isset() and var_dump() functions in PHP

Note: This topic is not a part of cookies/sessions. However, functions isset() and var_dump() are used quite often and is therefore described here.

The var_dump() function is used to dump information about a variable or expression. This function displays the ***datatype & value*** of its argument.

Syntax: void var_dump ($expression)

The function takes a single argument $expression that may be one single variable or an expression containing several space separated variables of any type.

The isset() function is used to check whether a variable is set or not. If a variable is already unset with unset() function, it will no longer be set. The isset() function returns false if testing variable contains a NULL value.

Syntax: isset(variable1, variable2……)

Return value: TRUE if variable (variable1,variable2..) exists and has value not equal to NULL, FALSE otherwise.

isset.php

```
<?php

$a="ip";
$b = NULL;

var_dump($a);
var_dump(isset($a));
var_dump(isset($a, $b));

unset ($a);
var_dump($a);
var_dump(isset($a));

var_dump($b);
var_dump(isset($b));
?>
```

Output:
string(2) "ip" bool(true) bool(false)
Notice: Undefined variable: a in C:\xampp\htdocs\PHP\isset.php on line 11
NULL bool(false) NULL bool(false)

Using Cookies in PHP

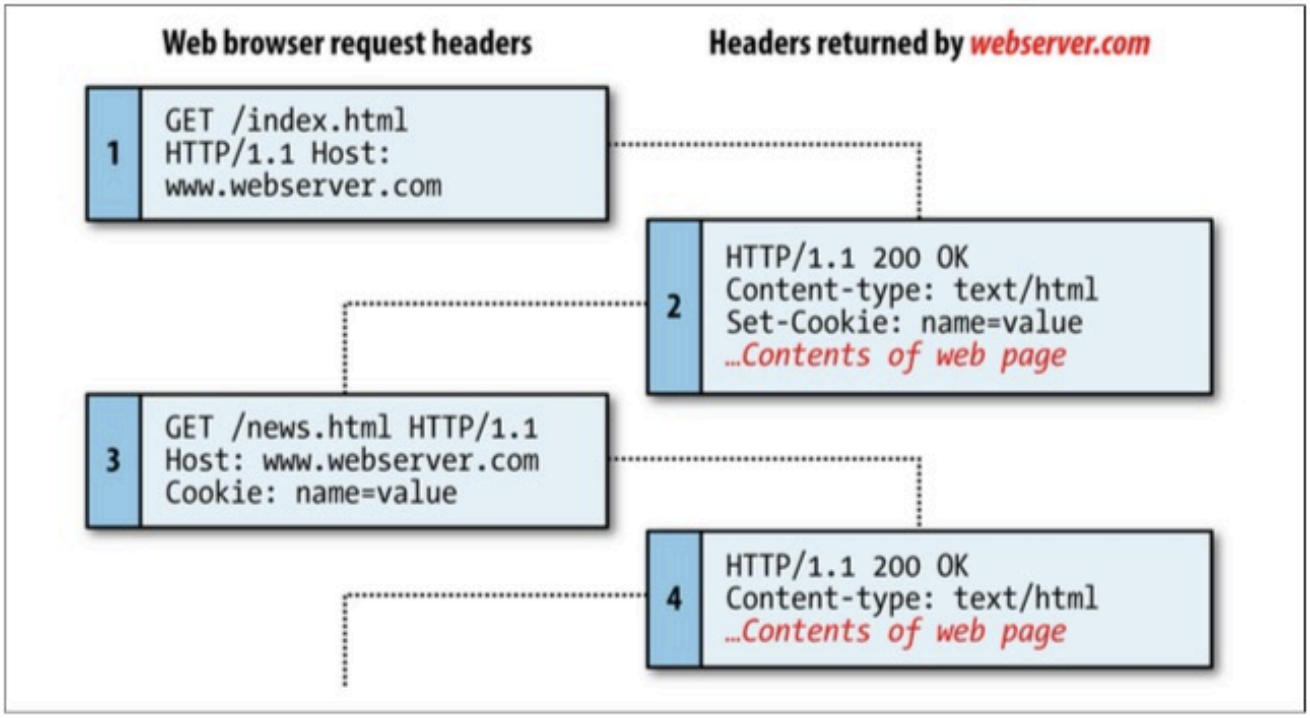
Definition: A cookie is an item of data that a web server saves to your computer's hard disk via a web browser. It can contain almost any alphanumeric information (as long as it's under 4 KB) and can be retrieved from your computer and returned to the server.

Common uses include session tracking, maintaining data across multiple visits, holding shopping cart contents, storing login details, and more.

Because of their privacy implications, cookies can be read only from the issuing domain. In other words, if a cookie is issued by, for example, www.sandeepgupta.org, it can be retrieved only by a web server using that domain. This prevents other websites from gaining access to details for which they are not authorized.

Because of the way the Internet works, multiple elements on a web page can be embedded from multiple domains, each of which can issue its own cookies. When this happens, they are referred to as third-party cookies. Most commonly, these are created by advertising companies in order to track users across multiple websites. Because of this, most browsers allow users to turn cookies off either for the current server's domain, third-party servers, or both. Fortunately, most people who disable cookies do so only for third-party websites.

Cookies are exchanged during the transfer of headers, before the actual HTML of a web page is sent, and it is impossible to send a cookie once any HTML has been transferred. Therefore, careful planning of cookie usage is important. Figure 12-1 illustrates a typical request and response dialog between a web browser and web server passing cookies.



A browser/server request/response dialog with cookies

This exchange shows a browser receiving two pages:

1. The browser issues a request to retrieve the main page, *index.html*, at the website *http://www.webserver.com*. The first header specifies the file, and the second header specifies the server.

2. When the web server at *webserver.com* receives this pair of headers, it returns some of its own. The second header defines the type of content to be sent (text/ html), and the third one sends a cookie of the name *name* and with the value *value*. Only then are the contents of the web page transferred.

3. Once the browser has received the cookie, it will then return it with every future request made to the issuing server until the cookie expires or is deleted. So, when the browser requests the new page */news.html*, it also returns the cookie *name* with the value *value*.

4. Because the cookie has already been set, when the server receives the request to send */news.html*, it does not have to resend the cookie, but just returns the requested page.

A) Setting a Cookie

Setting a cookie in PHP is a simple matter. As long as no HTML has yet been transferred, you can call the setcookie function, which has the following syntax:

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

Parameter	Description	Example
`name`	The name of the cookie. This is the name that your server will use to access the cookie on subsequent browser requests.	`username`
`value`	The value of the cookie, or the cookie's contents. This can contain up to 4 KB of alphanumeric text.	`Hannah`
`expire`	(*Optional.*) Unix timestamp of the expiration date. Generally, you will probably use `time()` plus a number of seconds. If not set, the cookie expires when the browser closes.	`time() + 2592000`
`path`	(*Optional.*) The path of the cookie on the server. If this is a `/` (forward slash), the cookie is available over the entire domain, such as *www.webserver.com*. If it is a subdirectory, the cookie is available only within that subdirectory. The default is the current directory that the cookie is being set in, and this is the setting you will normally use.	`/`
`domain`	(*Optional.*) The Internet domain of the cookie. If this is *.webserver.com*, the cookie is available to all of *webserver.com* and its subdomains, such as *www.webserver.com* and *images.webserver.com*. If it is *images.webserver.com*, the cookie is available only to *images.webserver.com* and its subdomains such as *sub.images.webserver.com*, but not, say, to *www.webserver.com*.	`.webserver.com`
`secure`	(*Optional.*) Whether the cookie must use a secure connection (*https://*). If this value is TRUE, the cookie can be transferred only across a secure connection. The default is FALSE.	`FALSE`
`httponly`	(*Optional*; implemented since PHP version 5.2.0.) Whether the cookie must use the HTTP protocol. If this value is TRUE, scripting languages such as JavaScript cannot access the cookie. (Not supported in all browsers.) The default is FALSE.	`FALSE`

So, to create a cookie with the name username and the value Hannah that is accessible across the entire web server on the current domain, and will be removed from the browser's cache in seven days, use the following:

```
setcookie('username', 'Hannah', time() + 60 * 60 * 24 * 7, '/');
```

B) Accessing a Cookie

Reading the value of a cookie is as simple as accessing the $_COOKIE system array. For example, if you wish to see whether the current browser has the cookie called username already stored and, if so, to read its value, use the following:

```
if (isset($_COOKIE['username'])) $username = $_COOKIE['username'];
```

Note that you can read a cookie back only after it has been sent to a web browser. This means that when you issue a cookie, you cannot read it in again until the browser reloads the page (or another with access to the cookie) from your website and passes the cookie back to the server in the process.

C) Destroying a Cookie

To delete a cookie, you must issue it again and set a date in the past. It is important for all parameters in your new setcookie call except the timestamp to be identical to the parameters when the cookie was first issued; otherwise, the deletion will fail. Therefore, to delete the cookie created earlier, you would use the following:

```
setcookie('username', 'Hannah', time() - 2592000, '/');
```

As long as the time given is in the past, the cookie should be deleted. However, I have used a time of 2,592,000 seconds (one month) in the past in case the client computer's date and time are not correctly set.

cookies1.php

```
<!DOCTYPE html>
<html>
  <head>
    <title>Using Cookies</title>
  </head>
  <body>
    <p>The first time this page loads no cookie has been set and you will see the message NOT SET (meaning cookie is not set). Then the value IPcookie is assigned to cookie mycookie. To see this new cookie's value Refresh the page within 10 seconds. If you Refresh the page again after 10 seconds, you will see NOT SET because by then the cookie has expired</p>
    <?php
      $value = "NOT SET";

      if ( isset($_COOKIE['mycookie']) )
                $value = $_COOKIE['mycookie'];

      echo "<p>$value</p>";

      setcookie('mycookie', 'IPCOOKIE', time()+10);
    ?>
  </body>
</html>
```

HTTP Authentication

auth1.php

```
<?php
  $username = 'jack';
  $password = '123';

  if (isset($_SERVER['PHP_AUTH_USER']) && isset($_SERVER['PHP_AUTH_PW']))
  {
    if ($_SERVER['PHP_AUTH_USER'] == $username &&
        $_SERVER['PHP_AUTH_PW']   == $password)
                echo "You are now logged in";
    else
                die("Invalid username/password combination");
  }
  else
  {
        header('WWW-Authenticate: Basic realm="Restricted Area"');
        header('HTTP/1.0 401 Unauthorized');
        die ("Please enter your username and password");
  }
?>
```

auth2.php

```
<?php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);
  if ($connection->connect_error) die("Fatal Error");

  if (isset($_SERVER['PHP_AUTH_USER']) && isset($_SERVER['PHP_AUTH_PW']))
  {
    $un = $_SERVER['PHP_AUTH_USER'];
    $pw = $_SERVER['PHP_AUTH_PW'];
    $query   = "SELECT * FROM users WHERE username='$un'";
    $result  = $connection->query($query);

    if ($result->num_rows==0)
                die("Invalid username/password combinationn");
    else
    {
        $row = $result->fetch_array(MYSQLI_NUM);
        $result->close();
        if ($pw == $row[2])
                echo  "Hi $row[0], you are now logged in.";
        else
                die("Invalid username/password combination");
    }
  }
  else
  {
    header('WWW-Authenticate: Basic realm="Restricted Area"');
    header('HTTP/1.0 401 Unauthorized');
    die ("Please enter your username and password");
  }
  $connection->close();    ?>
```

PHP Sessions

A PHP Session makes data accessible across the various pages of an entire website for a the current user. A session uses a group of variables that are stored on the server but relate only to the current user.

Starting a PHP Session

A PHP session is easily started by making a call to the session_start() function. This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to session_start() at the beginning of the page.

Session variables are stored in associative array called $_SESSION[]. These variables can be accessed during lifetime of a session.

Destroying a PHP Session

A PHP session can be destroyed by session_destroy() function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use unset() function to unset a session variable.

Here is the example to unset a single variable –

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

Difference between Sessions & Cookies

Sr. No.	Cookies	Sessions
1	Cookies store data in the visitor's browser.	Session data is stored on the server.
2	Cookie can be accessed easily from browser and hence less secure.	Session is more secure than cookie as it is stored in server.
3	Data stored in cookie can be stored for months depending on the life span of the cookie.	Data in the session is lost when the web browser is closed.
4	Maximum cookie size is 4kb.	No size limit for session variables.
5	A cookie comprises of a single data item.	You can have multiple session variables.

session1.html

```
<!DOCTYPE html>
<html>
  <head>
              <title>Session</title>
              <style type="text/css">
          *{
              font-size: 16px;
              font-family: verdana;
              padding: 5px;
          }
        </style>
  </head>

  <body>
    <form action="session1.php" method="post" >
       <input  type="text"  name="username"  size="30"  placeholder="Username"
       autofocus /><br><br>
       <input  type="password"  name="password"  size="30"  placeholder="Password"
       /><br><br>
       <input type="submit" value="Submit" />
    </form>
  </body>
</html>
```

session1.php

```
<?php
  require_once 'login.php';
  $connection = new mysqli($hn, $un, $pw, $db);

  if ($connection->connect_error) die("Fatal Error");

  if ($_POST['username']!="" && $_POST['password']!="")
  {
    $un = $_POST['username'];
    $pw = $_POST['password'];
    $query   = "SELECT * FROM users WHERE username='$un'";
    $result  = $connection->query($query);

    if ($result->num_rows==0)
                die("<p><a href='session1.html'>Invalid username/password
                combination. Click here</a></p>");
    else
    {
       $row = $result->fetch_array(MYSQLI_NUM);
       $result->close();

       if ($pw == $row[2])
        {
                        echo  "Hi $row[0], you are now logged in.";

                        session_start();
                        $_SESSION['session_name'] = $row[0];
                        $_SESSION['session_username'] = $row[1];

                        die ("<p><a href='session2.php'>Click here to
                        continue</a></p>");
        }
       else
                        die("<p><a href='session1.html'>Invalid username/password
                        combination. Click here</a></p>");
    }
  }
  else
        die ("<p><a href='session1.html'>Click here to enter username &
        password</a></p>");

  $connection->close();
?>
```

session2.php

```
<?php
  session_start();

  if (isset($_SESSION['session_name']) && isset($_SESSION['session_username']))
  {
          $name = $_SESSION['session_name'];
          $username = $_SESSION['session_username'];
          echo "Welcome back $name. You are logged in as $username.";
          session_destroy();
  }
  else
        echo "<p><a href='session1.html'>Session terminated. Click here</p>";
        // Reload this page and see what happens!!
?>
```