INF8007 – Languages de scripts
Expressions régulières

Antoine Lefebvre-Brossard

Hiver 2018

Ce que c'est

- Moyen de manipuler les chaînes de caractères selon des règles générales
- Utile lorsqu'on connaît la forme du texte que l'on cherche, mais pas le texte exact
- Par exemple, un numéro de téléphone serait sous la forme :

```
\d{3}-\d{3}-\d{4}
```

Syntaxe de base

- "`\w`" : Représente une lettre (minuscule ou majuscule), un chiffre ou "_"
- "`\s`" : Représente un espace, une tabulation ("`\t`") ou un saut de ligne ("`\n`")
- "`\d`" : Représente un chiffre
- "`\W`" : Représente ce qui n'est pas dans "`\w`"
- "`\S`" : Représente ce qui n'est pas dans "`\s`"
- "`\D`" : Représente ce qui n'est pas dans "`\d`"

Exemples

Est-ce que les expressions régulières font matcher le texte à côté ?

`\d\d\d-\d\d\d-\d\d\d\d`	514-123-9876	**Oui**
`\w\w\w-\w\w\w-\w\w\w\w`	514-123-9876	**Oui**
`\w\w\s\w\d\w\w`	Le chat	**Non**

Répétitions

- "`*`" : Représente 0 ou plus répétitions du caractère précédent
- "`+`" : Représente 1 ou plus répétitions du caractère précédent
- "`?`" : Représente 0 ou 1 répétition du caractère précédent
- "`{`b`}`" : Le caractère précédent répété 'b' fois
- "`{`a`,`b`}`" : Le caractère précédent répété entre 'a' et 'b' fois (inclusivement)
- "`{,`b`}`" : Le caractère précédent répété jusqu'à 'b' fois
- "`{`a`,}`" : Le caractère précédent répété au moins 'a' fois

Exemples

Est-ce que les expressions régulières font matcher le texte à côté ?

`\d{3}-\d{3}-\d{4}`	514–123–9876	**Oui**
`\d{,3}-\d{3,}-\d{3,5}`	514–123–9876	**Oui**
`\d{,4}-\d{4,}-\d{2,6}`	514–123–9876	**Non**

Caractères spéciaux

- "`.`" : Représente n'importe quel caractère sauf un saut de ligne ("`\n`")
- "`^`" : Représente le début du texte
- "`$`" : Représente la fin du texte
- "`\b`" : Représente la bordure d'un mot, c'est-à-dire le fait qu'il y ait un "`\s`" et un "`\w`" côte-à-côte

Exemples

Est-ce que les expressions régulières font matcher le texte à côté ?

`\bch`	chat ou chien	**Oui**
`\ben`	chat ou chien	**Non**
`^ch`	chat ou chien	**Oui**
`ch$`	chat ou chien	**Non**
`\bch.{3}\b`	chat	**Non**
`\bch.{3}\b`	chien	**Oui**
`\bch.{,3}\b`	chat	**Oui**

Choix

- "`|`" : Choix entre le caractère ou groupe de chaque côté
- "`[`...`]`" : N'importe quel caractère entre les crochets
- "`[^`...`]`" : N'importe quel caractère qui n'est pas entre les crochets
- "`[`a`-`b`]`" : Entre les crochets, le trait d'union représente un intervalle. Ceux normalement utilisés sont "a-z", "A-Z" et "0–9"

Exemples

Est-ce que les expressions régulières font matcher le texte à côté ?

`chat|chien`	Le chat est noir	**Oui**
`ch[aeint]+`	Le chien est noir	**Oui**
`[2-5]{,2}\s[6-9]{3}`	35 600 minutes	**Non**

Groupes

- "(`...`)" : Ce qui est entre parenthèses est un groupe capturé qui peut potentiellement être utilisé plus tard
- "(?:`...`)" : Groupe non capturant (ne peut être utilisé plus tard)
- "\`i`" : Représente le 'i'ème groupe capturé

Exemples

Est-ce que les expressions régulières font matcher le texte à côté ?

`^Le (chat|chien) noir$`	Le chat est noir	**Oui**
`^(.+@[a-z]+\.(?:com|ca))$`	addresse@mail.com	**Oui**
`^\w{6} (\w+): \w{3}_\1\.py$`	Équipe b : td1_b.py	**Oui**

Conditions

- "`(?=`...`)`" : Match si les caractères après respectent '...'
- "`(?!`...`)`" : Match si les caractères après ne respectent pas '...'
- "`(?<=`...`)`" : Match si les caractères précédents respectent '...'
- "`(?<!`...`)`" : Match si les caractères précédents ne respectent pas '...'

Exemples

Est-ce que les expressions régulières font matcher le texte à côté ?

`\w+(?= noir)`	chat noir	**Oui**
`\w+(?= bleu)`	chat noir	**Non**
`(?<=blanc )cheval\w+`	blanc cheval	**Oui**
`(?<!blanc )cheval\w+`	blanc cheval	**Non**

Utilisation

- En python, il faut utiliser le module `re` en l'important : `import re`
- Le plus facile pour écrire l'expression régulière est le *raw string* `r"..."`. Sinon, il faut répéter toutes les "\" deux fois

Utilisation (suite)

- La fonction a utilisé pour obtenir le match est `re.match`
- Si celle-ci échoue, elle retourne `None`
- Si celle-ci réussie, elle retourne un objet
- Pour obtenir tous les groupes du match, l'objet a une fonction `` `match`.groups() ``
- Pour un seul groupe, l'objet a la fonction `` `match`.group(`i`) ``
- Le premier groupe est toujours le match complet

Utilisation (suite)

- Une autre fonction utile est `re.sub(`regex_match`, `regex_replace`, `text`)`
- Celle-ci permet de remplacer tous les matchs de `` `regex_match` `` par `` `regex_replace` ``
- Il est aussi possible de déclarer un groupe dans `` `regex_match` `` et de l'utiliser dans `` `regex_replace` ``
- **Exemple :**
  `re.sub(r"(\w)'(?=\w+)", r"\1' ", "l'imposteur")`
  va retourner `l' imposteur`

Utilisation (suite)

- La dernière fonction présenté est `re.split(`regex`, `text`)`
- Celle-ci permet de séparer un texte en une liste de `str` de façon plus générale que d'utiliser `` `text`.split(`c`) ``
- **Exemple** : `re.split(r"(?<!\d),(?!\d)", "id,item")` donne `["id", "item"]`, mais `re.split(r"(?<!\d),(?!\d)", "2,34")` donne `["2,34"]`