PHP -1

PHP Introduction

- PHP is a recursive acronym for “PHP: Hypertext Preprocessor”.
- It is a widely-used open source general-purpose scripting language that is especially suited for web development and can be embedded into HTML.
- "Preprocessor" indicates that PHP code is executed on the server before the resulting HTML is sent to the client's browser. This means that PHP scripts are parsed by the server, and the output is generated as HTML, which is then transmitted to the user's browser

PHP Introduction

- PHP is a server-side scripting language
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is open source software
- PHP is free to download and use

PHP Introduction

- 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

PHP basic syntax

How to write PHP in HTML

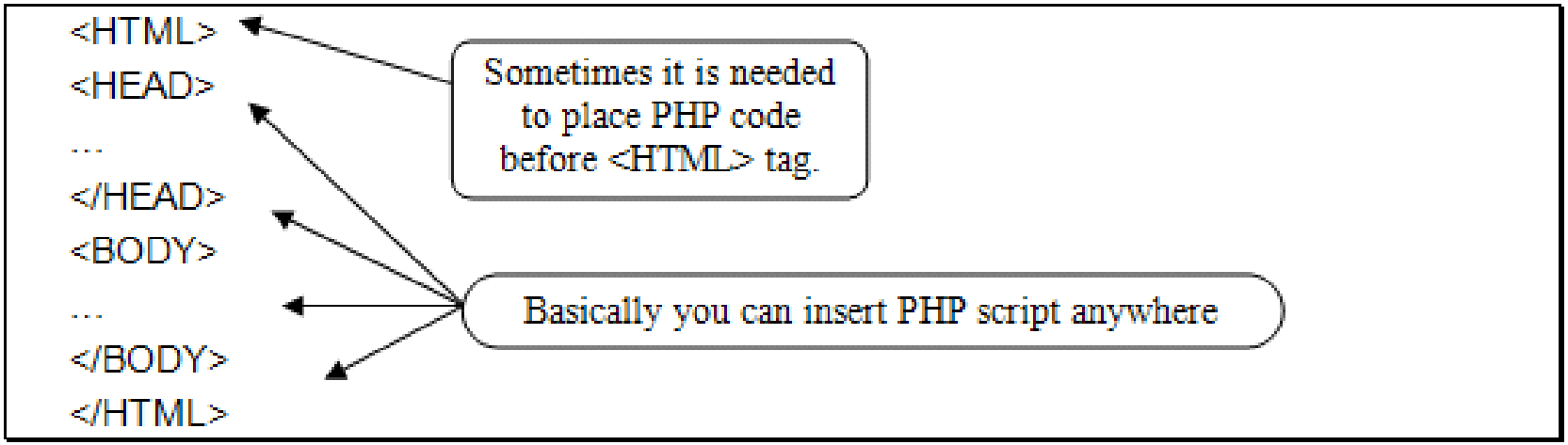
Begin with **<?php** , and end with **?>**

```
<?php
.....PHP code.....
?>
```

IMPORTANT: PHP code is case-sensitive (like JavaScript) **but** HTML is not case-sensitive.

Where to insert PHP script in HTML?

You can insert PHP script basically anywhere in HTML page. There is no limitation of the location.



IMPORTANT: The file must be saved with the extension of "**.php**", not "**.html**"

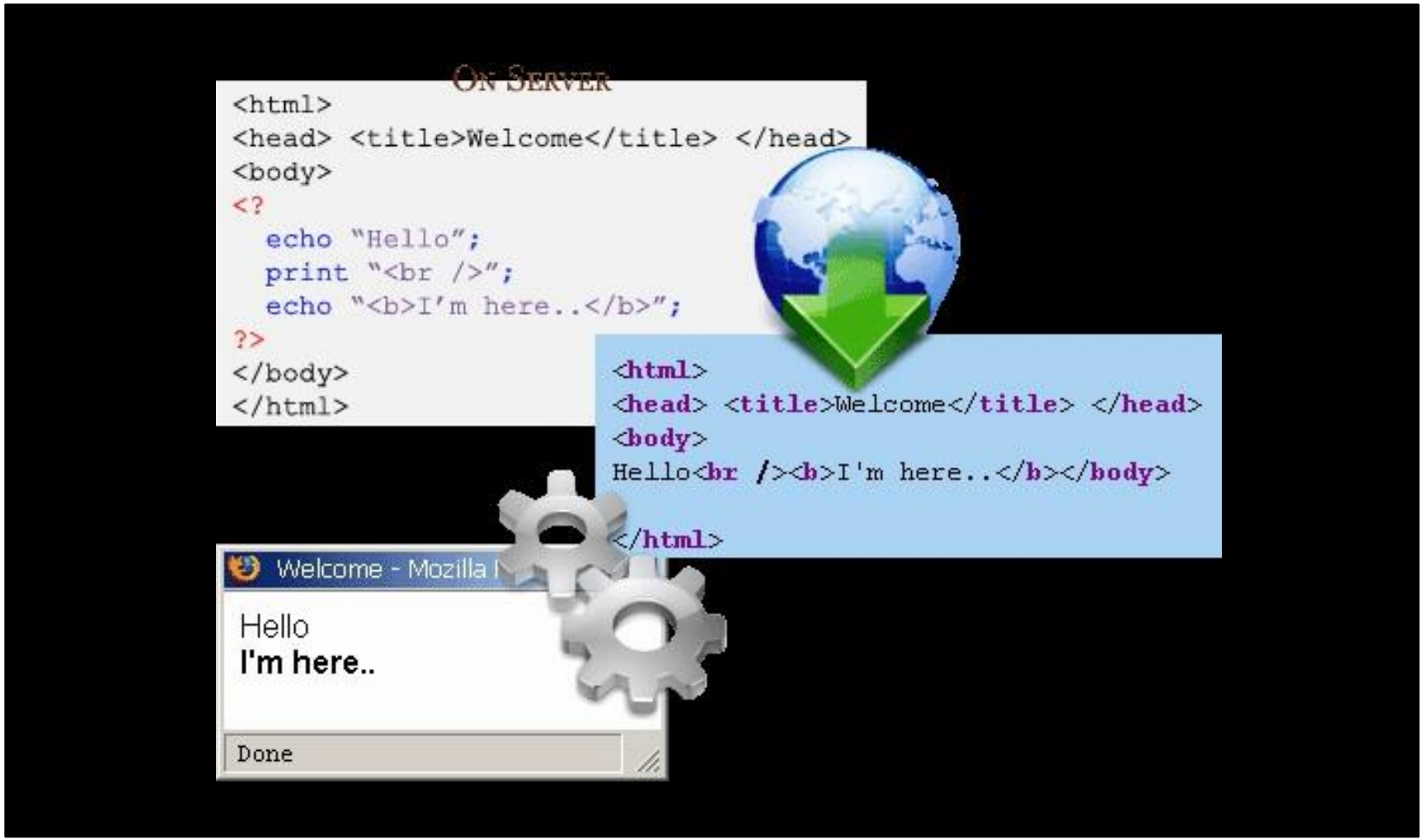
PHP Introduction

```
<!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.

PHP Introduction



PHP Getting Started

- On windows, you can download and install WAMP. With one installation and you get an Apache webserver, database server and php.
- http://www.wampserver.com

- On mac, you can download and install MAMP.
- http://www.mamp.info/en/index.html

PHP Hello World

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

- Above is the PHP source code.

PHP Hello World

- It renders as HTML that looks like this:

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

echo()

Outputs all parameters.

echo() is not actually a function (it is a language construct), so you are not required to use parentheses with it. **echo()** (unlike some other language constructs) does not behave like a function, so it cannot always be used in the context of a function.

- The **PHP command 'echo' is used to output the parameters passed to it**
- The typical usage for this is to send data to the client's web-browser
- Syntax
  - void **echo** (string arg ***1*** [, string arg ***n***...])

Additionally, if you want to pass more than one parameter to **echo()**, the parameters must not be enclosed within parentheses.

```
echo "<table border=1>";
```

echo is another command we can use to output message. echo is more useful because you can pass it several parameters, like this: <?php echo "This ", "is ", "a ", "test."; ?>

Echo example

```
<?php
$foo = 25;              // Numerical variable
$bar = "Hello";         // String variable

echo $bar;              // Outputs Hello
echo $foo,$bar;         // Outputs 25Hello
echo "5x5=",$foo;       // Outputs 5x5=25
echo "5x5=$foo";        // Outputs 5x5=25
echo '5x5=$foo';        // Outputs 5x5=$foo
?>
```

- **Notice** how echo '5x5=$foo' outputs $foo rather than replacing it with 25
- **Strings in single quotes (' ') are not interpreted or evaluated by PHP**
- This is true for both variables and character escape-sequences (such as "\n" or "\\")

PHP Comments

- In PHP, we 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

- Variables are used for storing values, like text strings, numbers or arrays.
- When a variable is declared, it can be used over and over again in your script.
- All variables in PHP start with a $ sign symbol.
- The correct way of declaring a variable in PHP:

```
$var_name = value;
```

PHP Variables

```
<?php
$txt="Hello World!";
$x=16;
?>
```

- In PHP, a **variable does not need to be declared before adding a value to it.**
- In the example above, you see that you **do not have to tell PHP which data type the variable is**.
- PHP **automatically converts the variable to the correct data type, depending on its value.**

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

PHP Variables

- A variable name must start with a letter or an underscore "_" and not a number,
- A variable name can only contain alpha-numeric characters, underscores (a-z, A-Z, 0-9, and _)
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with an underscore ($my_string) or with capitalization ($myString)

PHP Variables ...

- Case-sensitive ($Foo != $foo != $fOo)
- Global and locally-scoped variables
- Global variables can be used anywhere
- Local variables restricted to a function or class
- Certain variable names reserved by PHP
- Form variables ($_POST, $_GET)
- Server variables ($_SERVER)

PHP Variables ...Global and Local Scope

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

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

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

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

Output:-
Variable x inside function is:
Variable x outside function is: 5

PHP Variables ...Global and Local Scope

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

```
<?php
function myTest() {
    $x = 5; / /  local scope
    echo "<p>Variable x inside function is:
$x</p>";
}
myTest();

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

Output:-
Variable x inside function is: 5
Variable x outside function is:

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

PHP The global Keyword

- The global keyword is used to access a global variable from within a function.
- To do this, use the global keyword before the variables (inside the function):

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

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

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

Output:-
15

PHP The static Keyword

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

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

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

Output:-

0

1

2

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

Constants

A constant is an identifier (name) for a simple value. A constant is case-sensitive by default. By convention, constant identifiers are always uppercase.

```
<?php

// Valid constant names
define("FOO",     "something");
define("FOO2",    "something else");
define("FOO_BAR", "something more");

// Invalid constant names  (they shouldn't start
//      with a number!)

define("2FOO",    "something");

// This is valid, but should be avoided:
// PHP may one day provide a "magical" constant
// that will break your script

define("__FOO__", "something");

?>
```

You can access constants anywhere in your script without regard to scope.

PHP Concatenation

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

```
<?php
$txt1="Hello World!";
$txt2="What a nice day!";
echo $txt1 . " " . $txt2;
?>
```

PHP Concatenation

- The output of the code on the last slide will be:

```
Hello World! What a nice day!
```

- We have used the concatenation operator two times. This is because we had to insert a third string (a space character), to separate the two strings.

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 Arithmetic Operators

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

<?php
$x = 10;
$y = 6;

echo $x + $y;
?>

</body>
</html>
//output:16
```

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

<?php
$x = 10;
$y = 6;

echo $x - $y;
?>

</body>
</html>
//output:4
```

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

<?php
$x = 10;
$y = 6;

echo $x * $y;
?>

</body>
</html>
//output: 60
```

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

<?php
$x = 10;
$y = 6;

echo $x / $y;
?>

</body>
</hztml>
//OutPut:1.6666666666667
```

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

<?php
$x = 10;
$y = 6;
echo $x % $y;
?>
</body>
</html>
Output:4
```

Arithmetic Operations

```
<?php
        $a=15;
        $b=30;
        $total=$a+$b;
        Print $total;
        Print "<p><h1>$total</h1>";
        // total is 45
?>
```

- $a - $b / / subtraction
- $a * $b / / multiplication
- $a / $b / / division
- **$a += 5** / / **$a = $a+5** Also works for *= and /=

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 Assignment Operators

1.
```
<?php
$x = 10;
echo $x;
?>
```
Output:10

2.
```
<?php
$x = 20;
$x += 100;

echo $x;
?>
```
OutPut:120

3.
```
<?php
$x = 50;
$x -= 30;

echo $x;
?>
```
Output: 20

4.
```
<?php
$x = 10;
$y = 6;

echo $x * $y;
?>
```
Output: 60

5.
```
<?php
$x = 10;
$x / = 5;

echo $x;
?>
```
Output:2

6.
```
<?php
$x = 15;
$x %= 4;

echo $x;
?>
```
Output:3

1. The left operand gets set to the value of the expression on the right
2. Addition
3. Subtraction
4. Multiplication
5. Division
6. Modulus

PHP Comparison Operators

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y
!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

PHP Comparison Operators

1,
```
<?php
$x = 100;
$y = "100";

var_dump($x == $y); / /  returns true because values are equal
?>
```

2.
```
<?php
$x = 100;
$y = "100";

var_dump($x === $y); / /  returns false because types are not equal
?>
```

3.
```
<?php
$x = 100;
$y = "100";

var_dump($x != $y); / /  returns false because values are equal
?>
```

4.
```
<?php
$x = 100;
$y = "100";

var_dump($x <> $y); / /  returns false because values are equal
?>
```

PHP Comparison Operators ...

5,
```
<?php
$x = 100;
$y = "100";

var_dump($x !== $y); / /  returns true because types are not equal
?>
```

6.
```
<?php
$x = 100;
$y = 50;

var_dump($x > $y); / /  returns true because $x is greater than $y
?>
```

7.
```
<?php
$x = 10;
$y = 50;

var_dump($x < $y); / /  returns true because $x is less than $y
?>
```

8.
```
<?php
$x = 50;
$y = 50;

var_dump($x >= $y); / /  returns true because $x is greater than or equal to $y
?>
```

PHP Comparison Operators ...

```
9,
<?php
$x = 50;
$y = 50;

var_dump($x <= $y); / /  returns true because
$x is less than or equal to $y
?>
```

PHP Operators

The PHP logical operators are used to combine conditional statements.

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

PHP Logical Operators

1.
```
<?php
$x = 100;
$y = 50;

if ($x == 100 and $y == 50) {
    echo "Hello world!";
}
?>
```
Output:
Hello world!

2.
```
<?php
$x = 100;
$y = 50;

if ($x == 100 or $y == 80) {
    echo "Hello world!";
}
?>
```
Output:
Hello world!

3.
```
<?php
$x = 100;
$y = 50;

if ($x == 100 xor $y == 80) {
    echo "Hello world!";
}
?>
```
Output:
Hello world!

PHP Logical Operators …

4.
```
<?php
$x = 100;
$y = 50;

if ($x == 100 && $y == 50) {
    echo "Hello world!";
}
?>Output:
Hello world!
```

5.
```
<?php
$x = 100;
$y = 50;

if ($x == 100 | |  $y == 80) {
    echo "Hello world!";
}
?>Output:
Hello world!
```

6.
```
<?php
$x = 100;

if ($x !== 90) {
    echo "Hello world!";
}
?> Output:
Hello world!
```

PHP Increment / Decrement Operators

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

Operator	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--	Post-decrement	Returns $x, then decrements $x by one

PHP Increment / Decrement Operators ...

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

1.
```
<?php
$x = 10;
echo ++$x;
?>
```
Output:
11

2.
```
<?php
$x = 10;
echo $x++;
?>
```
Output:
10

3.
```
<?php
$x = 10;
echo --$x;
?>
?>
```
Output:
9

4.
```
php
$x = 10;
echo $x--;
?>
```
Output:
10

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

1.
```
<?php
$txt1 = "Hello";
$txt2 = " world!";
echo $txt1 . $txt2;
?>
```
Output:
Hello world!

2.
```
<?php
$txt1 = "Hello";
$txt2 = " world!";
$txt1 .= $txt2;
echo $txt1;
?>
```
Output:
Hello world!

PHP Array Operators

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 Array Operators ...

1. Union

```
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");

print_r($x + $y); / / union of $x and $y
?>
```

Output:

```
Array ( [a] => red [b] => green [c] => blue [d] => yellow )
```

2. Equality

```
<?php
$x = array("a" => "red", "b" => "green");
$y
= array("c" => "blue", "d" => "yellow");

var_dump($x == $y);
?>
```

Output:

```
bool(false)
```

PHP Array Operators ...

3.Identity

```
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");

var_dump($x === $y);
?>
```

Output:

```
bool(false)
```

4. inequality

```
<?php
$x = array("a" => "red", "b" => "green");
$y
= array("c" => "blue", "d" => "yellow");

var_dump($x != $y);
?>
```

Output:

```
bool(true)
```

PHP Array Operators ...

5.inequality

```
<?php
$x = array("a" => "red", "b" => "green");
$y = array("c" => "blue", "d" => "yellow");

var_dump($x <> $y);
?>
```

Output:

bool(true)

6.Non identity

```
<?php
$x = array("a" => "red", "b" => "green");
$y
= array("c" => "blue", "d" => "yellow");

var_dump($x !== $y);
?>
```

Output:

bool(true)

Escaping the Character

- If the string has a set of double quotation marks that must remain visible, use the \ [backslash] before the quotation marks to ignore and display them.

```
<?php
$heading="\"Computer Science\"";
Print $heading;
?>
```

```
"Computer Science"
```

PHP Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- You can use conditional statements in your code to do this.
- In PHP we have the following conditional statements…

PHP 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

- 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

- 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

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.

```
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

- 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.

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 Arrays

- An array variable is a storage area holding a number or text.The problem is, a variable will hold only one value.
- An array is a special variable, which can store multiple values in one single variable.

PHP Arrays

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

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

PHP Arrays

- However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?
- The best solution here is to use an array.
- An array can hold all your variable values under a single name. And you can access the values by referring to the array name.
- Each element in the array has its own index so that it can be easily accessed.

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.

PHP Numeric Arrays

- 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:

```
$cars[0]="Saab";
$cars[1]="Volvo";
$cars[2]="BMW";
$cars[3]="Toyota";
```

PHP Numeric Arrays

- In the following example you access the variable values by referring to the array name and index:

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

- The code above will output:

```
Saab and Volvo 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.

PHP Associative Arrays

- 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.

PHP Multidimensional Arrays

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
$arr_combined = array(
    "arr_colors"=>array("red", "green", "blue"),
    "arr_numbers"=>array("one", "two", "three"),
    "arr_cars"=>array("toyota", "honda", "nissan")
);

echo $arr_combined['arr_colors'][0];
echo "<br>";
echo $arr_combined['arr_numbers'][1];
echo "<br>";
echo $arr_combined['arr_cars'][2];
echo "<br>";

foreach ($arr_combined as $key => $value) {
    echo "$key: ";
    foreach ($value as $item) {
        echo "$item, ";
    }
    echo "<br>";
}
```

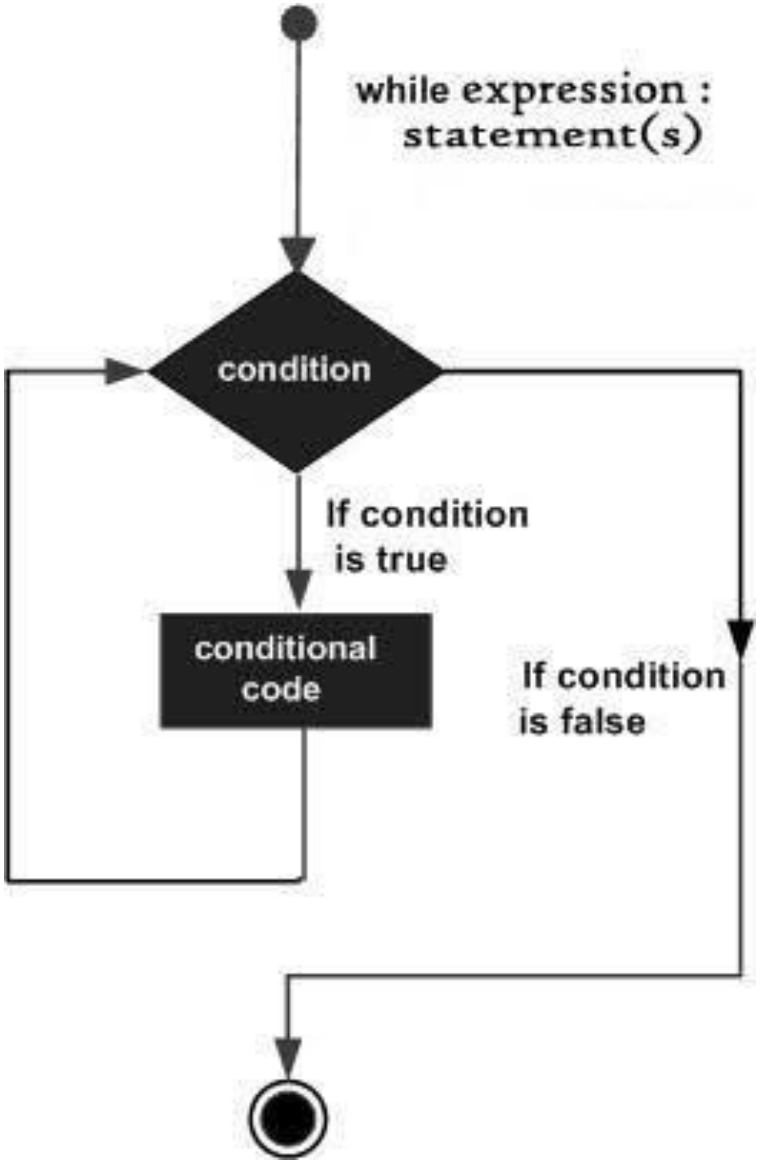
PHP Loops

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

PHP Loops

- **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 statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be executed. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

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 - While

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

```
<html>
  <body>
      <?php
      $i = 0;
      $num = 50;

      while( $i < 10) {
        $num--;
        $i++;
      }

      echo ("Loop stopped at i = $i and num = $num" );
    ?>

  </body>
</html>
```

This will produce the following result −

```
Loop stopped at i = 10 and num = 40
```

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 next example 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:

PHP Loops – Do ... While

```
<html>
<body>

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

</body>
</html>
```

PHP Loops – Do ... While

Output:

```
The number is 2
The number is 3
The number is 4
The number is 5
The number is 6
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 −

```
<html>
   <body>
      <?php
         $i = 0;
         $num = 0;

         do {
            $i++;
         }

         while( $i < 10 );
         echo ("Loop stopped at i = $i" );
      ?>

   </body>
</html>
```

This will produce the following result −

```
Loop stopped at i = 10
```

PHP Loops - For

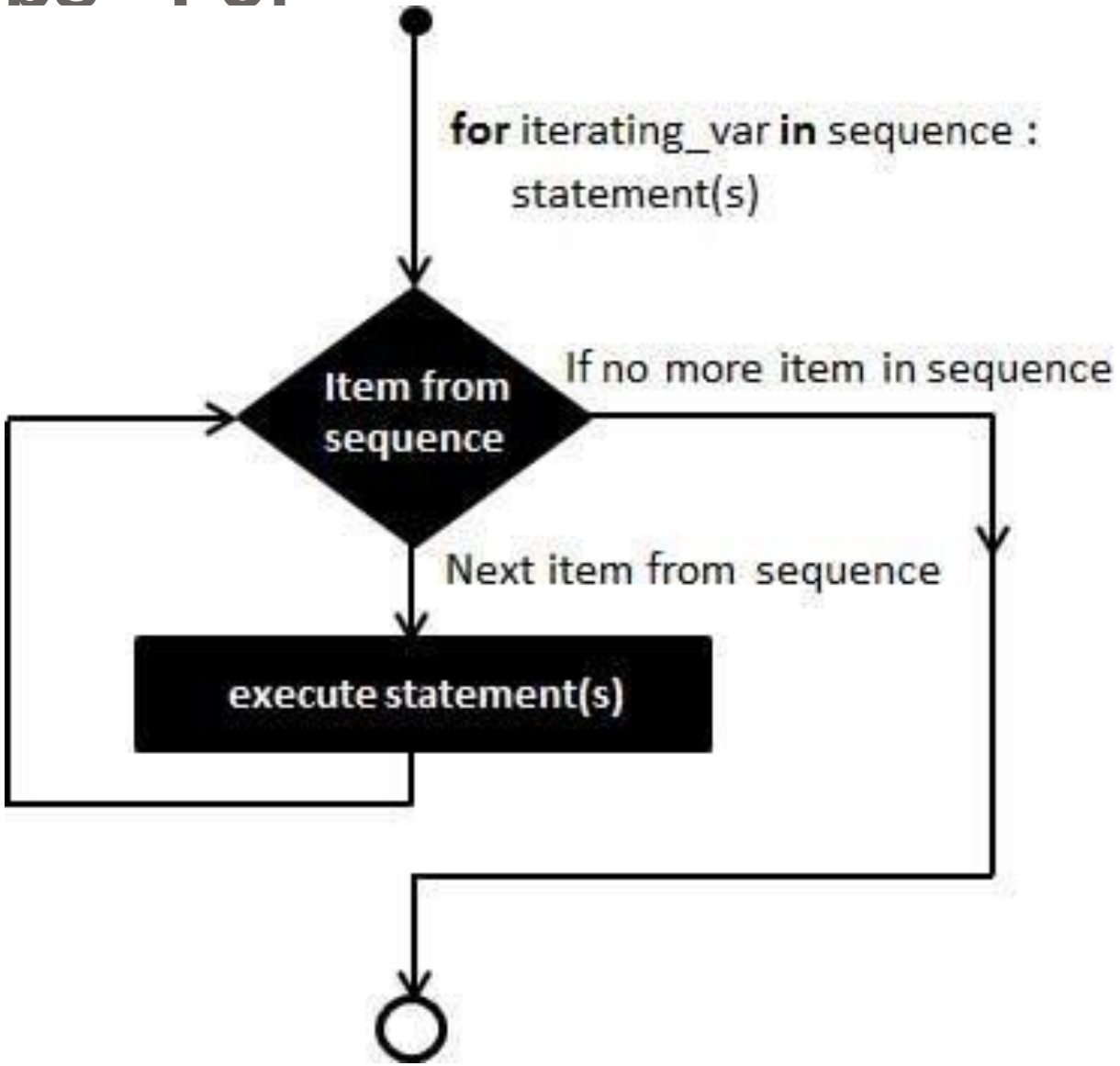
The for statement is used when you know how many times you want to execute a statement or a block of statements.

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;
  }
```

PHP Loops - For



PHP Loops - For

- Parameters:
- **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>
```

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

PHP Loops - For

Output:

```
The number is 1
The number is 2
The number is 3
The number is 4
The number is 5
```

PHP Loops - Foreach

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

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

PHP Loops - Foreach

- 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 Loops - Foreach

Output:

```
one
two
three
```

Foreach Loop

Example

Try out following example to list out the values of an array.

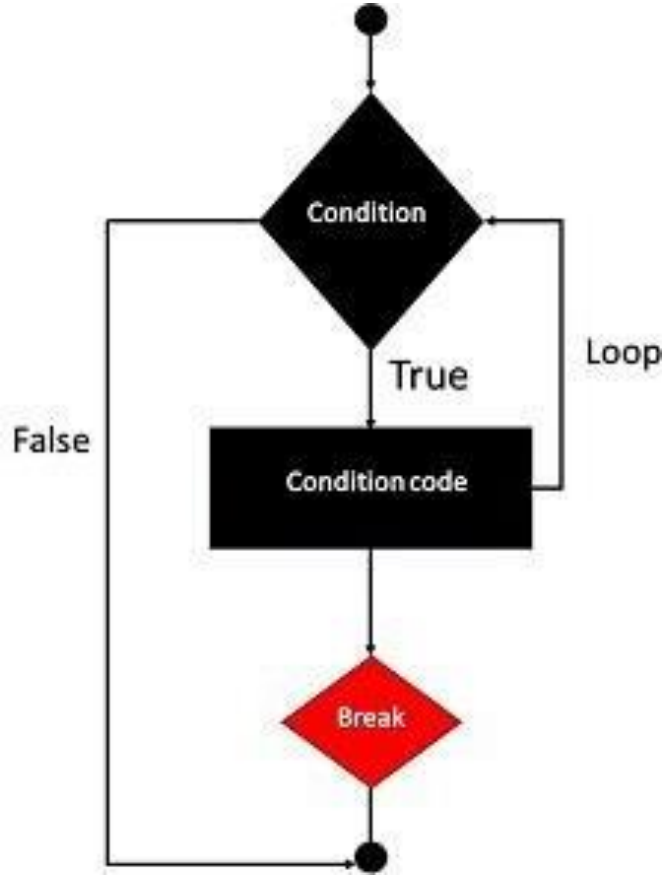
```
<html>
  <body>
      <?php
      $array = array( 1, 2, 3, 4, 5);
foreach( $array as $value ) {
         echo "Value is $value <br />";
      }
    ?>
</body>
</html>
```

This will produce the following result −

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

The break statement

The PHP **break** keyword is used to terminate the execution of a loop prematurely.
The **break** statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



The break statement

Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

```
<html>
  <body>
<?php
      $i = 0;
while( $i < 10) {
        $i++;
        if( $i == 3 )break;
      }
      echo ("Loop stopped at i = $i" );
    ?>
</body>
</html>
```

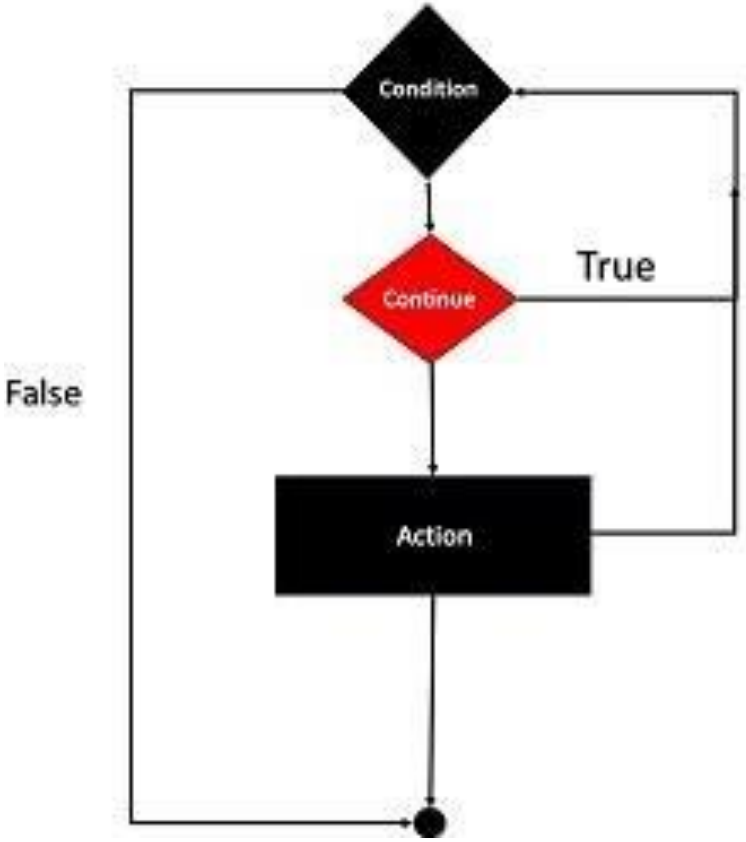
This will produce the following result −

```
Loop stopped at i = 3
```

The continue statement

The PHP **continue** keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the **break** statement the **continue** statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering **continue** statement, rest of the loop code is skipped and next pass starts.



The continue statement

Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

```
<html>
  <body>
      <?php
      $array = array( 1, 2, 3, 4, 5);
foreach( $array as $value ) {
         if( $value == 3 )continue;
         echo "Value is $value <br />";
      }
    ?>
</body>
</html>
```

This will produce the following result −

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP Functions

- We will now explore how to create your own functions.
- To keep the script from being executed when the page loads, you can put it into a function.
- A function will be executed by a call to the function.
- You may call a function from anywhere within a page.

Basic Statement and Function

- **echo statement**

Outputs text or variable onto the page. The most important function you will use in PHP!

IMPORTANT: If you don't output anything, then nothing will be shown on Web browser!

(Example 1)

```
<HTML>
<BODY>
<?php echo "<H1>Hello, PHP!</H1>"; ?>
</BODY>
</HTML>
```

You should output HTML tags

(Example 2)

```
<HTML><BODY>
<?php
$age = 35;
echo "<H1>I am $age years old</H1>\n";
?>
</BODY></HTML>
```

variable

"\n" means new line

PHP Functions

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

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

- Give the function a name that reflects what the function does.
- The function name can start with a letter or underscore (not a number).
- Better to declare before being called to avoid issues.

PHP Functions

- A simple function that writes a name when it is called:

```
<html>
<body>

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

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

</body>
</html>
```

PHP Functions - Parameters

- Adding parameters.
- To add more functionality to a function, we can add parameters. A parameter is just like a variable.
- Parameters are specified after the function name, inside the parentheses.

PHP Functions - Parameters

The following example will write different first names, but equal last name:

```
<html>
<body>

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

echo "My name is ";
writeName("Kai Jim");
echo "My sister's name is ";
writeName("Hege");
echo "My brother's name is ";
writeName("Stale");
?>

</body>
</html>
```

PHP Functions - Parameters

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes.
My brother's name is Stale Refsnes.
```

PHP Functions - Parameters

```
<html>
<body>

<?php
function writeName($fname,$punctuation)
{
echo $fname . " Refsnes" . $punctuation . "<br />";
}

echo "My name is ";
writeName("Kai Jim",".");
echo "My sister's name is ";
writeName("Hege","!");
echo "My brother's name is ";
writeName("Ståle","?");
?>

</body>
</html>
```

This example adds different punctuation.

PHP Functions - Parameters

Output:

```
My name is Kai Jim Refsnes.
My sister's name is Hege Refsnes!
My brother's name is Ståle Refsnes?
```

PHP Function Arguments

- Information can be passed to functions through arguments. An argument is just like a variable.
- Arguments are specified after the function name, inside the parentheses.You can add as many
- arguments as you want, just separate them with a comma.
- The following example has a function with one argument ($fname).When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

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

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

Output:

Jani Refsnes.
Hege Refsnes.
Stale Refsnes.
Kai Jim Refsnes.
Borge Refsnes.

PHP Function Arguments ...

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

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

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

Output:
Hege Refsnes. Born in 1975
Stale Refsnes. Born in 1978
Kai Jim Refsnes. Born in 1983

PHP Default Argument Value

- If we call the function setHeight() without arguments it takes the default value as argument:

```
<?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);
?>
```

Output:

```
The height is : 350
The height is : 50
The height is : 135
The height is : 80
```

PHP Functions - Returning values

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

```
<?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);
?>
```

Output:

5 + 10 = 15
7 + 13 = 20
2 + 4 = 6

Function...

- **date() function**

Get current date and time as a string according to the specified format.

```
$now = date("d/m/Y h:m:s");
echo "<P>Current date and time is $now</P>\n";
```

GET and POST Vs $_GET and $_POST

- GET - The GET method is an HTTP request method used to send data to the server by appending it to the URL as a query string.
- $_GET – is a **super global associative array** in PHP that retrieves data sent via the GET method.
- POST - POST method is an HTTP request method used to send data from a client (e.g., a web browser) to a server in the HTTP request body.
- $_POST – is a PHP **super global array** that is used to collect the data sent through the POST method.

GET

- It contains all the data passed through the URL as query parameters.
- It is an associative array where:
- The keys are the names of the form fields or query parameters.
- The values are the corresponding user inputs or parameter values.
- It is built-in and always available in PHP scripts.

PHP Forms - $_GET Function

- The built-in $_GET function is used to collect values from a form sent with 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. 100 characters).

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.w3schools.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

- The "welcome.php" file can now use the $_GET function to collect form data (the names of the form fields will automatically be the keys in the $_GET array)

```
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 100 chars.

PHP Forms - $_POST Function

- The built-in $_POST function is used to collect values from a form sent with method="post".
- Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
- Data is sent within the request body.
- 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

- When to use **method="post"**?
- Information sent from a form with the **POST** method is invisible to others and has no limits on the amount of information to send.
- However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

6.2 Getting parameter values by URL

Standard URL can contain parameters to a dynamic page by using "?" character as follows.

- http://*server_name*/*page_name*.php?*variable_name*=*value*

You can use this parameter in PHP code to change the behavior.

setoption.html (Normal HTML page to send parameter to PHP page)

```
<HTML>
<BODY>
<p><a href="getoption.php?option=1">Click here for option 1</a>
<p><a href="getoption.php?option=2">Click here for option 2</a>
</BODY>
</HTML>
```

Add name of the variable and its value to URL

getoption.php (PHP page to receive the option from above page)

```
<HTML>
<BODY>
<p>You click the option <?php echo $_GET["option"]; ?></p>
</BODY>
</HTML>
```

Name of the variable

If you want to use many parameters, you can separate each parameter by "&" character as follows.

- http://*server_name*/*page_name*.php?*variable_name*=*value*&*variable_name*=*value* …

6.3 Getting the response from user using URL parameters

JavaPHP.php

```
<HTML>
<BODY>
<P>Enter the year you born</P>
<P>Year:<INPUT type="text" id="year"></P>
<INPUT type="button" value="OK" onClick="getAge();">
<HR>
<?php
if (isset($_GET["y"])) {
      echo "Your age is ", date("Y") - $_GET["y"];
}
?>
</BODY>
<SCRIPT language="JavaScript">
function getAge() {
  location.href = "JavaPHP.php?y=" + year.value;
}
</SCRIPT>
</HTML>
```

Reference

- http://www.php.net
- http://www.phpbuilder.com/
- http://www.phpmyadmin.net/
- http://www.php.net/downloads
- http://www.php.net/manual/en/install.windows.php
- http://php.resourceindex.com/
- Httt://www.tinyurl.com/rpi123
- https://www.tutorialspoint.com/php/php_web_concepts.htm