# Able to Design and Develop Dynamic Websites with PHP

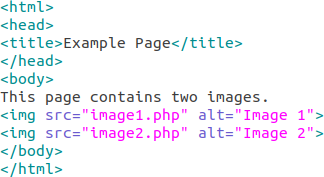
In this section, we will read about:

* Generating images with PHP – Basics of computer graphics
* Creating image, manipulating images using text in images
* PHP Forms, files and cookies
* Form validations
* Introduction to PHP Script
* Looping statement in PHP Script
* Working with Predefined functions
* Maintaining Validations in PHP Script
* Working with Different types of Mouse Events.
* Object Oriented Programming and PHP 5
* Debugging PHP Code
* PHP Session Handling Features
* Handling Date & Time in PHP

# Generating Images with PHP

## Embedding an Image in a Page

A standard web page containing text and graphics is created through a series of HTTP requests from the web browser, each answered by a response from the web server.



## The GD Extension

Image – Embedding image in a page

GD extension allows PHP to use the open source GD graphics library available from [http://www.boutell.com/gd/.](http://www.boutell.com/gd/)



Image – The GD extension

Reference - https://jpgraph.net/download/manuals/chunkhtml/ch03s02.html

## Basics of computer Graphics

An image is a rectangle of pixels that have various colors. Colors are identified by their position in the palette, an array of colors. Each entry in the palette has three separate color values—one for red, one for green, and one for blue. Each value ranges from 0 (this color not present) to 255 (this color at full intensity).

# Creating Image

## Creating An Image with the PHP GD Library

The 3 standard type of images that can be created from scratch with the PHP GD Library is JPG, GIF, and PNG. JPG Is Designed to Compress Full-Color Images, GIF Is Designed to Support Only 256 Colors, PNG Is Designed As An Alternative to GIF, but Does Not Support Animation. To send the image to the browser and complete the code, 3 functions are available for this purpose, one for each type of image.

### imagejpeg() imagegif() imagepng()

The imagedestroy() function clear up the memory that is being taken up by storing the image.

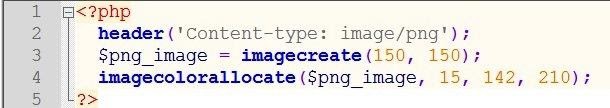


Image – Creating images with GD library

Reference - <http://www.phpforkids.com/php/php-gd-library-create-image.php>

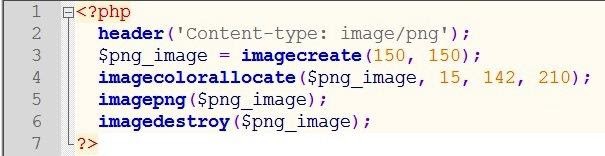


Image – Creating images with GD library

Reference - <http://www.phpforkids.com/php/php-gd-library-create-image.php>

# Manipulating Image

## How to Draw Lines on an Image

The imageline() function itself requires 6 parameters. The syntax is: imageline(image, x1,y1, x2, y2, color)

image = Refers to the Image Resource That the Line Will Be Applied to, x1 = x-coordinate For First Point, y1 = y-coordinate For First Point, x2 = x-coordinate For Second Point, y2 =

y-coordinate For Second Point, color = Refers to the Line Color Identifier Created With imagecolorallocate()

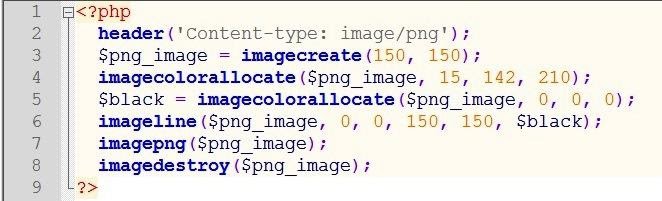


Image – Drawing lines on an image

Reference - <http://www.phpforkids.com/php/php-gd-library-drawing-lines.php>

This example is to draw a thick black border around the entire image.

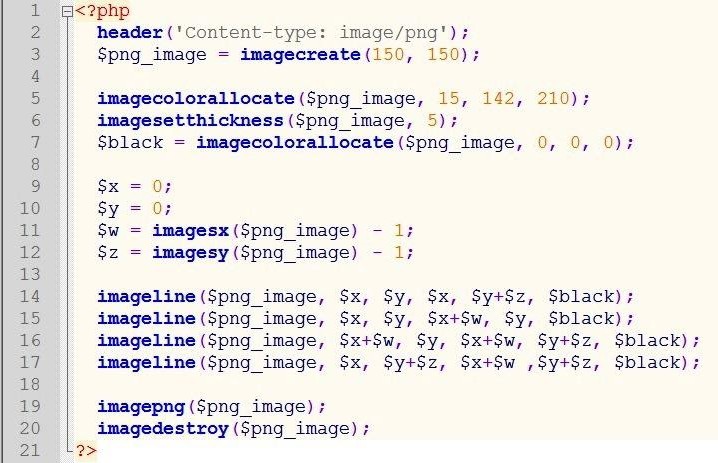


Image – Drawing lines on an image

Reference - <http://www.phpforkids.com/php/php-gd-library-drawing-lines.php>

## How to Draw Shapes On An Image

Use the imagefilledrectangle() function to draw squares and rectangles, specifying the top left and bottom right corner positions. Use the imagefilledellipse() function to draw circles and ellipses, specifying the center position, width, and height of the shape. Use the imagefilledpolygon() function to draw polygons, specifying the three point of the shape, and few more.

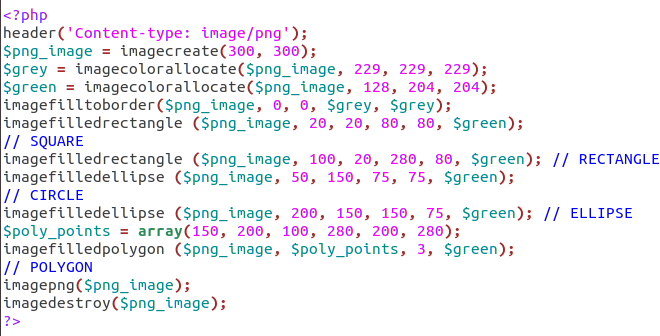


Image – Drawing shapes on an image

## How to Add Image Filters & Effects

The imagefilter() function can be used to apply various fun and/or useful effects to an existing image or photograph.



Image – Adding image filters and effects

# Using text in Image

To add text to an image, you need two things: the image and the font. Three functions are used to create an image from an existing image so that it can be used/edited.



Image – Using text in image

# PHP Forms

## PHP Form Handling

The PHP superglobals $\_GET and $\_POST are used to collect form-data.

### PHP - A Simple HTML Form

The example below displays a simple HTML form with two input fields and a submit button:

Example 1:

<html>

<body>

<form action="welcome.php" method="post">

Name: <input type="text" name="name"><br>

E-mail: <input type="text" name="email"><br>

<input type="submit">

</form>

</body>

</html>

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

<html>

<body>

Welcome <?php echo $\_POST["name"]; ?><br>

Your email address is: <?php echo $\_POST["email"]; ?>

</body>

</html>

The output could be something like this:

Welcome John

Your email address is john.doe@example.com

The same result could also be achieved using the HTTP GET method:

Example 2:

<html>

<body>

<form action="welcome\_get.php" method="get">

Name: <input type="text" name="name"><br>

E-mail: <input type="text" name="email"><br>

<input type="submit">

</form>

</body>

</html>

and "welcome\_get.php" looks like this:

<html>

<body>

Welcome <?php echo $\_GET["name"]; ?><br>

Your email address is: <?php echo $\_GET["email"]; ?>

</body>

</html>

The code above is quite simple. However, the most important thing is missing. You need to validate form data to protect your script from malicious code.

### GET vs. POST

Both GET and POST create an array (e.g. array( key1 => value1, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $\_GET and $\_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$\_GET is an array of variables passed to the current script via the URL parameters.

$\_POST is an array of variables passed to the current script via the HTTP POST method.

### When to use GET?

Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

### When to use POST?

Information sent from a form with the POST method is invisible to others (all names/values are embedded within the body of the HTTP request) and has no limits on the amount of information to send.

Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

## PHP Form Validation

The HTML form we will be working contains various input fields: required and optional text fields, radio buttons, and a submit button:

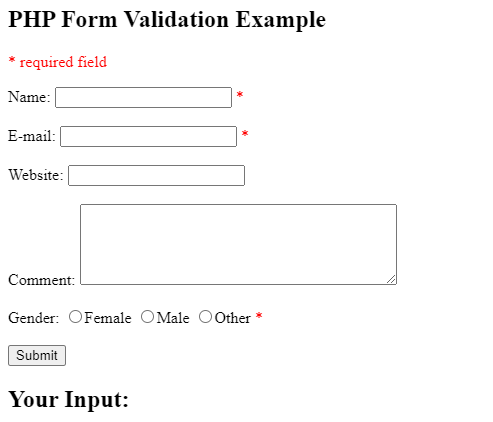


Image – PHP Form validation

Reference - <https://www.w3schools.com/php/php_form_validation.asp>

The validation rules for the form above are as follows:

Field Validation Rules

Name Required. + Must only contain letters and whitespace

E-mail Required. + Must contain a valid email address (with @ and .)

Website Optional. If present, it must contain a valid URL

Comment Optional. Multi-line input field (textarea)

Gender Required. Must select one

First we will look at the plain HTML code for the form:

### Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

Name: <input type="text" name="name">

E-mail: <input type="text" name="email">

Website: <input type="text" name="website">

Comment: <textarea name="comment" rows="5" cols="40"></textarea>

### Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

Gender:

<input type="radio" name="gender" value="female">Female

<input type="radio" name="gender" value="male">Male

<input type="radio" name="gender" value="other">Other

### The Form Element

The HTML code of the form looks like this:

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

When the form is submitted, the form data is sent with method="post".

What is the $\_SERVER["PHP\_SELF"] variable?

The $\_SERVER["PHP\_SELF"] is a super global variable that returns the filename of the currently executing script.

So, the $\_SERVER["PHP\_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

### What is the htmlspecialchars() function?

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Assume we have the following form in a page named "test\_form.php":

<form method="post" action="<?php echo $\_SERVER["PHP\_SELF"];?>">

Now, if a user enters the normal URL in the address bar like "http://www.example.com/test\_form.php", the above code will be translated to:

<form method="post" action="test\_form.php">

So far, so good.

However, consider that a user enters the following URL in the address bar:

http://www.example.com/test\_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E

In this case, the above code will be translated to:

<form method="post" action="test\_form.php/"><script>alert('hacked')</script>

This code adds a script tag and an alert command. And when the page loads, the JavaScript code will be executed (the user will see an alert box).

Validate Form Data With PHP

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.

When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:

<script>location.href('http://www.hacked.com')</script>

- this would not be executed, because it would be saved as HTML escaped code, like this:

&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;

The code is now safe to be displayed on a page or inside an e-mail.

We will also do two more things when the user submits the form:

Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)

Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).

We will name the function test\_input().

Now, we can check each $\_POST variable with the test\_input() function, and the script looks like this:

Example

<?php

// define variables and set to empty values

$name = $email = $gender = $comment = $website = "";

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {

$name = test\_input($\_POST["name"]);

$email = test\_input($\_POST["email"]);

$website = test\_input($\_POST["website"]);

$comment = test\_input($\_POST["comment"]);

$gender = test\_input($\_POST["gender"]);

}

function test\_input($data) {

$data = trim($data);

$data = stripslashes($data);

$data = htmlspecialchars($data);

return $data;

}

?>

Notice that at the start of the script, we check whether the form has been submitted using $\_SERVER["REQUEST\_METHOD"]. If the REQUEST\_METHOD is POST, then the form has been submitted - and it should be validated. If it has not been submitted, skip the validation and display a blank form.

## PHP File Handling

File handling is an important part of any web application. You often need to open and process a file for different tasks.

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

## PHP File Open/Read/Close

### PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

We will use the text file, "webdictionary.txt", during the lessons:

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 first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

<?php

$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");

echo fread($myfile,filesize("webdictionary.txt"));

fclose($myfile);

?>

The file may be opened in one of the following modes:

Modes Description

r Open a file for read only. File pointer starts at the beginning of the file

w Open a file for write only. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file

a Open a file for write only. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist

x Creates a new file for write only. Returns FALSE and an error if file already exists

r+ Open a file for read/write. File pointer starts at the beginning of the file

w+ Open a file for read/write. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file

a+ Open a file for read/write. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist

x+ Creates a new file for read/write. Returns FALSE and an error if file already exists

### PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter

specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

*fread($myfile,filesize("webdictionary.txt"));*

*PHP Close File - fclose()*

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

It's a good programming practice to close all files after you have finished with them. You don't

want an open file running around on your server taking up resources!

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

*<?php*

*$myfile = fopen("webdictionary.txt", "r");*

*// some code to be executed....*

*fclose($myfile);*

*?>*

### PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

Example

*<?php*

*$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");*

*echo fgets($myfile);*

*fclose($myfile);*

*?>*

Note: After a call to the fgets() function, the file pointer has moved to the next line.

### 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.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

*<?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.

The example below reads the "webdictionary.txt" file character by character, until end-of-file is

reached:

Example

*<?php*

*$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");*

*// Output one character until end-of-file*

*while(!feof($myfile)) {*

*echo fgetc($myfile);*

*}*

*fclose($myfile);*

*?>*

Note: After a call to the fgetc() function, the file pointer moves to the next character.

## PHP File Create/Write

### 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).

The example below creates a new file called "testfile.txt". The file will be created in the same

directory where the PHP code resides:

Example

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

### PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP

file access to write information to the hard drive.

### 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.

The example below writes a couple of names into a new file called "newfile.txt":

Example

*<?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);*

*?>*

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string

$txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing,

we closed the file using the fclose() function.

If we open the "newfile.txt" file it would look like this:

John Doe

Jane Doe

## 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:

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>*

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

Note: The value of the cookie is automatically URLencoded when sending the cookie, and

automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).

### 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>*

## 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:

Example

*<?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>*

Note: The session\_start() function must be the very first thing in your document. Before any HTML

tags.

### 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").

Notice that session variables are not passed individually to each new page, instead they are

retrieved from the session we open at the beginning of each page (session\_start()).

Also notice that all session variable values are stored in the global $\_SESSION variable:

Example

*<?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>*

Another way to show all the session variable values for a user session is to run the following code:

Example

*<?php*

*session\_start();*

*?>*

*<!DOCTYPE html>*

*<html>*

*<body>*

*<?php*

*print\_r($\_SESSION);*

*?>*

*</body>*

*</html>*

How does it work? How does it know it's me?

Most sessions set a user-key on the user's computer that looks something like this:

765487cf34ert8dede5a562e4f3a7e12. Then, when a session is opened on another page, it scans the

computer for a user-key. If there is a match, it accesses that session, if not, it starts a new session.

### Modify a PHP Session Variable

To change a session variable, just overwrite it:

Example

*<?php*

*session\_start();*

*?>*

*<!DOCTYPE html>*

*<html>*

*<body>*

*<?php*

*// to change a session variable, just overwrite it*

*$\_SESSION["favcolor"] = "yellow";*

*print\_r($\_SESSION);*

*?>*

*</body>*

*</html>*

### Destroy a PHP Session

To remove all global session variables and destroy the session, use session\_unset() and

session\_destroy():

Example

*<?php*

*session\_start();*

*?>*

*<!DOCTYPE html>*

*<html>*

*<body>*

*<?php*

*// remove all session variables*

*session\_unset();*

*// destroy the session*

*session\_destroy();*

*?>*

*</body>*

*</html>*

## PHP Date/Time Functions

The date/time functions allow you to get the date and time from the server where your PHP script

runs. You can then use the date/time functions to format the date and time in several ways.

Note: These functions depend on the locale settings of your server. Remember to take daylight

saving time and leap years into consideration when working with these functions.

### PHP Date/Time Functions

Function Description

checkdate() Validates a Gregorian date

date\_add() Adds days, months, years, hours, minutes, and seconds to a date

date\_create\_from\_format() Returns a new DateTime object formatted according to a specified format

date\_create() Returns a new DateTime object

date\_date\_set() Sets a new date

date\_default\_timezone\_get() Returns the default timezone used by all date/time functions

date\_default\_timezone\_set() Sets the default timezone used by all date/time functions

date\_diff() Returns the difference between two dates

date\_format() Returns a date formatted according to a specified format

date\_get\_last\_errors() Returns the warnings/errors found in a date string

date\_interval\_create\_from\_date\_string() Sets up a DateInterval from the relative parts of the string

date\_interval\_format() Formats the interval

date\_isodate\_set() Sets the ISO date

date\_modify() Modifies the timestamp

date\_offset\_get() Returns the timezone offset

date\_parse\_from\_format() Returns an associative array with detailed info about a specified date, according to a specified format

date\_parse() Returns an associative array with detailed info about a specified date

date\_sub() Subtracts days, months, years, hours, minutes, and seconds from a date

date\_sun\_info() Returns an array containing info about sunset/sunrise and twilight begin/end, for a specified day and location

date\_sunrise() Returns the sunrise time for a specified day and location

date\_sunset() Returns the sunset time for a specified day and location

date\_time\_set() Sets the time

date\_timestamp\_get() Returns the Unix timestamp

date\_timestamp\_set() Sets the date and time based on a Unix timestamp

date\_timezone\_get() Returns the time zone of the given DateTime object

date\_timezone\_set() Sets the time zone for the DateTime object

date() Formats a local date and time

getdate() Returns date/time information of a timestamp or the current local date/time

gettimeofday() Returns the current time

gmdate() Formats a GMT/UTC date and time

gmmktime() Returns the Unix timestamp for a GMT date

gmstrftime() Formats a GMT/UTC date and time according to locale settings

idate() Formats a local time/date as integer

localtime() Returns the local time

microtime() Returns the current Unix timestamp with microseconds

mktime() Returns the Unix timestamp for a date

strftime() Formats a local time and/or date according to locale settings

strptime() Parses a time/date generated with strftime()

strtotime() Parses an English textual datetime into a Unix timestamp

time() Returns the current time as a Unix timestamp

timezone\_abbreviations\_list() Returns an associative array containing dst, offset, and the timezone name

timezone\_identifiers\_list() Returns an indexed array with all timezone identifiers

timezone\_location\_get() Returns location information for a specified timezone

timezone\_name\_from\_ abbr() Returns the timezone name from abbreviation

timezone\_name\_get() Returns the name of the timezone

timezone\_offset\_get() Returns the timezone offset from GMT

timezone\_open() Creates new DateTimeZone object

timezone\_transitions\_get() Returns all transitions for the timezone

timezone\_version\_get() Returns the version of the timezonedb

### PHP Predefined Date/Time Constants

Constant Description

DATE\_ATOM Atom (example: 2019-01-18T14:13:03+00:00)

DATE\_COOKIE HTTP Cookies (example: Fri, 18 Jan 2019 14:13:03 UTC)

DATE\_ISO8601 ISO-8601 (example: 2019-01-18T14:13:03+0000)

DATE\_RFC822 RFC 822 (example: Fri, 18 Jan 2019 14:13:03 +0000)

DATE\_RFC850 RFC 850 (example: Friday, 18-Jan-19 14:13:03 UTC)

DATE\_RFC1036 RFC 1036 (example: Friday, 18-Jan-19 14:13:03 +0000)

DATE\_RFC1123 RFC 1123 (example: Fri, 18 Jan 2019 14:13:03 +0000)

DATE\_RFC2822 RFC 2822 (example: Fri, 18 Jan 2019 14:13:03 +0000)

DATE\_RFC3339 Same as DATE\_ATOM (since PHP 5.1.3)

DATE\_RFC3339\_EXTENDED RFC3339 Extended format (since PHP 7.0.0) (example: 2019-01-18T16:34:01.000+00:00)

DATE\_RSS RSS (Fri, 18 Jan 2019 14:13:03 +0000)

DATE\_W3C World Wide Web Consortium (example: 2019-01-18T14:13:03+00:00)

SUNFUNCS\_RET\_TIMESTAMP Timestamp (since PHP 5.1.2)

SUNFUNCS\_RET\_STRING Hours:minutes (example: 09:41) (since PHP 5.1.2)

SUNFUNCS\_RET\_DOUBLE Hours as a floating point number (example: 9.75) (since PHP

5.1.2)

## PHP Script Introduction

### What is a PHP File?

PHP files can contain text, HTML, CSS, JavaScript, and PHP code

PHP code is executed on the server, and the result is returned to the browser as plain HTML

PHP files have extension ".php"

### What Can PHP Do?

PHP can generate dynamic page content

PHP can create, open, read, write, delete, and close files on the server

PHP can collect form data

PHP can send and receive cookies

PHP can add, delete, modify data in your database

PHP can be used to control user-access

PHP can encrypt data

## PHP Loops

Often when you write code, you want the same block of code to run over and over again a certain

number of times. So, instead of adding several almost equal code-lines in a script, we can use loops.

Loops are used to execute the same block of code again and again, as long as a certain condition is

true.

In PHP, we have the following loop types:

**while -** loops through a block of code as long as the specified condition is true

**do...while** - loops through a block of code once, and then repeats the loop as long as the specified

condition is true

**for** - loops through a block of code a specified number of times

**foreach** - loops through a block of code for each element in an array

## Working with pre-defined functions in PHP

PHP comes standard with many functions and constructs. i.e.

phpinfo(), print(), mysqli\_connect(), error\_reporting(), error\_log(), array(), copy(), unlink(),

date(),time(), strlen(), strlen()

For a complete reference and examples of the predefine functions, please visit

<http://php.net/manual/en/funcref.php>

Some usage of built-in functions are as mentioned below:

* Converting a string of letters to uppercase and lowercase
* Displaying and using the date and time
* Initializing and closing a database connection
* Declaring and using an array
* Handling files
* Accessing data in forms
* Filesystem Functions
* Function to open FTP connections
* Email related functions
* Mathematical Functions
* MySQL specific functions
* URL Functions
* Image functions

## Working with Different types of Mouse Events

Events that occur when the mouse interacts with the HTML document belongs to the MouseEvent Object.

The MouseEvent inherits all the properties and methods from:

* The UiEvent
* The Event Object

### Event Types

These event types belongs to the MouseEvent Object:

**Event Description**

onclick The event occurs when the user clicks on an element

oncontextmenu The event occurs when the user right-clicks on an element to open a context menu

ondblclick The event occurs when the user double-clicks on an element

onmousedown The event occurs when the user presses a mouse button over an element

onmouseenter The event occurs when the pointer is moved onto an element

onmouseleave The event occurs when the pointer is moved out of an element

onmousemove The event occurs when the pointer is moving while it is over an element

onmouseout The event occurs when a user moves the mouse pointer out of an element, or out of one of its children

onmouseover The event occurs when the pointer is moved onto an element, or onto one of its children

onmouseup The event occurs when a user releases a mouse button over an element

## Object Oriented Programming with PHP 5

From PHP5, you can also write PHP code in an object-oriented style.

Object-Oriented programming is faster and easier to execute.

### What is OOP?

OOP stands for Object-Oriented Programming.

Procedural programming is about writing procedures or functions that perform operations on the data, while object-oriented programming is about creating objects that contain both data and functions.

Object-oriented programming has several advantages over procedural programming:

OOP is faster and easier to execute

OOP provides a clear structure for the programs

OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug

OOP makes it possible to create full reusable applications with less code and shorter development time.

### What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:

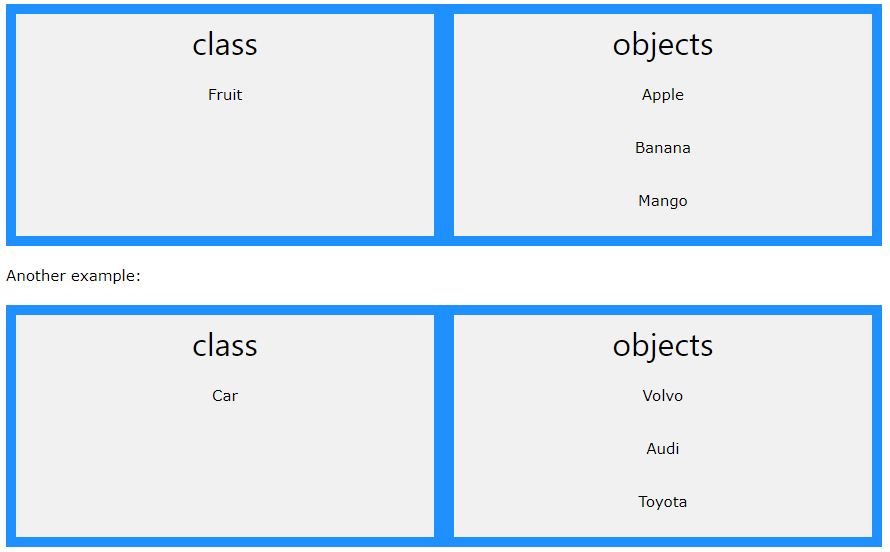


Image – Classes and Objects

Reference - <https://www.w3schools.com/php/php_oop_what_is.asp>

So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

### PHP OOP - Classes and Objects

A class is a template for objects, and an object is an instance of class.

Let's assume we have a class named Fruit. A Fruit can have properties like name, color, weight, etc. We can define variables like $name, $color, and $weight to hold the values of these properties.

When the individual objects (apple, banana, etc.) are created, they inherit all the properties and behaviors from the class, but each object will have different values for the properties.

### Define a Class

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods go inside the braces:

Syntax

*<?php*

*class Fruit {*

*// code goes here...*

*}*

*?>*

Below we declare a class named Fruit consisting of two properties ($name and $color) and two methods set\_name() and get\_name() for setting and getting the $name property:

*<?php*

*class Fruit {*

*// Properties*

*public $name;*

*public $color;*

*// Methods*

*function set\_name($name) {*

*$this->name = $name;*

*}*

*function get\_name() {*

*return $this->name;*

*}*

*}*

*?>*

Note: In a class, variables are called properties and functions are called methods!

### Define Objects

Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.

Objects of a class is created using the new keyword.

In the example below, $apple and $banana are instances of the class Fruit:

Example

*<?php*

*class Fruit {*

*// Properties*

*public $name;*

*public $color;*

*// Methods*

*function set\_name($name) {*

*$this->name = $name;*

*}*

*function get\_name() {*

*return $this->name;*

*}*

*}*

*$apple = new Fruit();*

*$banana = new Fruit();*

*$apple->set\_name('Apple');*

*$banana->set\_name('Banana');*

*echo $apple->get\_name();*

*echo "<br>";*

*echo $banana->get\_name();*

*?>*

In the example below, we add two more methods to class Fruit, for setting and getting the $color property:

Example

*<?php*

*class Fruit {*

*// Properties*

*public $name;*

*public $color;*

*// Methods*

*function set\_name($name) {*

*$this->name = $name;*

*}*

*function get\_name() {*

*return $this->name;*

*}*

*function set\_color($color) {*

*$this->color = $color;*

*}*

*function get\_color() {*

*return $this->color;*

*}*

*}*

*$apple = new Fruit();*

*$apple->set\_name('Apple');*

*$apple->set\_color('Red');*

*echo "Name: " . $apple->get\_name();*

*echo "<br>";*

*echo "Color: " . $apple->get\_color();*

*?>*

### PHP - The $this Keyword

The $this keyword refers to the current object, and is only available inside methods.

Look at the following example:

Example

*<?php*

*class Fruit {*

*public $name;*

*}*

*$apple = new Fruit();*

*?>*

So, where can we change the value of the $name property? There are two ways:

1. Inside the class (by adding a set\_name() method and use $this):

Example

*<?php*

*class Fruit {*

*public $name;*

*function set\_name($name) {*

*$this->name = $name;*

*}*

*}*

*$apple = new Fruit();*

*$apple->set\_name("Apple");*

*echo $apple->name;*

*?>*

2. Outside the class (by directly changing the property value):

Example

*<?php*

*class Fruit {*

*public $name;*

*}*

*$apple = new Fruit();*

*$apple->name = "Apple";*

*echo $apple->name;*

*?>*

### PHP – instanceof

You can use the instanceof keyword to check if an object belongs to a specific class:

Example

<?php

$apple = new Fruit();

var\_dump($apple instanceof Fruit);

?>

### PHP OOP – Constructor

### PHP - The \_\_construct Function

A constructor allows you to initialize an object's properties upon creation of the object.

If you create a \_\_construct() function, PHP will automatically call this function when you create an object from a class.

Notice that the construct function starts with two underscores (\_\_)!

We see in the example below, that using a constructor saves us from calling the set\_name() method which reduces the amount of code:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*function \_\_construct($name) {*

*$this->name = $name;*

*}*

*function get\_name() {*

*return $this->name;*

*}*

*}*

*$apple = new Fruit("Apple");*

*echo $apple->get\_name();*

*?>*

Another example:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*function \_\_construct($name, $color) {*

*$this->name = $name;*

*$this->color = $color;*

*}*

*function get\_name() {*

*return $this->name;*

*}*

*function get\_color() {*

*return $this->color;*

*}*

*}*

*$apple = new Fruit("Apple", "red");*

*echo $apple->get\_name();*

*echo "<br>";*

*echo $apple->get\_color();*

*?>*

### PHP OOP – Destructor

### PHP - The \_\_destruct Function

A destructor is called when the object is destructed or the script is stopped or exited.

If you create a \_\_destruct() function, PHP will automatically call this function at the end of the script.

Notice that the destruct function starts with two underscores (\_\_)!

The example below has a \_\_construct() function that is automatically called when you create an object from a class, and a \_\_destruct() function that is automatically called at the end of the script:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*function \_\_construct($name) {*

*$this->name = $name;*

*}*

*function \_\_destruct() {*

*echo "The fruit is {$this->name}.";*

*}*

*}*

*$apple = new Fruit("Apple");*

*?>*

Another example:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*function \_\_construct($name, $color) {*

*$this->name = $name;*

*$this->color = $color;*

*}*

*function \_\_destruct() {*

*echo "The fruit is {$this->name} and the color is {$this->color}.";*

*}*

*}*

*$apple = new Fruit("Apple", "red");*

*?>*

Tip: As constructors and destructors helps reducing the amount of code, they are very useful!

### PHP OOP - Access Modifiers

### PHP - Access Modifiers

Properties and methods can have access modifiers which control where they can be accessed.

There are three access modifiers:

**public** - the property or method can be accessed from everywhere. This is default

**protected** - the property or method can be accessed within the class and by classes derived from that class

**private** - the property or method can ONLY be accessed within the class

In the following example we have added three different access modifiers to three properties (name, color, and weight). Here, if you try to set the name property it will work fine (because the name property is public, and can be accessed from everywhere). However, if you try to set the color or weight property it will result in a fatal error (because the color and weight property are protected and private):

Example

*<?php*

*class Fruit {*

*public $name;*

*protected $color;*

*private $weight;*

*}*

*$mango = new Fruit();*

*$mango->name = 'Mango'; // OK*

*$mango->color = 'Yellow'; // ERROR*

*$mango->weight = '300'; // ERROR*

*?>*

In the next example we have added access modifiers to two functions. Here, if you try to call the set\_color() or the set\_weight() function it will result in a fatal error (because the two functions are considered protected and private), even if all the properties are public:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*public $weight;*

*function set\_name($n) { // a public function (default)*

*$this->name = $n;*

*}*

*protected function set\_color($n) { // a protected function*

*$this->color = $n;*

*}*

*private function set\_weight($n) { // a private function*

*$this->weight = $n;*

*}*

*}*

*$mango = new Fruit();*

*$mango->set\_name('Mango'); // OK*

*$mango->set\_color('Yellow'); // ERROR*

*$mango->set\_weight('300'); // ERROR*

*?>*

### PHP OOP – Inheritance

### PHP - What is Inheritance?

Inheritance in OOP = When a class derives from another class.

The child class will inherit all the public and protected properties and methods from the parent class. In addition, it can have its own properties and methods.

An inherited class is defined by using the extends keyword.

Let's look at an example:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*public function \_\_construct($name, $color) {*

*$this->name = $name;*

*$this->color = $color;*

*}*

*public function intro() {*

*echo "The fruit is {$this->name} and the color is {$this->color}.";*

*}*

*}*

*// Strawberry is inherited from Fruit*

*class Strawberry extends Fruit {*

*public function message() {*

*echo "Am I a fruit or a berry? ";*

*}*

*}*

*$strawberry = new Strawberry("Strawberry", "red");*

*$strawberry->message();*

*$strawberry->intro();*

*?>*

Example Explained

The Strawberry class is inherited from the Fruit class.

This means that the Strawberry class can use the public $name and $color properties as well as the public \_\_construct() and intro() methods from the Fruit class because of inheritance.

The Strawberry class also has its own method: message().

### PHP - Inheritance and the Protected Access Modifier

We learned that protected properties or methods can be accessed within the class and by classes derived from that class. What does that mean?

Let's look at an example:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*public function \_\_construct($name, $color) {*

*$this->name = $name;*

*$this->color = $color;*

*}*

*protected function intro() {*

*echo "The fruit is {$this->name} and the color is {$this->color}.";*

*}*

*}*

*class Strawberry extends Fruit {*

*public function message() {*

*echo "Am I a fruit or a berry? ";*

*}*

*}*

*// Try to call all three methods from outside class*

*$strawberry = new Strawberry("Strawberry", "red"); // OK. \_\_construct() is public*

*$strawberry->message(); // OK. message() is public*

*$strawberry->intro(); // ERROR. intro() is protected*

*?>*

In the example above we see that if we try to call a protected method (intro()) from outside the class, we will receive an error. public methods will work fine!

Let's look at another example:

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*public function \_\_construct($name, $color) {*

*$this->name = $name;*

*$this->color = $color;*

*}*

*protected function intro() {*

*echo "The fruit is {$this->name} and the color is {$this->color}.";*

*}*

*}*

*class Strawberry extends Fruit {*

*public function message() {*

*echo "Am I a fruit or a berry? ";*

*// Call protected method from within derived class - OK*

*$this -> intro();*

*}*

*}*

*$strawberry = new Strawberry("Strawberry", "red"); // OK. \_\_construct() is public*

*$strawberry->message(); // OK. message() is public and it calls intro() (which is protected) from within the derived class*

*?>*

In the example above we see that all works fine! It is because we call the protected method (intro()) from inside the derived class.

### PHP - Overriding Inherited Methods

Inherited methods can be overridden by redefining the methods (use the same name) in the child class.

Look at the example below. The \_\_construct() and intro() methods in the child class (Strawberry) will override the \_\_construct() and intro() methods in the parent class (Fruit):

Example

*<?php*

*class Fruit {*

*public $name;*

*public $color;*

*public function \_\_construct($name, $color) {*

*$this->name = $name;*

*$this->color = $color;*

*}*

*public function intro() {*

*echo "The fruit is {$this->name} and the color is {$this->color}.";*

*}*

*}*

*class Strawberry extends Fruit {*

*public $weight;*

*public function \_\_construct($name, $color, $weight) {*

*$this->name = $name;*

*$this->color = $color;*

*$this->weight = $weight;*

*}*

*public function intro() {*

*echo "The fruit is {$this->name}, the color is {$this->color}, and the weight is {$this->weight} gram.";*

*}*

*}*

*$strawberry = new Strawberry("Strawberry", "red", 50);*

*$strawberry->intro();*

*?>*

### PHP - The final Keyword

The final keyword can be used to prevent class inheritance or to prevent method overriding.

The following example shows how to prevent class inheritance:

Example

*<?php*

*final class Fruit {*

*// some code*

*}*

*// will result in error*

*class Strawberry extends Fruit {*

*// some code*

*}*

*?>*

The following example shows how to prevent method overriding:

Example

*<?php*

*class Fruit {*

*final public function intro() {*

*// some code*

*}*

*}*

*class Strawberry extends Fruit {*

*// will result in error*

*public function intro() {*

*// some code*

*}*

*}*

*?>*

### PHP OOP - Class Constants

### PHP - Class Constants

Constants cannot be changed once it is declared.

Class constants can be useful if you need to define some constant data within a class.

A class constant is declared inside a class with the const keyword.

Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters.

We can access a constant from outside the class by using the class name followed by the scope resolution operator (::) followed by the constant name, like here:

Example

*<?php*

*class Goodbye {*

*const LEAVING\_MESSAGE = "Thank you for visiting W3Schools.com!";*

*}*

*echo Goodbye::LEAVING\_MESSAGE;*

*?>*

Or, we can access a constant from inside the class by using the self keyword followed by the scope resolution operator (::) followed by the constant name, like here:

Example

*<?php*

*class Goodbye {*

*const LEAVING\_MESSAGE = "Thank you for visiting W3Schools.com!";*

*public function byebye() {*

*echo self::LEAVING\_MESSAGE;*

*}*

*}*

*$goodbye = new Goodbye();*

*$goodbye->byebye();*

*?>*

### PHP OOP - Abstract Classes

### PHP - What are Abstract Classes and Methods?

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks.

An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code.

An abstract class or method is defined with the abstract keyword:

Syntax

*<?php*

*abstract class ParentClass {*

*abstract public function someMethod1();*

*abstract public function someMethod2($name, $color);*

*abstract public function someMethod3() : string;*

*}*

*?>*

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

The child class method must be defined with the same name and it redeclares the parent abstract method

The child class method must be defined with the same or a less restricted access modifier

The number of required arguments must be the same. However, the child class may have optional arguments in addition

Let's look at an example:

Example

*<?php*

*// Parent class*

*abstract class Car {*

*public $name;*

*public function \_\_construct($name) {*

*$this->name = $name;*

*}*

*abstract public function intro() : string;*

*}*

*// Child classes*

*class Audi extends Car {*

*public function intro() : string {*

*return "Choose German quality! I'm an $this->name!";*

*}*

*}*

*class Volvo extends Car {*

*public function intro() : string {*

*return "Proud to be Swedish! I'm a $this->name!";*

*}*

*}*

*class Citroen extends Car {*

*public function intro() : string {*

*return "French extravagance! I'm a $this->name!";*

*}*

*}*

*// Create objects from the child classes*

*$audi = new audi("Audi");*

*echo $audi->intro();*

*echo "<br>";*

*$volvo = new volvo("Volvo");*

*echo $volvo->intro();*

*echo "<br>";*

*$citroen = new citroen("Citroen");*

*echo $citroen->intro();*

*?>*

Example Explained

The Audi, Volvo, and Citroen classes are inherited from the Car class. This means that the Audi, Volvo, and Citroen classes can use the public $name property as well as the public \_\_construct() method from the Car class because of inheritance.

But, intro() is an abstract method that should be defined in all the child classes and they should return a string.

### PHP OOP – Interfaces

### PHP - What are Interfaces?

Interfaces allow you to specify what methods a class should implement.

Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism".

Interfaces are declared with the interface keyword:

Syntax

*<?php*

*interface InterfaceName {*

*public function someMethod1();*

*public function someMethod2($name, $color);*

*public function someMethod3() : string;*

*}*

*?>*

### PHP - Interfaces vs. Abstract Classes

Interface are similar to abstract classes. The difference between interfaces and abstract classes are:

* Interfaces cannot have properties, while abstract classes can
* All interface methods must be public, while abstract class methods is public or protected
* All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary
* Classes can implement an interface while inheriting from another class at the same time

### PHP - Using Interfaces

To implement an interface, a class must use the implements keyword.

A class that implements an interface must implement all of the interface's methods.

Example

*<?php*

*interface Animal {*

*public function makeSound();*

*}*

*class Cat implements Animal {*

*public function makeSound() {*

*echo "Meow";*

*}*

*}*

*$animal = new Cat();*

*$animal->makeSound();*

*?>*

From the example above, let's say that we would like to write software which manages a group of animals. There are actions that all of the animals can do, but each animal does it in its own way.

Using interfaces, we can write some code which can work for all of the animals even if each animal behaves differently:

Example

*<?php*

*// Interface definition*

*interface Animal {*

*public function makeSound();*

*}*

*// Class definitions*

*class Cat implements Animal {*

*public function makeSound() {*

*echo " Meow ";*

*}*

*}*

*class Dog implements Animal {*

*public function makeSound() {*

*echo " Bark ";*

*}*

*}*

*class Mouse implements Animal {*

*public function makeSound() {*

*echo " Squeak ";*

*}*

*}*

*// Create a list of animals*

*$cat = new Cat();*

*$dog = new Dog();*

*$mouse = new Mouse();*

*$animals = array($cat, $dog, $mouse);*

*// Tell the animals to make a sound*

*foreach($animals as $animal) {*

*$animal->makeSound();*

*}*

*?>*

Example Explained

Cat, Dog and Mouse are all classes that implement the Animal interface, which means that all of them are able to make a sound using the makeSound() method. Because of this, we can loop through all of the animals and tell them to make a sound even if we don't know what type of animal each one is.

Since the interface does not tell the classes how to implement the method, each animal can make a sound in its own way.

### PHP OOP - Static Methods

### PHP - Static Methods

Static methods can be called directly - without creating an instance of the class first.

Static methods are declared with the static keyword:

Syntax

*<?php*

*class ClassName {*

*public static function staticMethod() {*

*echo "Hello World!";*

*}*

*}*

*?>*

To access a static method use the class name, double colon (::), and the method name:

Syntax

*ClassName::staticMethod();*

Let's look at an example:

Example

*<?php*

*class greeting {*

*public static function welcome() {*

*echo "Hello World!";*

*}*

*}*

*// Call static method*

*greeting::welcome();*

*?>*

Example Explained

Here, we declare a static method: welcome(). Then, we call the static method by using the class name, double colon (::), and the method name (without creating an instance of the class first).

### PHP OOP - Static Properties

### PHP - Static Properties

Static properties can be called directly - without creating an instance of a class.

Static properties are declared with the static keyword:

Syntax

*<?php*

*class ClassName {*

*public static $staticProp = "W3Schools";*

*}*

*?>*

To access a static property use the class name, double colon (::), and the property name:

Syntax

*ClassName::$staticProp;*

Let's look at an example:

Example

*<?php*

*class pi {*

*public static $value = 3.14159;*

*}*

*// Get static property*

*echo pi::$value;*

*?>*

Example Explained

Here, we declare a static property: $value. Then, we echo the value of the static property by using the class name, double colon (::), and the property name (without creating a class first).

### PHP - More on Static Properties

A class can have both static and non-static properties. A static property can be accessed from a method in the same class using the self keyword and double colon (::):

Example

*<?php*

*class pi {*

*public static $value=3.14159;*

*public function staticValue() {*

*return self::$value;*

*}*

*}*

*$pi = new pi();*

*echo $pi->staticValue();*

*?>*

To call a static property from a child class, use the parent keyword inside the child class:

Example

*<?php*

*class pi {*

*public static $value=3.14159;*

*}*

*class x extends pi {*

*public function xStatic() {*

*return parent::$value;*

*}*

*}*

*// Get value of static property directly via child class*

*echo x::$value;*

*// or get value of static property via xStatic() method*

*$x = new x();*

*echo $x->xStatic();*

*?>*

### PHP Iterables

### PHP - What is an Iterable?

An iterable is any value which can be looped through with a foreach() loop.

The iterable pseudo-type was introduced in PHP 7.1, and it can be used as a data type for function arguments and function return values.

### PHP - Using Iterables

The iterable keyword can be used as a data type of a function argument or as the return type of a function:

Example

Use an iterable function argument:

*<?php*

*function printIterable(iterable $myIterable) {*

*foreach($myIterable as $item) {*

*echo $item;*

*}*

*}*

*$arr = ["a", "b", "c"];*

*printIterable($arr);*

*?>*

Example

Return an iterable:

*<?php*

*function getIterable():iterable {*

*return ["a", "b", "c"];*

*}*

*$myIterable = getIterable();*

*foreach($myIterable as $item) {*

*echo $item;*

*}*

*?>*

### PHP - Creating Iterables

### Arrays

All arrays are iterables, so any array can be used as an argument of a function that requires an iterable.

### Iterators

Any object that implements the Iterator interface can be used as an argument of a function that requires an iterable.

An iterator contains a list of items and provides methods to loop through them. It keeps a pointer to one of the elements in the list. Each item in the list should have a key which can be used to find the item.

An iterator must have these methods:

current() - Returns the element that the pointer is currently pointing to. It can be any data type

key() - Returns the key associated with the current element in the list. It can only be an integer,

float, boolean or string

next() - Moves the pointer to the next element in the list

rewind() - Moves the pointer to the first element in the list

valid() - If the internal pointer is not pointing to any element (for example, if next() was called at the end of the list), this should return false. It returns true in any other case

Example

Implement the Iterator interface and use it as an iterable:

*<?php*

*// Create an Iterator*

*class MyIterator implements Iterator {*

*private $items = [];*

*private $pointer = 0;*

*public function \_\_construct($items) {*

*// array\_values() makes sure that the keys are numbers*

*$this->items = array\_values($items);*

*}*

*public function current() {*

*return $this->items[$this->pointer];*

*}*

*public function key() {*

*return $this->pointer;*

*}*

*public function next() {*

*$this->pointer++;*

*}*

*public function rewind() {*

*$this->pointer = 0;*

*}*

*public function valid() {*

*// count() indicates how many items are in the list*

*return $this->pointer < count($this->items);*

*}*

*}*

*// A function that uses iterables*

*function printIterable(iterable $myIterable) {*

*foreach($myIterable as $item) {*

*echo $item;*

*}*

*}*

*// Use the iterator as an iterable*

*$iterator = new MyIterator(["a", "b", "c"]);*

*printIterable($iterator);*

*?>*

## Debugging PHP Code

When we write huge lines of code in PHP and then some error occurs then removing that error might be a heck of the task. Some basic errors that programmer do while programming in PHP which are:

Missing Semicolon “;” and closing brackets “}”.

To debug the above errors, using a good PHP ide will be very helpful as it will suggest the closing bracket “}” and end of statement by “;”.

Misspelling a variable name. Remember $var != $Var as we know, PHP is a case sensitive language.

Using “=” instead of “==” (Assignment operator and Equal operator)

Example:

*if($a = $b) {*

*// Statement*

*}*

This will always result in True as it is never an error to assign one variable to another.

Missing quotes in SQL queries like ‘ ‘ and ” “. This is a very common and frequent error that occurs while PHP programming. To debug this kind of error always use mysqli\_error($con) command with echo to see what error you are doing in SQL statements where $con is the connection variable that you are using.

Example:

*if (!mysqli\_query($conn, $sql)) {*

*echo "Error: " . $sql . "*

*" . mysqli\_error($con);*

*}*

If your PHP script produces no output while running then make sure that “display\_errors” is set to on in php.ini file.

“Parse Error” – This error occurs when your code is not understood by PHP. This error generally occurs with a syntax error.

“Misunderstanding isset() behavior” – Despite its name, isset() not only returns false if an item does not exist but also returns false for null values. This behavior is more problematic than it might appear at first and is a common source of problems.

Example:

*$data = fetchRecordFromStorage($storage, $identifier);*

*if (!isset($data['keyShouldBeSet']) {*

*// do something here if 'keyShouldBeSet' is not set*

*}*

The author of this code presumably wanted to check if keyShouldBeSet was set in $data. But, as discussed, isset($data[‘keyShouldBeSet’]) will also return false if $data[‘keyShouldBeSet’] was set, but was set to null. So the above logic is flawed.

One common error is missing PHP closing before using HTML commands. So always close PHP with “?>”and then write HTML code and after ending HTML code use “<?php” to again start php coding.

PHP debugging tools: PHP code can be debug using one of many debugging tools to attach a debugger client. PhpStorm works with debug utilities like Xdebug and ZendDebugger.

Being a polyglot (knowing or using several languages), we need an IDE that supports multiple languages. The Xdebug with Visual Studio is used in the past, so let’s see how to set it up with the VS Code.

The debug server setup is the same, but each client (IDE or CLI) will have a slightly different setup. See the debug server (a Zend extension) opens a port, and the client communicates with the server through that port. It is just a matter of configuration and installing the right components.

Here are the steps to doing PHP programming:

Check for PHP extensions in VS Code.

Install the PHP Debug extension.

Click “reload” to reload VS Code.

Install Xdebug. The PHP Debug extension for VS Code is only integration to Xdebug. If we install PHP 7.0 then it must get the right version of Xdebug from the download page.

Now when you have the right version, put it in the PHP/ext directory.

Next, you need to configure PHP to use the extension and allow remote debugging. Add the following configuration to the php.ini file that’s listed in PHP Info:

; set the extension path

zend\_extension="C:/Program Files

(x86)/PHP/v7.0/ext/php\_xdebug-2.6.1-7.0-vc14-nts.dll"

; allow remote debugging

[XDebug]

xdebug.remote\_enable = 1

xdebug.remote\_autostart = 1

It will set up the PHP server to use XDebug. The steps here are the same no matter what IDE you use.

Xdebug opens an HTTP port so that your debugger can attach. The client still needs to be configured to attach and use the debugging protocol.

Finally, configure VS Code to connect to Xdebug. There are a few simple steps and then attaching is automatic.

**Configuring your IDE:** After installing Xdebug, you need to configure IDE to attach to the debugger. In VS Code, this means adding a debug configuration. Fortunately, it is automatic at this point. It’s just a few simple steps:

Switch to the debug view.

Click the gear to bring up the languages menu.

Select PHP. Visual Studio Code will generate the default configuration.

Reload the PHP server. We have to install another extension called “PHP Server” that makes this simple. Use the context menu (right-click) to control the PHP server.

It puts the IDE in a state which is ready to attach to Xdebug. Communications with the debugger happen through a TCP port on the debug server. Xdebug uses the DBGp protocol through port 9000 by default.

Attaching a debugger: The PHP Debug extension for VS Code generated a launch.json file. That file goes into a .vscode directory in the root of the project.

{

// Use IntelliSense to learn about possible attributes.

// Hover to view descriptions of existing attributes.

// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387

"version": "0.2.0",

"configurations": [

{

"name": "Listen for XDebug",

"type": "php",

"request": "launch",

"port": 9000

},

{

"name": "Launch currently open script",

"type": "php",

"request": "launch",

"program": "${file}",

"cwd": "${fileDirname}",

"port": 9000

}

]

}

It’s adding two launch configurations. Those are available in the debug view. We can either attach to a running server or launch a new one with the current script. Since I have phpinfo running already, I will start there by choosing Listen for XDebug to attach to that server. Once you are attached, you will see the debug toolbar. Most debuggers have a similar control mechanism which allows to start, stop, step, and restart debugger.

Switching error reporting level: PHP has a few ways to configure error reporting. You can use the php.ini file and you have to access it. Otherwise, you might use the htaccess configuration. If you can’t use configuration files, you have the option of changing the values via a script. A combination of settings will get the right levels of error logging. You want to consider the following settings:

error\_reporting sets the level of logging.

E\_NOTICE is useful during development since it will tell about defects such as unassigned variables.

display\_errors is used to display the error messages.

display\_startup\_errors should only be used when debugging.

log\_errors and error\_log work together to send errors to a log file. Do this in production rather than displaying them to end users.

The PHP manual spells out these settings in more detail and provides more information.