What is a Cookie?

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.

How to Create a Cookie?

The setcookie() function is used to set a cookie.

aNote: The setcookie() function must appear BEFORE the <html> tag.

Syntax : setcookie(name, value, expire, path, domain);

Example 1

In the example below, we will create a cookie named "user" and assign the value "Ashish Modi" to it. We also specify that the cookie should expire after one hour:

```
<?php
setcookie("user", "Ashish Modi", time()+3600);
?>
<html>
.....
```

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

Example 2

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

```
<?php
$expire=time()+60*60*24*30;
setcookie("user", "Ashish Modi", $expire);
?>
<html>
.....
```

In the example above the expiration time is set to a month (60 sec * 60 min * 24 hours * 30 days).

How to Retrieve a Cookie Value?

The PHP $_COOKIE variable is used to retrieve a cookie value. In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php
// Print a cookie
echo $_COOKIE["user"];
// A way to view all cookies
print_r($_COOKIE);
?>
```

In the following example we use the isset() function to find out if a cookie has been set:

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

```
<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br />";
else
  echo "Welcome guest!<br />";
?>

</body>
</html>
```

How to Delete a Cookie?

When deleting a cookie you should assure that the expiration date is in the past.

Delete example:

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

PHP Session Variables

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Starting a PHP Session

Before you can store user information in your PHP session, you must first start up the session.

Note: The session_start() function must appear BEFORE the <html> tag:

```
<?php session_start(); ?>
<html>
<body>

</body>
</html>
```

The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

Storing a Session Variable

The correct way to store and retrieve session variables is to use the PHP $_SESSION variable:

```
<?php
session_start();
```

```
// store session data
$_SESSION['views']=1;
?>

<html>
<body>
<?php
//retrieve session data
echo "Pageviews=". $_SESSION['views'];
?>
</body>
</html>
```

Output:

```
Pageviews=1
```

In the example below, we create a simple page-views counter. The isset() function checks if the "views" variable has already been set. If "views" has been set, we can increment our counter. If "views" doesn't exist, we create a "views" variable, and set it to 1:

```
<?php
session_start();

if(isset($_SESSION['views']))
$_SESSION['views']=$_SESSION['views']+1;
else
$_SESSION['views']=1;
echo "Views=". $_SESSION['views'];
?>
```

Destroying a Session

If you wish to delete some session data, you can use the unset() or the session_destroy() function. The unset() function is used to free the specified session variable:

```
<?php
unset($_SESSION['views']);
?>
```

You can also completely destroy the session by calling the session_destroy() function:

```
<?php
session_destroy();
?>
```

Note: session_destroy() will reset your session and you will lose all your stored session data.

$_SERVER variable

This array/variable ($_SERVER) is a replacement of deprecated $HTTP_SERVER_VARS variable in PHP, Which allows us to get information about server and Current execution environment of PHP script. This array/variable stores/contains information such as headers, paths, and script locations and is "superglobal" by default.

Named Constant	Description
'PHP_SELF'	The Filename of Current PHP Page excluding the hostname from URL i.e. "http://example.com/php/phptest.php" will be "/php/phptest.php"
'argv'	Array of arguments passed to the script. When the script is run on the command line, this gives C-style access to the command line parameters. When called via the GET method, this will contain the query string.
'argc'	Contains the number of command line parameters passed to the script (if run on the command line).
'GATEWAY_INTERFACE'	What revision of the CGI specification the server is using; i.e. 'CGI/1.1'.
'SERVER_ADDR'	The IP address of the server under which the current script is executing.
'SERVER_NAME'	The name of the server host under which the current script is executing. If the script is running on a virtual host, this will be the value defined for that virtual host.
'SERVER_SOFTWARE'	Server identification string, given in the headers when responding to requests.
'SERVER_PROTOCOL'	Name and revision of the information protocol via which the page was requested; i.e. 'HTTP/1.0';
'REQUEST_METHOD'	Which request method was used to access the page; i.e. 'GET', 'HEAD', 'POST', 'PUT'.
'REQUEST_TIME'	The timestamp of the start of the request. Available since PHP 5.1.0.
'QUERY_STRING'	The query string, if any, via which the page was accessed.
'DOCUMENT_ROOT'	The document root directory under which the current script is executing, as defined in the server's configuration file.
'HTTP_ACCEPT'	Contents of the Accept: header from the current request, if there is one.
'HTTP_ACCEPT_CHARSET'	Contents of the Accept-Charset: header from the current request, if there is one. Example: 'iso-8859-1,*,utf-8'.
'HTTP_ACCEPT_ENCODING'	Contents of the Accept-Encoding: header from the current request, if there is one. Example: 'gzip'.
'HTTP_ACCEPT_LANGUAGE'	Contents of the Accept-Language: header from the current request, if there is one. Example: 'en'.

Named Constant	Description
'HTTP_CONNECTION'	Contents of the Connection: header from the current request, if there is one. Example: 'Keep-Alive'.
'HTTP_HOST'	Contents of the Host: header from the current request, if there is one.
'HTTP_REFERER'	The address of the page (if any) which referred the user agent to the current page. This is set by the user agent. Not all user agents will set this, and some provide the ability to modify HTTP_REFERER as a feature. In short, it cannot really be trusted.
'HTTP_USER_AGENT'	Contents of the User-Agent: header from the current request, if there is one.
'HTTPS'	Set to a non-empty value if the script was queried through the HTTPS protocol.
'REMOTE_ADDR'	The IP address from which the user is viewing the current page.
'REMOTE_HOST'	The Host name from which the user is viewing the current page. The reverse dns lookup is based off the REMOTE_ADDR of the user.
'REMOTE_PORT'	The port being used on the user's machine to communicate with the web server.
'SCRIPT_FILENAME'	The absolute pathname of the currently executing script.
'SERVER_ADMIN'	The value given to the SERVER_ADMIN (for Apache) directive in the web server configuration file. If the script is running on a virtual host, this will be the value defined for that virtual host.
'SERVER_PORT'	The port on the server machine being used by the web server for communication. For default setups, this will be '80'; using SSL, for instance, will change this to whatever your defined secure HTTP port is.
'SERVER_SIGNATURE'	String containing the server version and virtual host name which are added to server-generated pages, if enabled.
'PATH_TRANSLATED'	Filesystem- (not document root-) based path to the current script, after the server has done any virtual-to-real mapping.
'SCRIPT_NAME'	Contains the current script's path. This is useful for pages which need to point to themselves. The __FILE__ constant contains the full path and filename of the current (i.e. included) file.
'REQUEST_URI'	The URI which was given in order to access this page; for instance, '/index.html'.
'PHP_AUTH_DIGEST'	When doing Digest HTTP authentication this variable is set to the 'Authorization' header sent by the client (which you should then use to make the appropriate validation).
'PHP_AUTH_USER'	When doing HTTP authentication this variable is set to the username provided by the user.

Named Constant	Description
'PHP_AUTH_PW'	When doing HTTP authentication this variable is set to the password provided by the user.
'AUTH_TYPE'	When doing HTTP authenticated this variable is set to the authentication type.
'PATH_INFO'	if the current script was accessed via the URL http://www.example.com/php/path_info.php/some/stuff?foo=bar, then $_SERVER['PATH_INFO'] would contain /some/stuff.
'ORIG_PATH_INFO'	Original version of 'PATH_INFO' before processed by PHP.

Example :

if you're running PHP script shown below on your own computer i.e. on server installed in your computer, then, it will output "localhost" or "127.0.0.1".

```
<?php
echo $_SERVER['SERVER_NAME'];
/*Above PHP statement will output The Domain Name on which you're running this PHP script, for
example: www.example.com */
?>
```

Example 2

PHP Code to Print All Data of $_SERVER into Tabular Format

```
<?php
echo '<table border="1">';

foreach ($_SERVER as $k => $v){
   echo "<tr><td>" . $k ."</td><td>" . $v . "</td></tr>";
}

echo "</table>"
?>
```