PHP

PHP is a server scripting language, and a powerful tool for making dynamic and interactive Web pages. PHP is a widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

```
<!DOCTYPE html>
<html>
<body>
<?php
echo "My first PHP script!";
?>
</body>
</html>
```

What is PHP?

- PHP is an acronym for "PHP: Hypertext Preprocessor"
- PHP is a widely-used, open source scripting language
- PHP scripts are executed on the server
- PHP is free to download and use
- **PHP is an amazing and popular language!**
- It is powerful enough to be at the core of the biggest blogging system on the web (WordPress)!
  It is deep enough to run the largest social network (Facebook)!
  It is also easy enough to be a beginner's first server side language!

What is a PHP File?

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

With PHP you are not limited to output HTML. You can output images, PDF files, and even Flash movies. You can also output any text, such as XHTML and XML.

Why PHP?

- PHP runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)

- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP supports a wide range of databases
- PHP is free. Download it from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

PHP Environment Setup

In order to develop and run PHP Web pages three vital components need to be installed on your computer system.

- **Web Server -** PHP will work with virtually all Web Server software, including Microsoft's Internet Information Server (IIS) but then most often used is freely availble Apache Server.
- **Database -** PHP will work with virtually all database software, including Oracle and Sybase but most commonly used is freely available MySQL database.
- **PHP Parser -** In order to process PHP script instructions a parser must be installed to generate HTML output that can be sent to the Web Browser.

What Do I Need?

To start using PHP, you can:

- Find a web host with PHP and MySQL support
- Install a web server on your own PC, and then install PHP and MySQL

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.

Set Up PHP on Your Own PC

However, if your server does not support PHP, you must:

- install a web server
- install PHP
- install a database, such as MySQL

Canonical PHP tags:

The most universally effective PHP tag style is:

```
<?php...?>
```

If you use this style, you can be positive that your tags will always be correctly interpreted.

PHP Syntax

A PHP script is executed on the server, and the plain HTML result is sent back to the browser.

Basic PHP Syntax

A PHP script can be placed anywhere in the document.

A PHP script starts with **<?php** and ends with **?>**:

```
<?php
// PHP code goes here
?>
```

The default file extension for PHP files is ".php". A PHP file normally contains HTML tags, and some PHP scripting code. Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

Note: PHP statements end with a semicolon (;).

Comments in PHP

A comment in PHP code is a line that is not read/executed as part of the program. Its only purpose is to be read by someone who is looking at the code.

Comments can be used to:

- Let others understand what you are doing
- Remind yourself of what you did - Most programmers have experienced coming back to their own work a year or two later and having to re-figure out what they did. Comments can remind you of what you were thinking when you wrote the code PHP supports several ways of commenting:

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
// This is a single-line comment
```

```
# This is also a single-line comment
/*
This is a multiple-lines comment block
that spans over multiple
lines
*/
// You can also use comments to leave out parts of a code line
$x = 5 /* + 15 */ + 5;
echo $x;
?>
</body>
</html>
```

PHP Case Sensitivity

In PHP, all keywords (e.g. if, else, while, echo, etc.), classes, functions, and user-defined functions are NOT case-sensitive. In the example below, all three echo statements below are legal (and equal):

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
ECHO "Hello World!<br>";
echo "Hello World!<br>";
EcHo "Hello World!<br>";
?>
</body>
</html>
```

However; all variable names are case-sensitive.

In the example below, only the first statement will display the value of the $color variable (this is because $color, $COLOR, and $coLOR are treated as three different variables):

Example

```
<!DOCTYPE html>
<html>
<body>
<?php
$color = "red";
echo "My car is " . $color . "<br>";
echo "My house is " . $COLOR . "<br>";
echo "My boat is " . $coLOR . "<br>";
?>
```

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

PHP Variables

Variables are "containers" for storing information.

Creating (Declaring) PHP Variables

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

After the execution of the statements above, the variable **$txt** will hold the value **Hello world!**, the variable **$x** will hold the value **5**, and the variable **$y** will hold the value **10.5**.

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.

Think of variables as containers for storing data.

PHP Variables

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)

Remember that PHP variable names are case-sensitive!

Output Variables

The PHP echo statement is often used to output data to the screen. The following example will show how to output text and a variable:

Example

```
<?php
$txt = "W3Schools.com";
echo "I love $txt!";
?>
```

The following example will produce the same output as the example above:

Example

```
<?php
$txt = "W3Schools.com";
echo "I love " . $txt . "!";
?>
```

The following example will output the sum of two variables:

Example

```
<?php
$x = 5;
$y = 4;
echo $x + $y;
?>
```

Note: You will learn more about the echo statement and how to output data to the screen in the next chapter.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In other languages such as C, C++, and Java, the programmer must declare the name and type of the variable before using it.

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

```
<?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>";
?>
```

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

Example

```
<?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>";
?>
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

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):

Example

```
<?php
$x = 5;
$y = 10;
function myTest() {
   global $x, $y;
   $y = $x + $y;
}
myTest();
echo $y; // outputs 15
?>
```

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:

Example

```
<?php
$x = 5;
$y = 10;
function myTest() {
   $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the **static** keyword when you first declare the variable:

Example

```
<?php
function myTest() {
   static $x = 0;
   echo $x;
   $x++;
}

myTest();
myTest();
myTest();
?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

PHP echo and print Statements

In PHP there are two basic ways to get output: echo and print. In this tutorial we use echo (and print) in almost every example. So, this chapter contains a little more info about those two output statements.

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 PHP echo Statement

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

Display Text

The following example shows how to output text with the echo command (notice that the text can contain HTML markup):

Example

```
<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>
```

Display Variables

The following example shows how to output text and variables with the echo statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;

echo "<h2>$txt1</h2>";
echo "Study PHP at $txt2<br>";
echo $x + $y;
?>
```

The PHP print Statement

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

Display Text

The following example shows how to output text with the print command (notice that the text can contain HTML markup):

Example

```
<?php
print "<h2>PHP is Fun!</h2>";
print "Hello world!<br>";
print "I'm about to learn PHP!";
?>
```

Display Variables

The following example shows how to output text and variables with the print statement:

Example

```
<?php
$txt1 = "Learn PHP";
$txt2 = "W3Schools.com";
$x = 5;
$y = 4;
print "<h2>$txt1</h2>";
print "Study PHP at $txt2<br>";
print $x + $y;
?>
```

PHP Data Types

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
- NULL
- Resource

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:

Example

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

```
echo $x;
echo "<br>";
echo $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 three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)

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

Example

```
<?php
$x = 5985;
var_dump($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:

Example

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

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

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:

Example

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

PHP Object

An object is a data type which stores data and information on how to process that data. In PHP, an object must be explicitly declared. First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
   function Car() {
      $this->model = "VW";
   }
}

// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

PHP NULL Value

Null is a special data type which can have only one value: NULL. A variable of data type NULL is a variable that has no value assigned to it. **Tip:** If a variable is created without a value, it is automatically assigned a value of NULL. Variables can also be emptied by setting the value to NULL:

Example

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP. A common example of using the resource data type is a database call. We will not talk about the resource type here, since it is an advanced topic.

PHP 5 Strings

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

PHP String Functions

In this chapter we will look at some commonly used functions to manipulate strings.

Get The Length of a String

The PHP strlen() function returns the length of a string.The example below returns the length of the string "Hello world!":

Example

```
<?php
echo strlen("Hello world!"); // outputs 12
?>
```

The output of the code above will be: 12.

Count The Number of Words in a String

The PHP str_word_count() function counts the number of words in a string:

Example

```
<?php
echo str_word_count("Hello world!"); // outputs 2
?>
```

The output of the code above will be: 2.

Reverse a String

The PHP strrev() function reverses a string:

Example

```
<?php
echo strrev("Hello world
```

```
// outputs !dlrow olleH
?>
```

The output of the code above will be: !dlrow olleH.

Search For a Specific Text Within a String

The PHP strpos() function searches for a specific text within a string. If a match is found, the function returns the character position of the first match. If no match is found, it will return FALSE. The example below searches for the text "world" in the string "Hello world!":

Example

```
<?php
echo strpos("Hello world!", "world"); // outputs 6
?>
```

The output of the code above will be: 6.

Tip: The first character position in a string is 0 (not 1).

Replace Text Within a String

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

Example

```
<?php
echo str_replace("world", "Dolly", "Hello world!"); // outputs Hello Dolly!
?>
```

The output of the code above will be: Hello Dolly!

Complete PHP String Reference

For a complete reference of all string functions, go to our complete PHP String Reference. The PHP string reference contains description and example of use, for each function!

PHP Constants

Constants are like variables except that once they are defined they cannot be changed or undefined.

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.

Create a PHP Constant

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

Syntax

define(*name*, *value*, *case-insensitive*)

Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive. Default is false

The example below creates a constant with a **case-sensitive** name:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
echo GREETING;
?>
```

The example below creates a constant with a **case-insensitive** name:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!", true);
echo greeting;
?>
```

Constants are Global

Constants are automatically global and can be used across the entire script. The example below uses a constant inside a function, even if it is defined outside the function:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");

function myTest() {
    echo GREETING;
}

myTest();
?>
```

What is Operator? Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

- Arithmetic Operators
- Comparision Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmatic Operators:

There are following arithmatic operators supported by PHP language:

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by denominator	B / A will give 2
%	Modulus Operator and remainder of after an integer division	B % A will give 0
++	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Comparison Operators:

There are following comparison operators supported by PHP

language Assume variable A holds 10 and variable B holds 20

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators:

There are following logical operators supported by PHP language

Assume variable A holds 10 and variable B holds 20 then:

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(A and B) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A or B) is true.

&&	Called Logical AND operator. If both the operands are non zero then then condition becomes true.	(A && B) is true.
||	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is false.

Assignment Operators:

There are following assignment operators supported by PHP

language:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assigne value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A

%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax:

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Categories:

All the operators we have discussed above can be categorised into following categories:

- Unary prefix operators, which precede a single operand.
- Binary operators, which take two operands and perform a variety of arithmetic and logical operations.
- The conditional operator (a ternary operator), which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.
- Assignment operators, which assign a value to a variable.

Precedence of PHP Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedenace than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Unary	! ++ --	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /=	

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:

- **if statement** - executes some code if one condition is true
- **if...else statement** - executes some code if a condition is true and another code if that condition is false
- **if...elseif....else statement** - executes different codes for more than two conditions
- **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.

Syntax

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

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

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

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

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.

Syntax

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

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?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.

Syntax

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

The example below will 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!":

Example

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

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

PHP 5 switch Statement

The switch statement is used to perform different actions based on different conditions.

The PHP switch Statement

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

Syntax

```
switch (n) {
   case label1:
      code to be executed if n=label1;
      break;
   case label2:
      code to be executed if n=label2;
      break;
   case label3:
```

```
    code to be executed if n=label3;
    break;
  ...
  default:
    code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use **break** to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

Example

```
<?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!";
}
?>
```

PHP 5 while Loops

PHP while loops execute a block of code while the specified condition is true.

PHP Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **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

---

The PHP while Loop

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

Syntax

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

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

Example

```
<?php
$x = 1;

while($x <= 5) {
   echo "The number is: $x <br>";
   $x++;
}
?>
```

The PHP do...while Loop

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

Syntax

```
do {
   code to be executed;
} while (condition is true);
```

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

```
<?php
$x = 1;

do {
   echo "The number is: $x <br>";
   $x++;
} while ($x <= 5);
?>
```

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

```
<?php
$x = 6;

do {
   echo "The number is: $x <br>";
   $x++;
} while ($x <= 5);
?>
```

PHP 5 for Loops

PHP for loops execute a block of code a specified number of times.

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
   code to be executed;
}
```

Parameters:

- *init counter*: Initialize the loop counter value

- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value

The example below displays the numbers from 0 to 10:

Example

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

The PHP foreach Loop

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

```
foreach ($array as $value) {
   code to be executed;
}
```

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors):

Example

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

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

The real power of PHP comes from its functions; it has more than 1000 built-in functions.

PHP User Defined Functions

Besides the built-in PHP functions, we can create our own functions.

A function is a block of statements that can be used repeatedly in a program.

A function will not execute immediately when a page loads.

A function will be executed by a call to the function.

Create a User Defined Function in PHP

A user defined function declaration starts with the word "function":

Syntax

```
function functionName() {
   code to be executed;
}
```

Note: A function name can start with a letter or underscore (not a number).

Tip: Give the function a name that reflects what the function does!

Function names are NOT case-sensitive.

In the example below, we create a function named "writeMsg()". 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!". To call the function, just write its name:

Example

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

writeMsg(); // call the function
?>
```

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:

Example

```
<?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 and $year):

Example

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

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

PHP Default Argument Value

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?php
function setHeight($minheight = 50) {
```

```
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

PHP Functions - Returning values

To let a function return a value, use the return statement:

Example

```
<?php
function sum($x, $y) {
    $z = $x + $y;
    return $z;
}

echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = " . sum(2, 4);
?>
```

An array stores multiple values in one single variable:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

```
$cars1 = "Volvo";
$cars2 = "BMW";
$cars3 = "Toyota";
```

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

```
array();
```

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

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Get The Length of an Array - The count() Function

The count() function is used to return the length (the number of elements) of an array:

Example

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

```
echo count($cars);
?>
```

Loop Through an Indexed Array

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

Example

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

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

PHP Associative Arrays

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

There are two ways to create an associative array:

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

or:

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

The named keys can then be used in a script:

Example

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

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:

Example

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

foreach($age as $x => $x_value) {
   echo "Key=" . $x . ", Value=" . $x_value;
   echo "<br>";
}
?>
```

Multidimensional Arrays

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

PHP understands 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.

The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

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):

Example

```
<?php
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>";
?>
```

Run example »

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):

Example

```
<?php
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>";
```

```
}
?>
```

Example of Associative array

In this example we create a two dimensional array to store marks of three students in three subjects:

This example is an associative array, you can create numeric array in the same fashion.

```
<html>
<body>
<?php

   $marks = array(
   "mohammad" => array (
                  "physics" => 35,
                  "maths" => 30,
                  "chemistry" => 39
                  ),

    "qadir" => array (
                  "physics" => 30,
                  "maths" => 32,
                 "chemistry" => 29
                  ),

    "zara" => array (
                  "physics" => 31,
                  "maths" => 22,
                  "chemistry" => 39
                  );

   /* Accessing multi-dimensional array values */ echo
   "Marks for mohammad in physics : " ;
```

This will produce following result:

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

Some Array Functions

1. PHP array_key_exists() Function

Example

Check if the key "Volvo" exists in an array:

```
<?php
$a=array("Volvo"=>"XC90","BMW"=>"X5");
if (array_key_exists("Volvo",$a))
  {
  echo "Key exists!";
```

```
 }
else
 {
 echo "Key does not exist!";
 }
?>
```

2. array_merge() Function

The array_merge() function merges one or more arrays into one array.

Tip: You can assign one array to the function, or as many as you like.

Note: If two or more array elements have the same key, the last one overrides the others.

Note: If you assign only one array to the array_merge() function, and the keys are integers, the function returns a new array with integer keys starting at 0 and increases by 1 for each value (See Example 2 below).

Tip: The difference between this function and the array_merge_recursive() function is when two or more array elements have the same key. Instead of override the keys, the array_merge_recursive() function makes the value as an array.

Syntax

array_merge(*array1,array2,array3...*)

Example

Merge two arrays into one array:

```
<?php
$a1=array("red","green");
$a2=array("blue","yellow");
print_r(array_merge($a1,$a2));
?>
```

3. The array_pad() function

It inserts a specified number of elements, with a specified value, to an array.

Tip: If you assign a negative size parameter, the function will insert new elements BEFORE the original elements (See example below).

Note: This function will not delete any elements if the size parameter is less than the size of the original array.

Syntax

array_pad(*array,size,value*)

Example

Return 5 elements and insert a value of "blue" to the new elements in the array:

```
<?php
$a=array("red","green");
print_r(array_pad($a,5,"blue"));
?>
```

4. The array_push() function

It inserts one or more elements to the end of an array.

Tip: You can add one value, or as many as you like.

Note: Even if your array has string keys, your added elements will always have numeric keys (See example below).

Syntax

array_push(*array,value1,value2...*)

Example

Insert "blue" and "yellow" to the end of an array:

```
<?php
$a=array("red","green");
array_push($a,"blue","yellow");
print_r($a);
?>
```

5. The array_pop() function

It deletes the last element of an array.

Syntax: array_pop(*array*)

Example

Delete the last element of an array:

```
<?php
$a=array("red","green","blue");
array_pop($a);
print_r($a);
?>
```

6. array_reverse() function

It returns an array in the reverse order.

Syntax**:** array_reverse(*array*)

Example

Return an array in the reverse order:

```
<?php
$a=array("a"=>"Volvo","b"=>"BMW","c"=>"Toyota");
print_r(array_reverse($a));
?>
```

7. The array_search() function

It search an array for a value and returns the key.

Syntax :array_search(*value,array*)

Example

Search an array for the value "red" and return its key:

```
<?php
$a=array("a"=>"red","b"=>"green","c"=>"blue");
echo array_search("red",$a);
?>
```

8. The sort() function

It sorts an indexed array in ascending order.

Tip: Use the rsort() function to sort an indexed array in descending order.

Example

Sort the elements of the $cars array in ascending alphabetical order:

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

9. The asort() function

It sorts an associative array in ascending order, according to the value.

Tip: Use the arsort() function to sort an associative array in descending order, according to the value.

Tip: Use the ksort() function to sort an associative array in ascending order, according to the key.

Example

Sort an associative array in ascending order, according to the value:

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

10.The count() function

It returns the number of elements in an array.

Example

Return the number of elements in an array:

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

PHP 5 Global Variables - Superglobals

Superglobals were introduced in PHP 4.1.0, and are built-in variables that are always available in all scopes.

PHP Global Variables - Superglobals

Several predefined variables in PHP 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.

The PHP superglobal variables are:

- $GLOBALS
- $_SERVER
- $_REQUEST
- $_POST
- $_GET
- $_FILES
- $_ENV
- $_COOKIE

- $_SESSION

This chapter will explain some of the superglobals, and the rest will be explained in later chapters.

PHP $GLOBALS

$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).

PHP stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable.

The example below shows how to use the super global variable $GLOBALS:

Example

```
<?php
$x = 75;
$y = 25;

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

addition();
echo $z;
?>
```

In the example above, since z is a variable present within the $GLOBALS array, it is also accessible from outside the function!

PHP $_SERVER

$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.

The example below shows how to use some of the elements in $_SERVER:

Example

```
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
```

```
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```

The following table lists the most important elements that can go inside $_SERVER:

Element/Code	Description
$_SERVER['PHP_SELF']	Returns the filename of the currently executing script
$_SERVER['GATEWAY_INTERFACE']	Returns the version of the Common Gateway Interface (CGI) the server is using
$_SERVER['SERVER_ADDR']	Returns the IP address of the host server
$_SERVER['SERVER_NAME']	Returns the name of the host server (such as www.w3schools.com)
$_SERVER['SERVER_SOFTWARE']	Returns the server identification string (such as Apache/2.2.24)
$_SERVER['SERVER_PROTOCOL']	Returns the name and revision of the information protocol (such as HTTP/1.1)
$_SERVER['REQUEST_METHOD']	Returns the request method used to access the page (such as POST)
$_SERVER['REQUEST_TIME']	Returns the timestamp of the start of the request (such as 1377687496)
$_SERVER['QUERY_STRING']	Returns the query string if the page is accessed via a query string
$_SERVER['HTTP_ACCEPT']	Returns the Accept header from the current request
$_SERVER['HTTP_ACCEPT_CHARSET']	Returns the Accept_Charset header from the current request (such as utf-8,ISO-8859-1)
$_SERVER['HTTP_HOST']	Returns the Host header from the current request
$_SERVER['HTTP_REFERER']	Returns the complete URL of the current page (not reliable because not all user-agents support it)
$_SERVER['HTTPS']	Is the script queried through a secure HTTP protocol

$_SERVER['REMOTE_ADDR']	Returns the IP address from where the user is viewing the current page
$_SERVER['REMOTE_HOST']	Returns the Host name from where the user is viewing the current page
$_SERVER['REMOTE_PORT']	Returns the port being used on the user's machine to communicate with the web server
$_SERVER['SCRIPT_FILENAME']	Returns the absolute pathname of the currently executing script
$_SERVER['SERVER_ADMIN']	Returns the value given to the SERVER_ADMIN directive in the web server configuration file (if your script runs on a virtual host, it will be the value defined for that virtual host) (such as someone@w3schools.com)
$_SERVER['SERVER_PORT']	Returns the port on the server machine being used by the web server for communication (such as 80)
$_SERVER['SERVER_SIGNATURE']	Returns the server version and virtual host name which are added to server-generated pages
$_SERVER['PATH_TRANSLATED']	Returns the file system based path to the current script
$_SERVER['SCRIPT_NAME']	Returns the path of the current script
$_SERVER['SCRIPT_URI']	Returns the URI of the current page

PHP $_REQUEST

PHP $_REQUEST is used to collect data after submitting an HTML form.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to this file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_REQUEST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_REQUEST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

PHP $_POST

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // collect value of input field
    $name = $_POST['fname'];
    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo $name;
    }
}
?>

</body>
</html>
```

PHP $_POST

PHP $_POST is widely used to collect form data after submitting an HTML form with method="post". $_POST is also widely used to pass variables.

The example below shows a form with an input field and a submit button. When a user submits the data by clicking on "Submit", the form data is sent to the file specified in the action attribute of the <form> tag. In this example, we point to the file itself for processing form data. If you wish to use another PHP file to process form data, replace that with the filename of your choice. Then, we can use the super global variable $_POST to collect the value of the input field:

Example

```
<html>
<body>

<form method="post" action="<?php echo $_SERVER['PHP_SELF'];?>">
  Name: <input type="text" name="fname">
  <input type="submit">
</form>

<?php
```

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   // collect value of input field
   $name = $_POST['fname'];
   if (empty($name)) {
      echo "Name is empty";
   } else {
      echo $name;
   }
}
?>

</body>
</html>
```

Two HTTP Request Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

- **GET** - Requests data from a specified resource
- **POST** - Submits data to be processed to a specified resource

---

The GET Method

Note that the query string (name/value pairs) is sent in the URL of a GET request:

/test/demo_form.php**?name1=value1&name2=value2**

Some other notes on GET requests:

- GET requests can be cached
- GET requests remain in the browser history
- GET requests can be bookmarked
- GET requests should never be used when dealing with sensitive data
- GET requests have length restrictions
- GET requests should be used only to retrieve data

The POST Method

Note that the query string (name/value pairs) is sent in the HTTP message body of a POST request:

POST /test/demo_form.php HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2

Some other notes on POST requests:

- POST requests are never cached
- POST requests do not remain in the browser history
- POST requests cannot be bookmarked
- POST requests have no restrictions on data length

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)

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
<?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.

Cookies	Sessions
Cookies are stored in browser as text file format.	Sessions are stored in server side.
It is stored limit amount of data.	It is stored unlimited amount of data
It is only allowing 4kb[4096bytes].	It is holding the multiple variable in sessions.
It is not holding the multiple variable in cookies.	It is holding the multiple variable in sessions.
we can accessing the cookies values in easily. So it is less secure.	we cannot accessing the session values in easily.So it is more secure.
setting the cookie time to expire the cookie.	using session_destory(), we we will destroyed the sessions.
The setcookie() function must appear BEFORE the <html> tag.	The session_start() function must be the very first thing in your document. Before any HTML tags.

what's an error?

An error is a type of mistake. We can say an error is a condition of having incorrect or false knowledge or an error is defined as an unexpected, invalid program state from which it is impossible to recover.

Error can also be defined as "a deviation from accuracy or correctness".

A "mistake" is an error caused by a fault: the fault being misjudgment, carelessness, or forgetfulness. An error message with filename, line number and a message describing the error is sent to the browser.

What are the different kinds of errors in PHP?

Basically there are four kinds of errors in PHP, which are as follows:

- **Parse Error (Syntax Error):** The parse error occurs if there is a syntax mistake in the script; the output is Parse error. A parse error stops the execution of the script. There are many reasons for the occurrence of parse errors in PHP. The common reasons for parse errors are as follows:

1. Unclosed quotes
2. Missing or Extra parentheses
3. Unclosed braces
4. Missing semicolon

Example:

```
<?php
echo "Cat";
echo "Dog"
echo "Lion";
?>
```

Output:

In the above code we missed the semicolon in the second line. When that happens there will be a parse or syntax error which stops execution of the script.

- **Fatal Error:** Fatal errors are caused when PHP understands what you've written, however what you're asking it to do can't be done. Fatal errors stop the execution of the script. If you are trying to access the undefined functions, then the output is a fatal error.

Example:

```
<?php
function fun1()
{
echo "Vineet Saini";
}
fun2();
echo "Fatal Error !!";
?>
```

Output:

In the above code we defined a function fun1 but we call another function fun2 i.e. func2 is not defined. So a fatal error will be produced that stops the execution of the script.

- **Warning Error:** Warning errors will not stop execution of the script. The main reason for warning error is to include a missing file or using the incorrect number of parameters in a function.

Example:

```
<?php
echo "Warning Error!!";
include ("Welcome.php");
?>
```

Output:

In the above code we include a welcome.php file, however the welcome.php file does not exist in the directory. So there will be a warning error produced but that does not stop the execution of the script

- **Notice Error:** Notice error is the same as a warning error i.e. in the notice error, execution of the script does not stop. Notice that the error occurs when you try to access the undefined variable, then produce a notice error.

Example:

```
<?php
$a="Vineet kumar saini";
echo "Notice Error !!";
echo $b;
?>
```

Output:

In the above code we defined a variable which named $a. But we call another variable i.e. $b, which is not defined. So there will be a notice error produced but execution of the script does not stop, you will see a message Notice Error!!

Basically there are 13 types of errors in PHP, which are as follows:

1. **E_ERROR**: A fatal error that causes script termination
2. **E_WARNING:** Run-time warning that does not cause script termination
3. **E_PARSE:** Compile time parse error
4. **E_NOTICE:** Run time notice caused due to error in code
5. **E_CORE_ERROR:** Fatal errors that occur during PHP's initial startup (installation)
6. **E_CORE_WARNING:** Warnings that occur during PHP's initial startup
7. **E_COMPILE_ERROR:** Fatal compile-time errors indication problem with script
8. **E_USER_ERROR:** User-generated error message
9. **E_USER_WARNING:** User-generated warning message
10. **E_USER_NOTICE**: User-generated notice message
11. **E_STRICT:** Run-time notices
12. **E_RECOVERABLE_ERROR:** Catchable fatal error indicating a dangerous error
13. **E_ALL:** Catches all errors and warnings

Hope by now, you have a clear understanding of the different types and kinds of errors in PHP…

☐ **Authentication** is the process of verifying who you are. When you log on to a PC with a user name and password you are authenticating.

☐ **Authorization** is the process of verifying that you have access to something. Gaining access to a resource (e.g. directory on a hard disk) because the permissions configured on it allow you access is authorization

PHP include and require Statements

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

The include and require statements are identical, except upon failure:

- require will produce a fatal error (E_COMPILE_ERROR) and stop the script
- include will only produce a warning (E_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

Syntax

include '*filename*';
or
require '*filename*';

Assume we have a standard menu file called "menu.php":

```
<?php
echo '<a href="/default.asp">Home</a> -
<a href="/html/default.asp">HTML Tutorial</a> -
<a href="/css/default.asp">CSS Tutorial</a> -
<a href="/js/default.asp">JavaScript Tutorial</a> -
<a href="default.asp">PHP Tutorial</a>';
?>
```

All pages in the Web site should use this menu file. Here is how it can be done (we are using a element so that the menu easily can be styled with CSS later):

Example

```
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
```

```
<p>Some more text.</p>

</body>
</html>
```

What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers.

HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

PHP 5 Form Handling
The PHP superglobals $_GET and $_POST are used to collect form-data.
The example below displays a simple HTML form with two input fields and a submit button:

Example

```
<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:

```
<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>
```

GET vs. POST

Both GET and POST create an array (e.g. array(key => value, 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.

The PHP Date() Function

The PHP date() function formats a timestamp to a more readable date and time.

Syntax

date(format,timestamp)

format : Required. Specifies the format of the timestamp

timestamp : Optional. Specifies a timestamp. Default is the current date and time

Get a Simple Date

The required format parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

- d - Represents the day of the month (01 to 31)
- m - Represents a month (01 to 12)
- Y - Represents a year (in four digits)
- l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

Example :

```
<?php
echo "Today is " . date("Y/m/d") . "<br>";
echo "Today is " . date("Y.m.d") . "<br>";
echo "Today is " . date("Y-m-d") . "<br>";
echo "Today is " . date("l");
?>
```

Get a Simple Time

Here are some characters that are commonly used for times:

- h - 12-hour format of an hour with leading zeros (01 to 12)
- i - Minutes with leading zeros (00 to 59)
- s - Seconds with leading zeros (00 to 59)
- a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

```
<?php
echo "The time is " . date("h:i:sa");
?>
```

PHP 5 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.

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

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

The readfile() function is useful if all you want to do is open up a file and read its contents.

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. 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:

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

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.

PHP Close File - fclose()

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

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:

```
<?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 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:

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

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":

```
<?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.

PHP MySQL Database

With PHP, you can connect to and manipulate databases.

MySQL is the most popular database system used with PHP.

What is MySQL?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Databases are useful for storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

Database Queries

A query is a question or a request.

We can query a database for specific information and have a recordset returned.

Look at the following query (using standard SQL):

SELECT LastName FROM Employees

Facts About MySQL Database

MySQL is the de-facto standard database system for web sites with HUGE volumes of both data and end-users (like Facebook, Twitter, and Wikipedia).

Another great thing about MySQL is that it can be scaled down to support embedded database applications.

PHP 5 and later can work with a MySQL database using:

- MySQLi extension (the "i" stands for improved)

- PDO (PHP Data Objects)

Earlier versions of PHP used the MySQL extension.

Should I Use MySQLi or PDO?

If you need a short answer, it would be "Whatever you like".

Both MySQLi and PDO have their advantages:

PDO will work on 12 different database systems, whereas MySQLi will only work with MySQL databases.

So, if you have to switch your project to use another database, PDO makes the process easy. You only have to change the connection string and a few queries. With MySQLi, you will need to rewrite the entire code - queries included.

Both are object-oriented, but MySQLi also offers a procedural API.

Both support Prepared Statements. Prepared Statements protect from SQL injection, and are very important for web application security.

Open a Connection to MySQL

Before we can access data in the MySQL database, we need to be able to connect to the server:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Example (PDO)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
   $conn = new PDO("mysql:host=$servername;dbname=myDB", $username, $password);
```

```
    // set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
    }
catch(PDOException $e)
    {
    echo "Connection failed: " . $e->getMessage();
    }
?>
```

Tip: A great benefit of PDO is that it has an exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows directly to the first catch(){ } block.

Close the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

Example (MySQLi Object-Oriented) : $conn->close();

PDO : $conn = null;

PHP Create MySQL Tables

The CREATE TABLE statement is used to create a table in MySQL.

We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```

Notes on the table above:

The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.

After the data type, you can specify other optional attributes for each column:

- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed
- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero

- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

// sql to create table
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)";

if (mysqli_query($conn, $sql))
{
   echo "Table MyGuests created successfully";
}
else
 {
   echo "Error creating table: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

PHP Insert Data Into MySQL

Insert Data Into MySQL Using MySQLi and PDO

After a database and a table have been created, we can start adding data in them.

Here are some syntax rules to follow:

- The SQL query must be quoted in PHP

- String values inside the SQL query must be quoted
- Numeric values must not be quoted
- The word NULL must not be quoted

The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

To learn more about SQL, please visit our SQL tutorial.

In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

Note: If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn)
{
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if (mysqli_query($conn, $sql))
{
   echo "New record created successfully";
}
else
{
   echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Select Data From a MySQL Database

The SELECT statement is used to select data from one or more tables:

SELECT column_name(s) FROM table_name

or we can use the * character to select ALL columns from a table:

SELECT * FROM table_name

The following example selects the id, firstname and lastname columns from the MyGuests table and displays it on the page:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);

if (mysqli_num_rows($result) > 0)
{
   // output data of each row
   while($row = mysqli_fetch_assoc($result)) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"].
"<br>";
   }
}
else
 {
   echo "0 results";
}

mysqli_close($conn);
?>
```

Delete Data From a MySQL Table

The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

To learn more about SQL, please visit our SQL tutorial.

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

The following examples delete the record with id=3 in the "MyGuests" table:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn)
{
   die("Connection failed: " . mysqli_connect_error());
}

// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";

if (mysqli_query($conn, $sql))
{
   echo "Record deleted successfully";
}
else
{
   echo "Error deleting record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Update Data In a MySQL Table

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

To learn more about SQL, please visit our SQL tutorial.

Let's look at the "MyGuests" table:

id	firstname	lastname	email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection
if (!$conn)
{
   die("Connection failed: " . mysqli_connect_error());
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if (mysqli_query($conn, $sql))
{
   echo "Record updated successfully";
}
else
{
   echo "Error updating record: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```