



# Introduction au langage de programmation PHP

[Accueil](#) ▶ [Mes cours](#) ▶ [Développement logiciel](#) ▶ [Intro PHP](#) ▶ [Les expressions rationnelles](#) ▶ [Introduction](#)

## Introduction

"Une expression rationnelle (ou expression régulière par traduction de l'anglais regular expression) est en informatique une chaîne de caractères que l'on appelle parfois un motif et qui décrit un ensemble de chaînes de caractères possibles selon une syntaxe précise." (Wikipedia, article [expression rationnelle](#)).

Les [expressions rationnelles](#) vont nous permettre de vérifier la validité d'une chaîne (un mail par exemple), de vérifier si un motif est présent dans une chaîne, ou encore de diviser ou remplacer une chaîne à partir d'un motif. PHP propose plusieurs fonctions utilisant des expressions rationnelles dont la fonction `preg_match()` qui vérifie si une chaîne correspond à un motif exprimé sous forme d'une expression rationnelle. Cette fonction renvoie 1 si la chaîne correspond, sinon 0.

Les regexps s'écrivent entre délimiteurs. Selon la documentation officielle : "Un délimiteur peut être n'importe quel caractère non alpha-numérique autre qu'un backslash ou qu'un espace. Les délimiteurs les plus courants sont les slashes (/), dièses (#) et les tildes (~)".

Voici un premier exemple qui vérifie si une chaîne contient le motif "voiture" :

```
1 <?php
2
3 // regexp_1.php
4 $pattern = '/voiture/';
5 $subject = 'une voiture bleue';
6 echo preg_match($pattern, $subject);
```

Nous testons si la chaîne contient le motif "bleue" ou le motif "rouge" en utilisant le métacaractère | :

```
1 <?php
2
3 // regexp_2.php
4 $pattern = '/bleue|rouge/';
5 $subject = 'une voiture rouge';
6 echo preg_match($pattern, $subject);
```

Nous pouvons aussi indiquer des cardinalités, c'est-à-dire le nombre d'occurrences du motif que nous souhaitons trouver. Les cardinalités peuvent s'exprimer de plusieurs façons :

- {n} : n fois
- {n,m} : de n à m fois
- {n,} : au moins n fois

Nous testons si la chaîne contient 4 fois le motif "z", c'est-à-dire "zzzz" (les occurrences doivent se suivre) :

```
1 <?php
2
3 // regexp_3.php
4 $pattern = '/z{4}/';
5 $subject = 'zzzzrrrraaaa';
6 echo preg_match($pattern, $subject);
```

Pour sélectionner une partie d'un motif, nous utiliserons les parenthèses. Nous testons si la chaîne contient 2 fois le motif "ba", c'est-à-dire "baba", nous sélectionnons le motif avec des parenthèses et lui appliquons une cardinalité :

```
1 <?php
```

```

2
3 // regexp_4.php
4 $pattern = '/(ba){2}/';
5 $subject = 'un baba au rhum';
6 echo preg_match($pattern, $subject);

```

Les cardinalités peuvent aussi s'exprimer sous forme de métacaractères :

- ? est équivalent à {0,1}
- + est équivalent à {1,}
- \* est équivalent à {0,}

Il est possible de définir un ensemble de valeurs possibles en utilisant les métacaractères [ ]. Nous testons si la chaîne contient un caractère appartenant à l'ensemble [aeiouy].

```

1 <?php
2
3 // regexp_5.php
4 $pattern = '/[aeiouy]/';
5 $subject = 'e';
6 echo preg_match($pattern, $subject);

```

Nous testons si la chaîne contient deux caractères successifs appartenant à l'ensemble [aeiouy].

```

1 <?php
2
3 // regexp_6.php
4 $pattern = '/[aeiouy]{2}/';
5 $subject = 'voiture';
6 echo preg_match($pattern, $subject);

```

Le métacaractère ^ placé juste après le crochet ouvrant permet d'introduire une négation. Nous testons si la chaîne ne contient aucun caractère de l'ensemble [aeiouy].

```

1 <?php
2
3 // regexp_7.php
4 $pattern = '/[^aeiouy]/';
5 $subject = 'voiture';
6 echo preg_match($pattern, $subject);

```

Il est possible de définir des intervalles de valeurs (par exemple toutes les lettres entre a et z) en utilisant le métacaractère -. Nous testons si la chaîne contient un caractère entre a et z ou entre A et Z :

```

1 <?php
2
3 // regexp_8.php
4 $pattern = '/[a-zA-Z]/';
5 $subject = 'voiture';
6 echo preg_match($pattern, $subject);

```

Nous testons si la chaîne contient un caractère entre 1 et 9.

```

1 <?php
2
3 // regexp_9.php
4 $pattern = '/[1-9]/';
5 $subject = '457 voitures';
6 echo preg_match($pattern, $subject);

```

Le métacaractère . permet de remplacer n'importe quel caractère. Nous testons si la chaîne contient le motif ba entre deux caractères quelconques :

```

1 <?php
2
3 // regexp_10.php
4 $pattern = '/.ba./';
5 $subject = '1ba456';
6 echo preg_match($pattern, $subject);

```

Jusqu'à maintenant nous avons recherché si une chaîne contenait un certain motif. Nous allons maintenant vérifier si la chaîne entière correspond à ce motif. Pour cela nous allons utiliser deux métacaractères :

- ^ pour marquer le début d'une chaîne
- \$ pour marquer la fin d'une chaîne

Nous testons si la chaîne commence par le motif "une" :

```

1 <?php
2
3 // regexp_11.php
4 $pattern = '/^une/';
5 $subject = 'une voiture bleue';
6 echo preg_match($pattern, $subject);

```

Nous testons si la chaîne se termine par le motif "bleue" :

```

1 <?php
2
3 // regexp_12.php
4 $pattern = '/bleue$/';
5 $subject = 'une voiture bleue';
6 echo preg_match($pattern, $subject);

```

Nous testons si la chaîne correspond exactement au motif "voiture" :

```

1 <?php
2
3 // regexp_13.php
4 $pattern = '/^voiture$/';
5 $subject = 'voiture';
6 echo preg_match($pattern, $subject);

```

Certains caractères spéciaux permettent de préciser des ensembles de valeurs, par exemple :

- \w → équivalent à [a-zA-Z0-9\_]
- \W → équivalent à [^a-zA-Z0-9\_]
- \s → caractère espace
- \S → aucun caractère espace
- \d → équivalent à [0-9]
- \D → équivalent à [^0-9]

Nous testons si une chaîne est un nom de variable correct (le premier caractère doit être un caractère alphabétique ou un underscore, les autres caractères peuvent être alphanumériques ou des underscores).

```

1 <?php
2
3 // regexp_14.php
4 $pattern = '/^[a-zA-Z_]{1}\w*$/';
5 $subject = '_MaVariable456';
6 echo preg_match($pattern, $subject);

```

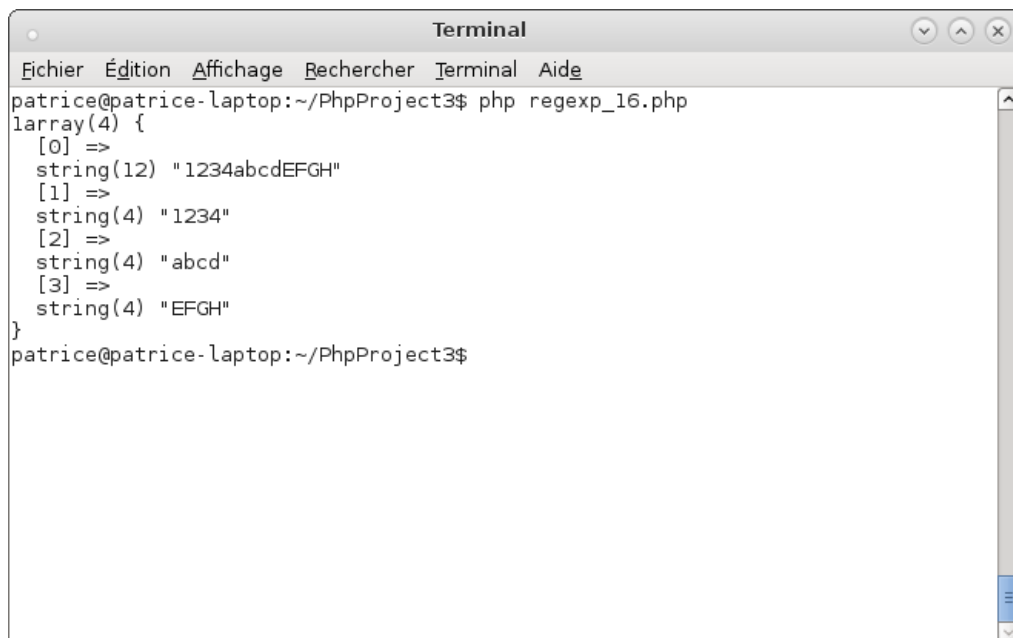
Si l'on souhaite chercher dans une chaîne des occurrences des métacaractères (\ \* + ? ( ) [ ] { } | . ), il est nécessaire d'échapper ces caractères avec un \. Nous vérifions si une chaîne commence par une majuscule et se termine par un point d'interrogation :

```
1 <?php
2
3 // regexp_15.php
4 $pattern = '/^[A-Z]{1}.*\?$/';
5 $subject = 'Est-ce une question ?';
6 echo preg_match($pattern, $subject);
```

Les parenthèses permettent de capturer des parties du motif. Ces "morceaux" de motif seront ajoutés à un tableau passé en argument à la fonction `preg_match()`. Par exemple voici un motif qui teste si une chaîne contient 4 chiffres entre 1 et 9, puis 4 lettres de a à z, puis 4 lettres de A à Z. Les 4 chiffres seront "captés" ainsi que les 4 lettres minuscules et les 4 lettres majuscules.

```
1 <?php
2
3 // regexp_16.php
4 $pattern = '/^([1-9]{4})([a-z]{4})([A-Z]{4})$/';
5 $subject = '1234abcdEFGH';
6 $tab = [];
7 echo preg_match($pattern, $subject, $tab);
8 var_dump($tab);
```

Ce qui affichera :



```
Terminal
Fichier Édition Affichage Rechercher Terminal Aide
patrice@patrice-laptop:~/PhpProject3$ php regexp_16.php
array(4) {
  [0] =>
    string(12) "1234abcdEFGH"
  [1] =>
    string(4) "1234"
  [2] =>
    string(4) "abcd"
  [3] =>
    string(4) "EFGH"
}
patrice@patrice-laptop:~/PhpProject3$
```

À l'index 0 du tableau passé en argument nous trouvons la chaîne entière qui a été captée, à l'index 1 la partie du motif qui correspond au 4 chiffres, à l'index 2 la partie du motif qui correspond au 4 lettres minuscules, à l'index 3 la partie du motif qui correspond au 4 lettres majuscules.

Fin

## NAVIGATION

### Accueil

#### ■ Ma page

Pages du site

Mon profil

Cours actuel

#### Intro PHP


Participants

Le langage PHP : introduction

Les types et les variables

Les opérateurs

Les structures de contrôle

Les structure de données  
Les fonctions  
Les erreurs  
Les fichiers  
Les expressions rationnelles  
 **Introduction**  
PHP et HTML  
Petite application

[Mes cours](#)

## ADMINISTRATION



Administration du cours

Réglages de mon profil

Connecté sous le nom « [Arnaud Lemais](#) » ([Déconnexion](#))  
[Intro PHP](#)