**PHP Regular Expressions Tutorial: Preg\_match, Preg\_split, Preg\_replace**

**What is a Regular Expressions?**

Regular expressions are powerful pattern matching algorithm that can be performed in a single expression.

Regular expressions use arithmetic operators such as (+,-,^) to create complex expressions.

Regular expressions help you accomplish tasks such as validating email addresses, IP address etc.

**Why to use regular expressions**

* Regular expressions simplify identifying patterns in string data by calling a single function. This saves us coding time.
* When validating user input such as email address, domain names, telephone numbers, IP addresses,
* Highlighting keywords in search results
* When creating a custom HTML template. Regular expressions can be used to identify the template tags and replace them with actual data.

**In this tutorial, you will learn-**

* [Regular expressions in PHP](https://www.guru99.com/php-regular-expressions.html#1)
* [Preg\_match](https://www.guru99.com/php-regular-expressions.html#2)
* [Preg\_split](https://www.guru99.com/php-regular-expressions.html#3)
* [Preg\_replace](https://www.guru99.com/php-regular-expressions.html#4)
* [Meta characters](https://www.guru99.com/php-regular-expressions.html#5)
* [Explaining the pattern](https://www.guru99.com/php-regular-expressions.html#6)

**Regular expressions in PHP**

PHP has built in functions that allow us to work with regular functions. Let’s now look at the commonly used regular expression functions in PHP.

* preg\_match – this function is used to perform a pattern match on a string. It returns true if a match is found and false if a match is not found.
* preg\_split – this function is used to perform a pattern match on a string and then split the results into a numeric array
* preg\_replace – this function is used to perform a pattern match on a string and then replace the match with the specified text.

Below is the syntax for a regular expression function such as preg\_match,preg\_split or preg\_replace.

<?php

function\_name('/pattern/',subject);

?>

HERE,

* "function\_name(...)" is either preg\_match, preg\_split or preg\_replace.
* "/.../" The forward slashes denote the beginning and end of our regular expression
* "'/pattern/'" is the pattern that we need to matched
* "subject" is the text string to be matched against

Let’s now look at practical examples that implement the above regular expression functions in PHP.

**PHP Preg\_match**

The first example uses the preg\_match function to perform a simple pattern match for the word guru in a given URL.

The code below shows the implementation for the above example.

<?php

$my\_url = "www.guru99.com";

if (preg\_match("/guru/", $my\_url))

{

echo "the url $my\_url contains guru";

}

else

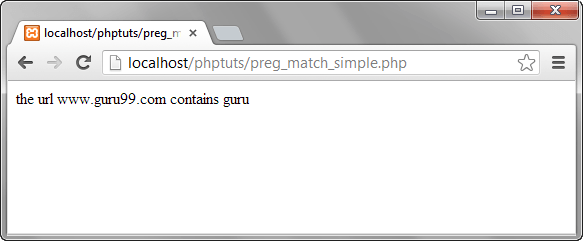
{

echo "the url $my\_url does not contain guru";

}

?>

  Browse to the URL **http://localhost/phptuts/preg\_match\_simple.php**

[](https://www.guru99.com/images/2013/04/preg_match.png)

Let’s examine the part of the code responsible for our output "*preg\_match('/guru/', $my\_url)"*   HERE,

* "preg\_match(...)" is the PHP regular expression function
* "'/guru/'" is the regular expression pattern to be matched
* "$my\_url" is the variable containing the text to be matched against.

The diagram below summarizes the above points

**PHP Preg\_split**

Let’s now look at another example that uses the preg\_split function.

We will take a string phrase and explode it into an array; the pattern to be matched is a single space.

The text string to be used in this example is "I Love Regular Expressions".

The code below illustrates the implementation of the above example.

<?php

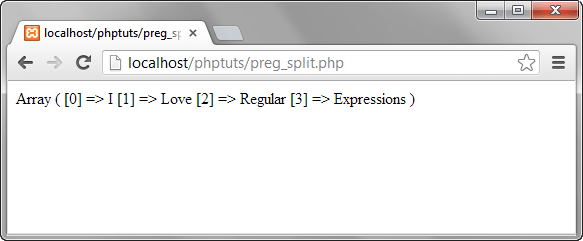
$my\_text="I Love Regular Expressions";

$my\_array = preg\_split("/ /", $my\_text);

print\_r($my\_array );

?>

  Browse to the URL **http://localhost/phptuts/preg\_split.php**

[](https://www.guru99.com/images/2013/04/preg_split.png)

**PHP Preg\_replace**

Let’s now look at the preg\_replace function that performs a pattern match and then replaces the pattern with something else.

The code below searches for the word guru in a string.

It replaces the word guru with the word guru surrounded by css code that highlights the background colour.

<?php

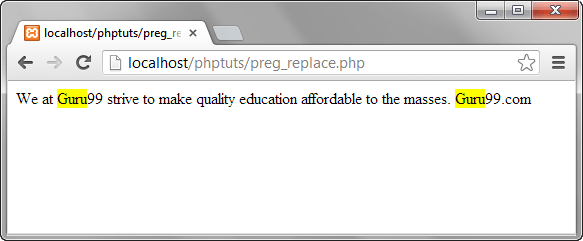
$text = "We at Guru99 strive to make quality education affordable to the masses. Guru99.com";

$text = preg\_replace("/Guru/", '<span style="background:yellow">Guru</span>', $text);

echo $text;

?>

  Assuming you have saved the file preg\_replace.php, browser to the URL **http://localhost/phptuts/preg\_replace.php**

[](https://www.guru99.com/images/2013/04/preg_replace.png)

**Meta characters**

The above examples used very basic patterns; metacharacters simply allow us to perform more complex pattern matches such as test the validity of an email address. Let’s now look at the commonly used metacharacters.

| **Metacharacter** | **Description** | **Example** |
| --- | --- | --- |
| . | Matches any single character except a new line | /./ matches anything that has a single character |
| ^ | Matches the beginning of or string / excludes characters | /^PH/ matches any string that starts with PH |
| $ | Matches pattern at the end of the string | /com$/ matches guru99.com,yahoo.com Etc. |
| \* | Matches any zero (0) or more characters | /com\*/ matches computer, communication etc. |
| + | Requires preceding character(s) appear at least once | /yah+oo/ matches yahoo |
| \ | Used to escape meta characters | /yahoo+\.com/ treats the dot as a literal value |
| [...] | Character class | /[abc]/ matches abc |
| a-z | Matches lower case letters | /a-z/ matches cool, happy etc. |
| A-Z | Matches upper case letters | /A-Z/ matches WHAT, HOW, WHY etc. |
| 0-9 | Matches any number between 0 and 9 | /0-4/ matches 0,1,2,3,4 |

  The above list only gives the most commonly used metacharacters in regular expressions.

Let’s now look at a fairly complex example that checks the validity of an email address.

<?php

$my\_email = "[name@company.com](mailto:name@company.com)";

if (preg\_match("/^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9-]+\.[a-zA-Z.]{2,5}$/", $my\_email)) {

echo "$my\_email is a valid email address";

}

else

{

echo "$my\_email is NOT a valid email address";

}

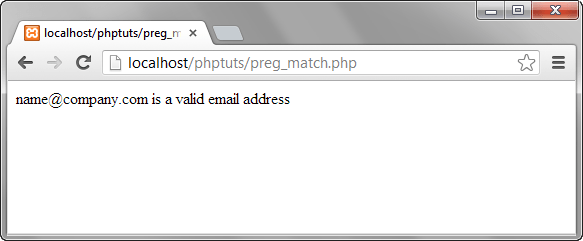
?>

**Explaining the pattern "[/^[a-zA-Z0-9.\_-]+@[a-zA-Z0-9-]+\.[a-zA-Z.]{2,5}$/]"**

HERE,

* "'/.../'" starts and ends the regular expression
* "^[a-zA-Z0-9.\_-]" matches any lower or upper case letters, numbers between 0 and 9 and dots, underscores or dashes.
* "+@[a-zA-Z0-9-]" matches the @ symbol followed by lower or upper case letters, numbers between 0 and 9 or dashes.
* "+\.[a-zA-Z.]{2,5}$/" escapes the dot using the backslash then matches any lower or upper case letters with a character length between 2 and 5 at the end of the string.

Browse to the URL **http://localhost/phptuts/preg\_match.php**

[](https://www.guru99.com/images/2013/04/validate_email.png)

As you can see from the above example breakdown, metacharacters are very powerful when it comes to matching patterns.

**Summary**

* A regular expression is a pattern match algorithm
* Regular expressions are very useful when performing validation checks, creating HTML template systems that recognize tags etc.
* PHP has built in functions namely preg\_match,preg\_split and preg\_replace that support regular expressions.
* Metacharacters allow us to create complex patterns

**Alternatively**

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.

Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.

* POSIX Regular Expressions
* PERL Style Regular Expressions

## POSIX Regular Expressions

The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements (operators) are combined to form more complex expressions.

The simplest regular expression is one that matches a single character, such as g, inside strings such as g, haggle, or bag.

Lets give explanation for few concepts being used in POSIX regular expression. After that we will introduce you with regular expression related functions.

### **Brackets**

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **[0-9]**  It matches any decimal digit from 0 through 9. |
| 2 | **[a-z]**  It matches any character from lower-case a through lowercase z. |
| 3 | **[A-Z]**  It matches any character from uppercase A through uppercase Z. |
| 4 | **[a-Z]**  It matches any character from lowercase a through uppercase Z. |

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

### **Quantifiers**

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, \*, ?, {int. range}, and $ flags all follow a character sequence.

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **p+**  It matches any string containing at least one p. |
| 2 | **p\***  It matches any string containing zero or more p's. |
| 3 | **p?**  It matches any string containing zero or one p's. |
| 4 | **p{N}**  It matches any string containing a sequence of **N** p's |
| 5 | **p{2,3}**  It matches any string containing a sequence of two or three p's. |
| 6 | **p{2, }**  It matches any string containing a sequence of at least two p's. |
| 7 | **p$**  It matches any string with p at the end of it. |
| 8 | **^p**  It matches any string with p at the beginning of it. |

### **Examples**

Following examples will clear your concepts about matching characters.

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **[^a-zA-Z]**  It matches any string not containing any of the characters ranging from a through z and A through Z. |
| 2 | **p.p**  It matches any string containing p, followed by any character, in turn followed by another p. |
| 3 | **^.{2}$**  It matches any string containing exactly two characters. |
| 4 | **<b>(.\*)</b>**  It matches any string enclosed within <b> and </b>. |
| 5 | **p(hp)\***  It matches any string containing a p followed by zero or more instances of the sequence php. |

### **Predefined Character Ranges**

For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set −

|  |  |
| --- | --- |
| **Sr.No** | **Expression & Description** |
| 1 | **[[:alpha:]]**  It matches any string containing alphabetic characters aA through zZ. |
| 2 | **[[:digit:]]**  It matches any string containing numerical digits 0 through 9. |
| 3 | **[[:alnum:]]**  It matches any string containing alphanumeric characters aA through zZ and 0 through 9. |
| 4 | **[[:space:]]**  It matches any string containing a space. |

## PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions −

|  |  |
| --- | --- |
| **Sr.No** | **Function & Description** |
| 1 | [**ereg()**](https://www.tutorialspoint.com/php/php_ereg.htm)  The ereg() function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise. |
| 2 | [**ereg\_replace()**](https://www.tutorialspoint.com/php/php_ereg_replace.htm)  The ereg\_replace() function searches for string specified by pattern and replaces pattern with replacement if found. |
| 3 | [**eregi()**](https://www.tutorialspoint.com/php/php_eregi.htm)  The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive. |
| 4 | [**eregi\_replace()**](https://www.tutorialspoint.com/php/php_eregi_replace.htm)  The eregi\_replace() function operates exactly like ereg\_replace(), except that the search for pattern in string is not case sensitive. |
| 5 | [**split()**](https://www.tutorialspoint.com/php/php_split.htm)  The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string. |
| 6 | [**spliti()**](https://www.tutorialspoint.com/php/php_spliti.htm)  The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive. |
| 7 | [**sql\_regcase()**](https://www.tutorialspoint.com/php/php_sql_regcase.htm)  The sql\_regcase() function can be thought of as a utility function, converting each character in the input parameter string into a bracketed expression containing two characters. |

## PERL Style Regular Expressions

Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions. In fact, you can use any of the quantifiers introduced in the previous POSIX section.

Lets give explanation for few concepts being used in PERL regular expressions. After that we will introduce you wih regular expression related functions.

### **Meta characters**

A meta character is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for large money sums using the '\d' meta character: **/([\d]+)000/**, Here **\d** will search for any string of numerical character.

Following is the list of meta characters which can be used in PERL Style Regular Expressions.

**Character Description**

. a single character

\s a whitespace character (space, tab, newline)

\S non-whitespace character

\d a digit (0-9)

\D a non-digit

\w a word character (a-z, A-Z, 0-9, \_)

\W a non-word character

[aeiou] matches a single character in the given set

[^aeiou] matches a single character outside the given set

(foo|bar|baz) matches any of the alternatives specified

### **Modifiers**

Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

**Modifier Description**

i Makes the match case insensitive

m Specifies that if the string has newline or carriage

return characters, the ^ and $ operators will now

match against a newline boundary, instead of a

string boundary

o Evaluates the expression only once

s Allows use of . to match a newline character

x Allows you to use white space in the expression for clarity

g Globally finds all matches

cg Allows a search to continue even after a global match fails

## PHP's Regexp PERL Compatible Functions

PHP offers following functions for searching strings using Perl-compatible regular expressions −

|  |  |
| --- | --- |
| **Sr.No** | **Function & Description** |
| 1 | [**preg\_match()**](https://www.tutorialspoint.com/php/php_preg_match.htm)  The preg\_match() function searches string for pattern, returning true if pattern exists, and false otherwise. |
| 2 | [**preg\_match\_all()**](https://www.tutorialspoint.com/php/php_preg_match_all.htm)  The preg\_match\_all() function matches all occurrences of pattern in string. |
| 3 | [**preg\_replace()**](https://www.tutorialspoint.com/php/php_preg_replace.htm)  The preg\_replace() function operates just like ereg\_replace(), except that regular expressions can be used in the pattern and replacement input parameters. |
| 4 | [**preg\_split()**](https://www.tutorialspoint.com/php/php_preg_split.htm)  The preg\_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern. |
| 5 | [**preg\_grep()**](https://www.tutorialspoint.com/php/php_preg_grep.htm)  The preg\_grep() function searches all elements of input\_array, returning all elements matching the regexp pattern. |
| 6 | [**preg\_ quote()**](https://www.tutorialspoint.com/php/php_preg_quote.htm)  Quote regular expression characters |