**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

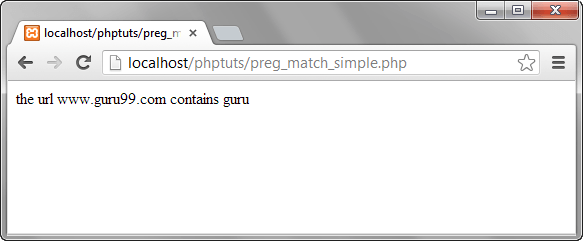
{

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

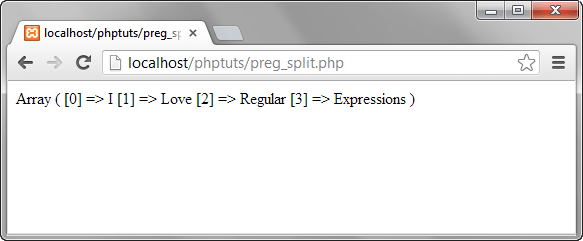
$my\_text="I Love Regular Expressions";

$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

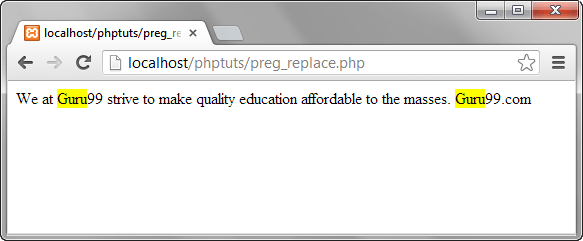
$text = "We at Guru99 strive to make quality education affordable to the masses. Guru99.com";

$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";

}

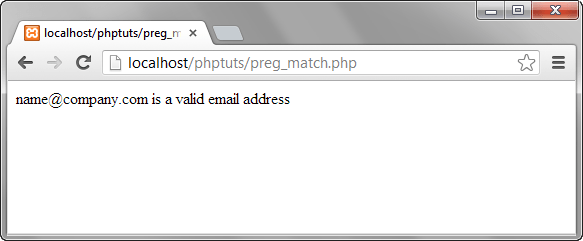
?>

**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