

PI2T Développement informatique

## Séance 3

# Chaines de caractères et expressions régulières

*Sébastien Combéfis, Quentin Lurkin*

*22 février 2016*



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.

# Objectifs

- Manipulation de chaînes de caractères
  - La classe `str`
  - Fonctions dans la librairie standard
  - Optimisations pour la manipulation de chaînes
- Utilisation des expressions régulières
  - Vérifier qu'une chaîne de caractères matche un motif
  - Chercher des sous-chaînes correspondant à un motif
  - Extraire des sous-chaînes correspondant à un motif



Chaine de caractères

# Séquence de caractères

- Une **chaîne de caractères** est une séquence non modifiable

*Séquence de caractères Unicode*

- **Opérations d'accès** des séquences utilisables

*Longueur, accès, slicing, appartenance (in), parcours (for)*

```
1 s = 'Hello world!'
2
3 print(len(s))           # 12
4 print(s[0])            # H
5 print(s[0:4])          # Hell
6 print(' ' in s)        # True
7 for c in s:            # Helloworld!
8     if c != ' ':
9         print(c, end=' ')

```

# Littéraux

- Littéraux **sur une ligne** délimités par ' ou "

*Choix libre du simple ou double guillemet*

- Littéraux étendus sur **plusieurs lignes** avec triples guillemets

*Notamment utilisés pour la documentation*

```
1 question = "Where's my Maneki-neko?"
2
3 affirmation = 'You can call me "The Beast"!'
```

```
4
5 haiku = """Yesterday it worked.
6 Today it is not working.
7 Windows is like that."""
```

Plus de haikus pour programmeurs ici : <http://www.libertybasicuniversity.com/lbnews/nl107/haiku.htm>.

# Séquence d'échappement (1)

- **Séquence d'échappement** pour insérer certains caractères

*Utilisation du caractère d'échappement \*

- Caractères spéciaux ou spécification de la **valeur Unicode**

*Caractère Unicode spécifié par sa valeur en hexadécimal*

```
1 s = 'Courses:\n- DVD\t12.99\u20ac\n- Café\t1.50\u20ac'  
2 print(s)
```

```
Courses:  
- DVD 12.99€  
- Café 1.50€
```

# Séquence d'échappement (2)

- Quelques exemples de **séquences d'échappement**

*Certaines sont associées à une valeur*

| Séquence                     | Description   |
|------------------------------|---|
| <code>\nouvelle_ligne</code> | Ignorer la nouvelle ligne                                     |
| <code>\\</code>              | Backslash   |
| <code>\'</code>              | Guillemet simple  |
| <code>\"</code>              | Guillemet double  |
| <code>\b</code>              | Backspace   |
| <code>\a</code>              | Cloche  |
| <code>\n</code>              | Nouvelle ligne  |
| <code>\r</code>              | Retour chariot  |
| <code>\t</code>              | Tabulation horizontale  |
| <code>\xhh</code>            | Caractère avec la valeur hexadécimale sur 8 bits <i>hh</i>    |
| <code>\uhhhh</code>          | Caractère avec la valeur hexadécimale sur 16 bits <i>hhhh</i> |

# Exemple : Compte à rebours

- Pause dans l'exécution du programme avec `time.sleep`

*Temps d'attente précisé en secondes*

- Effacer le texte dans le terminal avec `\b`

*Forcer l'écriture avec `sys.stdout.flush()`*

```
1 import sys
2 import time
3
4 counter = 5
5 while counter > 0:
6     print('\b{}'.format(counter), end='')
7     sys.stdout.flush()
8     counter -= 1
9     time.sleep(1)
10 print('\bBOOM!')
```

# Chaîne de caractères brute

- Séquences d'échappement ignorées dans les chaînes brutes

*Les caractères d'une chaîne brute sont pris tels quels*

- Définie avec le caractère `r` avant le guillemet ouvrant

```
1 print(r'Utilisez \u20ac pour ' + 'insérer un \u20ac !')
```

```
Utilisez \u20ac pour insérer un € !
```

# Classe bytes

- **Objet bytes** est une séquence d'entiers entre 0 et 255

*Représentation en mémoire d'une chaîne de caractères Unicode*

- **Littéral bytes** déclaré en préfixant la chaîne avec b

*Ne fonctionne qu'avec les caractères ASCII*

```
1 b = b'Listening Nightcore!'
2 print(b)
3 print(type(b))
4 print(b[0])
```

```
b'Listening Nightcore!'
<class 'bytes'>
76
```

# byte et str

- On ne **mélange pas** des byte et str dans une même opération

*Erreur de type incompatible générée par l'interpréteur Python*

- **Conversion possible** d'un type vers l'autre

*Utilisation des méthodes `encode` et `decode`*

```
1 s = 'Hello'
2 b = b'World!'
3
4 print(s + ' ' + b)
```

```
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print(s + ' ' + b)
TypeError: Can't convert 'bytes' object to str implicitly
```

# Encodage et décodage (1)

- Spécification de l'**encodage à utiliser** avec `encode` et `decode`

*Par défaut, il s'agira d'UTF-8 (8 octets par code point)*

- Erreur de décodage possible lors d'**incompatibilité**

*UnicodeEncodeError et UnicodeDecodeError*

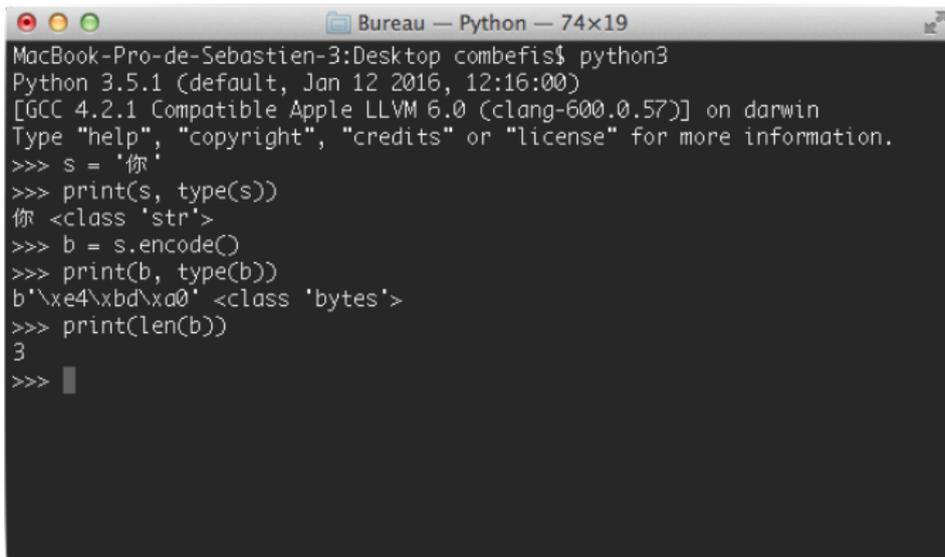
```
1 s = 'Même pas vrééé !'  
2 b = s.encode('iso-8859-1')  
3 print(b.decode('utf-8'))
```

```
Traceback (most recent call last):  
  File "test.py", line 3, in <module>  
    print(b.decode('utf-8'))  
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xea in  
position 1: invalid continuation byte
```

# Encodage et décodage (2)

- Un caractère peut être représenté par **plusieurs octets**

*Dépend évidemment de l'encodage utilisé*



```
Bureau — Python — 74x19
MacBook-Pro-de-Sebastien-3:Desktop combefis$ python3
Python 3.5.1 (default, Jan 12 2016, 12:16:00)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> s = '你'
>>> print(s, type(s))
你 <class 'str'>
>>> b = s.encode()
>>> print(b, type(b))
b'\xe4\xbd\xa0' <class 'bytes'>
>>> print(len(b))
3
>>>
```

# Classe bytearray

- La **classe bytearray** représente une séquence mutable d'octets

*Création à partir d'un str ou bytes*

- **Conversion** vers une chaîne de caractères avec `decode`

```
1 data = bytearray("J'aine les pâtes", 'utf-8')
2 print(data)
3
4 data[12:14] = b'o'
5 print(data.decode())
```

```
bytearray(b"J'aine les p\xc3\xa2tes")
J'aine les potes
```

# Classe str

- La **classe str** représente les chaînes de caractères

*Séquence immuable de caractères Unicode*

- Pleins de **méthodes et opérateurs** de manipulation disponibles

```
1 s = "Hello"
2 t = r'\t insère une tabulation'
3 copy = str(s)
4
5 data = [65, 71, 69]
6 u = bytes(data).decode()
7
8 print(s, t, copy, u, sep='\n')
```

```
Hello
\t insère une tabulation
Hello
AGE
```

# Opérateur

- Plusieurs **opérateurs** peuvent être appliqués aux str

*Dont tous ceux applicables aux séquences immuables*

| Méthode | Description                                 |
|---------|---|
| len     | Longueur                                    |
| +       | Concaténation de deux chaînes de caractères |
| in      | Test de la présence d'une sous-chaîne       |
| []      | Accès à un caractère ou slicing             |
| *       | Répétition                                  |

```
1 s = 'Po'
2 r = s * 4 + 'ooo'
3 print(r)                # PoPoPoPoooo
4 print('op' in r.lower()) # True
```

# Majuscule/minuscule

| Méthode                 | Description  |
|-------------------------|--|
| <code>capitalize</code> | Copie avec la première lettre convertie en majuscule                     |
| <code>upper</code>      | Copie avec toutes les lettres converties en minuscule                    |
| <code>lower</code>      | Copie avec toutes les lettres converties en majuscule                    |
| <code>casefold</code>   | Comme <code>lower</code> , mais avec remplacement de certains caractères |
| <code>swapcase</code>   | Copie avec inversion minuscules/majuscules                               |
| <code>title</code>      | Copie avec première lettre de chaque mot convertie en majuscule          |

```
1 s = 'das große Haus'
2 print(s.capitalize())      # Das große haus
3 print(s.upper())           # DAS GROSSE HAUS
4 print(s.lower())           # das große haus
5 print(s.casefold())        # das grosse haus
6 print(s.swapcase())        # DAS GROSSE hAUS
7 print(s.title())           # Das Große Haus
```

# Test

| Méthode    | Description  |
|------------|--|
| endswith   | Teste si la chaîne se termine par un suffixe                       |
| startswith | Teste si la chaîne commence par un préfixe                         |
| isalnum    | Teste si la chaîne ne contient que des caractères alpha-numériques |
| isalpha    | ... alphabétiques  |
| isdigit    | ... numériques   |
| islower    | ... minuscules   |
| isupper    | ... majuscules   |

```
1 s = 'große'
2 print(s.startswith('g')) # True
3 print(s.endswith('E')) # False
4 print(s.islower()) # True
5 print(s.isalnum()) # True
6 print(s.isalpha()) # True
7 print(s.isdigit()) # False
```

---

| Méthode | Description   |
|---------|---|
| count   | Compte le nombre d'occurrence non chevauchées d'une sous-chaine |
| find    | Trouve l'indice de la plus petite position d'une sous-chaine    |
| index   | Comme find, mais ValueError s'il ne trouve pas                  |
| rfind   | Trouve l'indice de la plus grande position d'une sous-chaine    |
| rindex  | Comme rfind, mais ValueError s'il ne trouve pas                 |
| replace | Remplace les occurrences d'une sous-chaine par une autre        |

---

```
1 s = 'das große Haus'
2 print(s.count('s'))           # 2
3 print(s.find('s'))           # 2
4 print(s.rfind('s'))          # 13
5 print(s.index('A'))          # ValueError
6 print(s.replace(' ', ';'))   # das;große;Haus
```

# Mise en page

| Méthode             | Description   |
|---------------------|---|
| <code>ljust</code>  | Complète pour aligner à gauche dans une largeur spécifiée |
| <code>center</code> | Complète pour centrer dans une largeur spécifiée          |
| <code>rjust</code>  | Complète pour aligner à droite dans une largeur spécifiée |

```
1 s = 'große'  
2 print(s.ljust(10, '.'))  
3 print(s.center(10, '_'))  
4 print(s.rjust(10))
```

```
große.....  
--größe---  
         große
```

# Découpe

| Méthode                 | Description   |
|-------------------------|---|
| <code>rstrip</code>     | Copie avec des caractères à gauche supprimés                  |
| <code>rstrip</code>     | Copie avec des caractères à droite supprimés                  |
| <code>strip</code>      | Copie avec des caractères à droite et à gauche supprimés      |
| <code>split</code>      | Découpe en fonction d'un séparateur                           |
| <code>splitlines</code> | Découpe en lignes   |
| <code>partition</code>  | Découpe en trois selon la première occurrence d'un séparateur |
| <code>rpartition</code> | Découpe en trois selon la dernière occurrence d'un séparateur |

```
1 s = '   grosßsseeeee'
2 print(s.rstrip('es'))      #   grosß
3 print(s.lstrip())         # grosßsseeeee
4 print(s.partition('s'))   # ('   gro', 's', 'ßsseeeee')
5 print(s.rpartition('s'))  # ('   grosßs', 's', 'eeeee')
```

---

| Méthode | Description |
|---------|-------------|
|---------|-------------|

---

|      |   |
|------|---|
| join | Fusionne des chaînes de caractères avec un caractère spécifié |
|------|---|

---

```
1 s = 'Je suis un gros lapin'
2 words = s.split(' ')
3
4 t = ', '.join(words)
5 print(t)
6 print(', '.join([x.upper() for x in words]))
```

```
Je,suis,un,gros,lapin
JE,SUIS,UN,GROS,LAPIN
```

# Conversion

- Donnée vers chaîne de caractères avec `str`

*Définir la méthode `__str__` pour les nouvelles classes*

- Chaîne de caractères vers une donnée par `parsing`

*Immédiatement ou via une fonction spécifique*

```
1 i = 847
2 s = str(i)
3 print(s, type(s))
4
5 r = '1,9,2,3,-4,12'
6 data = [int(x) for x in r.split(',')]
7 print(data, type(data))
```

```
847 <class 'str'>
[1, 9, 2, 3, -4, 12] <class 'list'>
```

# Concaténation

- Plus rapide de **passer par join** que concaténer

*Pour construire des chaînes de caractères à incrémentalement*

- La concaténation **crée pleins de nouvelles** chaînes de caractères

```
1 def concat(n):
2     result = ''
3     i = 1
4     while i <= n:
5         result += str(i) + ','
6         i += 1
7     return result[:-1]
```

concat(1000000)  $\approx$  575 ms

```
1 def concat(n):
2     elems = []
3     i = 1
4     while i <= n:
5         elems.append(str(i))
6         i += 1
7     return ','.join(elems)
```

concat(1000000)  $\approx$  454 ms

# Formatter

- **Insertion d'une valeur** à l'intérieur d'une chaîne de caractères

*Définition d'un emplacement et spécification d'une valeur*

- Utilisation de la **méthode format** sur une chaîne

*Balises dans la chaîne définies avec {}*

```
1 s = "J'ai {age} ans.".format(age=30)
2 print(s)
3
4 d = '{0}/{1}/{2}'.format(22, 1, 2016)
5 print(d)
```

```
J'ai 30 ans.
22/1/2016
```

# Appel des arguments

| Format | Description  |
|--------|--|
| {0}    | Premier paramètre positionnel                        |
| {}     | Prend implicitement le paramètre positionnel suivant |
| {name} | Paramètre nommé name                                 |
| 0.attr | Attribut attr du premier paramètre positionnel       |
| 0[0]   | Premier élément du premier paramètre positionnel     |

```
1 from collections import namedtuple
2
3 Point = namedtuple('Point', ['x', 'y'])
4 p = Point(7, -2)
5 data = [1, 2, 3]
6
7 print('{0.y} et {1[2]}'.format(p, data))
```

-2 et 3

# Définition du format

| Format         | Description  |
|----------------|--|
| {:char<n}      | Texte aligné à gauche, sur largeur de <i>n</i> , rempli avec <i>char</i> |
| {:char^n}      | Texte centré, sur largeur de <i>n</i> , rempli avec <i>char</i>          |
| {:char>n}      | Texte aligné à droite, sur largeur de <i>n</i> , rempli avec <i>char</i> |
| {:d}           | Conversion nombre entier   |
| {:s}           | Conversion chaîne de caractères  |
| {:.precisionf} | Conversion nombre flottant avec <i>précision</i> après la virgule        |

```
1 value = 15.9842
2 print('{0:f}\n{0:.2f}\n{0:>10.2f}\n{0:*>10.2f}'.format(value))
```

```
15.984200
15.98
      15.98
*****15.98
```

# Template

- Construction d'un **modèle** dans lequel incruster des valeurs  
*Ajout de balise dans une chaîne et insertion avec `substitute`*
- Le modèle ne doit être construit qu'une seule fois  
*Et ensuite réutilisable avec plusieurs valeurs différentes*

```
1 import string
2
3 s = string.Template('Bonjour $prenom $nom')
4
5 print(s.substitute(prenom='Quentin', nom='Lurkin'))
```

```
Bonjour Quentin Lurkin
```

# Encodage fichier source

- Par défaut, code source fichier Python en **UTF-8**

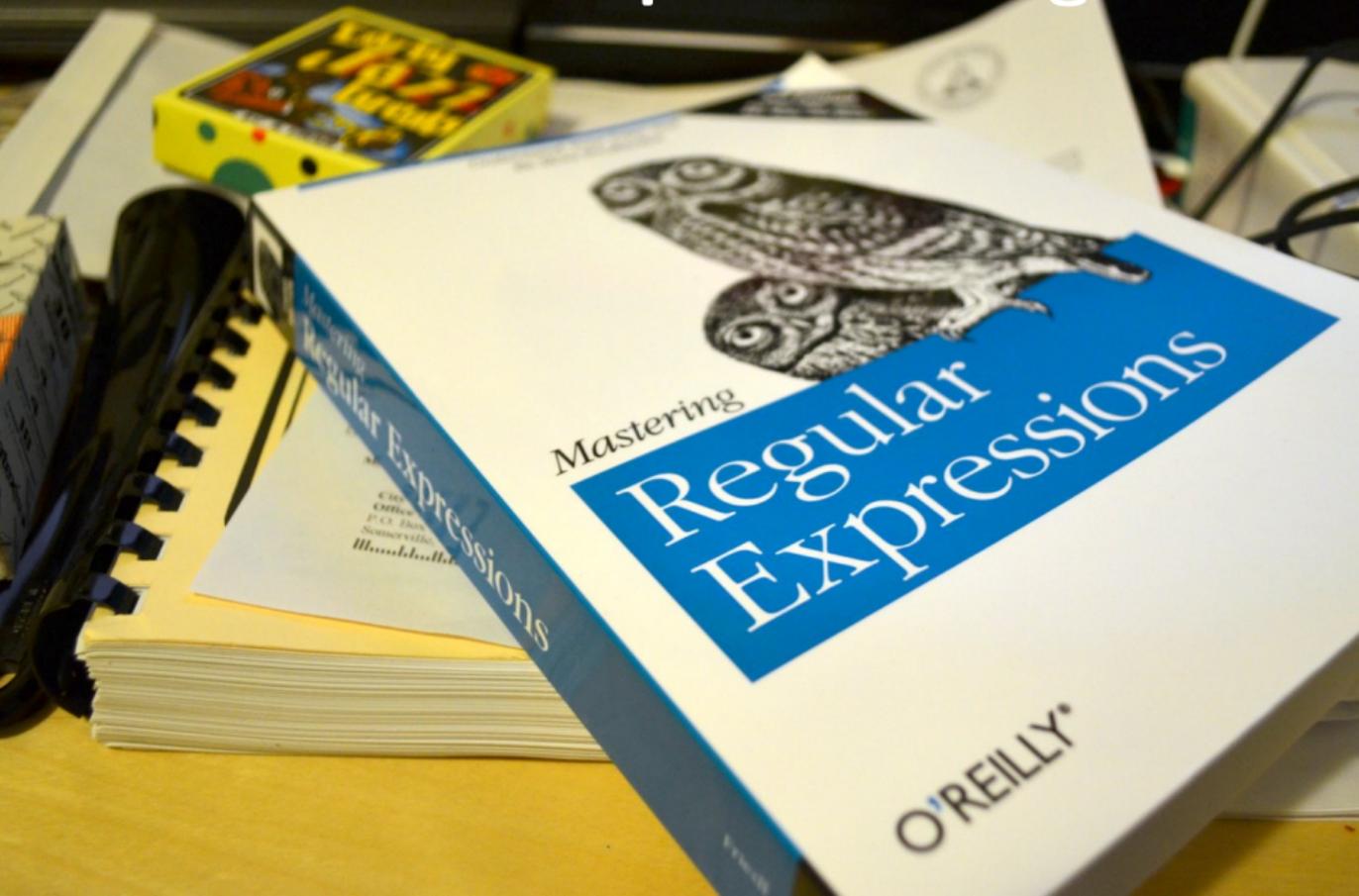
*Depuis Python 3 uniquement, ASCII en Python 2*

- Possibilité de déclarer l'**encodage du fichier**

*Utilisation d'un commentaire spécial en début de fichier*

```
1 # -*- coding: windows-1252 -*-  
2  
3 # blabla...
```

# Expressions régulières



# 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*

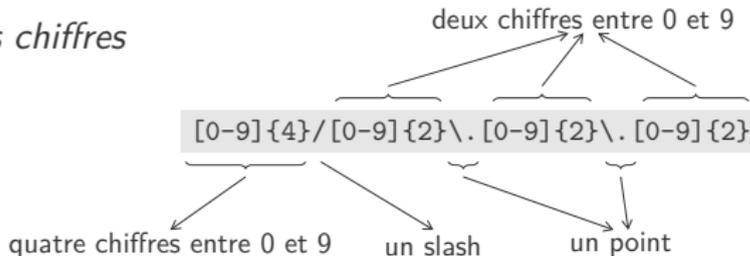
- Vérifier les données avec une **expression régulière** (module `re`)

*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

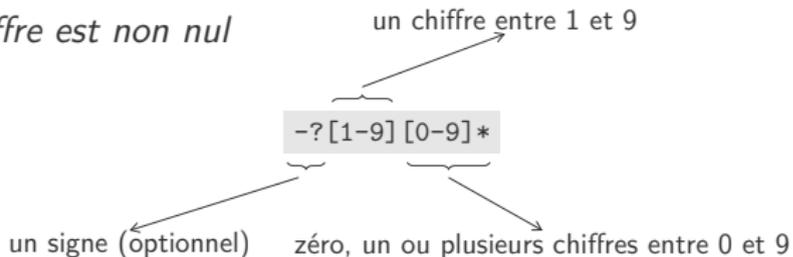
$\wedge$  : début de la ligne

$\$$  : 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  $\text{\LaTeX}$

| 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 matchs 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">'') :
9     print(m.group(1))
```

```
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>