



Module 6

Les expressions régulières

IG2 — Technologies web



Les expressions régulières

- **Introduction**

- Qu'est-ce que c'est ?
- À quoi ça sert ?

- **Syntaxe des expressions régulières**

- Comment écrire une expression régulière ?

- **Utilisation en Javascript**

- Comment construire/utiliser une expression régulière en Javascript ?

Introduction

- Une **expression régulière** est une séquence de caractères qui représente un « motif » textuel.
- En anglais
 - *Expression régulière* : regular expression, regexp, regex
 - *Motif* : pattern (correspondance = match)
- Exemples :
 - **b.t** chaînes de 3 caractères b - ? - t
bot, bit, but, bkt, bzt, b3t, b#t...
 - **[01]*** chaînes composées de 0 et de 1
01, 10, 0001111, 01010101, 00110...
 - **\d{4}** toutes les séquences de 4 chiffres
5000, 1024, 0001, 9876...

Introduction

- Utilisation des **expressions régulières** :
 - rechercher des motifs dans un texte,
 - *recupérer tous les numéros de téléphone, compter le nombre d'abréviations*
 - valider une entrée,
 - *adresse mail, url*
 - analyser des données textuelles,
 - *analyse de logs, détection de spams*
 - transformer des données textuelles,
 - *convertir les dates formats anglo/franco-phones*
 - ...

Introduction

- On trouve des outils permettant de manipuler des **expressions régulières**
 - dans la plupart des **langages de programmation** :
 - `java.util.regex.Pattern` en Java,
 - `System.Text.RegularExpressions.Regex` en C#,
 - `RegExp` en Javascript,
 - dans de nombreux **autres langages informatiques** :
 - MySQL,
 - « `grep` » sous Unix...
 - dans de nombreux **éditeurs de texte** :
 - Notepad++, Sublime Text...

Syntaxe

- Les expressions régulières se composent de **symboles** possédant une signification spécifique.

Éléments de motifs	Décorateurs
<ul style="list-style-type: none">- simples<ul style="list-style-type: none">- caractères- caractères échappés- caractères spéciaux- unicode- composites<ul style="list-style-type: none">- classes- classes négatives- classes prédéfinies	<ul style="list-style-type: none">- quantificateurs- disjonction- assertions- groupements

Syntaxe

- Deux remarques
 - Le but de ces slides n'est pas de donner une définition complète de la syntaxe des expressions régulières mais juste un aperçu ; pour cela, voir les notes de laboratoire.
 - Certains langages se distinguent par une sémantique légèrement différente de celle décrite ici, ou par l'ajout d'autres symboles.

Vocabulaire

Syntaxe : règles d'écriture

Sémantique : signification

Syntaxe

- Les expressions régulières se composent de **symboles** qui peuvent être...
 - des **éléments de motifs** ou
(qui correspondent directement à un motif)
 - **a** représente le caractère « a »
 - **\d** représente un chiffre (0, 1, 2, 3, ..., 9)
 - des **décorateurs**
(qui modifient les éléments de motifs)
 - **a+** une suite de 1 ou plusieurs « a »
 - **a|\d** le caractère « a » ou un chiffre

Syntaxe : éléments de motifs

- **Éléments de motifs simples**

- des **caractères** (qui se représentent eux-mêmes)

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

- des **caractères échappés**

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

Vocabulaire

Échappement : fait d'ajouter un symbole avant un autre pour changer sa fonction

- des **caractères spéciaux**

- Exemples : \t (tabulation), \n (retour à la ligne)

- des caractères décrits via **Unicode**

- Exemple : \u00E7 pour « ç »

Syntaxe : éléments de motifs

- **Éléments de motifs composites**

- des **classes** (positives/whitelist ou négatives/blacklist)
 - `[aeiouy]` représente 1 voyelle
 - `[0-9]` représente 1 chiffre
 - `[a-d$]` représente un caractère parmi « a », « b », « c », « d » et « \$ »
 - `.` représente un caractère quelconque (autre qu'un retour à la ligne)
 - `[^aeiouy]` représente 1 caractère qui n'est pas une voyelle
 - `[^A-Z]` représente 1 caractère qui n'est pas une lettre majuscule

Vocabulaire

Whitelist : liste des éléments autorisés

Blacklist : liste des éléments interdits

- des **classes prédéfinies**
 - `\d` représente 1 chiffre ; `\D` représente 1 caractère autre qu'un chiffre
 - `\w` représente 1 caractère alphanumérique
 - `\s` représente 1 « blanc » (espace, retour à la ligne, tabulation...)

Syntaxe : décorateurs

- **Décorateurs** (1/2)

- des **quantificateurs** (répétitions)

- `\d-?\d` : 2 chiffres avec, éventuellement, un tiret entre les deux
 - `[A-Z]*` : séquence de 0, 1 ou plusieurs lettres majuscules
 - `\w+` : séquence de 1 ou plusieurs caractères alphanumériques
 - `[13579]{3}` : séquence de 3 chiffres impairs
 - `[^\.]{4,8}` : séquence de 4 à 8 caractères autres que le point

- la **disjonction** (« ou ») :

- `\d|Z` : un chiffre ou la lettre « Z »
 - `Tic|Tac` : la chaîne « Tic » ou la chaîne « Tac »

Syntaxe : décorateurs

- **Décorateurs (2/2)**

- des **assertions** (restrictions sur l'endroit où le motif est recherché)
 - `^\d` : un chiffre qui se trouve au début d'une ligne
 - `zoo$` : un mot « zoo » situé en fin de ligne
 - `\bpoly` : la chaîne « poly » située au début d'un mot
- des **groupements** (sous-expressions)
 - `\w+(al|aux)` : chaîne terminée par -al ou -aux
 - `(.)v\1` : chaînes telles que ava, eve, 3v3, !v!
 - `(\d)(\d)\2\1` : palindromes à 4 chiffres

Références

\1 pour le 1^{er} groupe
\2 pour le 2^e groupe
etc.

Regex en Javascript

- Une expression régulière Javascript se compose de deux parties :
 - un **motif** (= séquence de symboles, voire avant)
 - des « **flags** » indiquant comment le motif doit être utilisé
- Flag « **i** » : **ignoreCase**
 - sans : majuscules et minuscules distinctes
 - avec : majuscules et minuscules confondues
- Flag « **g** » : **global**
 - sans : recherche de la 1^{re} occurrence du motif
 - avec : recherche de toutes les occurrences du motif
- Flag « **m** » : **multiline**
 - sans : texte considéré comme 1 seule ligne (^/\$ = début/fin du texte)
 - avec : texte considéré comme plusieurs lignes (^/\$ = début/fin de ligne)

Regex en Javascript

- Javascript propose 2 méthodes pour créer un objet « expression régulière ».

Vocabulaire

Littéral : manière d'écrire une valeur (constante)

- Utiliser un **littéral** : `/motif/flags`

```
let regex = /^l[ae]/i;    « le » ou « la » situé au début du texte
let regex = /^l[ae]/im;   « le » ou « la » situé en début de ligne
```

vérifié lors de la lecture du code (erreurs détectées à la lecture du code)

- Via la **fonction constructrice** `RegExp(motif,flags)`

```
let regex = new RegExp ("^l[ae]", "i");
let regex = new RegExp ("^l[ae]", "im");
```

*vérifié lors de l'exécution seulement (erreurs détectées à l'exécution)
permet l'utilisation de motifs construits par le programme*

Regex en Javascript

- Utilisation des expressions régulières en JS
 - Via les méthodes des expressions régulières
 - `regex.test(chaine)` : existe-t-il une correspondance (booléen) ?
 - `regex.exec(chaine)` : décomposition en groupes
 - Via les méthodes des chaînes de caractères
 - `chaine.search(regex)` : indice d'une correspondance
 - `chaine.match(regex)` : recherche de correspondance
 - `chaine.replace(regex, remplacement)` : remplacement
 - `chaine.split(regex)` : découpe selon des séparateurs
- Descriptions complètes : voir les classes RegExp et String sur https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Global_Objects/

Regexp en Javascript

- Méthodes de `RegExp.prototype`
 - `RegExp.prototype.test(chaine)`
booléen indiquant s'il y a correspondance
 - `/asc/.test("Javascript")` → true
 - `/asc/.test("JavaScript")` → false
 - `RegExp.prototype.exec(chaine)`
tableau reprenant la correspondance et les sous-groupes (ou null si aucune)
 - `/(\d)..(\d)/.exec("ab1cd5e")` → ["1cd5", "1", "5"]
 - `/(\d)..(\d)/.exec("ab1c5e")` → null
 - Note : avec le flag « g », on peut appeler ces méthodes plusieurs fois pour obtenir les correspondances suivantes

Regex en Javascript

- Méthodes de `String.prototype` (1/2)
 - `String.prototype.search(regex)`
position de la correspondance (-1 si aucune)
`"JavaScript".search(/asc/i) → 3`
`"JavaScript".search(/asc/) → -1`
 - `String.prototype.match(regex)`
sans le flag « g » : comme `regex.exec(chaine)`
avec le flag « g » : tableau des correspondances (null si aucune)
`"JavaScript".match(/a(.)/) → ["av", "v"]`
`"JavaScript".match(/A(.)/) → null`
`"JavaScript".match(/a(.)/g) → ["av", "aS"]`
`"JavaScript".match(/A(.)/g) → null`

Regexp en Javascript

- Méthodes de `String.prototype` (2/2)
 - `String.prototype.split(regex)`
découpe la chaîne en sous-chaînes pour des séparateurs correspondant à la regex
`"JavaScript".split(/a/) → ["J", "v", "Script"]`
`"1,23;4 5,6".split(/[;,]/) → ["1", "23", "4", "5", "6"]`
 - `String.prototype.replace(regex, remplacement)`
renvoie une nouvelle chaîne après un rechercher/remplacer
`"Javascript".replace(/a/g, "A") → "JAvAscript"`
`"Javascript".replace(/a(.)/, "$1$1") → "Jvvascript"`
`"Javascript".replace(/a(.)/g, "$1$1") → "Jvvsscript"`

Note : remplacement peut être

- une chaîne (\$1, \$2... pour référencer les sous-groupes, \$& pour référencer toute la correspondance) ou
- une fonction (pour des opérations plus complexes ; voir doc).