```
http://www.google.be" rel="popup" class="extlink
http://www.yahoo.be" class="extlink" rel="popup
http://www.alltheweb.com
```

## Exemple : Extraction de liens (2)

- Solution 1 : Ne pas capturer " dans l'URL

*Utilisation du motif `<a.*href="([~"]*)"*.*>`*

- Solution 2 : Utilisation des quantificateurs non greedy

*Capturent le moins possible (ajout d'un ? derrière)*

```
1 pattern = r'<a.*href="(.*)"*.*>'
2 p = re.compile(pattern, re.MULTILINE)
3
4 # ...
```

```
http://www.google.be
http://www.yahoo.be
http://www.alltheweb.com
```

# Référence en arrière

- On peut **faire référence** à un groupe précédemment capturé

*\ suivi du numéro du groupe*

- Permet par exemple la recherche de **répétitions** de mot

*Capture d'un mot suivi de blancs (\s) et du même mot*

```
1 p = re.compile(r'([a-z]+)\s+\1')
2 m = p.search('Il a une une pomme')
3 if m is not None:
4     print(m.group())
```

une une

# Découpe de chaîne

- La méthode `split` **découpe** une chaîne selon un motif

*Permet d'avoir une expression régulière comme séparateur*

```
1 pattern = r'[ ,;.?!-]+'
2 p = re.compile(pattern)
3
4 words = p.split('Bonjour ! Comment allez-vous ?')
5 print(words)
```

```
['Bonjour', 'Comment', 'allez', 'vous', '']
```

- **RegexLib**, <http://www.regexlib.com/>

*Recueil d'expressions régulières*

- **Regex 101**, <https://regex101.com/>

*Testeur en ligne d'expressions régulières*

- **Regex Crossword**, <https://regexcrossword.com/>

*Jeu en ligne pour apprendre les expressions régulières*

# Crédits

- <https://www.flickr.com/photos/b-tal/151881566>
- <https://www.flickr.com/photos/strathmeyer/7200873702>