Les expressions régulières

Table des matières

I. Contexte

Durée : 45 min

Environnement de travail : Repl.it

Pré-requis : Bases de PHP

Contexte

Nommées aussi « expressions rationnelles » ou encore « regex » (pour **reg**ular **ex**pression), les expressions régulières trouvent leur application dans l'analyse de texte. Elles seront d'une grande utilité pour, par exemple, vérifier la validité du format d'une adresse e-mail, vérifier le format d'une date ou encore remplacer des chaînes de caractères incluses à l'intérieur d'autres chaînes. Elles sont en réalité un moyen de rechercher des motifs de texte rapidement et efficacement.

II. Présentation et utilité des expressions régulières

Objectifs

- Écrire une expression régulière simple
- Tester des expressions régulières

Mise en situation

Pour rechercher une chaîne de caractères dans une autre chaîne de caractères, PHP possède deux méthodes de recherche : POSIX et PCRE (pour Perl Compatible Regular Expression). C'est cette seconde méthode, plus complète et performante, que nous utiliserons par la suite.

Remarque

Dans la suite de ce cours, les expressions régulières seront nommées **regex**.

Définition **Regex**

Une regex est une chaîne de caractères modélisant un ensemble de chaînes de caractères, en vérifiant des paramètres définis.

Dans les faits, nous allons définir un modèle ou pattern qui sera la chaîne à rechercher et la chaîne sur laquelle appliquer ce pattern.

Exemple

```php
<?php

$subject = 'Comment chercher une chaîne dans une autre ?';
$pattern = '/chaîne/';

preg_match($pattern, $subject, $matches);

var_dump($matches);
```

Résultat :

```
array(1) { [0]=> string(7) "chaîne" }
```

La chaîne `$pattern` est recherchée dans la chaîne `$subject` et retournée dans le tableau `$matches`. Le pattern a bien été trouvé.

- `$subject` est la chaîne cible, celle dans laquelle nous allons chercher.
- `$pattern` est notre modèle, il est entouré par le caractère spécial `/`. Celui-ci est nommé **délimiteur**.
- La fonction PHP `preg_match()` permet de faire le lien entre `$subject` et `$pattern` (elle applique le modèle à la chaîne).

Syntaxe **Appel de preg_match**

`preg_match($pattern, $subject, $matches=null, $flags=null, $offset=0):`

- `$pattern` = chaîne à chercher,
- `$subject` = chaîne dans laquelle chercher,
- `$matches` = tableau dans lequel retourner les résultats (optionnel),
- `$flags` = options pour le retour des résultats (optionnel),
- `offset` = spécifie une position pour le début de la recherche (optionnel).

Syntaxe **Résultat de preg_match**

`preg_match` retourne 1 si la chaîne a été trouvée, et 0 sinon.

Elle enregistre par ailleurs le résultat de la recherche dans le paramètre `$matches` sous la forme d'un tableau, qui contient à la position 0 la chaîne qui a été trouvée.

Attention

`preg_match()` s'arrêtera à la première occurrence trouvée. Et `preg_match_all()` permettra de retourner toutes les occurrences.

Syntaxe **Résultat de preg_match_all**

`preg_match_all` retourne le nombre de fois où la chaîne a été trouvée.

Elle enregistre dans le paramètre `$matches` un tableau qui contient les chaînes qui ont été trouvées.

Exemple

```php
$subject = '0123456789';
$search = '456';
$pattern = "/$search/";

if (preg_match($pattern, $subject)) {
  echo "$search est présent dans $subject";
} else {
  echo "$search n'est pas présent dans $subject";
}
```

Résultat :

```
456 est présent dans 0123456789
```

Exemple

```php
$subject = '0123456789';
$search = '456';
$pattern = "/$search/";

$offset = 5;

if (preg_match($pattern, $subject, $matches, null, $offset)) {
    echo "$search est présent dans ".substr($subject, $offset);
} else {
    echo "$search n'est pas présent dans ".substr($subject, $offset);
}
```

Résultat :

```
456 n'est pas présent dans 56789
```

Le début de la chaîne d'entrée a été décalé de 5 unités (`$offset`) : nous cherchons 456 dans 56789 et, cette fois, il ne s'y trouve pas.

Complément **Les délimiteurs**

Nous pouvons utiliser le caractère spécial de notre choix (`*`, `%`, `#`, `/`, `\`, etc.), il faut simplement que ce soit le même en début et fin de pattern.

```php
$pattern = '#chaîne#';
$pattern = '%chaîne%';
$pattern = '~chaîne~';
```

Complément **Tester les regex**

Les regex ne sont pas simples à écrire et à tester, même avec de l'expérience. Heureusement, des interfaces existent pour nous faciliter la tâche.

Par exemple, vous pouvez utiliser regex101.com[1]. Nous y trouverons de quoi tester nos regex, les explications associées et de nombreux modèles de regex existantes.

Syntaxe **À retenir**

- `preg_match($pattern, $subject)` permet de rechercher une chaîne modèle dans une chaîne de caractères donnée en entrée. Elle nous dira si oui ou non le modèle existe, et pourra éventuellement nous le retourner.

Complément

preg_match()[2]

preg_match_all()[3]

Tester avec regex101[4]

1 https://regex101.com/
2 https://www.php.net/manual/fr/function.preg-match.php
3 https://www.php.net/manual/fr/function.preg-match-all.php
4 https://regex101.com/

III. Exercice : Appliquez la notion

Vous disposez du texte suivant :

```
Alice was beginning to get very tired of sitting by her sister on the bank, and of having
nothing to do: once or twice she had peeped into the book her sister was reading, but it had
no pictures or conversations in it, “and what is the use of a book,” thought Alice “without
pictures or conversations?”.
```

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question 1

[solution n°1 p.23]

Grâce à la fonction PHP adéquate, vérifiez si *Alice* apparaît dans ce texte.

Question 2

[solution n°2 p.23]

Utilisez la fonction PHP `preg_match_all`, afin de compter le nombre de fois où le mot *`Alice`* apparaît dans ce texte.

IV. Groupes et alternatives

Objectifs

- Utiliser les groupes
- Utiliser les alternatives
- Factoriser les regex

Mise en situation

Les regex deviennent puissantes lorsqu'il s'agit de recherches complexes. Comme pour les expressions booléennes, les opérateurs ET et OU peuvent leur être appliqués. Nous parlerons ici de groupes et d'alternatives.

Alternative |

Dans une regex, l'utilisation seule du métacaractère `|` représente une alternative : `pattern1|pattern2` signifie que nous cherchons les occurrences de `pattern1` et de `pattern2`.

Exemple

```php
$subject = 'chaîner enchaîner enchaînement';
$pattern = '/enchaîner|enchaînement/';

preg_match_all($pattern, $subject, $matches);

var_dump($matches);
```

1 https://repl.it/

Résultat :

```
array(1) {
  [0]=>
  array(2) {
    [0]=>
    string(10) "enchaîner"
    [1]=>
    string(13) "enchaînement"
  }
}
```

Ici, nous recherchons les termes "enchaîner" et "enchaînement".

Les groupements

L'utilisation des métacaractères `(` et `)` signifie que nous allons regrouper deux modèles : `pattern(1|2)` sera équivalent à `pattern1|pattern2`. Cela pourrait nous servir, par exemple, à retourner les termes ayant le même préfixe sans avoir à répéter le préfixe autant de fois qu'il apparaît. C'est une factorisation de la regex.

Exemple

```php
$subject = 'chaîner enchaîner enchaînement';
$pattern = '/enchaîn(er|ement)/';

preg_match_all($pattern, $subject, $matches);

var_dump($matches);
```

Résultat :

```
array(2) {
  [0]=>
  array(2) {
    [0]=>
    string(10) "enchaîner"
    [1]=>
    string(13) "enchaînement"
  }
  [1]=>
  array(2) {
    [0]=>
    string(2) "er"
    [1]=>
    string(5) "ement"
  }
}
```

Là aussi, nous recherchons les termes "enchaîner" et "enchaînement", mais nous avons réduit l'écriture en utilisant la notion de groupe.

Complément

Comme vous le voyez dans le résultat, deux regex ont été appliquées :

- la première est `enchaîner|enchaînement`
- la seconde est `er|ement`

C'est un effet de bord de l'utilisation des groupes.

Syntaxe **À retenir**

- La notion d'alternative indique que nous cherchons l'un et l'autre des patterns. Une alternative peut comporter plus de deux patterns : `ab|ac|ad|ae|af`.
- La notion de groupe permet de factoriser les patterns : `ab|ac|ad|ae|af` devient `a(b|c|d|e|f)`.

V. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°3 p.23]

Grâce à la fonction PHP adéquate et aux groupes et alternatives, comptez le nombre de fois où les mots *she* et *the* apparaissent dans le texte.

```
Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversations?”
```

VI. Quantificateurs

Objectifs

- Contrôler le nombre d'apparitions d'un caractère
- Factoriser les regex

Mise en situation

Nous l'avons vu précédemment, certains patterns peuvent revenir plusieurs fois dans la chaîne en entrée. Notre intérêt pourrait être de tous les retourner, mais sans avoir à tous les écrire et sans connaître leur nombre d'apparitions.

Définition **Quantificateur**

Un quantificateur est un symbole permettant de contrôler le nombre d'apparitions d'un caractère ou d'un groupement de caractères.

Quantificateur ?

Le `?` indique que le caractère est optionnel. Il peut apparaître 0 ou 1 fois. En revanche, il ne peut pas apparaître plusieurs fois.

1 https://repl.it/

Exemple

```php
$password = 'aaazerty zerty';
$pattern = '/a?zerty/';

preg_match_all($pattern, $password, $matches);

var_dump($matches);
```

Résultat :

```
array(1) {
  [0]=>
  array(2) {
    [0]=>
    string(6) "azerty"
    [1]=>
    string(5) "zerty"
  }
}
```

Ici, les chaînes "azerty" et "zerty" sont correctes, car "a" peut être présent 0 ou 1 fois. Par contre, "aaazerty" n'est pas correct, car le "a" est présent plusieurs fois.

Quantificateur +

Le + indique que le caractère est obligatoire. Il doit apparaître 1 ou plusieurs fois.

Exemple

Prenons un cas concret : le contrôle du format d'une adresse e-mail.

```php
$subject = 'john@doe.com';
$pattern = '/@+/';

if (preg_match($pattern, $subject)) {
  echo "$subject semble être une adresse email valide";
}
```

Résultat :

```
john@doe.com semble être une adresse email valide
```

Le caractère "@" doit apparaître au moins 1 fois.

Quantificateur *

Le * indique que le caractère est optionnel, il peut apparaître 1 fois. Mais contrairement au ?, il peut aussi apparaître plusieurs fois.

Exemple

```php
$password = 'aaazerty';
$pattern = '/a*zerty/';

preg_match($pattern, $password, $matches);

var_dump($matches);
```

Résultat :

```
array(1) {
  [0]=>
  string(8) "aaazerty"
}
```

Cette fois-ci, la chaîne "aaaazerty" est correcte, car "a" peut être présent plusieurs fois.

Quantificateurs {x} et {x,y}

La notation `{x}` indique une répétition x fois, et `{x,y}` indique une répétition comprise entre x et y fois. L'écriture `{x,}` existe aussi et permet d'indiquer une répétition d'au moins x fois.

Exemple

```php
$password = 'zerty azerty aazerty aaazerty aaaazerty';
$pattern = '/a{2,4}zerty/';

preg_match_all($pattern, $password, $matches);

var_dump($matches);
```

Résultat :

```
array(1) {
  [0]=>
  array(3) {
    [0]=>
    string(7) "aazerty"
    [1]=>
    string(8) "aaazerty"
    [2]=>
    string(9) "aaaazerty"
  }
}
```

Dans ce cas, sont retournées toutes les chaînes possédant entre 2 et 4 fois le caractère "a" suivi de "zerty".

Complément

`?` est équivalent à `{0,1}`

`+` est équivalent à `{1,}`

`*` est équivalent à `{0,}`

Mélanger les quantificateurs avec les groupes et alternatives

Il est possible d'utiliser les quantificateurs avec les groupes et les alternatives, comme ceci :

```php
$subject = 'azerty azazerty boerty erty';
$pattern = '/(az|bo){1,2}erty/';

preg_match_all($pattern, $subject, $matches);

var_dump($matches);
```

Résultat :

```
array(2) {
  [0]=>
  array(3) {
    [0]=>
    string(6) "azerty"
    [1]=>
    string(8) "azazerty"
    [2]=>
    string(6) "boerty"
  }
  [1]=>
  array(3) {
    [0]=>
    string(2) "az"
    [1]=>
    string(2) "az"
    [2]=>
    string(2) "bo"
  }
}
```

Syntaxe **À retenir**

- Les quantificateurs `?`, `+`, `*` et `{x,y}` indiquent le nombre d'apparitions d'une occurrence. Ils peuvent être associés aux groupes et alternatives.

VII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°4 p.23]

Grâce au code PHP adéquat, vérifiez que les trois premiers numéros de téléphone sont corrects et que les deux derniers ne le sont pas.

1. 0102030405
2. 0112030405
3. 01 02 03 04 05
4. 1102030405
5. 01.02.03.04.05

VIII. Classes de caractères

1 https://repl.it/

Objectifs

- Rechercher des ensembles de caractères
- Exclure des ensembles de caractères
- Factoriser les regex

Mise en situation

Pour nous faciliter la tâche, le support PCRE des regex nous met à disposition des éléments de recherche prédéfinis concernant les caractères alphanumériques, permettant ainsi de réduire la taille et d'améliorer la lecture des regex.

Classe de caractères

Les classes peuvent s'écrire de différentes manières : `[]` pour rechercher exactement les caractères indiqués, `[-]` pour définir un intervalle de caractères (permet de ne pas tous les saisir). Et enfin `[^ ]` pour exclure un ensemble de caractères.

Exemple

```php
$password = '123456789 abcdefghijklmnopqrstuvwxyz';
$pattern = '/[789]|[lmnop]/';

preg_match_all($pattern, $password, $matches);

var_dump($matches);
```

Résultat :

```
array(1) {
  [0]=>
  array(8) {
    [0]=>
    string(1) "7"
    [1]=>
    string(1) "8"
    [2]=>
    string(1) "9"
    [3]=>
    string(1) "l"
    [4]=>
    string(1) "m"
    [5]=>
    string(1) "n"
    [6]=>
    string(1) "o"
    [7]=>
    string(1) "p"
  }
}
```

Ici, nous cherchons et obtenons les caractères 7, 8, 9, l, m, n, o, p.

Ensemble de caractères

Dans le cas d'une recherche avec de nombreux caractères, la simple écriture `[]` n'est pas adaptée : `[0123456789]`. Pour cela, il est possible d'écrire `[0-9]`. Cette méthode permet de créer un ensemble de caractères.

Exemple

```php
$password = '123456789 abcdefghijklmnopqrstuvwxyz';
$pattern = '/[7-9]|[l-p]/';

preg_match_all($pattern, $password, $matches);

var_dump($matches);
```

Résultat :

```
array(1) {
  [0]=>
  array(8) {
    [0]=>
    string(1) "7"
    [1]=>
    string(1) "8"
    [2]=>
    string(1) "9"
    [3]=>
    string(1) "l"
    [4]=>
    string(1) "m"
    [5]=>
    string(1) "n"
    [6]=>
    string(1) "o"
    [7]=>
    string(1) "p"
  }
}
```

Le résultat est le même, mais la regex est plus lisible et concise.

Exclusion de caractères

La notation `[^ ]` permet d'exclure un caractère ou un ensemble de caractères d'une recherche.

Exemple

```php
$password = '123456789';
$pattern = '/[^7-9]/';

preg_match_all($pattern, $password, $matches);

var_dump($matches);
```

Résultat :

```
array(1) {
  [0]=>
  array(6) {
    [0]=>
    string(1) "1"
    [1]=>
    string(1) "2"
    [2]=>
    string(1) "3"
    [3]=>
    string(1) "4"
    [4]=>
    string(1) "5"
    [5]=>
    string(1) "6"
  }
}
```

Cette fois, l'ensemble 7-9 n'est pas souhaité `[^7-9]`. Le résultat obtenu est donc l'ensemble des autres caractères, à savoir 0-6.

Classes génériques

Il existe un moyen encore plus court d'écrire certains ensembles de caractères :

- `\d` est l'équivalent de `[0-9]`.
- `\D` est l'équivalent de `[^0-9]`.
- `\w` est l'équivalent de `[a-zA-Z0-9_]`.
- `\W` est l'équivalent de `[^a-zA-Z0-9_]`.

Remarque

`\w` et `\W` recherchent également le `_`.

Attention

Une regex est sensible à la casse. Cela signifie qu'elle vérifiera si les caractères sont en minuscules ou majuscules. C'est pour cela que nous cherchons a-z et A-Z.

Exemple

```php
$string = 'recherche des caracteres autres qu\'alphanumerique !';
$pattern = '/\W/';

preg_match_all($pattern, $string, $matches);

var_dump($matches);
```

Résultat :

```
array(1) {
  [0]=>
  array(7) {
    [0]=>
    string(1) " "
```

```
    [1]=>
    string(1) " "
    [2]=>
    string(1) " "
    [3]=>
    string(1) " "
    [4]=>
    string(1) "'"
    [5]=>
    string(1) " "
    [6]=>
    string(1) "!"
  }
}
```

`\W` permet d'exclure l'ensemble des caractères alphanumériques. Nous obtenons donc les espaces et les autres caractères spéciaux. Cela nous permettrait, par exemple, de vérifier une URL.

Syntaxe **À retenir**

- `[0123456789]` peut s'écrire `[0-9]` mais aussi `\d`.
- `[^0123456789]` peut s'écrire `[^0-9]` mais aussi `\D`.
- L'ensemble des caractères alphanumériques peut s'écrire `\w` ou `\W` pour les exclure.

IX. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°5 p.24]

Grâce à la fonction PHP adéquate, vérifiez que toutes ces adresses e-mails sont correctes :

```
john@doe.fr
john@localhost
john+1@localhost
```

Mais pas celles-là :

```
@doe.fr
john@.fr
```

X. Ancres

Objectifs

- Rechercher en début de chaîne
- Rechercher en fin de chaîne

1 https://repl.it/

Mise en situation

Pour le moment, les recherches se sont concentrées sur des chaînes entières. Des "curseurs" peuvent être utilisés pour chercher en début ou en fin de chaîne. Il s'agit des **ancres**.

Ancre ^

Le métacaractère `^` indique que la regex agira sur le début de la chaîne. Il doit être placé au début de la regex.

Attention **^ (double usage)**

Dans le contexte des classes `[]`, `^` indique une exclusion.

Exemple

```php
$string = '456789 123456789';
$pattern = '/^123/';

preg_match($pattern, $string, $matches);

var_dump($matches);
```

Résultat :

```
array(0) {
}
```

Ici, aucune correspondance n'est trouvée, puisque la section "123" n'est pas au début de la chaîne.

Ancre $

Le métacaractère `$` indique que la regex agira sur la fin de la chaîne. Il doit être placé à la fin de la regex.

Exemple

```php
$string = '123456789 123456789';
$pattern = '/123$/';

preg_match($pattern, $string, $matches);

var_dump($matches);
```

Résultat :

```
array(0) {
}
```

Ici, aucune correspondance n'est trouvée, puisque la section "123" n'est pas à la fin de la chaîne.

Syntaxe **À retenir**

- `^` indique le début de la chaîne, alors que `$` indique la fin de la chaîne.

XI. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°6 p.24]

Grâce à la fonction PHP adéquate, vérifiez que toutes ces adresses e-mails sont correctes :

```
john@doe.fr
john@localhost
john+1@localhost
```

Mais pas celles-là :

```
john+1@localhost+1
+1@doe.fr
@doe.fr
john@.fr
```

XII. Options internes

Objectif

- Modifier le comportement des regex

Mise en situation

Il existe des options de configuration générales fournies par PCRE permettant d'agir directement sur le comportement d'une regex dans son ensemble.

Option "global"

Cette option permet de retourner toutes les occurrences d'une regex. Si elle est absente, la regex sera validée après la première apparition du pattern. Elle est symbolisée par un `g` et se place généralement à la fin.

En PHP, cette fonctionnalité est déjà implémentée au niveau des fonctions `preg_match()` (sans l'option `global`) et `preg_match_all()` (avec l'option `global`).

Exemple

Galerie option "global"

L'option `global` est utilisée dans le second exemple, donc toutes les occurrences "123" sont retournées.

Option "multiline"

Cette option permet d'autoriser les recherches sur plusieurs lignes. Elle est symbolisée par un `m`.

1 https://repl.it/

Exemple

Galerie option "multiline"

L'option `multiline` est utilisée dans le second exemple, donc tous les chiffres sont retournés.

Option "insensitive"

Pour rappel, jusqu'à maintenant, nous écrivions `[a-zA-Z]` pour couvrir l'ensemble des caractères minuscules ou majuscules.

Grâce à cette option, symbolisée par `i`, nous écrirons simplement `[a-z]`. Elle permet d'indiquer à la regex de ne pas vérifier la casse.

Exemple

Galerie option "insensitive"

L'option `insensitive` est utilisée dans le second exemple, donc les lettres en majuscules et minuscules sont retournées.

Remarque

Les options peuvent être enchaînées : `g`, `gi`, `gm`, `gmi`, etc.

Complément

Il existe d'autres options acceptées par les regex (voir le lien "options internes" ci-dessous).

Syntaxe **À retenir**

- L'option **global**, `g`, indique à la regex de retourner toutes les occurrences recherchées.
- L'option **multiline**, `m`, indique à la regex de rechercher le pattern sur plusieurs lignes.
- L'option **insensitive**, `i`, indique à la regex de ne pas respecter la casse.
- Ces options sont généralement placées en fin de regex après le délimiteur : `/pattern/gmi`.

Complément

Options internes[1]

1 https://www.php.net/manual/fr/reference.pcre.pattern.modifiers.php

XIII. Exercice : Appliquez la notion

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

1

Question

[solution n°7 p.25]

Grâce à la fonction PHP adéquate, vérifiez que toutes ces adresses e-mails sont correctes :

```
john@doe.fr
john@localhost
john+1@localhost
John+1@localhost
```

Mais pas celles-là :

```
john+1@localhost+1
+1@doe.fr
@doe.fr
john@.fr
```

Cette fois, vous partirez du pattern `^[a-z][a-z0-9+]*@[a-z]+(\.[a-z]+)?$`. Vous ne pourrez pas le modifier, mais vous pouvez jouer avec les options.

XIV. Auto-évaluation

A. Exercice final

Exercice 1

[solution n°8 p.25]

Exercice

Quelle fonction PHP permet de faire correspondre une expression régulière avec la première occurrence de la recherche dans une chaîne ?

Exercice

Quelle fonction PHP permet de faire correspondre une expression régulière avec toutes les occurrences de la recherche dans une chaîne ?

Exercice

Que contient `$matches` après l'exécution de l'instruction suivante ?

```php
preg_match($pattern, $subject, $matches);
```

- ○ Une chaîne de caractères contenant la première occurrence trouvée
- ○ Un tableau de chaînes de caractères contenant la première occurrence trouvée
- ○ Un tableau de chaînes de caractères contenant toutes les occurrences trouvées

Exercice

1 https://repl.it/

Que retourne la fonction `preg_match_all()` ?

- [ ] Le nombre de résultats qui satisfont le masque complet
- [ ] 1 si le pattern fourni correspond, 0 s'il ne correspond pas
- [ ] `false` si une erreur survient

Exercice

Quels sont les synonymes de `/père|mère/` ?

- [ ] `/p|mère/`
- [ ] `/[p|m]ère/`
- [ ] `/(p|m)ère/`
- [ ] `/{p|m}ère/`

Exercice

Quel quantificateur permet de dire qu'un caractère peut être facultatif ?

Exercice

Quel quantificateur permet de dire qu'un caractère peut apparaître une ou plusieurs fois ?

Exercice

Quelle classe de caractères équivaut à `\w` ?

- [ ] [A-Za-z0-9]
- [ ] [A-Za-z0-9_]
- [ ] [A-Za-z0-9-]
- [ ] [A-z0-9_]

Exercice

Quelle ancre permet de dire que la regex agira sur la fin de la chaîne ?

Exercice

Quelle option permet de définir que la regex doit s'appliquer globalement et insensiblement à la casse ?

B. Exercice : Défi

Pour réaliser cet exercice, vous aurez besoin de travailler sur l'environnement de travail :

[1]

1 https://repl.it/

Question

[solution n°9 p.27]

Grâce à la fonction PHP adéquate, vérifiez que toutes ces URL sont correctes :

```
www.google.fr
http://www.google.fr
https://www.google.fr
https://www.google.fr:8080
https://www.google.fr:8080/
```

Mais pas celles-là :

```
localhost
google.fr
://www.google.fr
https://www.google.fr:
```

La validation doit être insensible à la casse.

Solutions des exercices

p. 6 Solution n°1

```php
<?php

$sentence = 'Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversations?”.';

if (preg_match('/Alice/', $sentence)) {
echo 'Le mot Alice est présent dans la phrase.';
} else {
echo 'Le mot Alice n\'est pas présent';
}
```

Le `if` détermine si le mot *"Alice"* est dans la chaîne "`$sentence`";

p. 6 Solution n°2

```php
<?php

$sentence = 'Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversations?”.';

$count = preg_match_all('/Alice/', $sentence, $matches);

echo 'Le mot Alice est présent '. $count .' fois.';
```

La variable `$count` détermine le nombre de fois qu'il y a le mot “Alice” dans la chaîne “$sentence”;

p. 8 Solution n°3

```php
<?php

$sentence = 'Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, “and what is the use of a book,” thought Alice “without pictures or conversations?”';

if (preg_match_all('/(s|t)he/', $sentence, $matches)) {
    echo 'Les mots "she" et "the" sont présents '.count($matches[0]).' fois.';
} else {
    echo 'Les mots "she" et "the" ne sont pas présents';
}
```

Le if détermine s'il y a des mots qui commencent par "`t`" ou “s” et qui ont pour suite "`he`", ce qui correspond aux mots “the” et "`she`".

p. 11 Solution n°4

```php
<?php

$phoneNumbers = [
    '0102030405',
    '0112030405',
```

```
    '01 02 03 04 05',
    '1102030405',
    '01.02.03.04.05',
];

foreach ($phoneNumbers as $phoneNumber) {
    if (preg_match('/0(1|2|3|4|5|6|7|8|9|0)( ?((0|1|2|3|4|5|6|7|8|9|0)(0|1|2|3|4|5|6|7|8|9|0))){4}/', $phoneNumber)) {
        echo "$phoneNumber est au bon format.".PHP_EOL;
    } else {
        echo "$phoneNumber n'est pas au bon format.".PHP_EOL;
    }
}
```

La boucle parcourt l'intégralité du tableau "`$phoneNumbers`". Le `if` regarde dans un premier temps si le numéro commence bien par un 0 puis si le prochain caractère est bien un chiffre compris 0 et 9.

Ensuite il regarde les autres nombres, s'ils commencent et finissent avec des chiffres compris entre 0 et 9, et il le regarde 4 fois.

Les deux derniers numéros sont faux car le premier nombre ne commence pas par un 0 pour le numéro "1102030405", et pour le numéro "01.02.03.04.05", comme le `if` ne prend en compte que les chiffres, les "`.`" ne sont donc pas autorisés.

p. 15 Solution n°5

```
<?php

$emailAddresses = [
    'john@doe.fr',
    'john@localhost',
    'john+1@localhost',
    '@doe.fr',
    'john@.fr',
];

foreach ($emailAddresses as $emailAddress) {
    if (preg_match('/[\w+]+@\w+(\.\w*)?/', $emailAddress)) {
        echo "$emailAddress est au bon format.".PHP_EOL;
    } else {
        echo "$emailAddress n'est pas au bon format.".PHP_EOL;
    }
}
```

La boucle parcourt l'intégralité du tableau "`$emailAdresses`".

Le `if` vérifie si le mail commence bien par des lettres et des chiffres `(\w)`, ensuite s'il dispose bien d'un "`@`". Le "`\.`" qui se trouve ici "`(\.\w*)` permet au "`.`" de ne plus être considéré comme un caractère spécial mais plutôt comme un simple point. La vérification continue en regardant si le mail finit bien par un "." et des lettres.

p. 17 Solution n°6

```
<?php

$emailAddresses = [
    'john@doe.fr',
    'john@localhost',
    'john+1@localhost',
```

```
    'john+1@localhost+1',
    '+1@doe.fr',
    '@doe.fr',
    'john@.fr',
];

foreach ($emailAddresses as $emailAddress) {
    if (preg_match('/\w[\w+]@(\w(\.\w*)?)$/', $emailAddress)) {
        echo "$emailAddress est au bon format.".PHP_EOL;
    } else {
        echo "$emailAddress n'est pas au bon format.".PHP_EOL;
    }
}
```

La boucle parcourt l'intégralité du tableau "`$emailAdresses`" . Le `if` vérifie si le mail commence bien par des lettres et des chiffres `(\w)` et s'il y a d'autres chiffres ou lettres avant le "`@`" Le "`\.`" qui se trouve ici "`(\.\w*)` permet au "`.`" de ne plus être considéré comme un caractère spécial mais plutôt comme un simple point. La vérification continue en regardant si le mail finit bien par un "`.`" et des lettres.

p. 19 Solution n°7

```
<?php

$emails = [
  'john@doe.fr',
  'john@localhost',
  'john+1@localhost',
  'John+1@localhost',
  'john+1@localhost+1',
  '+1@doe.fr',
  '@doe.fr',
  'john@.fr',
];

foreach ($emails as $email) {
    if (preg_match('/^[a-z][a-z0-9+]*@[a-z]+(\.[a-z]+)?$/i', $email)) {
        echo "$email est au bon format.".PHP_EOL;
    } else {
        echo "$email n'est pas au bon format.".PHP_EOL;
    }
}
```

La boucle parcourt l'intégralité du tableau "`$emailAdresses`"

Le `if` vérifie si le mail commence bien par des lettres et s'il est bien suivi de chiffres ou de lettres avant le "`@`". La vérification continue en regardant si le mail finit bien par un "`.`" et des lettres.

Exercice p. 19 Solution n°8

Exercice

Quelle fonction PHP permet de faire correspondre une expression régulière avec la première occurrence de la recherche dans une chaîne ?

preg_match

Exercice

Quelle fonction PHP permet de faire correspondre une expression régulière avec toutes les occurrences de la recherche dans une chaîne ?

preg_match_all

Exercice

Que contient `$matches` après l'exécution de l'instruction suivante ?

```php
preg_match($pattern, $subject, $matches);
```

- ○ Une chaîne de caractères contenant la première occurrence trouvée
- ◉ Un tableau de chaînes de caractères contenant la première occurrence trouvée
- ○ Un tableau de chaînes de caractères contenant toutes les occurrences trouvées

Exercice

Que retourne la fonction `preg_match_all()` ?

- ☑ Le nombre de résultats qui satisfont le masque complet
- ☐ 1 si le pattern fourni correspond, 0 s'il ne correspond pas
- ☑ `false` si une erreur survient

Exercice

Quels sont les synonymes de `/père|mère/` ?

- ☐ `/p|mère/`
- ☐ `/[p|m]ère/`
- ☑ `/(p|m)ère/`
- ☐ `/{p|m}ère/`

Exercice

Quel quantificateur permet de dire qu'un caractère peut être facultatif ?

?

Exercice

Quel quantificateur permet de dire qu'un caractère peut apparaître une ou plusieurs fois ?

+

Exercice

Quelle classe de caractères équivaut à `\w` ?

- ☐ [A-Za-z0-9]
- ☑ [A-Za-z0-9_]
- ☐ [A-Za-z0-9-]
- ☑ [A-z0-9_]

Exercice

Quelle ancre permet de dire que la regex agira sur la fin de la chaîne ?

$

Exercice

Quelle option permet de définir que la regex doit s'appliquer globalement et insensiblement à la casse ?

gi

p. 21 **Solution n°9**

```php
<?php

$urls = [
    'www.google.fr',
    'http://www.google.fr',
    'https://www.google.fr',
    'https://www.google.fr:8080',
    'https://www.google.fr:8080/',
    'localhost',
    'google.fr',
    '://www.google.fr',
    'https://www.google.fr:',
];

foreach ($urls as $url) {
    if (preg_match('/^([a-z]+:\/\/)?[a-z]*(\.[a-z]*){2}(:\d+)?\/?$/i', $url)) {
        echo "$url est au bon format.".PHP_EOL;
    } else {
        echo "$url n'est pas au bon format.".PHP_EOL;
    }
}
```

La boucle parcourt l'intégralité du tableau "`$urls`" Le if vérifie si l'url commence par des lettres suivi de ":" ainsi que "`//`" mais c'est facultatif car il y a un "`?`" ("`/^([a-z]+:\/\/)?`").

L'url peut aussi simplement commencer avec des lettres suivi d'un point suivi d'autres lettres ("`[a-z]*(\.[a-z]*`") pour finir avec un point et des lettres.

"`(:\d+)?\/?$/i`" montre que l'url peut finir avec ":" suivi de chiffres ainsi qu'un "/", mais cela est facultatif.