Introduction to PHP

Last updated: Dec, 2012

Introduction

The following is a quick introduction and summary of many aspects of the PHP language for those who have some programming experience. Although this overview is not intended to be an exhaustive examination of PHP, it is comprehensive enough for you to get started building non-trivial web applications with PHP. See the official PHP manual for more detailed information: http://www.php.net/manual/en/. All syntax contained in this guide is for PHP 5 and may not be compatible with previous versions of PHP.

Background

- Nerdy recursive acronym: PHP: Hypertext Preprocessor (originally named Personal Home Page Tools)
- Invented by Rasmus Lerdorf in 1994 and is now under the Apache Software Foundation. Licensed under the GPL and is free. Current version as of October 2012 is PHP 5.4.8.
- Popular server-side technology for Apache web servers. Competing technologies include Oracle's JavaServer Pages, Microsoft's ASP.NET, and Adobe's ColdFusion.
- Available on a variety of web servers (Apache, IIS, NGINX, etc.) and operating systems (Windows, Linux, UNIX, Mac OS X, etc.).
- Supports many types of databases: MySQL, Oracle, ODBC (for MS Access and SQL Server), SQLite, etc.

On-line Resources

www.php.net – PHP distribution, tutorials, newsgroups, and more.
www.phpfreaks.com - PHP and MySQL tutorials, scripts, forums, and more.
www.phpbuilder.com – Collection of PHP resources.

Hello World

If your web server supports PHP, type this example into a text file called hello.php and access it in your browser by typing the complete URL (e.g., http://www.example.com/hello.php). Depending on how your web server is configured, your .php file will need the proper permissions so the web server can access and execute the PHP script.

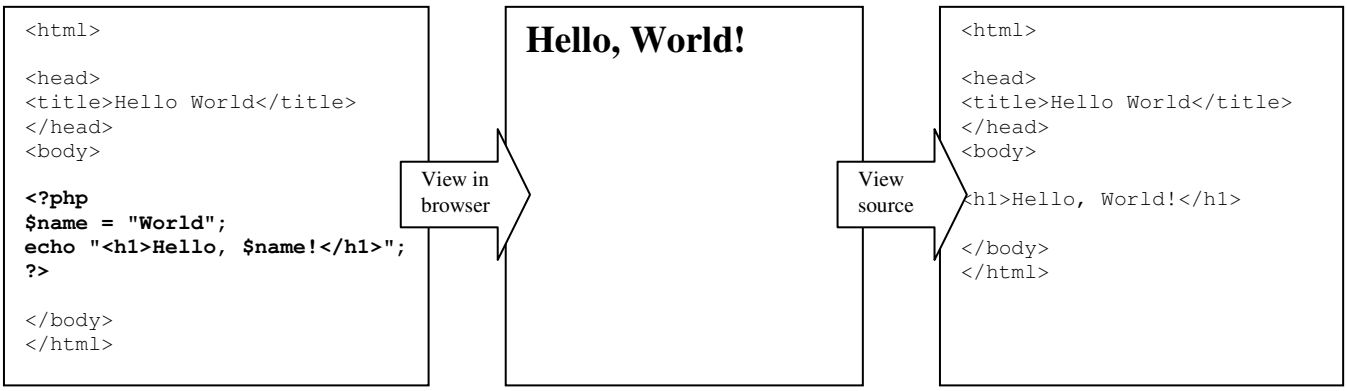

Table of Contents

I. Some Basics

A. PHP is a *scripting* language – it gets *interpreted* instead of being *compiled* like C++ and Java.

B. Unlike JavaScript which is executed by the *web browser*, all PHP code is executed on the *web server*.

C. The syntax is very similar to Perl and C. Variables are case sensitive, function names are not, and statements must be terminated with a semicolon.

D. PHP code should be placed between `<? code ?>` or `<?php code ?>` tags. The second method is preferred so your scripts are XML compatible. There is no limitation as to where PHP code can be inserted.

E. To see information about how PHP is configured, version information, and the settings of all environment variables (e.g., HTTP_USER_AGENT and QUERY_STRING), call the **`phpinfo()`** function in any script.

F. The **php.ini** file is the main configuration file for PHP. It can be edited by the system administrator to change any of the configuration settings. A change to this file requires the web server be restarted since the file is only read once when the web server starts up. (The `phpinfo()` function reports the location of php.ini on the server.)

G. It's a good idea to turn on error and warning output when developing your code so you don't misuse PHP syntax in unintended ways. Place the following lines of code at the top of your script so errors will be reported in the rendered web page:

```
ini_set('display_errors', '1');
error_reporting(E_ALL | E_STRICT);
```

Note that if the php.ini file already has these settings, you don't need to use these lines of code.

II. Comments

The three following styles are legal:

```
# Perl style single line comment

// Single line comment
```

```
/* Multiple
   line comments */
```

III. Variables and Data Types

A. Always starts with $ and letter or underscore. Can be composed of numbers, underscores, and letters.

```
$my_var = 10;
$a_2nd_var = "bison";
```

B. Data types: integers, doubles (numbers with a decimal point), boolean (true or false), NULL, strings, arrays, objects, and resources (like database connections). Variables do not have to be declared and neither do their data types.

C. Variables have a default value (0, empty string, false, or empty array) if they aren't initialized before trying to use them. It's always good practice to initialize all variables rather than relying on the default initialization value. If you try to use a variable before setting it to a value, strict error-reporting setting will give you an "Undefined variable" warning.

D. All variables have local scope (i.e., they are accessible only within the function or block in which they are initialized). Global variables may only be accessed within a function by using the `global` keyword.

```
$x = "test";

function display() {
   global $x;
   echo $x;
}
```

E. Constants are defined using **`define`** and by convention are usually named in ALL CAPITALS.

```
define("PI", 3.14);
define("HEADING", "<h1>My Web Site</h1>");
$area = PI * $radius * $radius;
print(HEADING);
```

IV. Operators

A. Assignment
   1. `=` `+=` `-=` `/=` `*=` `%=` `++` `--` - like most programming languages.
   2. `.=` - string concatenation operator (see strings section).

B. Arithmetic
   1. `+` `-` `*` `/` `%` - like most programming languages.

C. Comparison
   1. `==` `!=` `<` `>` `<=` `>=` - like most programming languages. Also `<>` is the same as `!=`.
   2. `===` - true if arguments are equal and the same data type.
   3. `!==` - true if arguments are not equal or they are not of the same data type.

D. Logical
   1. `&&` `||` `!` - like most programming languages (`&&` and `||` short-circuit)
   2. `and` `or` - like `&&` and `||` but have lower precedence than `&&` and `||`.
   3. `xor` - true if either (but not both) of its arguments are true.

V. Input/Output

A. `print` and `echo` are used to print to the browser.

```
echo "Go Bisons";
echo("Go Bisons");   // same thing
print("Go Bisons");  // same thing
```

B. `print` can only accept one argument, and `echo` can accept any number of arguments. `print` returns a value that indicates if the print statement succeeded.

C. Variables are interpolated inside of strings unless single quotes are used.

```
$a = "guts";
echo "You have $a.";   // prints "You have guts."
echo 'You have $a.';   // prints "You have $a."
```

D. Escape sequences: \n (newline), \r (carriage-return), \t (tab), \$ ($), \" ("), \\ (\)

```
echo "a\\b\tc\$d";   // prints "a\b      c$d"
echo 'a\\b\tc\$d';   // prints "a\b\tc\$d".  Only \\ is converted.
```

E. printf works like C's counter-part.

```
$title = "X-Men";
$amount = 54.235;
printf("The movie <b>%s</b> made %2.2f million.", $title, $amount);
// prints "The movie <b>X-Men</b> made 54.23 million."
```

F. PHP typically does not run from the command-line, but input from the keyboard can be accessed using the fopen function with "php://stdin". See the file I/O section for more information.

G. Output shortcut from within HTML:

```
Hello, <b><?= $name ?></b>
```
is the same as
```
Hello, <b><?php echo $name ?></b>
```

VI. Control Structures

A. Choice structures

1.
```
if ($x > 0)
  $y = 5;      // {} not required for only one statement
```

2.
```
if ($a) {      // tests if $a is true or non-zero or a non-empty string
  print($b);
  $b++;
}
else
  print($c);
```

3.
```
if ($a > $b)
    print "a is bigger than b";
elseif ($a == $b)                    // use "elseif" or "else if"
    print "a is equal to b";
else
    print "a is smaller than b";
```

4.
```
switch ($vehicle_type) {     // works for integers, floats, or strings
  case "car":   $car++;    break;
  case "truck": $truck++;  break;
  case "suv":   $suv++;    break;
  default:      $other++;
}
```

B. Looping structures

1.
```
while ($n < 10) {
  print("$n ");
  $n++;
}
```

2.
```
do {
  print("$n ");
  $n++;
} while ($n < 10);
```

3.
```
for ($n = 1; $n < 10; $n++)
  print("$n ");
```

4.
```
foreach ($myarray as $item)
  print("$item ");
```

VII. Arrays

A. Summary of all array functions in the PHP core: http://www.php.net/manual/en/ref.array.php

B. Arrays can have any size and contain any type of value. No danger of going beyond array bounds.

```
$my_array[0] = 25;
$my_array[1] = "Bisons";
```

C. PHP arrays are **associative arrays** which allow element values to be stored in relation to a key value rather than a strict linear index order.

```
$capitals["CO"] = "Denver";
$capitals["AR"] = "Little Rock";
```

D. Initialize an array:

```
$colors = array("red", "green", "blue");
print("The 2nd color is $colors[1].");    // prints green

$capitals = array("CO" => "Denver", "AR" => "Little Rock");
print("$capitals[CO]");   // prints Denver, no quotes around key inside ""
```

E. Print contents of an array for debugging:

```
print_r($colors);
```

produces:

```
Array
(
    [0] => red
    [1] => green
    [2] => blue
)
```

```
print_r($capitals);
```

produces:

```
Array
(
    [CO] => Denver
    [AR] => Little Rock
)
```

F. Pull values out of an array:

```
$colors = array("red", "green", "blue");
list($c1, $c2) = $colors;
print("$c1 and $c2");   // prints "red and green"
```

G. Delete from an array:

```
unset($colors[1]);  // $colors now contains red and blue at indexes 0 and 2.
```

H. Extracting array keys and values:

```
$states = array_keys($capitals);    // $states is ("CO", "AR")
$cities = array_values($capitals);  // $cities is ("Denver", "Little Rock")
```

I. Iterating through an array:

```
$heroes = array('Spider-Man', 'Hulk', 'Wolverine');
foreach ($heroes as $name)
  print("$name<br />");             // prints all three in order

foreach ($capitals as $state => $city)
  print("$city is the capital of $state.<br />");
```

J. Treat an array like a stack:

```
array_push($heroes, 'Iron Man');    // Pushed onto end of array
$heroes[] = 'Captain America';      // Same thing as array_push
$h = array_pop($heroes);            // Pops off last element (Iron Man)
```

K. Size of an array:

```
$num_items = count($heroes);   // returns 3
```

L. Sort an array:

```
sort($heroes);    // Heroes are now in alphabetical order (lowest to highest)
rsort($heroes);   // Reverse alphabetical order (highest to lowest)
```

VIII. Functions

A. PHP pre-defined functions are documented at http://www.php.net/manual/en/funcref.php .

B. Functions may be declared anywhere in the source code (i.e., they do not need to be defined before they are called as C++ requires).

C. Function names are case-insensitive, though it is usually good form to call functions as they appear in their declaration.

D. Defining and calling

1. General form:

```
function func_name($param_1, $param_2, ..., $param_n) {
    // code
    return $retval;  // optional: can return a scalar or an array
}
```

2. Call: `$result = func_name($arg1, $arg2, ..., $argn);`

E. Parameter passing and returning values

1. Arguments may be passed by value (default) or by reference (using &). Default argument values can also be used which must be initialized in the parameter list. Variable-length argument lists are also supported but are not covered here.

```
// Pass by value
function sum($a, $b) {
  return $a + $b;
}

// Pass by reference
function swap(&$a, &$b) {
  $temp = $a;
  $a = $b;
  $b = $temp;
}
```

```
// Default arguments must be on right side
function say_greeting($name, $greeting="Hello") {
  print "$greeting, $name!";
}

say_greeting("Susan");          // Hello, Susan!
say_greeting("Rita", "Hola");   // Hola, Rita!
```

2. Passing an array by value and by reference

```
// Pass by value
function sum_array($values) {
  $sum = 0;
  foreach ($values as $num)
    $sum += $num;
  return $sum;
}

$nums = array(1, 2, 3);
print "Sum of array = " .
  sum_array($nums); // 6
```

```
// Pass by reference
function randomize(&$nums) {
  for ($i = 0; $i < 10; $i++)
    $nums[$i] = rand(0, 100);  // 0-100
}

$n = array();
randomize($n);   // Place 10 random nums in $n
```

3. Return an array

```
// Return an array
function special_nums() {
  return array(3.142, 2.718, 1.618);
}
```

```
list($pi, $euler, $phi) = special_nums();
```

IX. Strings

A. Concatenation

```
$full_name = $first_name . " " . $last_name;  // results in "Bob Smith"
```

B. Some PHP string functions. View the complete list at http://www.php.net/manual/en/ref.strings.php

`int` **`strlen`**`($str)` Returns string length.

`int` **`strcmp`**`($str1, $str2)`
Returns < 0 if str1 is less than str2; > 0 if str1 is greater than str2, and 0 if they are equal. (strcasecmp for case-insensitive comparison.) The < > == operators can also be used if both arguments are strings. strcmp is useful if an argument may not be a string and has to be converted into one.

`string` **`strstr`**`($text, $search)`
Returns first occurrence of $search in $text, FALSE if not found. (stristr for case-insensitive search.)

`string` **`str_replace`**`($find, $replace, $text)`
Replaces all occurrences of $find with $replace in $text.

`string` **`chop`**`($str)` Removes all white space at end of string.
`string` **`ltrim`**`($str)` Removes all white space at beginning of string.
`string` **`trim`**`($str)` Removes all white space at beginning and end of string.

X. Regular Expressions

A. Regular expressions are patterns that can be used to match text in a string. They can be used, for example, to determine if a string contains a legal email address or phone number. PHP regular expressions are implemented very similarly in other programming languages. For a complete reference, see http://us2.php.net/manual/en/ref.pcre.php.

B. The examples here use Perl regular expressions which require forward slashes ("/") around the pattern.

C. Matching patterns

1. Find the given pattern anywhere in the string

```
if (preg_match("/ard/", "Harding"))
  echo "Matches";
else
  echo "No match";
```

2. Special symbols

`\d`	any digit (0-9)
`\s`	any white space (space, tab, EOL)
`\w`	any word char (a-z, A-Z, 0-9, _)
`.`	any character except EOL
`[abc]`	a, b, or c
`[^a-z]`	not any char between a and z
`{3}`	match only three of these
`?`	match zero or one character
`*`	match zero or more characters
`+`	match one or more characters
`^abc`	match at the beginning of the string
`abc$`	match at the end of the string

3. Email address example

```
$email = 'first_name.last_name@domain.Com';
$regexp = "/^[\w.]+@[\w.]+[a-z]{2,4}$/i";    // i switch for case-insensitive match
if (preg_match($regexp, $email))
  echo "Match email";
```

4. Remembering matched patterns

```
if (preg_match('/(\d\d):(\d\d) (am|pm)/', '03:15 pm', $matches)) {
  echo "Hour: $matches[1]\n";    // 03
  echo "Min: $matches[2]\n";     // 15
  echo "Ending: $matches[3]\n";  // pm
}
```

5. Match all occurrences

```
preg_match_all('/.ar/', 'the car was far from the bar', $matches);
print_r($matches[0]);  // Prints car, far, bar
```

D. Replacing patterns

1. Simple replacement

```
$new = preg_replace('/hard/', 'easy', 'this is hard!');   // Returns "this is easy!"
```

2. Replacement with remembered matches

```
// Convert MM/DD/YYYY to YYYY-MM-DD (must escape / in regex)
$date = preg_replace("/(\d\d)\/(\d\d)\/(\d\d\d\d)/", "$3-$1-$2", "08/15/2008");
```

E. Array processing

1. Split string into an array

```
$names = 'Fred, Erin, Alex, Sunshine';
$names_array = preg_split('/[, ]+/', $names);  // Returns one name in each slot
```

2. Find all items in an array that match a regex

```
// Returns only Erin and Alex
$starts_with_vowel = preg_grep('/^[aeiou]/i', $names_array);
```

XI. Exception Handling

A. PHP uses exception (error) handling much like C++, Java, and many other HLLs.

```
function divide($x, $y) {
    if ($y == 0)
        throw new Exception('Division by zero.');
    else
        return $x / $y;
}
```

```
try {
    echo divide(5, 2) . "\n";    // Prints 2.5
    echo divide(4, 0) . "\n";    // Causes exception to be thrown
}
catch (Exception $e) {
    echo 'Caught exception: ',  $e->getMessage(), "\n";
}
```

B. Exceptions that are not caught cause the script to fail with a fatal error.

C. Other information about the exception like the line number and the file name in which the exception occurred is also available. See http://us.php.net/manual/en/language.exceptions.php.

XII. File I/O

A. PHP can access any file that is stored on the web server, as long as it has the proper permissions.

B. HTTP, FTP, and STD read/write can also be used with file functions.

C. See http://www.php.net/manual/en/ref.filesystem.php for functions that set file permissions, copy and delete files, access file modification times, and a lot more.

D. Open file with `fopen`, close with `fclose`. File open modes:
1. "r" – Read from existing file.
2. "r+" – Read and write to already existing file.
3. "w" – Write to file, creating the file if it doesn't already exist, erasing it if it does.
4. "w+" – Write and read, creating the file if it doesn't already exist, erasing it if it does.
5. "a" – Append to end of file whether it exists or not.
6. "a+" – Append to end of file, doubling file contents if you read the file in as a string, edit it, and write it back to the file.

E. Reading from a file. File must have proper read permissions. If file is not owned by "nobody", it must have world read permissions.

1. Read entire file:

```
$fd = fopen("myfile.txt", "r") or die("Can't open myfile.txt for reading.");
$entire_file = fread($fd, filesize("myfile.txt"));
print $entire_file;
fclose($fd);
```

*The `die` operator is useful for halting executing of a PHP script and giving an error message. You may also call `exit` which does the same thing.

2. Read line by line:

```
while (!feof($fd)) {
  $line = fgets($fd, 4096);  // 4096 is the max bytes per line
  print "$line<br />";
}
```

F. Writing to a file. The directory containing the file to be written must have at least world execute and write permissions. Default owner of a created file will be "nobody".

```
$fd = fopen("myfile.txt", "w") or die("Can't write to myfile.txt.");
fwrite($fd, "This is output.");
fclose($fd);
```

XIII. Importing Scripts and HTML Files

A. A file containing HTML and/or PHP code can be imported into another PHP script by using the **require** statement. If the file being included cannot be found, the script halts with a fatal error.

heading.php

```
<h3>Date: <?php $today = date("D M d Y"); echo $today; ?></h3>
```

The code above can be used in a PHP script like this (assuming it resides in the same directory):

```
<?php
   require 'heading.php';  // Prints Date: Fri Aug 29 2008
?>
```

B. The **include** statement does the same thing, but the script does not produce a fatal error if the included file is not found.

C. The **require_once** and **include_once** statements do the same thing as require and include, but they will not reload a file that has already been included.

XIV. Web Form Input

A. Data from web forms can be accessed using the superglobal arrays **$_GET** and **$_POST**.

B. **$_GET**: for accessing data in the **query string** – the key=value pairs that appear in the URL after the "?" character. All values are unescaped (+ is converted to space, etc.).

Example: http://www.example.com/myscript.php?**custname=Bob+Smith&custage=21**

```
$name = $_GET["custname"];     // $name is "Bob Smith"
$age = $_GET["custage"];       // $age is 21
```

C. **$_POST**: for accessing posted for data from **standard input** (values are unescaped).

Example STDIN: custname=Bob+Smith&custage=21

```
$name = $_POST["custname"];     // $name is "Bob Smith"
$age = $_POST["custage"];       // $age is 21
```

D. It's always a good idea to use **isset** to check if the variable exists in $_POST and $_GET before accessing to avoid PHP warnings:

```
if (!isset($_POST["custname"]) || trim($_POST["custname"]) == "")
    echo "The customer's name was left blank.";
```

E. Shortcut to accessing variables in $_GET and $_POST: **extract** puts all key/value pairs in identically named variables. Warning: collisions occur between identically named variables.

```
extract($_POST);
if (isset($custname))
    echo "Hello, $custname!";
```

XV. Maintaining State

A. To keep track of data between HTTP requests, data can be stored in cookies using the **$_COOKIE** array, or it can be stored on the web server via session variables in the **$_SESSION** array.

B. setcookie() and session_start() functions below *must* be called before any other output is produced unless output_buffering is turned on in php.ini.

C. **$_COOKIE** – for accessing HTTP cookies which are stored on the client and are transmitted back to the web server in every HTTP request.

1. By default, cookies expire with the session. Closing the browser (not just the tab) ends the session.

```
setcookie("age", "21");
```

2. Cookies can be given expiration dates so they persist even after the browser is closed:

```
// Cookie expires in 24 hours
setcookie("name", "Betty", time() + 60 * 60 * 24);
```

3. Getting the value of a cookie:

```
echo $_COOKIE["name"];     // Print contents of name cookie
```

Note that $_COOKIE will not have values set from setcookie() until the php script is requested *after* a call to setcookie.

4. Deleting a cookie:

```
setcookie("name", FALSE);
```

5. *Warning:* Since cookies are stored on the client, they should not be used to store sensitive data.

D. **$_SESSION** – for accessing session variables which are stored on the web server. Variables are associated with a unique session ID which is stored in a cookie or passed in the URL if cookies are disabled. This technique is ideal for storing sensitive data since the data is not stored on the client.

1. Create a session (and a session ID) if one doesn't already exist:

```
session_start();
```

2. If you are curious what the session ID looks like:

```
echo session_id();
```

3. Set a session variable that is stored on the web server and tied to the user's session ID:

```
$_SESSION["name"] = "Betty";
```

4. Get the value of a session variable that is tied to the user's session ID:

```
$auto = $_SESSION["name"];
```

5. Session variables are cleared when the browser is closed or when destroying the session explicitly:

```
session_destroy();
```

XVI. Uploading Files

A. Files may be uploaded to the web server from the browser by using a specially-encoded form and the **$_FILES** superglobal array.

```
<form method="POST" action="upload.php" enctype="multipart/form-data">
    <!-- max files size in bytes -->
    <input type="hidden" name="MAX_FILE_SIZE" value="5000000" />
    <input type="file" name="myfile" /><br />
    <input type="submit" value="Submit" />
</form>
```

When submitted to upload.php, the **$_FILES** array contains an entry for each file uploaded.

```
print_r($_FILES);
```

produces

```
Array
(
    [myfile] => Array
        (
            [name] => archive-3.png
            [type] => image/png
            [tmp_name] => /tmp/phpUajj5s
            [error] => 0
            [size] => 131377
        )
)
```

Name of file on client
File's MIME type as determined by file extension
Temporary path of file stored on the server
Number indicating an error (0 means no error)
Size of file in bytes

You must copy the file from the temporary location (`$_FILES['myfile']['tmp_name']`) to a more permanent location if you wish to store the file contents for later use. A complete function that does this and more can be obtained from http://frankmccown.blogspot.com/2009/04/upload-image-in-php.html.

XVII. Miscellaneous Features

A. Other superglobals include:

1. **$GLOBALS** - all variables which are currently defined in the global scope of the script.
2. **$_ENV** – for accessing data about the PHP parser's environment.
3. **$_REQUEST** – contains all variables in $_GET, $_POST, and $_COOKIE
4. **$_SERVER** – for accessing web server environment variables like REQUEST_METHOD and HTTP_USER_AGENT.

```
if ($_SERVER["REQUEST_METHOD"] == "GET") { ... }

echo "Your browser is $_SERVER[HTTP_USER_AGENT]";
```

B. Setting HTTP headers in an HTTP response is possible using the **header()** function. Note: It must be called *before* any output is sent (normal HTML, blank lines, or PHP).

1. Send a 302 redirect:

```
header('Location: http://www.example.com/');
```

2. Respond with a custom 404 page:

```
header("HTTP/1.0 404 Not Found");
echo "We're sorry, but this web page could not be found.";
```

3. Prompt the user to save a PDF file:

```
header('Content-type: application/pdf');
header('Content-Disposition: attachment; filename="new_name.pdf"');
readfile('orig.pdf');
```

C. Helpful functions:

1. URL-encode a string:

```
echo '<a href="my.php?test=', urlencode('one + two = three'), '">link</a>';
```

produces:

```
<a href="my.php?test=one+%2B+two+%3D+three">link</a>
```

2. Convert special characters to HTML entities:

```
echo htmlspecialchars("one & <b>two</b>");   // one &amp; &lt;b&gt;two&lt;/b&gt;
```

XVIII. Classes and Objects

A. PHP supports many object-oriented programming concepts like constructors, destructors, abstract classes and methods, interfaces, dynamic creation of members and methods, etc. For a complete discussion, see http://www.php.net/manual/en/language.oop5.php.

B. Declare a base class with a constructor and destructor. The `__toString()` method is useful for serializing the object to a string.

```
class SuperPerson
{
  public $Name;           // Accessible to anyone
  public $PowerLevel;

  // Constructor
  public function __construct($Name, $PowerLevel = 0) {
    $this->Name = $Name;
    $this->PowerLevel = $PowerLevel;
  }

  // Destructor called when object goes out of scope
  function __destruct() {
    echo "Bye-bye";
  }

  // Convert object to string representation
  public function __toString() {
    return "Name = $this->Name, PowerLevel = $this->PowerLevel\n";
  }
}
```

C. Extend the base class from above (SuperPerson) to create a super hero:

```
class SuperHero extends SuperPerson
{
  private $savedVictims = 0;          // Accessible only within the class

  public function Save($victim) {
    echo "$this->Name is saving $victim.\n";
    $savedVictims++;
  }

  public function GetNumberOfSavedVictims() {
    return $savedVictims;
  }
}
```

D. Declaring and using objects:

```
$hero = new SuperHero("Spam-Man", 3);
echo "$hero";          // Prints Name = Spam-Man, PowerLevel = 3
$hero->Save("Laura Jones");
```

XIX. Database Access - MySQL

A. PHP supports most popular databases including MySQL, Oracle, MS Access, SQL Server, SQLite, etc.

B. Many PHP developers use MySQL (http://www.mysql.com/) because of its cost (free in most cases) and durability.

C. Detailed guide to MySQL Improved Extension (MySQLi) at http://www.php.net/manual/en/book.mysqli.php

D. You must first connect to the MySQL server and select your database before executing any database operations:

```
// Create a mysqli object which connects and selects
$mysqli = new mysqli($hostname, $username, $password, $db_name);

// Output error info if there was a connection problem
if ($mysqli->connect_errno)
   die("Failed to connect to MySQL: ($mysqli->connect_errno) $mysqli->connect_error");
```

E. CRUD database operations:

1. Query the database – Using the SELECT statement

```
$sql = "SELECT ID, Name FROM Students WHERE GPA >= 2.0";
$result = mysqli->query($sql) or die("Error $mysql->errno $mysqli->error " .
    "<br>SQL = $sql<br>");

// Loop through all rows returned by the query
while ($row = $result->fetch_row())
    echo "ID is $row[0] and name is $row[1]<br />\n";
```

```
// Same thing as above while loop but using an associative array
while ($row = $result->fetch_assoc())
    echo "ID=$row[ID]  name=$row[Name]<br />\n";


// Test to see if a single row is returned or not
$sql = "SELECT Name FROM Students WHERE ID = 123";
$result = $mysqli->query($sql) or die($mysqli->error);

if ($mysqli->num_rows == 0)
   echo "Student not found.";
else {
   $row = $result->mysql_fetch_assoc();
   echo "Hello, $row[Name]!";
}
```

2. Insert a new record into the database – Using the INSERT statement

```
$sql = "INSERT INTO Students VALUES (789, 'Jane', 2.5)";
$mysqli->query($sql) or die($mysqli->error);
$rows_inserted = $mysqli->affected_rows;  // Should return 1
echo "Successfully inserted $rows_inserted row.";


// See if duplicate ID was inserted
$sql = "INSERT INTO Students VALUES (789, 'Jane', 2.5)";
if ($mysqli->query($sql))
   echo "Inserted Jane";
elseif ($mysqli->errno == 1062)
   echo "Insert failed because ID 789 already exists";
else
   die("Error $mysql->errno $mysqli->error<br>SQL = $sql<br>");
```

3. Update a record(s) in the database – Using the UPDATE statement

```
$sql = "UPDATE Students SET GPA=3.1 WHERE ID = 123";
$mysqli->query($sql) or die($mysqli->error);
$rows_updated = $mysqli->affected_rows;  // Should return 1
echo "Successfully updated $rows_updated row.";
```

Note: `$mysqli->affected_rows` will return 0 if no rows match the WHERE clause.

4. Delete a record(s) from the database – Using the DELETE statement

```
$sql = "DELETE FROM Students WHERE GPA < 2.0";
$mysqli->query($sql) or die($mysqli->error);
$rows_deleted = $mysqli->affected_rows;
echo "Successfully deleted $rows_deleted row(s).";
```

Note: `$mysqli->affected_rows` will return 0 if no rows match the WHERE clause.

F. Several characters like NUL (ASCII 0), \n, \r, \, ', ", and Ctl-Z can cause problems in a SQL statement and need to be escaped.

```
$name = $_POST["name"];    // "Ed O'Reily"
$name = $mysqli->real_escape_string($name);  // Returns "Ed O\'Reily"
$sql = "INSERT INTO Students VALUES (999, '$name', 3.1)";
```

More preventative techniques can be found at http://www.php.net/manual/en/security.database.sql-injection.php

G. Prepared Statements – These allow the same SQL statement to be executed repeatedly with high efficiency, and they are effective at preventing many SQL injection attacks. See http://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php