UNIT 5

# PHP (Hypertext Preprocessor)

# Introduction

- PHP stands for **P**HP: **H**ypertext **P**reprocessor
- PHP is a server-side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software
- PHP is free to download and use
- PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures (COM and CORBA), making n-tier development a possibility for the first time.

## Syntax

- PHP code is executed on the **server**, and the plain **HTML** result is sent to the **browser**.
- A PHP scripting block always starts with **<?php and ends with ?>**. A PHP scripting block can be placed anywhere in the document.
- Each code line in PHP must end with a **semicolon**. The semicolon is a separator and is used to distinguish one set of instructions from another.
- There are two basic statements to output text with PHP: **echo** and **print**
- The file must have a **.php** extension

- There are two commenting formats in PHP:

1. **Single-line comments(# or //)**
2. **Multi-lines comments(/*----*/)**

- Eg: creating a variable containing a string, and a variable containing a number: <?php $txt="HelloWorld!"; $x=16; ?>

# variables

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and 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 String Variables:**

A string variable is used to store and manipulate text.

## strings

**String Variables in PHP:**

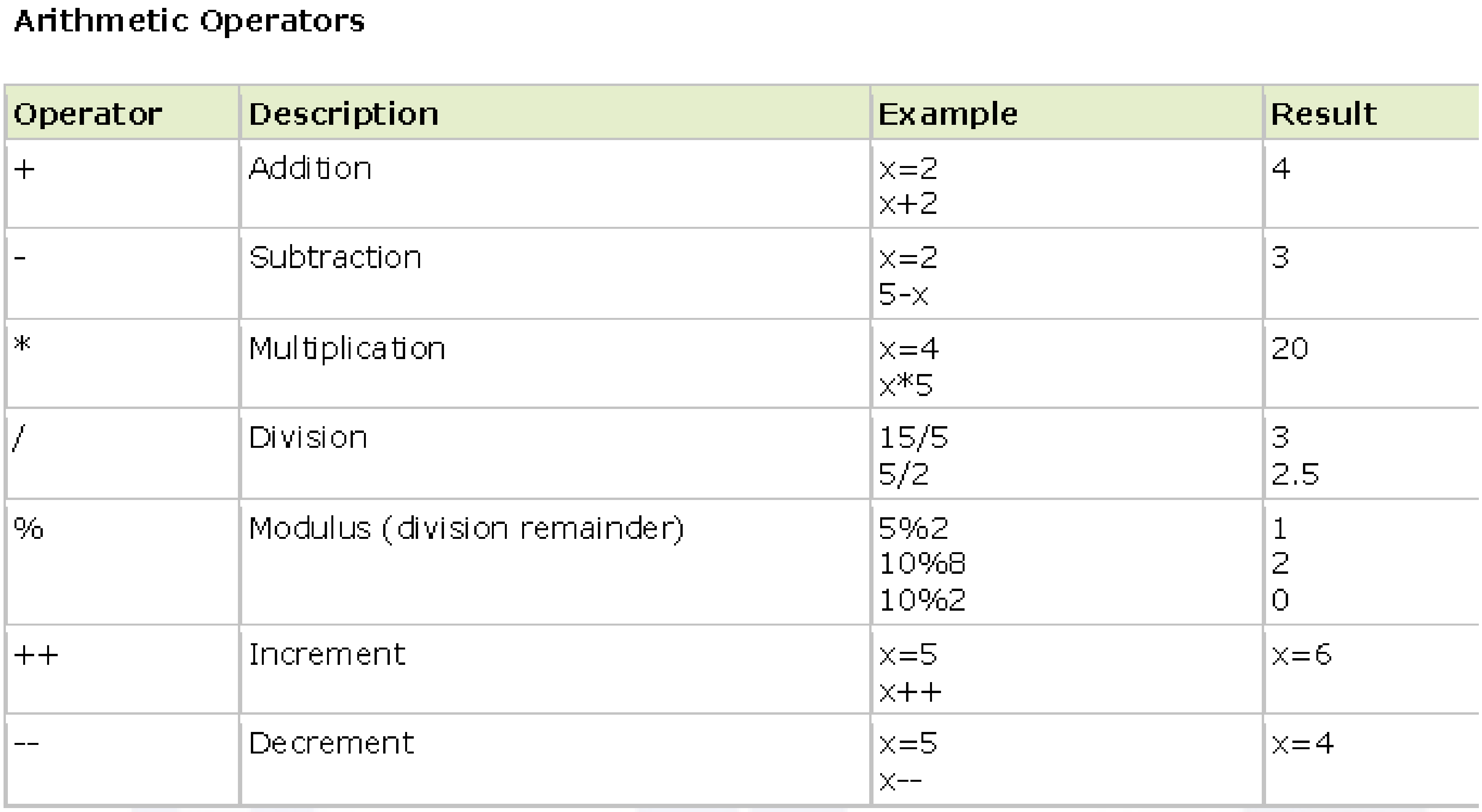
- String variables are used for values that contain characters.
- the PHP script assigns the text **"Hello World"** to a string variable called **$txt:**
- Eg: <?php $txt="Hello World"; echo $txt; ?>
- The output of the code above will be:
- Hello World

- **String** is a data type representing textual data in computer programs. Probably the single most important data type in programming.

- Since string are very important in every programming language, we will dedicate a whole chapter to them. Here we only drop a small example.

```
<?php
$a = "PHP ";
$b = 'PERL';
echo $a, $b;
echo "\n";
?>
```

# operators



**Arithmetic Operators**

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

## 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 |

## 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 |

## Logical Operators

| Operator | Description | Example |
|---|---|---|
| && | and | x=6<br>y=3<br><br>(x < 10 && y > 1) returns true |
| \|\| | or | x=6<br>y=3<br><br>(x==5 \|\| y==5) returns false |
| ! | not | x=6<br>y=3<br><br>!(x==y) returns true |

## Conditional operators:

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

## If-else

- If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.
- **Syntax**

if (*condition*)

*code to be executed if condition is true;*

else

*code to be executed if condition is false;*

## loops

- **for -** loops through a block of code a specified number of times.
- **while -** loops through a block of code if and as long as a specified condition is true.
- **do...while -** loops through a block of code once, and then repeats the loop as long as a special condition is trur.
- **foreach -** loops through a block of code for each element in an array.

- The for loop statement

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

**Syntax** for (*initialization*; *condition*; *increment*)

{

*code to be executed;*

}

## The while loop statement

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.

**Syntax** while (*condition*)

{

*code to be executed*;

}

**The do...while loop statement**

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

**Syntax** do

{

*code to be executed;*

}while (*condition*);

**The foreach loop statement**

The foreach statement is used to loop through arrays. For each pass the the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

**Syntax** foreach (*array* as *value*)

{

*code to be executed;*

}

## switch

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..else

**Syntax** switch (*expression*)

{

case *label1:*

*code to be executed if expression = label1;*

break;

case *label2:*

*code to be executed if expression = label2;*

break;

default:

*code to be executed*

*if expression is different*

*from both label1 and label2;*

}

## array

- An array stores multiple values in one single variable.
- A 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.

- 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

## Numeric Arrays

 A numeric array stores each array element with a numeric index.

 There are two methods to create a numeric array.

1. In the following example the index are automatically assigned (the index starts at 0):

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

2. In the following example we assign the index manually:

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

## Associative Arrays

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

**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.

**Example**

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" ) );

## funtions

A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

```
<html>
<head>
<title>Writing PHP Function</title>
</head>
<body>
<?php
/* Defining a PHP Function */
function writeMessage()
{
echo "You are really a nice person, Have a nice time!";
}
/* Calling a PHP Function */
writeMessage();
?>
</body>
</html>
```

## form

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

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```

The output could be something like this:

```
Welcome John
Your email address is john.doe@example.com
```

# mail

The mail() function allows you to send emails directly from a script.

Syntax

mail(*to,subject,message,headers,parameters*);

Example

Send a simple email:

```
<?php
// the message
$msg = "First line of text\nSecond line of text";

// use wordwrap() if lines are longer than 70 characters
$msg = wordwrap($msg,70);

// send email
mail("someone@example.com","My subject",$msg);
?>
```

# File upload

**Configure The "php.ini" File**

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the file_uploads directive, and set it to On:

```
file_uploads = On
```

**Create The HTML Form**

Next, create an HTML form that allow users to choose the image file they want to upload:

```
<!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>
```

## Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

```
<?php
$target_dir = "uploads/";
$target_file = $target_dir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$imageFileType = pathinfo($target_file,PATHINFO_EXTENSION);
// Check if image file is a actual image or fake image
if(isset($_POST["submit"])) {
    $check = getimagesize($_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;
    }
}
?>
```

# session

Start a PHP Session

A session is started with the session_start() function.

Session variables are set with the PHP global variable: $_SESSION.

```
<?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>
```

to show all the session variable values for a user session is to run the following code:

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
print_r($_SESSION);
?>

</body>
</html>
```

## error

**Using the die() function**

```
<?php
if(!file_exists("welcome.txt")) {
  die("File not found");
} else {
  $file=fopen("welcome.txt","r");
}
?>
```

Now if the file does not exist you get an error like this:

```
File not found
```

**Custom error handling**

```
function customError($errno, $errstr) {
  echo "<b>Error:</b> [$errno] $errstr<br>";
  echo "Ending Script";
  die();
}
```

The code above is a simple error handling function. When it is triggered, it gets the error level and an error message. It then outputs the error level and message and terminates the script.

## exception

- Exception handling is used to change the normal flow of the code execution if a specified error (exceptional) condition occurs. This condition is called an exception.

  This is what normally happens when an exception is triggered:
- The current code state is saved
- The code execution will switch to a predefined (custom) exception handler function
- Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

# filter

- PHP filters are used to validate and sanitize external input.

- The PHP filter extension has many of the functions needed for checking user input, and is designed to make data validation easier and quicker.
- The filter_list() function can be used to list what the PHP filter extension offers:
- Example

```
<table>
 <tr>
  <td>Filter Name</td>
  <td>Filter ID</td>
 </tr>
 <?php
 foreach (filter_list() as $id =>$filter) {
    echo '<tr><td>' . $filter . '</td><td>' . filter_id($filter) . '</td></tr>';
 }
 ?>
</table>
```

## PHP ODBC

Open DataBase Connectivity is an Application Programming Interface (API) that allows a programmer to abstract a program from a database. When writing code to interact with a database, you have to add code that talks to a particular database using a proprietary API.

```
<? # connect to a DSN "mydb" with a user and password "marin"
$connect = odbc_connect("mydb", "marin", "marin");# query the
users table for name and surname $query = "SELECT name,
surname FROM users";
# perform the query $result = odbc_exec($connect, $query);
# fetch the data from the database
while(odbc_fetch_row($result)){ $name = odbc_result($result, 1);
$surname = odbc_result($result, 2); print("$name $surname\n");
}
# close the connection odbc_close($connect); ?>
```

## Introduction to COM/DCOM/CORBA

- COM is the fundamental "object model" on which ActiveX Controls and OLE are built. COM allows an object to expose its functionality to other components and to host applications. It defines both how the object exposes itself and how this exposure works across processes and across networks. COM also defines the object's life cycle.

- Fundamental to COM are these concepts:

Interfaces — the mechanism through which an object exposes its functionality.

IUnknown — the basic interface on which all others are based. It implements the reference counting and interface querying mechanisms running through COM.

Reference counting — the technique by which an object (or, strictly, an interface) decides when it is no longer being used and is therefore free to remove itself.

QueryInterface — the method used to query an object for a given interface.

Marshaling — the mechanism that enables objects to be used across thread, process, and network boundaries, allowing for location independence.

Aggregation — a way in which one object can make use of another.

# DCOM

- DCOM is an acronym that stands for Distributed Component Object Model.
- It is Microsoft's solution for distributed computing.
- It allows one client application to remotely start a DCOM server object on another machine and invoke its methods. So, functionally it is similar to CORBA and RMI.
- Unlike RMI however, which is Java dependent, DCOM is language and platform independent - as is CORBA.
- The main difference between DCOM and CORBA lies in the implementation.
- DCOM is a *binary* standard whereas CORBA is only a *specification*: it defines the interfaces for how clients can use the objects but does not define the implementation itself. So different vendors are free to implement CORBA ORBs in many different ways, which makes CORBA possibly more flexible.

## CORBA

- The Common Object Request Broker Architecture (CORBA) is a standard developed by the Object Management Group (OMG) to provide interoperability among distributed objects.
- CORBA is the world's leading middleware solution enabling the exchange of information, independent of hardware platforms, programming languages, and operating systems.
- CORBA is essentially a design specification for an Object Request Broker (ORB), where an ORB provides the mechanism required for distributed objects to communicate with one another, whether locally or on remote devices, written in different languages, or at different locations on a network.

- The CORBA Interface Definition Language, or IDL, allows the development of language and location-independent interfaces to distributed objects. Using CORBA, application components can communicate with one another no matter where they are located, or who has designed them. CORBA provides the location transparency to be able to execute these applications.

- CORBA is often described as a "software bus" because it is a software-based communications interface through which objects are located and accessed. The illustration below identifies the primary components seen within a CORBA implementation.