PHP Regular Expressions

In this tutorial you will learn how regular expressions work, as well as how to use them to perform pattern matching in an efficient way in PHP.

What is Regular Expression

Regular Expressions, commonly known as "**regex**" or "**RegExp**", are a specially formatted text strings used to find patterns in text. Regular expressions are one of the most powerful tools available today for effective and efficient text processing and manipulations. For example, it can be used to verify whether the format of data i.e. name, email, phone number, etc. entered by the user was correct or not, find or replace matching string within text content, and so on.

PHP (version 5.3 and above) supports Perl style regular expressions via its `preg_` family of functions. Why Perl style regular expressions? Because Perl (*Practical Extraction and Report Language*) was the first mainstream programming language that provided integrated support for regular expressions and it is well known for its strong support of regular expressions and its extraordinary text processing and manipulation capabilities.

Let's begin with a brief overview of the commonly used PHP's built-in pattern-matching functions before delving deep into the world of regular expressions.

Function	What it Does
`preg_match()`	Perform a regular expression match.
`preg_match_all()`	Perform a global regular expression match.
`preg_replace()`	Perform a regular expression search and replace.
`preg_grep()`	Returns the elements of the input array that matched the pattern.
`preg_split()`	Splits up a string into substrings using a regular expression.
`preg_quote()`	Quote regular expression characters found within a string.

Note: The PHP `preg_match()` function stops searching after it finds the first match, whereas the `preg_match_all()` function continues searching until the end of the string and find all possible matches instead of stopping at the first match.

Regular Expression Syntax

Regular expression syntax includes the use of special characters (do not confuse with the HTML special characters). The characters that are given special meaning within a regular expression,

are: `.` `*` `?` `+` `[` `]` `(` `)` `{` `}` `^` `$` `|` `\`. You will need to backslash these characters whenever you want to use them literally. For example, if you want to match ".", you'd have to write `\.`. All other characters automatically assume their literal meanings.

The following sections describe the various options available for formulating patterns:

Character Classes

Square brackets surrounding a pattern of characters are called a character class e.g. `[abc]`. A character class always matches a single character out of a list of specified characters that means the expression `[abc]` matches only a, b or c character.

Negated character classes can also be defined that match any character except those contained within the brackets. A negated character class is defined by placing a caret (`^`) symbol immediately after the opening bracket, like this `[^abc]`.

You can also define a range of characters by using the hyphen (`-`) character inside a character class, like `[0-9]`. Let's look at some examples of character classes:

RegExp	What it Does
`[abc]`	Matches any one of the characters a, b, or c.
`[^abc]`	Matches any one character other than a, b, or c.
`[a-z]`	Matches any one character from lowercase a to lowercase z.
`[A-Z]`	Matches any one character from uppercase a to uppercase z.
`[a-Z]`	Matches any one character from lowercase a to uppercase Z.
`[0-9]`	Matches a single digit between 0 and 9.
`[a-z0-9]`	Matches a single character between a and z or between 0 and 9.

The following example will show you how to find whether a pattern exists in a string or not using the regular expression and PHP `preg_match()` function:

Example **Run this code »**

```php
<?php
$pattern = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
if(preg_match($pattern, $text)){
    echo "Match found!";
} else{
    echo "Match not found.";
}
?>
```

Similarly, you can use the `preg_match_all()` function to find all matches within a string:

Example | Run this code »

```php
<?php
$pattern = "/ca[kf]e/";
$text = "He was eating cake in the cafe.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
?>
```

Tip: Regular expressions aren't exclusive to PHP. Languages such as Java, Perl, Python, etc. use the same notation for finding patterns in text.

Predefined Character Classes

Some character classes such as digits, letters, and whitespaces are used so frequently that there are shortcut names for them. The following table lists those predefined character classes:

Shortcut	What it Does
.	Matches any single character except newline `\n`.
\d	matches any digit character. Same as `[0-9]`
\D	Matches any non-digit character. Same as `[^0-9]`
\s	Matches any whitespace character (space, tab, newline or carriage return character). Same as `[ \t\n\r]`
\S	Matches any non-whitespace character. Same as `[^ \t\n\r]`
\w	Matches any word character (definned as a to z, A to Z,0 to 9, and the underscore). Same as `[a-zA-Z_0-9]`
\W	Matches any non-word character. Same as `[^a-zA-Z_0-9]`

The following example will show you how to find and replace space with a hyphen character in a string using regular expression and PHP `preg_replace()` function:

Example | Run this code »

```php
<?php
$pattern = "/\s/";
$replacement = "-";
$text = "Earth revolves around\nthe\tSun";
// Replace spaces, newlines and tabs
echo preg_replace($pattern, $replacement, $text);
```

```php
echo "<br>";
// Replace only spaces
echo str_replace(" ", "-", $text);
?>
```

Repetition Quantifiers

In the previous section we've learnt how to match a single character in a variety of fashions. But what if you want to match on more than one character? For example, let's say you want to find out words containing one or more instances of the letter p, or words containing at least two p's, and so on. This is where quantifiers come into play. With quantifiers you can specify how many times a character in a regular expression should match.

The following table lists the various ways to quantify a particular pattern:

RegExp	What it Does
p+	Matches one or more occurrences of the letter p.
p*	Matches zero or more occurrences of the letter p.
p?	Matches zero or one occurrences of the letter p.
p{2}	Matches exactly two occurrences of the letter p.
p{2,3}	Matches at least two occurrences of the letter p, but not more than three occurrences of the letter p.
p{2,}	Matches two or more occurrences of the letter p.
p{,3}	Matches at most three occurrences of the letter p

The regular expression in the following example will splits the string at comma, sequence of commas, whitespace, or combination thereof using the PHP `preg_split()` function:

Example **Run this code »**

```php
<?php
$pattern = "/[\s,]+/";
$text = "My favourite colors are red, green and blue";
$parts = preg_split($pattern, $text);

// Loop through parts array and display substrings
foreach($parts as $part){
    echo $part . "<br>";
}
?>
```

Position Anchors

There are certain situations where you want to match at the beginning or end of a line, word, or string. To do this you can use anchors. Two common anchors are caret (`^`) which represent the start of the string, and the dollar (`$`) sign which represent the end of the string.

RegExp	What it Does
^p	Matches the letter p at the beginning of a line.
p$	Matches the letter p at the end of a line.

The regular expression in the following example will display only those names from the names array which start with the letter "J" using the PHP `preg_grep()` function:

Example **Run this code »**

```php
<?php
$pattern = "/^J/";
$names = array("Jhon Carter", "Clark Kent", "John Rambo");
$matches = preg_grep($pattern, $names);

// Loop through matches array and display matched names
foreach($matches as $match){
    echo $match . "<br>";
}
?>
```

Pattern Modifiers

A pattern modifier allows you to control the way a pattern match is handled. Pattern modifiers are placed directly after the regular expression, for example, if you want to search for a pattern in a case-insensitive manner, you can use the `i` modifier, like this: `/pattern/i`. The following table lists some of the most commonly used pattern modifiers.

Modifier	What it Does
i	Makes the match case-insensitive manner.
m	Changes the behavior of `^` and `$` to match against a newline boundary (i.e. start or end of each line within a multiline string), instead of a string boundary.
g	Perform a global match i.e. finds all occurrences.
o	Evaluates the expression only once.

Modifier	What it Does
s	Changes the behavior of `.` (dot) to match all characters, including newlines.
x	Allows you to use whitespace and comments within a regular expression for clarity.

The following example will show you how to perform a global case-insensitive search using the `i` modifier and the PHP `preg_match_all()` function.

Example **Run this code »**

```php
<?php
$pattern = "/color/i";
$text = "Color red is more visible than color blue in daylight.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
?>
```

Similarly, the following example shows how to match at the beginning of every line in a multi-line string using `^` anchor and `m` modifier with PHP `preg_match_all()` function.

Example **Run this code »**

```php
<?php
$pattern = "/^color/im";
$text = "Color red is more visible than \ncolor blue in daylight.";
$matches = preg_match_all($pattern, $text, $array);
echo $matches . " matches were found.";
?>
```

Word Boundaries

A word boundary character (`\b`) helps you search for the words that begins and/or ends with a pattern. For example, the regexp `/\bcar/` matches the words beginning with the pattern car, and would match cart, carrot, or cartoon, but would not match oscar.

Similarly, the regexp `/car\b/` matches the words ending with the pattern car, and would match scar, oscar, or supercar, but would not match cart. Likewise, the `/\bcar\b/` matches the words beginning and ending with the pattern car, and would match only the word car.

The following example will highlight the words beginning with car in bold:

Example **Run this code »**

```php
<?php
$pattern = '/\bcar\w*/';
$replacement = '<b>$0</b>';
$text = 'Words begining with car: cart, carrot, cartoon. Words ending
with car: scar, oscar, supercar.';
echo preg_replace($pattern, $replacement, $text);
?>
```

We hope you have understood the basics of regular expression. To learn how to validate form data using regular expression, please check out the tutorial on PHP Form Validation.