

Using Cookies

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Create Cookies With PHP

A cookie is created with the `setcookie()` function.

Syntax

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable `$_COOKIE`). We also use the `isset()` function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
```

```
</body>
</html>
```

Modify a Cookie Value

To modify a cookie, just set (again) the cookie using the `setcookie()` function:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "Alex Porter";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/");
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>

</body>
</html>
```

Delete a Cookie

To delete a cookie, use the `setcookie()` function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>
```

```
</body>
</html>
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the `setcookie()` function, then count the `$_COOKIE` array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Regular Expressions

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.

Expression	Description
<code>[0-9]</code>	It matches any decimal digit from 0 through 9.
<code>[a-z]</code>	It matches any character from lowercase a through lowercase z.
<code>[A-Z]</code>	It matches any character from uppercase A through uppercase Z.
<code>[a-Z]</code>	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.

Expression	Description
<code>p+</code>	It matches any string containing at least one p.
<code>p*</code>	It matches any string containing zero or more p's.
<code>p?</code>	It matches any string containing zero or more p's. This is just an alternative way to use <code>p*</code> .

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 characters.

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
<u>ereg()</u>	The <code>ereg()</code> function searches a string specified by <code>string</code> for a string specified by <code>pattern</code> , returning true if the pattern is found, and false otherwise.
<u>ereg_replace()</u>	The <code>ereg_replace()</code> function searches for string specified by <code>pattern</code> and replaces <code>pattern</code> with <code>replacement</code> if found.
<u>eregi()</u>	The <code>eregi()</code> function searches throughout a string specified by <code>pattern</code> for a string specified by <code>string</code> . The search is not case sensitive.
<u>eregi_replace()</u>	The <code>eregi_replace()</code> function operates exactly like <code>ereg_replace()</code> , except that the search for <code>pattern</code> in <code>string</code> is not case sensitive.
<u>split()</u>	The <code>split()</code> function will divide a string into various elements, the boundaries of each element based on the occurrence of <code>pattern</code> in <code>string</code> .
<u>spliti()</u>	The <code>spliti()</code> function operates exactly in the same manner as its sibling <code>split()</code> , except that it is not case sensitive.
<u>sql_regcase()</u>	The <code>sql_regcase()</code> function can be thought of as a utility function, converting each character in the input parameter <code>string</code> 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 with 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