Strings and Regular Expressions

Regular Expressions

Regular expressions provide the foundation for describing or matching data according to defined syntax rules. A regular expression is nothing more than a pattern of characters itself, matched against a certain parcel of text. They provide the foundation for pattern-matching functionality.

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.

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.

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.

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.

PHP’s Regular Expression Functions (POSIX Extended)

PHP offers seven functions for searching strings using POSIX-style regular expressions: ereg(),ereg_replace(), eregi(), eregi_replace(), split(), spliti(), and sql_regcase().

Performing a Case-Sensitive Search

1. The ereg() function executes a case-sensitive search of a string for a defined pattern, returning the length of the matched string if the pattern is found and FALSE otherwise.

```
<?php
$username = "jasoN";
if (ereg("([^a-z])",$username))
echo "Username must be all lowercase!";
else
echo "Username is all lowercase!";
?>
```

Performing a Case-Insensitive Search

The eregi() function searches a string for a defined pattern in a case-insensitive fashion.

```
<?php
$pswd = "jasonasdf";
if (!eregi("^[a-zA-Z0-9]{8,10}$", $pswd))
echo "Invalid password!";
else
echo "Valid password!";
?>
```

Replacing Text in a Case-Sensitive Fashion

The ereg_replace() function operates much like ereg(), except that its power is extended to finding and replacing a pattern with a replacement string instead of simply locating it.

```
<?php
        $num = '4';
        $string = "This string has four words.";
        $string = ereg_replace('four', $num, $string);
        echo $string;   /* Output: 'This string has 4 words.' */

?>
```

Regular Expression Syntax (Perl)

Perl has long been considered one of the most powerful parsing languages ever written. It provides a comprehensive regular expression language that can be used to search, modify, and replace even the most complicated of string patterns.

Searching an Array

The preg_grep() function searches all elements of an array, returning an array consisting of all elements matching a certain pattern.

```
<?php
        $foods = array("pasta", "steak", "fish", "potatoes");
        $food = preg_grep("/^p/", $foods);
        print_r($food);
?>
```

String-Specific Functions

1. strlen() Function:

The PHP function strlen() accomplishes this task quite nicely. This function returns the length of a string, where each character in the string is equivalent to one unit.

```
<?php
$str = 'abcdef';
echo strlen($str); // 6

$str = ' ab cd ';
echo strlen($str); // 7
?>
```

Comparing Two Strings Case Sensitively

2. The strcmp() function performs a binary-safe, case-sensitive comparison of two strings

```
<?php

$var1 = "Hello";
$var2 = "hello";
if (strcmp($var1, $var2) !== 0) {
    echo '$var1 is not equal to $var2 in a case sensitive string comparison';
}

?>
```

Manipulating String Case

Four functions are available to aid you in manipulating the case of characters in a string: strtolower(), strtoupper(), ucfirst(), and ucwords().

The strtolower() function converts a string to all lowercase letters, returning the modified string. Nonalphabetical characters are not affected.

```
<?php

$url = "http://WWW.EXAMPLE.COM/";

echo strtolower($url);

?>
```

```
<?php
$msg = "I annoy people by capitalizing e-mail text.";
echo strtoupper($msg);
?>
```

```
<?php
$sentence = "the newest version of PHP was released today!";
echo ucfirst($sentence);
?>
```

```
<?php
$title = "O'Malley wins the heavyweight championship!";
echo ucwords($title);
?>
```

The nl2br() function converts all newline (\n) characters in a string to their XHTML-compliant equivalent, <br />.

```
<?php
$recipe = "3 tablespoons Dijon mustard
1/3 cup Caesar salad dressing
8 ounces grilled chicken breast
3 cups romaine lettuce";
// convert the newlines to <br />'s.
echo nl2br($recipe);
?>
```

Creating a Customized Conversion List

The strtr() function converts all characters in a string to their corresponding match found in a predefined array.

```
    <?php
    $table = array('<b>' => '<strong>', '</b>' => '</strong>');
    $html = '<b>Today In PHP-Powered News</b>';
    echo strtr($html, $table);
    ?>
```

Tokenizing a String Based on Predefined Characters

The strtok() function parses the string based on a predefined list of characters.

```
    <?php
    $info = "J. Gilmore:jason@example.com|Columbus, Ohio";
    // delimiters include colon (:), vertical bar (|), and comma (,)
    $tokens = ":|,";
    $tokenized = strtok($info, $tokens);
    // print out each element in the $tokenized array
    while ($tokenized) {
    echo "Element = $tokenized<br>";
    // Don't include the first argument in subsequent calls.
    $tokenized = strtok($tokens);
    }
    ?>
```

Exploding a String Based on a Predefined Delimiter

The explode() function divides the string str into an array of substrings.

The original string is divided into distinct elements by separating it based on the character separator specified by separator. The number of elements can be limited with the optional inclusion of limit. Let’s use explode() in conjunction with sizeof() and strip_tags() to determine the total number of words in a given block of text:

```
<?php
$str = "Hello world. It's a beautiful day.";
print_r (explode(" ",$str));
?>
```

Performing Complex String Parsing

The strpos() function finds the position of the first case-sensitive occurrence of a substring in a string.

```
<?php
    echo strpos("I love php, I love php too!","php");
?>
```

- strpos() - Finds the position of the first occurrence of a string inside another string (case-sensitive)
- stripos() - Finds the position of the first occurrence of a string inside another string (case-insensitive)
- strripos() - Finds the position of the last occurrence of a string inside another string (case-insensitive)

The strrpos() function finds the last occurrence of a string, returning its numerical position.

```
<?php
  echo strrpos("I love php, I love php too!","php");
?>
```

The str_replace() function case sensitively replaces all instances of a string with another.

```
<?php
    echo str_replace("world","Peter","Hello world!");
  ?>
```

The strstr() function returns the remainder of a string beginning with the first occurrence of a predefined string.

```
<?php
echo strstr("Hello world!","world");
?
```

The substr() function returns the part of a string located between a predefined starting offset and length positions.

```
<?php
$car = "1944 Ford";
echo substr($car, 5);
?>
```

The following example uses the *length* parameter:

```
<?php
        $car = "1944 Ford";
        echo substr($car, 0, 4);
?>
```

The final example uses a negative *length* parameter:

```
<?php
        $car = "1944 Ford";
        echo substr($car, 2, -5);
?>
```

The substr_replace() function replaces a portion of a string with a replacement string, beginning the substitution at a specified starting position and ending at a predefined replacement length.

```
<?php
$ccnumber = "1234567899991111";
echo substr_replace($ccnumber,"************",0,12);
?>
```