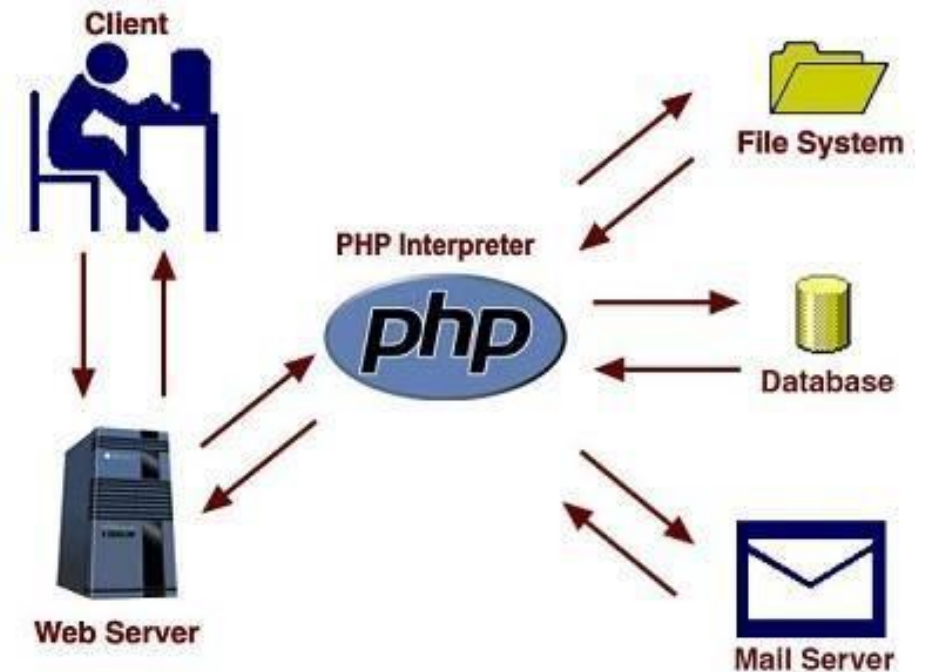
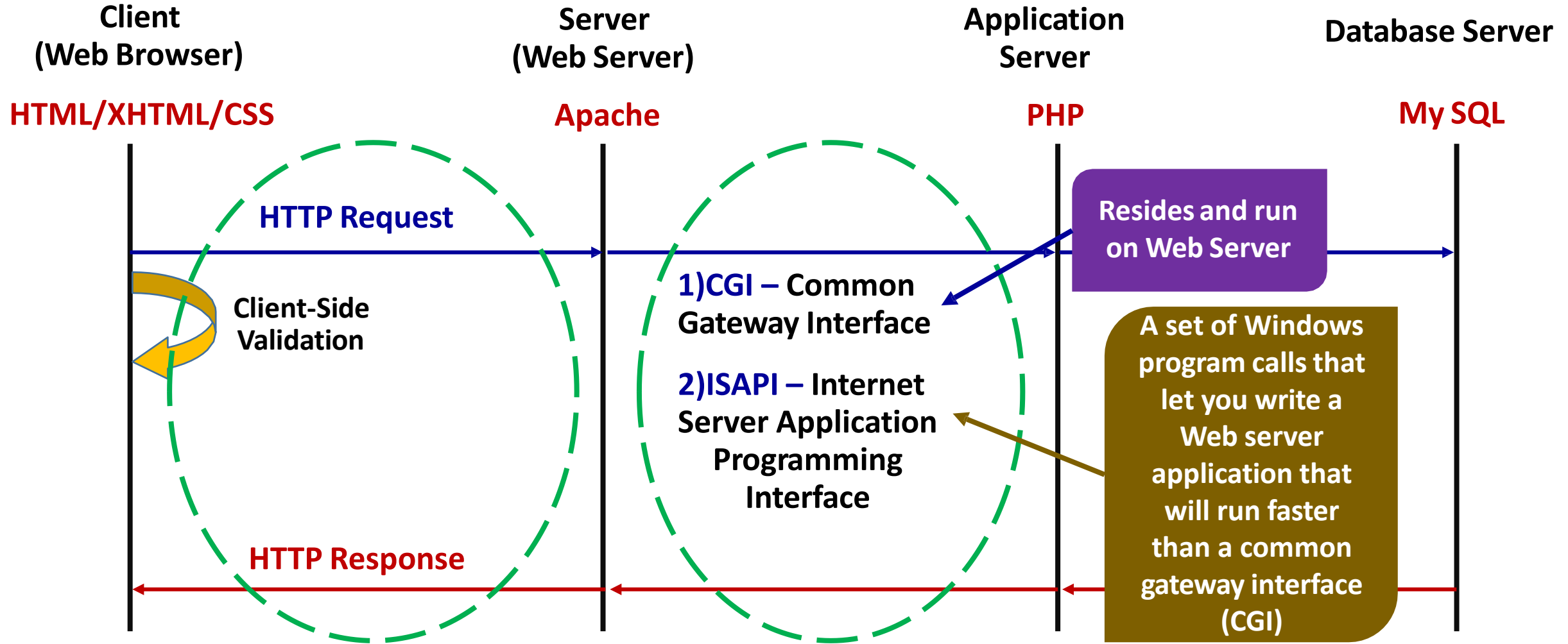


Server-side Scripting with PHP



The Architecture of the Internet



Activity 1

**Explain the difference between
a Client-Side Scripting and
a Server-Side Scripting**

Difference between a Client-Side Scripting and a Server-Side Scripting

- **Client-side Environment**

- The client-side environment **used to run scripts** is usually a **browser**.
- The **processing takes place on the end users computer**.
- The **source code is transferred from the web server to the users computer over the internet** and **run directly in the browser**.
- The scripting language needs to be **enabled** on the client computer.
- Sometimes if a user is conscious of **security risks** they may switch the scripting facility off.
- When this is the case a message usually pops up to alert the user when script is attempting to run.

Difference between a Client-Side Scripting and a Server-Side Scripting

- **Server-side Environment**
 - The **server-side environment** that runs a scripting language in a web server.
 - A user's request is fulfilled by **running a script directly on the web server** to generate **dynamic HTML pages**. This HTML is then sent to the client browser.
 - It is usually used to provide interactive web sites that interface to databases or other data stores on the server.
 - This is different from client-side scripting where scripts are run by the viewing web browser, usually in JavaScript.
 - The primary advantage to server-side scripting is the **ability to highly customize the response based on the user's requirements, access rights, or queries into data stores**.

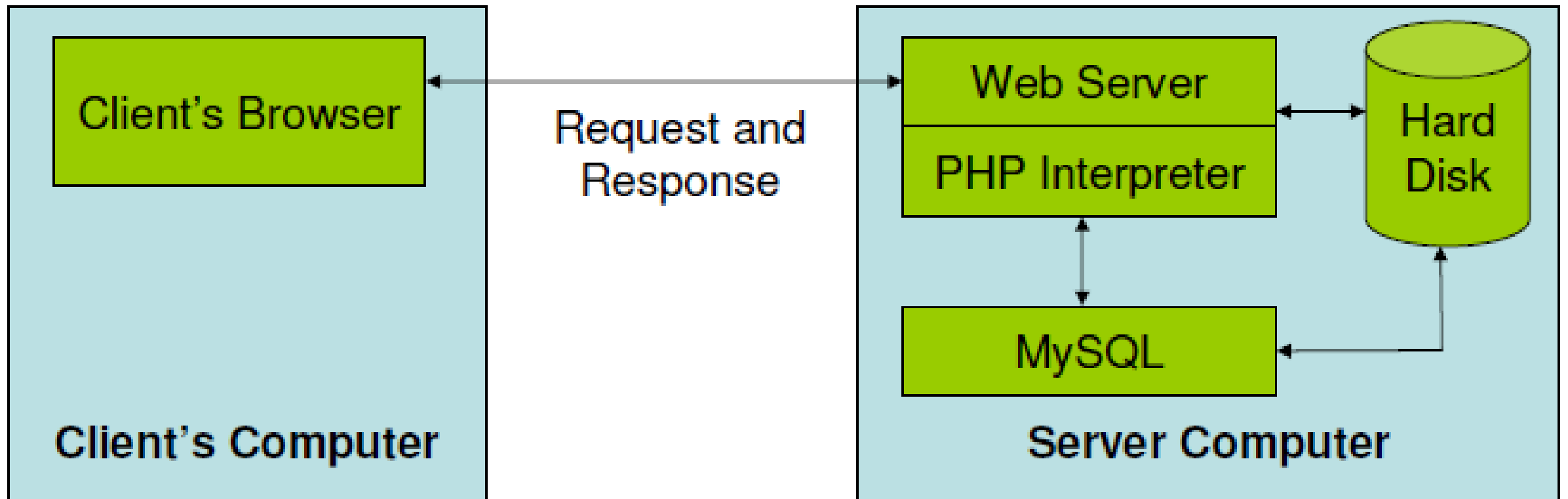
Some Server-side Scripting Technologies

- **ASP** - Microsoft designed, primarily Windows based, solution allowing various languages (generally VBscript) inside a HTML-like outer page
- **ASP.NET** - part of Microsoft's .NET platform and is the successor to ASP
- **JSP** - a Java-based system for embedding Java-related code in HTML pages
- **PHP** - open source solution based on including code in its own language into an HTML page
- **Ruby on Rails** - a free web application framework that aims to increase the speed and ease with databasedriven Web sites creation

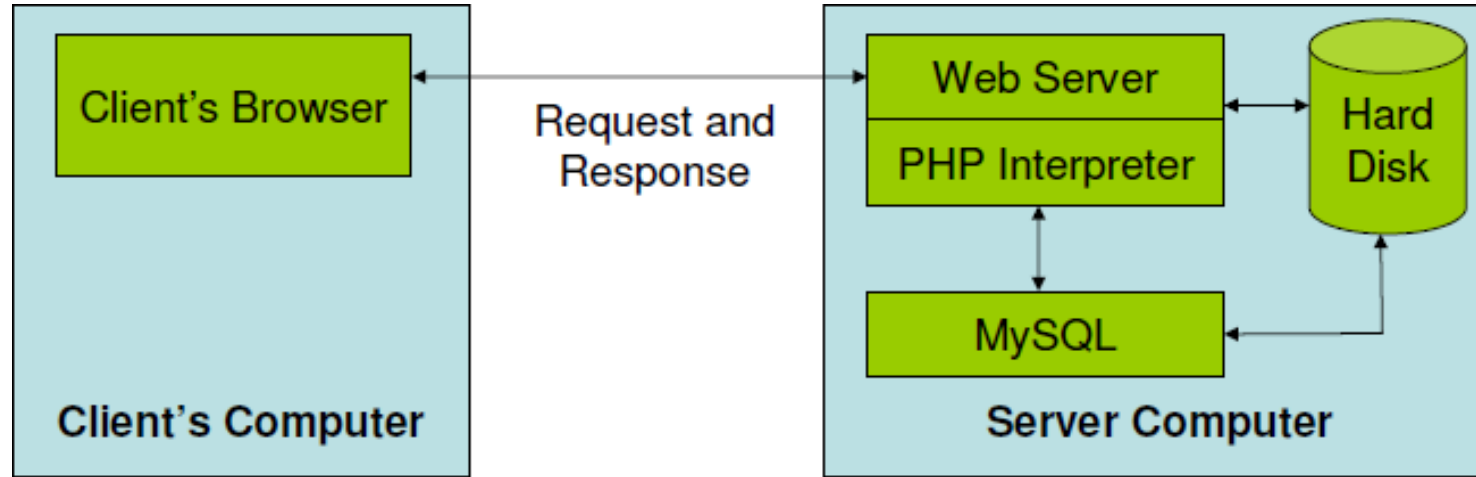
Why PHP?

- PHP **runs on different operating system** platforms (Windows, Linux, Unix and so on) seamlessly
- PHP is **compatible with almost all Web servers** used today (Apache, IIS and so on)
- PHP is **free to download** from the official website
- PHP is **easy to learn and runs efficiently** on any compatible Web server
- **Extensive help is available** through different sources (books, Internet community and so on)

Server-side Processing

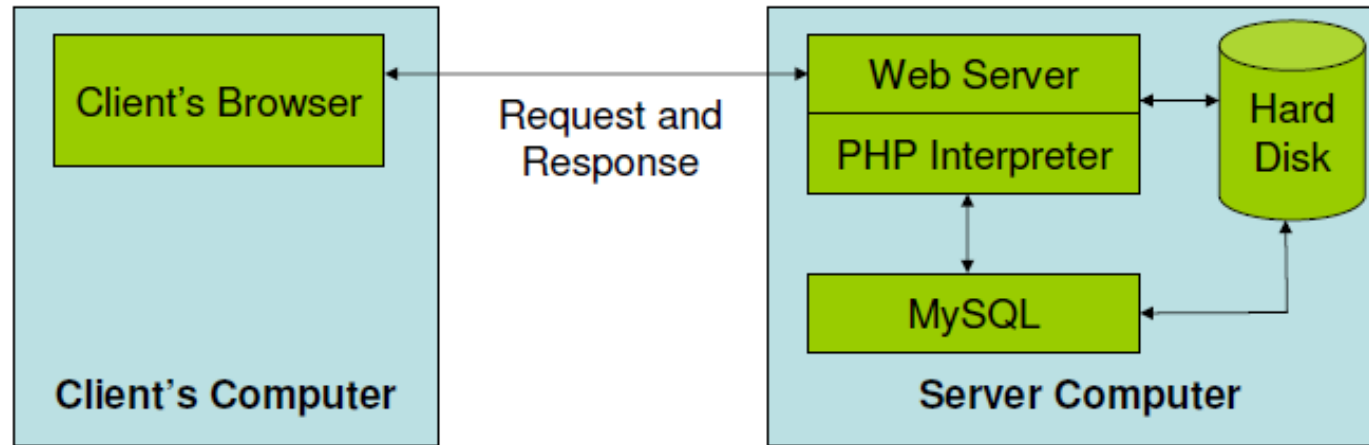


Server-side Processing



- You enter the address of the Web page in the browser's address bar, e.g. <http://www.horizoncampus.edu.lk/vle/login.php>
- Your browser sends the request to the **host (server)** requesting the Web page
- The Web server **process the request and reads the requested file (login.php) from the hard drive**

Server-side Processing



- **Web server detects that the file is a PHP file** and thus it asks the **PHP interpreter to process the file**
- PHP interpreter, **while executing the file, finds that the page has calls to MySQL database**, and as a result, **PHP interpreter asks MySQL to process the queries**
- PHP interpreter completes the execution of the file with the **results returned from MySQL**
- **Web server returns the resulting HTML text** returned by the PHP interpreter to **your Web browser**
- Your Web browser then renders a Web page based on the HTML text returned by the Web server

PHP History

- PHP was originally conceived and created by **Rasmus Lerdorf** in **1995** and he dubbed it **Personal Home Page**.
- He originally used PHP scripts to know how many visitors were reading his online resume.
- In 1997 **Andi Gutmans** and **Zeev Suraski** did a **complete rewrite of PHP** by including powerful extensions to the language and released the version 2.0 of **Personal Home Page – From Interpreter (PHP/FI)**.
- Version 4.0 was released in May 2000 with the Zend engine, and it had major differences from its previous versions in all aspects.
- PHP 5.0 (released in July 2004) was a big step forward for the language and the improvements included object-oriented support, MySQL database extension, SQLite and so on.

Solution Stacks

- A single s/w that concerns as an installer
- Running the installer will install and configure the different s/w in minimal amount of time and effort
- Benefits of using this is the developer can install the environment quickly and start programming

Solution Stacks

- **LAMP – Linux Apache MySQL PHP**
- **WAMP – Windows Apache MySQL PHP**
- **XAMP – Apache MySQL PHP**
- **XAMPP – Apache MySQL PHP PERL**
- **PHP Dev – Apache MySQL PHP**

WAMP Environment

- Examples of this document were tested under WAMP5 on Windows XP
- WAMP stands for Windows – Apache – MySQL – PHP
- WAMP5 solution included the following software:
 - Apache 2.0.58
 - PHP 5.1.4
 - MySQL 5.0.22
 - PHPmyadmin 2.8.1
 - SQLitemanager 1.2.0
 - Wampserver service manager

Placing a PHP Script within a Document

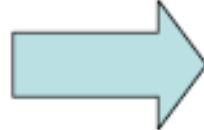
- All PHP code needs to be enclosed within beginning and ending tags, i.e. `<?php` and `?>`
- E.g.

```
<html>
<head>
<title>First PHP</title>
<meta http-equiv="Content-Type" content=
"text/html; charset=iso-8859-1">
</head>

<body>
<?php
print "<h1>Hello World.</h1>";
?>
</body>
</html>
```

At the server

To a client



```
<html>
<head>
<title>First PHP</title>
<meta http-equiv="Content-Type" content=
"text/html; charset=iso-8859-1">
</head>

<body>
<h1>Hello World.</h1></body>
</html>
```



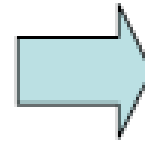
Hello World.

Note: `print` statement in PHP delivers text to a browser

PHP Blocks of Code

- PHP processing can be turned on and off within a script by closing and reopening the PHP tags.

```
<body>
<?php
print "<h1>Hello World.</h1>";
?>
<p>This is my first PHP document</p>
<?php
print "<h2>This is another PHP statement.</h2>";
?>
</body>
```



Hello World.

This is my first PHP document

This is another PHP statement.

Differences between Print and Echo

Print	Echo
print is a function - only one parameter but parenthesis are optional so it can look like a language construct	echo is a language construct and can be treated like a function with one parameter. Without parenthesis, it accepts multiple parameters separated by commas.
Print will return TRUE if it is successful in printing something	Echo will not return any value

Output

- **echo** is a language construct - can be treated like a function with one parameter. **Without parenthesis, it accepts multiple parameters.**
- **print** is a function - **only one parameter** but **parenthesis are optional** so it can look like a language construct

Comments

- Comment tags are embedded with code to describe what the script is doing. PHP provides three different ways to set comments:

- `//` at the beginning of each line (i.e. single line comment)

```
// Declaring a variable named $error  
$error = "Nothing";
```

- `/*` and `*/` to comment out several lines (i.e. multi line comment)

```
/* Declaring a variable named $error  
   and initializing the variable */  
$error = "Nothing";
```

- `#` at the beginning of each line (i.e. single line comment)

```
# Declaring a variable named $error  
$error = "Nothing";
```

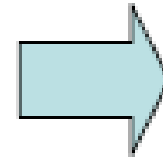
Variables in PHP

- All variables in PHP starts with a **dollar (\$) sign**.
- PHP is "loosely typed," meaning that the type of information to be held in a particular variable can change on the fly (similar to JavaScript).
- Variable names can be any length and contain numbers, letters, or underscores.
- **The starting letter of a variable must be a letter or an underscore.**
- Some examples of valid variable names:
 - `$name`
 - `$_code_ptr`
 - `$first name`
 - `$log99`

Variables in PHP

- PHP does not require variables to be declared before being initialized
- PHP variables are case sensitive. E.g.

```
<?php  
$a = "Lakmini";  
$A = 35;  
echo $a;  
echo $A;  
?>
```



Lakmini35

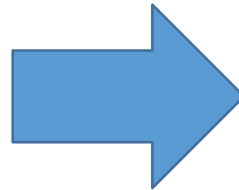
Note: `echo` statement (like `print`) in PHP delivers text to a browser

Data Types in PHP

Data Type	Description
Boolean	Has only two literal values: 'true' and 'false'
Integers	Stored as signed integers with 32 bits. Integer literals have 3 forms: decimal, octal (prefixed with '0'), and hexadecimal (prefixed with '0x')
Float	Stored as IEEE double floating point numbers with 64 bits
Strings	A sequence of 8-bit characters. String literals have 3 forms: single quoted, double quoted, and here-doc syntax
Arrays	Stores an ordered map of pairs of keys and values. This is quite different than the array type used in other languages
Objects	stores an instance of a class
Resource	represents an external resource like a file, or a database connection

Example

```
<body>
<?php
$b = true; // Assigns a Boolean value
print "b = $b<br>";
$idec = 34; // Assigns a decimal value
print "idec = $idec<br>";
$ioct = 034; // Assigns an octal value
print "ioct = $ioct<br>";
$ihex = 0xAF; // Assigns a hexadecimal value
print "ihex = $ihex<br>";
$f = 7.326; // Assigns a float value
print "f = $f<br>";
$strsq = 'Hello'; // Assigns a single-quoted string
print "strsq = $strsq<br>";
$strdq = "Hello"; // Assigns a double-quoted string
print "strdq = $strdq<br>";
// The following is a here-doc example
print <<<STR_HD
Here-docs enable you to embed large pieces of text in
your scripts, which may include lots of "double quotes"
and 'single quotes'.
STR_HD;
?>
</body>
```



```
b = 1
idec = 34
ioct = 28
ihex = 175
f = 7.326
strsq = Hello
strdq = Hello
Here-docs enable you to embed large pieces of text in your
scripts, which may include lots of "double quotes" and 'single
quotes'.
```

PHP Expressions

- An expression is any valid set of literals, variables, operators, and expressions that evaluates to a single value.
- The value may be a number, a string, or a logical value.
- There are two types of expressions:
 - those that assign a value to a variable

E.g. `$x = 3;`

- those that simply have a value

E.g. `3 + 8` or `($a+$b)/$c`

Operators

- PHP supports the standard operators for numbers and strings. The following is an outline of PHP operators:
 - Assignment operators
 - Arithmetic operators
 - Comparison operators
 - Logical operators
 - Bitwise operators
 - String operators

Assignment Operators

- An assignment operator assigns a value to its left operand based on the value of its right operand.
 - E.g. `$x = 23;`

Operator	Use	Meaning
<code>=</code>	Assignment	<code>\$x=\$y</code>
<code>+=</code>	Increment assignment	<code>\$x=\$x+\$y</code>
<code>-=</code>	Decrement assignment	<code>\$x=\$x-\$y</code>
<code>*=</code>	Multiplication assignment	<code>\$x=\$x*\$y</code>
<code>/=</code>	Division assignment	<code>\$x=\$x/\$y</code>
<code>%=</code>	Modulus assignment	<code>\$x=\$x%\$y</code>

Arithmetic Operators

- Arithmetic operators take numerical values (either literals or variables) as their operands and return a single numerical value.

Operator	Description	Example
+	Addition	4+5 returns 9
-	Subtraction	5-2 returns 3
*	Multiplication	5*4 returns 20
/	Division	5/2 returns 2.5
%	Modulus (division remainder)	5%2 returns 1 10%2 returns 0
++	Increment	\$x++ means \$x = \$x + 1
--	Decrement	\$x-- means \$x = \$x - 1

Comparison Operators

- A comparison operator compares its operands and returns a logical value based on whether the comparison is true or not.

Operator	Description	Example
<code>==</code>	Is equal to	<code>5==8</code> returns false
<code>===</code>	Exactly equal to (checks for both value and type)	If <code>\$x=5</code> and <code>\$y="5"</code> , <code>\$x==\$y</code> returns true <code>\$x=== \$y</code> returns false
<code>!=</code>	Is not equal to	<code>5!=8</code> returns true
<code>></code>	Is greater than	<code>5>8</code> returns false
<code><</code>	Is less than	<code>5<8</code> returns true
<code>>=</code>	Is greater than or equal to	<code>5>=8</code> returns false
<code><=</code>	Is less than or equal to	<code>5<=8</code> returns true

Logical Operators

- Logical operators take Boolean (logical) values as operands and return a Boolean value.
- Let \$x = 6 and \$y = 3,

Operator	Description	Example
&& or and	And	<code>(\$x<10)&&(\$y>1)</code> returns true
or or	Or	<code>(\$x==5) (\$y==5)</code> returns false <code>(\$x==6) (\$y==3)</code> returns true
!	Not	<code>!(\$x==\$y)</code> returns true
xor	Xor	<code>(\$x==6)xor(\$y==3)</code> returns false

Bitwise Operators

- Bitwise operators treat their operands as a set of bits (zeros and ones), rather than as decimal, hexadecimal, or octal numbers.

Operator	Use	Example
<code>&</code>	Bitwise AND	<code>9&5</code> returns 1 (1001 AND 0101 = 0001)
<code> </code>	Bitwise OR	<code>9 5</code> returns 13 (1001 OR 0101 = 1101)
<code>^</code>	Bitwise XOR	<code>9^5</code> returns 12 (1001 XOR 0101 = 1100)
<code>~</code>	Bitwise NOT	<code>~4294967295</code> returns 0 (32 zeros)
<code><<</code>	Left shift	<code>5<<2</code> returns 20 (00101 becomes 10100)
<code>>></code>	Right shift	<code>13>>2</code> returns 3 (1101 becomes 0011)

String Operators

- PHP has two string operators:

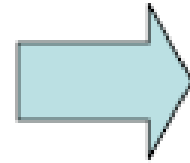
Operator	Use	Example
.	Concatenation	<pre>\$a = "Hello "; \$b = \$a . "World!";</pre> assigns "Hello World" to \$b
.=	Concatenation and assignment	<pre>\$a = "Hello "; \$a .= "World!";</pre> assigns "Hello World" to \$a

Do... While

- The do...while loop executes one or more lines of code as long as a specified condition remains true.

– E.g.

```
$i = 0;  
do {  
    print($i);  
    $i++;  
} while ($i < 10);
```



0123456789

- Note: Due to the expression being evaluated at the end of the structure, statement(s) in the do...while loop are executed at least once, even if the condition is false.

While

- The while loop executes one or more lines of code while specified expression remains true.
- – E.g.

```
$i = 0;  
while ($i<10) {  
    print($i);  
    $i++;  
};
```



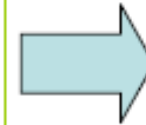
0123456789

- Note: Because the expression is evaluated at the beginning of the loop, the statement(s) will not be executed if the expression is false at the beginning.

If...Else

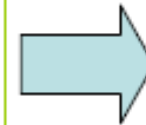
- The if and if...else constructs execute a block of code depending on the evaluation (true or false) of an expression.

```
$i = 1;  
if ($i == 1) {  
    print("One");  
}
```



One

```
$i = 2;  
if ($i == 1) {  
    print("One");  
} elseif ($i == 2) {  
    print("Two");  
} else {  
    print("Other");  
}
```

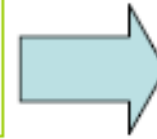


Two

Conditional Operator

- PHP also contains a conditional operator that assigns a value to a variable based on some condition.
- E.g.

```
$x = 2;  
$y = ($x==1)?"One":"Other";  
print($y);
```



Other

Similar to:

```
$x = 1;  
if ($x==1)  
    $y = "One";  
else  
    $y = "Other";  
print($y);
```

Switch

- The switch construct executes specific block(s) of code based on the value of a particular expression.
 - Note: Use break statement to prevent the code from running into the next case automatically.
 - E.g.

```
$day = "Mon";  
switch ($day) {  
    case "Mon": {  
        print("Mon");  
        break; }  
    case "Tue": {  
        print("Tue");  
        break; }  
    default: {  
        print("Other"); }  
}
```

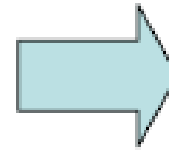


Break and Continue

- The break command will break the loop and continue executing the code that follows after the loop (if any).

– E.g.

```
for ($i=1; $i<10; $i++) {  
    if ($i==3) break;  
    print($i);  
}
```

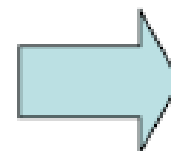


12

- The continue command will break the current loop and continue with the next value.

– E.g.

```
for ($i=1; $i<10; $i++) {  
    if ($i==3) continue;  
    print($i);  
}
```



12456789

Arrays in PHP

- An array in PHP is a collection of key/value pairs, which maps keys (indexes) to values.
- Array indexes can be either integers or strings whereas values can be of any type.

0	1	2
Monday	Tuesday	Wednesday

```
<?php
$days1 = array("Monday","Tuesday","Wednesday");
print "<br>Printing array 1 elements:<br>";
print $days1[0]; print "<br>";
print $days1[1]; print "<br>";
print $days1[2]; print "<br>";

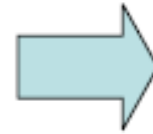
$days2 = array("M"=>"Monday","T"=>"Tuesday","W"=>"Wednesday");
print "<br>Printing array 2 elements:<br>";
print $days2["T"]; print "<br>";
print $days2["W"]; print "<br>";
print $days2["M"]; print "<br>";
?>
```

For and Foreach

- The for loop executes statement(s) a specified number of times governed by two expressions and a condition.

– E.g.

```
for ($i=1; $i<10; $i++) {  
    print($i);  
}
```



123456789

- The foreach allows an easy way to iterate over an array.

– E.g.

```
$days = array("Mon", "Tue", "Wed");  
foreach ($days as $value) {  
    print $value . "<br>";  
}
```

Functions in PHP

- A function in PHP is a set of statements that performs a specific task.
- A function will be executed by an event or by a call to that function.

– E.g.

```
$length = strlen("Horizon");
```

```
/* strlen is a standard PHP function that returns the length of a  
string*/
```

- A function can be built-in or user-defined
- PHP supports over 140 categories of built-in functions and many of these are available within the PHP core.

Built-in Functions

- E.g.
 - `print()` function basically accepts a string as an argument and outputs it to a browser. It works with or without parentheses

```
print "Hello World";  
print("My first PHP script");
```

- `include()` function allows embedding one file inside another

```
include "header.php";
```

- `header()` function allows sending a raw HTTP header to the client browser. E.g. redirecting a user to another URL

```
header("Location:http://www.bit.lk/login.php");
```

User-defined Functions

- The syntax for defining a function in PHP is as follows:

```
function <function_name>([<arg1>, <arg2>, <arg3>, ...])  
{  
    list of statements;  
    return(<value>);  
}
```

- Functions are generally replaced by their value and it's value is determined by the value supplied to the return construct.
- The function_name is the name of the function and is used when calling the function.
- When calling a function, the argument list should be supplied correctly (should match with the definition).

Example

```
<body>
<?php
// Function definition
function myName($n) {
    if ($n==1)
        return 'N.D. Wickramanayaka';
    else
        return 'Nanda Deepal Wickramanayaka';
}

// Function calls
print 'I am '.myName(1).'
```



I am N.D. Wickramanayaka
© My name is Nanda Deepal Wickramanayaka

Objects in PHP

- PHP version 5.0 added more object oriented support, placing PHP more inline with other modern programming languages.
- An object is a collection of properties (data) and methods (functions).
- A class is a template for an object and describes what methods and properties an object of this type will have.
- Class definitions in PHP begin with the class keyword and enclose the class definition within curly braces:

```
class <class_name> {  
    // List of methods  
    // List of properties  
}
```

Constructors and Destructors

- Instances of classes are created using the `new` keyword.
 - E.g. `newObject = new myClass();`
- During the `new` call, a new object is allocated with its own copies of properties defined in the requested class, and then the constructor of the object is called in case one was defined.
- An object's constructor is typically used to initialize the object's properties, and it is a method with the name `__construct()`
- In addition, a destructor function is called when the object is being destroyed to perform tasks when discarding an object.

Example

```
<?php
class Student {
    // Methods
    function __construct($name) {
        print "***Calling the constructor**</br>";
        $this->name = $name;
    }
    function getName() {
        return $this->name;
    }
    function __destruct() {
        print "***Calling the destructor**</br>";
    }

    // Properties
    private $name;
}

// Objects
$kamal = new Student("Kamal");
print $kamal->getName()."</br>";
?>
```



```
***Calling the constructor**
Kamal
***Calling the destructor**
```

Method and Properties

- `$this` is a special variable which denotes a reference to the object itself
- By using `this` variable and the `->` notation, the object's methods and properties can be further referenced, e.g.

`$this->getName()` or `$this->name`

- The variables and other structures in the class definition can be defined as public, protected or private:
 - **public**: these members can be accessed both from outside an object (`($obj->getName())`) or inside (`$this->getName()`)
 - **protected**: these members can be accessed only from within an object's method or other class inheriting from it
 - **private**: these members can be accessed only from within an object's method

Processing HTML Form Data Using PHP

- HTML supports at least two methods to pass data from a client to a server:
 - GET method encodes the form data in the URL, making it visible in the browser address window
 - POST method encodes the form data in the body of the HTML request so that the data is not shown in the URL (secure)
- PHP supports the following functions to retrieve the data which came from a client:
 - `$_GET` to possess data from a form that uses the GET method
 - `$_POST` to possess data from a form that uses the POST method
 - `$_REQUEST` to possess data from a form that uses either method (GET or POST)

Example

```
<body>
<h3>Login Page</h3>
<?php
// Checks whether the "login" key exists in the $_POST array to
// make sure that Submit button in the form hasn't been pressed
if (!isset($_POST["login"]) || ($_POST["login"] != "Login")) {
?>
<form name="form1" method="post" action="loginform.php">
  Username: <input name="uname" type="text"><br>
  Password: <input name="passwd" type="password"><br>
  <input type="submit" name="login" value="Login">
</form>
<?php
) else (
// If the form has been submitted then display the form values
// from the $_POST array
?>
<strong>Username:</strong> <?php echo $_POST["uname"] ?><br>
<strong>Password:</strong> <?php echo $_POST["passwd"] ?>
<?php
)
?>
</body>
```

Login Page

Username:

Password:



Login Page

Username: Caldera
Password: bit2008

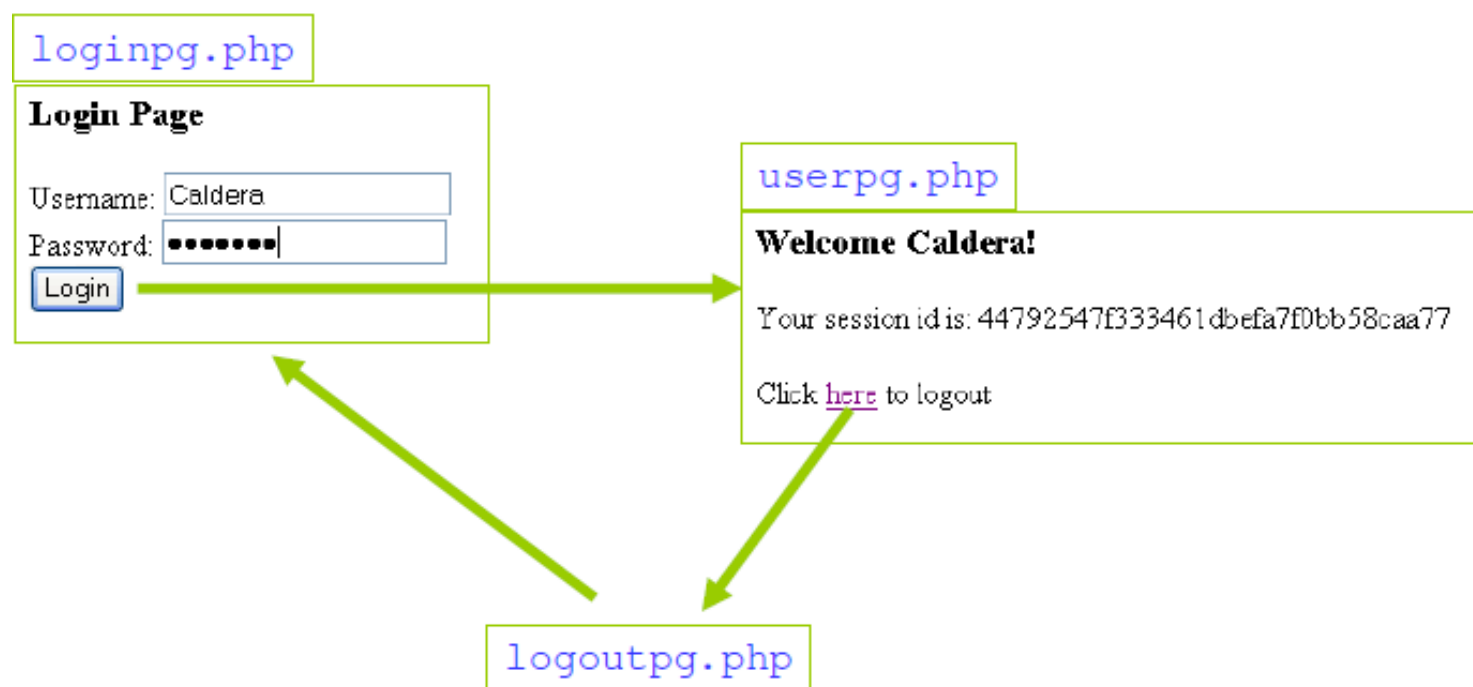
PHP Sessions

- HTTP is a stateless protocol which means that each server request knows nothing about other requests.
- However, PHP session allows an application to store information for the current “session,” which can be defined as one user being logged in to the application.
- A session is identified by a unique id, and it helps to uniquely identify each visitor who is accessing the server.
- PHP creates a session ID that is an MD5 hash of the remote IP address, the current time, and some extra randomness represented in a hexadecimal string.

Starting a Session

- The `session_start()` function must come after any session-related settings are done with `ini_set()`
 - `ini_set()` Can specify whether to pass the session ID in a cookie or to add to all URLs to navigate the web application
- `session_start()` function should be called before any output has been sent to the browser
- This function initializes the `$_SESSION` super global array where we can store session related data
- Logging out is the same as destroying the session and its associated data, and its done by using the
- `session_destroy()` function

Example



loginpg.php

```
<?php
if (isset($_SESSION['userid'])) { // If the user is already logged in, redirect to the userpage
    header('Location:userpg.php');
} elseif (isset($_POST["login"]) && ($_POST["login"]=='Login')) { // User is trying to login
    session_start(); // Start the session
    $_SESSION['userid'] = $_POST["uname"];
    header('Location:userpg.php');
}
// Otherwise display the login form
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Login Page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<h3>Login Page</h3>
<form name="form1" method="post" action="loginpg.php">
    Username: <input name="uname" type="text"> <br>
    Password: <input name="passwd" type="password"> <br>
    <input type="submit" name="login" value="login">
</form>
</body>
</html>
```

userpg.php

```
<?php
session_start(); // Start the session
if (!isset($_SESSION['userid']) || !$_SESSION['userid']) {
    // User is not logged in, therefore redirect to the login page
    header('Location:loginpg.php');
}
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>My Home Page</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
<h3>Welcome <?php echo $_SESSION['userid'] ?>!</h3>
<p>Your session id is: <?php echo session_id() ?></p>
<p>Click <a href="logoutpg.php">here</a> to logout </p>
</body>
</html>
```

logoutpg.php

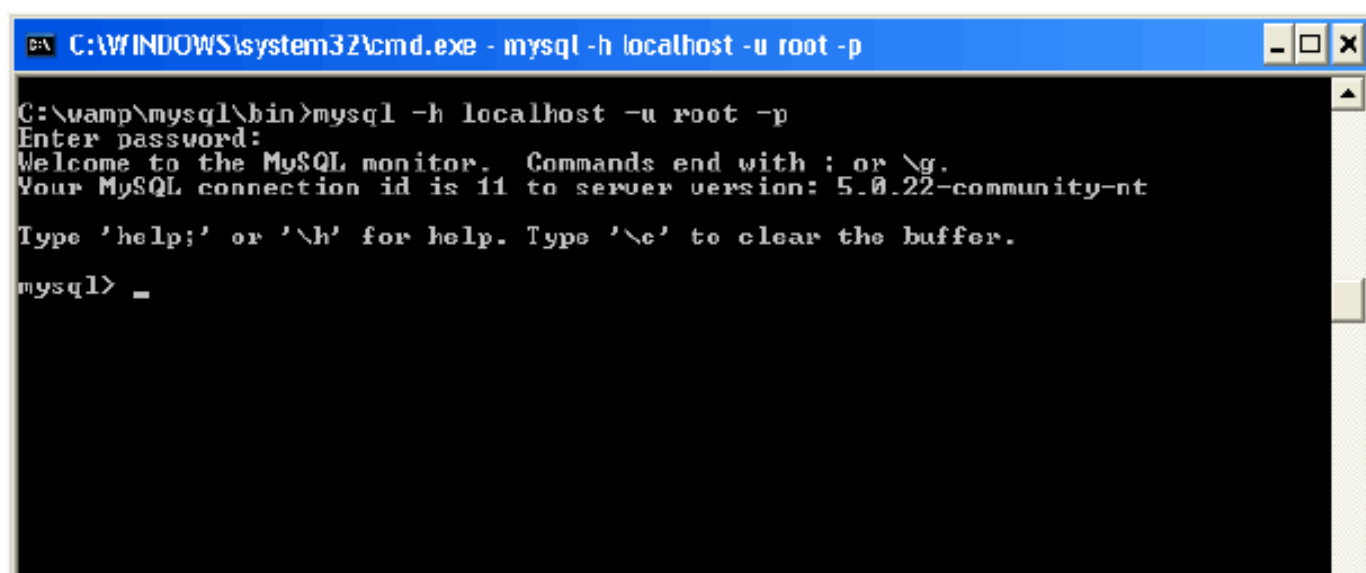
```
<?php
session_start(); // Start the session
$_SESSION = array(); // Set the $_SESSION to an empty array
session_destroy(); // Destroy the session
header('Location:loginpg.php'); // Redirect to the login page
?>
```

MySQL & PHP

- MySQL has the biggest market share of any open source database.
- MySQL and PHP have become the “bread and butter” of web application builders.
- mysqli (MySQL improved) is an extension that is bundled with PHP 5.0 to support the new features of MySQL 4.1 and 5.0 client API.

Starting with MySQL

- Before starting the MySQL client, the server has to be turned on by either starting `mysqld.exe` (Windows 9x) or running `mysqld-nt.exe` (Windows 2k/XP)
- Once the server is on, the following commands can be used to start with mysql:
 - `C:\>mysql` or
 - `C:\>mysql -h localhost -u root -p`



```
C:\WINDOWS\system32\cmd.exe - mysql -h localhost -u root -p

C:\wamp\mysql\bin>mysql -h localhost -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11 to server version: 5.0.22-community-nt

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> _
```

Creating a MySQL Database

- `CREATE database` command will create a new database called student:

```
mysql> CREATE database student;  
Query OK, 1 row affected (0.34 sec)
```

- `SHOW databases` command will show available databases in mysql:

```
mysql> SHOW databases;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| phpmyadmin |  
| student |  
| test |  
+-----+  
5 rows in set (0.09 sec)
```

Creating a New Table in a Database

- To create a table in the student database, we have to first select the database by typing `USE student`:

```
mysql> USE student;  
Database changed
```

- `CREATE TABLE` command is used to create a table named logininfo which has two fields named username and password of a student:

```
mysql> CREATE TABLE logininfo(username varchar(10) NOT NULL, password varchar(10)  
> NOT NULL, primary key(username));  
Query OK, 0 rows affected (0.52 sec)
```

- `SHOW TABLES` command will show available tables in the database:

```
mysql> SHOW tables;  
+-----+  
| Tables_in_student |  
+-----+  
| logininfo          |  
+-----+  
1 row in set (0.00 sec)
```

Inserting Data to a Table

- `DESCRIBE` command can be used to get a description of a table:

```
mysql> DESCRIBE logininfo;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| username   | varchar(10)   | NO   | PRI | NULL    |       |
| password   | varchar(10)   | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.05 sec)
```

- `INSERT` statement is used to insert a new record of data to a database:

```
mysql> INSERT INTO logininfo(username, password) VALUES('Caldera','bit2008');
Query OK, 1 row affected (0.01 sec)
```

- `SELECT` statement is used to retrieve data from a table:

```
mysql> SELECT * FROM logininfo;
+-----+-----+
| username | password |
+-----+-----+
| Caldera  | bit2008  |
+-----+-----+
1 row in set (0.02 sec)
```


Connect to MySQL Database

- `mysql_connect()` function is used to open a connection to MySQL database from PHP
- After connecting to mysql, `mysql_select_db()` function is used to select a database

```
<?php
$dbhost = 'localhost'; // MySQL server to connect
$dbuser = 'root'; // Username
$dbpassword = ''; // Password
$dbname = 'student'; // Name of the database

$conn = mysql_connect($dbhost,$dbuser,$dbpassword)
        or die("Could not connect to server!");

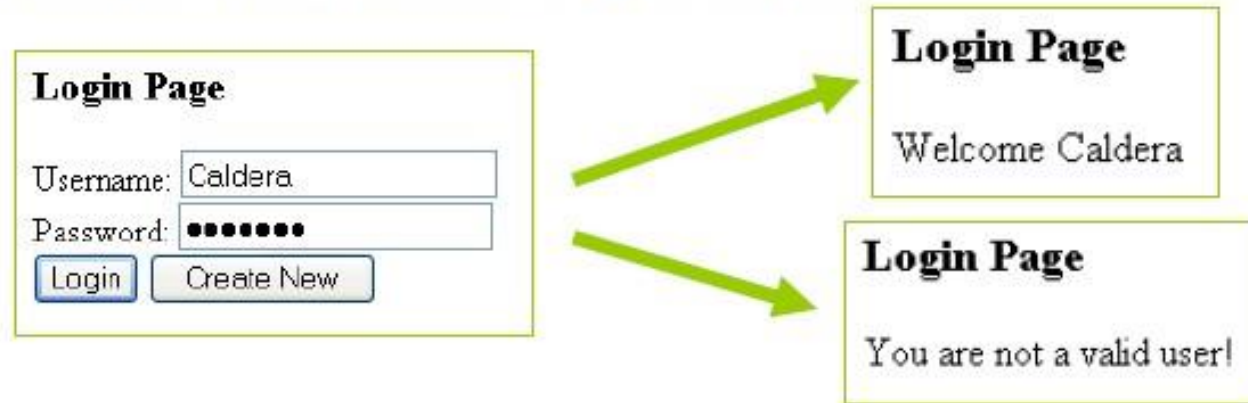
mysql_select_db($dbname)
        or die("Could not select database!");
?>
```

Processing MySQL Query

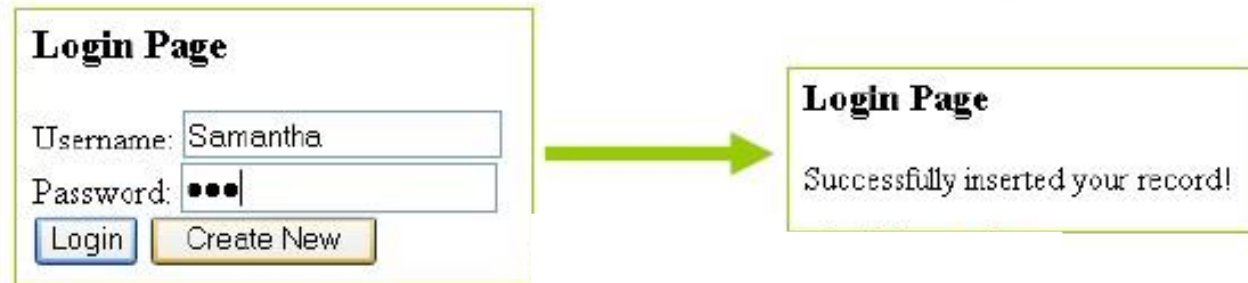
- `mysql_query()` function sends a query to the database and returns a result object
- The resulting rows of data can be fetched in numerous ways. One option is to use `mysql_fetch_array()` to fetch the resulting rows as an array.
 - The `result_type` parameter specifies whether to fetch it as an associative array, numeric array, or both.
- Once fetched a row, field data can be accessed using their name or the index.

Processing Form Data

An existing user will type username and password, and press Login button. The page will notify about the status of login to the user



A new user user will type username and password, and press Create New button. The page will notify the user about the account creation in the database



Display the Form

```
<?php
if (!isset($_POST["login"]) && !isset($_POST["create"])) {
?>
<form name="form1" method="post" action="loginform.php">
  Username: <input name="uname" type="text"><br>
  Password: <input name="passwd" type="password"><br>
  <input type="submit" name="login" value="Login">
  <input type="submit" name="create" value="Create New">
</form>
<?php
} else {
```


Processing the Form while Accessing Database

```
// PHP code to connect to the student database

$username = $_POST["uname"]; // Read form data
$password = $_POST["passwd"]; // Read form data

if (isset($_POST["login"])) { // Processing Login
    $query = "SELECT * FROM logininfo WHERE username='$username' AND password='$password'";
    $result = mysql_query($query)
        or die("Could not perform query!");
    if ($row=mysql_fetch_array($result,MYSQL_ASSOC)) // Get the first row
        echo "Welcome {$row['username']}";
    else
        echo "You are not a valid user!";
} elseif (isset($_POST["create"])) { // Procesing Create New login
    $query = "INSERT INTO logininfo(username,password) VALUES('$username','$password')";
    $result = mysql_query($query)
        or die("Failed to insert your record!");
    echo "Successfully inserted your record!";
}
mysql_close($conn); // Close the connection
}
```

?>

PHP Errors and Trouble Shooting

- An error is an unexpected, invalid program state from which it is impossible to recover unless it is debugged.
- PHP error messages can be one of the following categories:
 - Syntax or parse errors
 - Fatal errors
 - Warnings
 - Notices
 - Logical errors
 - Environmental errors
 - Runtime errors
 - Core errors

PHP Errors

- **Syntax or parse errors**

- These errors are identified when a PHP file is processed, before the PHP interpreter starts executing it.

- Common mistakes are missing semicolons (;), parenthesis (), dollar (\$) signs, curly brackets {}, misspelled keywords or improperly closed strings

- **Fatal errors**

- These errors occur when there is a severe problem with the content of code spec, such as calling a function that has not been defined.

- The interpreter simply stops running the code spec when a fatal error is encountered.

PHP Errors

- **Warnings**

- These errors are serious enough to prevent the correct execution of the program but not cause it to halt.
- E.g. invalid regular expressions or missing files

- **Notices**

- Notices will never terminate the script and it displays a message when PHP encounters something, which could be an error.
- E.g. reading a variable, which is undefined

- **Logical or semantic errors**

- These are faults in program design i.e. in the order of instructions
- These errors may cause a program to respond incorrectly to the user's request or to crash completely

PHP Errors

- Environmental errors
 - These errors crop up when external entities such as files, network connections and input data assumes a form or behave a manner unexpected by the program.
- Runtime errors
 - These errors occur during execution of the code, which are not usually programming errors but caused by factors outside PHP itself, such as disk or network operations or database calls.
- Core errors
 - Core errors are those generated by the PHP runtime core i.e. caused by an extension that failed to startup and caused PHP to abort.

Reporting Errors

- The PHP `error_reporting()` function controls the level of errors reported at runtime.
- PHP defines some constants that can be used and applied to set the value of `error_reporting()` so that only errors of certain types get reported.
- E.g.

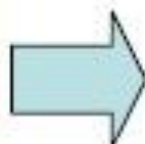
Constant	Use
<code>E_ALL</code>	For all errors
<code>E_PARSE</code>	For parse errors
<code>E_ERROR</code>	For Fatal errors
<code>E_WARNING</code>	Warnings
<code>E_NOTICE</code>	For notices

Example

Output

```
<?php
error_reporting(0); // No error reporting

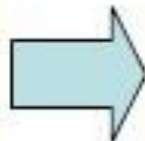
echo "Line before calling function Validate()<br>";
// Calling an undefined function to see fatal error
Validate();
echo "Line after calling function Validate()<br>";
?>
```



Line before calling function Validate()

```
<?php
error_reporting(E_ERROR); // Report fatal errors

echo "Line before calling function Validate()<br>";
// Calling an undefined function to see fatal error
Validate();
echo "Line after calling function Validate()<br>";
?>
```



Line before calling function Validate()

Fatal error: Call to undefined function Validate() in
C:\wamp\www\it3503\errors.php on line 14