Progettazione Web

PHP

Francesco Marcelloni

Dipartimento di Ingegneria dell’Informazione
Università di Pisa
ITALY

What is PHP

- PHP is recursive acronym for PHP: Hypertext Preprocessor
- 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 an open source software
- PHP is free to download and use

What is a PHP file

- PHP files can contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

- PHP+MySQL
  - PHP combined with MySQL are cross-platform (you can develop in Windows and serve on a Unix platform)

Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all web 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

Where to Start?

- To get access to a web server with PHP support, you can:
  - Install Apache (or IIS) on your own server, install PHP, and MySQL
  - Or find a web hosting plan with PHP and MySQL support

How is PHP included in HTML pages?

1. <?php echo 'if you want to serve XHTML or XML documents, do it like this'; ?>

2. <script language="php">
   echo 'some editors (like FrontPage) don\'t
   like processing instructions';
   </script>

3. <? echo 'this is the simplest, an SGML processing instruction'; ?>
   <?= expression ?> This is a shortcut for "<? echo expression ?>"

Tags in 1. and 2. are both always available. Tag 1. is the most commonly used, and recommended, of the two.

Short tags (example 3.) are only available when they are enabled via the short_open_tag php.ini configuration file directive, or if PHP was configured with the --enable-short-tags option.

PHP Syntax

- A PHP scripting block can be placed anywhere in the document.
- A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code

```
<?php if ($expression) { ?>
    <strong>This is true.</strong>
<?php } else {  ?>
    <strong>This is false.</strong>
    <?php } ?>
```

- When PHP hits the ?> closing tags, it simply starts outputting whatever it finds (except for an immediately following newline) until it hits another opening tag.

PHP Example

```
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset="utf-8">
    <title> PHP Example </title>
  </head>
  <body>
        <p><?php   echo "Hi, I'm a PHP script!";  ?></p>
    </body>
</html>
```

- NOTE: The file must have a .php extension. If the file has a .html extension, the PHP code will not be executed.

Instruction Separation and Comments

- NOTE: Each code line in PHP must end with a **semicolon**. The closing tag of a block of PHP code automatically implies a semicolon; you do not need to have a semicolon terminating the last line of a PHP block.
- NOTE: The closing tag for the block will include the immediately trailing newline if one is present.
- NOTE: In PHP, comments are expressed as:

  // to make a single-line comment

  /* to make a large comment block */

  # to make a single comment as in Unix shell

PHP Variables

- All variables in PHP start with a $ sign symbol.

  $var_name = value;
- In PHP, a variable does not need to be declared before adding a value to it.
- PHP automatically converts the variable to the correct data type, depending on its value.
- A variable name is case-sensitive and 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 _)

PHP Types

PHP supports eight primitive types.

- Four scalar types:
  - boolean
  - integer
  - float (floating-point number)
  - string
- Two special types:
  - NULL
  - resource
- Two compound types:
  - array
  - object

PHP Types
Boolean and Integer

- Boolean
  - To specify a boolean literal, use the keywords TRUE or FALSE. Both are case-insensitive.
- Integer
  - Integers can be specified in decimal (base 10), hexadecimal (base 16), or octal (base 8) notation, optionally preceded by a sign (- or +).
  - To use octal notation, precede the number with a 0 (zero). To use hexadecimal notation precede the number with 0x.
  - Integer size can be determined using the constant **PHP_INT_SIZE**, and maximum value using the constant **PHP_INT_MAX**
  - If PHP encounters a number beyond the bounds of the integer type, it will be interpreted as a float instead. Also, an operation which results in a number beyond the bounds of the integer type will return a float instead.

PHP Types
Floating-Point Numbers

- Floating point numbers

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

  - The size of a float is platform-dependent, although a maximum of ~1.8e308 with a precision of roughly 14 decimal digits is a common value (the 64 bit IEEE format).

PHP Types
NULL and Resource

- NULL

  The special NULL value represents a variable with no value. NULL is the only possible value of type NULL.

  A variable is considered to be null if:

  - the constant NULL has been assigned to it
  - it has not been set to any value yet.
  - it has been unset().

  The function is_null() allows evaluating whether the value of a variable is NULL

- Resource
  - A resource is a special variable, holding a reference to an external resource. Resources are created and used by special functions. For instance, dbase_open() opens a database.

PHP Types
String

- String
  1. Single quoted

     echo 'This is great <br>';

     - To specify a literal single quote, escape it with a backslash (\). To specify a literal backslash before a single quote, or at the end of the string, double it (\\).
  2. Double quoted
     - The most important feature of double-quoted strings is the fact that variable names will be expanded.
     - Simple syntax

       If a dollar sign ($) is encountered, the parser will greedily take as many tokens as possible to form a valid variable name. Enclose the variable name in curly brackets to explicitly specify the end of the name.

PHP Types String

- Complex Curly Syntax

  Any scalar variable, array variable or object property with a string representation can be included via this syntax.
  - Simply write the expression the same way as it would appear outside the string, and then wrap it in { and }.
  - Since { can not be escaped, this syntax will only be recognized when the $ immediately follows the {.
  - Use {\$ to get a literal {$.
- PHP uses the same escape sequences ("\") as in C++

  \n, \t, \\, \$, \", \0, *\[0-7]{1,3}, \x[0-9A-Fa-f]{1,2}*

PHP Types String

- Complex Curly Syntax (example)

```
<?php
$great = 'fantastic';

// Won't work, outputs: This is { fantastic}
echo "This is { $great}";

// Works, outputs: This is fantastic
echo "This is {$great}";

// Works
echo "This square is {$square->width}00 centimeters
   broad.";

// Works
echo "This works: {$arr[4][3]}";
?>
```

PHP Types String

3. Heredoc

   A third way to delimit strings is the heredoc syntax: <<<. After this operator, an identifier is provided, then a newline. The string itself follows, and then the same identifier again to close the quotation. Heredoc text behaves just like a double-quoted string.

```
<?php
$str = <<<EOD
Example of string
spanning multiple lines
using heredoc syntax.
EOD;
echo $str;
?>
```

Output: Example of string spanning multiple lines using heredoc syntax.

PHP Types String

- Characters within strings may be accessed and modified using square array brackets (or also braces)

```
<?php
// Get the first character of a string
$str = 'This is a test.';
$first = $str[0];

// Get the last character of a string.
$str = 'This is still a test.';
$last = $str[strlen($str)-1];

// Modify the last character of a string
$str = 'Look at the sea';
$str{strlen($str)-1} = 'e';
?>
```

PHP Types
String

- Concatenation operator ('.') - returns the concatenation of its right and left arguments.
- Concatenating assignment operator ('.=') - appends the argument on the right side to the argument on the left side.

```
<?php
$a = "Hello ";
$b = $a . "World!"; // now $b contains "Hello World!"
$a = "Hello ";
$a .= "World!";     // now $a contains "Hello World!"
?>
```

PHP Types
String

- The strpos() function is used to search for character within a string. If no match is found, it will return FALSE

```
<?php
echo strpos("Hello world!","world");
?>
```

- The position of the string "world" in our string is position 6 (the first position in the string is 0).

PHP Types
String Functions

- string chr (int $ascii) - Returns an one-character string containing the character specified by ascii.
- int ord (string $string) - Returns the ASCII value of the first character of string.
- void echo (string $arg1 [, string $...]) – Outputs all parameters. If you want to pass more than one parameter to echo(), the parameters must not be enclosed within parentheses.
  Es: echo $first, "<br>";
- array explode (string $delimiter , string $string [, int $limit]) - Returns an array of strings (at most limit), each of which is a substring of string formed by splitting it on boundaries formed by the string delimiter.

PHP Types
String Functions

- string implode (string $glue , array $pieces) - Join array elements with a glue string.

```
<?php
$array = array('lastname', 'email', 'phone');
$comma_separated = implode(",", $array);
echo "$comma_separated <br>";   // lastname,email,phone
$array = explode(",", $comma_separated);
for($i = 0, $size = sizeof($array); $i < $size; ++$i)
     echo "$array[$i] ";        // lastname email phone
?>
```

PHP Types String

- string number_format (float $number , int $decimals = 0 , string $dec_point = '.' , string $thousands_sep = ',') - generates the representation in string of a real number

  - Only one parameter: number will be formatted without decimals, but with a comma (",") between every group of thousands.
  - Two parameters: number will be formatted with decimals with a dot (".") in front, and a comma (",") between every group of thousands.
  - Four parameters: number will be formatted with decimals, dec_point instead of a dot (".") before the decimals and thousands_sep instead of a comma (",") between every group of thousands.

PHP Types
String

```
<?php
$number = 1234.56;
// english notation (default)
$english_format_number = number_format($number);
// 1,235
// French notation
$nombre_format_francais = number_format($number, 2, ',', ' ');
// 1 234,56
$number = 1234.5678;
// english notation without thousands separator
$english_format_number = number_format($number, 2, '.', '');
// 1234.57
?>
```

PHP Types
String

- void parse_str (string $str [, array &$arr])

  Parses str as if it were the query string passed via a URL and sets variables in the current scope.

```
<?php
$str = "first=value&arr[]=foo+bar&arr[]=baz";
parse_str($str);
echo "$first ";                          // value
echo "$arr[0] ";                         // foo bar
echo "$arr[1]<br>";                      // baz

parse_str($str, $output);
echo "{$output['first']} ";              // value
echo "{$output['arr'][0]} ";             // foo bar
echo "{$output['arr'][1]}<br>";          // baz

?>
```

PHP Types String

- int strncmp (string $str1 , string $str2 , int $len)

  This function is similar to strcmp(), with the difference that you can specify the (upper limit of the) number of characters from each string to be used in the comparison.

  Returns
  - < 0 if str1 is less than str2;
  - > 0 if str1 is greater than str2,
  - 0 if they are equal.

PHP Types
Array

- An array in PHP is actually an ordered map.
  - A map is a type that associates values to keys.
- Optimized for several different uses:
  - array, list (vector), hash table (an implementation of a map), dictionary, collection, stack, queue.
- Since array values can be other arrays, trees and multidimensional arrays are also possible.
- An array can be created by the *array()* language construct. It takes as parameters any number of comma-separated key => value pairs.

PHP Types
Array

```
array(  key =>  value
    , ...
    )
```

- key may only be an integer or string
  (booleans are converted into integers and NULL is managed as an empty string)
- value may be any value of any type

```
<?php
    $arr = array("foo" => "bar", 12 => true);
    echo $arr["foo"];      // bar
    echo $arr[12];         // 1
?>
```

PHP Types
Array

- If a key is not specified for a value, the maximum of the integer indices is taken and the new key will be that value plus 1.
- If a key that already has an assigned value is specified, that value will be overwritten.

```
<?php
// This array is the same as ...
array(5 => 43, 32, 56, "b" => 12, "a");

// ...this array
array(5 => 43, 6 => 32, 7 => 56, "b" => 12, 8 =>"a");
?>
```

PHP Types
Array

- A numeric array stores each array element with a numeric index.
- Two methods to create a numeric array

1.
```
$cars=array("Saab","Volvo","BMW","Toyota");
```

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

PHP Types
Array

```
<?php
$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);
// or
// $ages['Peter'] = 32;
// $ages['Quagmire'] = 30;
// $ages['Joe'] = 34;
echo "Peter is " . $ages['Peter'] . " years old.";
?>
```

Output: Peter is 32 years old.

PHP Types
Array

- An existing array can be modified by explicitly setting values in it.
  - This is done by assigning values to the array, specifying the key in brackets. The key can also be omitted.

```
<?php
$arr = array(5 => 1, 12 => 2);

$arr[] = 56;          // This is the same as $arr[13] = 56;
                      // at this point of the script

$arr["x"] = 42;       // This adds a new element to
                      // the array with key "x"

unset($arr[5]);       // This removes the element from the array

unset($arr);          // This deletes the whole array
?>
```

PHP Types Array

- To fill an array with *num* entries of the value of the *value* parameter, keys starting at the *start_index* parameter.

  array array_fill (int $start_index , int $num , mixed $value)

  If start_index is negative, the first index of the returned array will be start_index and the following indices will start from zero

```
<?php
$a = array_fill(5, 6, 'banana');
$b = array_fill(-2, 4, 'pear');
print_r($a);            //displays in a way that's readable by humans
echo '<br>';
print_r($b);
?>
```

```
Array ( [5] => banana [6] => banana [7] => banana [8] =>
banana [9] => banana [10] => banana )
Array ( [-2] => pear   [0] => pear   [1] => pear   [2] => pear )
```

PHP Types
Multidimensional Array

- 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
$arr = array("somearray" => array(6 => 5, 13 => 9, "a" =>
   42));
echo "{$arr["somearray"][6]}<br>";    // 5
echo "{$arr["somearray"][13]}<br>";   // 9
echo "{$arr["somearray"]["a"]}<br>";  // 42
?>
```

PHP Types
Array Functions

- array array_combine (array $keys , array $values) - Creates an array by using the values from the keys array as keys and the values from the values array as the corresponding values.
- array array_keys (array $input [, mixed $search_value [, bool $strict = false]]) - Returns the keys, numeric and string, from the input array.
- array array_merge (array $array1 [, array $array2 [, array $...]]) – Merges the elements of one or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.
  - If the input arrays have the same string keys, then the later value for that key will overwrite the previous one.
  - If the arrays contain numeric keys, the later value will *not* overwrite the original value, but will be appended.

PHP Types
Array Functions

- bool asort (array &$array [, int $sort_flags = SORT_REGULAR]) - sorts an array such that array indices maintain their correlation with the array elements they are associated with.

```
<?php
$fruits = array("d" => "lemon", "a" => "orange", "b" =>
  "banana", "c" => "apple");
asort($fruits);
foreach ($fruits as $key => $val) {
    echo "$key = $val\n";}
$num = array (4,2,7,3);
asort($num);
foreach ($num as $key => $val) {
    echo "$key = $val\n";}
?>
```

```
Output:
c = apple
b = banana
d = lemon
a = orange

1 = 2
3 = 3
0 = 4
2 = 7
```

PHP Types
Array Functions

- bool sort (array &$array [, int $sort_flags = SORT_REGULAR])- sorts an array. Elements will be arranged from lowest to highest when this function has completed.
  - # SORT_REGULAR - compare items normally (don't change types)
  - # SORT_NUMERIC - compare items numerically
  - # SORT_STRING - compare items as strings

```
<?php
$fruits = array("lemon", "orange", "banana", "apple");
sort($fruits);
foreach ($fruits as $key => $val) {
    echo "fruits[" . $key . "] = " . $val . "\n";
}
?>
```

```
Output:
fruits[0] = apple
fruits[1] = banana
fruits[2] = lemon
fruits[3] = orange
```

PHP Types
Array Functions

- bool ksort (array &$array [, int $sort_flags = SORT_REGULAR]) - sorts an array by key, maintaining key to data correlations.

```
<?php
$fruits = array("d"=>"lemon", "a"=>"orange", "b"=>"banana",
  "c"=>"apple");
ksort($fruits);
foreach ($fruits as $key => $val) {
    echo "$key = $val\n";
}
?>
```

Output:

```
a = orange
b = banana
c = apple
d = lemon
```

PHP Types
Array Functions

- array compact (mixed $varname [, mixed $...]) - creates an array containing variables and their values. For each of these, compact() looks for a variable with that name in the current symbol table and adds it to the output array such that the variable name becomes the key and the contents of the variable become the value for that key.

- int extract (array $var_array [, int $extract_type = EXTR_OVERWRITE [, string $prefix]]) - import variables from an array into the current symbol table. Checks each key to see whether it has a valid variable name. It also checks for collisions with existing variables in the symbol table. Returns the number of variables successfully imported.

PHP Types
Array Functions

```
$city  = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";
$location_vars = array("city", "state");
$result = compact("event", "nothing_here", $location_vars);
print_r($result);
$var_array = array("color" => "blue",
                   "size"  => "medium", "shape" => "sphere");
extract($var_array);
echo "<br>$color, $size, $shape\n";
?>
```

```
Array ( [event] => SIGGRAPH [city] => San Francisco
  [state] => CA )
  blue, medium, sphere
```

PHP Types
gettype

```
<?php
$a_bool = TRUE;          // a boolean
$a_str  = "foo";         // a string
$a_str2 = 'foo';         // a string
$an_int = 12;            // an integer

echo gettype($a_bool);          // prints out:  boolean
echo gettype($a_str);           // prints out:  string

// If this is an integer, increment it by four
if (is_int($an_int)) {    $an_int += 4;}

// If $a_bool is a string, print it out (does not print out anything)
if (is_string($a_bool)) {    echo "String: $a_bool";}
?>
```

PHP Types
settype

- To forcibly convert a variable to a certain type, either cast the variable or use the settype() function on it. It is forbidden to convert a variable to NULL or resouce types.

```
<?php
$foo = "5bar";          // string
$bar = true;            // boolean

settype($foo, "integer");      // $foo is now 5   (integer)
settype($bar, "string");       // $bar is now "1" (string)
?>
```

Variable scope

- Any variable used inside a function is by default limited to the local function scope

```
<?php
$a = 1;                 /* global scope */
function test()
{
    echo $a;            /* reference to local scope variable */
}
test();
?>
```

- Note: This script will not produce any output because the echo statement refers to a local version of the $a variable (undefined variable)

Variable scope
Accessing Global Variables

Two ways for accessing global variables.

- Use of global keyword

```
<?php
$a = 1;
$b = 2;
function sum()
{    global $a, $b;
      $b = $a + $b;
}
sum();
echo $b;
?>
```

- Note: The above script will output 3.

Variable scope
Accessing Global Variables

- Use of the special PHP-defined $GLOBALS array

```
<?php
$a = 1;
$b = 2;
function Sum()
{    $GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];
}
Sum();
echo $b;
?>
```

- The $GLOBALS array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element.

Variable scope
Static Variables

- A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope.

```
<?php
function test()
{
    static $a = 0;
    echo $a;
    $a++;
}
?>
```

- $a is initialized only in the first call of the function and, every time the test() function is called, it will print the value of $a and increment it.

Predefined Variable Superglobals

- Superglobals are built-in variables that are always available in all scopes
  - $GLOBALS — References all variables available in global scope
  - $_SERVER — an array containing server and execution environment information (headers, paths and script locations)
  - $_GET — An associative array of variables passed to the current script via the URL parameters
  - $_POST — An associative array of variables passed to the current script via the HTTP POST method.
  - $_FILES — An associative array of items uploaded to the current script via the HTTP POST method.
  - $_REQUEST — An associative array that by default contains the contents of $_GET, $_POST and $_COOKIE
  - $_SESSION — An associative array containing session variables available to the current script.

Predefined Variable Superglobals

- $_ENV — An associative array of variables passed to the current script via the environment method (provided by the shell under which PHP is running)
- $_COOKIE — An associative array of variables passed to the current script via HTTP Cookies
- $php_errormsg — $php_errormsg is a variable containing the text of the last error message generated by PHP. This variable will only be available within the scope in which the error occurred, and only if the track_errors configuration option is turned on (it defaults to off)
- $HTTP_RAW_POST_DATA — Raw POST data
- $http_response_header — HTTP response headers
- $argc — The number of arguments passed to script when running from the command line – php script.php arg1 arg2
- $argv — Array of arguments passed to the script

References

- By default, variables are always assigned by value.
- PHP also offers another way to assign values to variables: assign by reference.

```
<?php
$foo = 'Bob';                 // Assign the value 'Bob' to $foo
$bar = &$foo;                 // Reference $foo via $bar.
$bar = "My name is $bar";     // Alter $bar...
echo $bar;
echo $foo;                    // $foo is altered too.
?>
```

- There are three basic operations performed using references: assigning by reference, passing by reference, and returning by reference.

References

Function unset() allows separating the variable from its reference

```
<?php
  $a = 1000;
  $b =& $a;
  $b = 50;
  echo "$b<br>";          //50
  echo "$a<br>";          //50
  unset($b);
  $a = 0;
  echo "$a<br>";          //0
  echo "$b<br>";          //The variable is undefined
?>
```

Constants

- A valid constant name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

```
<?php
    define("FOO",     "something");
    define("FOO2",    "something else");
    define("FOO_BAR", "something more");
?>
```

- PHP provides a large number of predefined constants to any script which it runs.

Constants

Name	Description
__LINE__	The current line number of the file.
__FILE__	The full path and filename of the file. If used inside an include, the name of the included file is returned.
__DIR__	The directory of the file. If used inside an include, the directory of the included file is returned.
__FUNCTION__	The function name. The constant returns the function name as it was declared (case-sensitive).
__CLASS__	The class name. The constant returns the class name as it was declared (case-sensitive).
__METHOD__	The class method name. (Added in PHP 5.0.0) The method name is returned as it was declared (case-sensitive).

PHP Operators

- The PHP operators are practically equal to the C++ operators.
- In the following, we will show only the main differences

PHP Operators
Comparison Operators

- We show only the particularities

Example	Name	Result
$a = = $b	Equal	TRUE if $a is equal to $b.
$a = = = $b	Identical	TRUE if $a is equal to $b, and they are of the same type. (introduced in PHP 4)
$a != $b	Not equal	TRUE if $a is not equal to $b.
$a <> $b	Not equal	TRUE if $a is not equal to $b.
$a != = $b	Not identical	TRUE if $a is not equal to $b, or they are not of the same type. (introduced in PHP 4)

PHP Operators
Error Control Operators

- PHP supports one error control operator: the at sign (@). When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.
- If the track_errors feature is enabled, any error message generated by the expression will be saved in the variable $php_errormsg.

```
<?php
/* Intentional file error */
$my_file = @file ('non_existent_file') ;

$value = @$cache[$key];
// will not issue a notice if the index $key doesn't exist.
?>
```

PHP Operators
Execution Operators

- PHP supports one execution operator: backticks (` `). Note that these are not single-quotes!
- PHP will attempt to execute the contents of the backticks as a shell command; the output will be returned (i.e., it won't simply be dumped to output; it can be assigned to a variable).
- Use of the backtick operator identical to shell_exec().

```
<?php
// $output = `ls -al`;                Unix
   $output = `dir`;
  echo "<pre>$output</pre>";
?>
```

The backtick operator is disabled when safe mode is enabled or shell_exec() is disabled.

PHP Operators
Logical Operators

Example	Name	Result
$a and $b	And	TRUE if both $a and $b are TRUE.
$a or $b	Or	TRUE if either $a or $b is TRUE.
$a xor $b	Xor	TRUE if either $a or $b is TRUE, but not both.
! $a	Not	TRUE if $a is not TRUE.
$a && $b	And	TRUE if both $a and $b are TRUE.
$a || $b	Or	TRUE if either $a or $b is TRUE.

- The two different variations of "and" and "or" operate at different precedences:
  - $e = false || true;   // Acts like: ($e = (false || true))
  - $f = false or true;   // Acts like: (($f = false) or true)

PHP Operators
Array Operators

Example	Name	Result
$a + $b	Union	Union of $a and $b.
$a == $b	Equality	TRUE if $a and $b have the same key/value pairs.
$a === $b	Identity	TRUE if $a and $b have the same key/value pairs in the same order and of the same types.
$a != $b	Inequality	TRUE if $a is not equal to $b.
$a <> $b	Inequality	TRUE if $a is not equal to $b.
$a !== $b	Non-identity	TRUE if $a is not identical to $b.

The *+* operator returns the right-hand array appended to the left-hand array; for keys that exist in both arrays, the elements from the left-hand array will be used, and the matching elements from the right-hand array will be ignored.

```
var_dump($a == $b);              // bool(true)
var_dump($a === $b);             // bool(false)
//var_dump displays structured information
```

PHP Operators
Type Operators

- instanceof is used to determine whether a PHP variable is an instantiated object of a certain class

```
<?php
class MyClass {}
class NotMyClass {}

$a = new MyClass;
var_dump($a instanceof MyClass);
var_dump($a instanceof NotMyClass);
?>
```

```
bool(true)
bool(false)
```

PHP Statements

- The PHP statements are practically equal to the C++ statements, except for some particular use:
  - if
  - switch
  - while
  - do-while
  - for
  - break
  - continue

- In the following, we highlight the main differences

PHP Statements
If

```
if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

```
<?php
$d=date("D");   //Returns a string formatted according
                // to the given format string
if ($d=="Fri")   echo "Have a nice weekend!";
elseif ($d=="Sun")   echo "Have a nice Sunday!";
else   echo "Have a nice day!";
?>
```

PHP Statements
For

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

<?php
$people = array(
        array('name' => 'Kalle', 'salt' => 856412),
        array('name' => 'Pierre', 'salt' => 215863)
        );

for($i = 0, $size = sizeof($people); $i < $size; ++$i)
{    $people[$i]['salt'] = rand(000000, 999999);}
print_r($people);
?>
```

```
  Array ( [0] => Array ( [name] => Kalle [salt] => 64086 )
          [1] => Array ( [name] => Pierre [salt] => 49713 ) )
```

PHP Statements
Foreach

Foreach gives an easy way to iterate over arrays.

- Note: foreach works only on arrays
- Two possible syntaxes (the second is an extension of the first)

foreach (array_expression **as** $value)
            statement
\\ On each loop, the value of the current element is assigned to
\\ $value and the internal array pointer is advanced by one

foreach (array_expression **as** $key => $value)
    statement
\\ On each loop, the current element's key will be assigned to the
\\ variable $key and the internal array pointer is advanced by one

PHP Statements Foreach

- Unless the array is referenced, foreach operates on a copy of the specified array and not the array itself.
- You can easily modify array's elements by preceding $value with &. This will assign reference instead of copying the value.

```
<?php
$arr = array(1, 2, 3, 4);
foreach ($arr as &$value) {
    $value = $value * 2;
}
// $arr is now array(2, 4, 6, 8)
$value=1000;   //Note! $value refers to the last element
print_r($arr);   // $arr is now array(2, 4, 6, 1000)
unset($value); // break the reference with the last element
?>
```

PHP Statements
Break

- break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out

```
$i = 0;
while (++$i) {
    switch ($i) {
    case 5:
        echo "At 5<br>";
        break 1;              /* Exit only the switch. */
    case 10:
        echo "At 10; quitting<br>";
        break 2;              /* Exit the switch and the while. */
    default:
        break;
    }
}
```

PHP Statements Continue

- continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end
- Note: continue applies to switch and acts similar to break.

```
$i = 0;
while ($i++ < 5) {
    echo "Outer<br>";
    while (1) {
        echo "&nbsp;&nbsp;Middle<br>";
        while (1) {
            echo "&nbsp;&nbsp;Inner<br>";
            continue 3;
        }
        echo "This never gets output.<br>";
    }
    echo "Neither does this.<br>";
}
```

PHP Statements
Alternative syntax

- PHP offers an alternative syntax for some of its control structures; namely, if, while, for, foreach, and switch.
- The basic form of the alternate syntax is to change the opening brace to a colon (:) and the closing brace to, respectively:
  - endif;
  - endwhile;
  - endfor;
  - endforeach;
  - or endswitch;.

PHP Statements Alternative syntax

Example:

```
<?php
$a=6;
if ($a == 5):
    echo "a equals 5";
    echo "...";
elseif ($a == 6):
    echo "a equals 6";
    echo "!!!";
else:
    echo "a is neither 5 nor 6";
endif;
?>
```

PHP Statements
Include and Require

- The include() and require() statements include and evaluate the specified file.
- Files are included based on the file path given or, if none is given, the include_path specified.
- If the file isn't found in the include_path, include() and require() will finally check in the calling script's own directory and the current working directory before failing.
- The include() construct will emit a warning if it cannot find a file; this is different behavior from require(), which will emit a fatal error.

PHP Statements Include and Require

```
vars.php
<?php
$color = 'green';
$fruit = 'apple';
?>

test.php
<?php
echo "A $color $fruit";        // A
include 'vars.php';
echo "A $color $fruit";        // A green apple
?>
```

PHP Statements Include and Require

- It is possible to execute a return() statement inside an included file in order to terminate processing in that file and return to the script which called it. Also, it's possible to return values from included files.

return.php
```
<?php
$var = 'PHP';
return $var;
?>
```

noreturn.php
```
<?php
$var = 'PHP';
?>
```

PHP Statements
Include and Require

```
testreturns.php
<?php
$foo = include 'return.php';
echo "$foo<br>";                    // prints 'PHP'
$bar = include 'noreturn.php';
echo $bar;                    // prints 1 (the include was successful)
?>
```

NOTE: The include_once() and require_once() statements include and evaluate the specified file during the execution of the script. This is a behavior similar to the include() and require() statements, with the only difference being that if the code from a file has already been included, it will not be included again.

PHP Functions

- Function definition

```
<?php
function foo($arg_1, $arg_2, /* ..., */ $arg_n)
{
 //….
  echo "Example function.\n";
 return $retval;
}
?>
```

- Functions need not be defined before they are referenced, except when a function is conditionally defined

PHP Functions

- Conditionally defined function

```
<?php
$makefoo = true;
/* We can't call foo() from here   since it doesn't exist yet,
   but we can call bar() */
bar();
if ($makefoo) {
  function foo()
  {    echo "I don't exist until program execution reaches me.\n";
  }
}
/* Now we can safely call foo()  since $makefoo evaluated to true */
if ($makefoo) foo();
function bar()
{  echo "I exist immediately upon program start.\n";}
?>
```

PHP Functions

- Functions within functions

```
<?php
function foo()
{  function bar()
  {    echo "I don't exist until foo() is called.\n";  }
}

/* We can't call bar() yet   since it doesn't exist. */

foo();

/* Now we can call bar(),   foo()'s processesing has
    made it accessible. */
bar();

?>
```

PHP Functions

- All functions and classes in PHP have the global scope - they can be called outside a function even if they were defined inside and vice versa.
- PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.
- Function names are case-insensitive

PHP Functions Arguments

- PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are also supported

- Passing arrays (by value) to functions

```
<?php
  $arr=array(2,3);
  takes_array($arr);                 \\ 2 + 3 = 5
  takes_array($arr);                 \\ 2 + 3 = 5
function takes_array($input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1], "<br>";
   $input[0]=10;
}
?>
```

PHP Functions
Arguments

- Passing arrays (by reference) to functions

```
<?php
   $arr=array(2,3);
   takes_array($arr);                      \\ 2 + 3 = 5
   takes_array($arr);                      \\ 10 + 3 = 13
function takes_array(&$input)
{
    echo "$input[0] + $input[1] = ", $input[0]+$input[1], "<br>";
    $input[0]=10;
}
?>
```

PHP Functions Arguments

- Passing strings (by value) to functions

```
<?php
  function add_some_extra($string)
  {   $string .= 'and something extra.';}
  $str = 'This is a string, ';
  add_some_extra($str);
  echo $str;     // outputs 'This is a string, '
?>
```

- Passing strings (by reference) to functions

```
<?php
  function add_some_extra(&$string)
  //..
  echo $str;       // outputs 'This is a string, and something extra.'
?>
```

PHP Functions Arguments

- Default arguments

```
<?php
function makecoffee($type = "cappuccino")
{
    return "Making a cup of $type. <br>";
}
echo makecoffee();             \\  Making a cup of cappuccino.
echo makecoffee(null);         \\  Making a cup of .
echo makecoffee("espresso");   \\  Making a cup of espresso.
?>
```

The default value must be a constant expression, not (for example) a variable, a class member or a function call.

PHP Functions Arguments

- Note that when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected.

```
<?php
function makeyogurt($type = "acidophilus", $flavour)
{
    return "Making a bowl of $type $flavour.\n";
}
echo makeyogurt("raspberry");   // won't work as expected
?>
```

```
Warning: Missing argument 2 for makeyogurt(), called in
  c:\tia\www\php34.php on line 16 and defined in
  c:\tia\www\php34.php on line 11

  Making a bowl of raspberry .
```

PHP Functions Arguments

- PHP 4 and above has support for variable-length argument lists in user-defined functions.
- This is really quite easy, using the func_num_args(), func_get_arg(int), and func_get_args() functions.

```
<?php
function foo()
{    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br>";
    $arg_list = func_get_args();
    for ($i = 0; $i < $numargs; $i++) {
        echo "Argument $i is: " . $arg_list[$i] . "<br>";  }
}
foo(1, 2, 3);
?>
```

```
Number of arguments: 3
Argument 0 is: 1
Argument 1 is: 2
Argument 2 is: 3
```

PHP Functions
Returning values

- Values are returned by using the optional return statement.
- Any type may be returned, including arrays and objects.

```
<?php
function small_numbers()
{    return array (0, 1, 2);
}
$arr = small_numbers();
print_r($arr);
?>
```

```
Array ( [0] => 0 [1] => 1 [2] => 2 )
```

PHP Functions
Variable Functions

- If a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it.

```
<?php
function foo() {    echo "In foo()<br>";}
function bar($arg = '')
{    echo "In bar(); argument was '$arg'.<br>";}
// This is a wrapper function around echo
function echoit($string) {    echo $string;}
$func = 'foo';
$func();                    // This calls foo()
$func = 'bar';
$func('test');              // This calls bar()
$func = 'echoit';
$func('test');              // This calls echoit()
?>
```

PHP Functions
Built-in Functions

- In PHP, there are more than 700 built-in functions.
- Functions are organized in categories in the function reference.
- For instance:
  - Affecting PHP's Behaviour
  - Audio Formats Manipulation
  - Authentication Services
  - Date and Time Related Extensions
  - Compression and Archive Extensions

PHP Functions
Date/Time Functions

- string date (string $format [, int $timestamp])

  Returns a string formatted according to the given format string using the given integer timestamp (the number of seconds since January 1 1970 00:00:00 UTC) or the current time if no timestamp is given.

Format character	**Description**	**Example returned values**
d	Day of the month, 2 digits with leading zeros	01 to 31
D	A textual representation of a day, three letters	Mon through Sun
j	Day of the month without leading zeros	1 to 31
l (lowercase 'L')	A full textual representation of the day of the week	Sunday through Saturday
…	…	…

PHP Functions
Date

The timestamp can be fixed by using function

- int strtotime (string $time [, int $now])

  Parse about any English textual datetime description into a Unix timestamp

```
<?php
echo date("F j, Y, g:i a") . "<br>";        // January 7, 2011, 1:02 pm
echo date("m.d.y")  . "<br>";               // 01.07.11
echo date("j, n, Y") . "<br>";              // 7, 1, 2011
echo date("Ymd") . "<br>";                  // 20110107
echo date("F j, Y, g:i a", strtotime("10 September 2000")) . "<br>";
    // September 10, 2000, 12:00 am
echo date("m.d.y")  . "<br>";                // 01.07.11
echo date("F j, Y, g:i a", strtotime("+1 day")) . "<br>";
   // January 8, 2011, 1:02 pm
?>
```

PHP Functions
Date/Time Functions

- array getdate ([int $timestamp = time()])

  Returns an associative array containing the date information of the timestamp, or the current local time if no timestamp is given.

  The keys to access to the values are: seconds, minutes, hours, mday, wday, mon, year,yday, weekday (full text representation of the day of the week), month (full textual representation of a month).

- int time (void)

  Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

PHP Functions
FileSystem Functions

- resource fopen (string $filename , string $mode [, bool $use_include_path = false [, resource $context]])

  fopen() binds a named resource, specified by filename, to a stream.

  If *filename* is of the form "scheme://...", it is assumed to be a URL and PHP will search for a protocol handler (also known as a wrapper) for that scheme.

  If PHP has decided that filename specifies a local file, then it will try to open a stream on that file.

  If PHP has decided that filename specifies a registered protocol, and that protocol is registered as a network URL, PHP will check to make sure that allow_url_fopen is enabled.

  Modes: ‘r’, ‘r+’, ‘w’, ‘w+’, ‘a’, ‘a+’.

  ‘+’ indicates reading and writing

PHP Functions
FileSystem Functions

- bool copy (string $source , string $dest)

  Makes a copy of the file source to dest.
- array file (string $filename)

  Reads an entire file into an array. Each element of the array corresponds to a line in the file, with the newline still attached.
- bool flock (resource $handle , int $operation)

  allows you to perform a simple reader/writer model which can be used on virtually every platform(all accessing programs have to use the same way of locking)

  handle - A file system pointer resource that is typically created using fopen().

  operation is one of the following:

  * LOCK_SH to acquire a shared lock (reader).

  * LOCK_EX to acquire an exclusive lock (writer).

  * LOCK_UN to release a lock (shared or exclusive).

PHP Functions FileSystem Functions

- int fwrite (resource $handle , string $string [, int $length])

  writes the contents of string to the file stream pointed to by handle.
- string fread (resource $handle , int $length)

  reads up to length bytes from the file pointer referenced by handle. Reading stops as soon as one of the following conditions is met:
  - length bytes have been read
  - EOF (end of file) is reached
  - a packet becomes available or the socket timeout occurs (for network streams)
  - 8192 bytes have been read (after opening userspace stream)

PHP Functions
FileSystem Functions

```
<?php
  $fp = fopen("lock.txt", "r+");
  if (flock($fp, LOCK_EX)) {                  // do an exclusive lock
    ftruncate($fp, 0);                        // truncate file
    fwrite($fp, "Write something here\n");
    flock($fp, LOCK_UN);                      // release the lock
  }
  else {     echo "Couldn't get the lock!";}
  fclose($fp);
  $fp = fopen("lock.txt", "r");
  echo fread($fp, filesize('lock.txt'));      // Write something here
  fclose($fp);
?>
```

```
Write something here
```

PHP Functions
Regular Expressions

- A regular expression is a pattern that is matched against a subject string from left to right. Most characters stand for themselves in a pattern, and match the corresponding characters in the subject.
- The expression must be enclosed in the delimiters, a forward slash (/), for example. Delimiters can be any non-alphanumeric, non-whitespace ASCII character except the backslash (\) and the null byte.
- The ending delimiter may be followed by various modifiers that affect the matching.

PHP Functions
Regular Expressions

Some examples of modifiers

- i - letters in the pattern match both upper and lower case letters
- A - If this modifier is set, the pattern is forced to be "anchored", that is, it is constrained to match only at the start of the string which is being searched (the "subject string").
- s - If this modifier is set, a dot metacharacter in the pattern matches all characters, including newlines. Without it, newlines are excluded.

PHP Functions
Regular Expressions

Some functions

- int preg_match (string $pattern , string $subject [, array &$matches)

  Searches $subject for a match to the regular expression given in $pattern.

  If matches is provided, then it is filled with the results of search.

- int preg_match_all (string $pattern , string $subject [, array &$matches)

  Searches subject for all matches to the regular expression given in pattern

- array preg_split (string $pattern , string $subject)

  Split the given string by a regular expression. The function returns an array containing substrings of subject split along boundaries matched by pattern

PHP Functions
Regular Expressions

```
 <?php
/* The \b in the pattern indicates a word boundary, so only the
  distinct word "the" is matched */
  $str="PHP is the scripting language of the web.";
  if (preg_match("/\bthe\b/i", $str,$output)) {
    echo "A match was found.<br>"; print_r($output);}
  else {    echo "A match was not found.";}
  if (preg_match_all("/\bthe\b/i", $str,$output)) {
    echo "<br>A match was found.<br>"; print_r($output);}
  else {   echo "A match was not found.";}
  echo "<br>";
  $result=preg_split("/[\s,]+/",  $str);
  if ($result) print_r($result);
?>
```

PHP Functions Regular Expressions

Output

```
A match was found.
Array ( [0] => the )
A match was found.
Array ( [0] => Array ( [0] => the [1] => the ) )
Array ( [0] => PHP [1] => is [2] => the [3] => scripting [4] =>
  language [5] => of [6] => the [7] => web. )
```

PHP Functions Other Functions

- mixed print_r (mixed $expression)

  print_r — Prints human-readable information about a variable

```
<pre>
<?php
$a = array ('a' => 'apple', 'b' => 'banana', 'c' => array ('x', 'y', 'z'));
print_r ($a);
?>
</pre>
```

PHP Functions
Other Functions

Output

```
<pre>
Array
(
    [a] => apple
    [b] => banana
    [c] => Array
        (
            [0] => x
            [1] => y
            [2] => z
        )
)
</pre>
```

PHP Classes

- Syntax equal to Java
- The visibility of a property or method can be defined by prefixing the declaration with the keywords public, protected or private.
- This declaration may include an initialization, but this initialization must be a constant value (that is, evaluated at compile time)
- Methods declared without any explicit visibility keyword are defined as public.
- PHP 5 allows developers to declare constructor methods for classes.

PHP Classes
Constructors and Destructors

```
class MyDestructableClass {
  private $name="Default: ";
   function __construct() {
        print "In constructor\n";
        $this->name .= "MyDestructableClass";
   }
   function __destruct() {
        print "Destroying " . $this->name . "\n";
   }
}
$obj = new MyDestructableClass;
?>
```

```
In constructor
Destroying Default: MyDestructableClass
```

PHP Classes

- NOTE: For backwards compatibility, if PHP 5 cannot find a __construct() function for a given class, it will search for the old-style constructor function, by the name of the class.
- NOTE: “$this” is not a pointer but rather a reference
- NOTE: if we do not desire to receive error alerts when using new, adopt the operator @, that is, @new

PHP Classes

```
<?php
class Stack {
  private $top, $vett;
   function Stack() {
       print "In Stack\n";
       $this->top =-1;
   }
   function push($elem) {
      $this -> vett[++$this->top]=$elem;
    }
```

PHP Classes

```
  function pop() {
   if ($this->top>-1)
     return $this -> vett[$this->top--];
   return null;
    }
}
$mystack = new Stack;
$mystack->push(15);
echo $mystack->pop()."<br>";
echo $mystack->pop()."<br>";
?>
```

```
In Stack 15
```

PHP Classes
Inheritance

```
class Foo
{
    public function printItem($string)
    {       echo 'Foo: ' . $string . "<br>";    }
    public function printPHP()
    {       echo 'PHP is great.' . "<br>";    }
}

class Bar extends Foo
{
    public function printItem($string)
    {       echo 'Bar: ' . $string . "<br>";    }
}
```

PHP Classes Inheritance

```
$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Output: 'Foo: baz'
$foo->printPHP();       // Output: 'PHP is great'
$bar->printItem('baz'); // Output: 'Bar: baz'
$bar->printPHP();       // Output: 'PHP is great'
?>
```

PHP Classes
Scope Resolution Operator

```
<?php
class MyClass {
    const CONST_VALUE = 'A constant value';
}
echo MyClass::CONST_VALUE;
?>
```

PHP Classes Scope Resolution Operator

```
<?php
class BaseClass {
   function __construct() {print "In BaseClass constructor<br>";   }
}
class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();        // explicit call
       print "In SubClass constructor <br>";
   }
}
$obj = new BaseClass();             //In BaseClass constructor
$obj = new SubClass();              //In BaseClass constructor
                                    //In SubClass constructor
?>
```

PHP Classes
Scope Resolution Operator

```
<?php
class MyClass
{    protected function myFunc() {
          echo "MyClass::myFunc()<br>";    }
}
class OtherClass extends MyClass
{    // Override parent's definition
    public function myFunc()
    {         // But still call the parent function
          parent::myFunc();   echo "OtherClass::myFunc()<br>";    }
}
$class = new OtherClass();
$class->myFunc();        //MyClass::myFunc()
                         //OtherClass::myFunc()
?>
```

PHP Classes
Static Keyword

- Declaring class members or methods as static makes them accessible without needing an instantiation of the class.
- A member declared as static can not be accessed with an instantiated class object (though a method can).

```
<?php
class Foo
{   public static $my_static = 0;
    public function staticValue() {
        return ++self::$my_static;    }
}

class Bar extends Foo
{    public function fooStatic() {
        return ++parent::$my_static;
    }
}
```

PHP Classes Static Keyword

```
print Foo::$my_static . "<br>";              //0

$foo = new Foo();
print $foo->staticValue() . "<br>";          //1
print $foo->my_static . "<br>";  // Undefined "Property" my_static

// $foo::my_static is not possible

print Bar::$my_static . "<br>";              //1
$bar = new Bar();
print $bar->fooStatic() . "<br>";            //2
?>
```

PHP Classes Object Serialization

- serialize() returns a string containing a byte-stream representation of any value that can be stored in PHP.
- unserialize() can use this string to recreate the original variable values. Using serialize to save an object will save all variables in an object.
- PHP checks if your class has a function with the magic name __sleep and will attempt to call it prior to serialization so as to allow the object to do any last minute clean-up, etc. prior to being serialized. Likewise, when the object is restored using unserialize() the __wakeup member function is called.

```
<?php
class A {
    public $one = 100;
    public function show_one() {  echo $this->one;  }
 }
?>
```

PHP Classes
Object Serialization

```
// page1.php:
  include("classa.inc");
  $a = new A;
  $s = serialize($a);
  // store $s somewhere where page2.php can find it.
  file_put_contents('store', $s);

// page2.php:
// this is needed for the unserialize to work properly.
  include("classa.inc");
  $s = file_get_contents('store');
  $a = unserialize($s);
  // now use the function show_one() of the $a object.
  $a->show_one();
```

Forms and User Input

- Any form element in an HTML page will automatically be available to PHP scripts
- The use of the global associative arrays $_GET , $_POST and $_REQUEST allows retrieving information from forms, like user input.
  - $_GET - An associative array of variables passed to the current script via the URL parameters.
  - $_POST - An associative array of variables passed to the current script via the HTTP POST method.
  - $_REQUEST - An associative array that by default contains the contents of $_GET, $_POST and $_COOKIE.

Forms and User Input

```
<head>
    <title>Form Example</title>
  </head>
  <body>
    <form action="elab.php" method="post">
      <p>
        Name:<br>
        <input type="text" name="username"><br>
        E-mail:<br>
        <input type="text" name="email"><br>
        <input type="submit" name="submit"
           value="SUBMIT">
      </p>
    </form>
  </body>
```

Forms and User Input

```
// file elab.php
<?php
  print $_POST['username'];
  print $_REQUEST['username'];
  ?>

<form action="elab.php" method="get">

// file elab.php
<?php
  print $_GET['username'];
  print $_REQUEST['username'];
  ?>
```

Forms and User Input

bool import_request_variables (string $types [, string $prefix])

- Imports GET/POST/Cookie variables into the global scope.
  - Types parameter – you can specify which request variables to import. You can use 'G', 'P' and 'C' characters respectively for GET, POST and Cookie. These characters are not case sensitive.
  - $prefix parameter - is prepended before all variable's name imported into the global scope. So if you have a GET value named "userid", and provide a prefix "pref_", then you'll get a global variable named $pref_userid.

Forms and User Input

```
// file elab.php

<?php
  import_request_variables('p', 'p_');
  print $p_username . "<br>";
  print $p_email;
  ?>
```

PHP Forms and User Input

- Form Validation
  - User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.
  - You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

Get or Post

- 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).

- 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.

Cookie

- A cookie is a small piece of data sent from a website and stored in a user's web browser while the user is browsing that website.
  - A web server specifies a cookie to be stored by sending an HTTP header called Set-Cookie.

  Set-Cookie: value[; expires=date][; domain=domain][; path=path][; secure]

  - When a cookie is present, and the optional rules allow, the cookie value is sent to the server with each subsequent request. The cookie value is stored in an HTTP header called Cookie and contains just the cookie value without any of the other options

  Cookie: value

Cookie

bool setcookie (string $name [, string $value [, int $expire = 0 [, string $path [, string $domain [, bool $secure = false [, bool $httponly = false]]]]]])

- setcookie() defines a cookie to be sent along with the rest of the HTTP headers.
  - Like other headers, cookies must be sent before any output from your script (this is a protocol restriction). This requires that you place calls to this function prior to any output, including <html> and <head> tags as well as any whitespace.

Cookie

Expire

- Indicates when the cookie should no longer be sent to the server and therefore may be deleted by the browser.
- Without the expires option, a cookie has a lifespan of a single session. A session is defined as finished when the browser is shut down, so session cookies exist only while the browser remains open.

Cookie

Domain

- Indicates the domain(s) for which the cookie should be sent.
- By default, domain is set to the host name of the page setting the cookie, so the cookie value is sent whenever a request is made to the same host name.
- For example, the default domain for a cookie set might be www.ing.unipi.it.
- The domain option is used to widen the number of domains for which the cookie value will be sent.
- Consider the case of a large network such as Yahoo! that has many sites in the form of name.yahoo.com (e.g., my.yahoo.com, finance.yahoo.com, etc.). A single cookie value can be set for all of these sites by setting the domain option to simply yahoo.com.

Cookie

Path

- Indicates a URL path that must exist in the requested resource before sending the Cookie header. This comparison is done by comparing the option value character-by-character against the start of the request URL. If the characters match, then the Cookie header is sent.

  Ex:

  ```
  Set-Cookie: name=Nicholas; path=/blog
  ```

- The path option would match /blog, /blogroll, etc.; anything that begins with /blog is valid. Note that this comparison is only done once the domain option has been verified. The default value for the path option is the path of the URL that sent the Set-Cookie header.

Cookie

Secure

- just a flag and has no additional value specified. A secure cookie will only be sent to the server when a request is made using SSL and the HTTPS protocol.
- confidential or sensitive information should never be stored or transmitted in cookies as the entire mechanism is inherently insecure. By default, cookies set over an HTTPS connection are automatically set to be secure.

Cookie

```
<?php
$value = 'something from somewhere';
setcookie("TestCookie", $value, time()+3600, "/~rasmus/",
  ".example.com", 1);
  /* expire in 1 hour, available within the /~rasmus/ directory and
  all sub-directories, available on all subdomains of example.com
  and set if a secure connection exists*/
?>
```

Cookie

To see the contents of the test cookie in a script

```
<?php
// Print an individual cookie
echo $_COOKIE["TestCookie"];
// Another way to debug/test is to view all cookies
print_r($_COOKIE);
?>
```

Output

```
something from somewhere
Array ( [TestCookie] => something from somewhere )
```

Cookie

- Cookies are usually set in an HTTP header (although JavaScript can also set a cookie directly on a browser)

An example of header returned by a PHP program:

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Feb-07 22:03:38 GMT;
            path=/; domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

Cookie

- If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server.

An example of header used by a browser:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

Cookie

- Example of use of the cookies

```
<html>
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
      echo $_COOKIE["name"]. "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"]. "<br />";
      echo $_COOKIE["age"] . "<br />";
      /* is equivalent to */
      echo $HTTP_COOKIE_VARS["name"] . "<br />";
    ?>
  </body>
</html>
```

Cookie

- Example of use of the cookies

```
<html>
  <head>
    <title>Accessing Cookies with PHP</title>
  </head>
  <body>
    <?php
      if( isset($_COOKIE["name"]))
        echo "Welcome " . $_COOKIE["name"] . "<br />";
      else
        echo "Sorry... Not recognized" . "<br />";
    ?>
  </body>
</html>
```

Cookie

- Deleting cookie with PHP

```
<?php
   setcookie( "name", "", time()- 60, "/","", 0);
   setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>
   <head>
      <title>Deleting Cookies with PHP</title>
   </head>
   <body>
      <?php echo "Deleted Cookies" ?>
   </body>
</html>
```

Sessions

- Session support allows preserving certain data across subsequent accesses.
  - This enables you to build more customized applications and increase the appeal of your web site.
- A visitor accessing your web site is assigned a unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.
- The session support allows you to register arbitrary numbers of variables to be preserved across requests.
  - When a visitor accesses your site, PHP will check automatically (if session.auto_start is set to 1) or on your request (explicitly through session_start() or implicitly through session_register()) whether a specific session id has been sent with the request.
  - If this is the case, the prior saved environment is recreated.

Sessions

```
//login.php
<?php
  session_start();
  $_SESSION['count'] = 0;
?>
  <p>Welcome :)<br>
  <a href="page.php">GO</a>.
  </p>
```

session_start() creates a session or resumes the current one based on a session identifier passed via a GET or POST request, or passed via a cookie

Sessions

```
//page.php
<?php
  session_start();
  if (!isset($_SESSION['count']))
          //Determine if a variable is set and is not NULL.
  {  echo "<p>You have to login</p>";  }
  else {  $_SESSION['count']++;
  echo "<p>Hi, you have visited this page " .
  $_SESSION['count'] . " times.</p>";
  echo "<p>
  <a href=\"page.php\">GO</a>, or
  <a href=\"logout.php\">Exit</a>.</p>";}
?>
```

Sessions

```
//logout.php
<?php
  session_start();  unset($_SESSION['count']);
?>
<p>Good Bye :(</p>
```

Sessions

```
//form.php
<!DOCTYPE html>
<html lang="en">
  <head>
   <meta charset="utf-8">
   <title>Login</title>
    <style type="text/css"> p { font-size: 30pt;} </style>
  </head>
  <body>
   <form method="post" action="login.php">
   <p>Username: <input type="text" name="username"></p>
   <p>Password: <input type="text" name="password"></p>
   <p><input type="submit" value="Let me in"></p>
   </form>
   </body>
   </html>
```

Sessions

```
//login.php
<!DOCTYPE html>
  <html lang="en">
  <head>
  <meta charset="utf-8">
  <title>Login</title>
  </head>
  <body>
  <?php
  // Check if username and password are correct
  if ($_POST["username"] == "php" && $_POST["password"] ==
  "php") {
```

Sessions

```
//login.php
// If correct, we set the session to YES
    session_start();
    $_SESSION["Login"] = "YES";
    echo "<h1>You are now logged correctly in</h1>";
    echo "<p><a href='document.php'>Link to protected
  file</a></p>";
  }
```

Sessions

```
//login.php
else {
  // If not correct, we set the session to NO
   session_start();
   $_SESSION["Login"] = "NO";
   echo "<h1>You are NOT logged correctly in </h1>";
   echo "<p><a href='document.php'>Link to protected
 file</a></p>";
 }
 ?>
 </body>
 </html>
```

Sessions

```
//document.php
<?php
  // Start up your PHP Session
  session_start();
  // If the user is not logged in send him/her to the login form
  if ($_SESSION["Login"] != "YES") {
    header("Location: form.php");
  }
  ?>
```

See laboratories 8 and 9 for more details on the use of the Session variables

Sessions

```
//document.php
<!DOCTYPE html>
 <html lang="en">
  <head>
   <meta charset="utf-8">
    <title>Protected File</title>
    <style type="text/css"> p { font-size: 30pt;} </style>
  </head>
  <body>
   <h1>This document is protected</h1>
   <p>You can only see it if you are logged in.</p>
   <a href='logout.php'>Exit</a> </p>
   </body>
   </html>
```

Sessions

```
//logout.php
<!DOCTYPE html>
<?php
   session_start();  unset($_SESSION['Login']);
?>
  <html lang="en">
  <head>
   <meta charset="utf-8">
    <title>Logout</title>
    <style type="text/css"> p { font-size: 30pt;} </style>
  </head>
  <body>
   <p>Good Bye :(</p>
   </body>
   </html>
```

PHP and mysql

- Connecting to MySQL server

  mysqli::__construct() ([string $host = ini_get("mysqli.default_host") [, string $username = ini_get("mysqli.default_user") [, string $passwd = ini_get("mysqli.default_pw") [, string $dbname = "" [, int $port = ini_get("mysqli.default_port") [, string $socket = ini_get("mysqli.default_socket")]]]]]])

  Opens a connection to the MySQL server.

  Returns an object which represents the connection to a MySQL Server.

PHP and mysql

- Connecting to MySQL server

```
<?php
$mysqli= new mysqli($nomeHost, $nomeUtente, $password, $db);
if ($mysqli->connect_error) {
    die('Connect Error (' . $mysqli->connect_errno . ') '
            . $mysqli->connect_error);
}
echo 'Success... ' . $mysqli->host_info . "\n";

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

```
Success... MySQL host info: localhost via TCP/IP
```

PHP and mysql

- Selecting a database

  bool mysqli::select_db (string $database_name)

  Selects the default database to be used when performing queries against the database connection.

  Every subsequent call to mysql_query() will be made on the active database.

```
// make pweb the current db
$mysqli->select_db($nomeDb) or
    die ('Can\'t use pweb: ' . mysql_error());
```

PHP and mysql

- Send a query

  mixed mysqli::query (string $query)

  Performs a query against the database.

  - For SELECT, SHOW, DESCRIBE, EXPLAIN and other statements returning resultset, returns a mysqli_result object on success, or FALSE on error.
  - For other type of SQL statements, INSERT, UPDATE, DELETE, DROP, etc, returns TRUE on success or FALSE on error.
  - The returned result resource should be passed to mysqli_fetch_array(), and other functions for dealing with result tables, to access the returned data.

PHP and mysql

```
<?php
 $nomeHost = "localhost";
 $nomeUtente = "root";
 $password = "";
 $nomeDb = “pweb";
 $mysqli= new mysqli($nomeHost, $nomeUtente, $password,
 $nomeDb);
 if ($mysqli->connect_error) {
   die('Connect Error (' . $mysqli->connect_errno . ') '
           . $mysqli->connect_error);
 }
 echo 'Success... ' . $mysqli->host_info . "\n";
```

PHP and mysql

```
// Formulate Query
// This is the best way to perform an SQL query
$query = "SELECT * FROM studenti";

// Perform Query
$result = $mysqli->query($query);

/* Check result
    This shows the actual query sent to MySQL, and the error.
    Useful for debugging.*/
checkResult($result,$query);

//show results
showResult($result);
```

PHP and mysql

```
  // Free the resources associated with the result set
  // This is done automatically at the end of the script
  $result->close();
  $mysqli->close();

function checkResult($result, $query)
{ if (!$result) {
    $message  = 'Invalid query: ' . $mysqli->error . "\n";
    $message .= 'Whole query: ' . $query;
    die($message);
    }
}
```

PHP and mysql

```
  function showResult($result)
  {       while ($row = $result->fetch_assoc()) {
          echo $row['MATRICOLA']. " ";
          echo $row['NOME'] . "<br>";
          }
          echo "<br>";
  }
?>
```

```
1111 ROMOLO
2222 REMO
3333 CESARE
4444 PIPPO
```

mysqli_result::fetch_assoc() - Returns an associative array that corresponds to the fetched row and moves the internal data pointer ahead.

PHP and mysql

```
$name  = 'ROMOLO';
$query = sprintf("SELECT * FROM studenti WHERE NOME =%s",
filterSQLQuery($name));
 $result = $mysqli->query($query);
 checkResult($result,$query);
 showResult($result);

 $mat  = 3333;
 $query = sprintf("SELECT * FROM studenti WHERE MATRICOLA
 =%s",  filterSQLQuery($mat));
 $result = $mysqli->query($query);
 checkResult($result,$query);
 showResult($result);
```

PHP and mysql

```
function filterSQLQuery($string) {
  if (get_magic_quotes_gpc())
 /*When magic_quotes are on for GPC (Get/Post/Cookie) operations,
all ', ", \ and NUL's are escaped with a backslash automatically –
This feature has been removed in PHP 5.4.0 and the function returns
always false*/
        $string = stripslashes($string);
        // removes slashes (\)
if (!is_numeric($string))
        $string = """ . mysqli_real_escape_string($string) . """;
/*Escapes special characters in a string for use in an SQL statement
in such a way that the SQL statements are sure ;*/
return $string }
```

1111 ROMOLO

3333 CESARE

SQL Injection Attack

- Exploiting SQL code is possible to obtain private information or modify a database

Esempio

Let us assume to have a login form which asks for username and passwd

These data are used to perform sql query:

```
$miaQuery=“SELECT COUNT(*) FROM utenti
  WHERE username=‘ ” .$_POST[‘username’]. “ ’
  AND password=‘ ” .$_POST[‘password’].  “ ’ ”;
```

username
password

SQL Injection Attack

username	GIANNI	password	‘ OR ‘’=‘

Input the following data:

```
SELECT COUNT(*)
WHERE username=‘GIANNI’
AND password= ' ' OR ' ' = ' ';
```

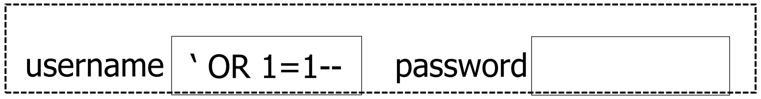
The query results to be:

```
SELECT COUNT(*) FROM utenti
WHERE username='GIANNI'
AND password= ' ' OR ' ' = ' ';
```

Same effect as knowing the GIANNI’s password. By using mysqli_real_escape_string, each quote is escaped.

SQL Injection Attack

Other examples:



username ` OR 1=1-- password

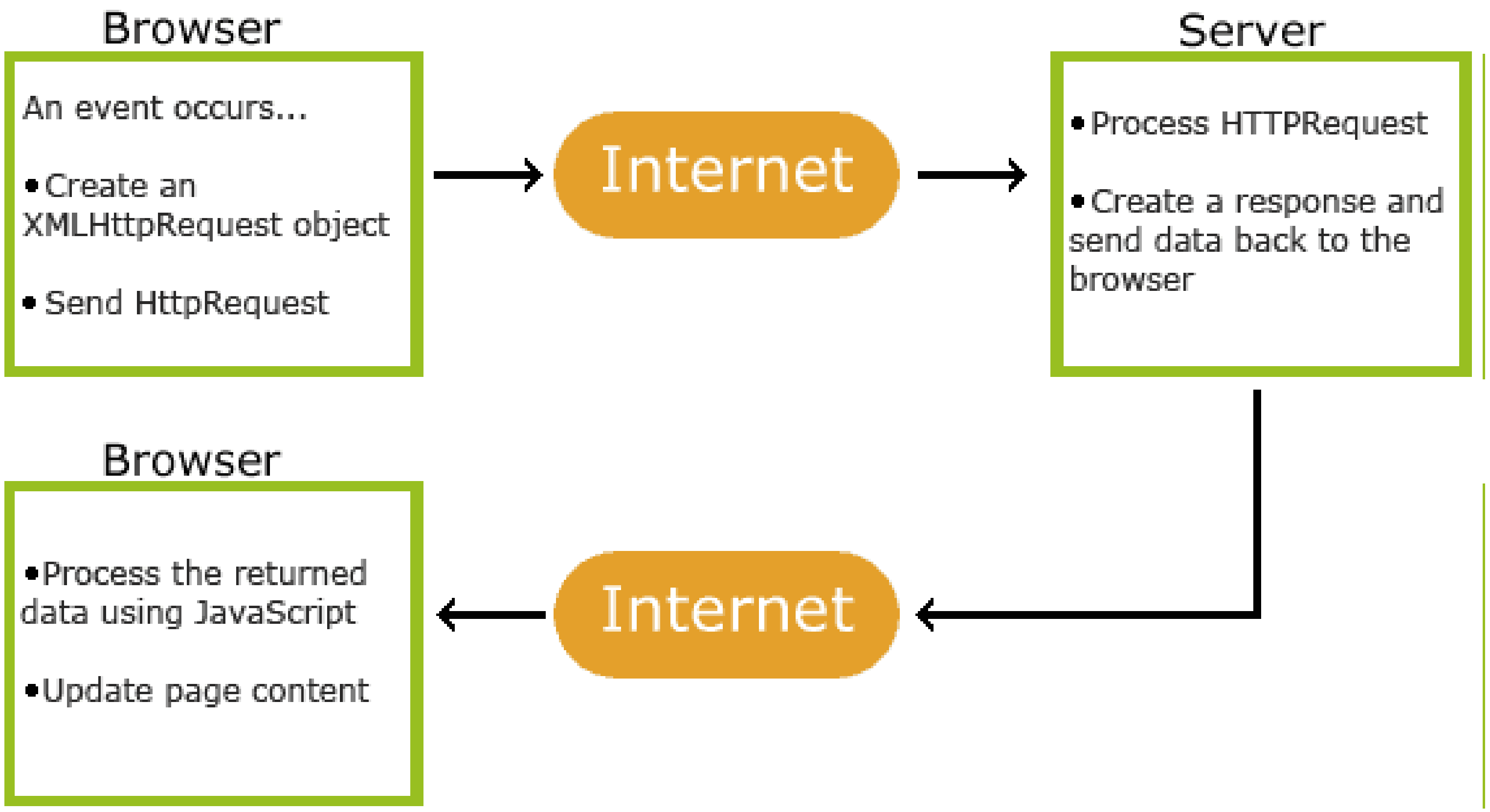
SELECT COUNT(*)
WHERE username= ' ' OR 1=1 --'
AND password= ' `;

The text (in italic) after -- is not considered since it indicates an SQL comment.
The query will return the number of rows in the table.

AJAX

- Asynchronous JavaScript and XML (AJAX) is a group of interrelated web development techniques used on the client-side to create interactive web applications.
- With Ajax, web applications can retrieve data from the server asynchronously in the background without interfering with the display and behavior of the existing page.
- Instead of using normal HTTP POST or GET requests (such as when using forms or hyperlinks), data is usually retrieved using the XMLHttpRequest object

AJAX


Browser
An event occurs...
• Create an XMLHttpRequest object
• Send HttpRequest
Internet
Server
• Process HTTPRequest
• Create a response and send data back to the browser
Browser
• Process the returned data using JavaScript
• Update page content
Internet

AJAX

- XMLHttpRequest object
  - used to send HTTP or HTTPS requests directly to a web server and load the server response data directly back into the script.
  - The data might be received from the server as XML text or as plain text.
  - Data from the response can be used directly to alter the DOM of the currently active document in the browser window without loading a new web page document.

AJAX

Three steps

- Create an XMLHttpRequest object

```
try { xmlHttp=new XMLHttpRequest(); }
catch (e) {
     try { xmlHttp=new ActiveXObject("Msxml2.XMLHTTP"); }
    //IE (recent versions)
    catch (e) {
    try { xmlHttp=new ActiveXObject("Microsoft.XMLHTTP"); }
    //IE (older versions)
      catch (e) {
       window.alert("Your browser does not support AJAX!");
    return false;
                  }
       }
    }
```

AJAX

- Write an event handler for sending requests for data to the server

```
xmlHttp.open("GET","data.php",true);
xmlHttp.onreadystatechange = useHttpResponse;
xmlHttp.send(null);
```

open(method,url,async)

Specifies the type of request, the URL, and if the request should be handled asynchronously or not.

method: the type of request: GET or POST

url: the location of the file on the server

async: true (asynchronous) or false (synchronous)

onreadystatechange

Stores a function (or the name of a function) to be called automatically each time the readyState property changes

AJAX

send(string)

Sends the request off to the server.

string: Only used for POST requests

Example:

```
xmlhttp.open("POST","ajax_test.php",true);
xmlhttp.setRequestHeader("Content-type","application/x-www-form-urlencoded");
xmlhttp.send("fname=Henry&lname=Ford");
```

setRequestHeader(header,value)

Adds HTTP headers to the request.

header: specifies the header name

value: specifies the header value

AJAX

- Write a function useHttpResponse which will establish when the server has completed the request and will do something useful with the returned data

```
function useHttpResponse() {
 if (xmlHttp.readyState == 4)
   document.mymodule.day.value = xmlHttp.responseText;
}
```

readyState

Holds the status of the XMLHttpRequest. Changes from 0 to 4:

0: request not initialized

1: server connection established

2: request received

3: processing request

4: request finished and response is ready

AJAX

responseText

get the response data as a string

responseXML

get the response data as XML data

AJAX

- Example (submit.html)

```
<html lang="en">
 <head>
  <meta charset="utf-8">
<title>Example</title>
</head>
<body>
<script type="text/javascript" src="./myajax.js"></script>
<form action="#" name="mymodule" id="mymodule">
<p>Name: <input type="text" onkeyup="ajaxHandler();"
  name="name">
  Date: <input type="text" name="day"></p>
</form>
</body>
</html>
```

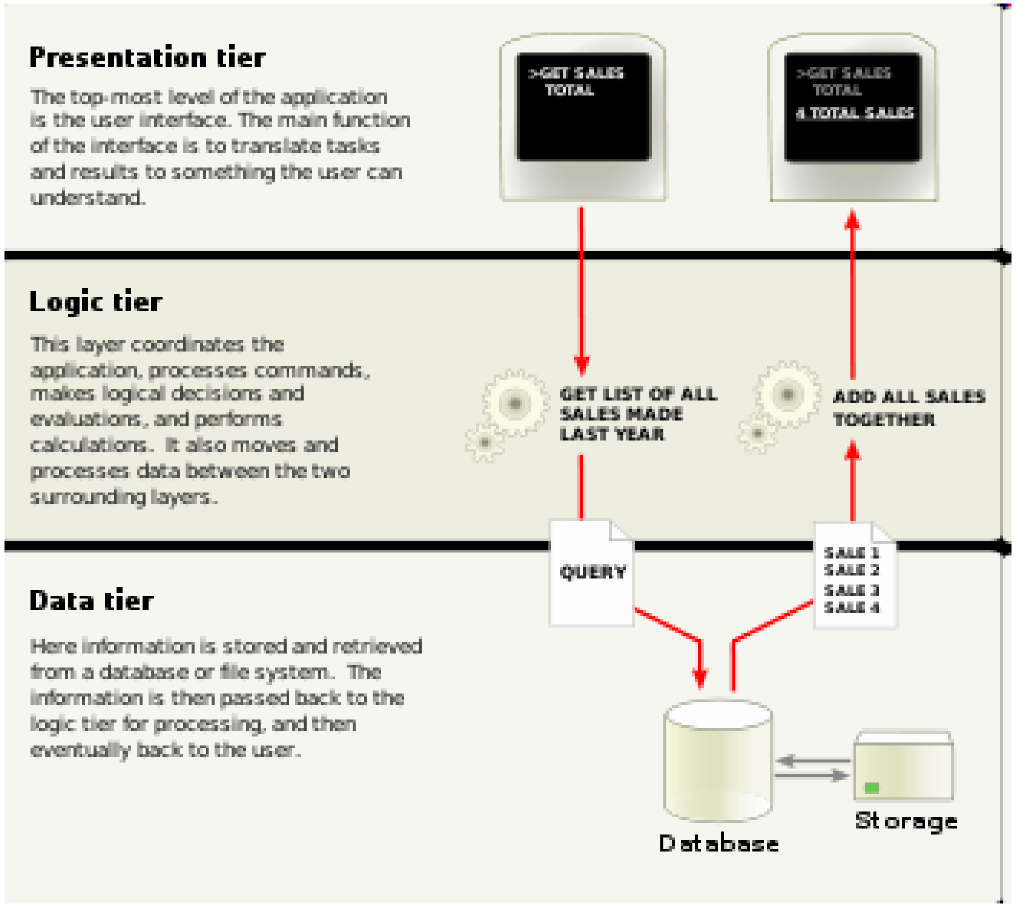
AJAX

```
\\date.php
<?php    $oggi = date("D M Y G:i:s");   echo($oggi); ?>

\\myajax.js
function ajaxHandler() {
     var xmlHttp;
     try { xmlHttp=new XMLHttpRequest(); }
     catch (e)
    {       try { xmlHttp=new ActiveXObject("Msxml2.XMLHTTP"); }
            catch (e)
            {   try { xmlHttp=new ActiveXObject("Microsoft.XMLHTTP"); }
                 catch (e)
                {       window.alert("Your browser does not support AJAX!");
                        return;
                 }
             }
        }
```

AJAX

```
xmlHttp.open("GET","date.php",true);
xmlHttp.onreadystatechange = useHttpResponse;
xmlHttp.send(null);

function useHttpResponse()
{       if(xmlHttp.readyState == 4)
        document.mymodule.day.value = xmlHttp.responseText;
}
}
```

Three-tier architecture



Presentation tier

The top-most level of the application is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.

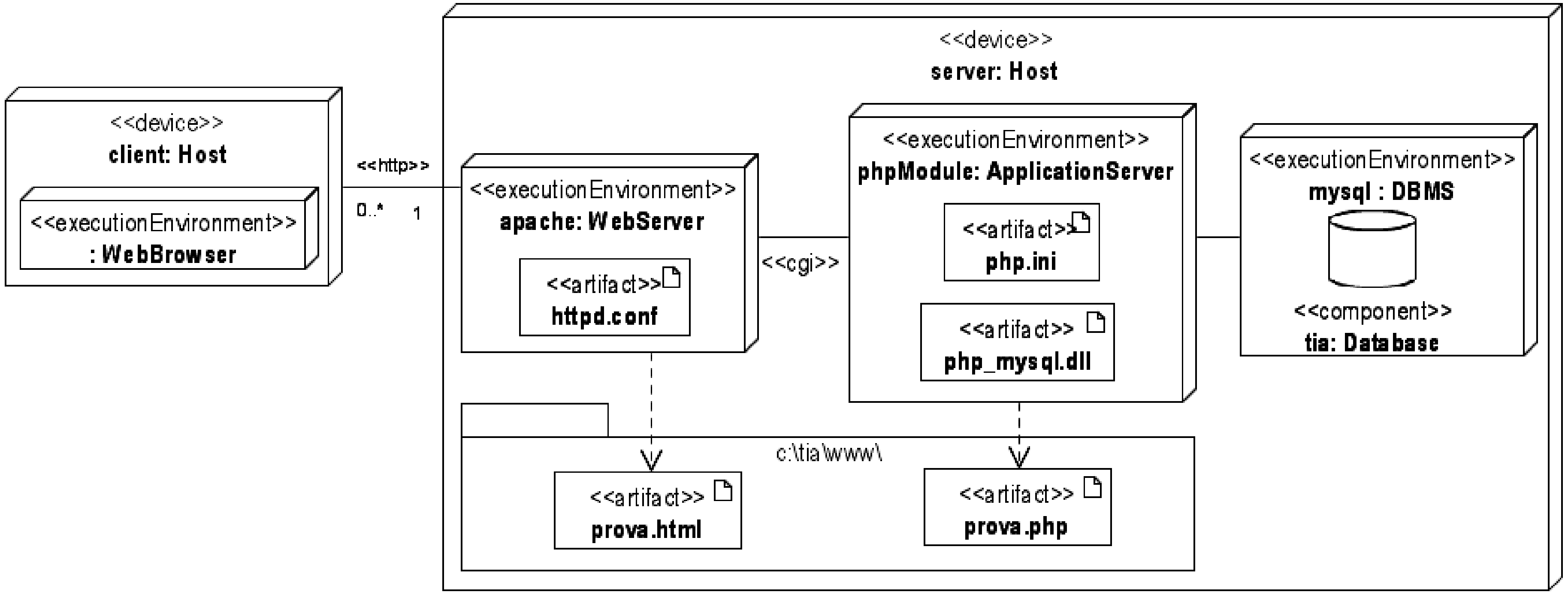
Logic tier

This layer coordinates the application, processes commands, makes logical decisions and evaluations, and performs calculations. It also moves and processes data between the two surrounding layers.

Data tier

Here information is stored and retrieved from a database or file system. The information is then passed back to the logic tier for processing, and then eventually back to the user.

Three-tier architecture UML - Deployment Diagram



Three-tier architecture

- The Common Gateway Interface (CGI) is a standard (see RFC 3875: CGI Version 1.1) that defines how web server software can delegate the generation of web pages to a text-based application.
- CGI is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user.
- httpd.conf – configuration file for determining for instance the root
- php.ini – configuration file for selecting the DBMS and the libraries

Three-tier architecture

- In the web development field, three-tier is often used to refer to websites, commonly electronic commerce websites, which are built using three tiers:
  - A front-end web server serving static content, and potentially some cached dynamic content. In web based application, Front End is the content rendered by the browser. The content may be static or generated dynamically.
  - A middle dynamic content processing and generation level application server, for example Java EE, ASP.NET, PHP platform.
  - A back-end database, comprising both data sets and the database management system or RDBMS software that manages and provides access to the data.