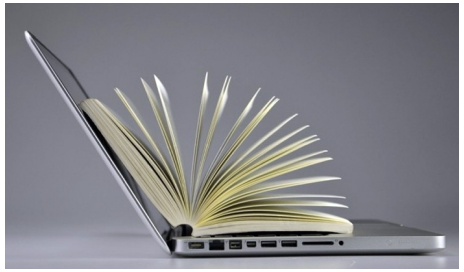
Sorbonne Nouvelle
université des cultures



Expressions régulières (*regex*)

Ljudmila PETKOVIC

Introduction aux humanités numériques (L1HN001)
Mineure « Humanités numériques », licence Lettres
Paris, le 16 novembre 2023, année 2023-2024

Les projets en humanités numériques

Quatre principales étapes :

1. Acquisition d'un objet d’étude
2. Traitement d'un objet d’étude
3. Exploitation d'un objet d’étude
4. Publication des résultats

⚠ ***NB***:
Les frontières entre ces étapes sont très poreuses et l’ordre des opérations ne respecte pas toujours cette progression logique.

Les projets en humanités numériques

Quatre principales étapes :

1. Acquisition d'un objet d’étude
2. Traitement d'un objet d’étude
3. **Exploitation d'un objet d’étude**
4. Publication des résultats

⚠ ***NB***:
Les frontières entre ces étapes sont très poreuses et l’ordre des opérations ne respecte pas toujours cette progression logique.

Recherches simples et avancées

On est habitués à faire des recherches dans un texte, mais elles sont souvent limitées. Nous avons donc besoin d'une méthode d'extraction de texte pour répondre **d'un coup** aux questions spécifiques, par exemple :

- comment trouver les occurrences des mots *Paris* et *Nice* ?
- comment trouver tous les chiffres ? etc.

Les **expressions régulières (*regex*)** permettent de faire des recherches avancées, mais aussi les remplacements (substitutions).

Définition de la regex

Une suite de caractères typographiques – **motif** / **patron** (angl. ***pattern***) – décrivant un ensemble de chaînes de caractères selon une syntaxe (règles) précise. C'est une recherche de caractères assortis de metacaractères ou de délimiteurs[1] qui permettent d'affiner la recherche.

Chaîne de caractères (angl. *string*) :

- (conceptuellement) une suite ordonnée de caractères
- (physiquement) une suite ordonnée d'unités de code (angl. *code unit*)

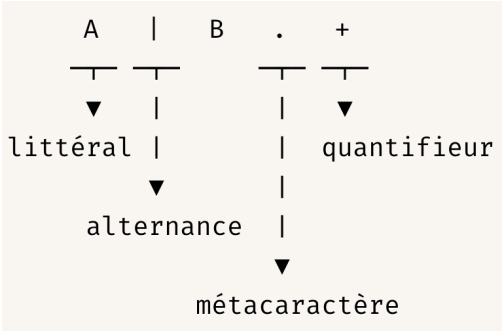
Regex : type de donnée[2] dans de nombreux langages informatiques (ex. : Python), ainsi que dans les éditeurs de texte (Notepad++, TextMate, TextEdit, Geany etc.)

[1] caractère, symbole qui limite une suite de caractères, sans en faire partie.

[2] informations représentant un texte, une page, une image, un son, un fichier exécutable, etc.

Regex = notation

- notation utilisée pour décrire un langage : lettres / mots gouvernés par une grammaire
  - p. ex. le langage décrit par `ab*c` inclut des chaînes de caractères qui contiennent la lettre *a* suivie d'un nombre indéterminé de *b*, suivi d'un *c* (***abc**, **abbc**, **abbbc**...*)
  - la syntaxe contient des caractères littéraux (`a` , `b` ...), des métacaractères (`.`), des quantificateurs (`+`), de l'alternance (`|`) et d'autres



Syntaxe des regex (Hodoul, 2023).

Regex et TAL

Dans le contexte de la programmation, elles sont utilisées pour trouver des chaînes de caractères qui correspondent au langage décrit.

Cas d'utilisation typiques :

- traitement de texte
- génération de texte
- validation de données

Quand utiliser une regex ?

- chercher une information simple dans un texte (*“Quels sont tous les numéros de téléphone français dans ce texte ?”*)
- valider une chaine de caractères basée sur un motif simple (*“Est-ce une date au format JJ/MM/AAAA ?”*)
- extraire d’une chaîne des composants finis et simples (*“Quel est le nom d’hôte de cette URL ?”*)
- utiliser la fonctionnalité *rechercher et remplacer*

Quand ne pas utiliser une regex ?

- analyser un langage de programmation ou de balisage qui permet des imbrications infinies (*“Combien y a-t’il de liens dans ce document HTML ?”*)
- valider une donnée complexe (*“Est-ce que cette adresse email est valide ?”*)
- analyser le langage naturel (*“Quel est le sujet de cette phrase ?”*)

Histoire des regex

- Stephen Kleene semble être considéré comme l'inventeur des regex, au début des années 1950
- dans les années 50 également, Noam Chomsky présente sa hiérarchie des langages (classification des grammaires formelles)
- `ed`, l'éditeur de texte de Ken Thompson est l'un des premiers programmes à utiliser les regex, à la fin des années 1960
- dans les années 1980, le langage de programmation Perl offre des regex aux fonctionnalités encore plus puissantes

Regex et automates finis

Il est utile d'imaginer une regex comme un programme extrêmement concis. Les regex peuvent être représentées comme les « automates finis » (angl. *finite-state automata / machines*).

Exemple d'un automate fini (Hodoul, 2023).

Classes de caractères

Les classes sont une manière de contourner des problèmes complexes en formant des ensembles de signes.

Caractères :

- la ponctuation
- les lettres en minuscules / majuscules
- les chiffres

Caractères non imprimables :

- un espace, une tabulation, un saut de ligne (nouvelle ligne) etc.

Démo

Allez sur le site https://regex101.com ➤ c'est un éditeur interactif qui permet de tester des regex et qui explique bien chaque composant d'une expression regulière. Lors des exercices, vous pouvez taper ou copier-coller (plus rapide) les textes utilisés pour les tests dans l'interface du site.

Accessoirement, le site iHateRegex permet de visualiser une regex par un graphe, ce qui illustre bien son origine d'automate.

La syntaxe des regex peut varier légèrement d'un éditeur ou d'un programme à l'autre, mais les fonctions restent les mêmes.

Classes de caractères : exemples

Avec les regex, on peut construire ces ensembles selon nos besoins :

- `a` pour la lettre minuscule *a*
- `A` pour la lettre majuscule *A*

Certaines classes étant très communes, il existe des versions abrégées :

- `[abcdefghijklmnopqrstuvwxyz]` n'importe quelle lettre minuscule de *a* à *z*
  - ou `[a-z]`
- `[ABCDEFGHIJKLMNOPQRSTUVWXYZ]` n'importe quelle lettre minuscule de *A* à *Z*
  - ou `[A-Z]`
- `[0123456789]` pour les chiffres
  - ou `[0-9]`

Classes de caractères : raccourcis

Certaines classes ne nécessitent pas l'emploi de crochets :

- chiffres `[0123456789]` = `[0-9]` = `\d` (angl. *digits*)
- `\D` tout ce qui n'est pas dans `\d`
- `\w` tout ce que l'on peut trouver dans un mot (angl. *word*) (= `[a-zA-Z0-9_]`)
- `\W` tout ce qui n'est pas dans `\w`
- `\s` tout ce qui représente un espace (angl. *space*) :
  - espace `\s`
  - tabulation `\t`
  - saut de ligne `\n` (angl. *new line*)
- `\S` tout ce qui n'est pas dans `\s`

Quantificateurs

Pour trouver une séquence précise, nous pouvons utiliser un quantificateur, élément attaché à un caractère (ou une classe de caractère) qui spécifie le nombre de répétition voulues.

Le quantificateur le plus simple est un nombre entre accolades : p. ex. `{5}`.

- `\d{5}` indique exactement 5 chiffres, identique à `\d\d\d\d\d`

Le quantificateur peut être encore affiné avec des bornes :

- `[0-9]{2,4}` permet de trouver un chiffre répété entre 2 et 4 fois

Autres quantificateurs

- `*` (angl. *joker, wildcard, étoile de Kleene*) récupère le caractère précédent répété **0 fois ou plus**
  - `pas*` récupère ***pas***, ***pass****er* et ***pa****tate*
- `+` récupère le caractère précédent répété **1 fois ou plus**
  - `pas+` récupère ***pas*** et ***pass****er*, mais pas *patate*
- `?` récupère un caractère absent ou présent
  - `pas?` récupère ***pas***, ***pass****er* et ***pa****tate*

Ancres

Les ancres permettent de préciser la position des caractères. Ainsi dans l'exemple d'une exclamation de rire :

```
ah ah ah
```

- `^` sélectionne les mots en **début** de ligne (`^ah` sélectionne le premier *ah*)
- `$` est le symétrique de `^` et sélectionne la **fin** de ligne (`ah$` sélectionne le dernier *ah*)
- `\b` permet de sélectionner les **début et fin** de mots (angl. *boundary*) (`\bah\b` sélectionne tous les *ah*)

Négation

Il est possible de chercher l'inverse d'un caractère ou d'une classe en utilisant le signe `^` entre crochets :

- `[^i]` est l'inverse de `[i]` (ou `i`) : tout ce qui n'est pas un *i* minuscule
- Attention aux type de **parenthèses** ! ne pas confondre :
  - `[^a-z]` tous les caractères sauf une minuscule, et
  - `(^a-z)` les caractères qui ne commencent pas par une minuscule

Alternatives

Il est possible de laisser ouverte une alternative entre deux solutions avec le signe `|` (barre verticale, tube, angl. *pipe*, opérateur booléen), signifiant *ou*.

- `Paris|Nice` récupère les occurrences des mots *Paris* ou *Nice* ou les deux

Caractère d'échappement

- angl. *escape character*
- déclenche une interprétation alternative du ou des caractères qui le suivent
    - le signe `\` (angl. *antislash*) est utilisé pour échapper les caractères utilisés dans la syntaxe des regex
        - p. ex. le signe `+` permet d'indiquer qu'un caractère ou une classe sont répétés un nombre infini de fois : `a+` = ***aaaaaa***... ➤ quantificateur
        - par contre, `\+` permet de chercher littéralement le signe **+** ➤ littéral

Deux types d'opérations

1. **Correspondance**
    - vérifier si une chaîne de caractères comporte un motif donné
        - `Paris|Nice`
2. **Substitution**
    - transformer le contenu d'une chaîne de caractères en fonction d'un motif donné (*rechercher / remplacer* perfectionné)
        - substituer les occurrences de *Paris* et *Nice* avec le mot *Bordeaux*

          a) correspondance : `Paris|Nice`

          b) substitution : `Bordeaux`

Exercices

Exercice 1

Capturer toutes les mentions de siècles, peu importe le format dans le texte suivant :

```
Victor Hugo est né au XIXème siècle, Jean Racine au XVIIème et Rabelais au 16ème.
```

Exercice 2

```
En 1666, il a passé le pas de 66 portes, n'est-ce pas ?
```

Si nous recherchons `pas`, que se passe-t-il ? et `[pas]` ? Décrire brièvement quels caractères / mots ont été récupérés par chaque méthode.

Exercice 3

Même phrase :

```
En 1666, il a passé le pas de 66 portes, n'est-ce pas ?
```

Prenons le chiffre *6* (1**666**, **66**).

- chercher `6` va nous donner autant de réponses qu'il y a d'occurrences
- chercher `6+` va nous permettre de trouver les cas où *6* est répété *x* fois

1. Comment repérer le mot ou *6* est répété trois fois et pas deux ?
2. Comment récupérer l'année *1666* (deux réponses possibles) ?
3. Comment récupérer les deux nombres *1666* et *66* (minimum quatre réponses possibles) ?

Exercice 4

```
Voici Louis et sa petite loutre.
```

- Trouver le mot commençant par un `L` en majuscule (*Louis*).

Solutions

1. `[A-Z|\d]+ème`
2.
   - `pas` récupère la chaîne de caractères *pas* dans les mots ***pas****sé*, ***pas*** et [n'est-ce] ***pas***
   - `[pas]` récupère les lettres *p*, *a* ou *s* dans les mots ***a***, ***pas****sé*, ***pas***, ***p****orte****s***, *n'e****st***, ***pas***
3.
   - `[6]{3}`
   - `[0-9][0-9][0-9][0-9]` ou `[0-9]{4}`
   - `\d` ou `[0-9]+` ou `[0-9]*` ou `[0-9]{2,4}`
4.
   - `\bL[a-z]+\b`

Références

- **Gabay, S., & Jeanrenaud, A.** (2021). « Les fondamentaux pour les humanités numériques ». Université de Genève [*diapositives consultées sur le dépôt GitHub* https://github.com/gabays/Fondamentaux].
- **Galleron, I.** (2021). « Introduction aux humanités numériques (L1HN001) [*diapositives en interne*].
- **Hodoul, C.** (2023). « Quand ne PAS utiliser les expressions régulières ». Programmation Orientée Sieste https://programmation-orientee-sieste.dev/quand-ne-pas-utiliser-expressions-regulieres/.