Introduction aux expressions régulières (regex)

METSEM, 1er juin 2023

Jean-Baptiste Pressac, Pierre-Guillaume Prigent, Sébastien de Villèle

Qui sommes-nous ?

Nous sommes trois utilisateurs des expressions régulières travaillant à l'UBO (Brest)

Sébastien de Villèle, chef de projet au Bureau de traduction de l'université (BTU)

Pierre-Guillaume Prigent, sociologue, membre du LABERS

Jean-Baptiste Pressac, resp. bases de données et chargé de diffusion de corpus numériques au CRBC

Séminaire Outils du Quanti, 2017

Nous avons déjà présenté les expressions régulières en novembre 2017 lors du séminaire brestois Les Outils du Quanti.

Ressources en ligne : Un billet d'introduction aux *expressions régulières avec LibreOffice* et un Notebook sur GitHub d'*exemples de regex appliqués à R*

Plan de la séance

1. ***Introduction générale*** (Jean-Baptiste)
2. ***La boite à outil de base des expressions régulières*** (Sébastien) : *la base de la syntaxe illustrée par des exemples*
3. ***Les expressions régulières appliquées à R*** (Pierre-Guillaume)

Introduction générale aux expressions régulières (regex)

Les regex, une définition abrégée (d'après Wikipédia)

Une expression régulière (abrégée en regex ou regexp) est une ***séquence de caractères*** qui spécifie un ***modèle de correspondance*** pour des données textuelles.

Habituellement, ces modèles sont utilisés pour les opérations de **recherche** ou de **chercher / remplacer**, ou pour la ***validation*** de chaînes de caractères.

(d'après https://en.wikipedia.org/wiki/Regular_expression)

regex, un concept de 1950

Le concept d'expressions régulières a été introduit dans les années **1950** par le mathématicien américain **Stephen Cole Kleene** (1909-1994).

Le concept a été implémenté pour la première fois dans **QED**, un éditeur de texte en **1967** sur un système d'exploitation développé à Berkeley.

(d'après https://en.wikipedia.org/wiki/Regular_expression)

Deux exemples de regex

Deux exemples d'*expressions régulières* :

18[5-9][0-9]

(regex correspondant à un chiffre entre 1850 et 1899)

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

(regex correspondant à un courriel)

Caractères et métacaractères

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

(regex correspondant à un courriel)

Toute la difficulté réside dans le fait que certains des ***caractères*** utilisés pour écrire cette regex sont pris tels quel (par le moteur de regex) et que d'autres, tels que les " [ ]" ou certains "+" ou " -" ont une signification particulière dans la syntaxe des expressions régulières.

Ces caractères particuliers sont appelés ***métacaractères***

Caractères et métacaractères

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

(regex correspondant à un courriel)

Les ***métacaractères*** les plus courants :

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

On verra leur signification plus loin…

Caractères et métacaractères

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

(regex correspondant à un courriel)

Cas particulier du tiret qui peut servir à définir des plages de caractères (ex. [a-z] ou [A-Z]) ou être pris tel quel (-)

Caractères et métacaractères

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

(regex correspondant à un courriel)

Les ***métacaractères*** peuvent être échappés avec la barre oblique inversée.

Les *caractères* traités par les regex

Les regex traitent des *caractères* **Unicode**

Unicode, un standard d'encodage de caractères

Unicode est un standard universel d'encodage des caractères pour les caractères écrits et les textes. Il permet d'encoder tous les caractères des langages écrits du Monde, les caractères alphabétiques, les idéogrammes et les symboles.

source : The Unicode® Standard Version 15.0 – Core Specification Introduction

Caractère	Point d'entrée Unicode
R	U+0052
E	U+0045
G	U+0047
E	U+0045
X	U+0078

Les *caractères* traités par les regex

Les regex permettent de traiter **les caractères de tous les alphabets** dans la mesure où ils sont dans l'Unicode, par exemple :

caractères chinois, coréens…
éthiopiens ቡ ፈ
cunéiformes 𒀀 𒀁
etc…

Voir https://en.wikipedia.org/wiki/Regular_expression#Unicode

Les *caractères* traités par les regex

Les regex permettent également de traiter les caractères suivants (codés en Unicode) :

les tabulations,
les espaces,
les sauts de ligne,
les sauts de paragraphes,
les pictogrammes 🥖 ❄…

Les *caractères* traités par les regex

La prise en charge totale ou partielle de Unicode par vos regex dépend du langage ou du logiciel utilisé.

Certains d'entre eux ne supportent qu'un codage spécifique de Unicode, tel que UTF-8

Pour en savoir plus sur la différence entre Unicode et UTF-8, lire Joel Spolsky The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)

Les *caractères* traités par les regex

Vous n’avez pas besoin de connaître Unicode
pour écrire des expressions régulières !

Comment utiliser les regex ?

Les regex ne nécessitent pas de logiciel spécifique.

Des moteurs de regex sont intégrés dans la plupart des langages de programmation et dans des logiciels de la vie courante, par ex. dans le formulaire de recherche de **LibreOffice**..

Un exemple : la recherche avancée de LibreOffice

Un exemple de la vie courante : **le formulaire de recherche avancé de LibreOffice Writer** (equiv. à Microsoft Word)**.**

Un exemple : la recherche avancée de LibreOffice

Hypothèse : un document Word contient des informations concernant les années entre 1800 et 2000.

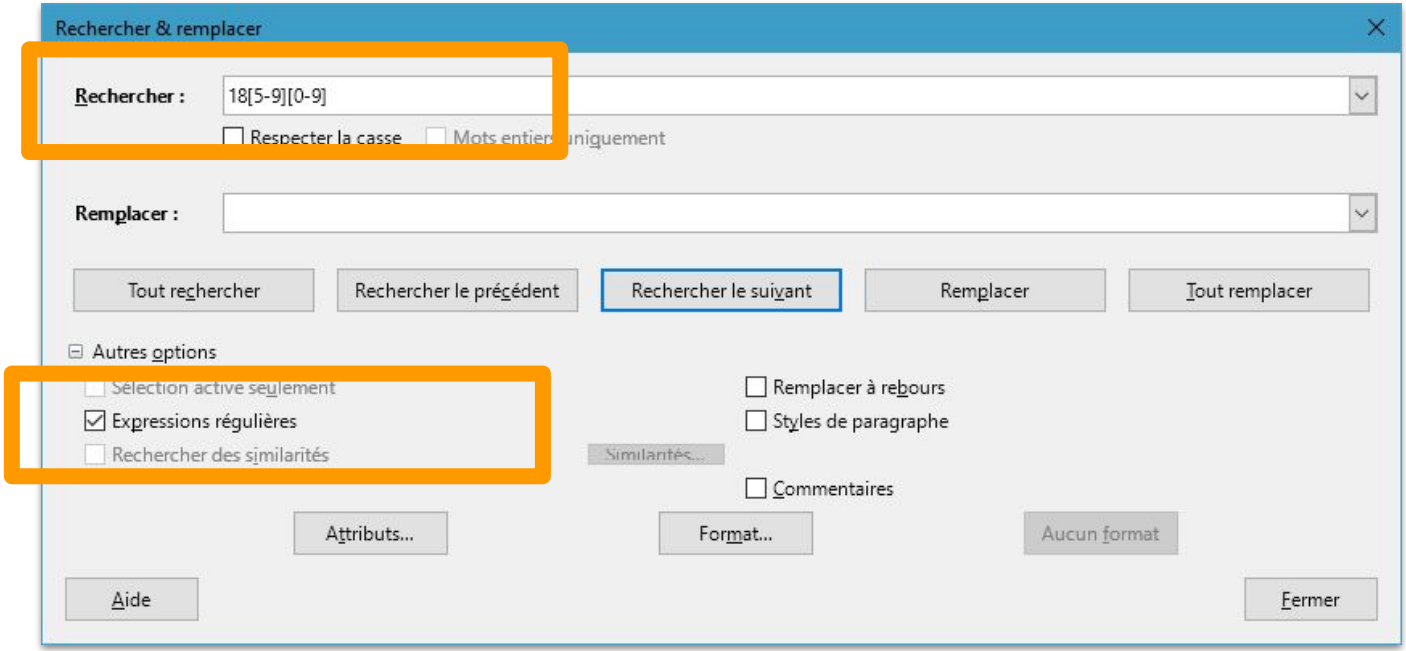
Vous voulez chercher toutes les informations concernant les années entre 1850 et 1899.

Première solution : Vous faites une recherche avec votre logiciel de traitement de texte sur “1850”, puis “1851”, “1852”, etc., jusqu’à “1899”.

Un exemple : la recherche avancée de LibreOffice

Deuxième solution plus rapide : Vous ouvrez le document avec LibreOffice et dans la fenêtre de *Rechercher & remplacer* (Ctrl + H), vous cochez la case *Expressions régulières* sous *Autres options* et vous faites une recherche avec l'expression régulière **18[5-9][0-9]**.

Un exemple : la recherche avancée de LibreOffice



Un exemple : la recherche avancée de LibreOffice

18[5-9][0-9] permet de chercher des chaînes de caractères qui :

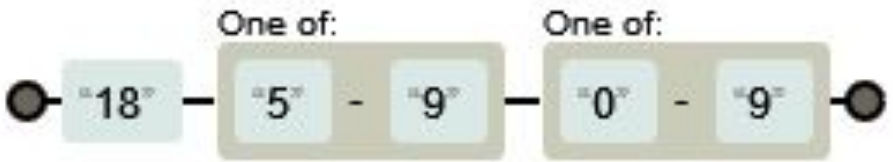
1. commencent par 1
2. suivi du chiffre 8
3. suivi d'un chiffre entre 5 et 9
4. suivi d'un chiffre entre 0 et 9

Soit "1850", "1857", "1876", "1883", etc.

Un exemple : la recherche avancée de LibreOffice

Pour mieux comprendre l'expression régulière, on peut utiliser le site Regexper qui crée des **diagrammes syntaxiques** (*railroad diagrams*) à partir de regex

la regex : **18[5-9][0-9]**



Un exemple : la recherche avancée de LibreOffice

Attention, il ne s'agit **pas d'une recherche sémantique**.

L'expression régulière ne permet **pas** de rechercher **des années**, ni des chiffres compris entre 1850 et 1899 mais des suites de caractères informatiques.

Un exemple : la recherche avancée de LibreOffice

Aussi, l'expression régulière **18[5-9][0-9]** trouvera aussi "1856" et "1899" dans la phrase :

"Ce projet nous aura coûté **1850** euros et mobilisé **1899** personnes." (ça matche aussi !)

Un exemple : la recherche avancée de LibreOffice

Autre exemple : Une expression régulière basique pour chercher des **courriels** dans un document :

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

Un exemple : la recherche avancée de LibreOffice

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

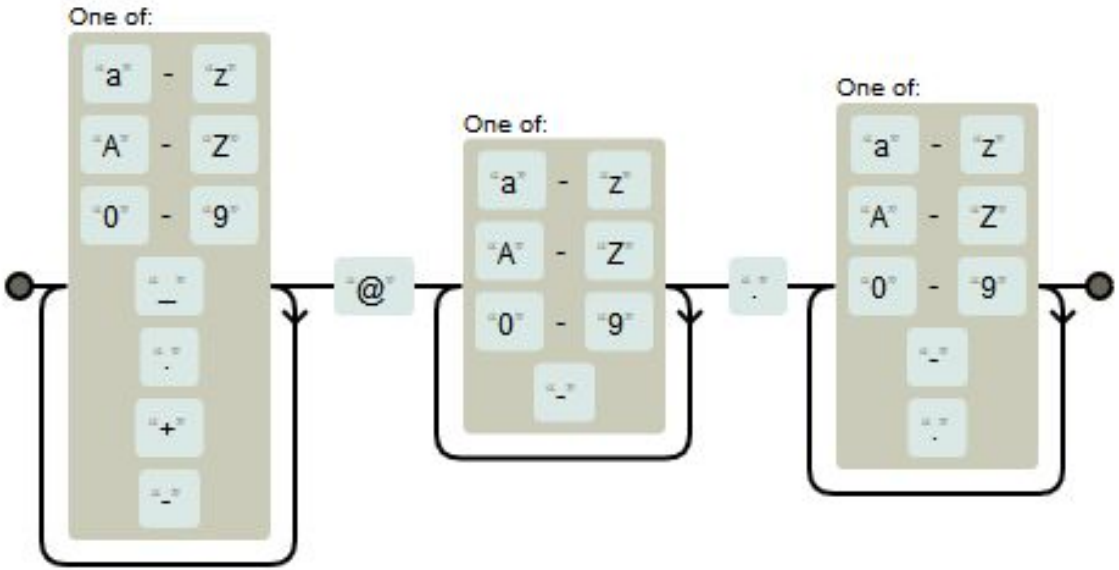
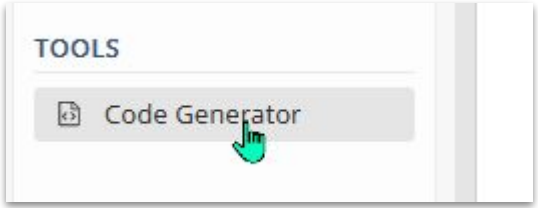
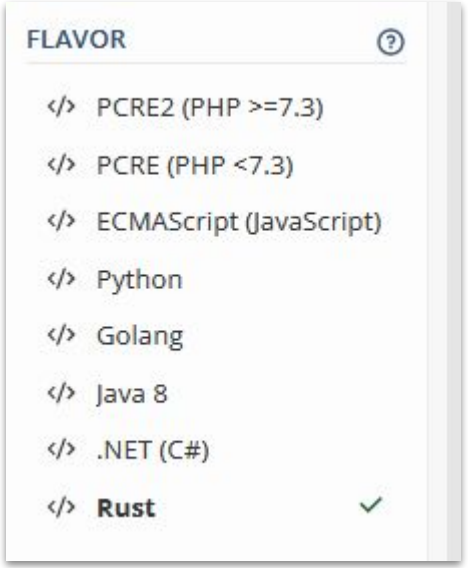


Diagramme syntaxique de l'expression régulière réalisée avec le site [Regexper]

Trouver la regex adaptée à ses besoins

Si utilisation occasionnelle des regex : chercher des motifs sur Internet, par exemple sur regexr.com sur regex101 (par ex. Email Validator) qui propose des solutions pour différentes syntaxes et langages de programmation et même des exemples de code.





Décoder notre regex avec regex101

[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+

https://regex101.com/r/xLMplQ/2

Trouver la regex adaptée à ses besoins

```
[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+
```

Avec cette regex, seule la 2eme ligne du texte ci-dessous sera trouvée car c'est la seule ligne qui commence et se termine par un courriel :

```
Contactez moi à l'adresse robindesbois@nottingham.uk.

sheriff@nottingham.uk

Je mangerai bien une pomme avec Petit Jean, je lui envoie un
mail à petitjean@nottingham.uk …
```

```
Sélectionne aussi le point
final…
```

Trouver la regex adaptée à ses besoins

Si vous trouvez une regex sur Internet, vérifiez si elle est dans la syntaxe adaptée au logiciel ou au langage utilisé.

Par ex. la regex pour détecter des courriels était à l'origine donnée pour ***Python*** :

```
r"(^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$)"
```

Pour fonctionner dans LibreOffice, il faut retirer le préfixe r, les guillemets et la parenthèse englobante.

Trouver la regex adaptée à ses besoins

Il faut également retirer le préfixe ^et le suffixe $, deux métacaractères qui indiquent respectivement le début et la fin d'une ligne

~~r"(~~^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+~~$)"~~

Trouver la regex adaptée à ses besoins

```
^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$
```

Avec cette regex, seule la 2eme ligne du texte ci-dessous sera trouvée car c'est la seule ligne qui commence et se termine par un courriel :

```
Contactez moi à l'adresse robindesbois@nottingham.uk.

sheriff@nottingham.uk

Je mangerai bien une pomme avec Petit Jean, je lui envoie un
mail à petitjean@nottingham.uk …
```

Rechercher…remplacer

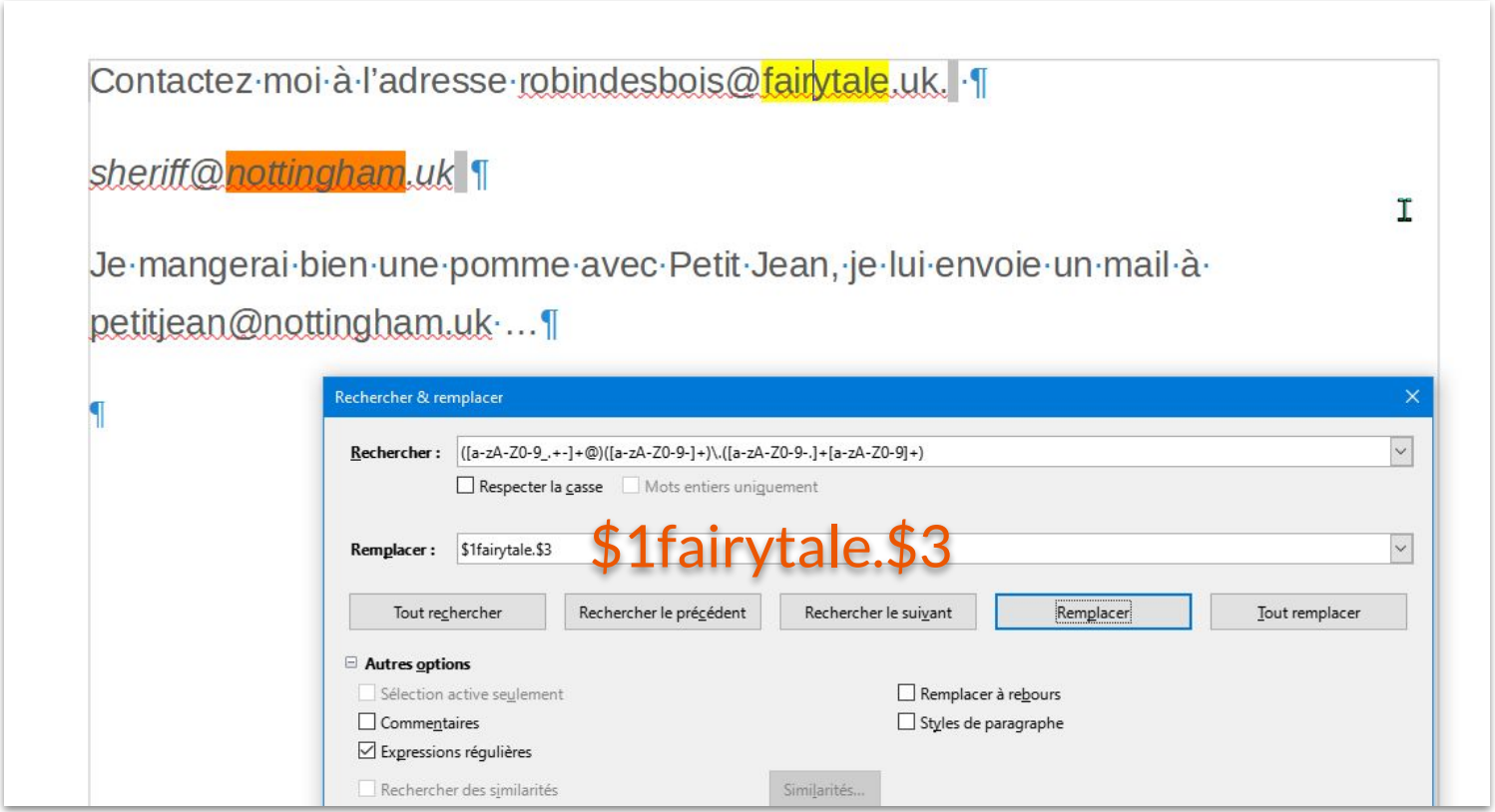
Les regex permettent aussi de “capturer” des blocs à l’intérieur d’une chaîne de caractères et de les remplacer par une autre.

([a-zA-Z0-9_.+-]+@)([a-zA-Z0-9-]+)\.([a-zA-Z0-9-.]+[a-zA-Z0-9]+)

sheriff@nottingham.uk

https://regex101.com/r/xLMplQ/3

Rechercher…remplacer



Comment utiliser les regex ?

Pas de logiciel spécifique mais intégrées dans de nombreux logiciels et langages de programmation :

- commandes Linux (ex. ***grep : global regular expressions print***),
- rechercher/remplacer des **éditeurs de texte**,
- langages de requêtes de bases de données (***SQL***, ***SPARQL***, ***XPath***)
- *Perl*, *C*, *Java*, *PHP*, *Ruby*, *Javascript*, *R*, *Python*…
- utilitaires de traitement de données tels que ***OpenRefine***…

(d'après https://en.wikipedia.org/wiki/Regular_expression)

Avec quoi utiliser les regex ?

Les regex s'appliquent à tout type de stockage de données, tant que que cette dernière n'est pas textuelle (pas de son, ou d'images)

- fichiers .DOC .DOCX
- fichiers .XLS
- fichiers .TXT .CSV .RTF …
- fichiers ou flux XML, HTML
- des données stockées dans des bases de données (MySQL, PostgreSQL)
- des variables d'un script Python ou R…
- etc…

La boîte à outils de base des regex

Les regex appliquées à R

L’utilisation des regex pour travailler des bases de données

- Les bases de données : une ligne par individu, une colonne par variable
- Elles peuvent être utilisées sur des bases plus ou moins propres, générées par un progiciel ou à partir d’“usages ordinaires” de Microsoft Excel et qui n’ont pas a priori vocation à faire l’objet d’analyses statistiques
    - Les regex peuvent servir à scinder des données de nature différentes contenues dans chaque cellule
    - Ou à homogénéiser les formats des valeurs contenues dans chaque variable

Sur des bases de qualités diverses

- Dans des bases où des données sont incomplètes
  - On peut trouver des données manquantes sur les secteurs ou les nationalités des entreprises dans une recherche sur le travail détaché
    - En extrayant ces données du nom de l'entreprise (par déduction) avec des expressions régulières
    - On peut construire d'autres variables : année dans la variable d'identification du dossier, types d'interventions ou de décisions données sous forme de phrases écrites de façon relativement homogène, etc.

Annexe

Différentes syntaxes

La syntaxe change en fonction de l'outil ou du langage utilisé

Wikibooks liste huit syntaxes possibles

Les syntaxes **POSIX** sont surtout utilisées par des logiciels ***Linux*** (grep, sed, awk)

La syntaxe ***Perl-compatible*** est implémentée par la plupart des langages de programmation

Les exemples de ce séminaire sont basés sur la syntaxe Perl-compatible.

Syntaxe PCRE

Une des implémentation de la syntaxe *Perl-compatible* est la librairie PCRE (*Perl Compatible Regular Expressions*), écrite en langage C

Un acronyme que vous pouvez trouver dans les exemples de regex trouvés sur Internet

Les regex ne sont pas toujours la solution

Les expressions régulières ne sont pas la solution à toutes les tâches de manipulation de chaînes de caractères : si la regex est trop complexe, trop gourmande en ressources, il vaut mieux écrire son propre code (Python, R, etc.)

Source : Guide des expressions régulières de la documentation de Python 3.11.3

Caractères et métacaractères

Les métacaractères et quelques-une de leurs utilisations les plus courantes :

. (le point) désigne n'importe quel caractère (excepté le saut de ligne)

**** (barre oblique inversée) combinée avec d'autres lettres désigne des groupes de caractères, ex. **\w** désigne tout caractères de mots (lettres et chiffres), **\t** une tabulation mais **** permet également d'échapper les métacaractères

[] définissent une classe de caractères, ex. **[abcd]** ou **[a-d]**

| permet de définir des alternatives **chien|chat**

{} * + ? sont des quantificateurs, **+** par exemple, désigne un ou plusieurs fois le caractère ou la classe précédente…

Caractères et métacaractères

() : capture d'un groupe de caractères

^ : début de ligne ou entre crochets, défini l'inverse de la classe [**^act**]

$: fin de ligne