PHP Advanced

PHP Date and Time

The PHP **date() function is used to format** a date and/or a time.

PHP Include Files

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

PHP include Examples

Example 1

Assume we have a standard footer file called "footer.php", that looks like this:

```
<?php
echo "<p>Copyright &copy; 1999-" . date("Y") . "
W3Schools.com</p>";
?>
```

To include the footer file in a page, use the include statement:

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>

</body>
</html>
```

PHP File Handling

PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language

SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

The PHP code to read the file and write it to the output buffer is as follows (the readfile() function returns the number of bytes read on success):

Example

```
<?php
echo readfile("webdictionary.txt");
?>
```

PHP File Open/Read/Close

PHP Open File - fopen()

A better method to open files is with the fopen() function.

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

PHP Read File - fread()

The fread() function reads from an open file.

PHP Close File - fclose()

The fclose() function is used to close an open file

PHP Close File - fclose()

The fclose() function is used to close an open file

```
echo fgets($myfile);
```

PHP Check End-Of-File - feof()

The feof() function **checks if the "end-of-file" (EOF) has been reached.**

The feof() function is useful for looping through data of unknown length.

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

PHP Read Single Character - fgetc()

The fgetc() function **is used to read a single character from a file.**

PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

$myfile = fopen("testfile.txt", "w")

PHP File Permissions

PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
```

```
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

PHP Append Text

You can append data to a file by using the "a" mode. The "a" mode appends text to the end of the file, while the "w" mode overrides (and erases) the old content of the file.

PHP File Upload

With PHP, it is easy to upload files to the server.

However, with ease comes danger, so always be careful when allowing file uploads!

Configure The "php.ini" File

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the file_uploads directive, and set it to On:

file_uploads = On

Create The HTML Form

```
<!DOCTYPE html>
<html>

<body>

   <form action="upload.php" method="post"
enctype="multipart/form-data">
      Select image to upload:
      <input type="file" name="fileToUpload"
id="fileToUpload">
      <input type="submit" value="Upload Image"
name="submit">
   </form>

</body>

</html>
```

Create The Upload File PHP Script

```php
<?php
$target_dir = "uploads/";
$target_file = $target_dir .
basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType =
strtolower(pathinfo($target_file,PATHINFO_EXTENSI
ON));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
  $check =
getimagesize($_FILES["fileToUpload"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;
  } else {
    echo "File is not an image.";
    $uploadOk = 0;
  }
}
?>
```

PHP script explained:

- $target_dir = "uploads/" - specifies the directory where the file is going to be placed
- $target_file specifies the path of the file to be uploaded
- $uploadOk=1 is not used yet (will be used later)
- $imageFileType holds the file extension of the file (in lower case)

- Next, check if the image file is an actual image or a fake image

Check if File Already Exists

```
// Check if file already exists
<? php
if (file_exists($target_file)) {
  echo "Sorry, file already exists.";
  $uploadOk = 0;
}
?>
```

Limit File Size

```
<? php
if ($_FILES["fileToUpload"]["size"] > 500000) {
    echo "Sorry, your file is too large.";
    $uploadOk = 0;
?>
```

Limit File Type

```
<? php
if($imageFileType != "jpg" && $imageFileType != "png"
&& $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
  echo "Sorry, only JPG, JPEG, PNG & GIF files are
allowed.";
  $uploadOk = 0;
}
```

```
?>
```

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir .
basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType =
strtolower(pathinfo($target_file,PATHINFO_EXTENSI
ON));

// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
  $check =
getimagesize($_FILES["fileToUpload"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;
  } else {
    echo "File is not an image.";
    $uploadOk = 0;
  }
}

// Check if file already exists
if (file_exists($target_file)) {
  echo "Sorry, file already exists.";
  $uploadOk = 0;
}

// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
```

```
  echo "Sorry, your file is too large.";
  $uploadOk = 0;
}

// Allow certain file formats
if($imageFileType != "jpg" && $imageFileType != "png"
&& $imageFileType != "jpeg"
&& $imageFileType != "gif" ) {
  echo "Sorry, only JPG, JPEG, PNG & GIF files are
allowed.";
  $uploadOk = 0;
}

// Check if $uploadOk is set to 0 by an error
if ($uploadOk == 0) {
  echo "Sorry, your file was not uploaded.";
// if everything is ok, try to upload file
} else {
  if
(move_uploaded_file($_FILES["fileToUpload"]["tmp_
name"], $target_file)) {
    echo "The file ". htmlspecialchars( basename(
$_FILES["fileToUpload"]["name"])). " has been
uploaded.";
  } else {
    echo "Sorry, there was an error uploading your file.";
  }
}
?>
```

PHP Cookies

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.

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:

```php
<?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>
```

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

Modify a Cookie Value

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

Delete a Cookie

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

```
<?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:

```
<?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>
```

PHP Sessions

A session is a way to store information (in variables) to be used across multiple pages.

Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work 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 or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc).

By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called **"demo_session1.php".** In this page, we start a new PHP session and set some session variables:

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Get PHP Session Variable Values

Next, we create another page called **"demo_session2.php".** From this page, we will access the session information we set on the first page ("demo_session1.php").

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

<?php
// Echo session variables that were set on previous page
echo "Favorite color is " . $_SESSION["favcolor"] .
".<br>";
echo "Favorite animal is " . $_SESSION["favanimal"] .
".";
?>

</body>
</html>
```

Destroy a PHP Session

To remove all global session variables and destroy the session, use session_unset() and session_destroy():

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

<?php
// remove all session variables
session_unset();

// destroy the session
session_destroy();
?>

</body>
</html>
```

PHP Filters

Validating data = Determine if the data is in proper form.

Sanitizing data = Remove any illegal character from the data.

The PHP Filter Extension

PHP filters are used to validate and sanitize external input.

The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.

The filter_list() function can be used to list what the PHP filter extension offers:

```
<table>
  <tr>
    <td>Filter Name</td>
    <td>Filter ID</td>
  </tr>
  <?php
  foreach (filter_list() as $id =>$filter) {
    echo '<tr><td>' . $filter . '</td><td>' .
filter_id($filter) . '</td></tr>';
  }
  ?>
</table>
```

Why Use Filters?

Many web applications receive external input. External input/data can be:

- User input from a form
- Cookies
- Web services data
- Server variables
- Database query results

PHP filter_var() Function

The filter_var() function both validate and sanitize data.

The filter_var() function filters a single variable with a specified filter. It takes two pieces of data:

- The variable you want to check
- The type of check to use

Sanitize a String

The following example uses the filter_var() function to remove all HTML tags from a string:

```
<?php
$str = "<h1>Hello World!</h1>";
$newstr = filter_var($str,
FILTER_SANITIZE_STRING);
echo $newstr;
?>
```

Validate an Integer

The following example uses the filter_var() function to check if the variable $int is an integer.

```
<?php
$int = 100;

if (!filter_var($int, FILTER_VALIDATE_INT) ===
false) {
  echo("Integer is valid");
} else {
  echo("Integer is not valid");
}
?>
```

Validate an IP Address

The following example uses the filter_var() function to check if the variable $ip is a valid IP address:

```php
<?php
$ip = "127.0.0.1";

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
  echo("$ip is a valid IP address");
} else {
  echo("$ip is not a valid IP address");
}
?>
```

Sanitize and Validate an Email Address

The following example uses the filter_var() function to first remove all illegal characters from the $email variable, then check if it is a valid email address:

```php
<?php
$email = "john.doe@example.com";

// Remove all illegal characters from email
$email = filter_var($email,
FILTER_SANITIZE_EMAIL);

// Validate e-mail
if (!filter_var($email, FILTER_VALIDATE_EMAIL)
=== false) {
  echo("$email is a valid email address");
} else {
  echo("$email is not a valid email address");
}
?>
```

Sanitize and Validate a URL

The following example uses the filter_var() function to first remove all illegal characters from a URL, then check if $url is a valid URL:

```php
<?php
$url = "https://www.w3schools.com";

// Remove all illegal characters from a url
$url = filter_var($url, FILTER_SANITIZE_URL);

// Validate url
if (!filter_var($url, FILTER_VALIDATE_URL) ===
false) {
  echo("$url is a valid URL");
} else {
  echo("$url is not a valid URL");
}
?>
```

Validate an Integer Within a Range

The following example uses the filter_var() function to check if a variable is both of type INT, and between 1 and 200:

```php
<?php
$int = 122;
$min = 1;
$max = 200;

if (filter_var($int, FILTER_VALIDATE_INT,
array("options" => array("min_range"=>$min,
"max_range"=>$max))) === false) {
```

```
  echo("Variable value is not within the legal range");
} else {
  echo("Variable value is within the legal range");
}
?>
```

PHP Callback Functions

Callback Functions

A callback function (often referred to as just "callback") is a function **which is passed as an argument into another function.**

Any existing function can be used as a callback function. To use a function as a callback function, pass a string containing the name of the function as the argument of another function:

```
// Pass a callback to PHP's array_map() function to calculate the length of every string in an array:

<?php
function my_callback($item) {
  return strlen($item);
}

$strings = ["apple", "orange", "banana", "coconut"];
$lengths = array_map("my_callback", $strings);
print_r($lengths);
?>
```

Callbacks in User Defined Functions

User-defined functions and methods can also take callback functions as arguments.

To use callback functions inside a user-defined function or method, call it by adding parentheses to the variable and pass arguments as with normal functions:

```
// Run a callback from a user-defined function:

<?php
function exclaim($str) {
  return $str . "! ";
}

function ask($str) {
  return $str . "? ";
}

function printFormatted($str, $format) {
  // Calling the $format callback function
  echo $format($str);
}

// Pass "exclaim" and "ask" as callback functions to
printFormatted()
printFormatted("Hello world", "exclaim");
printFormatted("Hello world", "ask");
?>
```

PHP and JSON

What is JSON?

JSON stands for **JavaScript Object Notation, and is a syntax for storing and exchanging data.**

Since the JSON format is a text-based format, it can easily be sent to and from a server, and used as a data format by any programming language.

PHP and JSON

PHP has some built-in functions to handle JSON.

First, we will look at the following two functions:

- json_encode()
- json_decode()

PHP - json_encode()

The json_encode() function is used to encode a value to JSON format.

```
<?php
$age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);

echo json_encode($age);
?>
```

PHP - json_decode()

The json_decode() function is used to decode a JSON object into a PHP object or an associative array.

```
<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';
```

```
var_dump(json_decode($jsonobj));
?>
```

What is an Exception?

An exception is an object that **describes an error or unexpected behaviour of a PHP script.**

Throwing an Exception

The throw statement allows a user defined function or method to throw an exception. When an exception is thrown, the code following it will not be executed.

If an exception is not caught, a fatal error will occur with an "Uncaught Exception" message.

```
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

echo divide(5, 0);
?>
```

The try...catch Statement

To avoid the error from the example above, we can use the try...catch statement to catch exceptions and continue the process.

Syntax

```
try {
  code that can throw exceptions
} catch(Exception $e) {
  code that runs when an exception is caught
}
```

```php
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

try {
  echo divide(5, 0);
} catch(Exception $e) {
  echo "Unable to divide.";
}
?>
```

The try...catch...finally Statement

The try...catch...finally statement can be used to catch exceptions. Code in the finally block will always run regardless of whether an exception was caught.
If finally is present, the catch block is optional.

Syntax

```
try {
  code that can throw exceptions
```

```
} catch(Exception $e) {
  code that runs when an exception is caught
} finally {
  code that always runs regardless of whether an
exception was caught
}
```

```php
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
  }
  return $dividend / $divisor;
}

try {
  echo divide(5, 0);
} catch(Exception $e) {
  echo "Unable to divide. ";
} finally {
  echo "Process complete.";
}
?>
```

Methods

When catching an exception, the following table shows some of the methods that can be used to get information about the exception:

Method	Description
getMessage()	Returns a string describing why the exception was thrown
getPrevious()	If this exception was triggered by another one, this method returns the previous exception. If not, then it returns *null*
getCode()	Returns the exception code
getFile()	Returns the full path of the file in which the exception was thrown
getLine()	Returns the line number of the line of code which threw the exception

```php
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero", 1);
  }
  return $dividend / $divisor;
```

```php
}

try {
  echo divide(5, 0);
} catch(Exception $ex) {
  $code = $ex->getCode();
  $message = $ex->getMessage();
  $file = $ex->getFile();
  $line = $ex->getLine();
  echo "Exception thrown in $file on line $line: [Code
$code]
  $message";
}
?>
```