Annai Women's College

(Affiliated to Bharathidasan University, tiruchirapalli 620 024)
TNPL Road Punnamchatram, Karur – 639 1356

By

Asst.Prof S.Sasikala., MCA., M.Phil.,B.Ed.,

Department of Computer Science & Application

PHP Scripting Language

Unit- I

Essentials of PHP - Operators and Flow Control - Strings and Arrays.

Unit- II

Creating Functions - Reading Data in Web Pages - PHP Browser - Handling Power.

Unit- III

Object-Oriented Programming –Advanced Object-Oriented Programming .

Unit- IV

File Handling –Working with Databases – Sessions, Cookies, and FTP

Unit- V

Ajax – Advanced Ajax – Drawing Images on the Server.

Text Book:

1.The PHP Complete Reference – Steven Holzner – Tata McGraw-Hill Edition.

Reference Books:

1. Spring into PHP5 – Steven Holzer, Tata McCraw Hill Edition.
2. Ajax Bible- Steven Holzer , Tata McCraw Hill Edition.

Unit- I

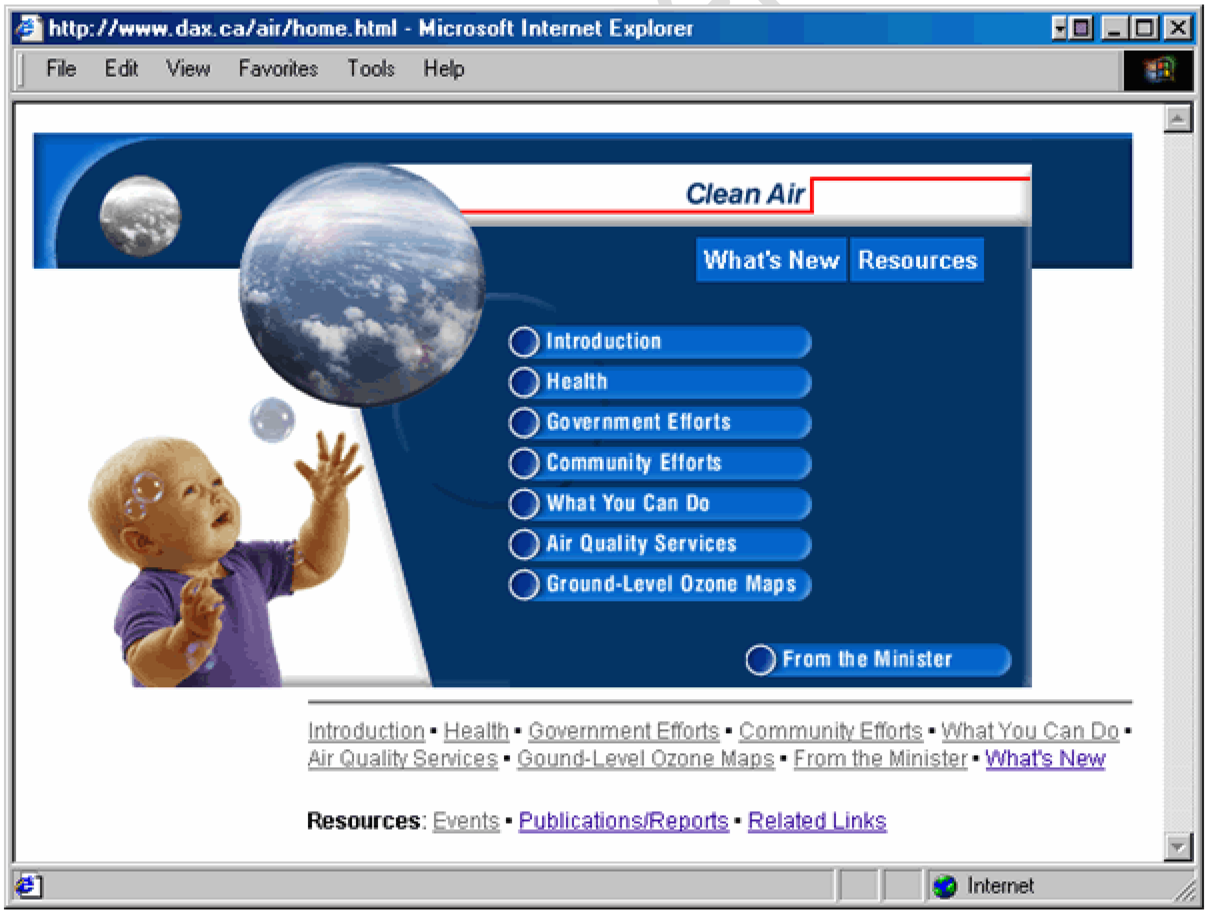
ESSENTIALS OF PHP

Enter PHP:

PHP: Hypertext Preprocessor.

PHP is a **server-side scripting** language designed primarily for **web development** but also used as a **general-purpose programming language**. Originally created **by Rasmus Lerdorf** in 1994, the PHP **reference implementation** is now produced by The PHP Development Team. PHP originally stood for ***Personal Home Page***

Eg:


http://www.dax.ca/air/home.html - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Clean Air
What's New
Resources
Introduction
Health
Government Efforts
Community Efforts
What You Can Do
Air Quality Services
Ground-Level Ozone Maps
From the Minister
Introduction • Health • Government Efforts • Community Efforts • What You Can Do • Air Quality Services • Gound-Level Ozone Maps • From the Minister • What's New
Resources: Events • Publications/Reports • Related Links
Internet

PHP on the Internet:

Internet service provider(ISP) quite probably already supports PHP.

Open acomand prompt for server and check on PHP that way.can open open a command command prompt window connected to server using various utilities telnet,SSH,orSSH2

Once a command prompt open for server can check if PHP is installed with the V option ,which gives the version ofPHP if it can be reached,I am going to use %as s generic command prompt

%php-v

If PHP is installed and accessible the PHP version and date displayed

%php-v

Php5.2.0(cli)(build:nov 2 2006 11:57:36)

Copyright © 1997-2006 the php group

Zend engine v2.2.0,copyright©1998-2006 zend technologies

PHP on your local machine:

1. If you use `localhost` rather than `0.0.0.0` you may hit a connection refused error.
2. If want to make the web server accessible to any interface, use `0.0.0.0`.
3. If a URI request does not specify a file, then either index.php or index.html in the given directory are returned.
4.

```
<?php
// router.php
if (preg_match('/\.(?:png|jpg|jpeg|gif)$/', $_SERVER["REQUEST_URI"])) {
    return false;    // serve the requested resource as-is.
} else {
    echo "<p>Welcome to PHP</p>";
}
?>

php -S 0.0.0.0:8000 router.php
```

... and navigate in your browser to http://localhost:8000/ and the following will be displayed:

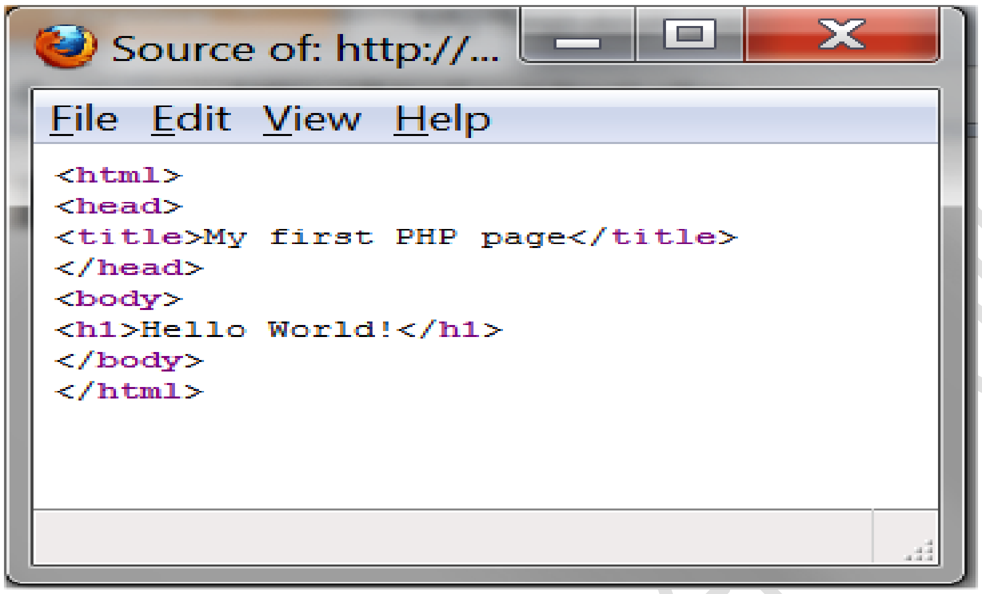
```
Welcome to PHP
```

Creating development environment:

Access to php on aserver at this point.to actually create php scripts to need to use a text editor of sonme kind ,PHP and save it in files with the extension

Text editor neds to be able to save files in plan text format,that is text without any special formatting codes.

Going to use word pad make sure to select the save as type option text document when you save file.not the default RTF type(rich text format). PHP use integrated development environment(IDE)to create PHP pages



```
<html>
<head>
<title>My first PHP page</title>
</head>
<body>
<h1>Hello World!</h1>
</body>
</html>
```

IDE available online that can handle PHP

- Komodo
- Maguma
- PHP edit
- Zend studio

Creating a first PHP page:

This is web applications from guest books to professional database lookup applications.this first PHP page will be a very simple one

```
<?php
.
Phpinfo()
.
.
?>
```

Running your first php page:

Web server is running IIS windows ans navigate to browser

Mixing HTML andPHP:

The php run on the server and php generate on the some html will be displayed in browser as well

```
<html
<head>
<title>A Web Page</title>
</head>
<body>

<p>

<?php
 echo "This is a basic PHP document";
?>

</p>

<p>

<?
 print "PHP is fun!";
?>

</p>

<p>

<script language="php">
 $myVar = "Hello World";
echo $myVar;

</script>

</p>
```

```
</body>
</html>
```

Printing some text:

Put text into php files much as HTML page,put an<h1>header and textinto apage

```
<html>
<body>

<h1>My first PHP page</h1>

<?php
echo "Hello Welcome!";
?>

</body>
</html>
```

Output:

My first PHP page

Hello Welcome!

Printing some HTML:

HTML not just simple text,that also gives the chane to make use of HTML to formate text

```
html>
<body>

<?php
echo "<h2>PHP is Fun!</h2>";
echo "Hello world!<br>";
echo "I'm about to learn PHP!<br>";
echo "This ", "string ", "was ", "made ", "with multiple parameters.";
?>

</body>
</html>
```

output:

PHP is Fun!
Hello world!
I'm about to learn PHP!
This string was made with multiple parameters.

More echo power:

Php run from command line,in fact simply by using the php command.HTML was not interpreted as html here,it was simply printed out as plain text.\n control character ,which php interpret as a newline character.

Eg:
```
echo"welcome\n";
echo"to\n";
echo"php.";
```

using PHP "here "document:

displaying text should be aware of and that using php"here"documents..a here document is just some text inserted directly in aphp pagebetween two instances of the same token.

Command line php:

The main focus of CLI SAPI is for developing shell applications with PHP. There are quite a few differences between the CLI SAPI and other SAPIs which are explained in this chapter. It is worth mentioning that CLI and CGIare different SAPIs although they do share many of the same behaviors.

The CLI SAPI is enabled by default using --enable-cli , but may be disabled using the --disable-cli option when running ./configure.

The name, location and existence of the CLI/CGI binaries will differ depending on how PHP is installed on your system. By default when executing make, both the CGI and CLI are built and placed as sapi/cgi/php-cgi andsapi/cli/php respectively, in your PHP source directory. You will note that both are named php. What happens during make install depends on your configure line. If a module SAPI is chosen during configure, such as apxs, or the --disable-cgi option is used, the CLI is copied to {PREFIX}/bin/php during make install otherwise the CGI is placed there. So, for example, if --with-apxs is in your configure line then the CLI is copied to {PREFIX}/bin/phpduring make install. If you want to override the installation of the CGI binary, use make install-cli after make install. Alternatively you can specify --disable-cgi in your configure line.

Note: The list of command line options provided by the PHP binary can be queried at any time by running PHP with the -h switch:

```
Usage: php [options] [-f] <file> [--] [args...]
   php [options] -r <code> [--] [args...]
   php [options] [-B <begin_code>] -R <code> [-E <end_code>] [--] [args...]
   php [options] [-B <begin_code>] -F <file> [-E <end_code>] [--] [args...]
   php [options] -- [args...]
   php [options] -a
```

```
-a              Run interactively
-c <path>|<file> Look for php.ini file in this directory
-n              No php.ini file will be used
-d foo[=bar]     Define INI entry foo with value 'bar'
-e              Generate extended information for debugger/profiler
-f <file>        Parse and execute <file>.
-h              This help
-i              PHP information
-l              Syntax check only (lint)
-m              Show compiled in modules
-r <code>        Run PHP <code> without using script tags <?..?>
-B <begin_code>  Run PHP <begin_code> before processing input lines
-R <code>        Run PHP <code> for every input line
-F <file>        Parse and execute <file> for every input line
-E <end_code>    Run PHP <end_code> after processing all input lines
-H              Hide any passed arguments from external tools.
-S <addr>:<port> Run with built-in web server.
-t <docroot>     Specify document root <docroot> for built-in web server.
-s              Output HTML syntax highlighted source.
-v              Version number
-w              Output source with stripped comments and whitespace.
-z <file>        Load Zend extension <file>.

args...          Arguments passed to script. Use -- args when first argument
               starts with - or script is read from stdin

--ini           Show configuration file names

--rf <name>      Show information about function <name>.
--rc <name>      Show information about class <name>.
--re <name>      Show information about extension <name>.
--rz <name>      Show information about Zend extension <name>.
--ri <name>      Show configuration for extension <name>.
```

Adding commants to PHP code:

If you have a longer, multi-line comment, the best way to comment is with /* and */ before and after a lengthy comment.

You can contain several lines of commenting inside a block. Here is an example:

```
<?php
echo "hello";
/*
Using this method
you can create a larger block of text
and it will all be commented out
```

```
*/
echo " there";
?>
```

Working with variables:

A good understanding of how variables are stored and manipulated is essential to becoming a Hacker. The engine attempts to cover up the complexity of the concept of a variable that can be any type by providing a uniform and intuitive set of macros for accessing the structures various fields. As the Hacker works through this chapter, they should become comfortable with the terminology and concepts involved with Variables in PHP.
PHP is a dynamic, loosely typed language, that uses copy-on-write and reference counting.
To clarify what exactly is meant by the statement above: PHP is a high level language, weak typing is implicit of the engines preference to convert, or coerce variables into the required type at execution time.
Reference counting is the means by which the engine can deduce when a variable no longer has any references in the users code, and so is able to free the structures associated with the variable.

All variables in PHP are represented by one structure, the zval:

```
typedef struct _zval_struct {
    zvalue_value value;        /* variable value */
    zend_uint refcount__gc;    /* reference counter */
    zend_uchar type;           /* value type */
    zend_uchar is_ref__gc;     /* reference flag */
} zval;
```

Storing datav in variables:

Rules for PHP variables:

- A variable starts with the $ sign, followed by the name of the variable
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive ($age and $AGE are two different variables)
  Output Variables
- The PHP echo statement is often used to output data to the screen.

The following example will show how to output text and a variable:
Example

```
<?php
$txt = "INDIA";
echo "I love $txt!";
?>
```

Output:

```
I  love  INDIA !
```

Interppolating Strings:

In computer programming, **string interpolation** (or **variable interpolation**, **variable substitution**, or **variable expansion**) is the process of evaluating a string literal containing one or more placeholders, yielding a result in which the placeholders are replaced with their corresponding values. It is a form of simple template processing or, in formal terms, a form of quasi-quotation (or logic substitution interpretation). *String interpolation* allows easier and more intuitive string formatting and content-specification compared with string concatenation. *String interpolation* is common in many programming languages which make heavy use of string representations of data, such as Apache Groovy, Kotlin, Perl, PHP, Python, Ruby, Scala, and Swift, and most Unix shells. Two modes of literal expression are usually offered: one with interpolation enabled, the other without (termed *raw string*). Placeholders are usually represented by a bare or a named sigil (typically `$` or `%`), e.g. `$placeholder` or `%123`. Expansion of the string usually occurs at run time.
Eg:

```
<?php
$apples = 5;
$bananas = 3;
echo "There are $apples apples and $bananas bananas.";
echo "\n"
echo "I have ${apples} apples and ${bananas} bananas.";
```

output :

```
There are 5 apples and 3 bananas.
I have 5 apples and 3 bananas.
```

Creating variable variables:

Sometimes it is convenient to be able to have variable variable names. That is, a variable name which can be set and used dynamically. A normal variable is set with a statement
such as:

```
<?php
$a = 'hello';
?>
```

A variable variable takes the value of a variable and treats that as the name of a variable. In the above example,*hello*, can be used as the name of a variable by using two dollar signs.
i.e.

```
<?php
$$a = 'world';
?>
```

Crating constants:

To create a constant, use the define() function.

Syntax

define(*name, value, case-insensitive*)

Parameters:

name: Specifies the name of the constant
***value*:** Specifies the value of the constant
case-insensitive: Specifies whether the constant name should be case-insensitive. Default is false
The example below creates a constant with a **case-sensitive** name:

Example

```
<?php
define("GREETING", "Welcome to annai!");
echo GREETING;
?>
```

Output:

```
Welcome to annai!
```

Understanding php's internal data types:

- String.
- Integer.
- Float (floating point numbers - also called double)
- Boolean.
- Array.
- Object.
- NULL.
- Resource.

PHP String

A string is a sequence of characters, like "Hello world!".
A string can be any text inside quotes. You can use single or double quotes:

Example

```
<?php
$x = "Hello world!";
$y = 'Hello world!';

echo $x;
echo "<br>";
echo $y;
?>
```

Output:

```
Hello world!
Hello world!
```

PHP Integer

An integer data type is a non-decimal number between -2,147,483,648 and 2,147,483,647.
Rules for integers:
An integer must have at least one digit
An integer must not have a decimal point
An integer can be either positive or negative
Integers can be specified in three formats: decimal (10-based), hexadecimal (16-based - prefixed with 0x) or octal (8-based - prefixed with 0)
In the following example $x is an integer. The PHP var_dump() function returns the data type and value:
Example

```
<?php
$x = 5985;
var_dump($x);
?>
```

Output:

```
int(5985)
```

PHP Float

A float (floating point number) is a number with a decimal point or a number in exponential form.
In the following example $x is a float. The PHP var_dump() function returns the data type and value:
Example

```
<?php
$x = 10.365;
var_dump($x);
?>
```

Output:

```
float(10.365)
```

PHP Boolean

A Boolean represents two possible states: TRUE or FALSE.

```
$x = true;
$y = false;
```

Booleans are often used in conditional testing. You will learn more about conditional testing in a later chapter of this tutorial.

PHP Array

An array stores multiple values in one single variable.
In the following example $cars is an array. The PHP var_dump() function returns the data type and value:
Example

```
<?php
$cars = array("Volvo","BMW","Toyota");
var_dump($cars);
?>
```

Output:

```
array(3) { [0]=> string(5) "Volvo" [1]=> string(3) "BMW" [2]=> string(6) "Toyota" }
```

PHP Object

An object is a data type which stores data and information on how to process that data.
In PHP, an object must be explicitly declared.
First we must declare a class of object. For this, we use the class keyword. A class is a structure that can contain properties and methods:

Example

```
<?php
class Car {
   function Car() {
      $this->model = "VW";
   }
}
// create an object
$herbie = new Car();

// show object properties
echo $herbie->model;
?>
```

Output:

```
 VW
```

PHP NULL Value

Null is a special data type which can have only one value: NULL.
A variable of data type NULL is a variable that has no value assigned to it.
Tip: If a variable is created without a value, it is automatically assigned a value of NULL.
Variables can also be emptied by setting the value to NULL:

Example

```
<?php
$x = "Hello world!";
$x = null;
var_dump($x);
?>
```

Output:

```
NULL
```

PHP Resource

The special resource type is not an actual data type. It is the storing of a reference to functions and resources external to PHP.
A common example of using the resource data type is a database call.
We will not talk about the resource type here, since it is an advanced topic.

OPERATORS AND FLOW CONTROL

Operators:

Operators are symbols that tell the PHP processor to perform certain actions. For example, the addition (+) symbol is an operator that tells PHP to add two variables or values, while the greater-than (>) symbol is an operator that tells PHP to compare two values.

Php's math operators:

The PHP arithmetic or math operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced PHP 5.6)

Eg:

```
<html>
<body>

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

echo $x + $y;
?>

</body>
</html>
```

Output:

```
16
```

Working with the assignment operators:

The basic assignment operator is "=". Your first inclination might be to think of this as "equal to". Don't. It really means that the left operand gets set to the value of the expression on the right (that is, "gets set to").

The value of an assignment expression is the value assigned. That is, the value of "$a = 3" is 3. This allows you to do some tricky things:

```
<?php

$a = ($b = 4) + 5; // $a is equal to 9 now, and $b has been set to 4.

?>
```

In addition to the basic assignment operator, there are "combined operators" for all of the binary arithmetic, array union and string operators that allow you to use a value in an expression and then set its value to the result of that expression.

Incrementing and decrementing values:

PHP supports C-style pre- and post-increment and decrement operators. The increment/decrement operators only affect numbers and strings. Arrays, objects and resources are not affected. Decrementing **NULL** values has no effect too, but incrementing them results in *1*.

Increment/decrement Operators

Example	Name	Effect
++$a	Pre-increment	Increments *$a* by one, then returns *$a*.
$a++	Post-increment	Returns *$a*, then increments *$a* by one.
--$a	Pre-decrement	Decrements *$a* by one, then returns *$a*.
$a--	Post-decrement	Returns *$a*, then decrements *$a* by one.

Here's a simple example script:

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "Should be 5: " . $a++ . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";

echo "<h3>Preincrement</h3>";
$a = 5;
echo "Should be 6: " . ++$a . "<br />\n";
echo "Should be 6: " . $a . "<br />\n";

echo "<h3>Postdecrement</h3>";
$a = 5;
echo "Should be 5: " . $a-- . "<br />\n";
echo "Should be 4: " . $a . "<br />\n";

echo "<h3>Predecrement</h3>";
$a = 5;
echo "Should be 4: " . --$a . "<br />\n";
echo "Should be 4: " . $a . "<br />\n";
?>
```

The PHP string operators:

There are two string operators. The first is the concatenation operator ('.'), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator ('.='), which appends the argument on the right side to the argument on the left side. Please read Assignment Operators for more information.

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

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

See also the manual sections on the String type and String functions.

The Bitwise operators:

Bitwise operators allow evaluation and manipulation of specific bits within an integer.

Bitwise Operators

Example	Name	Result
$a & $b	And	Bits that are set in both *$a* and *$b* are set.
$a | $b	Or (inclusive or)	Bits that are set in either *$a* or *$b* are set.
$a ^ $b	Xor (exclusive or)	Bits that are set in *$a* or *$b* but not both are set.
~ $a	Not	Bits that are set in *$a* are not set, and vice versa.
$a << $b	Shift left	Shift the bits of *$a* *$b* steps to the left (each step means "multiply by two")
$a >> $b	Shift right	Shift the bits of *$a* *$b* steps to the right (each step means "divide by two")

Bit shifting in PHP is arithmetic. Bits shifted off either end are discarded. Left shifts have zeros shifted in on the right while the sign bit is shifted out on the left, meaning the sign of an operand is not preserved. Right shifts have copies of the sign bit shifted in on the left, meaning the sign of an operand is preserved.

Use parentheses to ensure the desired precedence. For example, *$a & $b == true* evaluates the equivalency then the bitwise and; while *($a & $b) == true* evaluates the bitwise and then the equivalency.

If both operands for the *&*, *|* and *^* operators are strings, then the operation will be performed on the ASCII values of the characters that make up the strings and the result will be a string. In all other cases, both operands will beconverted to integers and the result will be an integer.

If the operand for the *~* operator is a string, the operation will be performed on the ASCII values of the characters that make up the string and the result will be a string, otherwise the operand and the result will be treated as integers.

Both operands and the result for the *<<* and *>>* operators are always treated as integers.

```
?php
/*
 * Here are the examples.
 */

echo "\n--- BIT SHIFT RIGHT ON POSITIVE INTEGERS ---\n";

$val = 4;
$places = 1;
$res = $val >> $places;
p($res, $val, '>>', $places, 'copy of sign bit shifted into left side');
```

```
$val = 4;
$places = 2;
$res = $val >> $places;
p($res, $val, '>>', $places);

$val = 4;
$places = 3;
$res = $val >> $places;
p($res, $val, '>>', $places, 'bits shift out right side');

$val = 4;
$places = 4;
$res = $val >> $places;
p($res, $val, '>>', $places, 'same result as above; can not shift beyond 0');


echo "\n--- BIT SHIFT RIGHT ON NEGATIVE INTEGERS ---\n";

$val = -4;
$places = 1;
$res = $val >> $places;
p($res, $val, '>>', $places, 'copy of sign bit shifted into left side');

$val = -4;
$places = 2;
$res = $val >> $places;
p($res, $val, '>>', $places, 'bits shift out right side');

$val = -4;
$places = 3;
$res = $val >> $places;
p($res, $val, '>>', $places, 'same result as above; can not shift beyond -1');


echo "\n--- BIT SHIFT LEFT ON POSITIVE INTEGERS ---\n";

$val = 4;
$places = 1;
$res = $val << $places;
p($res, $val, '<<', $places, 'zeros fill in right side');

$val = 4;
$places = (PHP_INT_SIZE * 8) - 4;
$res = $val << $places;
p($res, $val, '<<', $places);

$val = 4;
```

```
$places = (PHP_INT_SIZE * 8) - 3;
$res = $val << $places;
p($res, $val, '<<', $places, 'sign bits get shifted out');

$val = 4;
$places = (PHP_INT_SIZE * 8) - 2;
$res = $val << $places;
p($res, $val, '<<', $places, 'bits shift out left side');


echo "\n--- BIT SHIFT LEFT ON NEGATIVE INTEGERS ---\n";

$val = -4;
$places = 1;
$res = $val << $places;
p($res, $val, '<<', $places, 'zeros fill in right side');

$val = -4;
$places = (PHP_INT_SIZE * 8) - 3;
$res = $val << $places;
p($res, $val, '<<', $places);

$val = -4;
$places = (PHP_INT_SIZE * 8) - 2;
$res = $val << $places;
p($res, $val, '<<', $places, 'bits shift out left side, including sign bit');


/*
 * Ignore this bottom section,
 * it is just formatting to make output clearer.
 */

function p($res, $val, $op, $places, $note = '') {
   $format = '%0' . (PHP_INT_SIZE * 8) . "b\n";

   printf("Expression: %d = %d %s %d\n", $res, $val, $op, $places);

   echo " Decimal:\n";
   printf("  val=%d\n", $val);
   printf("  res=%d\n", $res);

   echo " Binary:\n";
   printf('  val=' . $format, $val);
   printf('  res=' . $format, $res);

   if ($note) {
      echo " NOTE: $note\n";
```

```
    }

    echo "\n";
}
?>
```

Output of the above example on 32 bit machines:

```
--- BIT SHIFT RIGHT ON POSITIVE INTEGERS ---
Expression: 2 = 4 >> 1
 Decimal:
  val=4
  res=2
 Binary:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000010
 NOTE: copy of sign bit shifted into left side

Expression: 1 = 4 >> 2
 Decimal:
  val=4
  res=1
 Binary:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000001

Expression: 0 = 4 >> 3
 Decimal:
  val=4
  res=0
 Binary:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000000
 NOTE: bits shift out right side

Expression: 0 = 4 >> 4
 Decimal:
  val=4
  res=0
 Binary:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000000
 NOTE: same result as above; can not shift beyond 0


--- BIT SHIFT RIGHT ON NEGATIVE INTEGERS ---
Expression: -2 = -4 >> 1
 Decimal:
  val=-4
  res=-2
 Binary:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111110
 NOTE: copy of sign bit shifted into left side

Expression: -1 = -4 >> 2
 Decimal:
  val=-4
  res=-1
 Binary:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111111
 NOTE: bits shift out right side

Expression: -1 = -4 >> 3
 Decimal:
  val=-4
  res=-1
 Binary:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111111
 NOTE: same result as above; can not shift beyond -1


--- BIT SHIFT LEFT ON POSITIVE INTEGERS ---
Expression: 8 = 4 << 1
 Decimal:
  val=4
  res=8
 Binary:
  val=00000000000000000000000000000100
  res=00000000000000000000000000001000
 NOTE: zeros fill in right side

Expression: 1073741824 = 4 << 28
 Decimal:
  val=4
  res=1073741824
 Binary:
  val=00000000000000000000000000000100
  res=01000000000000000000000000000000

Expression: -2147483648 = 4 << 29
 Decimal:
  val=4
  res=-2147483648
 Binary:
  val=00000000000000000000000000000100
  res=10000000000000000000000000000000
 NOTE: sign bits get shifted out

Expression: 0 = 4 << 30
 Decimal:
  val=4
  res=0
 Binary:
  val=00000000000000000000000000000100
  res=00000000000000000000000000000000
 NOTE: bits shift out left side


--- BIT SHIFT LEFT ON NEGATIVE INTEGERS ---
Expression: -8 = -4 << 1
 Decimal:
  val=-4
  res=-8
 Binary:
  val=11111111111111111111111111111100
  res=11111111111111111111111111111000
 NOTE: zeros fill in right side

Expression: -2147483648 = -4 << 29
 Decimal:
  val=-4
  res=-2147483648
 Binary:
  val=11111111111111111111111111111100
  res=10000000000000000000000000000000

Expression: 0 = -4 << 30
 Decimal:
  val=-4
  res=0
 Binary:
  val=11111111111111111111111111111100
  res=00000000000000000000000000000000
 NOTE: bits shift out left side, including sign bit
```

Output of the above example on 64 bit machines:

The Execution operator:

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 is identical to shell_exec().

```
<?php
$output = `ls -al`;
echo "<pre>$output</pre>";
?>
```
The backtick operator is disabled when safe mode is enabled or shell_exec() is disabled.

Unlike some other languages, backticks have no special meaning within double-quoted strings.
See also the manual section on Program Execution functions, popen() proc_open(), and Using PHP from the commandline.

The PHP operator precedence:

The precedence of an operator specifies how "tightly" it binds two expressions together. For example, in the expression *1 + 5 * 3*, the answer is *16* and not *18* because the multiplication ("*") operator has a higher precedence than the addition ("+") operator. Parentheses may be used to force precedence, if necessary. For instance: *(1 + 5) * 3* evaluates to *18*.
When operators have equal precedence their associativity decides how the operators are grouped. For example "-" is left-associative, so *1 - 2 - 3* is grouped as *(1 - 2) - 3* and evaluates to *-4*. "=" on the other hand is right-associative, so *$a = $b = $c* is grouped as *$a = ($b = $c)*.
Operators of equal precedence that are non-associative cannot be used next to each other, for example *1 < 2 > 1* is illegal in PHP. The expression *1 <= 1 == 1* on the other hand is legal, because the == operator has lesser precedence than the <= operator.
Use of parentheses, even when not strictly necessary, can often increase readability of the code by making grouping explicit rather than relying on the implicit operator precedence and associativity.
The following table lists the operators in order of precedence, with the highest-precedence ones at the top. Operators on the same line have equal precedence, in which case associativity decides grouping.

Operator Precedence

Associativity	**Operators**	**Additional Information**
non-associative	*clone new*	clone and new
left	*[*	array()
right	**	arithmetic
right	++ -- ~ *(int) (float) (string) (array) (object) (bool)* @	types and increment/decrement
non-associative	*instanceof*	types
right	*!*	logical
left	* / %	arithmetic
left	+ - .	arithmetic and string
left	<< >>	bitwise
non-associative	< <= > >=	comparison
non-associative	== != === !== <> <=>	comparison
left	&	bitwise and references
left	^	bitwise
left	|	bitwise
left	&&	logical
left	||	logical
right	??	comparison

Operator Precedence		
Associativity	**Operators**	**Additional Information**
left	*? :*	ternary
right	= += -= *= **= /= .= %= &= |= ^= <<= >>=	assignment
left	*and*	logical
left	*xor*	logical
left	*or*	logical

Example :

```
<?php
$a = 3 * 3 % 5; // (3 * 3) % 5 = 4
// ternary operator associativity differs from C/C++
$a = true ? 0 : true ? 1 : 2; // (true ? 0 : true) ? 1 : 2 = 2

$a = 1;
$b = 2;
$a = $b += 3; // $a = ($b += 3) -> $a = 5, $b = 5
?>
```

Operator precedence and associativity only determine how expressions are grouped, they do not specify an order of evaluation. PHP does not (in the general case) specify in which order an expression is evaluated and code that assumes a specific order of evaluation should be avoided, because the behavior can change between versions of PHP or depending on the surrounding code.

FLOW CONTROL

Using the if statement:

The if statement executes some code if one condition is true.

Syntax

```
if (condition) {
   code to be executed if condition is true;
}
```

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<?php
$t = date("H");

if ($t < "20") {
   echo "Have a good day!";
}
?>
```

output:

```
Have a good day!
```

The php comparison operators:

Comparison operators, as their name implies, allow you to compare two values. You may also be interested in viewing the type comparison tables, as they show examples of various type related comparisons.

Comparison Operators		
Example	**Name**	**Result**
$a == $b	Equal	**TRUE** if *$a* is equal to *$b* after type juggling.
$a === $b	Identical	**TRUE** if *$a* is equal to *$b*, and they are of the same type.
$a != $b	Not equal	**TRUE** if *$a* is not equal to *$b* after type juggling.
$a <> $b	Not equal	**TRUE** if *$a* is not equal to *$b* after type juggling.
$a !== $b	Not identical	**TRUE** if *$a* is not equal to *$b*, or they are not of the same type.
$a < $b	Less than	**TRUE** if *$a* is strictly less than *$b*.
$a > $b	Greater than	**TRUE** if *$a* is strictly greater than *$b*.
$a <= $b	Less than or equal to	**TRUE** if *$a* is less than or equal to *$b*.
$a >= $b	Greater than or equal to	**TRUE** if *$a* is greater than or equal to *$b*.
$a <=> $b	Spaceship	An integer less than, equal to, or greater than zero when *$a* is respectively less than, equal greater than *$b*. Available as of PHP 7.

If you compare a number with a string or the comparison involves numerical strings, then each string is converted to a number and the comparison performed numerically. These rules also apply to the switch statement. The type conversion does not take place when the comparison is === or !== as this involves comparing the type as well as the value.

```
<?php
// Arrays are compared like this with standard comparison operators
function standard_array_compare($op1, $op2)
{
    if (count($op1) < count($op2)) {
        return -1; // $op1 < $op2
    } elseif (count($op1) > count($op2)) {
        return 1; // $op1 > $op2
    }
    foreach ($op1 as $key => $val) {
        if (!array_key_exists($key, $op2)) {
            return null; // uncomparable
        } elseif ($val < $op2[$key]) {
            return -1;
        } elseif ($val > $op2[$key]) {
            return 1;
        }
    }
    return 0; // $op1 == $op2
}
?>
```

The Php logical 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 reason for the two different variations of "and" and "or" operators is that they operate at different precedences. (See Operator Precedence.)

```
?php

if ( !isset($x)) {
   $x = 123;
}

// or

$x = isset($x) ? $x : 123;

// or

$x = isset($x) ?: 123;
?>
```

The else statement:

The if....else statement executes some code if a condition is true and another code if that condition is false.

Syntax

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

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?php
$t = date("H");

if ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>
```

Output:

```
 Have a good day!
```

The else if statement:

The if....elseif...else statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
   code to be executed if this condition is true;
} elseif (condition) {
   code to be executed if this condition is true;
} else {
   code to be executed if all conditions are false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php
$t = date("H");

if ($t < "10") {
   echo "Have a good morning!";
} elseif ($t < "20") {
   echo "Have a good day!";
} else {
   echo "Have a good night!";
}
?>
```

Output:

The hour (of the server) is 04, and will give the following message:
Have a good morning!

The ternary operator:

When I learned how to use the ternary operator years ago, I fell in love with it. What a cool way to simplify assignments based on a condition. If you're not sure what the ternary operator is or how it works, you're missing out on a really cool piece of programming knowledge.
Let's start out with an example and I'll explain how it works below. I'll use PHP, but the syntax is exactly the same for JavaScript. Let's say we want to assign one of two values to $x based on a certain condition. Using a conditional (if/then/else), it would look like this:

```
if( $valid ) {
   $x = 'yes';
} else {
   $x = 'no';
}
```

That should be pretty easy to read. If $valid is true set $x to *yes*, otherwise set it to *no*. But this example is rather long for such a simple assignment. Let's do it in one line with ternary logic:

```
$x = $valid ? 'yes' : 'no';
```

This produces the same result as before, but it's much shorter. Read it out loud like this:

If x is valid set it to yes; otherwise set it to no.

It's not that intimidating once you wrap your head around it. You can even nest them if you feel like being dangerous:

```
$valid = true;
$lang = 'french';

$x = $valid ? ($lang === 'french' ? 'oui' : 'yes') : ($lang === 'french' ? 'non' : 'no');

echo $x; // outputs 'oui'
```

Try running this script in your dev environment and playing with the values to better understand how it works.

The switch Statement:

Use the switch statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
   case label1:
      code to be executed if n=label1;
      break;
   case label2:
      code to be executed if n=label2;
      break;
   case label3:
      code to be executed if n=label3;
      break;
   ...
   default:
      code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The **default** statement is used if no match is found.

Example

```
<?php
$favcolor = "red";

switch ($favcolor) {
   case "red":
      echo "Your favorite color is red!";
      break;
   case "blue":
      echo "Your favorite color is blue!";
      break;
   case "green":
      echo "Your favorite color is green!";
      break;
   default:
      echo "Your favorite color is neither red, blue, nor green!";
}
?>
```

Output:

```
 Your favorite color is red!
```

Using for loops:

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (init counter; test counter; increment counter) {
   code to be executed;
}
```

Parameters:

init counter: Initialize the loop counter value

test counter: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.

increment counter: Increases the loop counter value

The example below displays the numbers from 0 to 10:

Example

```
<?php
for ($x = 0; $x <= 10; $x++) {
   echo "The number is: $x <br>";
}
?>
```

Output:

```
The number is: 0
The number is: 1
The number is: 2
The number is: 3
The number is: 4
The number is: 5
The number is: 6
The number is: 7
The number is: 8
The number is: 9
The number is: 10
```

Using while loops:

The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
    code to be executed;
}
```

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

Example

```php
<?php
$x = 1;

while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```

Output:

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

Using do-while loops:

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true

Syntax

```
do {
    code to be executed;
} while (condition is true);
```

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

```
<?php
$x = 1;

do {
   echo "The number is: $x <br>";
   $x++;
} while ($x <= 5);
?>
```

Output:

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

Using the foreach loop:

The foreach loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

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

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.
The following example demonstrates a loop that will output the values of the given array ($colors):

Example

```
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
   echo "$value <br>";
}
?>
```

Output:

```
red
green
blue
yellow
```

Terminating loops early:

break ends execution of the current *for*, *foreach*, *while*, *do-while* or *switch* structure.
break accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of. The default value is *1*, only the immediate enclosing structure is broken out of.

```
<?php
$arr = array('one', 'two', 'three', 'four', 'stop', 'five');
while (list(, $val) = each($arr)) {
   if ($val == 'stop') {
       break;    /* You could also write 'break 1;' here. */
   }
   echo "$val<br />\n";
}
```

Skipping iterations:

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration.
Note: In PHP the switch statement is considered a looping structure for the purposes of *continue*. *continue*behaves like *break* (when no arguments are passed). If a *switch* is inside a loop, *continue 2* will continue with the next iteration of the outer loop.
continue accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of. The default value is *1*, thus skipping to the end of the current loop.

```
<?php
while (list($key, $value) = each($arr)) {
   if (!($key % 2)) { // skip even members
       continue;
   }
   do_something_odd($value);
}

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

PHP alternate syntax:

Sometimes you wind up working on a page that switches back and forth between php and html. It can be tricky keeping track of code or making the mistake of over using the *echo* function. Luckily php supports an alternative syntax for control structures allowing you to jump in and out of php and html. Any code (html, php, anything) nested within a conditional will only parse if the defined conditions are matched. The end result is clean, organized, efficient code.
One of my interns once put all his html code into an array and printed each part while looping through function calls. It worked, but it his source code was so hard to read that when I asked him to take a look at it a few months later he couldn't even make sense of what he wrote.
PHP's alternative control structure syntax makes it easy to identify code blocks and it's widely used in many popular open source projects. WordPress for example makes use of this syntax all over the place. I you ever looked at the code inside a WordPress theme, I'm sure you've taken notice to this syntax. Here's a quick example.
?

```
<?php// quick example, prints hungry if sandwich is false - it isn't
$sandwich = true;
if($sandwich == false):
  echo 'I am hungry.';
else:
  echo'Thank you, that was delicious.';
endif;
?>
```

Anyone familiar with Visual Basic might be drawing comparisons to the syntax and noticing the end if statement. Some might even suggest that this a counterproductive approach to clean code. PHP does this on purpose, because now you can jump out of the language and things can become a lot cleaner, and easy to read. So I'd sacrifice having to close an if statement with 'endif' any day over a complicated list of echo commands.

STRINGS AND ARRAYS

In PHP, two types of data (strings and arrays) warrant special attention and a more complete explanation. This chapter details how and why these special data types are employed in PHP 5 with copious examples to get you started using these in your code.

The String Functions:

data types merit some special attention—strings and arrays. We've already seen strings at work, including single- and double-quoted strings (recall also that double-quoted strings allow variable interpolation). PHP also comes packed with more string power, and we're going to dig into that in this chapter—tons of functions are built into PHP that work with strings, from sorting strings to searching them, trimming extra spaces off of them, and getting their lengths. We'll get a handle on those functions in this chapter.
Besides strings, we're also going to get a handle on *arrays* in this chapter. We've seen how to store data in simple variables, but there's more to the story here. Arrays can hold multiple data items, assigning each one a numeric or text *index* (also called a *key*). For example, if you want to store some student test scores, you can store them in an array, and then you can access each score in the array via a numeric index. That's great as far as computers are concerned because you can work through all the elements in an array simply by steadily incrementing that index, as you might do with a loop. In that way, you can use your computer

to iterate over all the elements in an array in order to print them out or find their average value, for example.
Arrays represent the first time we're associating data items together. Up to this point, we've only worked with simple variables, but working with arrays is fundamental to PHP for such tasks as reading the data that users enter in web pages. We'll get the details on strings and arrays in this chapter, and I'll start with the string functions.
PHP has plenty of built-in string functions. Table 3-1 lists a selection of them.
Table 3-1. The String Functions

Function	**Purpose**
chr	Returns a specific character, given its ASCII code
chunk_split	Splits a string into smaller chunks
crypt	Supports one-way string encryption (hashing)
echo	Displays one or more strings
explode	Splits a string on a substring
html_entity_decode	Converts all HTML entities to their applicable characters
htmlentities	Converts all applicable characters to HTML entities
htmlspecialchars	Converts special characters to HTML entities
implode	Joins array elements with a string
ltrim	Strips whitespace from the beginning of a string
number_format	Formats a number with grouped thousand separators
ord	Returns the ASCII value of character
parse_str	Parses the string into variables
print	Displays a string
printf	Displays a formatted string
rtrim	Strips whitespace from the end of a string
setlocale	Sets locale information
similar_text	Calculates the similarity between two strings
sprintf	Returns a formatted string
sscanf	Parses input from a string according to a format

Function	**Purpose**
str_ireplace	Case-insensitive version of the str_replace function.
str_pad	Pads a string with another string
str_repeat	Repeats a string
str_replace	Replaces all occurrences of the search string with the replacement string
str_shuffle	Shuffles a string randomly
str_split	Converts a string to an array
str_word_count	Returns information about words used in a string
strcasecmp	Binary case-insensitive string comparison
strchr	Alias of the strstr function
strcmp	Binary-safe string comparison
strip_tags	Strips HTML and PHP tags from a string
stripos	Finds position of first occurrence of a case-insensitive string
stristr	Case-insensitive version of the strstr function
strlen	Gets a string's length
strnatcasecmp	Case-insensitive string comparisons
strnatcmp	String comparisons using a "natural order" algorithm
strncasecmp	Binary case-insensitive string comparison of the first n characters
strncmp	Binary-safe string comparison of the first n characters
strpos	Finds position of first occurrence of a string
strrchr	Finds the last occurrence of a character in a string
strrev	Reverses a string
strripos	Finds the position of last occurrence of a case-insensitive string
strrpos	Finds the position of last occurrence of a char in a string
strspn	Finds the length of initial segment matching mask

Function	Purpose
strstr	Finds the first occurrence of a string
strtolower	Converts a string to lowercase
strtoupper	Converts a string to uppercase
strtr	Translates certain characters
substr_compare	Binary-safe (optionally case-insensitive) comparison of two strings from an offset
substr_count	Counts the number of substring occurrences
substr_replace	Replaces text within part of a string
substr	Returns part of a string
trim	Strips whitespace from the beginning and end of a string

Home > Articles > Programming > PHP

Using the String Functions:

Here's an example that puts some of the useful string functions to work:

```
<?php
    echo trim("    No worries."), "\n";
    echo substr("No worries.", 3, 7), "\n";
    echo "\"worries\" starts at position ", strpos("No worries.", "worries"), "\n";
    echo ucfirst("no worries."), "\n";
    echo "\"No worries.\" is ", strlen("No worries."), " characters long.\n";
    echo substr_replace("No worries.", "problems.", 3, 8), "\n";
    echo chr(65), chr(66), chr(67), "\n";
    echo strtoupper("No worries."), "\n";
?>
```

In this example, we're using trim to trim leading spaces from a string, substr to extract a substring from a string, strpos to search a string for a substring, ucfirst to convert the first character of a string to uppercase, strlen to determine a string's length, substr_replace to replace a substring with another string, chr to convert an ASCII code to a letter (ASCII 65 = "A", ASCII 66 = "B", and so on), and strtoupper to convert a string to uppercase
.

Converting to and from trings:

Converting between string format and other formats is a common task on the Internet because the data passed from the browser to the server and back in text strings. To convert to a string, you can use the (string) cast or the strval function; here's what this might look like:

Eg:

```
<?php
   $float = 1.2345;
   echo (string) $float, "\n";
   echo strval($float), "\n";
?>
```

A boolean TRUE value is converted to the string "1", and the FALSE value is represented as "" (empty string). An integer or floating point number (float) is converted to a string representing the number with its digits (including the exponent part for floating point numbers). The value NULL is always converted to an empty string.
You can also convert a string to a number. The string will be treated as a float if it contains any of the characters '.', 'e', or 'E'. Otherwise, it will be treated as an integer.
PHP determines the numeric value of a string from the *initial part* of the string. If the string starts with numeric data, it will use that. Otherwise, the value will be 0 (zero). Valid numeric data consists of an optional sign (+ or -), followed by one or more digits (including a decimal point if you're using it) and an optional exponent (the exponent part is an 'e' or 'E', followed by one or more digits).
PHP will do the right thing if you start using a string in a numeric context, as when you start adding values together. Here are some examples to make all this clearer:

Eg:

```
<?php
   $number = 1 + "14.5";
   echo "$number\n";
   $number = 1 + "-1.5e2";
   echo "$number\n";
   $text = "5.0";
   $number = (float) $text;
   echo $number / 2.0, "\n";
?>
```

Output:
And here's what you see when you run this script:

```
15.5
-149
2.5
```

Formatting text strings:

Like many other languages, PHP features the versatile printf() and sprintf() functions that you can use to format strings in many different ways. These functions are handy when you need convert data between different formats — either to make it easy for people to read, or for passing to another program.
PHP features many other functions to format strings in specific ways — for example, the date() function is ideal for formatting date strings. However, printf() and sprintf() are great for general-purpose formatting. In this tutorial you look at how to work with printf() and sprintf() to format strings.

A simple printf() example

The easiest way to understand printf() is to look at an example. The following code displays a string containing 2 numbers:

```
// Displays "Australia comprises 6 states and 10 territories"
printf( "Australia comprises %d states and %d territories", 6, 10 );
```

Notice how the first %d in the string is replaced with the first argument after the string (6), while the second %d is replaced with the second argument (10). Here's how it works:
The first argument is always a string, and is called the ***format string***. The format string contains regular text, as well as some optional ***format specifiers*** (the %ds in this example).
Each format specifier begins with a % (percent) sign. It takes an additional argument after the format string, formats the argument in a certain way, and inserts the result into the final string, which is then displayed in the Web page.
To specify a literal percent character, write %%.

Type specifiers

The example above uses the %d format specifier. This formats an argument as a signed decimal integer. The 'd' is known as a ***type specifier***, and printf() supports a wide range of them. Here's a full list of type specifiers:

b
Format the argument as a binary integer (e.g. 10010110)
c
Format the argument as a character with the argument's ASCII value
d
Format the argument as a signed decimal integer
e
Format the argument in scientific notation (e.g. 1.234e+3)
f
Format the argument as a floating-point number using the current locale settings (e.g. in France a comma is used for the decimal point)
F
As above, but ignore the locale settings
o
Format the argument as an octal integer
s
Format the argument as a string
u
Format the argument as an unsigned decimal integer
x
Format the argument as a lowercase hexadecimal integer (e.g. 4fdf87)
X
Format the argument as an uppercase hexadecimal integer (e.g. 4FDF87)

Here's a simple example of type specifiers in action:

```
printf( "Here's the number %s as a float (%f), a binary integer (%b), an octal integer (%o), and a hex integer (%x).", 543.21, 543.21, 543.21, 543.21, 543.21 );
```

This code displays:

Here's the number 543.21 as a float (543.210000), a binary integer (1000011111), an octal integer (1037), and a hex integer (21f).

Sign specifier

Ordinarily, printf() only puts a sign symbol in front of negative numbers, not positive numbers:

```
printf( "%d", 36 );   // Displays "36"
printf( "%d", -36 );  // Displays "-36"
```

If you'd rather printf() also put a + symbol in front of positive numbers, you can add the ***sign specifier***, +, before the type specifier:

```
printf( "%+d", 36 );   // Displays "+36"
printf( "%+d", -36 );  // Displays "-36"
```

Padding

printf() lets you pad out an argument value to a fixed width. You can use any character you like for the padding, and you can pad to the left or the right of the value. Padding is useful for adding leading zeroes to numbers, and for right-aligning strings.
To add padding, insert a ***padding specifier*** between the '%' character and the type specifier. A padding specifier takes the format:

<padding character><width>
<padding character> can be a zero or a space. If you miss it out, spaces are used. If you want to pad using a different character, write an apostrophe (') followed by the character to use.
<width> is the number of characters to pad the value out to. A positive number adds padding to the left; a negative number adds padding to the right.
Here are some padding examples:

```
printf( "%04d", 12 );         // Displays "0012"
printf( "%04d", 1234 );       // Displays "1234"
printf( "%04d", 12345 );      // Displays "12345"
printf( "% 10s", "Hello" );   // Displays "     Hello"
printf( "%10s", "Hello" );    // Displays "     Hello"
printf( "%'*10s", "Hello" );  // Displays "*****Hello"
printf( "%'*-10s", "Hello" ); // Displays "Hello*****"
```

Notice that, in the 3rd example, the padding specifier doesn't truncate the value to 4 characters. Padding specifiers only add characters.

Number precision

When using the f or F type specifier to format floats, PHP defaults to a precision of 6 decimal places:

```
printf( "%f", 123.456 );  // Displays "123.456000"
```

To specify a different precision, you can use a ***precision specifier***. This is a dot (.) followed by the number of decimal places to use, and it goes right before the type specifier. For example:

```
printf( "%.2f", 123.456 );  // Displays "123.46"
printf( "%.10f", 123.456 );  // Displays "123.4560000000"
printf( "%.0f", 123.456 );  // Displays "123"
```

If you use a padding specifier with a precision specifier then printf() pads the entire value, including the decimal point and decimal digits, to the specified length:

```
printf( "%08.2f", 123.456 );  // Displays "00123.46"
```

If you use a precision specifier with the s type specifier then printf()truncates the string value to the specified number of characters:

```
printf( "%.2s", "Hello" );  // Displays "He"
```

Argument swapping

By default, the first format specifier in the format string is used with the first argument after the format string, the second format specifier is used with the second argument, and so on. However, you can change this ordering if you like.
To do this, place a number followed by a dollar ($) symbol between the %and the type specifier. For example:

```
// Displays "Australia comprises 10 territories and 6 states"
printf( 'Australia comprises %2$d territories and %1$d states', 6, 10 );
```

In the above example, the first format specifier is %2$d. This means: "Take the second argument after the format string and display it as a decimal integer". The second format specifier, %1$d, reads: "Take the first argument after the format string and display it as a decimal integer". So the arguments are used in a different order.
In the above example, the format string is enclosed by single quotes rather than double quotes. This prevents each dollar ($) symbol in the string from being interpreted as starting a PHP variable name. (You can find out more about this in Creating PHP Strings.)

Storing the result in a variable

So what about sprintf()? This function is identical to printf(), except that rather than directly outputting the result, it returns it so that you can store it in a variable (or otherwise manipulate it). This is useful if you want to process the result before displaying it, or store it in a database. Here's an example:

```
$result = sprintf( "Australia comprises %d states and %d territories", 6, 10 );

// Displays "Australia comprises 6 states and 10 territories"
echo $result;
```

Building yourself some arrays:

An array stores multiple values in one single variable:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?
```

What is an Array?

An array is a special variable, which can hold more than one value at a time.

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

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

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is to create an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

Create an Array in PHP

In PHP, the array() function is used to create an array:

array();

In PHP, there are three types of arrays:

- **Indexed arrays** - Arrays with a numeric index
- **Associative arrays** - Arrays with named keys
- **Multidimensional arrays** - Arrays containing one or more arrays

PHP Indexed Arrays

There are two ways to create indexed arrays:

The index can be assigned automatically (index always starts at 0), like this:

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

or the index can be assigned manually:

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

The following example creates an indexed array named $cars, assigns three elements to it, and then prints a text containing the array values:

Example

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>
```

Output:

```
I like Volvo, BMW and Toyota.
```

Modifying the data in arrays:

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, resulting in an empty pair of brackets (*[]*).

```
$arr[key] = value;
$arr[] = value;
// key may be an integer or string
// value may be any value of any type
```

If $arr doesn't exist yet, it will be created, so this is also an alternative way to create an array. This practice is however discouraged because if $arr already contains some value (e.g. string from request variable) then this value will stay in the place and *[]* may actually stand for string access operator. It is always better to initialize a variable by a direct assignment.

Deleting array elements:

There are different ways to delete an array element, where some are more useful for some specific tasks than others.

Delete one array element

If you want to delete just one array element you can use unset() or alternative array_splice().
Also if you have the value and don't know the key to delete the element you can use array_search()to get the key.
unset() method
that when you use unset() the array keys won't change/reindex. If you want to reindex the keys you can use array_values() after unset() which will convert all keys to numerical enumerated keys starting from 0.

Eg:
```
<?php

   $array = array(0 => "a", 1 => "b", 2 => "c");
   unset($array[1]);
              //↑ Key which you want to delete

?>
```

Output
```
Array (
   [0] => a
   [2] => c
)
```

Handling arrays with loops:

already know you can loop over an array using a for loop and the count function, which determines how many elements an array contains:

the for loop:

```
<?php
   $fruits[0] = "pineapple";
   $fruits[1] = "pomegranate";
   $fruits[2] = "tangerine";
   for ($index = 0; $index < count($fruits); $index++)
{
  echo $fruits[$index], "\n";

  }

?>
```

Here's what you get:

```
pineapple
pomegranate
tangerine
```

the print_r function:

There's also a function for easily displaying the contents of an array, print_r:

```
<?php
   $fruits[0] = "pineapple";
   $fruits[1] = "pomegranate";
   $fruits[2] = "tangerine";
   print_r($fruits);
?>
```

Here are the results:

```
Array
(
   [0] => pineapple
   [1] => pomegranate
   [2] => tangerine
)
```

The foreach loop:

The foreach statement was specially created to loop over collections such as arrays. This statement has two forms:

```
foreach (array_expression as $value) statement
foreach (array_expression as $key => $value) statement
```

The first form of this statement assigns a new element from the array to $value each time through the loop. The second form places the current element's key, another name for its index, in $key and its value in $value each time through the loop. For example, here's how you can display all the elements in an array using foreach:

```
<?php
```

```
    $fruits = array("pineapple", "pomegranate", "tangerine");
    foreach ($fruits as $value) {
        echo "Value: $value\n";
    }
?>
```

Here are the results:

```
Value: pineapple
Value: pomegranate
Value: tangerine
```

And here's how you can display both the keys and values of an array:

```
<?php
    $fruits = array("pineapple", "pomegranate", "tangerine");

    foreach ($fruits as $key => $value) {
        echo "Key: $key; Value: $value\n";
    }
?>
```

Here are the results:

```
Key: 0; Value: pineapple
Key: 1; Value: pomegranate
Key: 2; Value: tangerine
```

The While loop:

You can even use a while loop to loop over an array if you use a new function, *each* . The each function is meant to be used in loops over collections such as arrays; each time through the array, it returns the current element's key and value and then moves to the next element. To handle a multiple-item return value from an array, you can use the *list* function, which will assign the two return values from each to separate variables.
Here's what this looks like for our $fruits array:

```
<?php
    $fruits = array("pineapple", "pomegranate", "tangerine");

    while (list($key, $value) = each ($fruits)) {
        echo "Key: $key; Value: $value\n";
    }
?>
```

Here's what you get from this script:

```
Key: 0; Value: pineapple
Key: 1; Value: pomegranate
Key: 2; Value: tangerine
```

The PHP array functions:

Just as it has many string functions, PHP also has many array functions. You can see a sample of them in

Function Name	**Purpose**
array_chunk	Splits an array into chunks
array_combine	Creates an array by using one array for the keys and another for the values
array_count_values	Counts the values in an array
array_diff	Computes the difference of arrays
array_fill	Fills an array with values
array_intersect	Computes the intersection of arrays
array_key_exists	Checks whether the given key or index exists in the array
array_keys	Returns the keys in an array
array_merge	Merges two or more arrays
array_multisort	Sorts multiple or multidimensional arrays
array_pad	Pads array to the specified length with a value
array_pop	Pops the element off the end of an array
array_push	Pushes one or more elements onto the end of array
array_rand	Picks one or more random elements out of an array
array_reduce	Reduces the array to a single value with a callback function
array_reverse	Returns an array with elements in reverse order
array_search	Searches the array for a given value and returns the corresponding key
array_shift	Shifts an element off the beginning of array
array_slice	Extracts a slice of the array
array_sum	Calculates the sum of values in an array
array_unique	Removes duplicate elements from an array
array_unshift	Adds one or more elements to the beginning of an array
array_walk	Calls a user-supplied function on every member of an array
array	Creates an array

Function Name	Purpose
asort	Sorts an array and maintains index association
count	Counts the elements in an array
current	Returns the current element in an array
each	Returns the current key and value pair from an array and advances the array cursor
in_array	Checks whether a value exists in an array
key	Gets a key from an associative array
krsort	Sorts an array by key in reverse order
ksort	Sorts an array by key
list	Assigns variables as if they were an array
natcasesort	Sorts an array using a case-insensitive "natural order" algorithm
natsort	Sorts an array using a "natural order" algorithm
pos	Alias of the current function
reset	Sets the pointer of an array to its first element
rsort	Sorts an array in reverse order
shuffle	Shuffles an array's elements
sizeof	Alias of the count function
sort	Sorts an array
usort	Sorts an array by values with a user-defined comparison function

Converting between strings and arrays using impload and explpoad:

strings and arrays by using the PHP *implode* and *explode* functions: implode implodes an array to a string, and explode explodes a string into an array.
For example, say you want to put an array's contents into a string. You can use implode, passing it the text you want to separate each element with in the output string (in this example, we use a comma) and the array to work on:

```
<?php
   $vegetables[0] = "corn";
   $vegetables[1] = "broccoli";
   $vegetables[2] = "zucchini";
   $text = implode(",", $vegetables);
```

```
    echo $text;
?>
```
This gives you:

```
corn,broccoli,zucchini
```
There are no spaces between the items in this string, however, so we change the separator string from "," to ", ":

```
$text = implode(", ", $vegetables);
```
The result is:

```
corn, broccoli, zucchini
```
What about exploding a string into an array? To do that, you indicate the text that you want to split the string on, such as ", ", and pass that to explode. Here's an example:

```
<?php
    $text = "corn, broccoli, zucchini";
    $vegetables = explode(", ", $text);
    print_r($vegetables);
?>
```
And here are the results. As you can see, we exploded the string into an array correctly:

```
Array
(
    [0] => corn
    [1] => broccoli
    [2] => zucchini
)
```

Extracting Variables from Arrays:

The *extract* function is handy for copying the elements in arrays to variables if your array is set up with string index values. For example, take a look at this case, where we have an array with string indexes:

```
$fruits["good"] = "tangerine";
$fruits["better"] = "pineapple";
$fruits["best"] = "pomegranate";
    .
    .
    .
```
When you call the extract function on this array, it creates variables corresponding to the string indexes: $good, $better, and so on:

```
$fruits["good"] = "tangerine";
$fruits["better"] = "pineapple";
$fruits["best"] = "pomegranate";

extract($fruits);
    .
    .
    .
```
Take a look at how this works in Example 3-3, phpextract.php.

Example 3-3. Extracting variables from an array, phpextract.php

```
<HTML>
        <HEAD>
            <TITLE>Extracting variables from an array</TITLE>
```

```
    </HEAD>

    <BODY>
      <H1>Extracting variables from an array</H1>
    <?php
      $fruits["good"] = "tangerine";
      $fruits["better"] = "pineapple";
      $fruits["best"] = "pomegranate";

      extract($fruits);


  echo "\$good = $good<BR>";

  echo "\$better = $better<BR>";

  echo "\$best = $best<BR>";
    ?>
   </BODY>
</HTML>
```

Now $good will hold "tangerine", $better will hold "pineapple", and $best will hold "pomegranate". You can see the results in Figure 3-3.

Figure 3-3 Filling variables from an array.

You can also use the PHP list function to get data from an array like this and store it in as many variables as you like. Here's an example:

```
<?php
   $vegetables[0] = "corn";
   $vegetables[1] = "broccoli";
   $vegetables[2] = "zucchini";
   list($first, $second) = $vegetables;
   echo $first, "\n";
   echo $second;
?>
```

And here is the result:

```
corn
broccoli
```

Can you go the opposite way and copy variables into an array? Sure, just use the *compact* function. You pass this function the *names* of variables (with the $), and compact finds those variables and stores them all in an array:

```
<?php
   $first_name  = "Cary";
   $last_name = "Grant";
   $role = "Actor";
   $subarray = array("first_name", "last_name");
   $resultarray = compact("role", $subarray);
?>
```

Sorting Arrays:

PHP offers all kinds of ways to sort the data in arrays, starting with the simple sort function, which you use on arrays with numeric indexes. In the following example, we create an array, display it, sort it, and then display it again:

```
<?php
   $fruits[0] = "tangerine";
   $fruits[1] = "pineapple";
   $fruits[2] = "pomegranate";

   print_r($fruits);


  sort($fruits);


  print_r($fruits);
?>
```

Here are the results—as you can see, the new array is in sorted order; note also that the elements have been given new numeric indexes:

```
Array
(
   [0] => tangerine
   [1] => pineapple
   [2] => pomegranate
)
Array
(
   [0] => pineapple
   [1] => pomegranate
   [2] => tangerine
)
```

You can sort an array in reverse order if you use rsort instead:

```
<?php
   $fruits[0] = "tangerine";
   $fruits[1] = "pineapple";
   $fruits[2] = "pomegranate";

   print_r($fruits);

  rsort($fruits);

  print_r($fruits);
?>
```

Here's what you get:

```
Array
(
   [0] => tangerine
```

```
    [1] => pineapple
    [2] => pomegranate
)
Array
(
    [0] => tangerine
    [1] => pomegranate
    [2] => pineapple
)
```

What if you have an array that uses text keys? Unfortunately, if you use sort or rsort, the keys are replaced by numbers. If you want to retain the keys, use asort instead, as in this example:

```
<?php
    $fruits["good"] = "tangerine";
    $fruits["better"] = "pineapple";
    $fruits["best"] = "pomegranate";

    print_r($fruits);


   asort($fruits);


   print_r($fruits);
?>
```

Here are the results:

```
Array
(
    [good] => tangerine
    [better] => pineapple
    [best] => pomegranate
)
Array
(
    [better] => pineapple
    [best] => pomegranate
    [good] => tangerine
)
```

Using php's array operators:

This is a Comprehensive PHP array operators tutorial from w3resource.com

List of array operators

Name	**Example**	**Result**
Union	$x + $y	Union of $x and $y. The + operator appends elements of remaining keys from the right-sided array to the left-handed, but duplicated keys are not overwritten.

Equality	$x == $y	TRUE if $x and $y have the same key/value pairs.
Identity	$x === $y	TRUE if $x and $y have the same key/value pairs in the same order and of the same types.
Inequality	$x != $y	TRUE if $x is not equal to $y.
Inequality	$x <> $y	TRUE if $x is not equal to $y.
Non-identity	$x !== $y	TRUE if $x is not identical to $y.

Example : array equality (==) and identity(===) operators
In the following example equality operator returns true as the two arrays have same key/value pairs whereas identity operator returns false as the key/value of the comparing arrays are same but not in same order.
view plaincopy to clipboardprint?

```
<?php
$a = array("1" => "apple", "0" => "banana");
$b = array( "banana", "apple");
var_dump($a == $b);
var_dump($a === $b);
?>
bool(true) bool(false)
```

comparing arrays to each other:

it seems that every PHP function I read about for comparing arrays (array_diff(), array_intersect(), etc) compares for the **existence** of array elements.
Given two multidimensional arrays with identical structure, how would you list the differences in **values**?
Example
Array 1

```
[User1] => Array ([public] => 1
             [private] => 1
             [secret] => 1
            )
[User2] => Array ([public] => 1
             [private] => 0
             [secret] => 0
            )
```

Array 2

```
[User1] => Array ([public] => 1
             [private] => 0
             [secret] => 1
            )
[User2] => Array ([public] => 1
             [private] => 0
             [secret] => 0
             )
```
Difference

```
[User1] => Array ([public] => 1
             [private] => 0 //this value is different
             [secret] => 1
            )
```
So my result would be - "Of all the users, User1 has changed, and the difference is that private is 0 instead of 1."

Using multidimensional arrays in loops:

Array elements in PHP can hold values of any type, such as numbers, strings and objects. They can also hold other arrays, which means you can create ***multidimensional***, or ***nested***, arrays.
In this tutorial you learn how to create multidimensional arrays, how to access elements in a multidimensional array, and how to loop through multidimensional arrays.

How to create a multidimensional array

You create a multidimensional array using the array() construct, much like creating a regular array. The difference is that each element in the array you create is itself an array.
For example:

```
$myArray = array(
  array( value1, value2, value3 ),
  array( value4, value5, value6 ),
  array( value7, value8, value9 )
);
```
Example

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold: ".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold: ".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold: ".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold: ".$cars[3][2].".<br>";
?>
```
```
Volvo: In stock: 22, sold: 18.
BMW: In stock: 15, sold: 13.
Saab: In stock: 5, sold: 2.
Land Rover: In stock: 17, sold: 15.
```

Moving through arrays:

an array element to a new position in the array and resequence the indexes so that there are no gaps in the sequence. It doesnt need to work with associative arrays. Anyone got ideas for this one?

```
$a = array(0=>'a', 1=>'c', 2=>'d', 3=>'b', 4=>'e');
print_r(moveElement(3,1))
//should output [0=>'a', 1=>'b', 2=>'c', 3=>'d', 4=>'e']
```

Merging and Splitting Arrays:

You can also cut up and merge arrays when needed. For example, say you have a three-item array of various fruits and want to get a subarray consisting of the last two items. You can do this with the array_slice function, passing it the array you want to get a section of, the *offset* at which to start, and the *length* of the array you want to create:

```
<?php
   $fruits["good"] = "tangerine";
   $fruits["better"] = "pineapple";
   $fruits["best"] = "pomegranate";
   $subarray = array_slice($fruits, 1, 2);
   foreach ($subarray as $value) {
      echo "Fruit: $value\n";
   }
?>
```

Here are the results:

```
Fruit: pineapple
Fruit: pomegranate
```

If offset is negative, the sequence will be measured from the end of the array. If length is negative, the sequence will stop that many elements from the end of the array.

If you don't give the length of the subarray you want, you'll get all the elements to the end (or the beginning, if you're going in the opposite direction) of the array.

You can also merge two or more arrays with array_merge:

```
<?php
   $fruits = array("pineapple", "pomegranate", "tangerine");
   $vegetables = array("corn", "broccoli", "zucchini");

   $produce = array_merge($fruits, $vegetables);

   foreach ($produce as $value) {
      echo "Produce item: $value\n";
   }
?>
```

And here's what you get (see also "Using the Array Operators" in this chapter):

```
Produce item: pineapple
Produce item: pomegranate
Produce item: tangerine
Produce item: corn
Produce item: broccoli
Produce item: zucchini
```

Other array function:

avigating Within an Array: each(), current(), reset(), end(), next(), pos(), and prev()

We mentioned previously that every array has an internal pointer that points to the current element in the array. You indirectly used this pointer earlier when using the each() function, but you can directly use and manipulate this pointer.

If you create a new array, the current pointer is initialized to point to the first element in the array. Calling current($array_name) returns the first element.

Calling either next() or each() advances the pointer forward one element. Calling each($array_name) returns the current element before advancing the pointer. The function next()behaves slightly differently: Calling next($array_name) advances the pointer and then returns the new current element. You have already seen that reset() returns the pointer to the first element in the array. Similarly, calling end($array_name) sends the pointer to the end of the array. The first and last elements in the array are returned by reset() and end(), respectively.
To move through an array in reverse order, you could use end() and prev(). The prev() function is the opposite of next(). It moves the current pointer back one and then returns the new current element.
For example, the following code displays an array in reverse order:

```
$value = end ($array);
while ($value)
{
 echo "$value<br />";
 $value = prev($array);
}
```

For example, you can declare $array like this:

```
$array = array(1, 2, 3);
```

In this case, the output would appear in a browser as follows:

```
3
2
1
```

Unit- II

CREATING FUNCTIONS

Creating functions in php:

PHP has a lot of built-in functions, addressing almost every need. More importantly, though, it has the capability for you to define and use your own functions for whatever purpose. The syntax for making your own function is

```
function function_name () {
  // Function code.
}
```

The name of your function can be any combination of letters, numbers, and the underscore, but it must begin with either a letter or the underscore. The main restriction is that you cannot use an existing function name for your function (*print, echo, isset*, and so on).
In PHP, as I mentioned in the first chapter, function names are case-insensitive (unlike variable names), so you could call that function using function_name() or FUNCTION_NAME() or function_Name(), etc.
The code within the function can do nearly anything, from generating HTML to performing calculations.
In this chapter, I'll demonstrate many different uses.
To create your own function
Create a new PHP document in your text editor

```
<?php # Script 3.7 - dateform.php
$page_title = 'Calendar Form';
include ('./includes/header.html');
```

passing functions some data:

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

Example

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

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

Output:

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

passing arrays to functions:

to learn how to pass arrays through a function, so that I can get around PHP's inability to return multiple values.

```
function foo($array)
{
   $array[3]=$array[0]+$array[1]+$array[2];
   return $array;
}

$waffles[0]=1;
$waffles[1]=2;
$waffles[2]=3;
foo($waffles);

echo $waffles[3];
```

passing by reference:

pass a variable by reference to a function so the function can modify the variable. The syntax is as follows:

```
<?php
function foo(&$var)
{
   $var++;
}

$a=5;
foo($a);
// $a is 6 here
?>
```

There is no reference sign on a function call - only on function definitions. Function definitions alone are enough to correctly pass the argument by reference. As of PHP 5.3.0, you will get a warning saying that "call-time pass-by-reference" is deprecated when you use & in *foo(&$a);*. And as of PHP 5.4.0, call-time pass-by-reference was removed, so using it will raise a fatal error.

The following things can be passed by reference:

Variables, i.e. *foo($a)*

New statements, i.e. *foo(new foobar())*

References returned from functions, i.e.:

```
<?php
function foo(&$var)
{
   $var++;
}
function &bar()
{
   $a = 5;
   return $a;
}
foo(bar());
?>
```

See more about returning by reference.

No other expressions should be passed by reference, as the result is undefined. For example, the following examples of passing by reference are invalid:

```
<?php
function foo(&$var)
{
   $var++;
}
function bar() // Note the missing &
{
   $a = 5;
   return $a;
}
foo(bar()); // Produces fatal error as of PHP 5.0.5, strict standards notice
```

```
        // as of PHP 5.1.1, and notice as of PHP 7.0.0

foo($a = 5); // Expression, not variable
foo(5); // Produces fatal error
?>
```

Using default arguments:

The default value must be a constant expression, not (for example) a variable, a class member or a function call.
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. Consider the following code snippet:
Example #5 Incorrect usage of default function arguments

```
<?php
function makeyogurt($type = "acidophilus", $flavour)
{
   return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt("raspberry");   // won't work as expected
?>
```

The above example will output:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/phptest/functest.html on line 41
Making a bowl of raspberry .
```

Passing variable numbers of argument:

arguments in an array, you might be interested by the call_user_func_arrayfunction.
If the number of arguments you want to pass depends on the length of an array, it probably means you can pack them into an array themselves -- and use that one for the second parameter of call_user_func_array.
Elements of that array you pass will then be received by your function as distinct parameters.

For instance, if you have this function :

```
function test() {
  var_dump(func_num_args());
  var_dump(func_get_args());
}
```

You can pack your parameters into an array, like this :

```
$params = array(
  10,
  'glop',
  'test',
);
```

And, then, call the function :

```
call_user_func_array('test', $params);
```

This code will the output :

```
int 3
```

```
array
  0 => int 10
  1 => string 'glop' (length=4)
  2 => string 'test' (length=4)
```

Returning data from functions:

Values are returned by using the optional return statement. Any type may be returned, including arrays and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called. See return for more information.
Note:
If the return is omitted the value **NULL** will be returned.
Use of return
Example #1 Use of return

```
<?php
function square($num)
{
    return $num * $num;
}
echo square(4);   // outputs '16'.
?>
```

Returning arrays:

In PHP you can return one and only one value from your user functions, but you are able to make that single value an array, thereby allowing you to return many values.
This following code shows how easy it is:

```
<?php
    function dofoo() {
        $array["a"] = "Foo";
        $array["b"] = "Bar";
        $array["c"] = "Baz";
        return $array;
    }

    $foo = dofoo();
?>
```

Without returning an array, the only other way to pass data back to the calling script is by accepting parameters by reference and changing them inside the function. Passing arrays by reference like this is generally preferred as it is less of a hack, and also frees up your return value for some a true/false to check whether the function was successful.

Returning lists:

list — Assign variables as if they were an array

Description ¶
array list (mixed $var1 [, mixed $...])
Like array(), this is not really a function, but a language construct. list() is used to assign a list of variables in one operatio

Returning reference:

Returning by reference is useful when you want to use a function to find to which variable a reference should be bound. Do *not* use return-by-reference to increase performance. The engine will automatically optimize this on its own. Only return references when you have a valid technical reason to do so. To return references, use this syntax:

```
<?php
class foo {
    public $value = 42;

    public function &getValue() {
        return $this->value;
    }
}

$obj = new foo;
$myValue = &$obj->getValue(); // $myValue is a reference to $obj->value, which is 42.
$obj->value = 2;
echo $myValue;              // prints the new value of $obj->value, i.e. 2.
?>
```

In this example, the property of the object returned by the *getValue* function would be set, not the copy, as it would be without using reference syntax.

Introducing variable scope in php:

Scope can be defined as the range of availability a variable has to the program in which it is declared. PHP variables can be one of four scope types:

1. Local variables
2. Global variables
3. Static variables
4. Function parameters

1. **Local variables** :
   A variable declared within a PHP function is local and can only be accessed within that function. (the variable has local scope): The script above will not produce any output because the echo statement refers to the local scope variable $a, which has not been assigned a value within this scope. You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared. Local variables are deleted as soon as the function is completed.

2. **Global variables :**
   Global scope refers to any variable that is defined outside of any function. Global variables can be accessed from any part of the script that is not inside a function. To access a global variable from within a function, use the global keyword: The script above will output 15. PHP also stores all global variables in an array called

$GLOBALS[index]. Its index is the name of the variable. This array is also accessible from within functions and can be used to update global variables directly. The example above can be rewritten as this

3. **Static variables**:
 When a function is completed, all of its variables are normally deleted. However, sometimes you want a local variable to not be deleted. To do this, use the static keyword when you first declare the variable: static $rememberMe; Then, each time the function is called, that variable will still have the information it contained from the last time the function was called. Note: The variable is still local to the function

4. **Function parameters**:
 A parameter is a local variable whose value is passed to the function by the calling code. Parameters are declared in a parameter list as part of the function declaration: function myTest($para1,$para2,...) { // function code } Parameters are also called arguments.

Accessing global data:

```
<?PHP
  class A {
    function Show(){
      echo "ciao";
    }
  }

  $a = new A();
  $b = new B();

  class B {
    function __construct() {
      $a->Show();
    }
  }
?>
```

With a bit of surprise I cannot access the globally defined $a variable from within the class and I get a Undefined variable exception

Working with a staic variable:

The final type of variable scoping that I discuss is known as static. In contrast to the variables declared as function parameters, which are destroyed on the function's exit, a static variable will not lose its value when the function exits and will still hold that value should the function be called again.
You can declare a variable to be static simply by placing the keyword STATIC in front of the variable name.

```
<?php
  function keep_track() {
    STATIC $count = 0;
    $count++;
    print $count;
    print "<br />";
  }

  keep_track();
  keep_track();
```

```
   keep_track();
?>
```

This will produce the following result −

```
1
2
3
```

PHP conditional functions:

When a function is defined in a conditional manner such as the two examples shown. Its definition must be processed prior to being called.

Example #2 Conditional functions

```
<?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 variable functions:

PHP supports the concept of variable functions. This means that 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. Among other things, this can be used to implement callbacks, function tables, and so forth.

Variable functions won't work with language constructs such as echo, print, unset(), isset(), empty(), include,require and the like. Utilize wrapper functions to make use of any of these constructs as variable functions.

Example #1 Variable function example

```
<?php
function foo() {
   echo "In foo()<br />\n";
}

function bar($arg = '')
{
   echo "In bar(); argument was '$arg'.<br />\n";
}

// 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()
?>
```

Nesting functions:

When you define a function within another function it does not exist until the parent function is executed. Once the parent function has been executed, the nested function is defined and as with any function, accessible from anywhere within the current document. If you have nested functions in your code, you can only execute the outer function once. Repeated calls will try to redeclare the inner functions, which will generate an error.

Eg:

```
<?php
function outer( $msg ) {
   function inner( $msg ) {
      echo 'inner: '.$msg.' ';
   }
   echo 'outer: '.$msg.' ';
   inner( $msg );
}

inner( 'test1' );  // Fatal error:  Call to undefined function inner()
```

```
outer( 'test2' );  // outer: test2 inner: test2
inner( 'test3' );  // inner: test3
outer( 'test4' );  // Fatal error:  Cannot redeclare inner()
```

created include file:

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.
Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.
Include files can make sites easier to manage. For example, you can create a piece of content such as a page banner, site information block, or menu that you want to include on multiple pages in your site. When you want to change the content, you can make the change in a single file, and the change will be reflected on every page in which the include file appears.
A PHP include statement is a code block that pulls content from an external file into a web page. The following is an example of a PHP include statement:
<?php include('pageBanner.php'); ?>

There are two basic PHP include functions: **include()** and **require()**. Both behave in the same way, but return different errors. An **include()** function, if not parsed correctly, will continue processing the rest of the page and display a warning in the page where the included file should appear. If a **require()** function refers to a missing file, the function will stop processing the page and display an error page in the browser. For more information about **include()** and **require()**, see PHP Include Files on the W3 Schools website.
The **include_once()** and **require_once()** functions specify that an include file be used only once in a page. If two **include()** functions refer to the same include file, only the first **include()** function will display in the browser. For more information, see include_ and require_once on the PHP.NET website.
Microsoft Expression Web supports using four different file types as include files: HTML, INC, PHP, and TXT files. In addition, you can also create nested include files. For example, you can create an include file which contains a reference to another include file.
To insert an include() function into a web page
On the **Insert** menu, click **PHP**, and then click one of the following **include()** functions:
Include   Includes the file each time it is referenced in the page.
Include_once   Includes the file the first time it is referenced in the page.
Require   Requires that the file be processed before the page is returned, and includes the file each time it is referenced in the page.
Require_once   Requires that the file be processed before the page is returned, and includes the file the first time it is referenced in the page.
In the **Select a PHP page** dialog box, select the file that you want to include, and then click **Open**.

Returning errors from functions:

Values are returned by using the optional return statement. Any type may be returned, including arrays and objects. This causes the function to end its execution immediately and pass control back to the line from which it was called. See return for more information.
Note:
If the return is omitted the value **NULL** will be returned.
Use of return
Example #1 Use of return

```
<?php
function square($num)
{
   return $num * $num;
}
echo square(4);   // outputs '16'.
?>
```

A function can not return multiple values, but similar results can be obtained by returning an array.

READING DATA IN WEB PAGES

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

Setting up web pages to communicate with PHP:

The HTML form we will be working at in these chapters, contains various input fields: required and optional text fields, radio buttons, and a submit button:

The validation rules for the form above are as follows:

Field	Validation Rules
Name	Required. + Must only contain letters and whitespace
E-mail	Required. + Must contain a valid email address (with @ and .)
Website	Optional. If present, it must contain a valid URL
Comment	Optional. Multi-line input field (textarea)
Gender	Required. Must select one

Handling Text Fields:

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

```
Name: <input type="text" name="name">
E-mail: <input type="text" name="email">
Website: <input type="text" name="website">
Comment: <textarea name="comment" rows="5" cols="40"></textarea>
```

Handling Text Areas:

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.
To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:
<html>
<body>

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

</body>
</html>

Handling check boxes:

Checkboxes are a little unwieldy from a data standpoint. Part of this is that there are essentially two different ways to think about their functionality. Frequently, a set of checkboxes represents a single question which the user can answer by selecting any number of possible answers. They are, importantly, not exclusive of each other. (If you want the user to only be able to pick a single option, use radio boxes or the <select> element.)

<form> <p>Check all the languages you have proficiency in.</p> <input type="checkbox" id="HTML" value="HTML"><label for="HTML"> HTML</label><br> <input type="checkbox" id="CSS" value="CSS"><label for="CSS"> CSS</label><br> <input type="checkbox" id="JS" value="JS"><label for="JS"> JS</label><br> <input type="checkbox" id="PHP" value="PHP"><label for="PHP"> PHP</label><br> <input type="checkbox" id="Ruby" value="Ruby"><label for="Ruby"> Ruby</label><br> <input type="checkbox" id="INTERCAL" value="INTERCAL"><label for="INTERCAL"> INTERCAL</label><br> </form>
Top of Form
Check all the languages you have proficiency in.
HTML
CSS
Bottom of Form

Read more: https://html.com/input-type-checkbox/#ixzz4yaHoKVfD

Handling Radio Buttons:

The gender fields are radio buttons and the HTML code looks like this:

Gender:
<input type="radio" name="gender" value="female">Female
<input type="radio" name="gender" value="male">Male

Handling list boxes:

List Box - Single Select

```
<p>List Box - Single Select<br>
<select name="listbox" size="3">
<option value="Option 1" selected>Option 1</option>
<option value="Option 2">Option 2</option>
<option value="Option 3">Option 3</option>
<option value="Option 4">Option 4</option>
<option value="Option 5">Option 5</option>
</select>
</p>
```

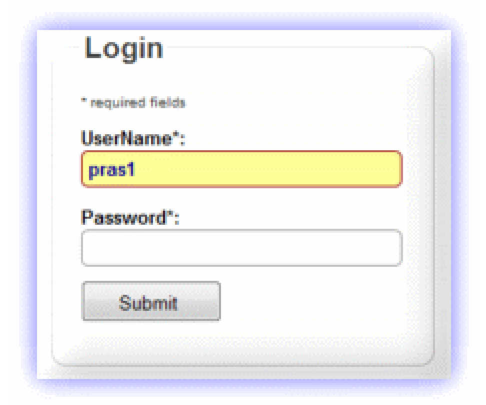
This form is processed by the listbox.php script. You can see the code for this script as follows:
ListBox.php

```
<html>
<head>
<title>List Box Form Data</title>
</head>
<body>
<h3>List Box Form Data</h3>
<p>Form data passed from the form</p>
   <?php
       echo "<p>select: " . $_POST['select']."</p>\n";
       echo "<p>listbox: " . $_POST['listbox'] . "</p>\n";
       $values = $_POST['listmultiple'];
       echo "<p>listmultiple: ";
       foreach ($values as $a){
           echo $a;
       }
       echo "</p>\n";
   ?>
</body>
</html>
```

Handling password controls:

This is in continuation of the tutorial on making a membership based web site. Please see the previous page PHP registration form for more details.

The login form



pras1

Here is the HTML code for the login form.
view source
print?

```
<form id='login' action='login.php' method='post' accept-charset='UTF-8'>
<fieldset >
<legend>Login</legend>
<input type='hidden' name='submitted' id='submitted' value='1'/>

<label for='username' >UserName*:</label>
<input type='text' name='username' id='username'  maxlength="50" />

<label for='password' >Password*:</label>
                              <input type='password' name='password' id='password' maxlength="50" />

<input type='submit' name='Submit' value='Submit' />

</fieldset>
</form>
```

```
function Login()
          {
    if(empty($_POST['username']))
    {
        $this->HandleError("UserName is empty!");
        return false;
    }

     if(empty($_POST['password']))
     {
         $this->HandleError("Password is empty!");
         return false;
     }

     $username = trim($_POST['username']);
     $password = trim($_POST['password']);

     if(!$this->CheckLoginInDB($username,$password))
     {
         return false;
     }

     session_start();

     $_SESSION[$this->GetLoginSessionVar()] = $username;

     return true;
 }
```

In order to identify a user as authorized, we are going to check the database for his combination of username/password, and if a correct combination was entered, we set a session variable.

Handling hidden controls:

For hidden fields, can i use a field of the type
<input type="hidden" name="field_name" value="<?php print $var; ?>"/>
and retrieve it after the GET / POST method as $_GET['field_name'] / $_POST['field_name'] ?

Handling image maps:

want to add some interactivity to your images, you may consider using image maps. With image maps, you can define multiple clickable regions on a single graphic. To define clickable regions on a single image, set up hotspots within a single image. A **hotspot** is a defined area on an image that acts as a hypertext link. The hotspots are defined through the use of image maps. **Image maps** list the coordinates that define the boundaries of the hotspots (or the regions that act as hypertext links) on an image.
The whole idea behind using image maps is to link one image to multiple destinations. In how to insert graphics page, we discuss how to link one image to just one destination. On this page, you will learn how to create image maps or multiple hyperlinks on a single image. There are two types of image maps:

- **server-side image maps**
- **client-side image maps**
- **Server-side image maps**

In a server-side image map, the server controls the image map. A server is a computer that store web pages and serves those pages when a client requests a page. When we use a server-side image map, we define the coordinates of the hotspots in a server-side script. Whenever a user clicks on a hotspot on

an inline image, the appropriate coordinates are sent back to the server to activate the appropriate hyperlink. One of the main drawbacks of using server-side image map is that server-side image maps can be slow to operate. This is so because every time a user clicks on an inline image map, that information has to be sent to the server and then the server has to process that request.
Client-side image maps
In a client-side image map, the image map is defined in an HTML file and that is processed by the browser locally. Because client-side image maps are processed locally, they tend to be more responsive than server-side image maps. Thus client-side image maps can be tested using a local computer; whereas, to test the server-side image maps, you'll need a server. To show you how to create image maps, we will use client-side image maps so you can easily test without using a sever!
There are two easy steps to create an image map:
define image map hotspots
use the image map
Defining image map hotspots
To create image an image map, you need coordinates of the points corresponding to the hotspot boundaries. In other words, you will define an area, by using coordinates, for each hyperlink that you want on an inline image. To find coordinates for a specific area for an image, you will need a special program that shows you coordinates. As an example, Macromedia's Dreamweaver 2004 allows you to create image maps by letting you draw the areas on an image. For each area you draw, the program will write the appropriate coordinates in your web page code file.
For our example, we will show you the coordinates for each area that we want to define on an inline image. The general syntax for an image map tag is:
<map name="mapName">
<area shape="areaShape" coords="coordinates" href="URL">
</map>
So an image map is defined with the <map> tag. The *name* attribute inside the <map> tag gives a name to the image map. To be able to use an image map, we must assign a name to an image map. Within the <map> tag, we use the <area> tag to specify the areas of the image that will act as hotspots. We can include as many <area> tags within the <map> tag we choose. Each of the <area> tag will act as a seperate hyperlink.
The <area> tag has three attributes:
shape – refers to the type of shape you want for the hotspot. You have three choices for the shape: **rect**, **circle**, and **poly**.
coords – refers to the coordinates for the location of a hotspot. The value for this attibute depend on the type of shape you want. The coordinates are expressed as a point's distance in pixels from the left and the top edges of an inline image. The coordinates (0,0) refers to a point where the image starts to get displayed. For instance, the coordinates (31,9) refer to a point that is 31 pixels from the left edge and 9 pixels down from the top on an inline image.
href – refers to the URL of the hypertext link that the hotspot points to.
Access these pages to learn how to create

- rectangular hotspot
- circular hotspot
- polygonal hotspot

Creating a rectangular hotspot
the upper-left corner coordinates are (6, 4) and the lower-right corner coordinates are (93, 38). The coordinates (6, 4) refer to a point on the image that is 6 pixels to the right and 4 pixels down from the top of the image. The coordinates (93, 38) refer to a point on the image that is 93 pixels to the right and 38 pixels down from the top of the image. Figure 1 shows where the image starts (coordinates (0, 0)), the

upper-left corner of the hotspot (coordinates (6, 4)), the lower-right corner of the hotspot (coordinates (93, 38)).
The following shows the HTML code to create the rectangular hotspot:

```
<map name="ScriptHTML">
<area shape="rect" coords="6,4,93,38" href="HTML-introduction.asp">
</map>
```

Finally, to add the image map to our web page, we need to use the *ScriptHTML* image map we defined above. To use an image map, simply add the *usemap* attribute to the image map graphic. For instance,

```
<img src="http://www.scriptingmaster/images/html/script-
html.GIF"usemap="#ScriptHTML" alt="Learn to script HTML">
```

Handling file uploads:

First, ensure that PHP is configured to allow file uploads.
In your "php.ini" file, search for the file_uploads directive, and set it to On:

```
file_uploads = On
```

Create The HTML Form

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

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

<form action="upload.php" method="post" enctype="multipart/form-data">
    Select image to upload:
    <input type="file" name="fileToUpload" id="fileToUpload">
    <input type="submit" value="Upload Image" name="submit">
</form>

</body>
</html>
```

Some rules to follow for the HTML form above:
Make sure that the form uses method="post"
The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form
Without the requirements above, the file upload will not work.
Other things to notice:
The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

Handling buttons:

Processing form data in PHP is significantly simpler than most other Web programming languages. This simplicity and ease of use makes it possible to do some fairly complex things with forms, including handling multiple submit buttons in the same form.

Processing form data in PHP is significantly simpler than most other Web programming languages. This simplicity and ease of use makes it possible to do some fairly complex things with forms, including handling multiple submit buttons in the same form.
'll begin with a simple example—a single form with a single submit button—to ensure you are clear on the basics and to provide a base for the more-complex example. Here is a form:

```
<html><head>Single-button form</head>
<body>

<form action="processor.php" method="post"> Enter a number: <input type="text" name="number"
size="3"> <br>
<input type="submit" name="submit"> </form>
```

To show the values in the input fields after the user hits the submit button, we add a little PHP script inside the value attribute of the following input fields: name, email, and website. In the comment textarea field, we put the script between the <textarea> and </textarea> tags. The little script outputs the value of the $name, $email, $website, and $comment variables.
Then, we also need to show which radio button that was checked. For this, we must manipulate the checked attribute (not the value attribute for radio buttons):

```
Name: <input type="text" name="name" value="<?php echo $name;?>">

E-mail: <input type="text" name="email" value="<?php echo $email;?>">

Website: <input type="text" name="website" value="<?php echo $website;?>">

Comment: <textarea name="comment" rows="5" cols="40"><?php echo $comment;?></textarea>

Gender:
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="female") echo "checked";?>
value="female">Female
<input type="radio" name="gender"
<?php if (isset($gender) && $gender=="male") echo "checked";?>
value="male">Male
```

PHP - Complete Form Example
Here is the complete code for the PHP Form Validation Example:
Example

```
</body>
</html>
```

And here is the processor.php script that gets invoked on submission:

```
<?php

// check for submission
// retrieve value from posted data
if ($_POST['submit'])
{
   echo "You entered the number " . $_POST['number']; }

?>
```
When a form is submitted to a PHP script, PHP automatically creates a special $_POST or $_GET associative array, depending on which method of submission was used (I'll assume POST throughout this tutorial). Values entered into the form input fields are automatically converted to key-value pairs in this array and can then be accessed using regular array notation.
Pay special attention to how the submit button is handled in the above script. When the form is submitted, the submit button itself becomes an element in the $_POST array, with a key corresponding to its "name". This is clearly visible by adding the line:
print_r($_POST);
PHP Form Validation Example
* required field.
Top of Form

Name: *

E-mail: *

Website:

Comment:

Gender: Female Male *

Submit

Bottom of Form
Your Input:

Output:
PHP Form Validation Example
* required field.
Top of Form

Name: *

E-mail: *

Website:

Comment:

Gender: ○ Female ○ Male *

Submit

Bottom of Form
Your Input:

PHP BROWSER-HANDLING POWER

Using php's server variables:

PHP 4 >= 4.1.0, PHP 5, PHP 7)
$_SERVER -- $HTTP_SERVER_VARS [removed] — Server and execution environment information
Description ¶
$_SERVER is an array containing information such as headers, paths, and script locations. The entries in this array are created by the web server. There is no guarantee that every web server will provide any of these; servers may omit some, or provide others not listed here. That said, a large number of these variables are accounted for in the » CGI/1.1 specification, so you should be able to expect those.

Using HTTP headers:

- '*HTTP_ACCEPT*'

Contents of the *Accept:* header from the current request, if there is one.

- '*HTTP_ACCEPT_CHARSET*'

Contents of the *Accept-Charset:* header from the current request, if there is one. Example: '*iso-8859-1,*,utf-8*'.

- '*HTTP_ACCEPT_ENCODING*'

Contents of the *Accept-Encoding:* header from the current request, if there is one. Example: '*gzip*'.

- '*HTTP_ACCEPT_LANGUAGE*'

Contents of the *Accept-Language:* header from the current request, if there is one. Example: '*en*'.

- '*HTTP_CONNECTION*'

Contents of the *Connection:* header from the current request, if there is one. Example: '*Keep-Alive*'.

- '*HTTP_HOST*'

Contents of the *Host:* header from the current request, if there is one.

- '*HTTP_REFERER*'

The address of the page (if any) which referred the user agent to the current page. This is set by the user agent. Not all user agents will set this, and some provide the ability to modify HTTP_REFERER as a feature. In short, it cannot really be trusted.

- '*HTTP_USER_AGENT*'

Contents of the *User-Agent:* header from the current request, if there is one. This is a string denoting the user agent being which is accessing the page. A typical example is: Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586). Among other things, you can use this value with get_browser() to tailor your page's output to the capabilities of the user agent.

Getting the user's browser type:

Look up the browscap.ini file and return the capabilities of the browser:

```
<?php
echo $_SERVER['HTTP_USER_AGENT'];
$browser = get_browser();
print_r($browser);
?>
```

Output:

```
Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.2988.0
Safari/537.36
```

Redirecting browsers with HTTP headers:

PHP/4.4.0 and Content-Type: text/html that were returned when we finally got to the homepage. PHP was designed from the beginning to output HTML (the 'H' in PHP stands for 'Hypertext'), and the first time a script generates output (e.g. by using echo), PHP automatically includes those headers for you. This is very convenient, but also contributes to the confusion many PHP beginners have regarding headers — in more 'bare bones' languages like Perl that were not originally designed for the web, sending output without including your own headers produces the dreaded '500 Internal Server Error', so Perl web programmers have no choice but to learn about headers immediately.
The header() function sends HTTP response headers; nothing more, nothing less.

Using this function, you can make your scripts send headers of your choosing to the browser, and create some very useful and dynamic results. However, the first thing you need to know about the header()function is that you have to use it **before** PHP has sent any output (and therefore its default headers).
I doubt there is a PHP programmer in the world who has never seen an error that looks like
Warning: Cannot modify header information - headers already sent by.....
As we said above, the response headers are separated from the content by a blank line. This means you can only send them *once*, and if your script has any output (even a blank line or space before your opening <?php tag), PHP does so without asking you. For example, consider the script below, which seems logical enough:

```
Welcome to my website!<br />
<?php
  if($test){
   echo "You're in!";
  }
  else{
    header('Location: http://www.mysite.com/someotherpage.php');
  }
?>
```

Dumping aform’s data all at once:

(PHP 4, PHP 5, PHP 7)
var_dump — Dumps information about a variable
Description ¶
void var_dump (mixed $expression [, mixed $...])
This function displays structured information about one or more expressions that includes its type and value. Arrays and objects are explored recursively with values indented to show structure.
All public, private and protected properties of objects will be returned in the output unless the object implements a __debugInfo() method (implemented in PHP 5.6.0).

Handling form data with custom arrays:

hen a form is submitted to a PHP script, the information from that form is automatically made available to the script. There are few ways to access this information, for example:
Example #1 A simple HTML form

```
<form action="foo.php" method="post">
    Name:  <input type="text" name="username" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" name="submit" value="Submit me!" />
</form>
```

As of PHP 5.4.0, there are only two ways to access data from your HTML forms.

Putting it all in one page:

I know that the way most people treat multiple forms on one page is to have each form post to another PHP file where the form is validated, its information is entered into a database or an email is sent off. So you usually have something like this:

```
   <form name="contactform" method="post" action="sendmail.php"> blah blah blah </form> <form
   name="mailinglist" method="post" action="join.php"> blah blah blah </form>
```

That work great, but why would you create all those extra files when you can just have the form post to the same file and create multiple functions to process your multiple forms.

The solution is very simple and super efficient.

Now lets put some PHP code before the

<head> tag to have different processes for each form.

```
  <?php
   if (!empty($_POST['mailing-submit']))
   { //do something here; }
  if (!empty($_POST['contact-submit']))
   { //do something here; }
   ?>
```

Performing data validation:

The first thing we will do is to pass all variables through PHP's htmlspecialchars() function.
When we use the htmlspecialchars() function; then if a user tries to submit the following in a text field:
<script>location.href('http://www.hacked.com')</script>
- this would not be executed, because it would be saved as HTML escaped code, like this:
&lt;script&gt;location.href('http://www.hacked.com')&lt;/script&gt;
The code is now safe to be displayed on a page or inside an e-mail.
We will also do two more things when the user submits the form:
Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
Remove backslashes (\) from the user input data (with the PHP stripslashes() function)
The next step is to create a function that will do all the checking for us (which is much more convenient than writing the same code over and over again).
We will name the function test_input().
Now, we can check each $_POST variable with the test_input() function, and the script looks like this:
Example

```
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}

function test_input($data) {
  $data = trim($data);
  $data = stripslashes($data);
  $data = htmlspecialchars($data);
  return $data;
}
?>
```

Output:

PHP Form Validation Example
Top of Form
Name:

E-mail:

Website:

Comment:

Gender: ○ Female ○ Male

Submit
Bottom of Form
Your Input:

Requiring numbers:

(PHP 4, PHP 5, PHP 7)
is_numeric — Finds whether a variable is a number or a numeric string
Description ¶
bool is_numeric (mixed $var)
Finds whether the given variable is numeric. Numeric strings consist of optional sign, any number of digits, optional decimal part and optional exponential part. Thus *+0123.45e6* is a valid numeric value. Hexadecimal (e.g.*0xf4c3b00c*) and binary (e.g. *0b10100111001*) notation is not allowed.
Parameters ¶
var
The variable being evaluated.
Return Values ¶
Returns **TRUE** if **var** is a number or a numeric string, **FALSE** otherwise.
Examples ¶
Example #1 is_numeric() examples

```
<?php
$tests = array(
    "42",
    1337,
    0x539,
    02471,
    0b10100111001,
    1337e0,
    "not numeric",
    array(),
    9.1,
    null
);

foreach ($tests as $element) {
    if (is_numeric($element)) {
        echo var_export($element, true) . " is numeric", PHP_EOL;
    } else {
        echo var_export($element, true) . " is NOT numeric", PHP_EOL;
    }
}
?>
```

The above example will output:

```
'42' is numeric
1337 is numeric
1337 is numeric
1337 is numeric
1337 is numeric
1337 is numeric
'not numeric' is NOT numeric
array () is NOT numeric
9.0999999999999996447286321199499070644378662109375 is numeric
NULL is NOT numeric
```

Requiring text:

Write applications that require the user to enter text ,or even specific text.

```
<html>
<body>

<div class="menu">
<?php include 'menu.php';?>
</div>

<h1>Welcome to my home page!</h1>
<p>Some text.</p>
<p>Some more text.</p>

</body>
</html>
```

Persisting user data:

In this context, data persistence is taken to mean any data that is intended to survive the current request. The memory management within the engine is very focused on request bound allocations, but this is not always practical or appropriate. Persistent memory is sometimes required in order to satisfy requirements of external libraries, it can also be useful while Hacking.

A common use of persistent memory is to enable persistent SQL server connections, though this practice is frowned upon, it is none the less the most common use of this feature.
Note: All of the following functions take the additional persistent parameter, should this be false, the engine will use its regular allocators (emalloc) and the memory should not be considered persistent. Where memory is allocated as persistent, system allocators are invoked, under most circumstances they are still not able to return NULL pointers just as the Main memory APIs.

Persistent memory APIs

Prototype	**Description**
void *pemalloc(size_t size, zend_bool persistent)	Allocate size bytes of memory.
void *pecalloc(size_t nmemb, size_t size, zend_bool persistent)	Allocate a buffer for nmemb elements of size bytes and makes sure it is initialized with zeros.
void *perealloc(void *ptr, size_t size, zend_bool persistent)	Resize the buffer ptr, which was allocated using emalloc to hold sizebytes of memory.
void pefree(void *ptr, zend_bool persistent)	Free the buffer pointed by ptr. The buffer had to be allocated by pemalloc.
void *safe_pemalloc(size_t nmemb, size_t size, size_t offset, zend_bool persistent)	Allocate a buffer for holding nmemb blocks of each size bytes and an additional offset bytes. This is similar to pemalloc(nmemb * size + offset) but adds a special protection against overflows.
char *pestrdup(const char *s, zend_bool persistent)	Allocate a buffer that can hold the NULL-terminated string s and copy the s into that buffer.
char *pestrndup(const char *s, unsigned int length, zend_bool persistent)	Similar to pestrdup while the length of the NULL-terminated string is already known.

Client-side data validation:

Is it better to validate form of our website both client and server side. Both have their own advantages.
In client side validation we use JavaScript so it's faster than the server side validation but sometimes when JavaScript disabled on browser then JavaScript doesn't work then server side validation will work.
Create below mentioned table in your database.
create table user (id int primary key auto_increment, name varchar(50), email varchar(50), gender varchar(1), phone varchar(10), address varchar(50))
Program 40: Working with form validation (client and server side).
File 1. connect.php :
```
<?php
$con = mysql_connect("localhost","root","");
mysql_select_db("mydb",$con);
?>
```
File 2. register.js : Used for client side validation.
```
function validate() {
var frm = document.getElementById("frmRegister");
var err = "";
var errDiv = document.getElementById("msg");
var name = frm.txtName.value;
if(name.trim() == "") {
```

err = “*name is required.<br/>”;
}
var email = frm.txtEmail.value;
var expEmail = /^(([^<>()[\]\\.,;:\s@\”]+(\.[^<>()[\]\\.,;:\s@\”]+)*)|(\”.+\”))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
if(email.trim()==””) {
err = err+”*email is required.<br/>”;
}
else if(!expEmail.test(email)) {
err = err+”*Invalid email<br/>”;
}
var password = frm.txtPassword.value;
if(password.trim()==””) {
err = err+”*Password required.<br/>”;
}
else if(password.length < 4) { err = err+"*Password should be minimun 4 characters.<br/>"; } var confPassword = frm.txtConfPassword.value; if(confPassword!=password) { err = err+"*Password and confirm password should be same.<br/>"; } var phone = frm.txtPhone.value; var expPhone = /^\d*$/; if(phone!="" && !expPhone.test(phone)) { err = err+"*Phone should be in integer.<br/>"; } if(err.trim()!="") { errDiv.innerHTML = err; return false; } else { return true; } }

handling HTML tags in user input:

PHP and HTML interact a lot: PHP can generate HTML, and HTML can pass information to PHP. Before reading these faqs, it's important you learn how to retrieve variables from external sources. The manual page on this topic includes many examples as well.
What encoding/decoding do I need when I pass a value through a form/URL?
I'm trying to use an <input type="image"> tag, but the $foo.x and $foo.y variables aren't available. $_GET['foo.x'] isn't existing either. Where are they?
How do I create arrays in a HTML <form>?
How do I get all the results from a select multiple HTML tag?
How can I pass a variable from Javascript to PHP?
What encoding/decoding do I need when I pass a value through a form/URL?
There are several stages for which encoding is important. Assuming that you have a string $data, which contains the string you want to pass on in a non-encoded way, these are the relevant stages:
HTML interpretation. In order to specify a random string, you *must* include it in double quotes, andhtmlspecialchars() the whole value.
URL: A URL consists of several parts. If you want your data to be interpreted as one item, you *must*encode it with urlencode().
Example #1 A hidden HTML form element
```
<?php
   echo '<input type="hidden" value="' . htmlspecialchars($data) . '" />'."\n";
?>
```
Note: It is wrong to urlencode() $data, because it's the browsers responsibility to urlencode() the data. All popular browsers do that correctly. Note that this will happen regardless of the method (i.e., GET or POST). You'll only notice this in case of GET request though, because POST requests are usually hidden.
Example #2 Data to be edited by the user

```
<?php
   echo "<textarea name='mydata'>\n";
   echo htmlspecialchars($data)."\n";
   echo "</textarea>";
?>
```

Unit- III

OBJECT-ORIENTED PROGRAMMING

Object Oriented Concepts:

Before we go in detail, lets define important terms related to Object Oriented Programming.

- **Class** − This is a programmer-defined data type, which includes local functions as well as local data. You can think of a class as a template for making many instances of the same kind (or class) of object.

- **Object** − An individual instance of the data structure defined by a class. You define a class once and then make many objects that belong to it. Objects are also known as instance.

- **Member Variable** − These are the variables defined inside a class. This data will be invisible to the outside of the class and can be accessed via member functions. These variables are called attribute of the object once an object is created.

- **Member function** − These are the function defined inside a class and are used to access object data.

- **Inheritance** − When a class is defined by inheriting existing function of a parent class then it is called inheritance. Here child class will inherit all or few member functions and variables of a parent class.

- **Parent class** − A class that is inherited from by another class. This is also called a base class or super class.

- **Child Class** − A class that inherits from another class. This is also called a subclass or derived class.

- **Polymorphism** − This is an object oriented concept where same function can be used for different purposes. For example function name will remain same but it make take different number of arguments and can do different task.

- **Overloading** − a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation.

- **Data Abstraction** − Any representation of data in which the implementation details are hidden (abstracted).

- **Encapsulation** − refers to a concept where we encapsulate all the data and member functions together to form an object.

- **Constructor** − refers to a special type of function which will be called automatically whenever there is an object formation from a class.

- **Destructor** − refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.

Defining PHP Classes
The general form for defining a new class in PHP is as follows –

```
<?php
  class phpClass {
    var $var1;
    var $var2 = "constant string";

    function myfunc ($arg1, $arg2) {
      [..]
    }
    [..]
  }
?>
```

Here is the description of each line −
The special form class, followed by the name of the class that you want to define.
A set of braces enclosing any number of variable declarations and function definitions.
Variable declarations start with the special form var, which is followed by a conventional $ variable name; they may also have an initial assignment to a constant value.
Function definitions look much like standalone PHP functions but are local to the class and will be used to set and access object data.

Example
Here is an example which defines a class of Books type −

```
<?php
  class Books {
    /* Member variables */
    var $price;
```

```
      var $title;

      /* Member functions */
      function setPrice($par){
         $this->price = $par;
      }

      function getPrice(){
         echo $this->price ."<br/>";
      }

      function setTitle($par){
         $this->title = $par;
      }

      function getTitle(){
         echo $this->title ." <br/>";
      }
   }
?>
```

The variable $this is a special variable and it refers to the same object ie. itself.

Creating Objects in PHP:

Once you defined your class, then you can create as many objects as you like of that class type. Following is an example of how to create object using new operator.
Eg:

```
$physics = new Books;
$maths = new Books;
$chemistry = new Books;
```

Here we have created three objects and these objects are independent of each other and they will have their existence separately. Next we will see how to access member function and process

member variables:

Calling Member Functions
After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.
Following example shows how to set title and prices for the three books by calling member functions.

```
$physics->setTitle( "Physics for High School" );
$chemistry->setTitle( "Advanced Chemistry" );
$maths->setTitle( "Algebra" );

$physics->setPrice( 10 );
$chemistry->setPrice( 15 );
$maths->setPrice( 7 );
```

Now you call another member functions to get the values set by in above example −

```
$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();
$physics->getPrice();
$chemistry->getPrice();
$maths->getPrice();
```

This will produce the following result −

```
Physics for High School
Advanced Chemistry
Algebra
10
15
7
```

Constructor Functions:

Constructor Functions are special type of functions which are called automatically whenever an object is created. So we take full advantage of this behaviour, by initializing many things through constructor functions.

PHP provides a special function called __construct() to define a constructor. You can pass as many as arguments you like into the constructor function

.

Following example will create one constructor for Books class and it will initialize price and title for the book at the time of object creation.

```
function __construct( $par1, $par2 ) {
   $this->title = $par1;
   $this->price = $par2;
}
```

Now we don't need to call set function separately to set price and title. We can initialize these two member variables at the time of object creation only. Check following example below −

```
$physics = new Books( "Physics for High School", 10 );
$maths = new Books ( "Advanced Chemistry", 15 );
$chemistry = new Books ("Algebra", 7 );

/* Get those set values */
$physics->getTitle();
$chemistry->getTitle();
$maths->getTitle();

$physics->getPrice();
$chemistry->getPrice();
$maths->getPrice();
```

This will produce the following result −

```
Physics for High School
Advanced Chemistry
Algebra
10
15
7
```

Destructor:

Like a constructor function you can define a destructor function using function __destruct(). You can release all the resources with-in a destructor.
Inheritance
PHP class definitions can optionally inherit from a parent class definition by using the extends clause. The syntax is as follows −

```
class Child extends Parent {
   <definition body>
}
```

The effect of inheritance is that the child class (or subclass or derived class) has the following characteristics −
Automatically has all the member variable declarations of the parent class.
Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.
Following example inherit Books class and adds more functionality based on the requirement.

```
class Novel extends Books {
   var $publisher;

   function setPublisher($par){
      $this->publisher = $par;
   }

   function getPublisher(){
      echo $this->publisher. "<br />";
   }
}
```

Now apart from inherited functions, class Novel keeps two additional member functions.

Function Overriding:

Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.
In the following example getPrice and getTitle functions are overridden to return some values.

```
function getPrice() {
   echo $this->price . "<br/>";
   return $this->price;
}

function getTitle(){
   echo $this->title . "<br/>";
   return $this->title;
}
```

Function overloading:

Overloading is defining functions that have similar signatures, yet have different parameters. **Overriding** is only pertinent to derived classes, where the parent class has defined a method and the derived class wishes to **override** that method.
In PHP, you can only overload methods using the magic method __call.
An example of **overriding**:

```
<?php

class Foo {
  function myFoo() {
    return "Foo";
  }
}

class Bar extends Foo {
  function myFoo() {
    return "Bar";
  }
}

$foo = new Foo;
$bar = new Bar;
echo($foo->myFoo()); //"Foo"
echo($bar->myFoo()); //"Bar"
?>
```

Autoloading classes:

Many developers writing object-oriented applications create one PHP source file per class definition. One of the biggest annoyances is having to write a long list of needed includes at the beginning of each script (one for each class)

this is no longer necessary. The spl_autoload_register() function registers any number of autoloaders, enabling for classes and interfaces to be automatically loaded if they are currently not defined. By registering autoloaders, PHP is given a last chance to load the class or interface before it fails with an error.

Although the __autoload() function can also be used for autoloading classes and interfaces, it's preferred to use the spl_autoload_register() function. This is because it is a more flexible alternative (enabling for any number of autoloaders to be specified in the application, such as in third party libraries). For this reason, using __autoload() is discouraged and it may be deprecated in the future.

exceptions thrown in the __autoload() function could not be caught in the catch block and would result in a fatal error. From PHP 5.3 and upwards, this is possible provided that if a custom exception is thrown, then the custom exception class is available. The __autoload() function may be used recursively to autoload the custom exception class

Autoloading is not available if using PHP in CLI interactive mode.

If the class name is used e.g. in call_user_func() then it can contain some dangerous characters such as ../. It is recommended to not use the user-input in such functions or at least verify the input in __autoload().
Example #1 Autoload example
This example attempts to load the classes *MyClass1* and *MyClass2* from the files *MyClass1.php* and *MyClass2.php*respectively.

```
<?php
spl_autoload_register(function ($class_name) {
    include $class_name . '.php';
});

$obj  = new MyClass1();
$obj2 = new MyClass2();
?>
```

Public Members:

Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations −
From outside the class in which it is declared
From within the class in which it is declared
From within another class that implements the class in which it is declared
Till now we have seen all members as public members. If you wish to limit the accessibility of the members of a class then you define class members as private or protected.

Private members:

By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.
A class member can be made private by using private keyword infront of the member.

```
class MyClass {
   private $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }

   function myPublicFunction() {
      return("I'm visible!");
   }
   private function myPrivateFunction() {
      return("I'm  not visible outside!");
   }
}
```

When *MyClass* class is inherited by another class using extends, myPublicFunction() will be visible, as will $driver. The extending class will not have any awareness of or access to myPrivateFunction and $car, because they are declared private.

Protected members:

A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes. A class member can be made protected by using protected keyword in front of the member.
Here is different version of MyClass −

```
class MyClass {
   protected $car = "skoda";
   $driver = "SRK";

   function __construct($par) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }

   function myPublicFunction() {
      return("I'm visible!");
   }

   protected function myPrivateFunction() {
      return("I'm  visible in child class!");
   }
}
```

ADVANCED OBJECT ORIENTED PROGRAMMING

Creating static methods :

When create static methods ,call the method without having to first create an object of that class.static methods are class methods .

Eg:

```
class test
{
private static $no_of_call = 0;
public function __construct()
{
self::$no_of_call = self::$no_of_call + 1;
echo "No of time object of the class created is: ". self::$no_of_call;
}
}
$objT = new test(); // Prints No of time object of the class created is 1
$objT2 = new test(); //Prints No of time object of the class created is 2
```

passing data to static method:

n PHP is it possible to do something like this:

```
myFunction( MyClass::staticMethod );
```

so that 'myFunction' will have a reference to the static method and be able to call it. When I try it, I get an error of "Undefined class constant" (PHP 5.3) so I guess it isn't directly possible, but is there a way to do something similar? The closest I've managed so far is pass the "function" as a string and use call_user_func().

using properties in static methods:

Sometimes, it is useful if we can access methods and properties in the context of a class rather than an object. To do this, you can use static keyword.

To add a static method to a class, you use the ***static* keyword** as follows:

```
public static function static_method(){
    // method implementation
}
```

You can put the static keyword before or after the method’s visibility. However, by convention, the visibility is declared first.

To add a static property to a class, you also use the static keyword as the following syntax:

```
private static $static_property;
```

The static methods and static properties are not linked to any particular object of the class but the class itself.

To call a static method outside the class, you use the :: operator as follows:

```
MyClass::static_method();
```

To access a public static property outside the class, you also use the :: operator:

```
MyClass::$static_property;
```

However to access static methods and static properties from within an instance of the class, you use self instead of $this as follows:

```
<?php
class MyClass{
        private static $static_property;

        public static function static_method(){
                //...
        }

        public function method(){
                self::$static_property;
                self::static_method();
        }
}
```

Static members and inheritance:

In PHP, if a static attribute is defined in the parent class, it cannot be overridden in a child class. But I'm wondering if there's any way around this.
I'm trying to write a wrapper for someone else's (somewhat clunky) function. The function in question can be applied to lots of different data types but requires different flags and options for each. But 99% of the time, a default for each type would suffice.
It would be nice if this could be done with inheritance, without having to write new functions each time. For example:

```
class Foo {
   public static $default = 'DEFAULT';

   public static function doSomething ($param = FALSE ) {
       $param = ($param === FALSE) ? self::$default : $param;
       return $param;
   }
}

class Bar extends Foo {
   public static $default = 'NEW DEFAULT FOR CHILD CLASS';
}

echo Foo::doSomething() . "\n";
// echoes 'DEFAULT'

echo Bar::doSomething() . "\n";
```

supporting object iteration:

- **public function current()**
- **public function key()**
- **public function next()**
- **public function valid()**
- **public function rewind()**

the foreach iterated through all of the visible properties that could be accessed.
To take it a step further, the Iterator interface may be implemented. This allows the object to dictate how it will be iterated and what values will be available on each iteration.
Example #2 Object Iteration implementing Iterator

```
<?php
class MyIterator implements Iterator
{
    private $var = array();

    public function __construct($array)
    {
        if (is_array($array)) {
            $this->var = $array;
        }
    }

    public function rewind()
    {
        echo "rewinding\n";
        reset($this->var);
    }

    public function current()
    {
        $var = current($this->var);
        echo "current: $var\n";
        return $var;
    }

    public function key()
    {
        $var = key($this->var);
        echo "key: $var\n";
        return $var;
    }

    public function next()
    {
        $var = next($this->var);
        echo "next: $var\n";
        return $var;
    }

    public function valid()
    {
        $key = key($this->var);
        $var = ($key !== NULL && $key !== FALSE);
        echo "valid: $var\n";
        return $var;
    }

}
```

```
$values = array(1,2,3);
$it = new MyIterator($values);

foreach ($it as $a => $b) {
    print "$a: $b\n";
}
?>
```

Comparing objects:

When using the comparison operator (==), object variables are compared in a simple manner, namely: Two object instances are equal if they have the same attributes and values (values are compared with ==), and are instances of the same class.
When using the identity operator (===), object variables are identical if and only if they refer to the same instance of the same class.
An example will clarify these rules.
Example #1 Example of object comparison in PHP 5

```
<?php
function bool2str($bool)
{
   if ($bool === false) {
      return 'FALSE';
   } else {
      return 'TRUE';
   }
}

function compareObjects(&$o1, &$o2)
{
   echo 'o1 == o2 : ' . bool2str($o1 == $o2) . "\n";
   echo 'o1 != o2 : ' . bool2str($o1 != $o2) . "\n";
   echo 'o1 === o2 : ' . bool2str($o1 === $o2) . "\n";
   echo 'o1 !== o2 : ' . bool2str($o1 !== $o2) . "\n";
}

class Flag
{
   public $flag;

   function __construct($flag = true) {
      $this->flag = $flag;
   }
}

class OtherFlag
{
   public $flag;
```

```
    function __construct($flag = true) {
        $this->flag = $flag;
    }
}

$o = new Flag();
$p = new Flag();
$q = $o;
$r = new OtherFlag();

echo "Two instances of the same class\n";
compareObjects($o, $p);

echo "\nTwo references to the same instance\n";
compareObjects($o, $q);

echo "\nInstances of two different classes\n";
compareObjects($o, $r);
?>
```

Interfaces:

Interfaces are defined to provide a common function names to the implementers. Different implementors can implement those interfaces according to their requirements. You can say, interfaces are skeletons which are implemented by developers.

As of PHP5, it is possible to define an interface, like this −

```
interface Mail {
   public function sendMail();
}
```

Then, if another class implemented that interface, like this −

```
class Report implements Mail {
   // sendMail() Definition goes here
}
```

Constants:

A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable. Once you declare a constant, it does not change.

Declaring one constant is easy, as is done in this version of MyClass −

```
class MyClass {
   const requiredMargin = 1.7;

   function __construct($incomingValue) {
      // Statements here run every time
      // an instance of the class
      // is created.
   }
}
```

In this class, requiredMargin is a constant. It is declared with the keyword const, and under no circumstances can it be changed to anything other than 1.7. Note that the constant's name does not have a leading $, as variable names do.

Abstract Classes:

An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword abstract, like this −
When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.

```
abstract class MyAbstractClass {
  abstract function myAbstractFunction() {
  }
}
```

Note that function definitions inside an abstract class must also be preceded by the keyword abstract. It is not legal to have abstract function definitions inside a non-abstract class.

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 static method can).
Try out following example −

```
<?php
  class Foo {
    public static $my_static = 'foo';

    public function staticValue() {
      return self::$my_static;
    }
  }

  print Foo::$my_static . "\n";
  $foo = new Foo();

  print $foo->staticValue() . "\n";
?>
```

Final Keyword:

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.
Following example results in Fatal error: Cannot override final method BaseClass::moreTesting()

```
<?php

  class BaseClass {
      public function test() {
         echo "BaseClass::test() called<br>";
      }

      final public function moreTesting() {
         echo "BaseClass::moreTesting() called<br>";
      }
   }

   class ChildClass extends BaseClass {
      public function moreTesting() {
         echo "ChildClass::moreTesting() called<br>";
      }
   }
?>
```

Calling parent constructors
Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass. Here's a simple example −

```
class Name {
   var $_firstName;
   var $_lastName;

   function Name($first_name, $last_name) {
      $this->_firstName = $first_name;
      $this->_lastName = $last_name;
   }

   function toString() {
      return($this->_lastName .", " .$this->_firstName);
   }
}
class NameSub1 extends Name {
   var $_middleInitial;

   function NameSub1($first_name, $middle_initial, $last_name) {
      Name::Name($first_name, $last_name);
      $this->_middleInitial = $middle_initial;
   }

   function toString() {
      return(Name::toString() . " " . $this->_middleInitial);
   }
}
```

In this example, we have a parent class (Name), which has a two-argument constructor, and a subclass (NameSub1), which has a three-argument constructor. The constructor of NameSub1 functions by calling its parent constructor explicitly using the :: syntax (passing two of its arguments along) and then setting an additional field. Similarly, NameSub1 defines its non constructor toString() function in terms of the parent function that it overrides.

Cloning object:

creating a copy of an object with fully replicated properties is not always the wanted behavior. A good example of the need for copy constructors, is if you have an object which represents a GTK window and the object holds the resource of this GTK window, when you create a duplicate you might want to create a new window with the same properties and have the new object hold the resource of the new window. Another example is if your object holds a reference to another object which it uses and when you replicate the parent object you want to create a new instance of this other object so that the replica has its own separate copy.
An object copy is created by using the *clone* keyword (which calls the object's __clone() method if possible). An object's __clone() method cannot be called directly.
$copy_of_object = clone $object;
When an object is cloned, PHP 5 will perform a shallow copy of all of the object's properties. Any properties that are references to other variables will remain references.
void __clone (void)
Once the cloning is complete, if a __clone() method is defined, then the newly created object's __clone() method will be called, to allow any necessary properties that need to be changed.
Example #1 Cloning an object

```
<?php
class SubObject
{
    static $instances = 0;
    public $instance;

    public function __construct() {
        $this->instance = ++self::$instances;
    }

    public function __clone() {
        $this->instance = ++self::$instances;
    }
}

class MyCloneable
{
    public $object1;
    public $object2;

    function __clone()
    {
        // Force a copy of this->object, otherwise
        // it will point to same object.
        $this->object1 = clone $this->object1;
    }
}
```

```
$obj = new MyCloneable();

$obj->object1 = new SubObject();
$obj->object2 = new SubObject();

$obj2 = clone $obj;


print("Original Object:\n");
print_r($obj);

print("Cloned Object:\n");
print_r($obj2);

?>
```

Reflection:

he ReflectionClass class reports information about a class.

Reflector {
/* Methods */
public static string export (void)
public string __toString (void)
}

Unit- IV

FILE HANDLING

PHP has several functions for creating, reading, uploading, and editing files.

The file may be opened in one of the following modes:

r	**Open a file for read only**. File pointer starts at the beginning of the file
w	**Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a	**Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x	**Creates a new file for write only**. Returns FALSE and an error if file already exists
r+	**Open a file for read/write**. File pointer starts at the beginning of the file

w+	**Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file
a+	**Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist
x+	**Creates a new file for read/write**. Returns FALSE and an error if file already exists

Opening files using fopen:

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.
We will use the text file, "webdictionary.txt", during the lessons:
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
echo fread($myfile,filesize("webdictionary.txt"));
fclose($myfile);
?>
```

Output:

```
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup
Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector
Graphics XML = EXtensible Markup Language
```

PHP Read File - fread():

The fread() function reads from an open file.
The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.
The following PHP code reads the "webdictionary.txt" file to the end:

```
fread($myfile,filesize("webdictionary.txt"));
```

PHP Check End-Of-File - feof():

The feof() function checks if the "end-of-file" (EOF) has been reached.
The feof() function is useful for looping through data of unknown length.
The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>
```

Output:

```
AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language
```

PHP Read Single Character - fgetc():

The fgetc() function is used to read a single character from a file.
The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

Example

```
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one character until end-of-file
while(!feof($myfile)) {
  echo fgetc($myfile);
}
fclose($myfile);
?>
```

Output:

```
AJAX = Asynchronous JavaScript and XML CSS = Cascading Style Sheets HTML = Hyper Text Markup
Language PHP = PHP Hypertext Preprocessor SQL = Structured Query Language SVG = Scalable Vector
Graphics XML = EXtensible Markup Language
```

PHP Close File - fclose():

The fclose() function is used to close an open file.
It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources!
The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

```
<?php
$myfile = fopen("webdictionary.txt", "r");
```

```
// some code to be executed....
fclose($myfile);
?>
```

PHP Write to File - fwrite():

The fwrite() function is used to write to a file.
The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.
The example below writes a couple of names into a new file called "newfile.txt":
Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "John Doe\n";
fwrite($myfile, $txt);
$txt = "Jane Doe\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

PHP Overwriting:

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

Example

```
<?php
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");
$txt = "Mickey Mouse\n";
fwrite($myfile, $txt);
$txt = "Minnie Mouse\n";
fwrite($myfile, $txt);
fclose($myfile);
?>
```

Configure The "php.ini" File:

First, ensure that PHP is configured to allow file uploads.
In your "php.ini" file, search for the file_uploads directive, and set it to On:

```
file_uploads = On
```

Check if File Already Exists:

Now we can add some restrictions.
First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

```
// Check if file already exists
if (file_exists($target_file)) {
    echo "Sorry, file already exists.";
    $uploadOk = 0;
}
```

WORKING WITH DATABASES

Databases are useful for storing information categorically. A company may have a database with the following tables:

- Employees
- Products
- Customers
- Orders

What is a batabase?

- MySQL is a database system used on the web
- MySQL is a database system that runs on a server
- MySQL is ideal for both small and large applications
- MySQL is very fast, reliable, and easy to use
- MySQL uses standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use
- MySQL is developed, distributed, and supported by Oracle Corporation
- MySQL is named after co-founder Monty Widenius's daughter: My

The data in a MySQL database are stored in tables. A table is a collection of related data, and it consists of columns and rows.

Some essential SQL:

A query is a question or a request.

We can query a database for specific information and have a recordset returned.

Look at the following query (using standard SQL):

```
SELECT LastName FROM Employees
```

The query above selects all the data in the "LastName" column from the "Employees" table.

To learn more about SQL, please visit our SQL tutorial.

Creating amysql database:

A database consists of one or more tables.
You will need special CREATE privileges to create or to delete a MySQL database.

The CREATE DATABASE statement is used to create a database in MySQL.
The following examples create a database named "myDB":

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);
// Check connection
if ($conn->connect_error) {
   die("Connection failed: " . $conn->connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
   echo "Database created successfully";
} else {
   echo "Error creating database: " . $conn->error;
}

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

Creating a new table:

The CREATE TABLE statement is used to create a table in MySQL.
We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":

```
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```

The data type specifies what type of data the column can hold. For a complete reference of all the available data types, go to our Data Types reference.

- After the data type, you can specify other optional attributes for each column:
- NOT NULL - Each row must contain a value for that column, null values are not allowed
- DEFAULT value - Set a default value that is added when no other value is passed

- UNSIGNED - Used for number types, limits the stored data to positive numbers and zero
- AUTO INCREMENT - MySQL automatically increases the value of the field by 1 each time a new record is added
- PRIMARY KEY - Used to uniquely identify the rows in a table. The column with PRIMARY KEY setting is often an ID number, and is often used with AUTO_INCREMENT

Each table should have a primary key column (in this case: the "id" column). Its value must be unique for each record in the table.

Putting data into the new database:

After a database and a table have been created, we can start adding data in them.
Here are some syntax rules to follow:
The SQL query must be quoted in PHP
String values inside the SQL query must be quoted
Numeric values must not be quoted
The word NULL must not be quoted
The INSERT INTO statement is used to add new records to a MySQL table:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

To learn more about SQL, please visit our SQL tutorial.
In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.

Accessing the database in php:

he **sqlsrv_connect** function is used to establish a connection to the server. The code shown here (from the Example Application in the product documentation) establishes a connection to the local server and specifies the **AdventureWorks** database as the database in use:

```
        $serverName = "(local)";
$connectionOptions = array("Database"=>"AdventureWorks");

/* Connect using Windows Authentication. */
$conn = sqlsrv_connect( $serverName, $connectionOptions);
if( $conn === false )
    { die( FormatErrors( sqlsrv_errors() ) ); }
```

By default, the **sqlsrv_connect** function uses Windows Authentication to establish a connection. In most scenarios, this means that the Web server's process identity or thread identity (if the Web server is using impersonation) is used to connect to the server, not an end-user's identity.
The **sqlsrv_connect** function accepts two parameters: *$serverName* and *$connectionOptions* (optional).
· *$serverName* – This required parameter is used to specify the name of the server to which you want to connect. In the code above, a connection is established to the local server. This parameter can also be use to specify a SQL Server instance or a port number. For example:
$serverName = "myServer\instanceName";

Connecting to the database server:

Before you can get content out of your MySQL database, you must know how to establish a connection to MySQL from inside a PHP script. To perform basic queries from within MySQL is very easy. This article will show you how to get up and running.
Let's get started. The first thing to do is connect to the database.The function to connect to MySQL is called mysql_connect. This function returns a resource which is a pointer to the database connection. It's also called a database handle, and we'll use it in later functions. Don't forget to replace your connection details.

```
<?php
$username = "your_name";
$password = "your_password";
$hostname = "localhost";

//connection to the database
$dbhandle = mysql_connect($hostname, $username, $password)
  or die("Unable to connect to MySQL");
echo "Connected to MySQL<br>";
?>
```

All going well, you should see "Connected to MySQL" when you run this script. If you can't connect to the server, make sure your password, username and hostname are correct.
Once you've connected, you're going to want to select a database to work with. Let's assume the database is called 'examples'. To start working in this database, you'll need the mysql_select_db() function:

```
<?php
//select a database to work with
$selected = mysql_select_db("examples",$dbhandle)
  or die("Could not select examples");
?>
```

Now that you're connected, let's try and run some queries. The function used to perform queries is named - mysql_query(). The function returns a resource that contains the results of the query, called the result set. To examine the result we're going to use the mysql_fetch_array function, which returns the results row by row. In the case of a query that doesn't return results, the resource that the function returns is simply a value true or false.
A convenient way to access all the rows is with a while loop. Let's add the code to our script:

```
<?php
//execute the SQL query and return records
$result = mysql_query("SELECT id, model, year FROM cars");
//fetch tha data from the database
while ($row = mysql_fetch_array($result)) {
   echo "ID:".$row{'id'}." Name:".$row{'model'}."
   ".$row{'year'}."<br>";
}
?>
```

Finally, we close the connection. Although this isn't strictly speaking necessary, PHP will automatically close the connection when the script ends, you should get into the habit of closing what you open.

```
<?php
```

```
//close the connection
mysql_close($dbhandle);
?>
```

Connecting to the database:

Open a new connection to the MySQL server:

```
<?php
$con = mysqli_connect("localhost","my_user","my_password","my_db");

// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }
?>
```

Reading the table:

To read records from a database, the technique is usually to loop round and find the ones you want. To specify which records you want, you use something called SQL. This stands for Structured Query Language. This is a natural, non-coding language that uses words like SELECT and WHERE. At it's simplest level, it's fairly straightforward. But the more complex the database, the more trickier the SQL is. We'll start with something simple though.

What we want to do, now that we have a connection to our database, is to read all the records from our address book, and print them out to the page. Here's some new code, added to the PHP script you already have. The new lines are in blue:

```
<?PHP
require '../configure.php'
$db_handle = mysqli_connect(DB_SERVER, DB_USER, DB_PASS );
$database = "addressbook";
$db_found = mysqli_select_db($db_handle, $database);
if ($db_found) {
$SQL = "SELECT * FROM tbl_address_book";
$result = mysqli_query($db_handle, $SQL);
while ( $db_field = mysqli_fetch_assoc($result) ) {
print $db_field['ID'] . "<BR>";
print $db_field['First_Name'] . "<BR>";
print $db_field['Surname'] . "<BR>";
print $db_field['Address'] . "<BR>";
}
}
else {
print "Database NOT Found ";
}
mysqli_close($db_handle);
?>
```

Before we go through the new code to see what's happening, run your script. You should find that the address you added in a previous section is printed out. (We only have one record at the moment.)
1
Test
Name
12 Test Street
The first line in the new code is this:
$SQL = "SELECT * FROM tbl_address_book";
The $SQL is just a normal variable. But we're putting into it a long string. This is a SQL statement. Here's a brief run down on SQL.

Inserting new data items into a database:

After a database and a table have been created, we can start adding data in them.
Here are some syntax rules to follow:
The SQL query must be quoted in PHP
String values inside the SQL query must be quoted
Numeric values must not be quoted
The word NULL must not be quoted
The INSERT INTO statement is used to add new records to a MySQL table:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

To learn more about SQL, please visit our SQL tutorial.
In the previous chapter we created an empty table named "MyGuests" with five columns: "id", "firstname", "lastname", "email" and "reg_date". Now, let us fill the table with data.
If a column is AUTO_INCREMENT (like the "id" column) or TIMESTAMP (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.
The following examples add a new record to the "MyGuests" table:
Eg:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
```

```
if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

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

Deleting records:

The DELETE statement is used to delete records from a table:
DELETE FROM table_name
WHERE some_column = some_value
Notice the WHERE clause in the DELETE syntax: The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!
To learn more about SQL, please visit our SQL tutorial.
Let's look at the "MyGuests" table:

id	firstname	lastname	Email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30
3	Julie	Dooley	julie@example.com	2014-10-26 10:48:23

Creating a new table:

The CREATE TABLE statement is used to create a table in MySQL.
We will create a table named "MyGuests", with five columns: "id", "firstname", "lastname", "email" and "reg_date":
CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)

Updating databases:

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Sorting data:

MySQL provides a LIMIT clause that is used to specify the number of records to return.
The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance.
Assume we wish to select all records from 1 - 30 (inclusive) from a table called "Orders". The SQL query would then look like this:

$sql = "SELECT * FROM Orders LIMIT 30";

When the SQL query above is run, it will return the first 30 records.
What if we want to select records 16 - 25 (inclusive)?
Mysql also provides a way to handle this: by using OFFSET.

The SQL query below says "return only 10 records, start on record 16 (OFFSET 15)":
$sql = "SELECT * FROM Orders LIMIT 10 OFFSET 15";
You could also use a shorter syntax to achieve the same result:
$sql = "SELECT * FROM Orders LIMIT 15, 10";

id	firstname	lastname	Email	reg_date
1	John	Doe	john@example.com	2014-10-22 14:26:15
2	Mary	Moe	mary@example.com	2014-10-23 10:22:30

The following examples update the record with id=2 in the "MyGuests" table:
Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if ($conn->query($sql) === TRUE) {
    echo "Record updated successfully";
} else {
    echo "Error updating record: " . $conn->error;
}

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

COOKIES

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

Setting a cookie:

A cookie is created with the setcookie() function.

Syntax

setcookie(*name, value, expire, path, domain, secure, httponly*);
Only the *name* parameter is required. All other parameters are optional.

Reading acookie:

creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).
We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the isset() function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 = 1 day
?>
<html>
<body>

<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
    echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
```

```
</body>
</html>
```

Output:

```
Cookie named 'user' is not set!
```

Setting cookies expiration:

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $_COOKIE array variable:

Example

```
<?php
setcookie("test_cookie", "test", time() + 3600, '/');
?>
<html>
<body>

<?php
if(count($_COOKIE) > 0) {
    echo "Cookies are enabled.";
} else {
    echo "Cookies are disabled.";
}
?>

</body>
</html>
```

Output:

```
Cookies are enabled.
```

Delete cookies:

To delete a cookie, use the setcookie() function with an expiration date in the past

Example

```
<?php
// set the expiration date to one hour ago
setcookie("user", "", time() - 3600);
?>
<html>
<body>

<?php
echo "Cookie 'user' is deleted.";
?>

</body>
</html>
```

Output:

```
Cookie 'user' is deleted.
```

FTP(file transfor protocol)

The FTP functions give client access to file servers through the File Transfer Protocol (FTP).
The FTP functions are used to open, login and close connections, as well as upload, download, rename, delete, and get information on files from file servers. Not all of the FTP functions will work with every server or return the same results. The FTP functions became available with PHP

If you only wish to read from or write to a file on an FTP server, consider using the ftp:// wrapper with the Filesystem functions which provide a simpler and more intuitive interface.

Working with FTP:

Function	Description
ftp_alloc()	Allocates space for a file to be uploaded to the FTP server
ftp_cdup()	Changes to the parent directory on the FTP server
ftp_chdir()	Changes the current directory on the FTP server
ftp_chmod()	Sets permissions on a file via FTP
ftp_close()	Closes an FTP connection
ftp_connect()	Opens an FTP connection
ftp_delete()	Deletes a file on the FTP server
ftp_exec()	Executes a command on the FTP server
ftp_fget()	Downloads a file from the FTP server and saves it into an open local file
ftp_fput()	Uploads from an open file and saves it to a file on the FTP server
ftp_get_option()	Returns runtime options of the FTP connection
ftp_get()	Downloads a file from the FTP server
ftp_login()	Logs in to the FTP connection
ftp_mdtm()	Returns the last modified time of a specified file
ftp_mkdir()	Creates a new directory on the FTP server

ftp_nb_continue()	Continues retrieving/sending a file (non-blocking)
ftp_nb_fget()	Downloads a file from the FTP server and saves it into an open file (non-blocking)
ftp_nb_fput()	Uploads from an open file and saves it to a file on the FTP server (non-blocking)
ftp_nb_get()	Downloads a file from the FTP server (non-blocking)
ftp_nb_put()	Uploads a file to the FTP server (non-blocking)
ftp_nlist()	Returns a list of files in the specified directory on the FTP server
ftp_pasv()	Turns passive mode on or off
ftp_put()	Uploads a file to the FTP server
ftp_pwd()	Returns the current directory name
ftp_quit()	An alias of ftp_close()
ftp_raw()	Sends a raw command to the FTP server
ftp_rawlist()	Returns a list of files with file information from a specified directory
ftp_rename()	Renames a file or directory on the FTP server
ftp_rmdir()	Deletes an empty directory on the FTP server
ftp_set_option()	Sets runtime options for the FTP connection
ftp_site()	Sends an FTP SITE command to the FTP server
ftp_size()	Returns the size of the specified file
ftp_ssl_connect()	Opens a secure SSL-FTP connection
ftp_systype()	Returns the system type identifier of the FTP server

Downloading files with FTP:

ftp_fget — Downloads a file from the FTP server and saves to an open file
Description ¶
bool ftp_fget (resource $ftp_stream , resource $handle , string $remote_file , int $mode [, int$resumepos = 0])
ftp_fget() retrieves **remote_file** from the FTP server, and writes it to the given file pointer.

Parameters ¶
ftp_stream
The link identifier of the FTP connection.
handle
An open file pointer in which we store the data.
remote_file
The remote file path.
mode
The transfer mode. Must be either **FTP_ASCII** or **FTP_BINARY**.
resumepos
The position in the remote file to start downloading from.
Return Values ¶
Returns **TRUE** on success or **FALSE** on failure.

Uploading file with FTP:
The ftp_put() function uploads a file to the FTP server.

Syntax
ftp_put(*ftp_connection,remote_file,local_file,mode,startpos*);

Example
Upload local file to a file on the FTP server:
```
<?php
// connect and login to FTP server
$ftp_server = "ftp.example.com";
$ftp_conn = ftp_connect($ftp_server) or die("Could not connect to $ftp_server");
$login = ftp_login($ftp_conn, $ftp_username, $ftp_userpass);

$file = "localfile.txt";

// upload file
if (ftp_put($ftp_conn, "serverfile.txt", $file, FTP_ASCII))
  {
  echo "Successfully uploaded $file.";
  }
else
  {
  echo "Error uploading $file.";
  }

// close connection
ftp_close($ftp_conn);
?>
```

Deleting a file with FTP:
The ftp_delete() function deletes a file on the FTP server.

Syntax
ftp_delete(ftp_connection,file);

creating and removing directories with ftp:

The ftp_rmdir() function deletes a directory on the FTP server.

Syntax
ftp_rmdir(ftp_connection,dir);
eg:

Delete a directory on FTP server:

```
<?php
// connect and login to FTP server
$ftp_server = "ftp.example.com";
$ftp_conn = ftp_connect($ftp_server) or die("Could not connect to $ftp_server");
$login = ftp_login($ftp_conn, $ftp_username, $ftp_userpass);

$dir = "php/";

// try to delete $dir
if (ftp_rmdir($ftp_conn, $dir))
  {
  echo "Directory $dir deleted";
  }
else
  {
  echo "Problem deleting $dir";
  }

// close connection
ftp_close($ftp_conn);
?>
```

Sending E-mail:

The mail() function allows you to send emails directly from a script.
The mail() function allows you to send emails directly from a script.
Syntax
mail(*to,subject,message,headers,parameters*);

Parameter	Description
to	Required. Specifies the receiver / receivers of the email
subject	Required. Specifies the subject of the email. **Note:** This parameter cannot contain any newline characters

message	Required. Defines the message to be sent. Each line should be separated with a LF (\n). Lines should not exceed 70 characters.
	Windows note: If a full stop is found on the beginning of a line in the message, it might be removed. To solve this problem, replace the full stop with a double dot:
	`<?php`
	`$txt = str_replace("\n.", "\n..", $txt);`
	`?>`

headers	Optional. Specifies additional headers, like From, Cc, and Bcc. The additional headers should be separated with a CRLF (\r\n). **Note:** When sending an email, it must contain a From header. This can be set with this parameter or in the php.ini file.
parameters	Optional. Specifies an additional parameter to the sendmail program (the one defined in the sendmail_path configuration setting). (i.e. this can be used to set the envelope sender address when using sendmail with the -f sendmail option)

```
<?php
$to = "somebody@example.com, somebodyelse@example.com";
$subject = "HTML email";

$message = "
<html>
<head>
<title>HTML email</title>
</head>
<body>
<p>This email contains HTML Tags!</p>
<table>
<tr>
<th>Firstname</th>
<th>Lastname</th>
</tr>
<tr>
<td>John</td>
<td>Doe</td>
</tr>
</table>
</body>
</html>
";

// Always set content-type when sending HTML email
$headers = "MIME-Version: 1.0" . "\r\n";
$headers .= "Content-type:text/html;charset=UTF-8" . "\r\n";

// More headers
$headers .= 'From: <webmaster@example.com>' . "\r\n";
$headers .= 'Cc: myboss@example.com' . "\r\n";
```

```
mail($to,$subject,$message,$headers);
?>
```

Sending advanced E-mail:

The simplest way to send an email with PHP is to send a text email. In the example below we first declare the variables — recipient's email address, subject line and message body — then we pass these variables to the mail() function to send the email.

Example

Download

```
<?php
$to = 'maryjane@email.com';
$subject = 'Marriage Proposal';
$message = 'Hi Jane, will you marry me?';
$from = 'peterparker@email.com';

// Sending email
if(mail($to, $subject, $message)){
    echo 'Your mail has been sent successfully.';
} else{
    echo 'Unable to send email. Please try again.';
}
?>
```

Adding attachments to email:

Sending Email with Attachment

The last variation that we will consider is email with attachments. To send an email with attachment we need to use the multipart/mixed MIME type that specifies that mixed types will be included in the email. Moreover, we want to use multipart/alternative MIME type to send both plain-text and HTML version of the email. Have a look at the example:

```
<?php
//define the receiver of the email
$to = 'youraddress@example.com';
//define the subject of the email
$subject = 'Test email with attachment';
//create a boundary string. It must be unique
//so we use the MD5 algorithm to generate a random hash
$random_hash = md5(date('r', time()));
//define the headers we want passed. Note that they are separated with \r\n
$headers = "From: webmaster@example.com\r\nReply-To: webmaster@example.com";
//add boundary string and mime type specification
$headers .= "\r\nContent-Type: multipart/mixed; boundary=\"PHP-mixed-".$random_hash."\"";
//read the atachment file contents into a string,
//encode it with MIME base64,
//and split it into smaller chunks
$attachment = chunk_split(base64_encode(file_get_contents('attachment.zip')));
//define the body of the message.
ob_start(); //Turn on output buffering
?>
--PHP-mixed-<?php echo $random_hash; ?>
```

```
Content-Type: multipart/alternative; boundary="PHP-alt-<?php echo $random_hash; ?>"

--PHP-alt-<?php echo $random_hash; ?>
Content-Type: text/plain; charset="iso-8859-1"
Content-Transfer-Encoding: 7bit

Hello World!!!
This is simple text email message.

--PHP-alt-<?php echo $random_hash; ?>
Content-Type: text/html; charset="iso-8859-1"
Content-Transfer-Encoding: 7bit

<h2>Hello World!</h2>
<p>This is something with <b>HTML</b> formatting.</p>

--PHP-alt-<?php echo $random_hash; ?>--

--PHP-mixed-<?php echo $random_hash; ?>
Content-Type: application/zip; name="attachment.zip"
Content-Transfer-Encoding: base64
Content-Disposition: attachment

<?php echo $attachment; ?>
--PHP-mixed-<?php echo $random_hash; ?>--

<?php
//copy current buffer contents into $message variable and delete current output buffer
$message = ob_get_clean();
//send the email
$mail_sent = @mail( $to, $subject, $message, $headers );
//if the message is sent successfully print "Mail sent". Otherwise print "Mail failed"
echo $mail_sent ? "Mail sent" : "Mail failed";
?>
```

As you can see, sending an email with attachment is easy to accomplish. In the preceding example we have multipart/mixed MIME type, and inside it we have multipart/alternative MIME type that specifies two versions of the email. To include an attachment to our message, we read the data from the specified file into a string, encode it with base64, split it in smaller chunks to make sure that it matches the MIME specifications and then include it as an attachment.

SESSIONS

A session is a way to store information (in variables) to be used across multiple pages.
Unlike a cookie, the information is not stored on the users computer.

What is a PHP Session?

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you

end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.
Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.
So; Session variables hold information about one single user, and are available to all pages in one application.
Tip: If you need a permanent storage, you may want to store the data in a database.

Start a PHP Session(or) Sorting data in sessions:

A session is started with the session_start() function.
Session variables are set with the PHP global variable: $_SESSION.
Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:
Example

```
<?php
// Start the session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>

</body>
</html>
```

Output:

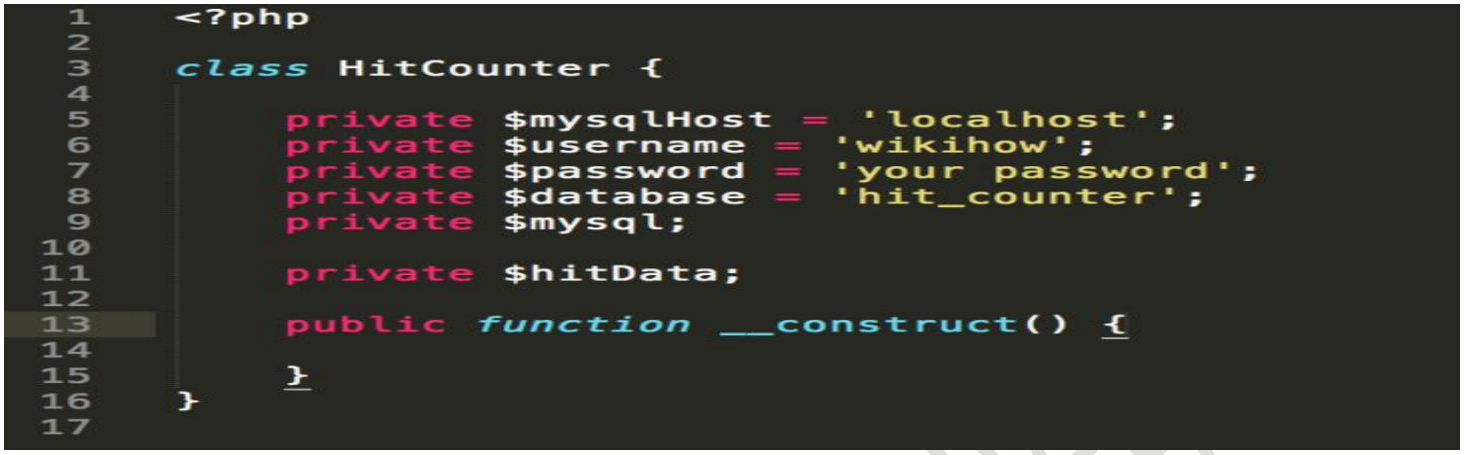
```
Session variables are set.
```

Writing a hit counter using sessions:

Creating the Hit Counter

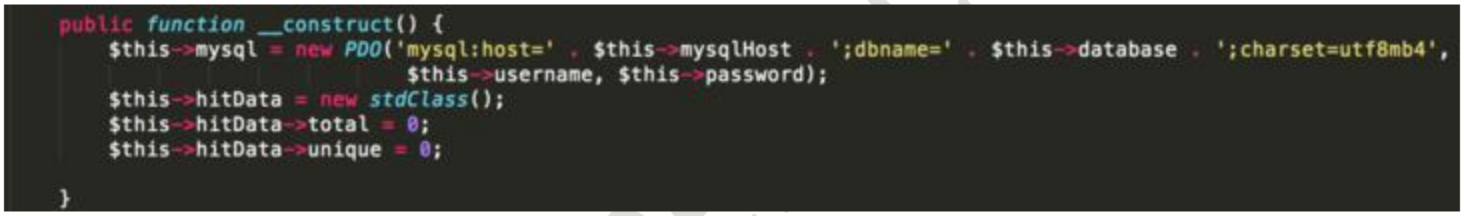
- getData() retrieves the current view counts from the database. If there isn't any data, it sets this data to zero.
- isNewVisitor() determines whether the visitor has already visited our website in their current session
- visit() increments the total hit counter, and increments the unique visitor counter if the user has not visited the website in their current session.

Create a new file called "HitCounter.php". This file will contain a class you can include in other PHP scripts when you want it to log a hit.



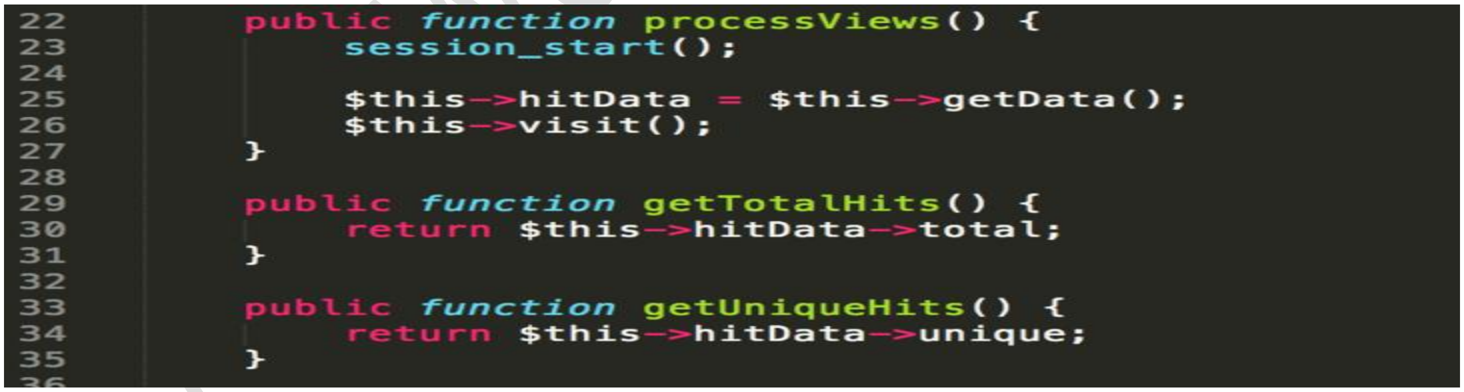
```php
<?php

class HitCounter {

    private $mysqlHost = 'localhost';
    private $username = 'wikihow';
    private $password = 'your password';
    private $database = 'hit_counter';
    private $mysql;

    private $hitData;

    public function __construct() {

    }
}

```

2
Stub out the HitCounter class. Create local class members to hold your credentials and the database connection info.



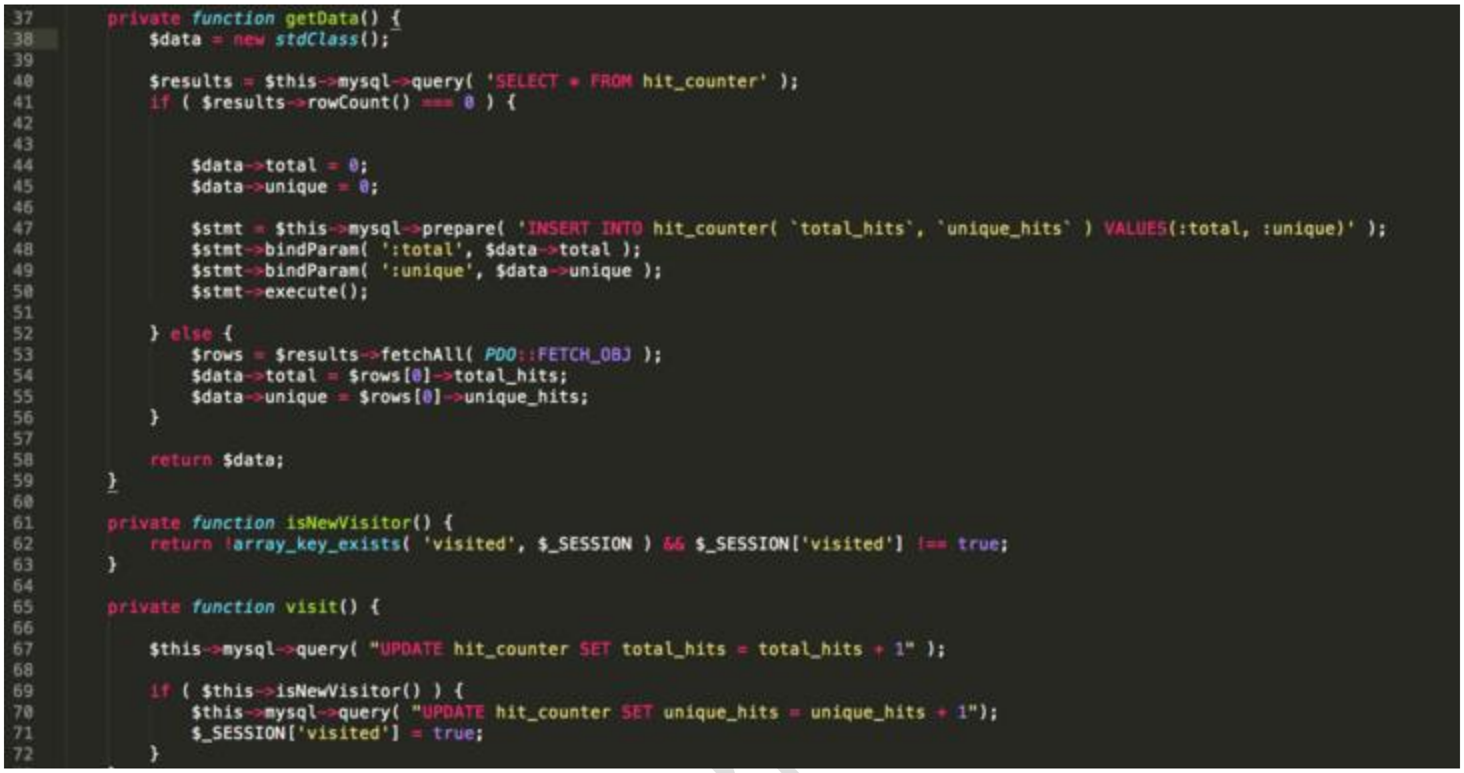
```php
public function __construct() {
    $this->mysql = new PDO('mysql:host=' . $this->mysqlHost . ';dbname=' . $this->database . ';charset=utf8mb4',
                        $this->username, $this->password);
    $this->hitData = new stdClass();
    $this->hitData->total = 0;
    $this->hitData->unique = 0;

}
```

3
Write the constructor. In the constructor, you should establish the database connection and initialize the hit counts at zero.



```php
    public function processViews() {
        session_start();

        $this->hitData = $this->getData();
        $this->visit();
    }

    public function getTotalHits() {
        return $this->hitData->total;
    }

    public function getUniqueHits() {
        return $this->hitData->unique;
    }
```

4
Write the public functions. These methods can be called from any other script that instantiates the HitCounter class.
Add a method for processing views. This method gets called on every page load that should be counted towards a hit.
Add a getter for the total views. This will get called in places where you want to show the total view count.
Add a getter for the unique hits. You'll call this where you want to show the unique view count.



```php
    private function getData() {
        $data = new stdClass();

        $results = $this->mysql->query( 'SELECT * FROM hit_counter' );
        if ( $results->rowCount() === 0 ) {


            $data->total = 0;
            $data->unique = 0;

            $stmt = $this->mysql->prepare( 'INSERT INTO hit_counter( `total_hits`, `unique_hits` ) VALUES(:total, :unique)' );
            $stmt->bindParam( ':total', $data->total );
            $stmt->bindParam( ':unique', $data->unique );
            $stmt->execute();

        } else {
            $rows = $results->fetchAll( PDO::FETCH_OBJ );
            $data->total = $rows[0]->total_hits;
            $data->unique = $rows[0]->unique_hits;
        }

        return $data;
    }

    private function isNewVisitor() {
        return !array_key_exists( 'visited', $_SESSION ) && $_SESSION['visited'] !== true;
    }

    private function visit() {

        $this->mysql->query( "UPDATE hit_counter SET total_hits = total_hits + 1" );

        if ( $this->isNewVisitor() ) {
            $this->mysql->query( "UPDATE hit_counter SET unique_hits = unique_hits + 1");
            $_SESSION['visited'] = true;
        }
```

5

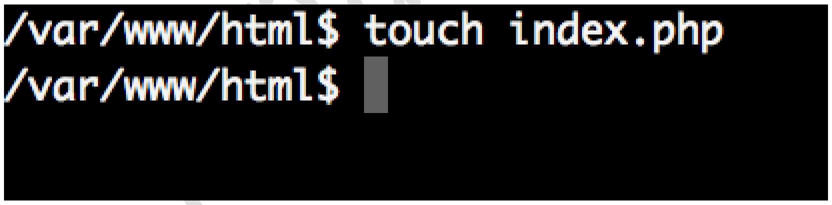
Fill in the remaining helper methods. These methods do the brunt of the work for the hit counter. They're marked private so that they can only be used internally.

getData() retrieves the current view counts from the database. If there isn't any data, it sets this data to zero.

isNewVisitor() determines whether the visitor has already visited our website in their current session

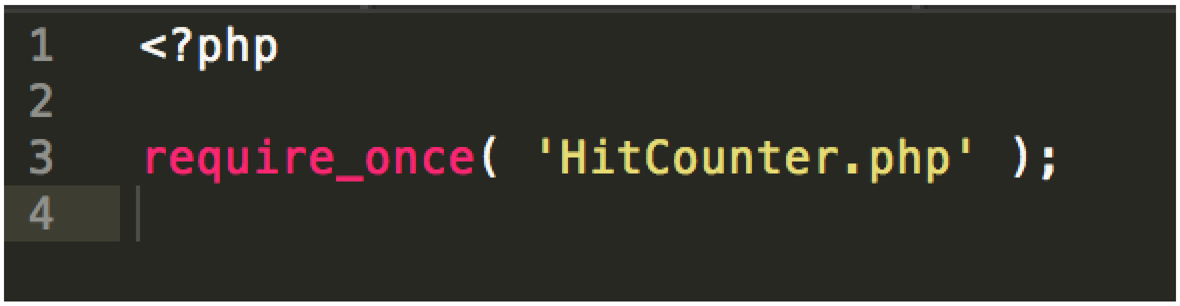
visit() increments the total hit counter, and increments the unique visitor counter if the user has not visited the website in their current session.

Using the Hit Counter



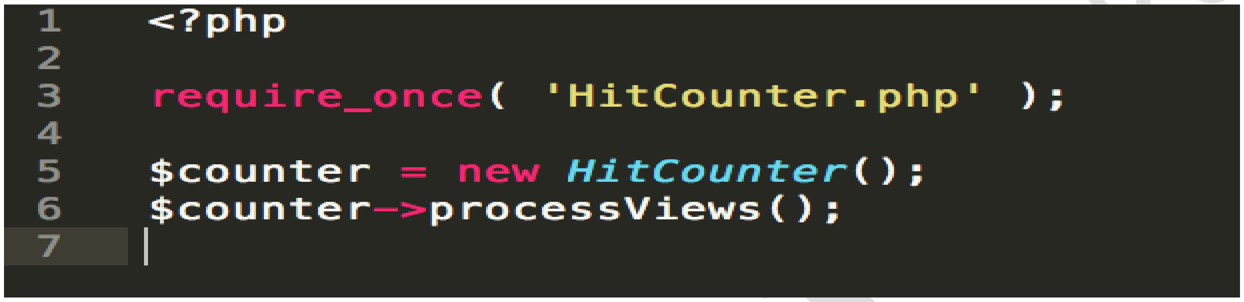
```
/var/www/html$ touch index.php
/var/www/html$
```

Create a new file to use your hit counter. This should be an actual page you intend the visitor to see. If you already have a website, this should be the front facing PHP script (usually index.php). Basically, any PHP file that is accessible from the web and you want to use to update the counter.



```php
<?php

require_once( 'HitCounter.php' );

```

2
Include the file containing the HitCounter class. It's recommended to use require_once over include.



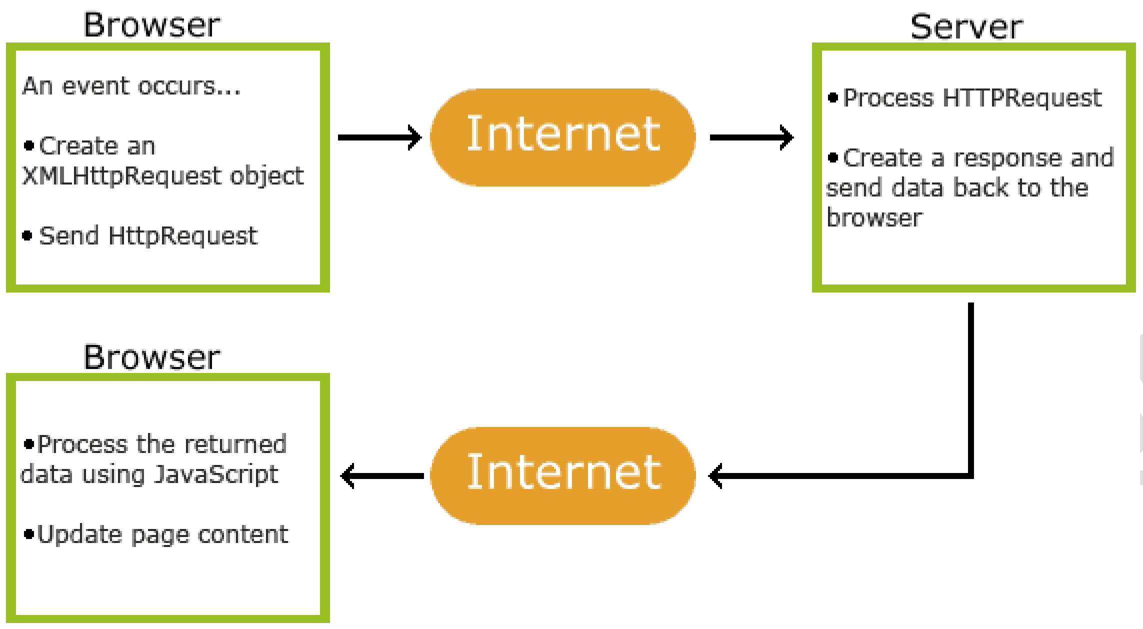
```php
<?php

require_once( 'HitCounter.php' );

$counter = new HitCounter();
$counter->processViews();

```

Unit-V
AJAX

Getting started with ajax:

AJAX = Asynchronous JavaScript and XML.
AJAX is a technique for creating fast and dynamic web pages.
AJAX allows web pages to be updated asynchronously by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, without reloading the whole page.
Classic web pages, (which do not use AJAX) must reload the entire page if the content should change.
Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

How AJAX Works


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

Writing Ajax:

Creating the XMLHttp request object:

First, check if the input field is empty (str.length == 0). If it is, clear the content of the txtHint placeholder and exit the function.
However, if the input field is not empty, do the following:
Create an XMLHttpRequest object
Create the function to be executed when the server response is ready
Send the request off to a PHP file (gethint.php) on the server
Notice that q parameter is added to the url (gethint.php?q="+str)
And the str variable holds the content of the input field

Example

```
<html>
<head>
<script>
function showHint(str) {
    if (str.length == 0) {
        document.getElementById("txtHint").innerHTML = "";
        return;
    } else {
        var xmlhttp = new XMLHttpRequest();
        xmlhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                document.getElementById("txtHint").innerHTML = this.responseText;
            }
        };
        xmlhttp.open("GET", "gethint.php?q=" + str, true);
        xmlhttp.send();
    }
}
</script>
</head>
<body>

<p><b>Start typing a name in the input field below:</b></p>
<form>
First name: <input type="text" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
</body>
</html>
```

Start typing a name in the input field below:

First name:

Suggestions:

Opening the XMLHttp request object:

When a user selects an RSS-feed in the dropdown list above, a function called "showRSS()" is executed. The function is triggered by the "onchange" event:

```
<html>
<head>
<script>
function showRSS(str) {
  if (str.length==0) {
    document.getElementById("rssOutput").innerHTML="";
    return;
  }
  if (window.XMLHttpRequest) {
    // code for IE7+, Firefox, Chrome, Opera, Safari
    xmlhttp=new XMLHttpRequest();
  } else {  // code for IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
  xmlhttp.onreadystatechange=function() {
    if (this.readyState==4 && this.status==200) {
      document.getElementById("rssOutput").innerHTML=this.responseText;
    }
  }
  xmlhttp.open("GET","getrss.php?q="+str,true);
  xmlhttp.send();
}
</script>
</head>
<body>

<form>
<select onchange="showRSS(this.value)">
```

```
<option value="">Select an RSS-feed:</option>
<option value="Google">Google News</option>
<option value="NBC">NBC News</option>
</select>
</form>
<br>
<div id="rssOutput">RSS-feed will be listed here...</div>
</body>
</html>
```

handling downloaded data:

In this tutorial you'll learn how to collect user inputs submitted through a form using the PHP superglobal variables *$_GET*, *$_POST* and *$_REQUEST*.
Creating a Simple Contact Form
In this tutorial we are going to create a simple HMTL contact form that allows users to enter their comment and feedback then displays it to the browser using PHP.
Open up your favorite code editor and create a new PHP file. Now type the following code and save this file as "contact-form.php" in the root directory of your project.
Example
Download

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Contact Form</title>
</head>
<body>
   <h2>Contact Us</h2>
   <p>Please fill in this form and send us.</p>
   <form action="process-form.php" method="post">
      <p>
         <label for="inputName">Name:<sup>*</sup></label>
         <input type="text" name="name" id="inputName">
      </p>
      <p>
         <label for="inputEmail">Email:<sup>*</sup></label>
         <input type="text" name="email" id="inputEmail">
      </p>
      <p>
         <label for="inputSubject">Subject:</label>
         <input type="text" name="subject" id="inputSubject">
      </p>
      <p>
         <label for="inputComment">Message:<sup>*</sup></label>
         <textarea name="message" id="inputComment" rows="5" cols="30"></textarea>
      </p>
      <input type="submit" value="Submit">
```

```
        <input type="reset" value="Reset">
    </form>
</body>
</html>
```

Explanation of code
Notice that there are two attributes within the opening <form> tag:
The action attribute references a PHP file "process-form.php" that receives the data entered into the form when user submit it by pressing the submit button.
The method attribute tells the browser to send the form data through POST method.
Rest of the elements inside the form are basic form controls to receive user inputs. To learn more about HTML form elements please check out the HTML Forms tutorial.

Starting download:

To access the value of a particular form field, you can use the following superglobal variables. These variables are available in all scopes throughout a script.

Superglobal	Description
$_GET	Contains a list of all the field names and values sent by a form using the get method (i.e. via the URL parameters).
$_POST	Contains a list of all the field names and values sent by a form using the post method (data will not visible in the URL).
$_REQUEST	Contains the values of both the $_GET and $_POST variables as well as the values of the $_COOKIE superglobal variable.

When a user submit the above contact form through clicking the submit button, the form data is sent to the "process-form.php" file on the server for processing. It simply captures the information submitted by the user and displays it to browser.
The PHP code of "process-form.php" file will look something like this:
Example
Download

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Contact Form</title>
</head>
<body>
    <h1>Thank You</h1>
    <p>Here is the information you have submitted:</p>
    <ol>
        <li><em>Name:</em> <?php echo $_POST["name"]?></li>
        <li><em>Email:</em> <?php echo $_POST["email"]?></li>
        <li><em>Subject:</em> <?php echo $_POST["subject"]?></li>
        <li><em>Message:</em> <?php echo $_POST["message"]?></li>
    </ol>
</body>
```

</html>
The PHP code above is quite simple. Since the form data is sent through the post method, you can retrieve the value of a particular form field by passing its name to the $_POST superglobal array, and displays each field value using echo() statement.
In real world you cannot trust the user inputs; you must implement some sort of validation to filter the user inputs before using them. In the next chapter you will learn how sanitize and validate this contact form data and send it through the email using PHP.

Ajax with some php:

To clearly illustrate how easy it is to access information from a database using Ajax and PHP, we are going to build MySQL queries on the fly and display the results on "ajax.html". But before we proceed, lets do ground work. Create a table using the following command.

```
CREATE TABLE `ajax_example` (
  `name` varchar(50) NOT NULL,
  `age` int(11) NOT NULL,
  `sex` varchar(1) NOT NULL,
  `wpm` int(11) NOT NULL,
  PRIMARY KEY  (`name`)
)
```

Now dump the following data into this table using the following SQL statements.

```
INSERT INTO `ajax_example` VALUES ('Jerry', 120, 'm', 20);
INSERT INTO `ajax_example` VALUES ('Regis', 75, 'm', 44);
INSERT INTO `ajax_example` VALUES ('Frank', 45, 'm', 87);
INSERT INTO `ajax_example` VALUES ('Jill', 22, 'f', 72);
INSERT INTO `ajax_example` VALUES ('Tracy', 27, 'f', 0);
INSERT INTO `ajax_example` VALUES ('Julie', 35, 'f', 90);
```

Handling XML:

The XMLHttpRequest Object has a built in XML Parser.
The **responseText** property returns the response as a string.
The **responseXML** property returns the response as an XML DOM object.
If you want to use the response as an XML DOM object, you can use the responseXML property.
Example
Request the file cd_catalog.xml and use the response as an XML DOM object:

```
xmlDoc = xmlhttp.responseXML;
txt = "";
x = xmlDoc.getElementsByTagName("ARTIST");
for (i = 0; i < x.length; i++) {
   txt += x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("demo").innerHTML = txt;
```

Handling XML with PHP:

The PHP simplexml_load_file() function is used to read XML data from a file.
Assume we have an XML file called "note.xml", that looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

The example below shows how to use the simplexml_load_file() function to read XML data from a file:

Example

```
<?php
$xml=simplexml_load_file("note.xml") or die("Error: Cannot create object");
print_r($xml);
?>
```

Output:

```
SimpleXMLElement Object ( [to] => Tove [from] => Jani [heading] => Reminder [body] => Don't forget
me this weekend! )
```

Advanced ajax

There’s no doubt that AJAX is a powerful and user-enhancing group of technologies, but its many possibilities still aren’t widely known. In this article, we’ll take a look at how easy it can be to create an extremely powerful data transfer between the server and the client-side AJAX engine, using JavaScript Object Notation (JSON) and the JSON parser. We’ll explore how to create a group of objects (often referred to as a package in other languages), how to serialize the objects as JSON to be sent to the server, and how to deserialize server-side JSON as client-side JavaScript objects.

This article assumes that you understand JavaScript and how to create a basic AJAX engine, make requests, and receive responses from the server via AJAX. To learn more about these topics, see my article "How To Use AJAX." To follow along with the examples, you’ll need to download the source files. (You can also view a live sample.)

Handling concurrent ajax:

This article uses an AJAX engine that I created in order to abstract our AJAX requests and help us to share AJAX engine code between different applications. In order to use this engine, we simply import three JavaScript files and make requests to an object named AjaxUpdater. The engine will handle the rest, including delegating the response to the callback method specified in the request. Here’s an example of how we would make requests with this engine as well as import the associated files:

```
<script type="text/javascript" src="javascript/model/Ajax.js"></script>
<script type="text/javascript" src="javascript/model/HTTP.js"></script>
<script type="text/javascript" src="javascript/model/AjaxUpdater.js"></script>
<script type="text/javascript">
   document.load = AjaxUpdater.Update(’GET’, URL, callback);
</script>
```

Java script inner functions:

JavaScript is often misunderstood, with a bad reputation from the silly graphic effects of the past. JavaScript is truly a powerful language, especially when combined with AJAX and the server side of an

application, but even on the client side JavaScript can accomplish much more than you would probably expect. Object-oriented JavaScript is one example, enabling us to create objects, extend intrinsic objects, and even create packages for our objects for easier management.
For this example, we'll create three objects: Auto, Car, and Wheel. Each is a very simple object, but will be used to show how to create a basic package.
The Auto object is declared as a new object:

```
var Auto = new Object();
```

The Auto object doesn't actually end here, though; it's used as the parent of the Car object. Therefore, the Car object is a property of the Auto object, but separated into another file for easier management. (This concept is used in other object-oriented languages, but JavaScript isn't often thought of in these terms.)
Here's the code for the Car object:

```
Auto.Car = new Object();

Auto.Car.color = "#fff";

Auto.Car.setColor = function(_color)
{
   Auto.Car.color = _color;
}
Auto.Car.setColor("#333");
```

As you can see, the Car object is a child of the Auto object—a classic case of object hierarchy. This object has a property named color and a method to set it. We're setting the color to gray, overwriting the default white. Keep this fact in mind until we serialize the object.
The next object, Wheel, is a child object of Car:

```
Auto.Car.Wheel = new Object();

Auto.Car.Wheel.color = "#000";
```

Wheel is a basic object, but it shows another layer to the object hierarchy. This object has one property called color, with a default value of black.

Downloading images using ajax:

AJAX was initially designed to be used with text data (JSON/HTML/XML) and that is why this requirement of downloading images using AJAX were never fulfilled.
But with HTML5, XHR2 has been introduced which allows us to get ArrayBuffer in ajax response and this is something which can be used to download image.
There are two approaches that could be taken.
Download the image, create a blob object and use it as src in img.
Download the image, and create data url and use it as src in img.

```
 var xhr = new XMLHttpRequest();
 xhr.onreadystatechange = function() {
   if (xhr.readyState==4 && xhr.status==200) {
       var blob = new Blob([xhr.response], {
          type: xhr.getResponseHeader("Content-Type")
       });
       var imgUrl = window.URL.createObjectURL(blob);
```

```
        document.getElementById("img").src = imgUrl;
      }
    }
  xhr.responseType = "arraybuffer";
  xhr.open("GET","Hacker.jpg",truexhr.send();

xhr.send();

  var xhr = new XMLHttpRequest();
  xhr.onreadystatechange = function() {
    if (xhr.readyState==4 && xhr.status==200) {
        document.getElementById("img").src = "data:"+xhr.getResponseHeader("Content-
Type")+";base64," + btoa(String.fromCharCode.apply(null, new Uint8Array(xhr.response)));
      }
    }
  xhr.responseType = "arraybuffer";
  xhr.open("GET","Hacker.jpg",truexhr.send();

xhr.send();
```

downloading javascript with ajax:

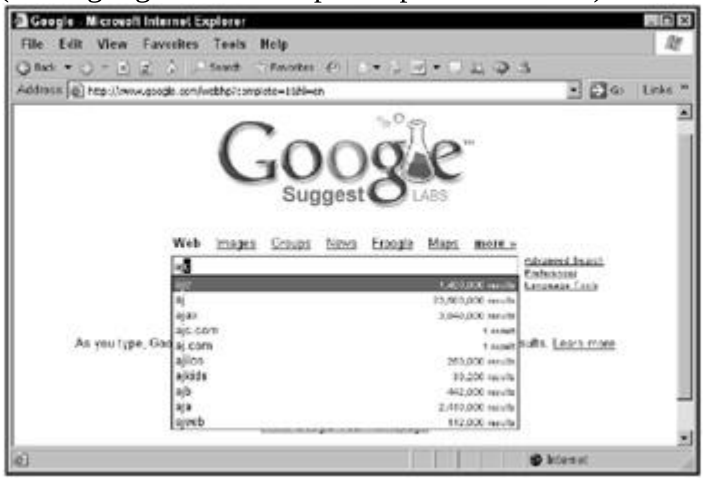
I have a javascript app that sends ajax POST requests to a certain URL. Response might be a JSON string or it might be a file (as an attachment). I can easily detect Content-Type and Content-Disposition in my ajax call, but once I detect that the response contains a file, how do I offer the client to download it? I've read a number of similar threads here but none of them provide the answer I'm looking for.
Please, please, please do not post answers suggesting that I shouldn't use ajax for this or that I should redirect the browser, because none of this is an option. Using a plain HTML form is also not an option. What I do need is to show a download dialog to the client. Can this be done and how?
EDIT:
Apparently, this cannot be done, but there is a simple workaround, as suggested by the accepted answer.
For anyone who comes across this issue in the future, here's how I solved it:

```
$.ajax({
    type: "POST",
    url: url,
    data: params,
    success: function(response, status, request) {
        var disp = request.getResponseHeader('Content-Disposition');
        if (disp && disp.search('attachment') != -1) {
            var form = $('<form method="POST" action="' + url + '">');
            $.each(params, function(k, v) {
                form.append($('<input type="hidden" name="' + k +
                        '" value="' + v + '">'));
            });
            $('body').append(form);
            form.submit();
        }
    }
});
```

So basically, just generate a HTML form with the same params that were used in AJAX request and submit it.

connecting to google suggest:

Among famous Ajax applications, there are a few that stand out, such as Google Suggest (www.google.com/webhp?complete=1&hl=en). It is shown in Figure 4.6.

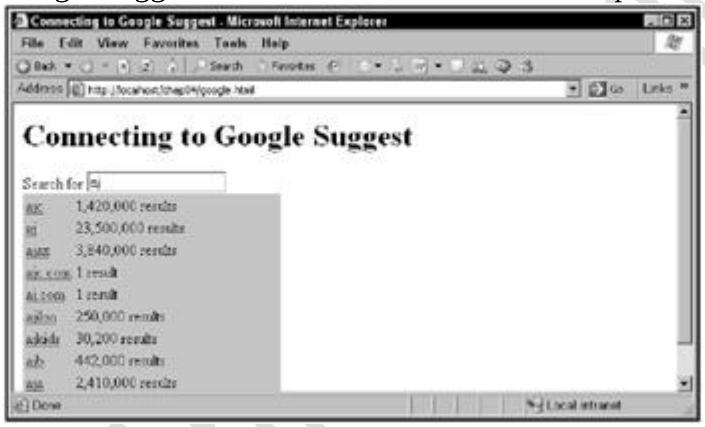

Google - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://www.google.com/webhp?complete=1&hl=en
Google Suggest LABS
Web Images Groups News Froogle Maps more »
Advanced Search
Preferences
Language Tools
Learn more
Internet

ajc	1,420,000 results
aj	23,500,000 results
ajax	3,840,000 results
ajc.com	1 result
aj.com	1 result
ajilon	265,000 results
ajkids	33,200 results
ajb	442,000 results
aja	2,410,000 results
ajweb	112,000 results

Figure 4.6: Google Suggest

When you enter a partial search term, as shown in the figure, Google Suggest connects, using Ajax, to the Google servers and finds possible matches to your partial search term and displays them-no page refresh needed. Clicking a hyperlink in the drop-down list opens the corresponding search page in Google.

You can create a search page yourself that connects to Google Suggest, and download JavaScript from Google Suggest to do so. You can see this example at work in Figure 4.7.


Connecting to Google Suggest - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/chap04/google.html
Connecting to Google Suggest
Search for aj
Done
Local intranet

ajc	1,420,000 results
aj	23,500,000 results
ajax	3,840,000 results
ajc.com	1 result
aj.com	1 result
ajilon	250,000 results
ajkids	30,200 results
ajb	442,000 results
aja	2,410,000 results

Figure 4.7: Connecting to Google Suggest with a custom page

So how can you connect to Google Suggest? Say that you've stored the partial search term in a variable named searchTerm. You could then connect to Google Suggest at:

```
 http://www.google.com/complete/search?hl=en&js=true&qu=" + searchTerm;
```

How does Google Suggest communicate with you? It sends back JavaScript code that calls a function named sendRPCDone. Here are the parameters passed to that function:

```
 sendRPCDone(unusedVariable, searchTerm, arrayTerm, arrayResults, unusedArray)
```

So what does the JavaScript call you get back from Google Suggest actually look like? If you're searching

for ajax, this is the kind of JavaScript you'll get back from Google:

sendRPCDone(frameElement, "ajax", new Array("ajax", "ajax amsterdam", "ajax fc", "ajax ontario", "ajax grips", "ajax football club", "ajax public library", "ajax football", "ajax soccer", "ajax pickering transit"), new Array("3,840,000 results", "502,000 results", "710,000 results", "275,000 results", "8,860 results", "573,000 results", "40,500 results", "454,000 results", "437,000 results", "10,700 results"), new Array(""));
You take it from there, writing your own sendRPCDone function that displays the results sent back to you from Google Suggest.

It's time to create google.html. The action starts when the user enters a search term in the search text field, which has the ID textField:
<body>   <H1>Connecting to Google Suggest</H1>   **Search for <input id = "textField" type = "text" name = "textField" onkeyup = "connectGoogleSuggest(event)">**   <div id = "targetDiv"><div></div></div> </body>
Note how this works. Every time a key goes up, the onkeyup event attribute calls a function named connectGoogleSuggest, which means that you can watch what users type as they type it. Every time the user types a key, the connectGoogleSuggest function is called:
function connectGoogleSuggest(keyEvent) {   .   .   . }
You can start the connectGoogleSuggest function by creating an object that corresponds to the input text field that the user has been typing into:
function connectGoogleSuggest(keyEvent) {   **var input = document.getElementById("textField");**   .   .   . }
Next, you can check whether that text field contains any text, and if not, you can clear the target <div> element, which displays the drop-down list of items from Google Suggest:
function connectGoogleSuggest(keyEvent) {   var input = document.getElementById("textField");   **if (input.value) {   .   .   .   }   else {   var targetDiv = document.getElementById("targetDiv"); targetDiv.innerHTML = "<div></div>";   }** }
If, on the other hand, the input text field does contain text, you can pass that text on to the getData function to connect to Google Suggest and get suggestions. You pass a relative URL, google.php?qu= and the search term, to getData like this:

function connectGoogleSuggest(keyEvent) {   var input = document.getElementById("textField");   if (input.value) {   **getData("google.php?qu=" + input.value);**   }   else {   var targetDiv = document.getElementById("targetDiv");   targetDiv.innerHTML = "<div></div>";   } }
The getData function creates an XMLHttpRequest object in the usual way:
function getData(dataSource) {   **var XMLHttpRequestObject = false;   if (window.XMLHttpRequest) {   XMLHttpRequestObject = new XMLHttpRequest();   } else if (window.ActiveXObject) {   XMLHttpRequestObject = new ActiveXObject("Microsoft.XMLHTTP");   }**   .   .   .   } }
and then connects to the relative URL passed to it, which is google.php?qu= and the search term:
function getData(dataSource) {   var XMLHttpRequestObject = false;   if (window.XMLHttpRequest) { XMLHttpRequestObject = new XMLHttpRequest();   } else if (window.ActiveXObject) { XMLHttpRequestObject = new   ActiveXObject("Microsoft.XMLHTTP");   }
if(XMLHttpRequestObject) {   XMLHttpRequestObject.open("GET", dataSource); XMLHttpRequestObject.onreadystatechange = function()   {   .   .   .   } XMLHttpRequestObject.send(null);   } }

DRAWING IMAGES ON THE SERVER

Creating and Drawing Images:

For now, let's start with the simplest possible GD example is a script that generates a black filled square. The code works with any version of GD that supports the PNG image format.

```
A black square on a white background (black.php)
<?php
 $im = ImageCreate(200,200);
 $white = ImageColorAllocate($im,0xFF,0xFF,0xFF);
 $black = ImageColorAllocate($im,0x00,0x00,0x00);
 ImageFilledRectangle($im,50,50,150,150,$black);
 header('Content-Type: image/png');
 ImagePNG($im);
?>
```

To see the result, simply point your browser at the *black.php* PHP page. To embed this image in a web page, use:

<img src="black.php">

Creating an image:

The first thing the code does is to call the imagecreate() function with the dimensions of the image, namely its width and height in that order. This function returns a resource identifier for the image which we save in $my_img. The identifier is needed for all our operations on the image.
If the function fails for some reason, it will return FALSE. If you want your code to be robust, you should test for this. The demo code above doesn't do any error checking, since it will clutter the example, making it harder to understand.
Since the output of my example script is the image itself, I send an "image/png" content type header to the browser telling it that what follows are the bytes of a PNG image. The function imagepng() is then called to generate the necessary image from my $my_img image identifer. Since I called imagepng()without a second parameter, the function automatically sends its output to the browser. If you prefer to save your image, don't call the header() function to output the header, and call imagepng() with the filename of the image for its second parameter, like the following:
imagepng($my_img, "my_new_image.png");
Your image does not have to be a PNG image. You can use imagegif() or imagejpeg() to create GIF and JPG images respectively. You should of course send the correct content type header for the type of image you are creating. For example, a jpeg image should have a content type of "image/jpeg" while a gif image "image/gif". Note though that GIF support may or may not necessarily be compiled into the version of the GD library your web host is using, so if you're not sure, use one of the other file formats.

- imagecreatefromgif (string $filename)
- imagecreatefromjpeg (string $filename)
- imagecreatefrompng (string $filename)

For example, if you created a GIF file called "mytemplate.gif", the function can be called as follows:
$myimage = imagecreatefromgif ("mytemplate.gif");
Like the basic imagecreate() function, these functions return FALSE if they fail to load the image for any reason.

Drawing lines:

Before we begin drawing on our image, there are two functions that we should consider, for added variety.
1. Line color can be modified using the imagecolorallocate() function, which we learned about before. It should be stored in a variable to be used later.
2. Line thickness can be modified using the imagesetthickness() function, which requires two parameters: imagesetthickness(image, thickness)
The imageline() function itself requires 6 parameters. The syntax is: imageline(image, x1, y1, x2, y2, color)

- image = Refers to the Image Resource That the Line Will Be Applied to
- x1 = x-coordinate For First Point
- y1 = y-coordinate For First Point
- x2 = x-coordinate For Second Point
- y2 = y-coordinate For Second Point
- color = Refers to the Line Color Identifier Created With imagecolorallocate()

```
<?php
 header('Content-type: image/png');
 $png_image = imagecreate(150, 150);
 imagecolorallocate($png_image, 15, 142, 210);
 $black = imagecolorallocate($png_image, 0, 0, 0);
 imageline($png_image, 0, 0, 150, 150, $black);
 imagepng($png_image);
 imagedestroy($png_image);
?>
```

Drawing rectangles:

Draw a line, rectangle, or polygon. You also want to be able to control if the rectangle or polygon is open or filled in. For example, you want to be able to draw bar charts or create graphs of stock quotes.

To draw a line, use ImageLine( ):
ImageLine($image, $x1, $y1, $x2, $y2, $color);

To draw an open rectangle, use ImageRectangle( ):
ImageRectangle($image, $x1, $y1, $x2, $y2, $color);

To draw a solid rectangle, use ImageFilledRectangle( ) :
ImageFilledRectangle($image, $x1, $y1, $x2, $y2, $color);

Drawing arcs:

To draw an arc, use ImageArc( ):

ImageArc($image, $x, $y, $width, $height, $start, $end, $color);

Drawing Polygons:

To draw an open polygon, use ImagePolygon( ):

$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImagePolygon($image, $points, count($points)/2, $color);

To draw a filled polygon, use ImageFilledPolygon( ):

$points = array($x1, $y1, $x2, $y2, $x3, $y3);
ImageFilledPolygon($image, $points, count($points)/2, $color);

Drawing individual pixels:

set individual pixels using the imagesetpixel method

imagesetpixel(resource image, int x, int y, int color)

As you'd expect, this function draws a pixel at x, y in image image of color color. When you're drawing graphics, working pixel-by-pixel is usually too slow. But you can generally get away with drawing extensive pixel graphics on the server because the extra few seconds usually aren't noticedthe Internet is slow enough anyway.
For example, say you wanted to draw a dotted line, which you can do with imagesetpixel. You can use a for loop, as here, where we're drawing one pixel and then skipping three pixels before drawing the next pixel:
$image_height = 100; $image_width = 300; $image = imagecreate($image_width, $image_height); $back_color = imagecolorallocate($image, 200, 200, 200); $drawing_color = imagecolorallocate($image, 0, 0, 0); for($loop_index = 20; $loop_index < 280; $loop_index += 3){ . . . }

Tilling images:

They all have the same width (120px) and differing heights.
This is what I have:

```
$finalbg = null;
for($i=0; $i<7; $i++) {
    $addbg = imagecreatefromjpeg('images/left/'.$url[$drawn]);
    $addsize = imagesy($addbg);

    if($finalbg != null) $basesize = imagesy($finalbg); else $basesize = 0;
    $newsize = $addsize+$basesize;

    $newbg = imagecreatetruecolor(120, $newsize);
    if($finalbg != null) imagecopy($newbg, $finalbg, 0, 0, 0, 0, 120, $basesize);
    imagecopy($newbg, $addbg, 0, $basesize, 0, 0, 120, $addsize);
    $finalbg = $newbg;
}

header( "Content-type: image/jpeg" );
imagejpeg($finalbg);
```

The sizes are outputting correctly, but it keeps telling the image contains errors, and I have no idea why :(Same thing if I try to output addbg or newbg.

Copying images:

bool copy (string $source , string $dest [, resource $context])
Makes a copy of the file **source** to **dest**.

If you wish to move a file, use the rename() function.