5.5 PHP - Regular Expressions : Basic Pattern Matching

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 explaination for few concepts being used in POSIX regular expression. After that we will introduce you wih 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.

Expression	Description
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[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 pecial character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing zero or more p's. This is just an alternative way to use p*.
p{**N**}	It matches any string containing a sequence of **N** p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Examples

Following examples will clear your concepts about matching chracters.

Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^.{2}$	It matches any string containing exactly two characters.
(.*)	It matches any string enclosed within and .
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp.

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:

Expression	Description
[[:alpha:]]	It matches any string containing alphabetic characters aA through zZ.
[[:digit:]]	It matches any string containing numerical digits 0 through 9.
[[:alnum:]]	It matches any string containing alphanumeric characters aA through zZ and 0 through 9.
[[: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:

Function	Description
ereg()	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.
ereg_replace()	The ereg_replace() function searches for string specified by pattern and replaces pattern with replacement if found.
eregi()	The eregi() function searches throughout a string specified by pattern for a string specified by string. The search is not case sensitive.
eregi_replace()	The eregi_replace() function operates exactly like ereg_replace(), except that the search for pattern in string is not case sensitive.
split()	The split() function will divide a string into various elements, the boundaries of each element based on the occurrence of pattern in string.
spliti()	The spliti() function operates exactly in the same manner as its sibling split(), except that it is not case sensitive.
sql_regcase()	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 explaination for few concepts being used in PERL regular expressions. After that we will introduce you wih regular expression related functions.

Metacharacters

A metacharacter 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' metacharacter:**/([\d]+)000/**, Here **\d** will search for any string of numerical character.

Following is the list of metacharacters 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:

Function	Description
preg_match()	The preg_match() function searches string for pattern, returning true if pattern exists, and false otherwise.
preg_match_all()	The preg_match_all() function matches all occurrences of pattern in string.
preg_replace()	The preg_replace() function operates just like ereg_replace(), except that regular expressions can be used in the pattern and replacement input parameters.
preg_split()	The preg_split() function operates exactly like split(), except that regular expressions are accepted as input parameters for pattern.
preg_grep()	The preg_grep() function searches all elements of input_array, returning all elements matching the regexp pattern.
preg_ quote()	Quote regular expression characters

Example 1: Basic Match Anywhere

```
<?php
// create a string
$string = 'abcdefghijklmnopqrstuvwxyz0123456789';
echo preg_match("/abc/", $string);
?>
```

Example 2: Match From the Begining Of string

```
<?php
// create a string
$string = 'abcdefghijklmnopqrstuvwxyz0123456789';

// try to match the beginning of the string
if(preg_match("/^abc/", $string))
    {
    // if it matches we echo this line
```

```
    echo 'The string begins with abc';
    }
else
    {
    // if no match is found echo this line
    echo 'No match found';
    }
?>
```

Example 3: Case Sensitive

```
<?php
// create a string
$string = 'abcdefghijklmnopqrstuvwxyz0123456789';

// try to match our pattern
if(preg_match("/^ABC/i", $string))
{
    // echo this is it matches
        echo 'The string begins with abc';
 }
 else
 {
    // if not match is found echo this line
        echo 'No match found';
   }
  ?>
```

Example 4: End of String

```
<?php
// create a string
$string = 'abcdefghijklmnopqrstuvwxyz0123456789';
```

```
// try to match our pattern
if(preg_match("/89$/", $string))
{
    // echo this is it matches
        echo 'The string ends with 89';
 }
 else
 {
    // if not match is found echo this line
        echo 'No match found';
  }
  ?>
```

Ref:

http://php.net/manual/en/function.preg-match.php

http://www.tutorialspoint.com/php/php_regular_expression.htm

http://www.phpro.org/tutorials/Introduction-to-PHP-Regex.html

Examples From PHP.NET

Example #1 Find the string of text "php"

```
<?php
// The "i" after the pattern delimiter indicates a case-insensitive search
if (preg_match("/php/i", "PHP is the web scripting language of choice.")) {
    echo "A match was found.";
} else {
    echo "A match was not found.";
}
?>
```

Example #2 Find the word "web"

```
<?php
/* The \b in the pattern indicates a word boundary, so only the distinct
 * word "web" is matched, and not a word partial like "webbing" or "cobweb" */
if (preg_match("/\bweb\b/i", "PHP is the web scripting language of choice.")) {
    echo "A match was found.";
} else {
    echo "A match was not found.";
}

if (preg_match("/\bweb\b/i", "PHP is the website scripting language of choice."))
{
    echo "A match was found.";
} else {
    echo "A match was not found.";
}
?>
```

Example #3 Getting the domain name out of a URL

```
<?php
// get host name from URL
preg_match('@^(?:http://)?([^/]+)@i',
    "http://www.php.net/index.html", $matches);
$host = $matches[1];

// get last two segments of host name
```

```
preg_match('/[^.]+\.[^.]+$/', $host, $matches);
echo "domain name is: {$matches[0]}\n";
?>
```

The above example will output:

```
domain name is: php.net
```

Example #4 Using named subpattern

```
<?php

$str = 'foobar: 2008';

preg_match('/(?P<name>\w+): (?P<digit>\d+)/', $str, $matches);

/* This also works in PHP 5.2.2 (PCRE 7.0) and later, however
 * the above form is recommended for backwards compatibility */
// preg_match('/(?<name>\w+): (?<digit>\d+)/', $str, $matches);

print_r($matches);

?>
```

The above example will output:

```
Array
(
    [0] => foobar: 2008
    [name] => foobar
    [1] => foobar
    [digit] => 2008
    [2] => 2008
)
```