What is PHP?

PHP (recursive acronym for *PHP: Hypertext Preprocessor*) is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.

Nice, but what does that mean? An example:

Example #1 An introductory example

```
<!DOCTYPE html>
<html>
<head>
<title>Example</title>
</head>
<body>

<?php
echo "Hi, I'm a PHP script!";
?>

</body>
</html>
```

Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does *something* (in this case, output `Hi, I'm a PHP script!`). The PHP code is enclosed in special start and end processing instructions <?php and ?> that allow jumping in and out of "PHP mode".

What distinguishes PHP from something like client-side JavaScript is that the code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was. A web server can even be configured to process all HTML files with PHP, and then there's no way that users can tell that PHP is being used.

The best part about using PHP is that it is extremely simple for a newcomer, but offers many advanced features for a professional programmer. Don't be afraid to read the long list of PHP's features. With PHP, almost anyone can get up and running and be writing simple scripts in no time at all.

What can PHP do?

Anything. PHP is mainly focused on server-side scripting, so it can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies. But PHP can do much more.

There are two main areas where PHP scripts are used.

- Server-side scripting. This is the most widely used and main target field for PHP. Three things are needed to make this work: the PHP parser (CGI or server module), a web server, and a web browser. All these can run on a local machine in order to just experiment with PHP programming.

- Command line scripting. A PHP script can be run without any server or browser, only the PHP parser is needed to use it this way. This type of usage is ideal for scripts regularly executed using **cron** (on Unix or macOS) or Task Scheduler (on Windows). These scripts can also be used for simple text processing tasks.

PHP can be used on all major operating systems, including Linux, many Unix variants (including HP-UX, Solaris and OpenBSD), Microsoft Windows, macOS, RISC OS, and probably others. PHP also has support for most of the web servers today. This includes Apache, IIS, and many others. And this includes any web server that can utilize the FastCGI PHP binary, like lighttpd and nginx. PHP works as either a module, or as a CGI processor.

So with PHP, developers have the freedom of choosing an operating system and a web server. Furthermore, they also have the choice of using procedural programming or object-oriented programming (OOP), or a mixture of them both.

PHP is not limited to outputting HTML. PHP's abilities include outputting rich file types, such as images or PDF files, encrypting data, and sending emails. It can also output easily any text, such as JSON or XML. PHP can autogenerate these files, and save them in the file system, instead of printing it out, forming a server-side cache for dynamic content.

PHP tags

When PHP processes a file, it recognizes the opening and closing tags, <?php and ?>, to define the boundaries of PHP code execution. Content outside these tags is ignored by the PHP parser, allowing PHP to seamlessly embed in various document types.
A whitespace character (space, tab, or newline) must follow <?php to ensure proper token separation. Omitting this whitespace will result in a syntax error.
PHP also includes the short echo tag <?=, which is shorthand for <?php echo.

Example #1 PHP Opening and Closing Tags

```
1. <?php echo 'if you want to serve PHP code in XHTML or XML documents,
use these tags'; ?>

2. You can use the short echo tag to <?= 'print this string' ?>.
It's equivalent to <?php echo 'print this string' ?>.
```

3. <? echo 'this code is within short tags, but will only work '.
'if short_open_tag is enabled'; ?>

Comments

PHP supports 'C', 'C++' and Unix shell-style (Perl style) comments. For example:

```
<?php
echo "This is a test\n"; // This is a one-line c++ style comment
/* This is a multi line comment
yet another line of comment */
echo "This is yet another test\n";
echo "One Final Test\n"; # This is a one-line shell-style comment
?>
```

The "one-line" comment styles only comment to the end of the line or the current block of PHP code, whichever comes first. This means that HTML code after // ... ?> or # ... ?> WILL be printed: ?> breaks out of PHP mode and returns to HTML mode, and // or # cannot influence that.

PHP Variables

Variables are "containers" for storing information.In PHP, a variable starts with the $ sign, followed by the name of the variable:

```
$x = 5;
$y = "John";
```

In the example above, the variable $x will hold the value 5, and the variable $y will hold the value "John".
Note: When you assign a text value to a variable, put quotes around the value.

Note: Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5;
$y = "John";

echo $x;
echo "<br>";
echo $y;
?>

</body>
</html>
```

A variable can have a short name (like $x and $y) or a more descriptive name ($age, $carname, $total_volume).
Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)

PHP Variables Scope

In PHP, variables can be declared anywhere in the script.

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has three different variable scopes:

- local
- global
- static

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
$x = 5; // global scope

function myTest() {
  // using x inside this function will generate an error
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();

echo "<p>Variable x outside function is: $x</p>";
?>

</body>
</html>
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

```
<?php
function myTest() {
$x = 5; // local scope
```

```
  echo "<p>Variable x inside function is: $x</p>";
}
myTest();

// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

PHP The global Keyword

The global keyword is used to access a global variable from within a function.

To do this, use the global keyword before the variables (inside the function):

```
<?php
$x = 5;
$y = 10;

function myTest() {
  global $x, $y;
  $y = $x + $y;
}

myTest();  // run function
echo $y; // output the new value for variable $y
?>
```

PHP also stores all global variables in an array called $GLOBALS[*index*]. The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

```
<?php
$x = 5;
$y = 10;

function myTest() {
  $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}

myTest();
echo $y;
?>
```

PHP echo and print Statements

Echo and print are more or less the same. They are both used to output data to the screen.

The differences are small: echo has no return value while print has a return value of 1 so it can be used in expressions. Echo can take multiple parameters (although such usage is rare) while print can take one argument. Echo is marginally faster than print.

The echo statement can be used with or without parentheses: echo or echo().

```
<?php
echo "Hello";
//same as:
echo("Hello");
?>
```

Display Variables

```
<?php
$txt1 = "Learn PHP";
$txt2 = "demo.com";

echo "<h2>$txt1</h2>";
echo "<p>Study PHP at $txt2</p>";
?>
```

Using Single Quotes

Strings are surrounded by quotes, but there is a difference between single and double quotes in PHP.

When using double quotes, variables can be inserted to the string as in the example above.

When using single quotes, variables have to be inserted using the . operator, like this:

```
<?php
$txt1 = "Learn PHP";
$txt2 = "demo.com";

echo '<h2>' . $txt1 . '</h2>';
echo '<p>Study PHP at ' . $txt2 . '</p>';
?>
```

The print statement can be used with or without parentheses: print or print().

```
<?php
print "Hello";
//same as:
print("Hello");
?>
```

PHP Data Types

Variables can store data of different types, and different data types can do different things.

PHP supports the following data types:

- String
- Integer
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object

Getting the Data Type

You can get the data type of any object by using the var_dump() function.

```
$x = 5;
var_dump($x);
```

PHP String

A string is a sequence of characters, like "Hello world!".

A string can be any text inside quotes. You can use single or double quotes:

```
<?php
$x = "Hello world!";
$y = 'Hello world!';
```

```
var_dump($x);
echo "<br>";
var_dump($y);
?>
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.

Rules for integers:

- An integer must have at least one digit
- An integer must not have a decimal point
- An integer can be either positive or negative
- Integers can be specified in: decimal (base 10), hexadecimal (base 16), octal (base 8), or binary (base 2) notation

In the following example $x is an integer. The PHP var_dump() function returns the data type and value:

```
$x = 5985;
var_dump($x);
```

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.

In the following example $x is a float. The PHP var_dump() function returns the data type and value:

```
<?php
$x = 10.365;
var_dump($x);
?>
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
var_dump($x);
```

PHP Array

An array stores multiple values in one single variable.

In the following example $cars is an array. The PHP var_dump() function returns the data type and value:

```
<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>
```

PHP Object

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

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.

Let's assume we have a class named Car that can have properties like model, color, etc. We can define variables like $model, $color, and so on, to hold the values of these properties.

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

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

```
<?php
class Car {
  public $color;
  public $model;
  public function __construct($color, $model) {
    $this->color = $color;
    $this->model = $model;
  }
  public function message() {
    return "My car is a " . $this->color . " " . $this->model . "!";
  }
}
$myCar = new Car("red", "Volvo");
var_dump($myCar);
?>
```

PHP -Modify Strings

PHP has a set of built-in functions that you can use to modify strings.
The strtoupper() function returns the string in upper case:

```
<?php
$x = "Hello World!";
echo strtoupper($x);
?>
```

The strtolower() function returns the string in lower case:

```
<?php
$x = "Hello World!";
echo strtolower($x);
?>
```

The PHP str_replace() function replaces some characters with some other characters in a string.Replace the text "World" with "Dolly":

```
<?php
$x = "Hello World!";
```

```
echo str_replace("World", "Dolly", $x);
?>
```

String Concatenation

```
<?php
$x = "Hello";
$y = "World";
$z = $x . $y;
echo $z;
?>
```

PHP Constants

A constant is an identifier (name) for a simple value. The value cannot be changed during the script.

A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script.

To create a constant, use the define() function.

define(*name*, *value*);

Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant

```
<?php
// case-sensitive constant name
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

PHP Operators

Operators are used to perform operations on variables and values.

PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional assignment operators

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y
<=>	Spaceship	$x <=> $y	Returns an integer less than, equal to, or greater than zero, depending on if $x is less than, equal to, or greater than $y. Introduced in PHP 7.

PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value.

The PHP decrement operators are used to decrement a variable's value.

Operator	Same as...	Description

++$x	Pre-increment	Increments $x by one, then returns $x
$x++	Post-increment	Returns $x, then increments $x by one
--$x	Pre-decrement	Decrements $x by one, then returns $x
$x--	Post-decrement	Returns $x, then decrements $x by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	Name	Example	Result
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2
.=	Concatenation assignment	$txt1 .= $txt2	Appends $txt2 to $txt1

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x === $y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
<>	Inequality	$x <> $y	Returns true if $x is not equal to $y
!==	Non-identity	$x !== $y	Returns true if $x is not identical to $y

PHP Conditional Assignment Operators

The PHP conditional assignment operators are used to set a value depending on conditions:

Operator	Name	Example	Result
?:	Ternary	$x = *expr1* ? *expr2* : *expr3*	Returns the value of $x. The value of $x is *expr2* if *expr1* = TRUE. The value of $x is *expr3* if *expr1* = FALSE
??	Null coalescing	$x = *expr1* ?? *expr2*	Returns the value of $x. The value of $x is *expr1* if *expr1* exists, and is not NULL. If *expr1* does not exist, or is NULL, the value of $x is *expr2*. Introduced in PHP 7

PHP Conditional Statements

Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.

In PHP we have the following conditional statements:

- ➓ if statement - executes some code if one condition is true
- ➓ if...else statement - executes some code if a condition is true and another code if that condition is false
- ➓ if...elseif...else statement - executes different codes for more than two conditions
- ➓ switch statement - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement executes some code if one condition is true.

```
if (condition) {
  // code to be executed if condition is true;
}
```

Example

Output "Have a good day!" if 5 is larger than 3:

```
<?php
if (5 > 3) {
  echo "Have a good day!";
}
?>
```

If statements usually contain conditions that compare two values.

```
<?php
$t = 14;
if ($t == 14) {
  echo "Have a good day!";
}
?>
```

To compare two values, we need to use a comparison operator.

Here are the PHP comparison operators to use in if statements:

Operator	Name	Result
==	Equal	Returns true if the values are equal
===	Identical	Returns true if the values and data types are identical
!=	Not equal	Returns true if the values are not equal
<>	Not equal	Returns true if the values are not equal
!==	Not identical	Returns true if the values or data types are not identical
>	Greater than	Returns true if the first value is greater than the second value
<	Less than	Returns true if the first value is less than the second value
>=	Greater than or equal to	Returns true if the first value is greater than, or equal to, the second value

<=	Less than or equal to	Returns true if the first value is less than, or equal to, the second value

PHP - The if...else Statement

The if...else statement executes some code if a condition is true and another code if that condition is false.

```
if (condition) {
  // code to be executed if condition is true;
} else {
  // code to be executed if condition is false;
}
```

Example

Output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

```
<?php
$t = date("H");

if ($t < "20") {
  echo "Have a good day!";
} else {
  echo "Have a good night!";
}
?>
```

PHP - The if...elseif...else Statement

The if...elseif...else statement executes different codes for more than two conditions.

```
if (condition) {
  code to be executed if this condition is true;
} elseif (condition) {
  // code to be executed if first condition is false and this condition is true;
} else {
  // code to be executed if all conditions are false;
}
```

Output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

```
<?php
$t = date("H");
echo "<p>The hour (of the server) is " . $t;
echo ", and will give the following message:</p>";

if ($t < "10") {
  echo "Have a good morning!";
} elseif ($t < "20") {
  echo "Have a good day!";
```

```
} else {
  echo "Have a good night!";
}
?>
```

The PHP switch Statement

Use the switch statement to **select one of many blocks of code to be executed**.

```
switch (expression) {
   case label1:
      //code block
      break;
   case label2:
      //code block;
      break;
   case label3:
      //code block
      break;
   default:
      //code block
}
```

This is how it works:

- The expression is evaluated once
- The value of the expression is compared with the values of each case
- If there is a match, the associated block of code is executed
- The break keyword breaks out of the switch block
- The default code block is executed if there is no match

```
<?php
$favcolor = "red";

switch ($favcolor) {
  case "red":
    echo "Your favorite color is red!";
    break;
  case "blue":
    echo "Your favorite color is blue!";
    break;
  case "green":
    echo "Your favorite color is green!";
    break;
  default:
    echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

The break Keyword

When PHP reaches a break keyword, it breaks out of the switch block.

This will stop the execution of more code, and no more cases are tested.

The last block does not need a break, the block breaks (ends) there anyway.

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Example

Print $i as long as $i is less than 6:

```
<?php
$i = 1;

while ($i < 6) {
  echo $i;
  $i++;
}
?>
```

The while loop does not run a specific number of times, but checks after each iteration if the condition is still true.
The condition does not have to be a counter, it could be the status of an operation or any condition that evaluates to either true or false.

The PHP do...while Loop

The do...while loop will always execute the block of code at least once, it will then check the condition, and repeat the loop while the specified condition is true.

Print $i as long as $i is less than 6:

```
<?php
$i = 1;
do {
  echo $i;
  $i++;
} while ($i < 6);
?>
```

PHP for Loop

The for loop - Loops through a block of code a specified number of times.The for loop is used when you know how many times the script should run.

```
for (expression1, expression2, expression3) {
   // code block
}
```

This is how it works:

- expression1 is evaluated once
- expression2 is evaluated before each iteration
- expression3 is evaluated after each iteration

```
<?php
for ($x = 0; $x <= 10; $x++) {
  echo "The number is: $x <br>";
}
?>
```

Example Explained

1. The first expression, $x = 0;, is evaluated once and sets a counter to 0.
2. The second expression, $x <= 10;, is evaluated before each iteration, and the code block is only executed if this expression evaluates to true. In this example the expression is true as long as $x is less than, or equal to, 10.
3. The third expression, $x++;, is evaluated after each iteration, and in this example, the expression increases the value of $x by one at each iteration.

The foreach Loop on Arrays

The most common use of the foreach loop, is to loop through the items of an array.

Loop through the items of an indexed array:

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $x) {
  echo "$x <br>";
}
?>
```

The array above is an indexed array, where the first item has the key 0, the second has the key 1, and so on.

Associative arrays are different, associative arrays use named keys that you assign to them, and when looping through associative arrays, you might want to keep the key as well as the value.

PHP Functions

The real power of PHP comes from its functions.

PHP has more than 1000 built-in functions, and in addition you can create your own custom functions.

PHP User Defined Functions

Besides the built-in PHP functions, it is possible to create your own functions.

- A function is a block of statements that can be used repeatedly in a program.
- A function will not execute automatically when a page loads.
- A function will be executed by a call to the function.

Create a Function

A user-defined function declaration starts with the keyword function, followed by the name of the function:

```
function myMessage() {
echo "Hello world!";
}
```

Call a Function

To call the function, just write its name followed by parentheses ():

```
<?php
function myMessage() {
  echo "Hello world!";
}
myMessage();
?>
```

In our example, we create a function named myMessage().

The opening curly brace { indicates the beginning of the function code, and the closing curly brace } indicates the end of the function.

The function outputs "Hello world!".

PHP Function Arguments

Information can be passed to functions through arguments. An argument is just like a variable.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name, e.g. ("Jani"), and the name is used inside the function, which outputs several different first names, but an equal last name:

```
<?php
function familyName($fname) {
  echo "$fname Refsnes.<br>";
}


familyName("Jani");
familyName("Hege");
familyName("Stale");
familyName("Kai Jim");
familyName("Borge");
```

?>

The following example has a function with two arguments ($fname, $year):

```
<?php
function familyName($fname, $year) {
  echo "$fname Refsnes. Born in $year <br>";
}

familyName("Hege","1975");
familyName("Stale","1978");
familyName("Kai Jim","1983");
?>
```

PHP Array Types

An array is a special variable that can hold many values under a single name, and you can access the values by referring to an index number or name.In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

In indexed arrays each item has an index number.

By default, the first item has index 0, the second item has item 1, etc.

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
var_dump($cars);
?>
```

PHP Associative Arrays

Associative arrays are arrays that use named keys that you assign to them.

```
<?php
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);
var_dump($car);
?>
```

Access Associative Arrays

To access an array item you can refer to the key name.

```
<?php
```

```
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

echo $car["model"];

?>
```

Change Value

To change the value of an array item, use the key name:

```
<?php
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);
$car["year"] = 2024;
var_dump($car);
?>
```

Loop Through an Associative Array

To loop through and print all the values of an associative array, you could use a foreach loop, like this:

```
<?php
$car = array("brand"=>"Ford", "model"=>"Mustang", "year"=>1964);

foreach ($car as $x => $y) {
  echo "$x: $y <br>";
}
?>
```

PHP - Sort Functions For Arrays

In this chapter, we will go through the following PHP array sort functions:

- sort() - sort arrays in ascending order
- rsort() - sort arrays in descending order
- asort() - sort associative arrays in ascending order, according to the value
- ksort() - sort associative arrays in ascending order, according to the key
- arsort() - sort associative arrays in descending order, according to the value
- krsort() - sort associative arrays in descending order, according to the key

The following example sorts the elements of the $cars array in ascending alphabetical order:

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);

$clength = count($cars);
for($x = 0; $x < $clength; $x++) {
  echo $cars[$x];
  echo "<br>";
}
?>
```

PHP - Multidimensional Arrays

A multidimensional array is an array containing one or more arrays.

PHP supports multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.

PHP - Two-dimensional Arrays

A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).

First, take a look at the following table:

Name	Stock	Sold
Volvo	22	18
BMW	15	13
Saab	5	2
Land Rover	17	15

We can store the data from the table above in a two-dimensional array, like this:

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
```

Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.
To get access to the elements of the $cars array we must point to the two indices (row and column):

```
<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);

echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```

We can also put a for loop inside another for loop to get the elements of the $cars array (we still have to point to the two indices):

```
<?php
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
```

```
  array("Saab",5,2),
  array("Land Rover",17,15)
);

for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

PHP - A Simple HTML Form

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

```
<!DOCTYPE HTML>
<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.

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.

Use a Web Host With PHP Support

If your server has activated support for PHP you do not need to do anything.

Just create some .php files, place them in your web directory, and the server will automatically parse them for you.

You do not need to compile anything or install any extra tools.

Because PHP is free, most web hosts offer PHP support.

Installation and Configuration

1] Go to https://www.apachefriends.org/download.html

2] Download 8.0.30 / PHP 8.0.30

3] Follow the steps from https://www.apachefriends.org/faq_windows.html