PHP

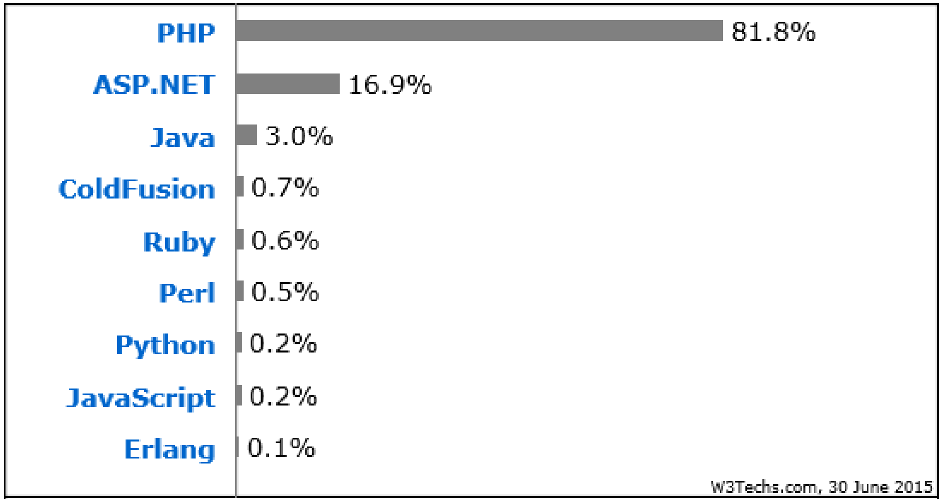
PHP Introduction

- PHP is a recursive acronym (or backronym) for “PHP: Hypertext Preprocessor”
- PHP is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML
  - As of December 2017, PHP makes up over 83% of server side languages used on the internet. Much of that is made up of PHP-based content management systems such as WordPress, but even if you remove pre-built CMS from the equation, PHP still makes up over 54% of the web.

- PHP runs on different platforms (Windows, Linux/Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

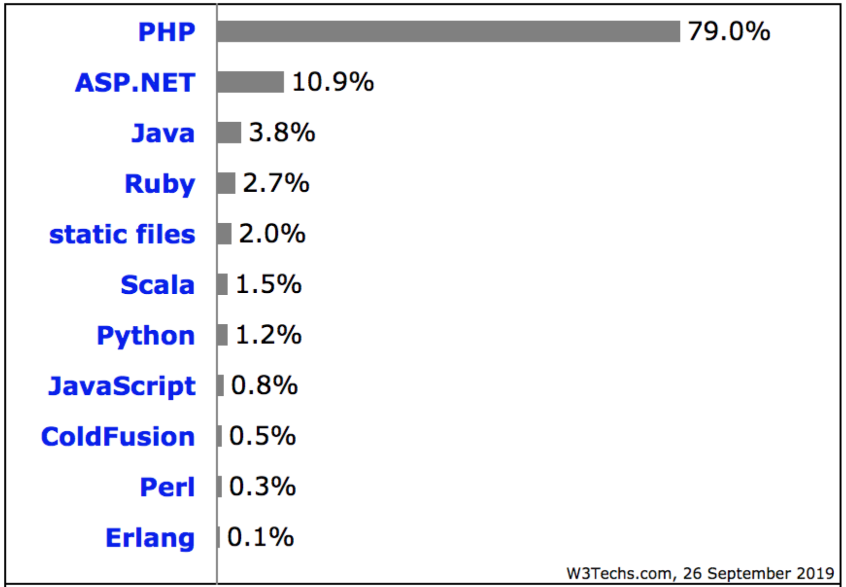
Market Share

This diagram shows the percentages of websites using various server-side programming languages. The reports are updated daily. PHP is used by 81.8% of all the websites whose server-side programming language we know. (Survey on June 29, 2015)



W3Techs.com, 30 June 2015

Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

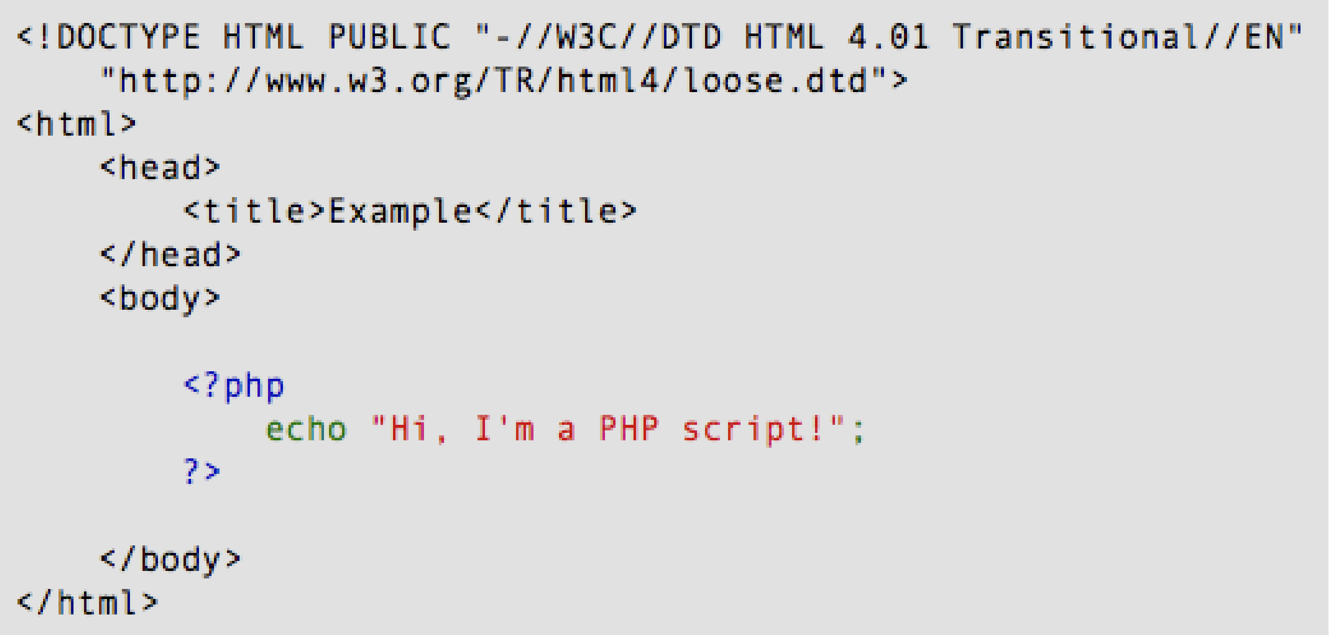


W3Techs.com, 26 September 2019

Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

PHP Introduction

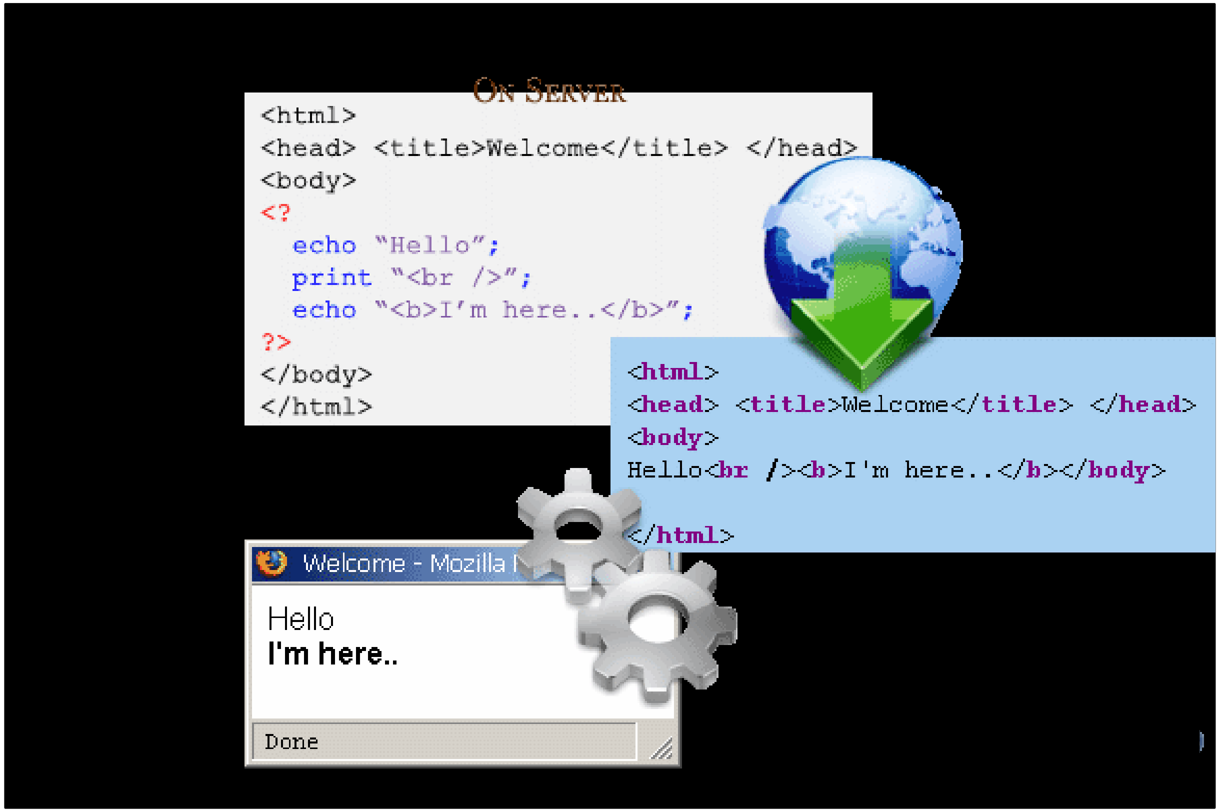
- Instead of lots of commands to output HTML (as seen in C or Perl), PHP pages contain HTML with embedded code that does "something"
- The PHP code is enclosed in special start and end processing instructions **<?php** and **?>** that allow you to jump into and out of "PHP mode."



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

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

    </body>
</html>
```

PHP Introduction

- PHP code is executed on the server, generating HTML which is then sent to the client. The client would receive the results of running that script, but would not know what the underlying code was.



```
<html>
<head> <title>Welcome</title> </head>
<body>
<?
  echo “Hello”;
  print “<br />”;
  echo “<b>I’m here..</b>”;
?>
</body>
</html>
```

```
<html>
<head> <title>Welcome</title> </head>
<body>
Hello<br /><b>I'm here..</b></body>

</html>
```

Hello
I'm here..

PHP Introduction

- PHP files:
  - contain text, HTML, CSS, JavaScript, and PHP code
  - are executed on the server, and the result is returned to the browser as plain HTML
  - have extension ".php“
- PHP can:
  - generate dynamic page content
  - create, open, read, write, delete, and close files on the server
  - collect form data
  - send and receive cookies
  - add, delete, modify data in your database
  - restrict users to access some pages on your website
  - encrypt data
  - also output images, PDF files, and even Flash movies
  - also output any text, such as XHTML and XML

PHP Getting Started

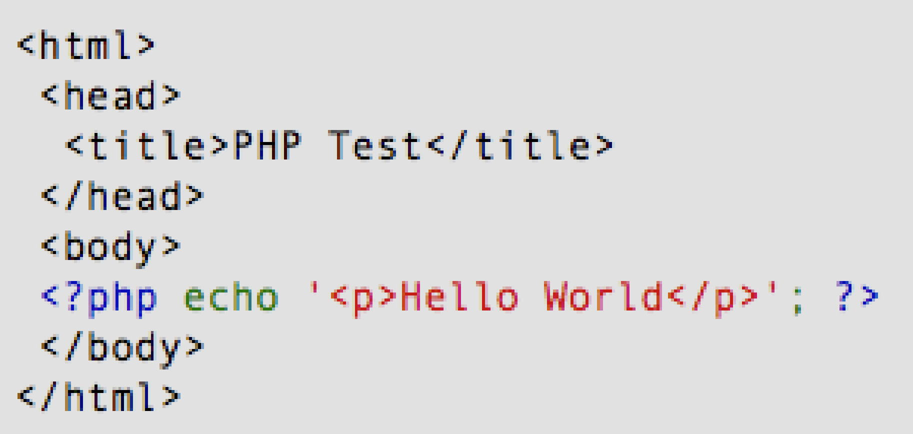
- For both Windows and Mac, you can download and install MAMP
  - http://www.mamp.info/en/index.html

- Note: you might have to install additional support for PHP:
  - MySQL/PostgreSQL
  - XML
  - SOAP

PHP Hello World

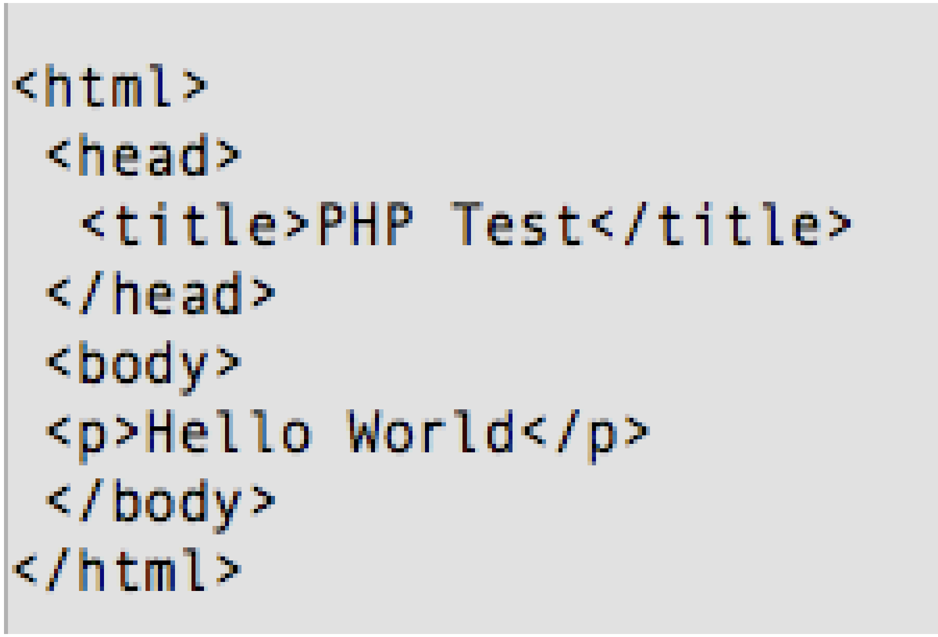
- Sample program to display “Hello World” using the PHP **echo()** statement.

HelloWorld.php on Server



```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <?php echo '<p>Hello World</p>'; ?>
 </body>
</html>
```

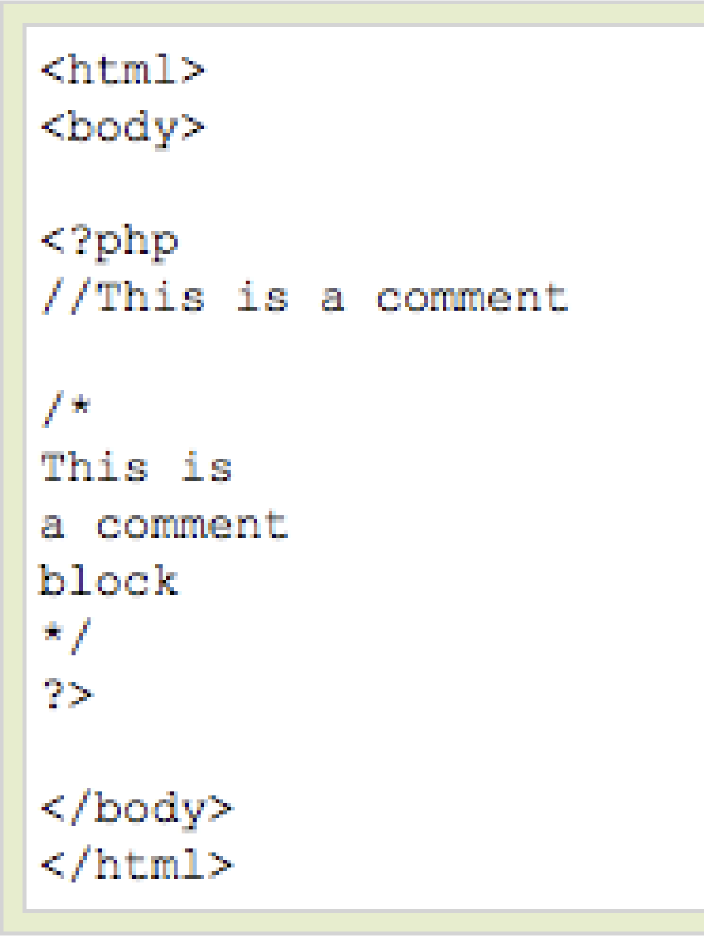
HTML returned to client (e.g. web browser) from HelloWorld.php on Server



```
<html>
 <head>
  <title>PHP Test</title>
 </head>
 <body>
 <p>Hello World</p>
 </body>
</html>
```

PHP Comments

- Use **//** to make a single-line comment or **/*** and ***/** to make a large comment block



```
<html>
<body>

<?php
//This is a comment

/*
This is
a comment
block
*/
?>

</body>
</html>
```

PHP Variables

- All variables in PHP start with a $ sign symbol
- Variables are used for:
  - storing values, like text strings, numbers or arrays (e.g. $y=5)
  - hold expressions ($z=$x+$y)
- A variable name:
  - must start with a letter or the underscore character
  - cannot start with a number
  - can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
  - is case sensitive i.e. $lower and $LOWER are two different variables
  - should not contain spaces
    - a variable name that is more than one word, should be separated with an underscore ($my_string) or with capitalization ($myString)
- A variable is created when a value is assigned to it

PHP Variables

- In PHP, a variable does not need to be declared before adding a value to it
- PHP automatically converts the variable to the correct data type, depending on its value.
- Examples:
  - $txt=“Hello World!”;
  - $x=16;
  - $y=10.5;

PHP Variables

- Variables can be declared anywhere in the PHP script
- The scope of a variable is the part of the script where the variable can be referenced/used
- PHP has three different variable scopes:
  - Local
  - Global
  - Static

- http://php.net/manual/en/language.variables.scope.php

PHP Variables

- Local
  - variable declared within a function has a LOCAL SCOPE and can only be accessed within that function
- Global
  - variable declared outside a function has a GLOBAL SCOPE and can only be accessed outside a function

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

function myTest()
{
    $y=10; // local scope
    echo "<p>Test variables inside the function:<p>";
    echo "Variable x is: $x";
    echo "<br>";
    echo "Variable y is: $y";
}

myTest();
echo "<p>Test variables outside the function:<p>";
echo "Variable x is: $x";
echo "<br>";
echo "Variable y is: $y";
?>
```

PHP Variables

- Global (continued)
  - Exception:
    - *global* keyword is used to access a global variable from within a function
    - use the global keyword before the variables (inside the function)
    - PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

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

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

myTest();
echo $y; // outputs 15
?>
```

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

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

 myTest();
 echo $y; // outputs 15
?>
```

PHP Variables

- Static
  - a variable that has been allocated statically—whose lifetime or "extent" extends across the entire run of the program

```
<?php

function myTest()
{
        static $x=0;
        echo $x;
        $x++;
}

myTest();
myTest();
myTest();

?>
```

PHP Output Statements

- The two PHP output statements are:
  - echo
    - output one or more strings

```
<?php
$txt1=“Echo this text";
$cars=array("Volvo","BMW","Toyota");

echo "<h2>PHP is fun!</h2>";
echo "<br>";
echo "This", " string", " was", " made", " with multiple strings.";
echo $txt1;
echo “Again $txt2";
echo "My car is a {$cars[0]}";
?>
```

PHP Output Statements

- print
    - can only output one string, and returns always 1

```
<?php
$txt1=“Print this text";
$cars=array("Volvo","BMW","Toyota");

print "<h2>PHP is fun!</h2>";
print "<br>";
print $txt1;
print “Again $txt2";
print "My car is a {$cars[0]}";
?>
```

- Note: output strings for both echo() and print() can contain HTML markup

PHP Data Types

- String
  - a sequence of characters inside single or double quotes
    - $x = "Hello world!";
    - $x = 'Hello world!';
- Integer
  - three formats:
    - Decimal
      - $x = 5985;
      - $x = -345;
    - Hexadecimal (prefixed with 0x)
      - $x = 0x8C;
    - octal (prefixed with 0)
      - $x = 047;

PHP Data Types

- Floating Point Number
  - number with a decimal point or in exponential form
    - $x = 10.365;
    - $x = 2.4e3;
    - $x = 8E-5;
- Boolean
  - True or false
    - $x=true;
    - $y=false;
- Array
  - stores multiple values in one single variable
    - $cars=array("Volvo","BMW","Toyota");

PHP Data Types

- Object
  - stores data and information on how to process that data
  - use *class* keyword to declare a structure that can contain properties and methods
  - create instances/objects of that class

```
<?php
class Car
{
    var $color;
    function Car($color="green")
    {
      $this->color = $color;
    }
    function what_color()
    {
      return $this->color;
    }
}
?>
```

PHP Data Types

- NULL value
  - NULL is the only possible value of data type NULL
  - NULL value represents that a variable has no value.
  - Variables can be emptied by setting the value to NULL
  - Example: $x=null;

- The PHP var_dump() function returns the data type and value of variables:

```
<?php
$a = array(1, 2, array("a", "b", "c"));
var_dump($a);
?>
```

The above example will output:

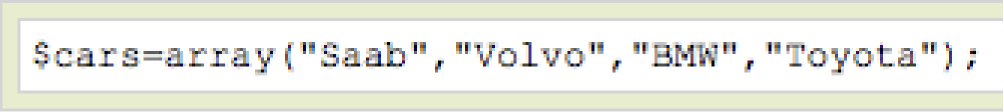
```
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(2)
  [2]=>
  array(3) {
    [0]=>
    string(1) "a"
    [1]=>
    string(1) "b"
    [2]=>
    string(1) "c"
  }
}
```

PHP Arrays

- In PHP, there are three kind of arrays:
  - **Numeric array** - An array with a numeric index
  - **Associative array** - An array where each ID key is associated with a value
  - **Multidimensional array** - An array containing one or more arrays

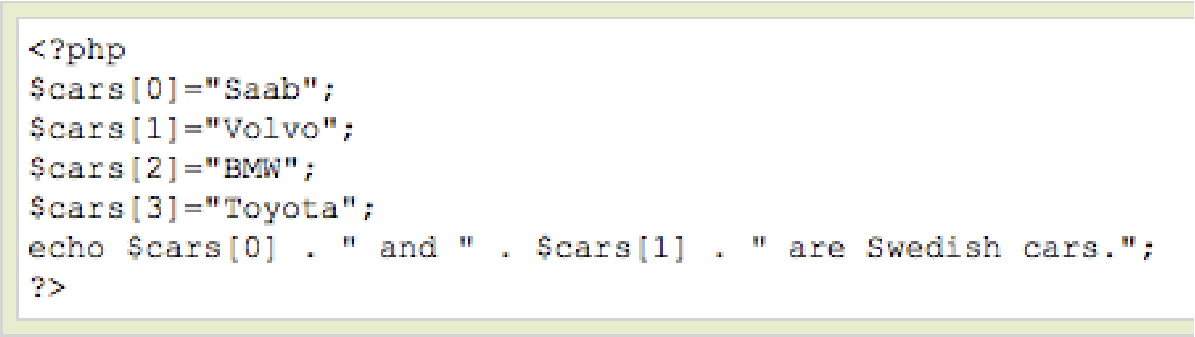
PHP Numeric Arrays

- A numeric array stores each array element with a numeric index
- There are two methods to create a numeric array
- In the following example the index is automatically assigned (the index starts at 0):



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

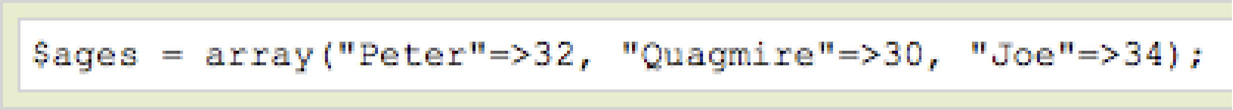
- In the following example we assign the index manually:



```
<?php
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";
?>
```

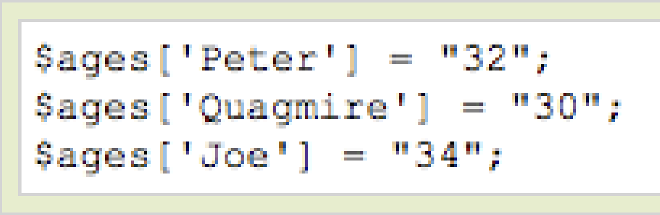
PHP Associative Arrays

- With an associative array, each ID key is associated with a value
- When storing data about specific named values, a numerical array is not always the best way to do it
- With associative arrays we can use the values as keys and assign values to them
- In this example we use an array to assign ages to the different persons:



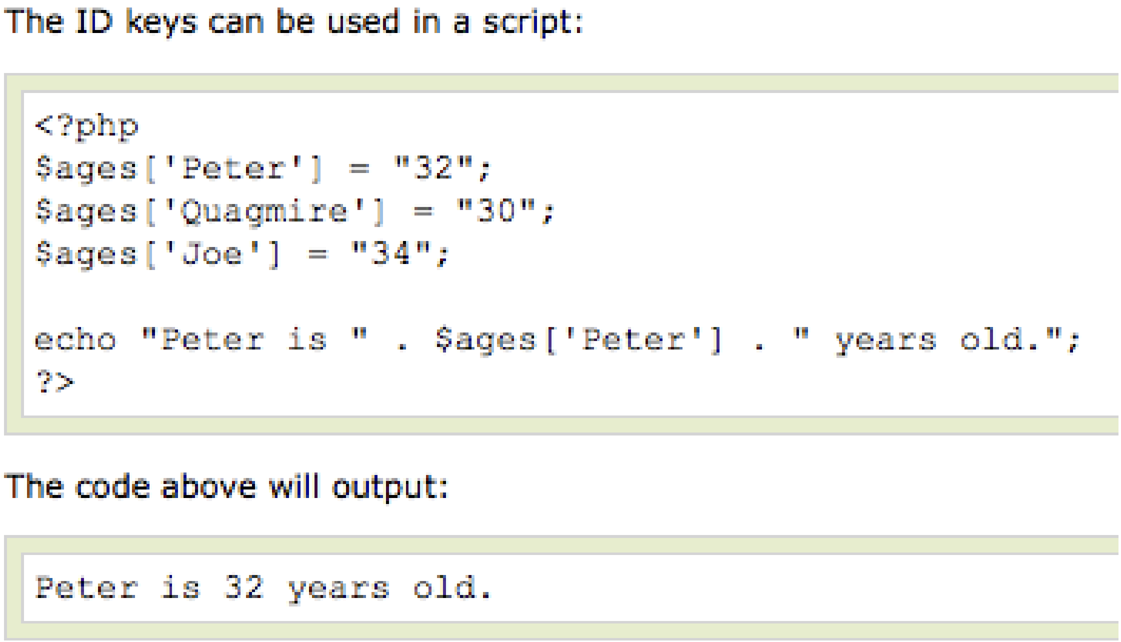
```
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
```

- This example is the same as the one above, but shows a different way of creating the array:



```
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";
```

PHP Associative Arrays



The ID keys can be used in a script:

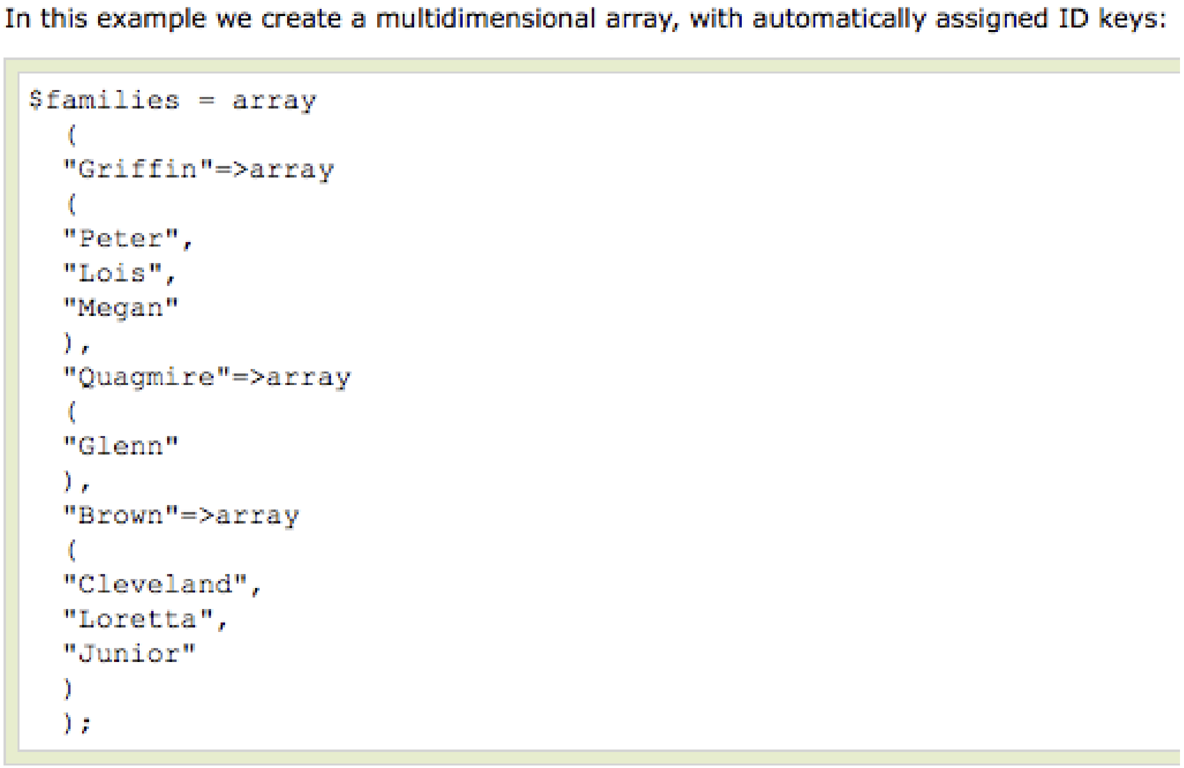
```
<?php
$ages['Peter'] = "32";
$ages['Quagmire'] = "30";
$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

PHP Multidimensional Arrays

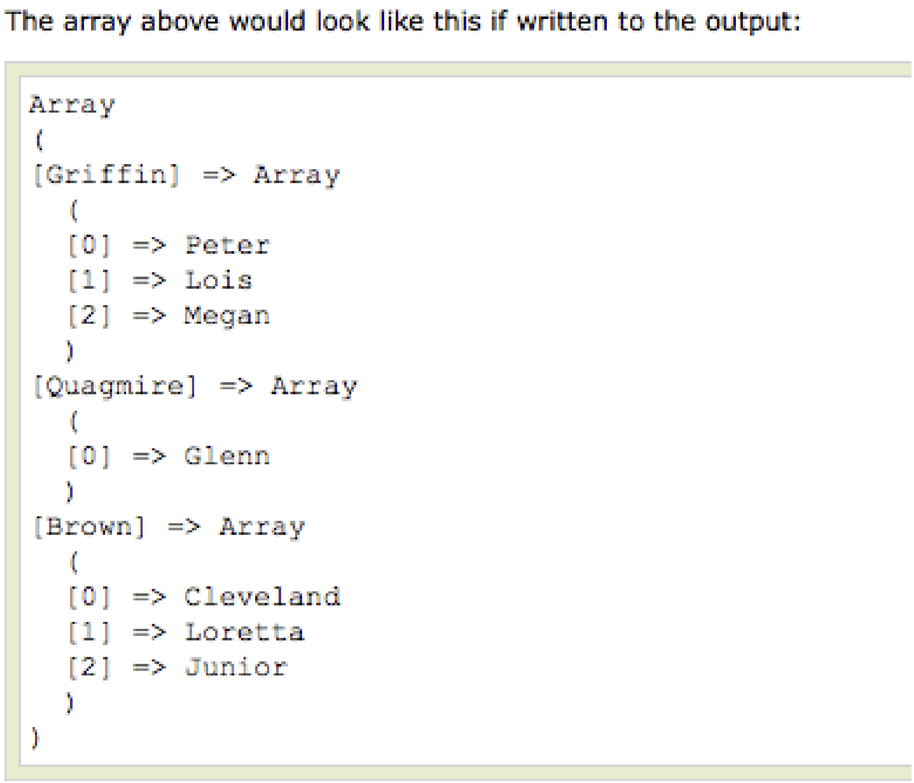
- In a multidimensional array, each element in the main array can also be an array
- And each element in the sub-array can be an array, and so on



In this example we create a multidimensional array, with automatically assigned ID keys:

```
$families = array
  (
  "Griffin"=>array
  (
  "Peter",
  "Lois",
  "Megan"
  ),
  "Quagmire"=>array
  (
  "Glenn"
  ),
  "Brown"=>array
  (
  "Cleveland",
  "Loretta",
  "Junior"
  )
  );
```

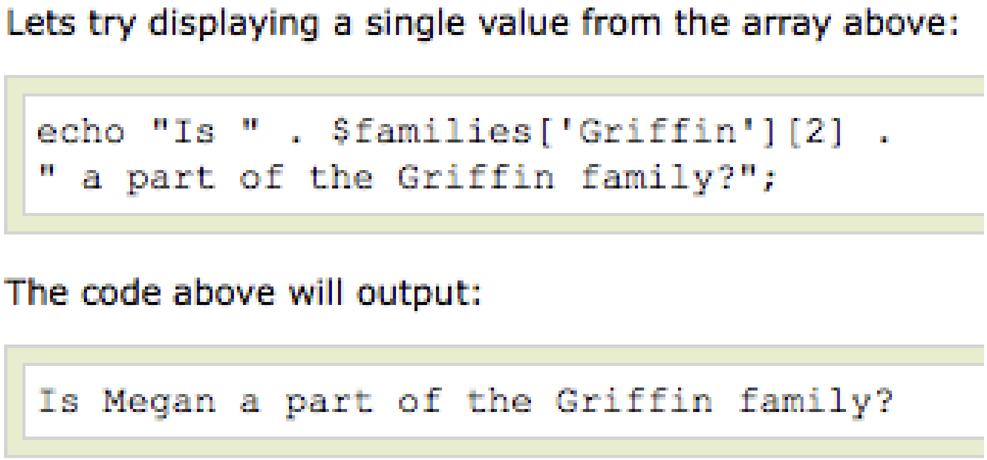
PHP Multidimensional Arrays



The array above would look like this if written to the output:

```
Array
(
[Griffin] => Array
  (
  [0] => Peter
  [1] => Lois
  [2] => Megan
  )
[Quagmire] => Array
  (
  [0] => Glenn
  )
[Brown] => Array
  (
  [0] => Cleveland
  [1] => Loretta
  [2] => Junior
  )
)
```

PHP Multidimensional Arrays



Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

PHP Multidimensional Arrays

```
<?php

    $cars = array
     (
      array("Volvo",22,18),
      array("BMW",15,13),
      array("Saab",5,2),
      array("Land Rover",17,15)
      );
    echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
    echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
    echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
    echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";


?>
```

PHP Concatenation

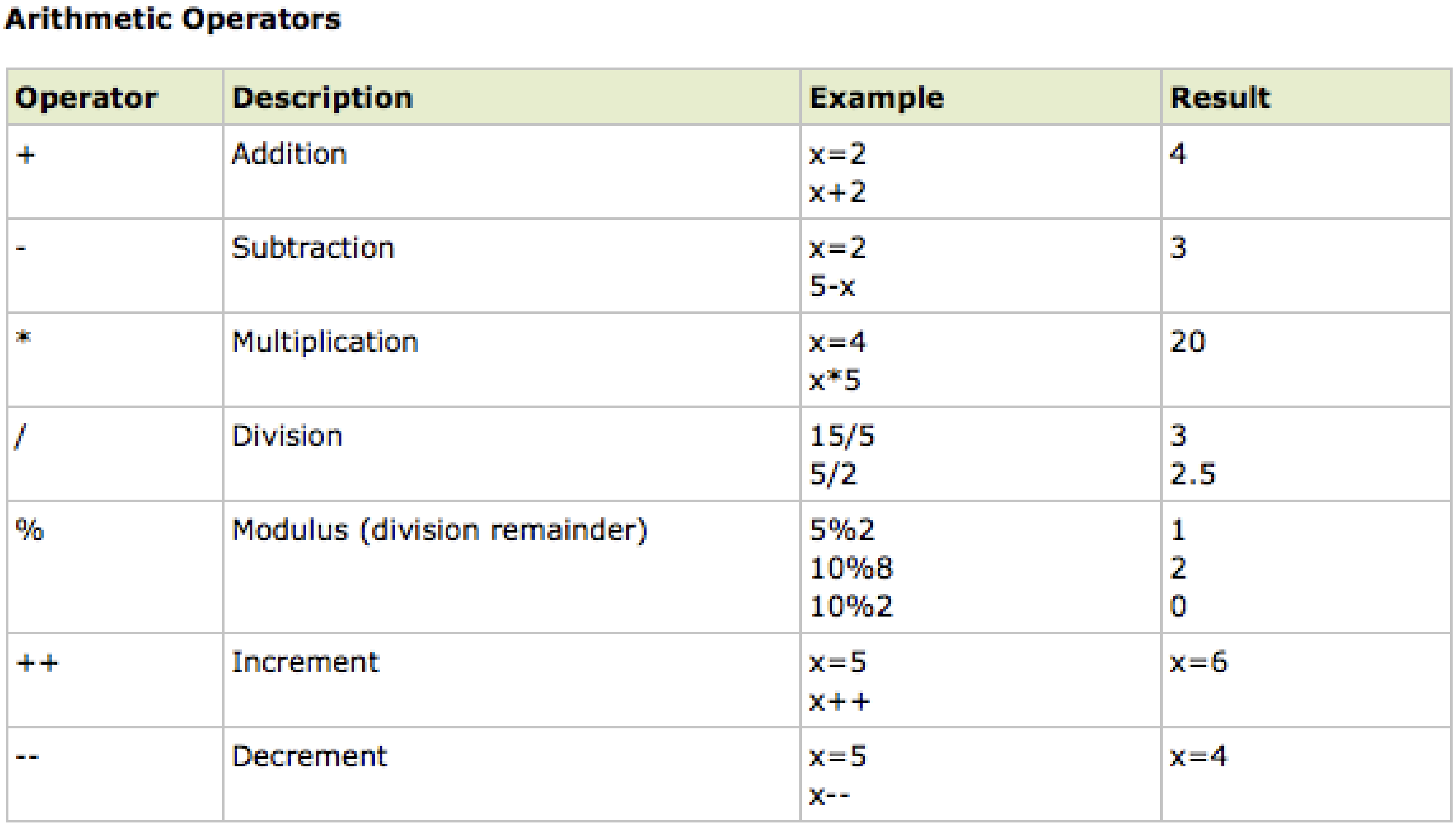
- The concatenation operator (.) is used to put two string values together
- To concatenate two string variables together, use the concatenation operator:

```
<?php
   $first_name=“John";
   $last_name=“Smith”;
   $full_name=$first_name . $last_name;
?>
```

PHP Operators

- Operators are used to operate on values. There are four classifications of operators:
  - Arithmetic
  - Assignment
  - Comparison
  - Logical

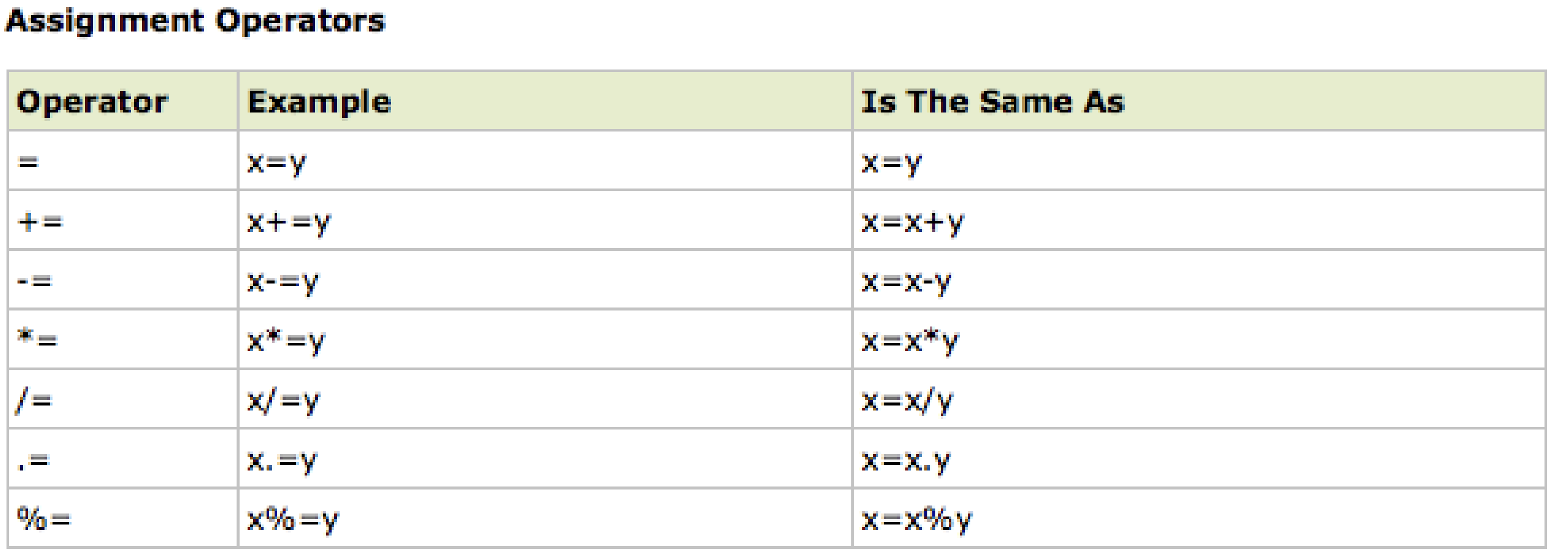
PHP Operators



Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

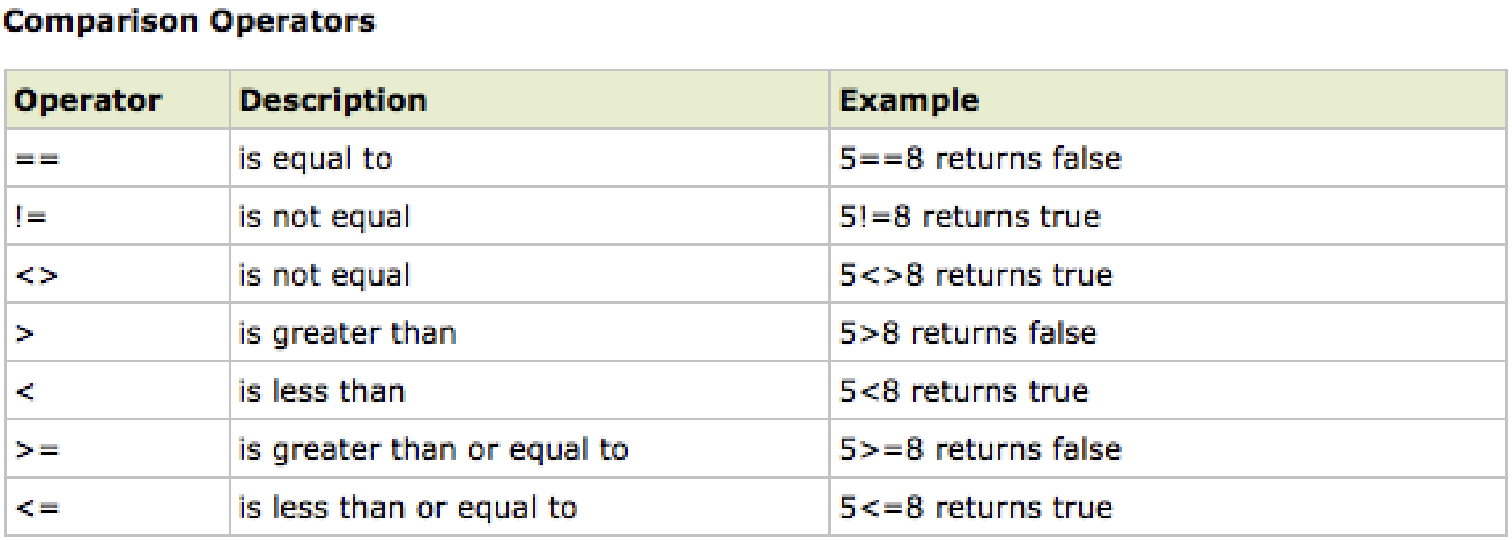
PHP Operators



Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

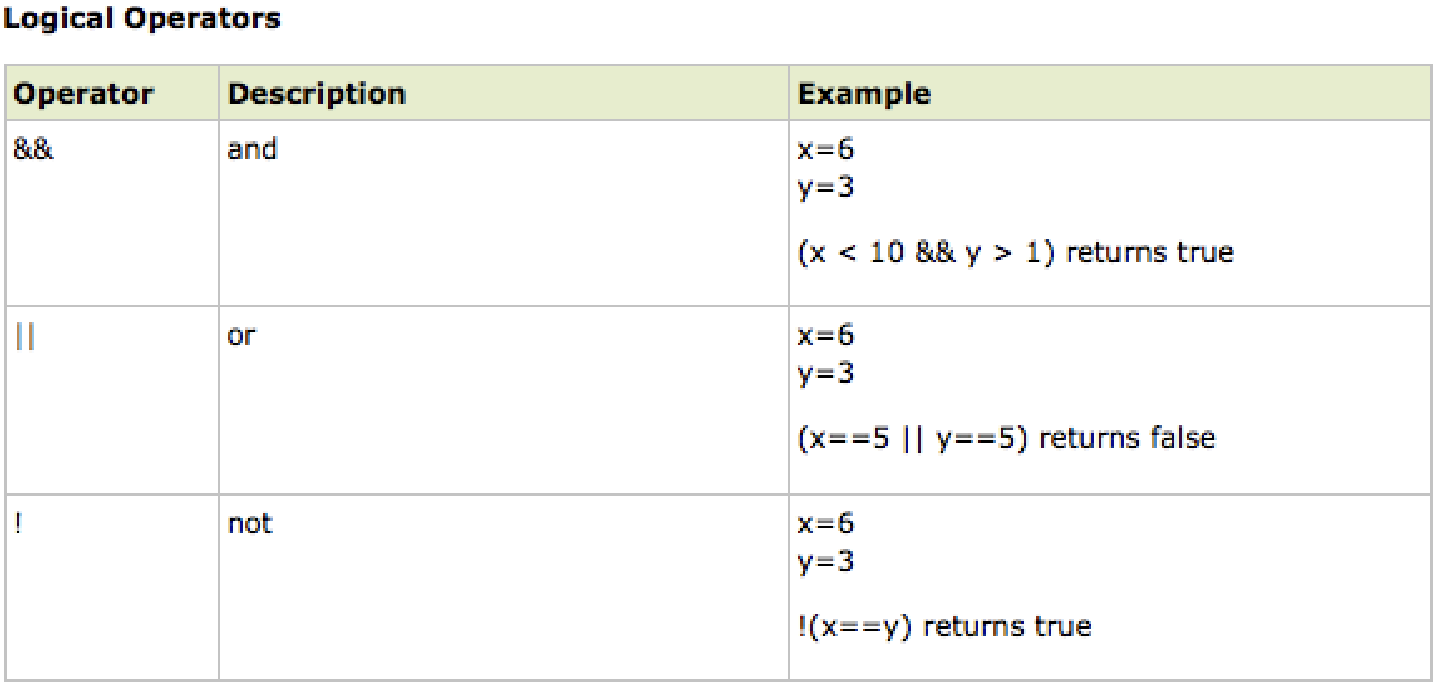
PHP Operators



Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
<>	is not equal	5<>8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

PHP Operators



Logical Operators

Operator	Description	Example

&&	and	x=6
		y=3
		(x < 10 && y > 1) returns true

||	or	x=6
		y=3
		(x==5 || y==5) returns false

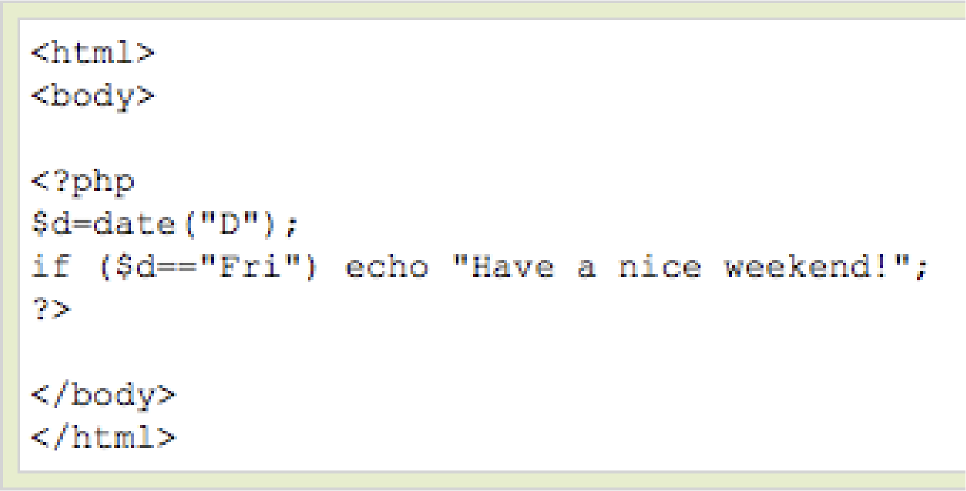
!	not	x=6
		y=3
		!(x==y) returns true

PHP Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- PHP has the following conditional statements:
  - **if** statement - use this statement to execute some code only if a specified condition is true
  - **if...else** statement - use this statement to execute some code if a condition is true and another code if the condition is false
  - **if...elseif....else** statement - use this statement to select one of several blocks of code to be executed
  - **switch** statement - use this statement to select one of many blocks of code to be executed

PHP Conditional Statements

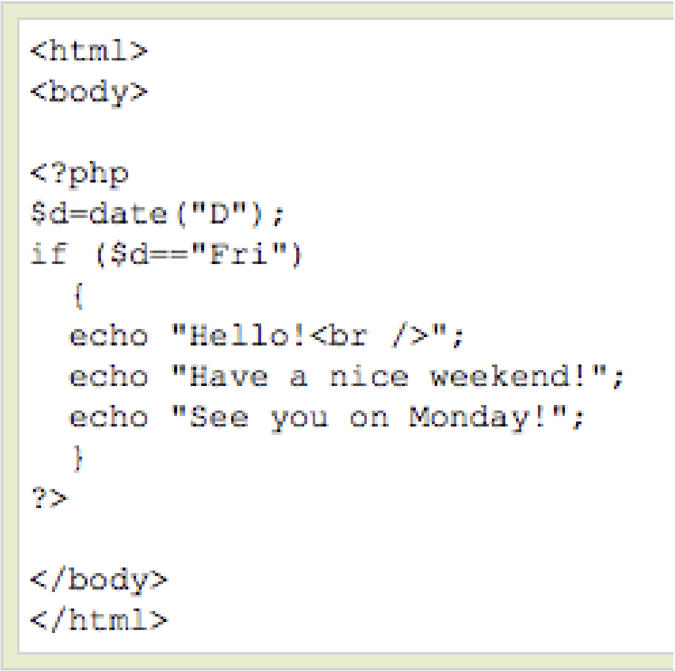
- The following example will output "Have a nice weekend!" if the current day is Friday:



```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri") echo "Have a nice weekend!";
?>

</body>
</html>
```

PHP Conditional Statements

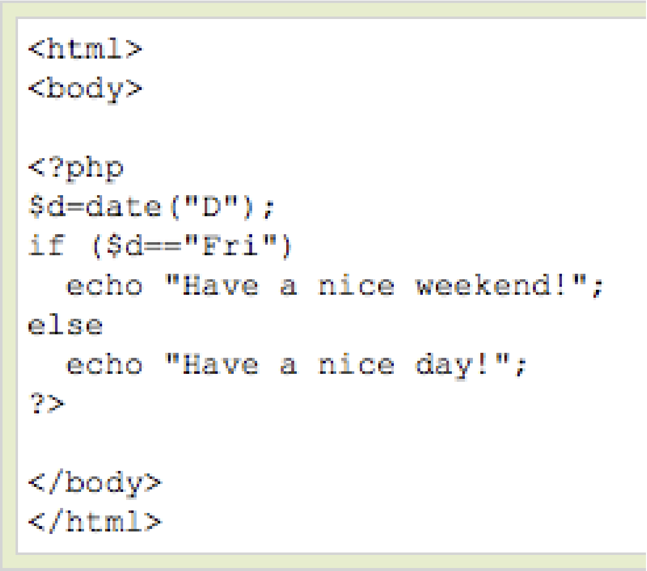
- If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces **{ }**



```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>

</body>
</html>
```

PHP Conditional Statements

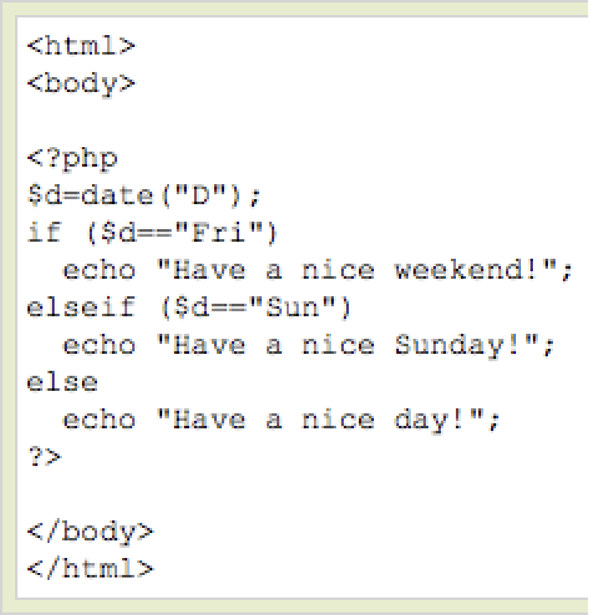
- Use the **if....else** statement to execute some code if a condition is true and another code if a condition is false.



```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

PHP Conditional Statements

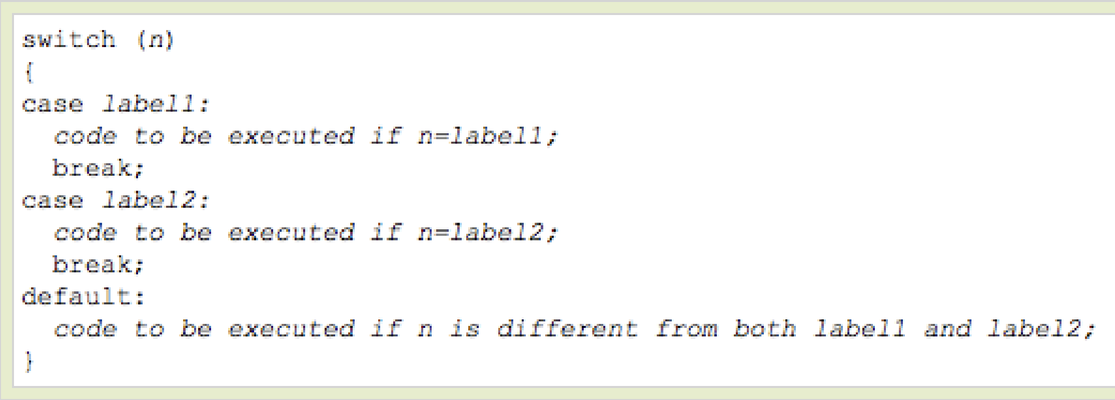
- The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":



```
<html>
<body>

<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>

</body>
</html>
```

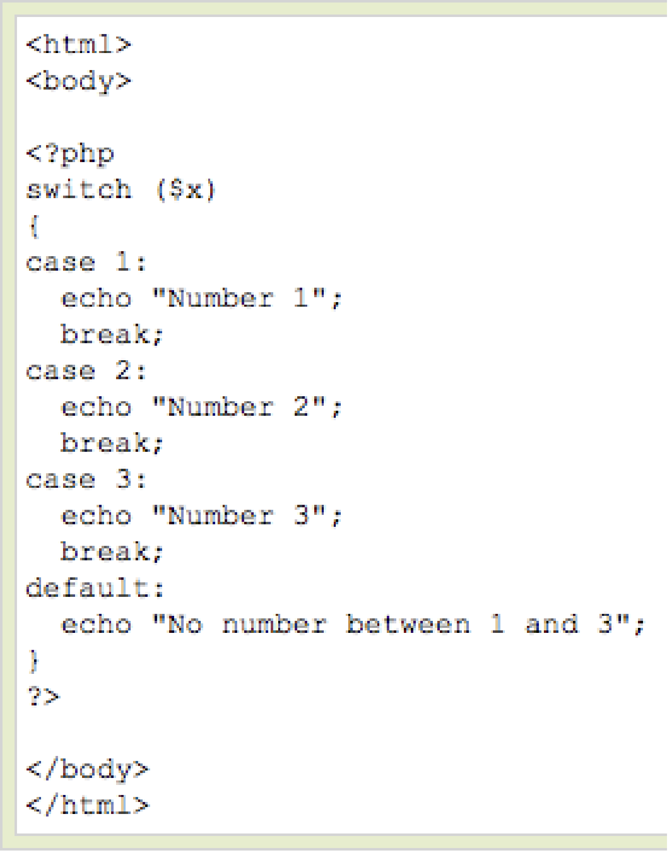
PHP Conditional Statements

- Use the switch statement to select one of many blocks of code to be executed.
- For switches, first we have a single expression n (most often a variable), that is evaluated once.
- The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed.
- Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.



```
switch (n)
{
case label1:
  code to be executed if n=label1;
  break;
case label2:
  code to be executed if n=label2;
  break;
default:
  code to be executed if n is different from both label1 and label2;
}
```

PHP Conditional Statements



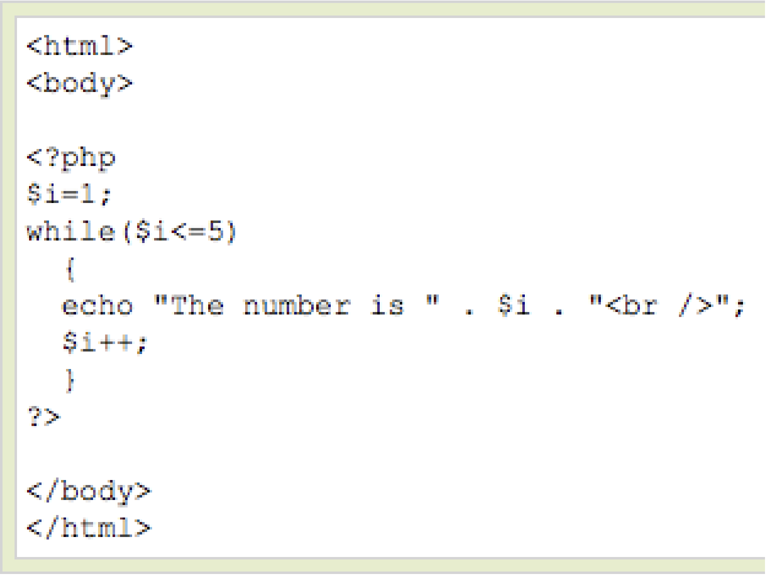
```
<html>
<body>

<?php
switch ($x)
{
case 1:
  echo "Number 1";
  break;
case 2:
  echo "Number 2";
  break;
case 3:
  echo "Number 3";
  break;
default:
  echo "No number between 1 and 3";
}
?>

</body>
</html>
```

PHP Loops

- In PHP, we have the following looping statements:
  - **while** - loops through a block of code while a specified condition is true
  - **do...while** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
  - **for** - loops through a block of code a specified number of times
  - **foreach** - loops through a block of code for each element in an array

PHP Loops - While

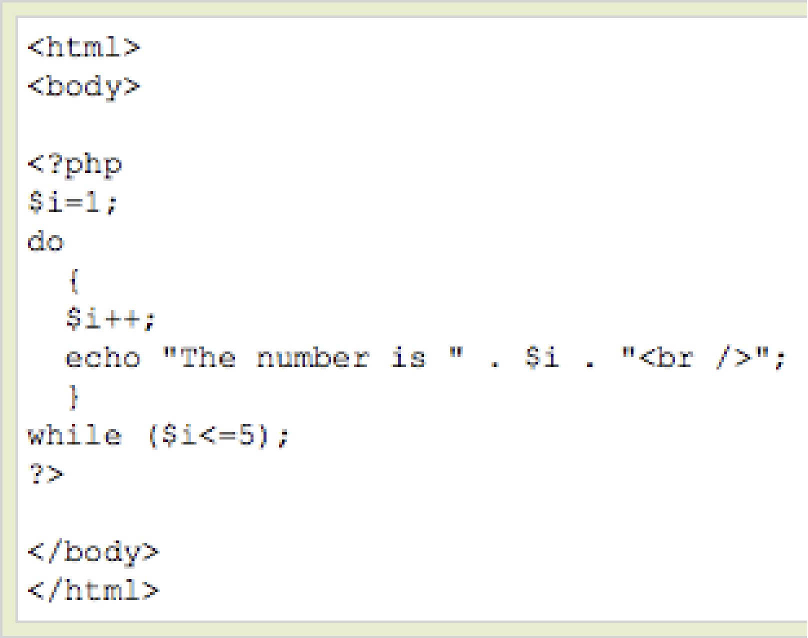
- The while loop executes a block of code while a condition is true
- The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:



```
<html>
<body>

<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>

</body>
</html>
```

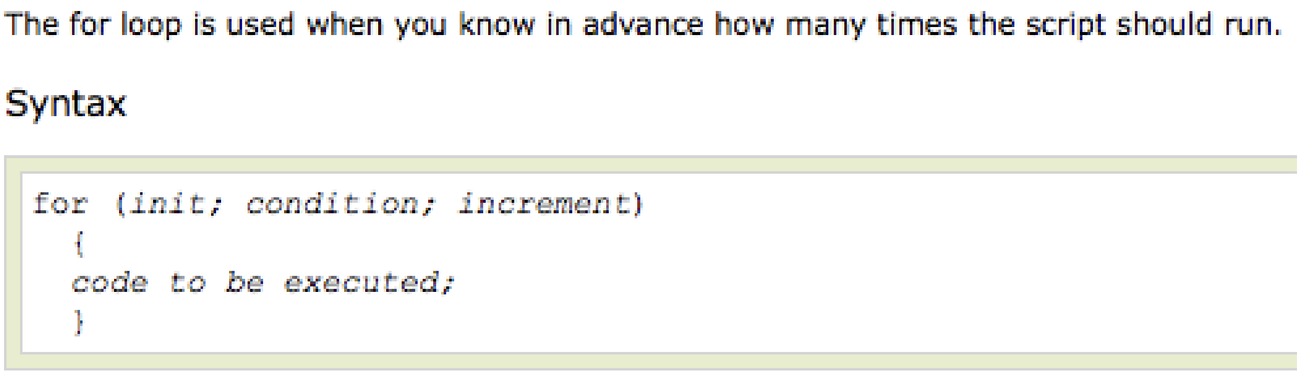
PHP Loops – Do ... While

- The do...while statement will always execute the block of code once, it will then check the condition, and repeat the loop while the condition is true
- The example below defines a loop that starts with i=1. It will then increment i with 1, and write some output. Then the condition is checked, and the loop will continue to run as long as i is less than, or equal to 5:



```
<html>
<body>

<?php
$i=1;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<=5);
?>

</body>
</html>
```

PHP Loops - For



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

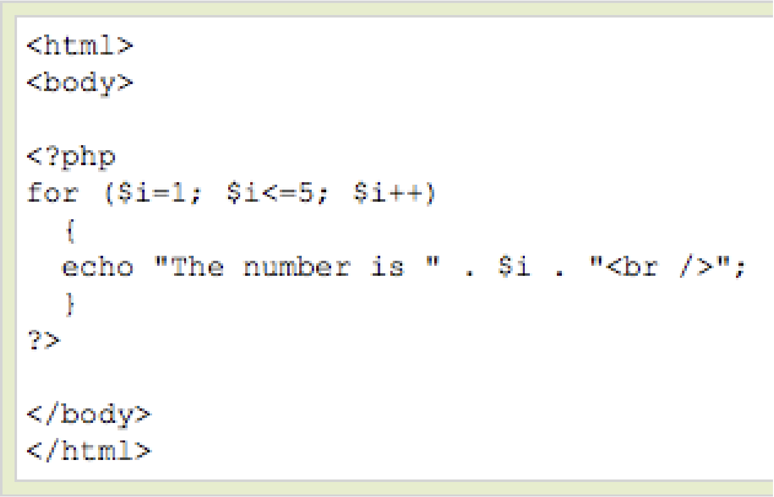
Syntax

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

- **init**: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
- **condition**: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- **increment**: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

PHP Loops - For

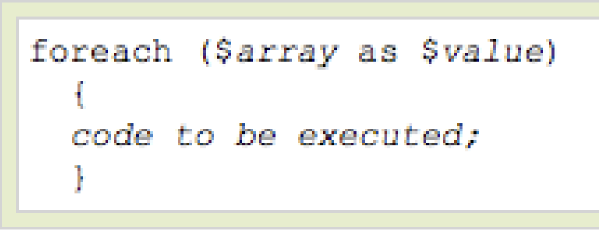
- The example below defines a loop that starts with i=1. The loop will continue to run as long as i is less than, or equal to 5. i will increase by 1 each time the loop runs:



```
<html>
<body>

<?php
for ($i=1; $i<=5; $i++)
  {
  echo "The number is " . $i . "<br />";
  }
?>

</body>
</html>
```

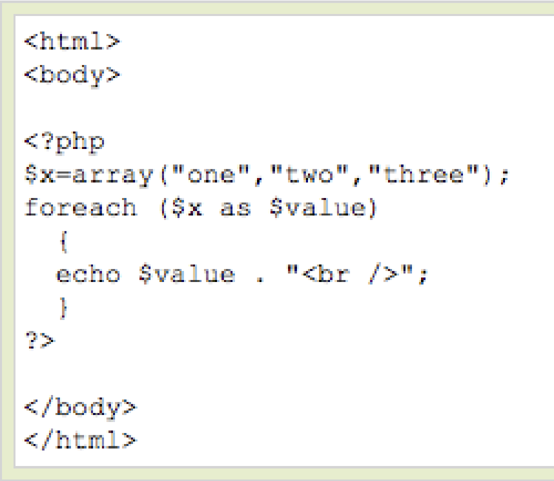
PHP Loops - Foreach

- For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.



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

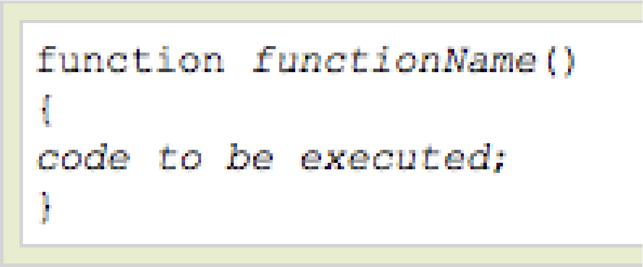
- The following example demonstrates a loop that will print the values of the given array:



```
<html>
<body>

<?php
$x=array("one","two","three");
foreach ($x as $value)
  {
  echo $value . "<br />";
  }
?>

</body>
</html>
```

PHP Functions

- To prevent a script from being executed when the page loads, you can put it into a function
- Give the function a name that reflects what the function does
- The function name can start with a letter or underscore (not a number)

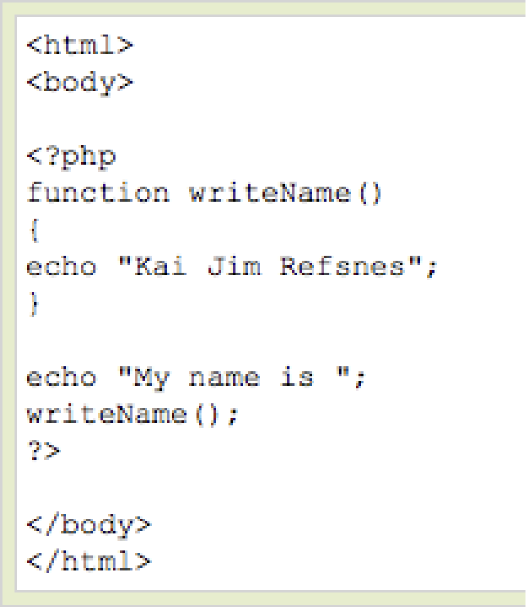


```
function functionName()
{
code to be executed;
}
```

- A function will be executed by a call to the function
- You may call a function from anywhere within a page

PHP Functions

- A function will be executed by a call to the function
- You may call a function from anywhere within a page



```
<html>
<body>

<?php
function writeName()
{
echo "Kai Jim Refsnes";
}

echo "My name is ";
writeName();
?>

</body>
</html>
```

PHP Functions - Parameters

- Adding parameters:
  - Parameters are specified after the function name, inside the parentheses

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

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

PHP Functions - Parameters

- Parameter with default value:

```
<?php
   function setHeight($minheight=50)
   {
             echo "The height is : $minheight <br>";
   }

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

PHP Functions - Parameters

- Return a value:

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

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

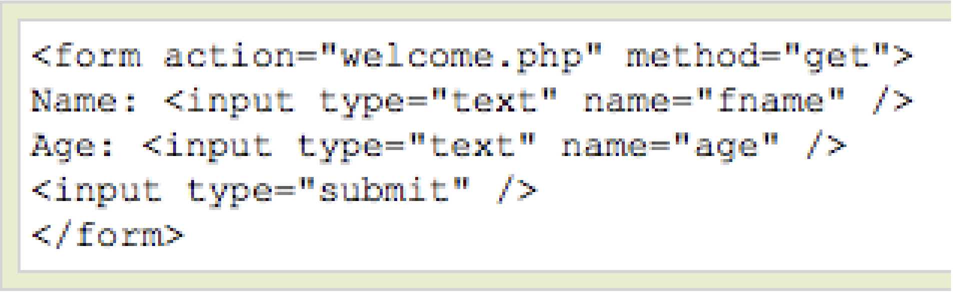
PHP Predefined Global Variables

- Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope
- Accessible from any function, class or file without having to do anything special
- The PHP predefined global variables are:
  - $GLOBALS: which is used to access user defined global variables from anywhere in the PHP script
  - $_SERVER: holds information about headers, paths, and script locations
  - $_REQUEST: used to collect data after submitting an HTML form
  - $_POST: used to collect form data after submitting an HTML form with method="post"
  - $_GET: used to collect form data after submitting an HTML form with method="get"
  - $_FILES: used to access the files uploaded from a client computer to the remote server
  - $_ENV: used to access the environment variables
  - $_COOKIE: used to retrieve a cookie value
  - $_SESSION: used to store and retrieve session variables

PHP Forms - $_GET Function

- The built-in $_GET function is used to collect values from a form sent with the *method=“get”*
- Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 2000 characters on most browsers)

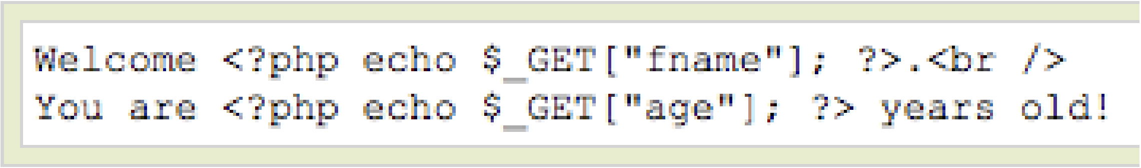
PHP Forms - $_GET Function



```
<form action="welcome.php" method="get">
Name: <input type="text" name="fname" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

- http://www.example.com/welcome.php?fname=Peter&age=37

Notice how the URL carries the information after the file name.



```
Welcome <?php echo $_GET["fname"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

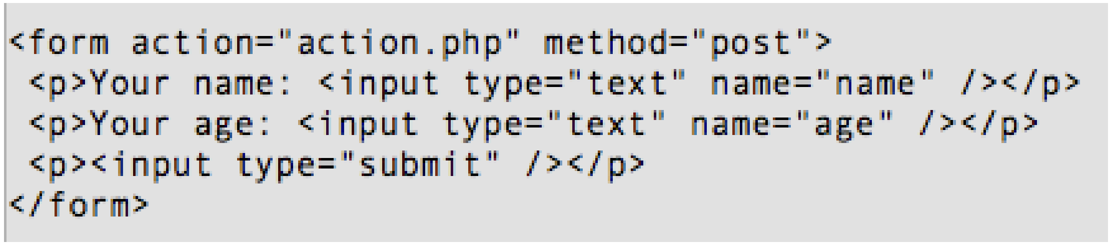
PHP Forms - $_GET Function

- When using *method="get"* in HTML forms, all variable names and values are displayed in the URL
- This method should not be used when sending passwords or other sensitive information!
- However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
- The get method is not suitable for large variable values; the value cannot exceed 2000 chars on most browsers.

PHP Forms - $_POST Function

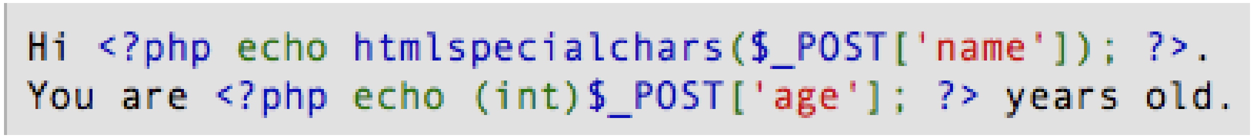
- The built-in $_POST function is used to collect values from a form sent with *method="post“*
- Appends form-data inside the body of the HTTP request
- Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send
- Note: However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post_max_size in the php.ini file).

PHP Forms - $_POST Function



```
<form action="action.php" method="post">
 <p>Your name: <input type="text" name="name" /></p>
 <p>Your age: <input type="text" name="age" /></p>
 <p><input type="submit" /></p>
</form>
```

And here is what the code of action.php might look like:



```
Hi <?php echo htmlspecialchars($_POST['name']); ?>.
You are <?php echo (int)$_POST['age']; ?> years old.
```

PHP Forms - $_POST Function

- Apart from **htmlspecialchars()** and **(int)**, it should be obvious what this does. **htmlspecialchars()** makes sure any characters that are special in html are properly encoded so people can't inject HTML tags or Javascript into your page
- For the age field, since we know it is a number, we can just convert it to an integer which will automatically get rid of any stray characters. The **$_POST['name']** and **$_POST['age']** variables are automatically set for you by PHP.

PHP – Form Validation

```
<?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 – Form Validation

```
<?php

        // define variables and set to empty values
        $nameErr = $emailErr = "";
        $name = $email = "";
        if ($_SERVER["REQUEST_METHOD"] == "POST")
        {

          if (empty($_POST["name"]))
          {$nameErr = "Name is required";}
          else
          {
                $name = test_input($_POST["name"]);
                 if (!preg_match("/^[a-zA-Z ]*$/",$name))
                 {
                      $nameErr = "Only letters and white space allowed";
                 }
          }
          if (empty($_POST["email"]))
          {$emailErr = "Email is required";}
          else
          {
                  $email = test_input($_POST["email"]);}
                 if (!preg_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email))
                 {
                   $emailErr = "Invalid email format";
                 }
          }
?>
```

PHP - Cookie

- A cookie is often used to identify a user
- A cookie is a small file that is stored on the user's computer
- Each time the same computer requests a page with a browser, it will send the cookie too
- With PHP, you can both create and retrieve cookie values
  - Create:
    - setcookie(name, value, expire, path, domain);
      - The setcookie() function must appear BEFORE the <html> tag

```
<?php
setcookie("user", "Alex Porter", time()+3600);
?>

<html>
.....
```

PHP - Cookie

- Read/Retrieve:
  - The PHP $_COOKIE variable is used to retrieve a cookie value

```
<html>
<body>

<?php
if (isset($_COOKIE["user"]))
  echo "Welcome " . $_COOKIE["user"] . "!<br>";
else
  echo "Welcome guest!<br>";
?>

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

PHP - Session

- A PHP session variable is used to store information about, or change settings for a user session on the server
- Session variables hold information about one single user, and are available to all pages in one application
- A unique session id is associated with a user’s session
  - Session id returned to the client inside a cookie
  - Example use case: session based user authentication/authorization
- Starting/Resuming PHP Session:
  - Before storing user information in PHP session:
    - first start up the session
- Storing/Retrieve Session Variable:
  - PHP $_SESSION variable used to store and retrieve session variables
- Delete Session Data:
  - Use unset() function to delete some session data
- Destroy Session:
  - session_destroy() function to destroy session

PHP - Simple Login using Session

```
<?php
//Start session
session_start();

//Check whether the session variable SESS_MEMBER_ID is present or not
if(!isset($_SESSION['sess_user_id']) || (trim($_SESSION['sess_user_id']) == '')) {
  header("location: login.html");
  exit();
}
?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>Home Page</title>
  </head>

  <body>
    <h1>Welcome, <?php echo $_SESSION["sess_username"] ?></h1>
  </body>
</html>
```

PHP - Simple Login using Session

```
<?php
session_start();

$username = $_POST['username'];
$password = $_POST['password'];
$con=mysqli_connect("example.com","peter","abc123","my_db");
$username = mysql_real_escape_string($username);
$query = "SELECT id, username, password, salt FROM member WHERE username = '$username';";
$result = mysqli_query ($con,$query);
if($result->num_rows == 0) // User not found. So, redirect to login_form again.
{
  header('Location: login.html');
  exit();
}

$userData = mysqli_fetch_array($result, MYSQL_ASSOC);
$hash = hash('sha256', $userData['salt'] . hash('sha256', $password) );
mysqli_close($con);

if($hash != $userData['password']) // Incorrect password. So, redirect to login_form again.
{
  header('Location: login.html');
  exit();
}else{ // Redirect to home page after successful login.
  session_regenerate_id(); //recommended since the user session is now authenticated
  $_SESSION['sess_user_id'] = $userData['id'];
  $_SESSION['sess_username'] = $userData['username'];
  session_write_close();
  header('Location: home.php');
}
?>
```

PHP - MySQL

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

PHP - MySQL

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

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

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

PHP - MySQL

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

PHP – MySQL

- MySQLi - Procedural
- mysqli_close($conn);

- MySQLi - Object oriented
- $conn->close();

- PDO
- $conn = null;

PHP – MySQL

- Insert data into tables:
  - INSERT INTO table_name VALUES (value1, value2, value3,...)
  - INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,...)
- MySQL:

```
<?php
$con=mysqli_connect("example.com","peter","abc123","my_db");
// Check connection
if (mysqli_connect_errno())
{
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}
// escape variables for security
$firstname = mysqli_real_escape_string($_POST['firstname']);
$lastname = mysqli_real_escape_string($_POST['lastname']);
$age = mysqli_real_escape_string($_POST['age']);
$sql="INSERT INTO Persons (FirstName, LastName, Age)
VALUES ($firstname, $lastname, $age)";
if (!mysqli_query($con,$sql))
{
  die('Error: ' . mysqli_error($con));
}
echo "1 record added";
mysqli_close($con);
?>
```

PHP – MySQL/PostGreSQL

- Select data from tables:
  - SELECT statement is used to select data from a database
    - SELECT column_name(s) FROM table_name WHERE column_name operator value

```
<?php
$con=mysqli_connect("example.com","peter","abc123","my_db");
// Check connection
if (mysqli_connect_errno())
{
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

$result = mysqli_query($con,"SELECT * FROM Persons
WHERE FirstName='Peter'");

while($row = mysqli_fetch_array($result))
{
  echo $row['FirstName'] . " " . $row['LastName'];
  echo "<br>";
}
?>
```

PHP – Prepared Statements

- $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();

- echo "New records created successfully";

PHP – AJAX

```
<html><head>
<script>
function showUser(str)
{
  if (str=="")
  {
    document.getElementById("txtHint").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest)
  {// code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  }
  else
  {// code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function()
  {
    if (xmlhttp.readyState==4 && xmlhttp.status==200)
    {
       document.getElementById("txtHint").innerHTML=xmlhttp.responseText;
    }
  }
  xmlhttp.open("GET","getuser.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<form>
<select name="users" onchange="showUser(this.value)">
<option value="">Select a person:</option>
<option value="1">Peter Griffin</option>
<option value="2">Lois Griffin</option>
</select>
</form><br>
<div id="txtHint"><b>Person info will be listed here.</b></div>
</body></html>
```

PHP – AJAX

```
<?php
$q = intval($_GET['q']);

$con = mysqli_connect('localhost','peter','abc123','my_db');
if (!$con)
{
  die('Could not connect: ' . mysqli_error($con));
}

mysqli_select_db($con,"ajax_demo");
$sql="SELECT * FROM user WHERE id = '".$q."'";
$result = mysqli_query($con,$sql);

echo "<table border='1'>
<tr>
<th>Firstname</th>
<th>Lastname</th>
<th>Age</th>
<th>Hometown</th>
<th>Job</th>
</tr>";

while($row = mysqli_fetch_array($result))
  {
  echo "<tr>";
  echo "<td>" . $row['FirstName'] . "</td>";
  echo "<td>" . $row['LastName'] . "</td>";
  echo "<td>" . $row['Age'] . "</td>";
  echo "<td>" . $row['Hometown'] . "</td>";
  echo "<td>" . $row['Job'] . "</td>";
  echo "</tr>";
  }
echo "</table>";

mysqli_close($con);
?>
```