PHP

Constructure

The php constructs like this:

```
<?php
content
?>
```

It will appear in the file that the extend name is .php,but note that php can not exist in html, however the .php file has the same constructor and content as html beside it can also be a container of php, like the jsp in J2ee.

variable

In php, the variable starts with '$', si=o it's format is like this:

```
$x= 'Hello World!';
$y=24;
$a=3.14;
```

Global scope

When we set a variable outside of functions, we call it a global variable, not like the other programming languages, at this place the global variable cannot be invoked in a function directly. We could invoke them by passing parmeters to the function or we could use "global" array that contains all the global variables, and it's index is the names of the global variables, so we could use "global[x]" to invoke the variable "$x" above.

Local scope

We could set a global variable, so we could also set a local variable, and like the other programming languages, the local variable can only be used inside a function which one includes it.

Static keyword

In php, if we set a local variable, and then we edit it after we invoked this variable, the next time we invoke this function again, we will found that this variable will be reset again,like this:

```
function Add() {
        $a=1;
        echo $a,"<br/>";
        $a++;
    }
    Add();
    Add();
```

We will always get the same result of 1, but after we asign this $a to a static one, the result will become 1 and 2 .

echo and print

In php, there are two different ways to output the data to the screen: the echo and print. Their deference are that print has a return value of 1, so it can be used in expressions and echo can take not only one parameter(rare use). And the html makeup is permited in both of two methods

```

```

PHP array

In php, we do not need use new to create a new array, it means we will create a array like this:

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

PHP Object

In php we use class as a template of an object and take the object as an instance of a class. And the construct function will have the same importance as in Java. But but before we build a constructor, there should be two underline before, and when we set or invoke the properties and methods of the object, we use -> instead of dot.

String's methods

strlen(String):

We use it to get the length of the string

str_word_count(String)

We use it to get the counts of the number of words in string.

strrev(String)

We use it to reverse the string

strpos(current String,target String)

We use it to get the start index of the of the specific String in current String

str_replace(target String,substitute String,current String)

We will use the substitute string to replace the target String in current String

PHP Numbers

Integer

An integer data type is a non-decimal number between -2147483648 and 2147483647 in 32 bit systems, and between -9223372036854775808 and 9223372036854775807 in 64 bit systems.
And there are three constants in integer:
1.PHP_INT_MAX - The largest integer supported
2.PHP_INT_MIN - The smallest integer supported
3.PHP_INT_SIZE - The size of an integer in bytes

PHP has the following functions to check if the type of a variable is integer:

is_int(int)
is_integer(int) - alias of is_int()
is_long(int) - alias of is_int()

PHP has the following functions to check if the type of a variable

is float:
is_float()
is_double() - alias of is_float()

PHP Infinity

A numeric value that is larger than PHP_FLOAT_MAX is considered infinite.
PHP has the following functions to check if a numeric value is finite or infinite:
is_finite()
is_infinite()

Note:var_dump() function returns the data type and value:

PHP NaN

NaN stands for Not a Number. NaN is used for impossible mathematical operations.PHP has the following functions to check if a value is not a number: is_nan()

PHP Numerical Strings

The PHP is_numeric() function can be used to find whether a variable is numeric. The function returns true if the variable is a number or a numeric string, false otherwise.

Note: From PHP 7.0: The is_numeric() function will return FALSE for numeric strings in hexadecimal form (e.g. 0xf4c3b00c), as they are no longer considered as numeric strings.

PHP Math

1.The pi() function returns the value of PI
2.The min() and max() functions can be used to find the lowest or highest value in a list of arguments.
3.The abs() function returns the absolute (positive) value of a number
4.The sqrt() function returns the square root of a number
5.The round() function rounds a floating-point number to its nearest integer:

```
<?php
echo(round(0.60));  // returns 1
echo(round(0.49));  // returns 0
?>
```

6.The rand([from,to]) function generates a random number

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

```
//define(name, value, case-insensitive)
define("BESTFRI","TAO XUETING",false);
```

In PHP7, you can create an Array constant using the define() function:

```
<?php
define("cars", [
  "Alfa Romeo",
  "BMW",
  "Toyota"
]);
?>
```

PHP divides the operators in the following groups:

1.Arithmetic operators
2.Assignment operators
3.Comparison operators
4.Increment/Decrement operators
5.Logical operators
6.String operators
7.Array operators
8.Conditional assignment operators

PHP Arithmetic Operators

Operator	Name	Example	Result
+	Addition	$x+$y	Sum of both values
-	Substraction	$x-$y	Difference of both values
*	Multiplication	$x*$y	Product of both values
/	Division	$x/$y	Qutient of both values
%	Modulus	$x%$y	Remainder of $x devided by $y
**	Exponentition	$x**$y	Result of raising $x to the $y'th power

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable.The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

Assignment	Same as	Description
x=y	x=y	The left operand gets set to the value of the expression on the right
x+=y	x=x+y	Addition
x-=y	x=x-y	Substraction
x*=y	x=x*y	Multiplication
x/=y	x=x/y	Division
x%=y	x=x%y	Modulus

PHP Comparison Operators

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

Operator	Name	Example	Result
==	Equal	$X==$y	Return true if $x is equal to $y
===	Identical	$x==$y	Return true if $x is equal to $y, and they are the samle type
!=	Not equal	$x!=$y	Returns true is $x in not equal to $y
<>	Not Equal	$x<>$y	Returns true is $x in not equal to $y
!==	Not Identical	$x!==$y	Returns true is $x in not equal to $y, or they are not the same type
>	Greater Than	$x>$y	Returns true if $x is greater than $y

<	Less Than	$x<$y< td>	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
<=< td>	Less Than	$x<=$y< td>	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.

Spaceship example

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

echo ($x <=> $y); // returns -1 because $x is less than $y
echo "<br>";

$x = 10;
$y = 10;

echo ($x <=> $y); // returns 0 because values are equal
echo "<br>";

$x = 15;
$y = 10;

echo ($x <=> $y); // returns +1 because $x is greater than $y
?>
```

HP Increment / Decrement Operators

Operator	Name	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--	Psot-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 conditions 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
?:	Tenary	$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

IF ElSE in PHP

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:

1.if statement - executes some code if one condition is true
2.if...else statement - executes some code if a condition is true and another code if 3.that condition is false
3.if...elseif...else statement - executes different codes for more than two conditions
4.switch statement - selects one of many blocks of code to be executed

SWITCH in PHP

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

```
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;
}
```

Loops in PHP

In PHP, we have the following loop types:
1.while - loops through a block of code as long as the specified condition is true

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

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

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

3.for - loops through a block of code a specified number of times

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

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

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

PHP is a Loosely Typed Language

PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing an error.

In PHP 7, type declarations were added. This gives us an option to specify the expected data type when declaring a function, and by adding the strict declaration, it will throw a "Fatal Error" if the data type mismatches.

In the following example we try to send both a number and a string to the function without using strict:

```
//Do not add strict declaration
<?php
function addNumbers($a,$b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is changed to int(5), and it will return
10
?>
```

```
//Set the type, but not use the strict declaration
<?php
function addNumbers(int $a, int $b) {
  return $a + $b;
}
echo addNumbers(5, "5 days");
/* Cast a fatal error, but we can still add a String like "5" to the parameter, it
will run correctly*/
?>
```

```
//Add strict declaration
<?php declare(strict_types=1);/*declare() must be the very first line of the PHP
file including html code*/
function addNumbers(int $a,int $b) {
  return $a + $b;
}
echo addNumbers(5, 5);
// We have to give all parameters with matched type
?>
```

PHP Return Type Declarations

PHP 7 also supports Type Declarations for the return statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.
To declare a type for the function return, add a colon (:) and the type right before the opening curly ({)bracket when declaring the function.
In the following example we specify the return type for the function:

```
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
  return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

Passing Arguments by Reference

In PHP, arguments are usually passed by value, which means that a copy of the value is used in the function and the variable that was passed into the function cannot be changed.

When a function argument is passed by reference, changes to the argument also change the variable that was passed in. To turn a function argument into a reference, the & operator is used:

```
<?php
function add_five(&$value) {
  $value += 5;
}

$num = 2;
```

```
add_five($num);
echo $num;
?>
```

Array

Create an Array in PHP

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

```
<?php
//array();
$cars = array("Volvo", "BMW", "Toyota");//PHP Indexed Arrays
?>
```

In PHP, there are three types of arrays:
1.Indexed arrays - Arrays with a numeric index

```
<?php
//array();
$cars = array("Volvo", "BMW", "Toyota");//PHP Indexed Arrays
?>
//loop array
for($x = 0; $x < $arrlength; $x++) {
  echo $cars[$x];
  echo "<br>";
}
```

2.Associative arrays - Arrays with named keys

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
or
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
foreach($age as $x => $value) {
  echo "Key=" . $x . ", Value=" . $value;
  echo "<br>";
}
```

3.Multidimensional arrays - Arrays containing one or more arrays

```
$cars = array (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
);
//loop array:
for ($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>";
}
```

Methods in array

count(array)-used to return the length (the number of elements) of an array
sort() - sort arrays in ascending order
rsort() - sort arrays in descending order
asort() - sort associative arrays in ascending order, according to the value
arsort() - sort associative arrays in descending order, according to the value
ksort() - sort associative arrays in ascending order, according to the key
krsort() - sort associative arrays in descending order, according to the key

PHP Superglobal

Super global variables are built-in variables that are always available in all scopes.

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:

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

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

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

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:

```php
<?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 result is:

```
/demo/demo_global_server.php
35.194.26.41
35.194.26.41
https://tryphp.w3schools.com/showphp.php?filename=demo_global_server
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:99.0) Gecko/20100101 Firefox/99.0
/demo/demo_global_server.php
```

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 a PHP super global variable which 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:
Here's an 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

Both _POST and $_REQUEST are the super global variable, and they are designed to be retrive the values in matching method, and here's a simple 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 $_GET

PHP $_GET is a PHP super global variable which is used to collect form data after submitting an HTML form with method="get".

$_GET can also collect data sent in the URL.

Assume we have an HTML page that contains a hyperlink with parameters:

```
<html>
<body>

<a href="test_get.php?subject=PHP&web=W3schools.com">Test $GET</a>

</body>
</html>
```

When a user clicks on the link "Test $GET", the parameters "subject" and "web" are sent to "test_get.php", and you can then access their values in "test_get.php" with $_GET.

The example below shows the code in "test_get.php":

```
<html>
<body>

<?php
echo "Study " . $_GET['subject'] . " at " . $_GET['web'];
?>

</body>
</html>
```

Regular Expression

A regular expression is a sequence of characters that forms a search pattern. When you search for data in a text, you can use this search pattern to describe what you are searching for.

A regular expression can be a single character, or a more complicated pattern.

Regular expressions can be used to perform all types of text search and text replace operations.

In PHP, regular expressions are strings composed of delimiters, a pattern and optional modifiers.

```
$exp = "/w3schools/i";
```

In the example above, / is the delimiter, w3schools is the pattern that is being searched for, and i is a modifier that makes the search case-insensitive.

The delimiter can be any character that is not a letter, number, backslash or space. The most common delimiter is the forward slash (/), but when your pattern contains forward slashes it is convenient to choose other delimiters such as # or ~.

Regular Expression Functions

Function	Description
preg_match(pattern,target)	Returns 1 if the pattern was found in the string and 0 if not
preg_match_all(pattern,target)	Returns the number of times the pattern was found in the string, which may also be 0
preg_replace(pattern,replacement,target)	Returns a new string where matched patterns have been replaced with another string

Regular Expression Modifiers

Modifer	Description
i	Performs a case-insensitive search
m	Performs a multiline search (patterns that search for the beginning or end of a string will match the beginning or end of each line)
u	Enables correct matching of UTF-8 encoded patterns

Expression	Description
[abc]	Find one character from the options between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find one character from the range 0 to 9

Matacharacter	Description
|	Find a match for any one of the patterns separated by | as in: cat|dog|fish
.	Find a match for any one of the patterns separated by | as in: cat|dog|fish
^	Finds a match as the beginning of a string as in: ^Hello
$	Finds a match at the end of the string as in: World$
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers

Quantifiers define quantities:

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n
n{x}	Matches any string that contains a sequence of X n's
n{x,y}	Matches any string that contains a sequence of X to Y n's
n{x,}	Matches any string that contains a sequence of at least X n's

PHP Forms

We have learned when we want retrive the data from client-side in the server, we could use

```
$_GET['name']
```

to get the data sent with GET method and use

```
$_POST['name']
```

to get the data sent with POST method.
Now we start to learn the PHP Form Validation:

Method: htmlspecialchars(str)

This method is foremost important for preventing XSS attacks. So what is a XSS attack? Fore Example:

```
/*It is a server page:*/
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = $_POST["name"];
  $email = $_POST["email"];
  $website = $_POST["website"];
  $comment = $_POST["comment"]);
  $gender = $_POST["gender"];
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
```

```
    $data = htmlspecialchars($data);
    return $data;
  }
  ?>

  <?php
  echo "<h2>Your Input:</h2>";
  echo $name;
  echo "<br>";
  echo $email;
  echo "<br>";
  echo $website;
  echo "<br>";
  echo $comment;
  echo "<br>";
  echo $gender;
  ?>
```

This is a client-side page:

```

<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
<h2>PHP Form Validation Example</h2>
<form method="post" action="<?php echo "target.php"?>">
  Name: <input type="text" name="name">
  <br><br>
  E-mail: <input type="text" name="email">
  <br><br>
  Website: <input type="text" name="website">
  <br><br>
  Comment: <textarea name="comment" rows="5" cols="40"></textarea>
  <br><br>
  Gender:
  <input type="radio" name="gender" value="female">Female
  <input type="radio" name="gender" value="male">Male
  <input type="radio" name="gender" value="other">Other
  <br><br>
  <input type="submit" name="submit" value="Submit">
</form>

</body>
</html>
```

When a hacker add a string:"<script>alert("Hello")</script>" after your input . After you submit the form to the server, this script will be executed. And you will find this alert on the page.But when you use method: test_input($data) everything will change:

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>

<?php
echo "<h2>Your Input:</h2>";
echo $name;
echo "<br>";
echo $email;
echo "<br>";
echo $website;
echo "<br>";
echo $comment;
echo "<br>";
echo $gender;
?>
```

Beacause when you are using a htmlspecialchars, all the <>will become &lt and &gt, so the script won't be executed.

Note:When we are using $_SERVER["PHP_SELF"], we need add a htmlspecialchars too in order to protect the submit.

And when we want to set the Required attribute, we could do something like this:

```
<?php
// define variables and set to empty values
$nameErr = $emailErr = $genderErr = $websiteErr = "";
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  if (empty($_POST["name"])) {
    $nameErr = "Name is required";
  } else {
    $name = test_input($_POST["name"]);
  }

  if (empty($_POST["email"])) {
    $emailErr = "Email is required";
  } else {
    $email = test_input($_POST["email"]);
  }

  if (empty($_POST["website"])) {
    $website = "";
  } else {
    $website = test_input($_POST["website"]);
  }

  if (empty($_POST["comment"])) {
    $comment = "";
  } else {
    $comment = test_input($_POST["comment"]);
  }

  if (empty($_POST["gender"])) {
    $genderErr = "Gender is required";
  } else {
    $gender = test_input($_POST["gender"]);
  }
}
?>
<?php
function test_input($data){
  $data=htmlspecialchars($data);
  return $data;
}
?>
<form method="post" action="<?php echo htmlspecialchars($_SERVER["PHP_SELF"]);?>">

Name: <input type="text" name="name">
<span class="error">* <?php echo $nameErr;?></span>
<br><br>
E-mail:
<input type="text" name="email">
<span class="error">* <?php echo $emailErr;?></span>
<br><br>
Website:
<input type="text" name="website">
<span class="error"><?php echo $websiteErr;?></span>
<br><br>
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
<br><br>
Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male
<input type="radio" name="gender" value="other">Other
<span class="error">* <?php echo $genderErr;?></span>
<br><br>
```

```
<input type="submit" name="submit" value="Submit">

</form>
```

Date and Time

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

```
date(format,[timestamp])
```

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.

The example below formats today's date in three different ways:

```
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 Time

Here are some characters that are commonly used for times:

H - 24-hour format of an hour (00 to 23) 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");
?>
```

Get Your Time Zone

If the time you got back from the code is not correct, it's probably because your server is in another country or set up for a different timezone.

So, if you need the time to be correct according to a specific location, you can set the timezone you want to use.

The example below sets the timezone to "America/New_York", then outputs the current time in the specified format:

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

Create a Date With mktime() The optional timestamp parameter in the date() function specifies a timestamp. If omitted, the current date and time will be used (as in the examples above).

The PHP mktime() function returns the Unix timestamp for a date. The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

```
mktime(hour, minute, second, month, day, year)
```

The example below creates a date and time with the date() function from a number of parameters in the mktime() function:

```
<?php
$d=mktime(11, 14, 54, 8, 12, 2014);
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

Create a Date From a String With strtotime()

The PHP strtotime() function is used to convert a human readable date string into a Unix timestamp (the number of seconds since January 1 1970 00:00:00 GMT).

```
strtotime(time, now)
```

Example:

```
<?php
$d=strtotime("10:30pm April 15 2014");
echo "Created date is " . date("Y-m-d h:i:sa", $d);
?>
```

PHP is clever to convert the date string:

```
<?php
$d=strtotime("tomorrow");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("next Saturday");
echo date("Y-m-d h:i:sa", $d) . "<br>";

$d=strtotime("+3 Months");
echo date("Y-m-d h:i:sa", $d) . "<br>";
?>
```

Include in PHP

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

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

```
include 'filename';

or

require 'filename';
```

Here's an example:

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

We assume the code above is in the file called footer.php

```
<html>
<body>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>
<?php include 'footer.php';?>
//Now we have included the code above into here
</body>
</html>
```

And we mentioned that expect Include we also have require to include the file into the page, but they have a big difference: When the include file is not found, use include, the script will continue to work, but when use the require, the script below will not work anymore.

ReadFile(path)

Use this file we could get all the content of the file in txt and at the end of the text file will display the total bytes of the file

To open/read a file we can also use fopen() which has two parameters: one for the filename and another for options, here's all the options:

Modes	Description
r	Open a file for read only. File pointer starts at the beginning of the file
r+	Open a file for read/write. 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
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 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
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 write only. Returns FALSE and an error if file already exists
x+	Creates a new file for read/write. Returns FALSE and an error if file already exists

PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

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

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

filesize(file):return size of the file

After we manipulate the file, we shou use fclose(filename) to close the file to prevent it using the resources of the server.

```
<?php
$myfile = fopen("webdictionary.txt", "r");
// some code to be executed....
fclose($myfile);
?>
```

fgets(file)

This method is used to read the single line from the file, so in order to read the whole file we need use an another method called feof(file) that will check if the "end-of-file"(EOF) has been reached. And here's an example for that:

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

PHP Read Single Character - fgetc():

With this function we could read a single character of the file;

File create and write in PHP:

When we are using fopen(), if there is no such file, it will be created. So we create a new file with the function fopen() too. But when we need to write this new file, we need to use method: fwrite(file,content).

PHP Overwriting and Append

When we finished writing a new file. If we start writing it again, besides the fopen()'s option is "w", the initial data will all be overwrited, we call it Overwriting. And when you choose the option "a", the initial data will be preserved.

File Upload:

Beacause the file upload is a little complicated, so we firstly showcase a example here:

```
//index page
<!DOCTYPE html>
<html>
<body>

<form action="upload.php" method="post" enctype="multipart/form-data">
```

```
  Select image to upload:
  <input type="file" name="fileToUpload" id="fileToUpload">
  <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

```
//upload page
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
//basename:eturns the filename from a path
//$_FILES["filename",["name"]],get the file  uploaded by post, we could use
filename to get the file like this,
//we can also use type or size ect to get the file too, like when we use the
type:$_FILES["filetype",["type"]]
$uploadOk = 1;
$imageFileType = strtolower(pathinfo($target_file,PATHINFO_EXTENSION));
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
//The isset function in PHP is used to determine whether a variable is set or not
  $check = getimagesize($_FILES["fileToUpload"]["tmp_name"]);
  if($check !== false) {
    echo "File is an image - " . $check["mime"] . ".";
    $uploadOk = 1;
  } else {
    echo "File is not an image.";
    $uploadOk = 0;
  }
}

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

// Check file size
if ($_FILES["fileToUpload"]["size"] > 500000) {
  echo "Sorry, your file is too large.";
  $uploadOk = 0;
}

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

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

Create Cookies With PHP

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.

setcookie(name, value, expire, path("'/'means"), domain, secure, httponly):

All the parameters above except for the name, the rest are optional.We could use the function isset() to find out if the cookie is set. Please note that set cookie should appear BEFORE the <html> tag. And here's an example for cookie use:

```
<!DOCTYPE html>
<?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];
     $_COOKIE[$cookie_name] ="Rongxin";
}
?>

<p><strong>Note:</strong> You might have to reload the page to see the value of
the cookie.</p>
```

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

Note: The value of the cookie is automatically URLencoded when sending the cookie, and automatically decoded when received (to prevent URLencoding, use setrawcookie() instead).
To modify a cookie, just set (again) the cookie using the setcookie() function

Delete a Cookie

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

```php
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

Check if Cookies are Enabled

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

```php
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
  echo "Cookies are enabled.";
} else {
  echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Sesssions

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

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

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.

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:

Note: The session_start() function must be the very first thing in your document. Before any HTML tags.

We could use the session_start() function to start a new session and session_unset() and session_destroy() functions to destroy the session, and here's an example for that:

```
//page1
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>
<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
</html>

//get the session variables
//page 2
<?php
session_start();
?>
<!DOCTYPE html>
<html>
```

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

//edit the session variables
//page3
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// to change a session variable, just overwrite it
$_SESSION["favcolor"] = "yellow";
print_r($_SESSION);
?>

</body>
</html>



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

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

// destroy the session
session_destroy();
?>

</body>
</html>
```

PHP Filter

PHP filter_var() Function The filter_var() function both validate and sanitize data.

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

The variable you want to check

The type of check to use

Here's an example:

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

if (!filter_var($ip, FILTER_VALIDATE_IP) === false) {
  echo("$ip is a valid IP address");
} else {
  echo("$ip is not a valid IP address");
}
?>
//Of course, you could validate like integer, and string etc...
```

PHP callback

In php, we could use a callback like this by name:

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

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

or we could use the annoymous function like javascript:

```
//example1:
<?php
$strings = ["apple", "orange", "banana", "coconut"];
$lengths = array_map( function($item) { return strlen($item); } , $strings);
print_r($lengths);
?>
//example2:
<?php
function exclaim($str) {
  return $str . "! ";
}
```

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

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

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

What is JSON?

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

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

PHP and JSON

PHP has some built-in functions to handle JSON.

First, we will look at the following two functions:

json_encode() json_decode()

PHP - json_encode():

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

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

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

PHP - json_decode():

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

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

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

Both json_decode() and json_encode has two arguments, and the second argument is default set as false, then we will get a object, when it's set to true, we will get a array

Exception

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

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

Lets try to throw an exception without catching it:

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

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

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

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

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

And also we could use try...catch...finally Statement:

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

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

And we could create an exception object like Java:

```
new Exception(message, code, previous)
```

Parameter Values

Parameter	Description
message	Optional. A string describing why the exception was thrown
code	Optional. An integer that can be used used to easily distinguish this exception from others of the same type
previous	Optional. If this exception was thrown in a catch block of another exception, it is recommended to pass that exception into this parameter

Methods

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

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

Here's an example:

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

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

PHP What is OOP?

OOP stands for Object-Oriented Programming.

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

Object-oriented programming has several advantages over procedural programming:

OOP is faster and easier to execute OOP provides a clear structure for the programs OOP helps to keep the PHP code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug OOP makes it possible to create full reusable applications with less code and shorter development time Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

PHP OOP - Classes and Objects

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

Example:

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}
?>
```

And we use "new" keyword to define an object:

```
<?php
$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');
//get the properties
echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

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

Look at the following example:

```
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
echo $apple->name;
?>
```

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

```
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

PHP - The __construct Function

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

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

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

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

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

In php, we do not only get the constructor, we still have a destructor function which will be called as soon as a object is destructed or script is stopped or exit.If you create a __destruct() function, PHP will automatically call this function at the end of the script.

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

Example:

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}

$apple = new Fruit("Apple");
?>
```

But here's one difference to Java: in java we could use method overriding to create so many differnt constructor with different arguments, but in php we could only create only one constructor and also the destructor.

Access Modifiers

In php, we got three different access modifiers:public, private, and protected. The public is the default access modifier, and when we create an object, we could not access the properties with modifier private and protected.

Inheritance

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

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

An inherited class is defined by using the extends keyword.

Let's look at an example:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
```

So as we said the protected properties or methods can be accessed within the class and by classes derived from that class. What does that mean?

Let's look at an example:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}

// Try to call all three methods from outside class
$strawberry = new Strawberry("Strawberry", "red");  // OK. __construct() is public
$strawberry->message(); // OK. message() is public
$strawberry->intro(); // ERROR. intro() is protected
?>
```

And when we call it inside the class:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  protected function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
    // Call protected method from within derived class - OK
    $this -> intro();
  }
}

$strawberry = new Strawberry("Strawberry", "red"); // OK. __construct() is public
$strawberry->message(); // OK. message() is public and it calls intro() (which is
protected) from within the derived class
?>
```

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

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

```
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

class Strawberry extends Fruit {
  public $weight;
  public function __construct($name, $color, $weight) {
    $this->name = $name;
    $this->color = $color;
    $this->weight = $weight;
  }
  public function intro() {
    echo "The fruit is {$this->name}, the color is {$this->color}, and the weight
is {$this->weight} gram.";
  }
}

$strawberry = new Strawberry("Strawberry", "red", 50);
$strawberry->intro();
?>
```

PHP - The final Keyword

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

The following example shows how to prevent class inheritance:

```
<?php
final class Fruit {
  // some code
}
```

```
// will result in error
class Strawberry extends Fruit {
  // some code
}
?>
```

PHP - Class Constants

Constants cannot be changed once it is declared.

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

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

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

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

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

Or we could use self keyword to invoke the scope resolution operator:

```
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
  public function byebye() {
    echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```

Abstract Classes

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

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

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

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

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

- The child class method must be defined with the same name and it redeclares the parent abstract method
- The child class method must be defined with the same or a less restricted access modifier
- The number of required arguments must be the same. However, the child class may have optional
arguments in addition
Let's look at an example:

```php
<?php
// Parent class
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
  public function intro() : string {
    return "Choose German quality! I'm an $this->name!";
  }
}

class Volvo extends Car {
  public function intro() : string {
    return "Proud to be Swedish! I'm a $this->name!";
  }
}

class Citroen extends Car {
  public function intro() : string {
    return "French extravagance! I'm a $this->name!";
  }
}

// Create objects from the child classes
$audi = new audi("Audi");
echo $audi->intro();
echo "<br>";
```

```
$volvo = new volvo("Volvo");
echo $volvo->intro();
echo "<br>";

$citroen = new citroen("Citroen");
echo $citroen->intro();
?>
```

As we see in the above example, there has one statement:

```
abstract public function intro() : string;
```

And all the methods below will also add :" : string", so what does it do? In fact, it's a limition for return type, with ":string", our return type can only be a string

Interfaces

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

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

Interfaces are declared with the interface keyword:

```
<?php
// Interface definition
interface Animal {
  public function makeSound();
}

// Class definitions
class Cat implements Animal {
  public function makeSound() {
    echo " Meow ";
  }
}

class Dog implements Animal {
  public function makeSound() {
    echo " Bark ";
  }
}

class Mouse implements Animal {
  public function makeSound() {
    echo " Squeak ";
  }
}
```

```
// Create a list of animals
$cat = new Cat();
$dog = new Dog();
$mouse = new Mouse();
$animals = array($cat, $dog, $mouse);

// Tell the animals to make a sound
foreach($animals as $animal) {
  $animal->makeSound();
}
?>
```

So what;s the differences between the abstract class and the interface:

- Interfaces cannot have properties, while abstract classes can
- All interface methods must be public, while abstract class methods is public or protected
- All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary
- Classes can implement an interface while inheriting from another class at the same time
- A class can implement so many different interfaces with comas, but can noly extends one abstract class. As we learned, php is noly supporting the single inheritance. So, what if a class needs to inherit multiple behaviors? OOP traits solve this problem.

Traits are used to declare methods that can be used in multiple classes. Traits can have methods and abstract methods that can be used in multiple classes, and the methods can have any access modifier (public, private, or protected).

Traits are declared with the trait keyword:

```
<?php
trait message1 {
public function msg1() {
    echo "OOP is fun! ";
  }
}

class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
?>
```

Note:We could not set variable in trait, but we could pass the arguments into the method

Static Methods

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

Static methods are declared with the static keyword:

```
<?php
class ClassName {
  public static function staticMethod() {
    echo "Hello World!";
  }
}
ClassName::staticMethod();
?>
```

More on Static Methods

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

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }

  public function __construct() {
    self::welcome();
  }
}

new greeting();
?>
```

Static methods can also be called from methods in other classes. To do this, the static method should be public:

```
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

class SomeOtherClass {
  public function message() {
    greeting::welcome();
  }
}
?>
```

To call a static method from a child class, use the parent keyword inside the child class. Here, the static method can be public or protected:

```
<?php
class domain {
  protected static function getWebsiteName() {
    return "W3Schools.com";
  }
}

class domainW3 extends domain {
  public $websiteName;
  public function __construct() {
    $this->websiteName = parent::getWebsiteName();
  }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

Static Properties

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

Static properties are declared with the static keyword:

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

```
<?php
class pi {
  public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

And also like the static method above, we could use self::$value to invoke the property in itself, and parent::$value to invoke the property in child classes

PHP Namespaces

Namespaces are qualifiers that solve two different problems:

- They allow for better organization by grouping classes that work together to perform a task
- They allow the same name to be used for more than one class

For example, you may have a set of classes which describe an HTML table, such as Table, Row and Cell while also having another set of classes to describe furniture, such as Table, Chair and Bed. Namespaces can be used to organize the classes into two different groups while also preventing the two classes Table and Table from being mixed up.

Namespaces are declared at the beginning of a file using the namespace keyword:

```
namespace Html;
```

Note: A namespace declaration must be the first thing in the PHP file. The following code would be invalid:

```
<?php
//namespace Html;:correct namespace
echo "Hello World!";
namespace Html;//Wrong
...
?>
```

Constants, classes and functions declared in this file will belong to the Html namespace:

Example:

```
<?php
namespace Html;
class Table {
  public $title = "";
  public $numRows = 0;
  public function message() {
    echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
  }
}
$table = new Table();
$table->title = "My table";
$table->numRows = 5;
?>

<!DOCTYPE html>
<html>
<body>

<?php
$table->message();
?>

</body>
</html>
```

We can invoke the the class with namespace like this:

```
namespace Code\Html;
```

To invoke the class above:

```
$table = new Html\Table()
```

Or if you want invoke two objects with same namespace, you could do like this:

```
namespace Html;
$table = new Table();
$row = new Row();
```

For easier access we could give the namespace a alias:

```
use Html as H;
$table = new H\Table();
```

PHP Iterables

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

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

```
<?php
function getIterable():iterable {
  return ["a", "b", "c"];
  //return "a,b,c";->Fatal error
}

$myIterable = getIterable();
foreach($myIterable as $item) {
  echo $item;
}
```

Creating Iterables

Arrays

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

Iterators

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

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

An iterator must have these methods:

- current() - Returns the element that the pointer is currently pointing to. It can be any data type key() Returns the key associated with the current element in the list. It can only be an integer, float, boolean or string
- next()- Moves the pointer to the next element in the list
- rewind()- Moves the pointer to the first element in the list
- valid()- If the internal pointer is not pointing to any element (for example, if next() was called at the end of the list), this should return false. It returns true in any other case

```php
<?php
// Create an Iterator
class MyIterator implements Iterator {
  private $items = [];
  private $pointer = 0;

  public function __construct($items) {
    // array_values() makes sure that the keys are numbers
    $this->items = array_values($items);
  }

  public function current() {
    return $this->items[$this->pointer];
  }

  public function key() {
    return $this->pointer;
  }

  public function next() {
    $this->pointer++;
  }

  public function rewind() {
    $this->pointer = 0;
  }

  public function valid() {
    // count() indicates how many items are in the list
    return $this->pointer < count($this->items);
  }
}
```

```
// A function that uses iterables
function printIterable(iterable $myIterable) {
  foreach($myIterable as $item) {
    echo $item;
  }
}

// Use the iterator as an iterable
$iterator = new MyIterator(["a", "b", "c"]);
printIterable($iterator);
?>
```

PHP MySQL Database

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

PHP Connect to MySQL

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. However, this extension was deprecated in 2012.

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

```
//Example (MySQLi Object-Oriented)
<?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);
  //In php, die means exit
}
echo "Connected successfully";
?>
```

Close Database

```
$conn->close();
```

```
//Example (MySQLi Procedural)
<?php
$servername = "localhost";
$username = "username";
$password = "password";

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

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

Close Database

```
mysqli_close($conn);
```

```
<?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();
}
?>
```

Close Database

```
$conn = null;
```

Create a MySQL Database

Create a MySQL Database Using MySQLi and PDO

The CREATE DATABASE statement is used to create a database in MySQL.

The following examples create a database named "myDB":

```
//Example (MySQLi Object-oriented)
<?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);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . $conn->error;
}
```

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

Note: When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

Tip: If you have to use a specific port, add an empty string for the database-name argument, like this: new mysqli("localhost", "username", "password", "", port)

```
//Example (MySQLi Procedural)
<?php
$servername = "localhost";
$username = "username";
$password = "password";

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

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Note: The following PDO example create a database named "myDBPDO":

```
//Example (PDO)
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new PDO("mysql:host=$servername", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "CREATE DATABASE myDBPDO";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Database created successfully<br>";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Create Table

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.

The following examples shows how to create the table in PHP:

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->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 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";
```

```
if ($conn->query($sql) === TRUE) {
  echo "Table MyGuests created successfully";
} else {
  echo "Error creating table: " . $conn->error;
}

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

```
//Example (MySQLi Procedural)
<?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 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

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

mysqli_close($conn);
?>
```

```
//Example (PDO)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
```

```
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // 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 DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  )";

  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Table MyGuests created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

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 with default update of current_timesamp (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

The following examples add a new record to the "MyGuests" table:

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

```
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

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

if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

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

```
//Example (MySQLi Procedural)
<?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);
?>
```

```
//Example (PDO)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  echo "New record created successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Get Last Inserted ID("insert_id" property)

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

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

if ($conn->query($sql) === TRUE) {
  $last_id = $conn->insert_id; //We use this property to get the last id we
inserted
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
```

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

```
//Example (MySQLi Procedural)
<?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)) {
  $last_id = mysqli_insert_id($conn);
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} else {
  echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}

mysqli_close($conn);
?>
```

```
//Example (PDO)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $sql = "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";
  // use exec() because no results are returned
  $conn->exec($sql);
  $last_id = $conn->lastInsertId();
  echo "New record created successfully. Last inserted ID is: " . $last_id;
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Insert Multiple Records

If we want to insert multiple records into the database, we should use the method "mysqli_multi-query()" instead of "query()":

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

if ($conn->multi_query($sql) === TRUE) {
  echo "New records created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}

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

```
//Example (MySQLi Procedural)
<?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');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Mary', 'Moe', 'mary@example.com');";
$sql .= "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('Julie', 'Dooley', 'julie@example.com')";

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

mysqli_close($conn);
?>
```

```
//Example (PDO)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // begin the transaction
  $conn->beginTransaction();
  // our SQL statements
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')");
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Mary', 'Moe', 'mary@example.com')");
  $conn->exec("INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Julie', 'Dooley', 'julie@example.com')");

  // commit the transaction
  $conn->commit();
  echo "New records created successfully";
} catch(PDOException $e) {
  // roll back the transaction if something failed
  $conn->rollback();
  echo "Error: " . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Prepared Statements

A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency.

Prepared statements basically work like this:

Prepare: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?) The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it Execute: At a later time, the application binds the values to the parameters, and the database executes the statement. The application may execute the statement as many times as it wants with different values Compared to executing SQL statements directly, prepared statements have three main advantages:

Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times) Bound parameters minimize bandwidth to the server as you need send only the parameters each time, and not the whole query Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

```php
<?php
//Example (MySQLi with Prepared Statements)
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

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

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

// prepare and bind
$stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email) VALUES
(?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "John";
$lastname = "Doe";
$email = "john@example.com";
```

```
$stmt->execute();

$firstname = "Mary";
$lastname = "Moe";
$email = "mary@example.com";
$stmt->execute();

$firstname = "Julie";
$lastname = "Dooley";
$email = "julie@example.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

Note:The argument may be one of four types:

- **i - integer**
- **d - double**
- **s - string**
- **b - BLOB**

So we could get the conclusion:

```
bind_param("type1,type2...",$value1,$value2...)
```

We must have one of these for each parameter.

By telling mysql what type of data to expect, we minimize the risk of SQL injections.

So Note: If we want to insert any data from external sources (like user input), it is very important that the data is sanitized and validated.

```
//Example (PDO with Prepared Statements)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // prepare sql and bind parameters
```

```
    $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)
    VALUES (:firstname, :lastname, :email)");
    $stmt->bindParam(':firstname', $firstname);
    $stmt->bindParam(':lastname', $lastname);
    $stmt->bindParam(':email', $email);

    // insert a row
    $firstname = "John";
    $lastname = "Doe";
    $email = "john@example.com";
    $stmt->execute();

    // insert another row
    $firstname = "Mary";
    $lastname = "Moe";
    $email = "mary@example.com";
    $stmt->execute();

    // insert another row
    $firstname = "Julie";
    $lastname = "Dooley";
    $email = "julie@example.com";
    $stmt->execute();

    echo "New records created successfully";
  } catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
  }
  $conn = null;
  ?>
```

PHP MySQL Select Data

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

```
SELECT column_name(s) FROM table_name
```

Each column name seperated with comas

If you want select all the columns from the table:

```
SELECT * FROM table_name
```

Example:

```
//Example (MySQLi Object-oriented)
<?php
```

```
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

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

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

Code lines to explain from the example above:

First, we set up an SQL query that selects the id, firstname and lastname columns from the MyGuests table. The next line of code runs the query and puts the resulting data into a variable called $result.

Then, the function num_rows() checks if there are more than zero rows returned.

If there are more than zero rows returned, the function fetch_assoc() puts all the results into an associative array that we can loop through. The while() loop loops through the result set and outputs the data from the id, firstname and lastname columns.

The following example shows the same as the example above, in the MySQLi procedural way:

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

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

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

And also you could use the tags to display them on HTML:

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

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

if ($result->num_rows > 0) {
  echo "<table><tr><th>ID</th><th>Name</th></tr>";
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]."
".$row["lastname"]."</td></tr>";
  }
  echo "</table>";
} else {
  echo "0 results";
}
$conn->close();
?>
```

```
//Example (PDO)
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
  function __construct($it) {
    parent::__construct($it, self::LEAVES_ONLY);
  }

  function current() {
    return "<td style='width:150px;border:1px solid black;'>" . parent::current().
"</td>";
  }

  function beginChildren() {
    echo "<tr>";
  }

  function endChildren() {
    echo "</tr>" . "\n";
  }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests");
  $stmt->execute();

  // set the resulting array to associative
  $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
  foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v)
{
    echo $v;
  }
} catch(PDOException $e) {
  echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

PHP MySQL Use The WHERE Clause

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified condition.

```
SELECT column_name(s) FROM table_name WHERE column_name operator value
```

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
$result = $conn->query($sql);

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

```
//Example (MySQLi Procedural)
<?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 WHERE lastname='Doe'";
$result = mysqli_query($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);
?>
```

```
//Example (MySQLi Object-oriented)

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

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname='Doe'";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  echo "<table><tr><th>ID</th><th>Name</th></tr>";
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]."
".$row["lastname"]."</td></tr>";
  }
  echo "</table>";
} else {
  echo "0 results";
}
$conn->close();
?>
```

Select Data With PDO (+ Prepared Statements):

```
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
  function __construct($it) {
    parent::__construct($it, self::LEAVES_ONLY);
  }

  function current() {
    return "<td style='width:150px;border:1px solid black;'>" . parent::current().
"</td>";
  }

  function beginChildren() {
    echo "<tr>";
  }

  function endChildren() {
    echo "</tr>" . "\n";
  }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests WHERE
lastname='Doe'");
  $stmt->execute();

  // set the resulting array to associative
  $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
  foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v)
{
    echo $v;
  }
}
catch(PDOException $e) {
  echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

Note:The use of RecursiveIteratorIterator::__construct:

```
public RecursiveIteratorIterator::__construct(iterator, mode,flag)
```

iterator The iterator being constructed from. Either a RecursiveIterator or IteratorAggregate.

mode Optional mode. Possible values are

- **RecursiveIteratorIterator::LEAVES_ONLY - The default. Lists only leaves in iteration.**
- **RecursiveIteratorIterator::SELF_FIRST - Lists leaves and parents in iteration with parents coming first.**
- **RecursiveIteratorIterator::CHILD_FIRST - Lists leaves and parents in iteration with leaves coming first. Flags**

Optional flag. Possible values are RecursiveIteratorIterator::CATCH_GET_CHILD which will then ignore exceptions thrown in calls to RecursiveIteratorIterator::getChildren().

Example:

```
<?php
$array = array(
    array(
        array(
            array(
                'leaf-0-0-0-0',
                'leaf-0-0-0-1'
            ),
            'leaf-0-0-0'
        ),
        array(
            array(
                'leaf-0-1-0-0',
                'leaf-0-1-0-1'
            ),
            'leaf-0-1-0'
        ),
        'leaf-0-0'
    )
);

$iterator = new RecursiveIteratorIterator(
    new RecursiveArrayIterator($array),
    $mode
);
foreach ($iterator as $key => $leaf) {
    echo "$key => $leaf", PHP_EOL;
}
?>
```

When $mode= RecursiveIteratorIterator::LEAVES_ONLY

```
0 => leaf-0-0-0-0
1 => leaf-0-0-0-1
0 => leaf-0-0-0
0 => leaf-0-1-0-0
1 => leaf-0-1-0-1
0 => leaf-0-1-0
0 => leaf-0-0
```

When $mode= RecursiveIteratorIterator::SELF_FIRST

```
0 => Array
0 => Array
0 => Array
0 => leaf-0-0-0-0
1 => leaf-0-0-0-1
1 => leaf-0-0-0
1 => Array
0 => Array
0 => leaf-0-1-0-0
1 => leaf-0-1-0-1
1 => leaf-0-1-0
2 => leaf-0-0
```

When $mode= RecursiveIteratorIterator::CHILD_FIRST

```
0 => leaf-0-0-0-0
1 => leaf-0-0-0-1
0 => Array
1 => leaf-0-0-0
0 => Array
0 => leaf-0-1-0-0
1 => leaf-0-1-0-1
0 => Array
1 => leaf-0-1-0
1 => Array
2 => leaf-0-0
0 => Array
```

PHP MySQL Use The ORDER BY Clause

Select and Order Data From a MySQL Database The ORDER BY clause is used to sort the result-set in ascending or descending order.

The ORDER BY clause sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column_name(s) FROM table_name ORDER BY column_name(s) ASC|DESC
```

Example:

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = $conn->query($sql);

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

```
//Example (MySQLi Procedural)
<?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 ORDER BY lastname";
$result = mysqli_query($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);
?>
```

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

$sql = "SELECT id, firstname, lastname FROM MyGuests ORDER BY lastname";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  echo "<table><tr><th>ID</th><th>Name</th></tr>";
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "<tr><td>".$row["id"]."</td><td>".$row["firstname"]."
".$row["lastname"]."</td></tr>";
  }
  echo "</table>";
} else {
  echo "0 results";
}
$conn->close();
?>
```

Select Data With PDO (+ Prepared Statements)

```
//Example (PDO)
<?php
echo "<table style='border: solid 1px black;'>";
echo "<tr><th>Id</th><th>Firstname</th><th>Lastname</th></tr>";

class TableRows extends RecursiveIteratorIterator {
  function __construct($it) {
    parent::__construct($it, self::LEAVES_ONLY);
  }

  function current() {
    return "<td style='width:150px;border:1px solid black;'>" . parent::current().
"</td>";
  }

  function beginChildren() {
    echo "<tr>";
  }

  function endChildren() {
    echo "</tr>" . "\n";
  }
}

$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
  $stmt = $conn->prepare("SELECT id, firstname, lastname FROM MyGuests ORDER BY
lastname");
  $stmt->execute();

  // set the resulting array to associative
  $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
  foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v)
{
    echo $v;
  }
} catch(PDOException $e) {
  echo "Error: " . $e->getMessage();
}
$conn = null;
echo "</table>";
?>
```

Delete Data From a MySQL Table Using MySQLi and PDO

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

```
Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

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

if ($conn->query($sql) === TRUE) {
  echo "Record deleted successfully";
} else {
  echo "Error deleting record: " . $conn->error;
}

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

```
//Example (MySQLi Procedural)
<?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);
?>
```

```
//Example (PDO)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

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

  // use exec() because no results are returned
  $conn->exec($sql);
  echo "Record deleted successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Update Data

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

```
//Example (MySQLi Object-oriented)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
```

```
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}

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

if ($conn->query($sql) === TRUE) {
  echo "Record updated successfully";
} else {
  echo "Error updating record: " . $conn->error;
}

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

```
//Example (MySQLi Procedural)
<?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);
?>
```

```
//Example (PDO)
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDBPDO";
```

```
try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
  // set the PDO error mode to exception
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

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

  // Prepare statement
  $stmt = $conn->prepare($sql);

  // execute the query
  $stmt->execute();

  // echo a message to say the UPDATE succeeded
  echo $stmt->rowCount() . " records UPDATED successfully";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e->getMessage();
}

$conn = null;
?>
```

PHP MySQL Limit Data Selections

MySQL provides a LIMIT clause that is used to specify the number of records to return.

The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.

Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

```
$sql = "SELECT * FROM Orders LIMIT 30";
```

The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":

```
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
```

PHP XML Parsers

The XML language is a way to structure data for sharing across websites.

Several web technologies like RSS Feeds and Podcasts are written in XML.

XML is easy to create. It looks a lot like HTML, except that you make up your own tags.

If you want to learn more about XML, please visit our XML tutorial.

XML Parser:

To read and update, create and manipulate an XML document, you will need an XML parser.

In PHP there are two major types of XML parsers:

- **Tree-Based Parsers**
- **Event-Based Parsers**

Tree-Based Parsers:

Tree-based parsers holds the entire document in Memory and transforms the XML document into a Tree structure. It analyzes the whole document, and provides access to the Tree elements (DOM).

This type of parser is a better option for smaller XML documents, but not for large XML document as it causes major performance issues.

Example of tree-based parsers:

- **SimpleXML**
- **DOM**

Event-Based Parsers:

Event-based parsers do not hold the entire document in Memory, instead, they read in one node at a time and allow you to interact with in real time. Once you move onto the next node, the old one is thrown away.

This type of parser is well suited for large XML documents. It parses faster and consumes less memory.

Example of event-based parsers:

- **XMLReader**
- **XML Expat Parser**

The SimpleXML Parser:

SimpleXML is a tree-based parser.

SimpleXML provides an easy way of getting an element's name, attributes and textual content if you know the XML document's structure or layout.

SimpleXML turns an XML document into a data structure you can iterate through like a collection of arrays and objects.

Compared to DOM or the Expat parser, SimpleXML takes a fewer lines of code to read text data from an element.

The PHP simplexml_load_string() function is used to read XML data from a string.

Assume we have a variable that contains XML data, like this:

Example:

```
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";

//Invoke
<?php
$myXMLData =
"<?xml version='1.0' encoding='UTF-8'?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>";

$xml=simplexml_load_string($myXMLData) or die("Error: Cannot create object");
print_r($xml);
?>
//Then you get an associtive array like this:
```

```
SimpleXMLElement Object ( [to] => Tove [from] => Jani [heading] => Reminder [body]
=> Don't forget me this weekend! )
```

Also you could invoke a xml file directly with the method simplexml_load_file():

```
<!-- Declare -->
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

```
//Invoke
<?php
$xml=simplexml_load_file("note.xml") or die("Error: Cannot create object");
print_r($xml);
?>
```

```
```

So now we get a XML object, we could get the values from it:

```php
<?php
$xml=simplexml_load_file("note.xml") or die("Error: Cannot create object");
echo $xml->to . "<br>";
echo $xml->from . "<br>";
echo $xml->heading . "<br>";
echo $xml->body;
?>
```

And if we hace more than one object in xml, like this:

```xml
<?xml version="1.0" encoding="utf-8"?>
<bookstore>
  <book category="COOKING">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="CHILDREN">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">XQuery Kick Start</title>
    <author>James McGovern</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="WEB">
    <title lang="en-us">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

We could Get Node Values of Specific Elements like this:

```php
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
echo $xml->book[0]->title . "<br>";
echo $xml->book[1]->title;
?>
```

And xml can be considered as an array, so we could loop it with arrays' way:

```
<?php
$xml=simplexml_load_file("books.xml") or die("Error: Cannot create object");
foreach($xml->children() as $books) {
  echo $books->title . ", ";
  echo $books->author . ", ";
  echo $books->year . ", ";
  echo $books->price . "<br>";
}
?>
```

Then we will get :

```
Everyday Italian, Giada De Laurentiis, 2005, 30.00
Harry Potter, J K. Rowling, 2005, 29.99
XQuery Kick Start, James McGovern, 2003, 49.99
Learning XML, Erik T. Ray, 2003, 39.95
```

Besides, we could also get the attributes from the each object, like the example above, we found each book has an property:"category", and we could use:

```
 $xml->book[0]['category'];
```

And in every book, they all get a title and in the title they get a "lang" property, so we could get them in this way:

```
$xml->book[1]->title['lang'];
```

XML Expat Parser

```
<?php
// Initialize the XML parser
$parser=xml_parser_create();

// Function to use at the start of an element,this start function must have three
arguments: $parser->A variable containing the XML parser calling the handler.
$name->A variable containing the name of the elements, that triggers this
function, from the XML file as a string.$data->An array containing the elements
attributes from the XML file as a string
function start($parser,$element_name,$element_attrs) {
  switch($element_name) {
    case "NOTE":
    echo "-- Note --<br>";
    break;
    case "TO":
    echo "To: ";
    break;
    case "FROM":
    echo "From: ";
    break;
    case "HEADING":
    echo "Heading: ";
    break;
    case "BODY":
    echo "Message: ";
  }
}

// Function to use at the end of an element,this method should have these two
arguments:$parser->A variable containing the XML parser calling the handler.$name-
> A variable containing the name of the elements, that triggers this function,
from the XML file as a string
function stop($parser,$element_name) {
  echo "<br>";
}

// Function to use when finding character data
function char($parser,$data) {
  echo $data;
}

// Specify element handler
xml_set_element_handler($parser,"start","stop");

// Specify data handler
xml_set_character_data_handler($parser,"char");

// Open XML file
$fp=fopen("note.xml","r");

// Read data,fread() reads up to length bytes from the file pointer referenced by
stream. Reading stops as soon as one of the following conditions is met:1.length
bytes have been read,2.EOF (end of file) is reached,3.a packet becomes available
or the socket timeout occurs (for network streams),4.if the stream is read
buffered and it does not represent a plain file, at most one read of up to a
number of bytes equal to the chunk size (usually 8192) is made; depending on the
previously buffered data, the size of the returned data may be larger than the
chunk size.
while ($data=fread($fp,4096)) {
  //feof — Tests for end-of-file on a file pointer
  xml_parse($parser,$data,feof($fp)) or
  die (sprintf("XML Error: %s at line %d",
  xml_error_string(xml_get_error_code($parser)),
  xml_get_current_line_number($parser)));
```

```
}

// Free the XML parser
xml_parser_free($parser);
?>
```

Lastly, we got this conclusion:The Expat parser is an event-based parser, it's all operated on methods.In fact, we could use PHP XML DOM Parser also which is a tree-based parse. And there are three levels of parser:Level 1: XML Document,Level 2: Root element: <from>,Level 3: Text element: "Jani".Here's an example:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

```php
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

print $xmlDoc->saveXML();
?>
```

And you will get this:

```
Tove Jani Reminder Don't forget me this weekend!
```

If you want get the result like Expat, we could do like this:

```php
<?php
$xmlDoc = new DOMDocument();
$xmlDoc->load("note.xml");

$x = $xmlDoc->documentElement;
foreach ($x->childNodes AS $item) {
  print $item->nodeName . " = " . $item->nodeValue . "<br>";
}
?>
```

Result:

```
#text =
to = Tove
#text =
from = Jani
#text =
heading = Reminder
#text =
body = Don't forget me this weekend!
#text =
```

And we found: in the example above you see that there are empty text nodes between each element.When XML generates, it often contains white-spaces between the nodes. The XML DOM parser treats these as ordinary elements, and if you are not aware of them, they sometimes cause problems.