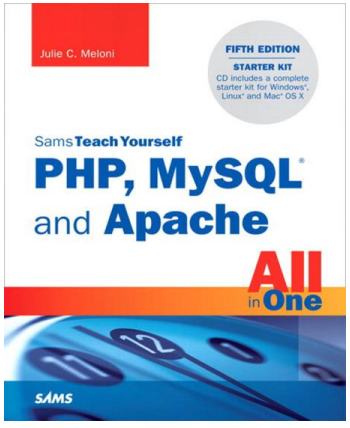
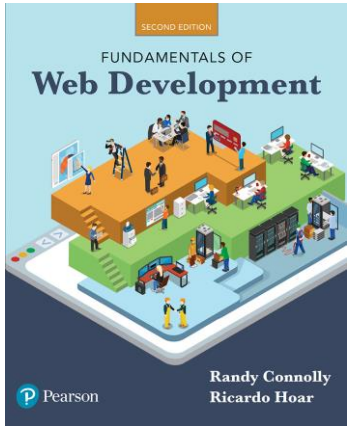
CSE 686 Internet Programming

Week 14: PHP

Edmund Yu, PhD
Associate Teaching Professor
esyu@syr.edu

April 16 & 18, 2018

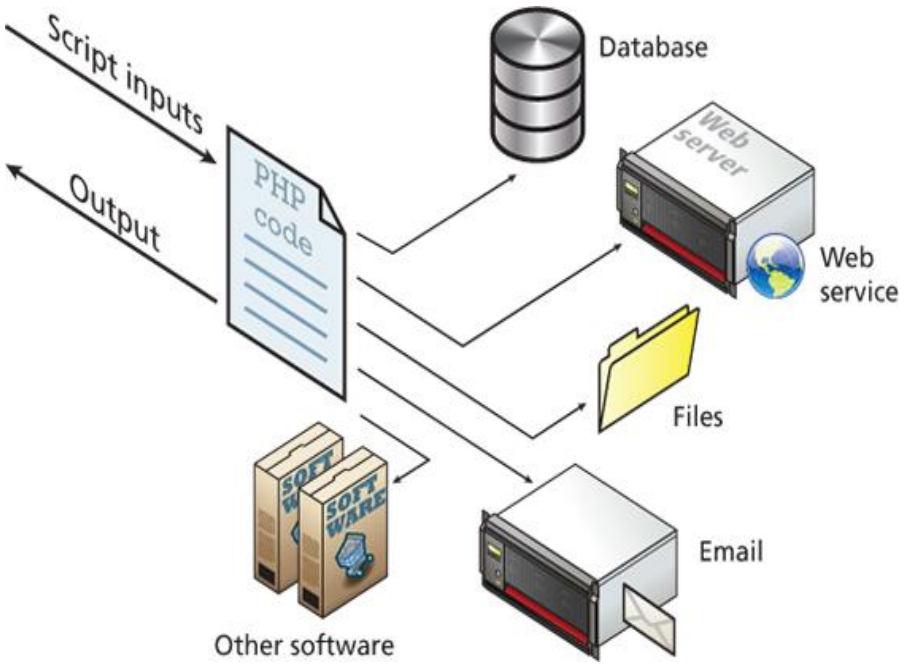




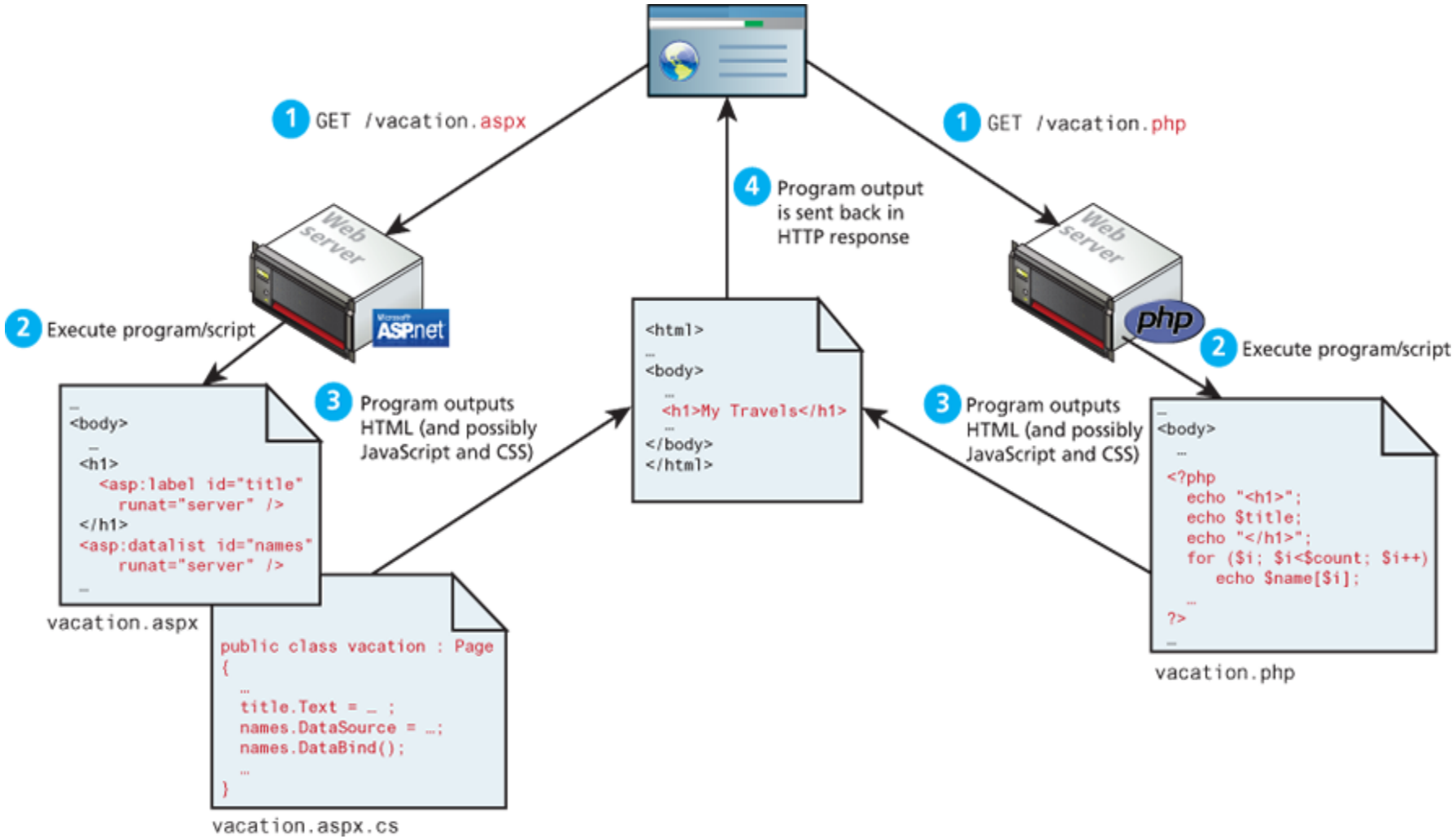
PHP

- PHP, which stands for "**PHP: Hypertext Preprocessor**" is a widely-used open-source general-purpose scripting language that is especially suited for Web development

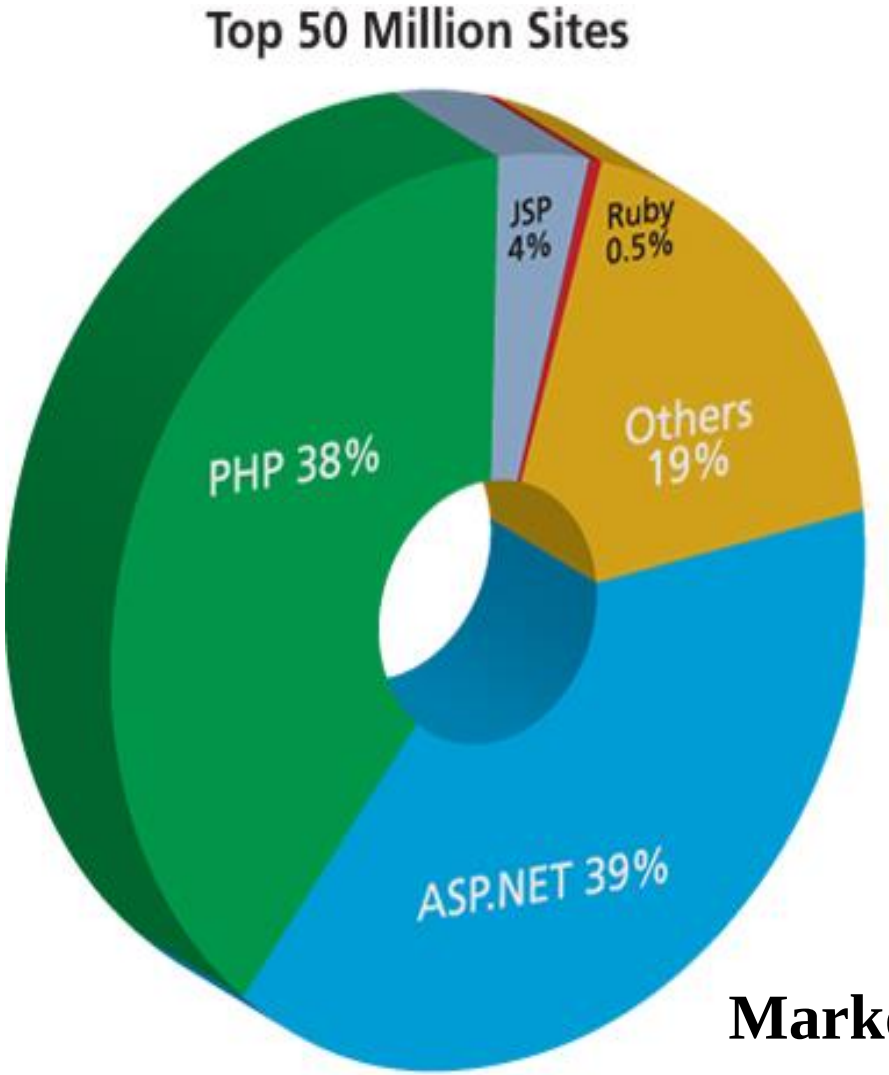
- It can be also embedded into HTML.
- Its syntax draws upon C, Java, and Perl, and is easy to learn.
- The main goal of the language is to allow web developers to write dynamically generated web pages quickly, but you can do much more with PHP.

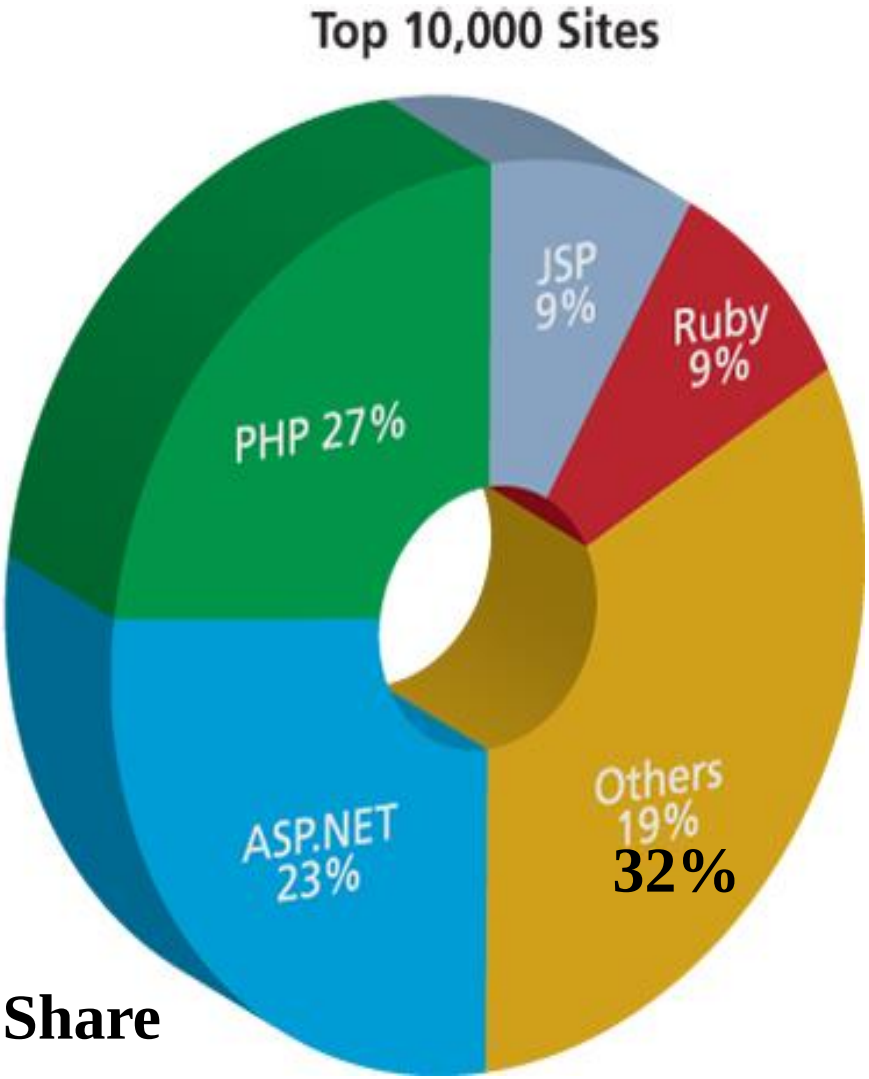


PHP vs ASP.NET Revisited



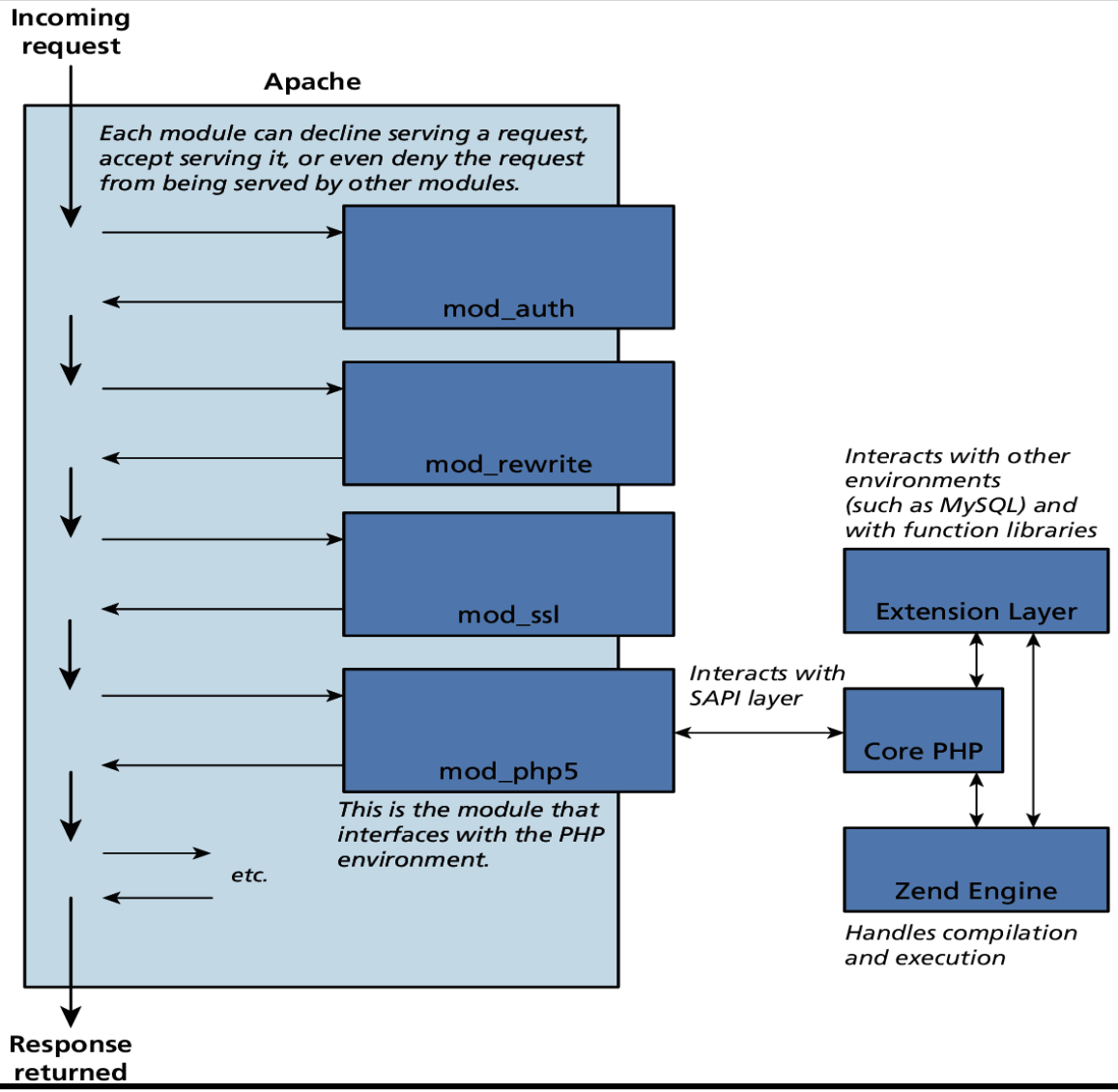
Server-Side Technologies



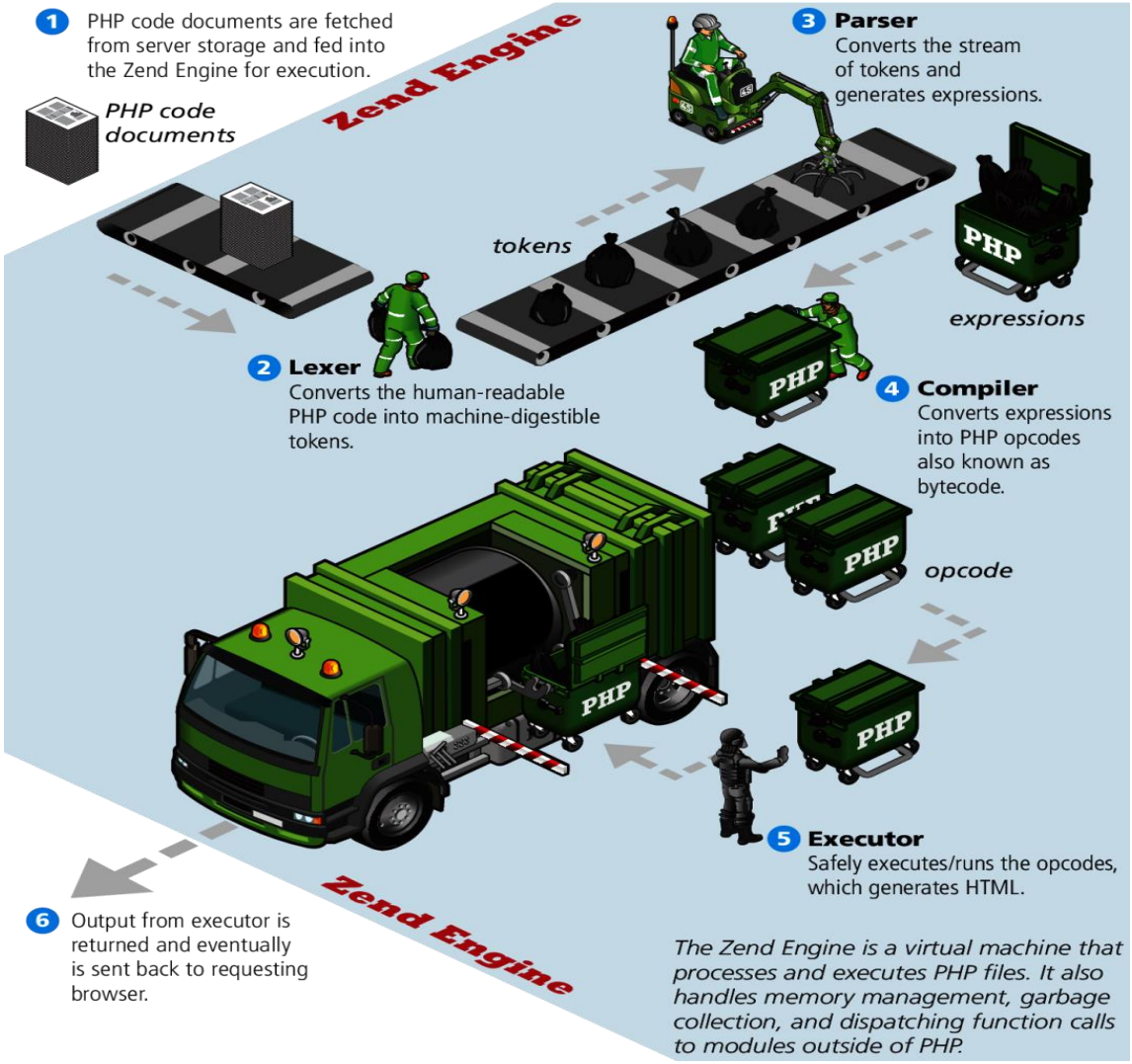


Market Share

Apache and PHP



Zend Engine


1 PHP code documents are fetched from server storage and fed into the Zend Engine for execution.
PHP code documents
Zend Engine
3 Parser
Converts the stream of tokens and generates expressions.
tokens
PHP
expressions
2 Lexer
Converts the human-readable PHP code into machine-digestible tokens.
PHP
4 Compiler
Converts expressions into PHP opcodes also known as bytecode.
PHP
opcode
PHP
PHP
5 Executor
Safely executes/runs the opcodes, which generates HTML.
Zend Engine
6 Output from executor is returned and eventually is sent back to requesting browser.
The Zend Engine is a virtual machine that processes and executes PHP files. It also handles memory management, garbage collection, and dispatching function calls to modules outside of PHP.

PHP

- Assuming that your server has activated support for PHP and that all files ending in *.php* are handled by PHP.
  - If you are using XAMPP, PHP is activated by default.
  - On most servers, this is the default extension for PHP files, but ask your server administrator to be sure.
  - If your server supports PHP, then you do not need to do anything.
    - Just create your *.php* files, put them in your **web directory** (Document Root) and the server will automatically parse them for you.
  - There is no need to compile anything nor do you need to install any extra tools.
    - Think of these PHP-enabled files as simple HTML files with a whole new family of magical tags that let you do all sorts of things.

Test Your Installation

```
<?php  phpinfo();  ?>
```

- Use your browser to access the file with your web server's URL, ending with the ***/phpinfo.php*** file reference. When developing locally this URL will be something like ***http://localhost/phpinfo.php*** or ***http://127.0.0.1/phpinfo.php*** but this depends on the web server's configuration.

List of PHP editors

From Wikipedia, the free encyclopedia

This article contains a **list of text editors** with features specific to the PHP scripting language.

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(March 2010)*

Contents [hide]

Free editors [edit]

Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikimedia Shop

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox
Print/export
Languages

first.php

```
<?php
    echo "<h1>Hello Web!</h1>";
?>
```

Beginning and Ending a Block

- When writing PHP, you need to inform the PHP engine that you want it to execute your commands.
  - If you don't do this, the code you write will be mistaken for HTML and will be output to the browser.

- You can designate your code as PHP with special tags that mark the beginning and end of PHP code blocks.
  - Standard tags <?php ?>
  - Short tags <? ?>
  - ASP tags <% %>
  - Script tags <script language="php"> </script>

Beginning and Ending a Block

- Only the standard and script tags are guaranteed to work on any configuration.
  - You must explicitly enable the short and ASP-style tags in your **php.ini** file.
    - To activate recognition for short tags, you must make sure that the short_open_tag switch is set to On in php.ini:

      short_open_tag = **On**;

    - To activate recognition for the ASP-style tags, you must enable the asp_tags setting in php.ini:

      asp_tags **= On;**

The echo Statement

- You can use the **echo** statement to output data.
  - In most cases, anything output by echo ends up viewable in the browser.
  - Alternatively, you could have used the **print** statement in place of the echo statement.
  - Using echo or print is a matter of taste
  - When you look at other people's scripts, you might see either one used.

The echo Statement

- Note the only line of code in **first.php** ends with a semicolon.
  - The semicolon informs the PHP engine that you have completed a statement, and is probably the most important bit of coding syntax you could learn at this stage.
  - A **statement** represents an instruction to the PHP engine.
    - Exceptions to this rule include statements that enclose other statements and statements that end a block of code.
    - In most cases, however, failure to end a statement with a semicolon will confuse the PHP engine and result in an error.

helloworld.php

```
<!DOCTYPE html>
<html>
<head>
<title>A PHP script including HTML</title>
</head>
<body>
<h1><?php echo "hello world"; ?></h1>
</body>
</html>
```

- If everything is configured correctly, this file will be parsed by PHP and the following output will be sent to your browser:

helloworld.php Output

hello world

helloworld.php

- As you can see, incorporating PHP code into a predominantly HTML document is simply a matter of typing in the code.
  - The PHP engine ignores everything outside the PHP open and close tags.

Adding Comments to PHP Code

- Single-line comments (preferred) begin with two forward slashes (//), or a single hash or pound sign (#). The PHP engine ignores all text between these marks and either the end of the line or the PHP close tag:

```
// this is a comment
# this is another comment
```

- Multiline comments begin with a forward slash followed by an asterisk (/*) and end with an asterisk followed by a forward slash (*/):

```
/*
this is a comment
none of this will
be parsed by the
PHP engine
*/
```

Variables

- A variable consists of a name of your choosing, preceded by a dollar sign ($).
  - Variable names can include letters, numbers, and the underscores, but they cannot include spaces.
  - Names must begin with a letter or an underscore. The following list shows some legal variables:

```
$a;
$a_longish_variable_name;
$_2453;
$sleepyZZZZ;
```

  - **Names are case-sensitive.**

Variables

- The scope of a variable is the part of the script where the variable can be referenced/used.
  - PHP has four different variable scopes:
    - local
    - global
    - static
    - parameter

Local Variables

- A variable declared **within** a PHP function is local and can only be accessed within that function:

```
<?php
    function myTest()
    {
        $x=5; // local scope
        echo $x;
    }

    myTest();
?>
```

Global Variables

- A variable defined outside of any function has a global scope.
  - Global variables can be accessed from any part of the script, EXCEPT from within a function.
  - To access a global variable from within a function, use the **global** keyword:

```
<?php
    $x=5; // global scope
    $y=10; // global scope
    function myTest2()
    {
        global $x, $y;
        $y=$x+$y;
        echo $y;
        echo "<br>";
    }
    myTest2();
?>
```

Superglobals

- In addition to global variables of your own creation, PHP has several predefined variables called **superglobals**. These variables are always present, and their values are available to all your scripts. Each of the following superglobals is actually an array of other variables:

$_GET contains any variables provided to a script through the GET method.

$_POST contains any variables provided to a script through the POST method.

$_COOKIE contains any variables provided to a script through a cookie.

$_FILES contains any variables provided to a script through file uploads.

$_SERVER contains information such as headers, file paths, and script locations.

$_ENV contains any variables provided to a script as part of the server environment.

$_REQUEST contains any variables provided to a script via GET/POST/COOKIE.

$_SESSION contains any variables that are currently registered in a session.

Static Variables

- When a function is completed, all of its variables are normally deleted.
  - If you want a local variable to not be deleted. To do this, use the **static** keyword when you first declare the variable:

```
<?php
    function myTest3()
    {
        static $x=0;
        echo $x;
        $x++;
    }
    myTest3();
    myTest3();
    myTest3();
?>
```

Parameter Variables

- 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:

```
<?php
   function myTest4($x)
   {
   echo $x;
   }

   myTest4(5);
?>
```

TABLE 5.1 Standard Data Types

Type	Example	Description
Boolean	`true`	One of the special values `true` or `false`
Integer	`5`	A whole number
Float or double	`3.234`	A floating-point number
String	`"hello"`	A collection of characters
Object		An instance of a class
Array		An ordered set of keys and values
Resource		Reference to a third-party resource (a database, for example)
NULL		An uninitialized variable

testtype.php

```
<?php
$testing; // declare without assigning
echo "is null? ".is_null($testing); // checks if null
echo "<br/>";
$testing = 5;
echo "is an integer? ".is_int($testing); // checks if integer
echo "<br/>";
$testing = "five";
echo "is a string? ".is_string($testing); // checks if string
echo "<br/>";
$testing = 5.024;
echo "is a double? ".is_double($testing); // checks if double
echo "<br/>";
```

testtype.php, cont.

```
$testing = true;
echo "is boolean? ".is_bool($testing); // checks if boolean
echo "<br/>";
$testing = array('apple', 'orange', 'pear');
echo "is an array? ".is_array($testing); // checks if array
echo "<br/>";
echo "is numeric? ".is_numeric($testing); // checks if is numeric
echo "<br/>";
echo "is a resource? ".is_resource($testing); // checks if is a resource
echo "<br/>";
echo "is an array? ".is_array($testing); // checks if is an array
echo "<br/>";
?>
```

The Concatenation Operator

- The concatenation operator is represented by a single period:

  (left_operand)**.(**right_operand)

  - Treating both operands as strings, this operator appends the right-side operand to the left-side operand.
    - "hello"."□world" → "hello□world"
    - Note that the resulting space between the words occurs because there is a leading space in the second operand (" world" rather than "world").
  - The concatenation operator literally smashes together two strings without adding any padding.
    - "hello"."world" → "helloworld"

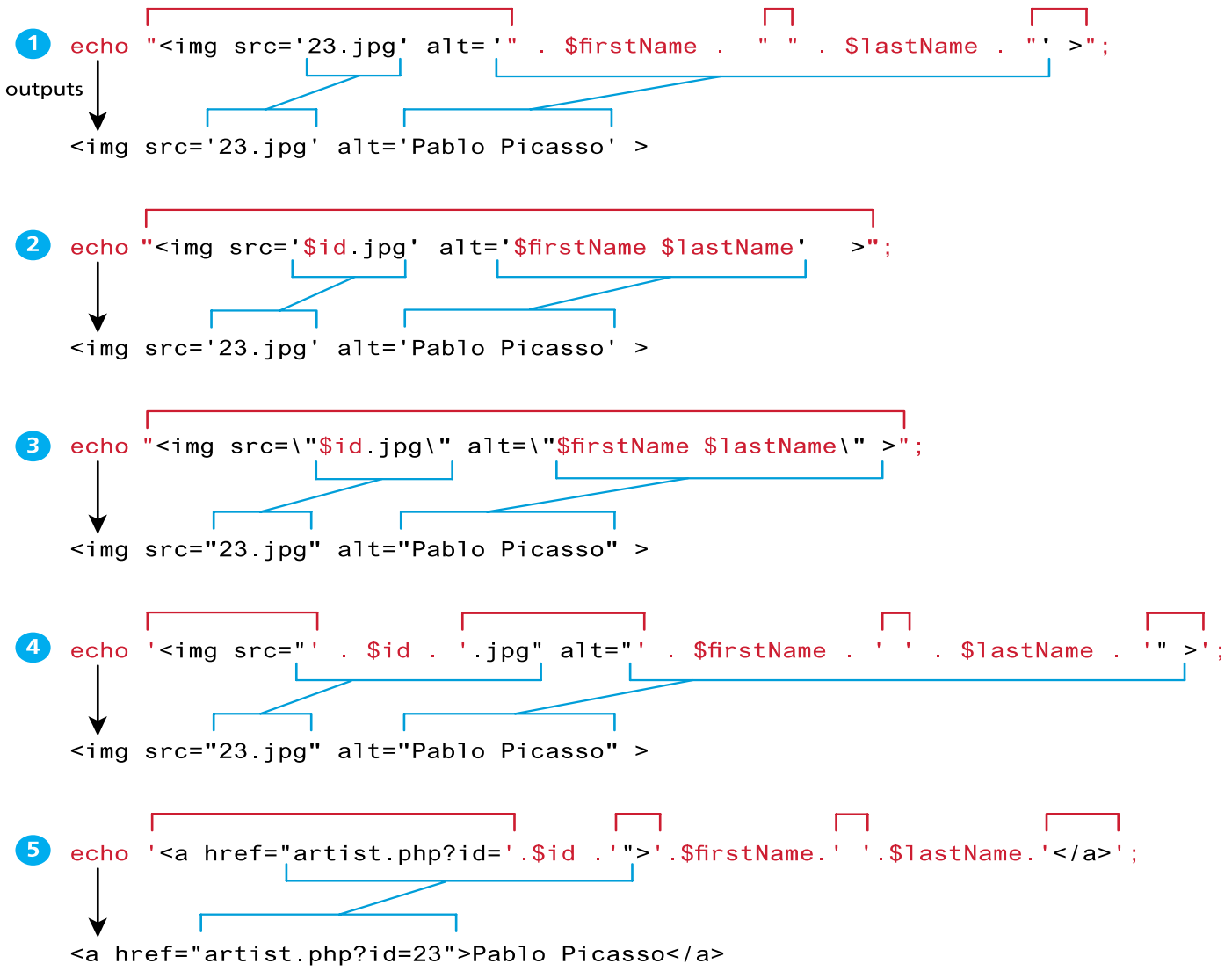
The Concatenation Operator

- Regardless of the data types of the operands used with the concatenation operator, they are treated as **strings**, and the result is always of the string type.

  - You will encounter concatenation frequently when the results of an expression of some kind must be combined with a string, as in the following:

```
$cm = 212;
echo “the width is ”.($cm/100). “ meters”;
```

The Concatenation Operator



settype()

- PHP also provides the function settype(), which is used to change the type of a variable.
  - To use settype(), you place the variable to change and the type to change it to between the parentheses and separate the elements with a comma, like this:

```
settype($variabletochange, ‘new type’);
```

settype.php

```
<?php
$undecided = 3.14;
echo "is ".$undecided." a double? ".is_double($undecided)."<br/>"; // double
settype($undecided, 'string');
echo "is ".$undecided." a string? ".is_string($undecided)."<br/>"; // string
settype($undecided, 'integer');
echo "is ".$undecided." an integer? ".is_integer($undecided)."<br/>"; // integer
settype($undecided, 'double');
echo "is ".$undecided." a double? ".is_double($undecided)."<br/>"; // double
settype($undecided, 'bool');
echo "is ".$undecided." a boolean? ".is_bool($undecided)."<br/>"; // boolean
?>
```

Casting

- The principal difference between using settype() to change the type of an existing variable and changing type by **casting** is that casting produces a copy, leaving the original variable untouched.
- To change type through casting, you indicate the name of a data type, in parentheses, in front of the variable you are copying.
- An example:

```
$newvar = (integer) $originalvar
```

  - It creates a copy of the $originalvar variable, with a specific type (integer) and a new name $newvar.
  - $originalvar is still available, and is its original type
  - $newvar is a completely new variable.

casttype.php

```
<?php
$undecided = 3.14;
$holder = (double) $undecided;
echo "is ".$holder." a double? ".is_double($holder)."<br/>"; // double
$holder = (string) $undecided;
echo "is ".$holder." a string? ".is_string($holder)."<br/>"; // string
$holder = (integer) $undecided;
echo "is ".$holder." an integer? ".is_integer($holder)."<br/>"; // integer
$holder = (double) $undecided;
echo "is ".$holder." a double? ".is_double($holder)."<br/>"; // double
$holder = (boolean) $undecided;
echo "is ".$holder." a boolean? ".is_bool($holder)."<br/>"; // boolean
echo "<hr/>";
echo "original variable type of $undecided: ";
echo gettype($undecided); // double
?>
```

Casting

- Casting is not a procedure that you will have to use often because PHP automatically casts your variables for you when the context of the script requires a change.
  - However, such an automatic cast is temporary, and you might want to make a variable persistently hold a particular data type, which is why PHP gives you the ability to specifically change types.

The Assignment Operator

- The assignment operator consists of the single character =
- The assignment operator takes the value of the right-side operand and assigns it to the left-side operand:

  $name = “Jimbo”;

- The variable $name now contains the string “Jimbo”.
- A statement that uses the assignment operator always resolves to a copy of the value of the right operand.
- Therefore:

  echo $name = “Jimbo”;

  prints the string “Jimbo” to the browser while it also assigns the value “Jimbo” to the $name variable.

TABLE 5.2 Arithmetic Operators

Operator	Name	Example	Sample Result
+	Addition	10+3	13
-	Subtraction	10-3	7
/	Division	10/3	3.333333333333
*	Multiplication	10*3	30
%	Modulus	10%3	1

TABLE 5.3 Some Combined Assignment Operators

Operator	Example	Equivalent To
+=	$x += 5	$x = $x + 5
.=	$x -= 5	$x = $x - 5
/=	$x /= 5	$x = $x / 5
*=	$x *= 5	$x = $x * 5
%=	$x %= 5	$x = $x % 5
.=	$x .= " test"	$x = $x." test"

The Post-Increment/Decrement

- Like other popular programming languages, PHP provides some special operators that allow you to add or subtract the integer constant 1 from an integer variable, assigning the result to the variable itself.
  - These are known as the **post-increment** and **post-decrement** operators.
  - The post-increment operator consists of two plus symbols appended to a variable name:

```
$x++; // $x is incremented by 1
```

  - Using two minus symbols in the same way decrements the variable:

```
$x--; // $x is decremented by 1
```

TABLE 5.4 Comparison Operators

Operator	Name	Returns True If...	Example ($x Is 4)	Result
==	Equivalence	Left is equivalent to right.	$x == 5	false
!=	Nonequivalence	Left is not equivalent to right.	$x != 5	true
===	Identical	Left is equivalent to right, and they are the same type.	$x === 4	true
	Nonequivalence	Left is equivalent to right, but they are not the same type.	$x === "4"	false
>	Greater than	Left is greater than right.	$x > 4	false
>=	Greater than or equal to	Left is greater than or equal to right.	$x >= 4	true
<	Less than	Left is less than right.	$x < 4	false
<=	Less than or equal to	Left is less than or equal to right.	$x <= 4	true

TABLE 5.5 Logical Operators

Operator	Name	Returns True If...	Example	Result
`||`	Or	Left or right is true.	`true || false`	`true`
`or`	Or	Left or right is true.	`true or false`	`true`
`xor`	Xor	Left or right is true, but not both.	`true xor true`	`false`
`&&`	And	Left and right are true.	`true && false`	`false`
`and`	And	Left and right are true.	`true and false`	`false`
`!`	Not	The single operand is not true.	`! true`	`false`

Constants

- You must use PHP's built-in define() function to create a constant, which subsequently cannot be changed unless you specifically define() it again.
  - To use the define() function, place the name of the constant and the value you want to give it within parentheses and separated by a comma:

    define("YOUR_CONSTANT_NAME", 42);

  - The value you want to set can be a number, a string, or a Boolean. By convention, the name of the constant should be in capital letters. Constants are accessed with the constant name only; no dollar symbol is required.

Constant.php

```
<?php
define(“THE_YEAR”, “2017”);
echo “It is the year “.THE_YEAR;
?>
```

define()

- The **define()** function can also accept a third Boolean argument that determines whether the constant name should be case sensitive.
  - **By default, constant names are case sensitive.**
  - However, by passing true to the define() function, you can change it to case-insensitive:

```
define("THE_YEAR", "2015", true);
```

  - You now could access its value without worrying about case:

```
echo the_year;
echo ThE_YeAr;
echo THE_YEAR;
```

Predefined Constants

- PHP provides some built-in constants for you:
  - __FILE__ : returns the name of the file that the PHP engine is currently reading.
  - __LINE__ : returns the current line number of the file.
  - PHP_VERSION: returns the version of PHP is interpreting the script. Useful for bug reports.

- For a list of predefined constants, see: http://www.php.net/manual/en/language.constants.predefined.php

The if Statement

- This functionality enables scripts to make decisions based on any number of factors:

```
if (expression) {
    // code to execute if the expression evaluates to true
}
```

- The if statement evaluates an expression found between parentheses. If this expression results in a true value, the statement is executed. Otherwise, the statement is skipped entirely.

testif.php

```
<?php
$mood = “happy”;
if ($mood == “happy”) {
        echo “Hooray! I’m in a good mood!”;
}
?>
```

The if-else Statement

```
if (expression) {
        // code to execute if the expression evaluates to true
} else {
        // code to execute in all other cases
}
```

```
<?php
$mood = “sad”;
if ($mood == “happy”) {
        echo “Hooray! I’m in a good mood!”;
} else {
        echo “I’m in a $mood mood.”;
}
?>
```

The elseif Clause

- You can use an if...elseif...else clause to test multiple expressions before offering a default block of code :

```
if (expression) {
        // code to execute if the expression evaluates to true
} elseif (another expression) {
        // code to execute if the previous expression failed
        // and this one evaluates to true
} else {
        // code to execute in all other cases
}
```

testifelseif.php

```
<?php
$mood = "sad";
if ($mood == "happy") {
        echo "Hooray! I'm in a good mood!";
} elseif ($mood == "sad") {
         echo "Awww. Don't be down!";
} else {
        echo "I'm neither happy nor sad, but $mood.";
}
?>
```

The elseif Clause

- The **elseif** clause can also be written as two words (**else if**).

- The syntax you use is a matter of taste, but coding standards employed by PEAR (the PHP Extension and Application Repository) use **elseif**.

PEAR - PHP Extension and Application Repository

» What is it?

PEAR is a framework and distribution system for reusable PHP components.

Sounds good? Perhaps you might want to know about **installing PEAR on your system** or installing pear packages.

You can find help using PEAR packages in the online manual and the FAQ.

If you have been told by other PEAR developers to sign up for a PEAR website account, you can use this interface.

» Hot off the Press

Security Vulnerability Announcement: HTML_AJAX

Another vulnerability in the HTML_AJAX package has been found which potentially allows **remote code execution**.

An new release of the package is available which fixes this issue. One is strongly encouraged to upgrade to it by using:

```
$ pear upgrade HTML_AJAX-0.5.8
```

This issue is CVE-2017-5677. More details can be found in bug #21165.

Thanks to Egidio Romano who reported this issue.

2nd Feb 2017 10:41pm. Read more or see comments

PEAR server fully restored

The pear.php.net server has been fully restored after we had to witness a fatal hard drive crash on 2015-11-29.

Recent Releases:

Mail_Mime 1.10.2
(Released Fri, 17th Nov 17)
PHP_CodeSniffer 3.1.1
(Released Mon, 16th Oct 17)
PHP_CodeSniffer 3.1.0
(Released Tue, 19th Sep 17)
Crypt_GPG 1.6.2
(Released Sun, 3rd Sep 17)
Net_NNTP 1.5.2
(Released Wed, 30th Aug 17)

Popular Packages*:

PHP_CodeSniffer 3.1.1
(1,295.88)
Mail_Mime 1.10.2
(921.00)
VersionControl_Git 0.5.0
(916.97)
PEAR 1.10.5
(360.39)
Archive_Tar 1.4.3
(154.82)
* downloads per day

Recently Proposed:

HTTP::HTTP_Request
by cequiel
Database::Database_dbQuer y
by cequiel
Web Services::Services_Tune

Indenting and Line Length
(Previous)
Function Calls
(Next)

Control Structures

These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated of them:

```
<?php
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
?>
```

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.
You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

For switch statements:

```
<?php
switch (condition) {
case 1:
```

Package Information: PHP_CodeSniffer

» Summary

PHP_CodeSniffer tokenises PHP, JavaScript and CSS files and detects violations of a defined set of coding standards.

» License

BSD 3-Clause License

» Current Release

1.5.0RC1 (beta) was released on 2013-02-08 (Changelog)

Easy Install

Not sure? Get more info.

```
pear install PHP_CodeSniffer
```

Pyrus Install

Try PEAR2's installer, Pyrus.

```
php pyrus.phar install pear/PHP_CodeSniffer
```

1.4.4 (stable) was released on 2013-02-06 (Changelog)
Development Roadmap

» Bug Summary

- Package Maintenance Rank: **10** of 204 packages with open bugs
- Number of open bugs: **5 (427 total bugs)**
- Average age of open bugs: **43 days**
- Oldest open bug: **132 days**
- Number of open feature requests: **33 (162 total feature requests)**

Report a new bug to PHP_CodeSniffer

» Description

PHP_CodeSniffer is a PHP5 script that tokenises PHP, JavaScript and CSS files to detect violations of a defined coding standard. It is an essential development tool that ensures your code remains clean and consistent. It can also help prevent some common semantic errors made by developers.

» Maintainers

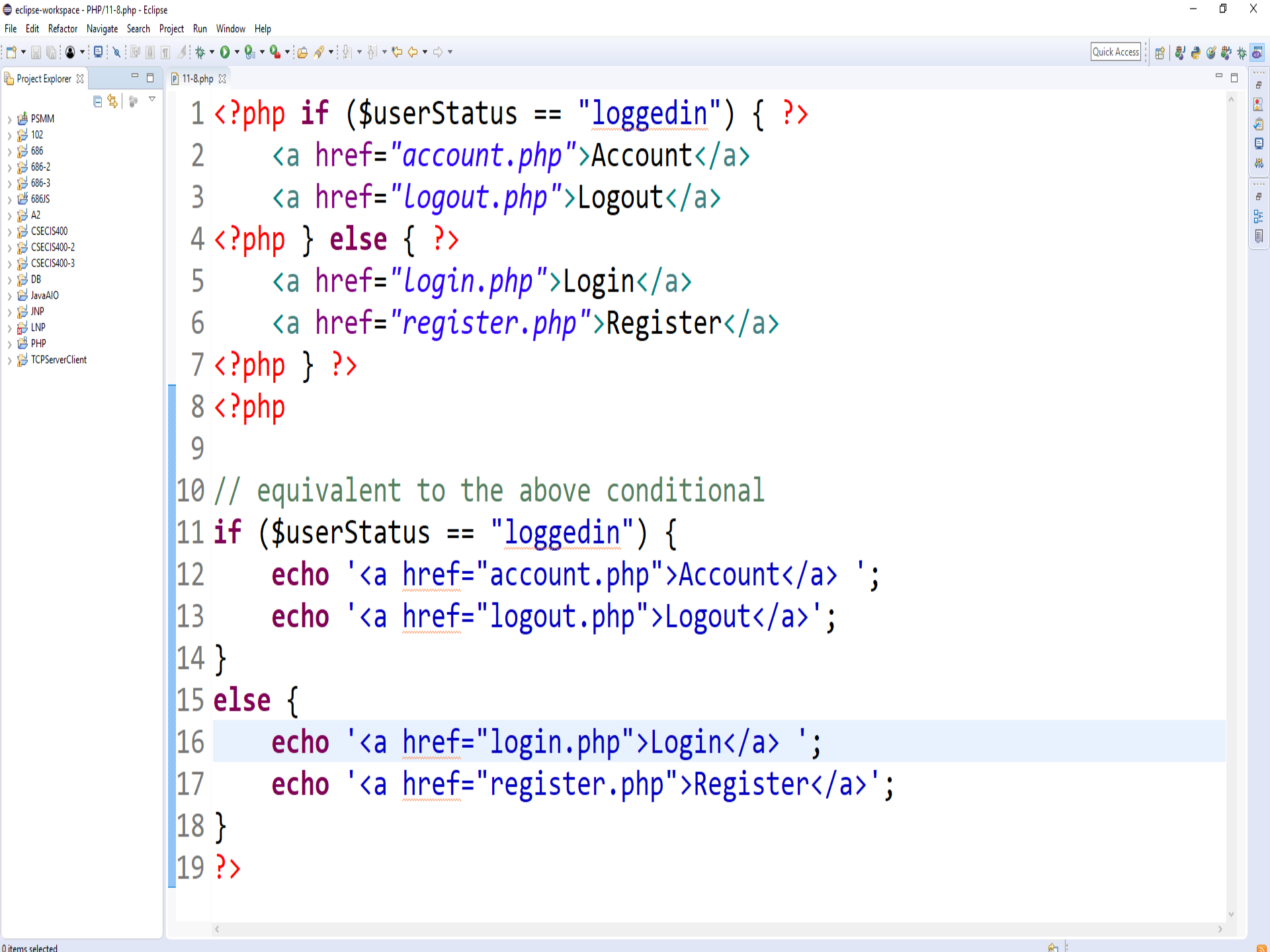
- Greg Sherwood (lead)

» More Information

- External Package Homepage
- Browse the source tree
- RSS release feed

The elseif Clause

```
<?php if ($userStatus == "loggedin") : ?>
        <a href="account.php">Account</a>
        <a href="logout.php">Logout</a>
<?php else : ?>
        <a href="login.php">Login</a>
        <a href="register.php">Register</a>
<?php endif; ?>
```


eclipse-workspace - PHP/11-8.php - Eclipse
File Edit Refactor Navigate Search Project Run Window Help
Quick Access
Project Explorer
PSMM
102
686
686-2
686-3
686JS
A2
CSECIS400
CSECIS400-2
CSECIS400-3
DB
JavaAIO
JNP
LNP
PHP
TCPServerClient
11-8.php
1 <?php if ($userStatus == "loggedin") { ?>
2 Account
3 Logout
4 <?php } else { ?>
5 Login
6 Register
7 <?php } ?>
8 <?php
9
10 // equivalent to the above conditional
11 if ($userStatus == "loggedin") {
12 echo 'Account ';
13 echo 'Logout';
14 }
15 else {
16 echo 'Login ';
17 echo 'Register';
18 }
19 ?>
0 items selected

The switch Statement

```
switch (expression) {
        case result1:
                // execute this if expression results in result1
                break;
        case result2:
                // execute this if expression results in result2
                break;
        default:
                // execute this if no break statement
                // has been encountered hitherto
}
```

testswitch.php

```
<?php
$mood = "sad";
switch ($mood) {
        case "happy":
                echo "Hooray! I'm in a good mood!";
                break;
        case "sad":
                echo "Awww. Don't be down!";
                break;
        default:
                echo "I'm neither happy nor sad, but $mood.";
                break;
}
?>
```

The ?: Operator

- The ?: or **ternary** operator is similar to the if statement, except that it returns a value derived from one of two expressions separated by a colon.
  - This construct provides you with three parts of the whole, hence the name *ternary*.
  - The expression used to generate the returned value depends on the result of a test expression:

(expression) **?** returned_if_expression_is_true **:** returned_if_expression_is_false;

```
<?php
$mood = "sad";
$text = ($mood == "happy") ? "I am in a good mood!" : "I am in a $mood mood.";
echo "$text";
?>
```

testwhile.php

```
while (expression) {
    // do something
}
```

```
<?php
$counter = 1;
while ($counter <= 12) {
    echo $counter." times 2 is ".($counter * 2)."<br>";
    $counter++;
}
?>
```

testdowhile.php

```
do {
    // code to be executed
} while (expression);
```

```
<?php
$num = 1;
do {
    echo "The number is: ".$num."<br>";
    $num++;
} while (($num > 200) && ($num < 400));
?>
```

The for Statement

for (*initialization expression*; *test expression*; *modification expression*) {
    // code to be executed
}

```
<?php
for ($counter=1; $counter<=12; $counter++) {
    echo $counter." times 2 is ".($counter * 2)."<br>";
}
?>
```

The for Statement: 2nd Example

```
<?php
for ($counter=1; $counter <= 10; $counter++) {
        $temp = 4000/$counter;
        echo "4000 divided by ".$counter." is... $temp<br>";
}
?>
```

The break Statement

```
<?php
$counter = -4;
for (; $counter <= 10; $counter++) {
 if ($counter == 0) {
        break;
 } else {
        $temp = 4000/$counter;
        echo  "4000 divided by ".$counter." is... $temp<br>";
 }
}
?>
```

The continue Statement

```
<?php
$counter = -4;
for (; $counter <= 10; $counter++) {
 if ($counter == 0) {
        continue;
 }
 $temp = 4000/$counter;
 echo "4000 divided by ".$counter." is... ".$temp."<br>";
}
?>
```

The continue Statement

- Using the break and continue statements can make code more difficult to read because they often add layers of complexity to the logic of the loop statements that contain them.
  - Use these statements with care, or comment your code to show other programmers (or yourself) exactly what you're trying to achieve with these statements.

Nesting Loops

```
<?php
echo "<table style=\"border: 1px solid #000;\"> \n";
for ($y=1; $y<=12; $y++) {
        echo "<tr> \n";
        for ($x=1; $x<=12; $x++) {
                echo "<td style=\"border: 1px solid #000; width: 25px; text-align:center;\">";
                echo ($x * $y);
                echo "</td> \n";
        }
        echo "</tr> \n";
}
echo "</table>";
?>
```

By convention, PHP files have the .php extension.

example.php

exampleData.inc.php

```php
<?php
$name = 'Randy Connolly';
$email = 'someone@example.com';
?>
```

Files that are included can have any extension, though in this example we are using the extension .inc.php to make it clearer later that this is an include file.

The include function inserts the contents of the specified file.

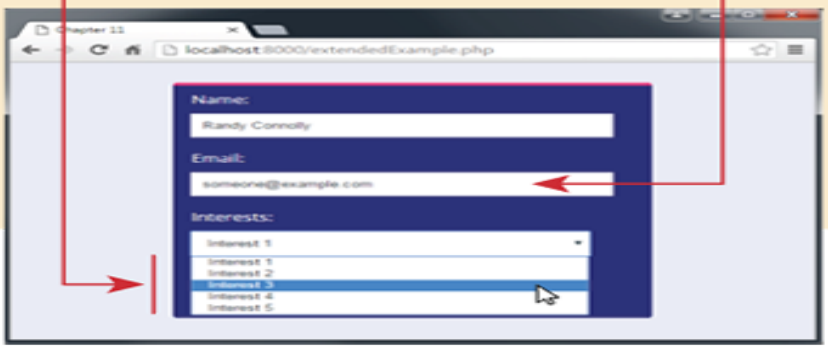
Common practice is to place include statements (and variables used throughout the page) at the top of the page.

```php
<?php
include('exampleData.inc.php');
?>
<!DOCTYPE html>
<html lang="en">
<head>
    ...
</head>
<body>
<form>
  <fieldset>
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" value="<?php echo $name; ?>" >

    <label for="mail">Email:</label>
    <input type="email" id="mail" name="email" value="<?php echo $email; ?>" >

    <label for="interests">Interests:</label>
    <select id="interests" name="interests">
        <?php
        for ($i=0; $i<5; $i++) {
            $count = $i + 1;
            echo "<option>Interest " . $count . "</option>";
        }
        ?>
    </select>
    <button type="submit">
    Contact us
    </button>
  </fieldset>
</form>
</body>
</html>
```

Here we are outputing the contents of the $name variable into the value attribute.

Here we are outputting the contents of the $email variable into the value attribute.

Use a loop to output five <option> elements.



Result in browser.

Functions in PHP

- As usual, functions come in two flavors:
  - Those built in to the language
  - Those you define yourself
- PHP has hundreds of built-in functions, with this format:

  some_function($*an_argument*, $*another_argument*);
- An example:

```
<?php
$num = -321;
$newnum = abs($num);
echo $newnum;
//prints "321"
?>
```

Defining Functions in PHP

- You can define your own functions using the function statement:

```
function some_function($argument1, $argument2)
{
        //function code here
}
```

Defining Functions in PHP

- The naming rules for functions are similar to the naming rules for variables.
  - Names cannot include spaces, and they must begin with a letter or an underscore.
  - As with variables, your function names should be meaningful and consistent in style.
    - The capitalization of function names is one such stylistic touch you can add to your code
    - Using mixed case in names, such as MyFunction() or handleSomeDifficultTask(), makes your code much easier to read.
    - You may hear this naming convention referred to as CamelCase or lower camelCase, depending on whether the first character is capitalized.

bighello.php

```
<?php
function bighello()
{
  echo "<h1>HELLO!</h1>";
}

bighello();
?>
```

printbr.php

```
<?php
function printBR($txt)
{
 echo $txt."<br>";
}
printBR("This is a line.");
printBR("This is a new line.");
printBR("This is yet another line.");
?>
```

Defining Functions in PHP

- Unlike variable names, function names are not case sensitive.

- In printbr.php, printBR() function could have been called as printbr(), PRINTBR(), or any combination thereof, with success.

A Function That Returns a Value

```
<?php
function addNums($firstnum, $secondnum)
{
 $result = $firstnum + $secondnum;
 return $result;
}
echo addNums(3,5);
//will print "8"
?>
```

Setting Default Values for Arguments

```
<?php
function fontWrap($txt, $fontsize)
{
    echo "<span style=\"font-size:$fontsize\">".$txt."</span>";
}

fontWrap("A Heading<br>","24pt");
fontWrap("some body text<br>","16pt");
fontWrap("smaller body text<br>","12pt");
fontWrap("even smaller body text<br>","10pt");
?>
```

A Function with an Optional Argument

```
<?php
function fontWrap($txt, $fontsize = "12pt")
{
    echo "<span style=\"font-size:$fontsize\">".$txt."</span>";
}
fontWrap("A Heading<br>","24pt");
fontWrap("some body text<br>");
fontWrap("smaller body text<br>");
fontWrap("even smaller body text<br>");
?>
```

Passing an Argument to a Function by Value

```
<?php
function addFive($num)
{
      $num += 5;
}
$orignum = 10;
addFive($orignum);
echo $orignum;
?>
```

Passing an Argument to a Function by Reference

```
<?php
function addFive(&$num)
{
      $num += 5;
}
$orignum = 10;
addFive($orignum);
echo $orignum;
?>
```

eclipse-workspace - PHP/11-19.php - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Quick Access

Project Explorer

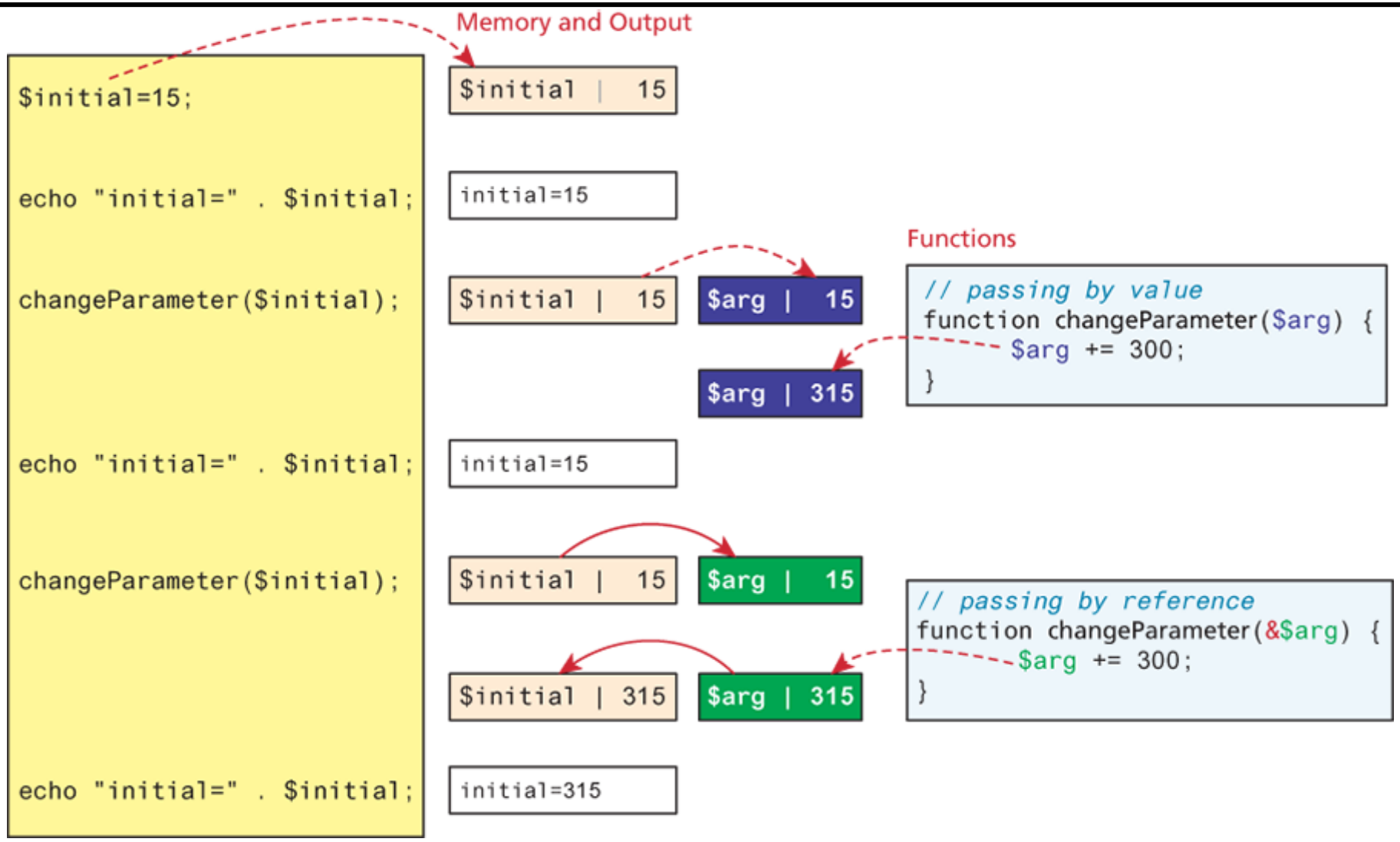
- PSMM
- 102
- 686
- 686-2
- 686-3
- 686JS
- A2
- CSECIS400
- CSECIS400-2
- CSECIS400-3
- DB
- JavaAIO
- JNP
- LNP
- PHP
  - PHP Language Library [PHP 7.2]
  - PHP Include Path
  - 10
  - 11-Forms
  - 12-Sessions
  - 8-Array
  - 9-Class
  - DB
  - FUN
  - IF
  - 11-19.php
  - 11-6.php
  - 11-8.php
  - casttype.php
  - cat.php
  - constant.php
  - first.php
  - helloworld.php
  - myTest.php
  - myTest2.php
  - myTest3.php
  - myTest4.php
  - myTest5.php
  - phpinfo.php
  - settype.php
  - testtype.php
- TCPServerClient

11-8.php | numberedheading.php | numberedheading2.php | addfive.php | addfive2.php | 11-19.php | http://localhost/PHP/11-19.php

```php
<?php

function changeParameter(&$arg) {
    $arg += 300;
    echo "<br/>arg=". $arg;
}

$initial = 15;
echo "<br/>initial=" . $initial; // output: initial=15
changeParameter($initial);   // output: arg=315
echo "<br/>initial=" . $initial; // output: initial=315


?>
```

PHP | Writable | Smart Insert | 14 : 3

Passing an Argument to a Function by Reference



Quiz #1

```
<?php
$number = 50;
function tenTimes() {
  $number = $number * 10;
}
tenTimes();
echo $number;
?>
```

Quiz #2

```
<?php
$number = 50;
function tenTimes() {
global $number;
  $number = $number * 10;
}
tenTimes();
echo $number;
 ?>
```

Quiz #3

```
<?php
$number = 50;
function tenTimes( &$n ) {
$n = $n * 10;
}
tenTimes( $number );
echo $number;
?>
```

Testing for a Function's Existence

```
<?php
function tagWrap($tag, $txt, $func = "")
{
	if ((!empty($txt)) && (function_exists($func))) {
		$txt = $func($txt);
		return "<".$tag.">".$txt."</".$tag."><br/>";
	} else {
		return "<strong>".$txt."</strong><br/>";
	}
}
function underline($txt)
{
	return "<span style=\"text-decoration: underline;\">".$txt."</span>";
}
echo tagWrap('strong', 'make me bold');
echo tagWrap('em', 'underline and italicize me', "underline");
echo tagWrap('em', 'make me italic and quote me', create_function('$txt', 'return
"&quot;$txt&quot;";'));
?>
```

TABLE 5.1 Standard Data Types

Type	Example	Description
Boolean	`true`	One of the special values `true` or `false`
Integer	`5`	A whole number
Float or double	`3.234`	A floating-point number
String	`"hello"`	A collection of characters
Object		An instance of a class
Array		An ordered set of keys and values
Resource		Reference to a third-party resource (a database, for example)
NULL		An uninitialized variable

Arrays

- The following code snippet shows how to create an array called $rainbow using the array() function, containing all its various colors:

```
$rainbow = array(“red”, “orange”, “yellow”, “green”, “blue”, “indigo”, “violet”);
```

- The following snippet shows the same array being created incrementally using the array operator:

```
$rainbow[] = “red”;
$rainbow[] = “orange”;
$rainbow[] = “yellow”;
$rainbow[] = “green”;
$rainbow[] = “blue”;
$rainbow[] = “indigo”;
$rainbow[] = “violet”;
```

Associative Arrays

- While numerically indexed arrays use an index position as the key (0, 1, 2, etc.), associative arrays use actual named keys.

```
$character = array(
    “name” => “Bob”,
    “occupation” => superhero”,
    “age” => 30,
    “special power” => “x-ray vision”
    );

echo $character[‘occupation’];
```

Associative Arrays

- As with numerically indexed arrays, you can use the array operator to add to an associative array:

  $character[‘**supername**’] = “Mega X-Ray Guy”;

- This example adds a key called **supername** with a value of Mega X-Ray Guy.
  - The only difference between an associative array and a numerically indexed array is the key name.
    - In a numerically indexed array, the key name is a number.
    - In an associative array, the key name is a meaningful word.

Multidimensional Arrays

```
<?php
$characters = array(
                array(
                        "name" => "Bob",
                        "occupation" => "superhero",
                        "age" => 30,
                        "special power" => "x-ray vision"
                ),
                array(
                        "name" => "Sally",
                        "occupation" => "superhero",
                        "age" => 24,
                        "special power" => "superhuman strength"
                ),
                array(
                        "name" => "Jane",
                        "occupation" => "arch villain",
                        "age" => 45,
                        "special power" => "nanotechnology"
                )
);
```

Multidimensional Arrays

- If you attempt to print the master elements like:

  echo $characters[1];

  the output will be:

  Array

  because the master element indeed holds an array as its content.

- To really get to the content you want (that is, the specific information found within the inner array element), you need to access the master element index position plus the associative name of the value you want to view.

Multidimensional Arrays

```
echo $characters[1][‘occupation’];
```

prints this:

superhero

- If you add the following lines to the end of the code it prints the information stored in each element, with an added line displayed in the browser :

```
foreach ($characters as $c) {
    while (list($k, $v) = each ($c)) {
        echo "$k ... $v <br>";
    }
    echo "<hr>";
}
```

The Enhanced For Statement in Java

- The for statement also has another form designed for iteration through Collections and arrays.
- This form is sometimes referred to as the enhanced for statement, and can be used to make your loops more compact and easy to read.

```
class EnhancedForDemo {
    public static void main(String[] args){
        int[] numbers = {1,2,3,4,5,6,7,8,9,10};
        for (int item : numbers) {
            System.out.println("Count is: " + item);
        }
    }
}
```

Array-Related Functions

- More than 70 array-related functions are built in to PHP, which you can read about in detail at:

  http://us3.php.net/manual/en/ref.array.php

- Some of the more common (and useful) functions are described briefly in this section:
  - **foreach()** - This control structure (that looks like a function) is used to step through an array, assigning the value of an element to a given variable.
  - **each() and list()**—These functions usually appear together, in the context of stepping through an array and returning its keys and values.

Array-Related Functions

- **count() and sizeof() -** Each of these functions counts the number of elements in an array; they are aliases of each other. Given the following array

  $colors = array(“blue”, “black”, “red”, “green”);

  both **count**($colors) and **sizeof**($colors) return a value of 4.

- **reset() -** This function rewinds the pointer to the beginning of an array, as in this example:

  reset($character);

  This function proves useful when you are performing multiple manipulations on an array, such as sorting, extracting values, and so forth.

Array-Related Functions

- **array_push() -** This function adds one or more elements to the **end** of an existing array, as in this example:

  array_push($existingArray, “element 1”, “element 2”, “element 3”);

- **array_pop() -** This function removes (and returns) the **last** element of an existing array, as in this example:

  $last_element = array_pop($existingArray);

- **array_unshift()** - This function adds one or more elements to the beginning of an existing array, as in this example:

  array_unshift($existingArray, “element 1”, “element 2”, “element 3”);

Array-Related Functions

- **array_shift()**—This function removes (and returns) the first element of an existing array, as in this example, where the value of the element in the first position of $existingArray is assigned to the variable $first_element:

  ```
  $first_element = array_shift($existingArray);
  ```

- **array_merge()**—This function combines two or more existing arrays, as in this example:

  ```
  $newArray = array_merge($array1, $array2);
  ```

- **array_keys()**—This function returns an array containing all the key names within a given array, as in this example:

  ```
  $keysArray = array_keys($existingArray);
  ```

Array-Related Functions

- **array_values()**—This function returns an array containing all the values within a given array, as in this example:

  ```
  $valuesArray = array_values($existingArray);
  ```

- **shuffle()**—This function randomizes the elements of a given array. The syntax of this function is simply as follows:

  ```
  shuffle($existingArray);
  ```

Multidimensional Arrays

- **How many dimensions can multidimensional arrays have?**
  - You can create as many dimensions in your multidimensional array as you can manage, but remember the more dimensions you have, the more you have to manage.
  - If you have data with more than a few dimensions, it might be wise to ask yourself whether that data should be stored differently, such as in a database and accessed that way.

TABLE 5.1 Standard Data Types

Type	Example	Description
Boolean	`true`	One of the special values `true` or `false`
Integer	`5`	A whole number
Float or double	`3.234`	A floating-point number
String	`"hello"`	A collection of characters
Object		An instance of a class
Array		An ordered set of keys and values
Resource		Reference to a third-party resource (a database, for example)
NULL		An uninitialized variable

Objects

```
// proofofclass.php
<?php
class myClass {
   //code will go here
}

$object1 = new myClass(); // constructor
echo "\$object1 is an ".gettype($object1).".<br/>";


if (is_object($object1)) {
   echo "Really! I swear \$object1 is an object!";
}
?>
```

Objects: Properties

- The variables declared inside an object are called **properties**.
- It is standard practice to declare your variables at the top of the class.
- These properties can be values, arrays, or even other objects.
- An example:

```
class myCar {
   public $color = “silver”;
   public $make = “Mazda”;
   public $model = “Protege5”;
}
```

Objects: Properties

- Keywords before the variable name
  - **Public** – the class member (variable) can be accessed everywhere
  - **Protected -** the class member (variable) can be accessed within the class itself or a parent class or an inherited class
  - **Private -** the class member (variable) can be accessed only by the class itself.

How to Access Object Properties

```
<?php
class myCar {
   public $color = "silver";
   public $make = "Mazda";
   public $model = "Protege5";
}

$car = new myCar(); // constructor
echo "I drive a: ".$car->color." ".$car->make." ".$car->model;
?>
```

How to Change Object Properties

```
<?php
class myCar {
   public $color = "silver";
   public $make = "Mazda";
   public $model = "Protege5";
}
$car = new myCar(); // constructor
$car -> color = "red";
$car -> make = "Porsche";
$car -> model = "Boxter";
echo "I drive a: ".$car -> color." ".$car -> make." ".$car -> model;
?>
```

Object Methods

```
<?php
class myClass {
  function sayHello() {
        echo "HELLO!";
  }
}
$object1 = new myClass(); // constructor
$object1 -> sayHello();
?>
```

Accessing Class Properties Within a Method

```
<?php
class myClass {
    public $name = "Jimbo";
    function sayHello() {
            echo "HELLO! My name is ".$this->name;
    }
}
$object1 = new myClass(); // constructor
$object1 -> sayHello();
?>
```

Changing the Value of a Property from Within a Method

```
<?php
class myClass {
  public $name = "Jimbo";
  function setName($n) {
        $this->name = $n;
  }
  function sayHello() {
        echo "HELLO! My name is ".$this->name;
  }
}
$object1 = new myClass();
$object1 -> setName("Julie");
$object1 -> sayHello();
?>
```

helloclass3.php

Inheritance

```
<?php
class myClass {
    public $name = "Matt";
    function myClass($n) {
            $this->name = $n;
    }
    function sayHello() {
            echo "HELLO! My name is ".$this->name;
    }
}
class childClass extends myClass {
    //code goes here
}
$object1 = new childClass("Baby Matt");
$object1 -> sayHello();
?>
```

inheritance.php

Overriding

```
<?php
class myClass {
   public $name = "Matt";
   function myClass($n) {
                $this->name = $n;
   }
   function sayHello() {
                echo "HELLO! My name is ".$this->name;
   }
}
class childClass extends myClass {
   function sayHello() {
                echo "I will not tell you my name.";
   }
}
$object1 = new childClass("Baby Matt");
$object1 -> sayHello();
?>
```

inheritance2.php

TABLE 5.1 Standard Data Types

Type	Example	Description
Boolean	`true`	One of the special values `true` or `false`
Integer	`5`	A whole number
Float or double	`3.234`	A floating-point number
String	`"hello"`	A collection of characters
Object		An instance of a class
Array		An ordered set of keys and values
Resource		Reference to a third-party resource (a database, for example)
NULL		An uninitialized variable

Formatting Strings

- PHP provides two functions that enable you first to apply formatting to strings:
  - printf()
  - sprintf()

Quiz

- What conversion specifier would you use with printf() to format an integer as a double?

TABLE 10.1 Type Specifiers

Specifier	Description
d	Display argument as a decimal number
b	Display an integer as a binary number
c	Display an integer as ASCII equivalent
f	Display an integer as a floating-point number (double)
o	Display an integer as an octal number (base 8)
s	Display argument as a string
x	Display an integer as a lowercase hexadecimal number (base 16)
X	Display an integer as an uppercase hexadecimal number (base 16)

printf(): An example

```
<?php
$number = 543;
printf("Decimal: %d<br/>", $number);
printf("Binary: %b<br/>", $number);
printf("ASCII: %c<br/>", $number);
printf("Double: %f<br/>", $number);
printf("Octal: %o<br/>", $number);
printf("String: %s<br/>", $number);
printf("Hex (lower): %x<br/>", $number);
printf("Hex (upper): %X<br/>", $number);
?>
```

Storing a Formatted String: sprintf()

- The printf() function outputs data to the browser. If you want to save the results, use **sprintf()**. It used just like printf(), except that it returns a string that can be stored in a variable for later use:

```
<?php
$cash = sprintf(“%.2f”, 21.334454);
echo “You have \$$cash to spend.”;
// Prints “You have $21.33 to spend.”
?>
```

Common String Operations: strlen()

- strlen($string): finds the length of a string

```
<?php
$membership = “pAB7”;
if (strlen($membership) == 4) {
        echo “<p>Thank you!</p>”;
} else {
        echo “<p>Your membership number must be four characters long.</p>”;
}
?>
```

Common String Operations: strstr()

- strstr($string, $sub-string): finds a substring within a string

```
<?php
$membership = “pAB7”;
if (strstr($membership, “AB”)) {
   echo “<p>Your membership expires soon!</p>”;
} else {
   echo “<p>Thank you!</p>”;
}
?>
```

Common String Operations: strpos()

- strpos($string, $sub-string): finds the position of a substring

```
<?php
$membership = “mz00xyz”;
if (strpos($membership, “mz”)  ==  0) {
        echo “Hello mz!”;
}
?>
```

Common String Operations: substr()

- substr($string, start_pos, [length]): extracts part of a string

```
<?php
$test = “phpcoder”;
echo substr($test, 3).”<br/>”; // prints “coder”
echo substr($test, 3, 2).”<br/>”; // prints “co”
?>
```

strtok()

- You can parse a string word by word using the strtok() function.
  - This function requires two arguments:
    - the **string** to tokenize
    - the **delimiters** by which to split the string, which can include as many characters as you want, and the function will return the first token found.
  - After strtok() has been called for the first time, the source string is **cached**—for subsequent calls, you should pass only the delimiter string to the strtok() function.
  - The function returns the next found token every time it is called, returning **false** when the end of the string is reached.

XAMPP for Windows

strtok

(PHP 4, PHP 5)

strtok — Tokenize string

Description

```
string strtok ( string $str , string $token )
```

```
string strtok ( string $token )
```

strtok() splits a string (**str**) into smaller strings (tokens), with each token being delimited by any character from **token**. That is, if you have a string like "This is an example string" you could tokenize this string into its individual words by using the space character as the token.

Note that only the first call to strtok uses the string argument. Every subsequent call to

Tokenizing a String with strtok()

```
<?php
$test  = "http://www.google.com/search?";
$test .= "hl=en&ie=UTF-8&q=php+development+books&btnG=Google+Search";
$delims = "?&";
$word = strtok($test, $delims);
while (is_string($word)) {
        if ($word) {
                echo $word."<br/>";
        }
        $word = strtok($delims);
}
?>
```

Calculate the Similarity Between 2 Strings

```
<?php
$str1 = “Edmund Yu”;
$str2 = “Edmund”;
$str3 = “Yu”;
$str4 = “Ed Yu”;

echo "The similarity between ".$str1." and ".$str2." is: ".similar_text($str1, $str2);
echo "<br/>";
echo "The similarity between ".$str1." and ".$str3." is: ".similar_text($str1, $str3);
echo "<br/>";
echo "The similarity between ".$str1." and ".$str4." is: ".similar_text($str1, $str4);
?>
```

XAMPP for Windows

similar_text

(PHP 4, PHP 5)

similar_text — Calculate the similarity between two strings

Description

```
int similar_text ( string $first , string $second [, float &$percent ] )
```

This calculates the similarity between two strings as described in Programming Classics: Implementing the World's Best Algorithms by Oliver (ISBN 0-131-00413-1). Note that this implementation does not use a stack as in Oliver's pseudo code, but recursive calls which may or may not speed up the whole process. Note also that the complexity of this algorithm is O(N**3) where N is the length of the longest string.

Parameters

first

The first string.

second

The second string.

percent

By passing a reference as third argument, **similar_text()** will calculate the similarity in percent for you.

Return Values

Calculate the Similarity Between 2 Strings

```
<?php
$str1 = “Book";
$str2 = “Nook";

similar_text($str1, $str2, $percent);
echo "The similarity between ".$str1." and ".$str2." is: ".$percent."%";
echo "<br/>";
echo "The levenshtein/edit distance between ".$str1." and ".$str2." is:
".levenshtein($str1, $str2);
?>
```

XAMPP for Windows

levenshtein

(PHP 4 >= 4.0.1, PHP 5)

levenshtein — Calculate Levenshtein distance between two strings

Description

```
int levenshtein ( string $str1 , string $str2 )
```

```
int levenshtein ( string $str1 , string $str2 , int $cost_ins , int $cost_rep , int $cost_del )
```

The Levenshtein distance is defined as the minimal number of characters you have to replace, insert or delete to transform **str1** into **str2**. The complexity of the algorithm is *O(m*n)*, where *n* and *m* are the length of **str1** and **str2** (rather good when compared to similar_text(), which is O(max(n,m)**3), but still expensive).

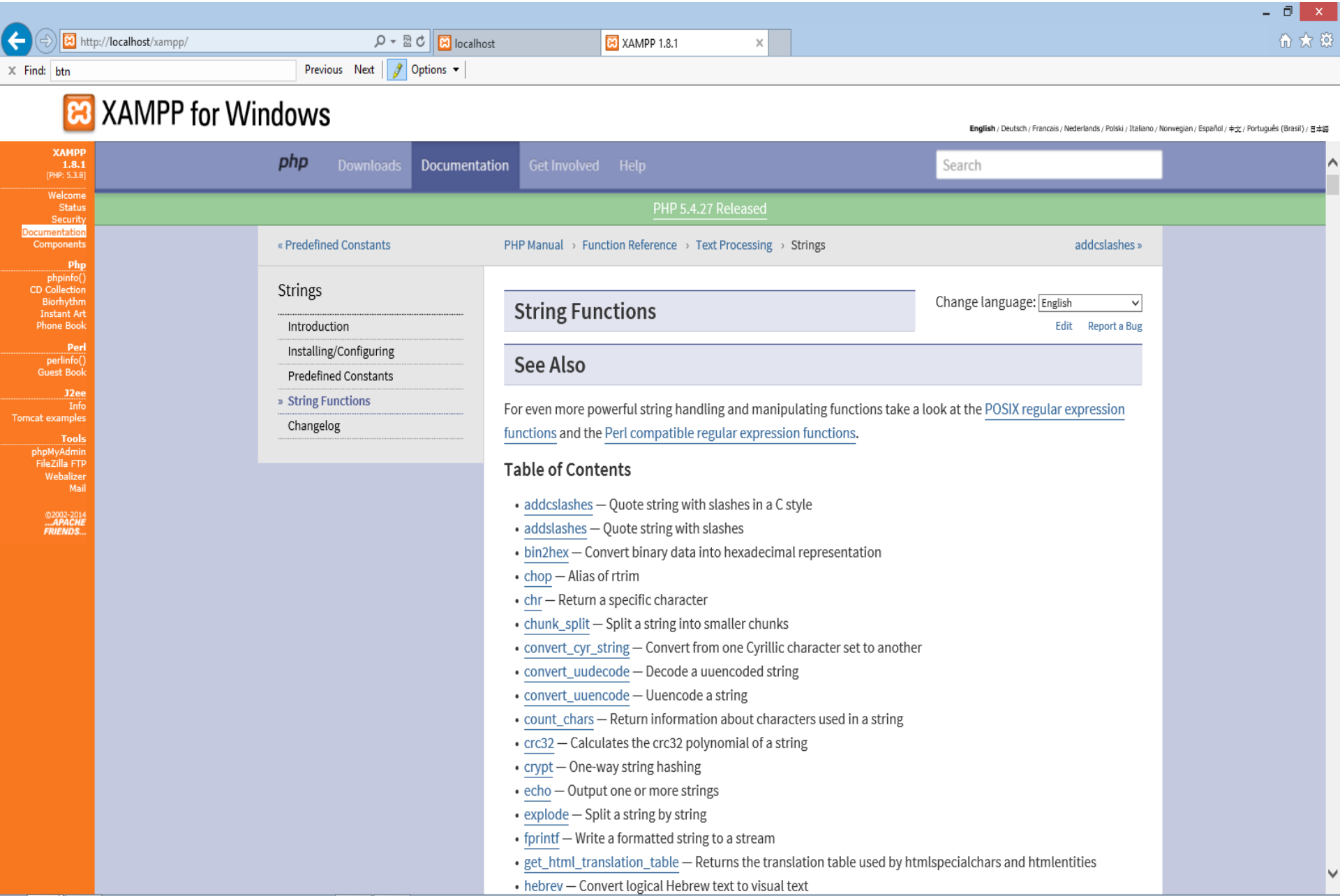
In its simplest form the function will take only the two strings as parameter and will calculate just the number of insert, replace and delete operations needed to transform **str1** into **str2**.

A second variant will take three additional parameters that define the cost of insert, replace and delete operations. This is more general and adaptive than variant one, but not as efficient.

Parameters

str1

More String Functions



Getting a Date with time()

- PHP's **time()** function gives you all the information you need about the current date and time.
- It requires no arguments and returns an integer.

```
echo time();
// sample output: 1364991914
// this represents April 3, 2013 at 08:25PM
```

- The integer returned by time() represents the number of seconds elapsed since midnight GMT on January 1, 1970.
- This moment is known as the **UNIX epoch**, and the number of seconds that have elapsed since then is referred to as a **timestamp**.

XAMPP for Windows

getdate

Change language: English

Edit Report a Bug

(PHP 4, PHP 5)

getdate — Get date/time information

Description

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

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

Parameters

`timestamp`

The optional `timestamp` parameter is an integer Unix timestamp that defaults to the current local time if a `timestamp` is not given. In other words, it defaults to the value of time().

Return Values

Returns an associative array of information related to the `timestamp`. Elements from the returned associative array are as follows:

Key elements of the returned associative array

Converting a Timestamp with getdate()

- Now that you have a timestamp to work with, you must convert it before you present it to the user.
  - **getdate()** accepts a timestamp and returns an associative array containing information about the date.
  - If you omit the timestamp, getdate() works with the current timestamp as returned by time().
  - The following table lists the elements contained in the array returned by getdate().

TABLE 10.3 The Associative Array Returned by `getdate()`

Key	Description	Example
`seconds`	Seconds past the minute (0–59)	53
`minutes`	Minutes past the hour (0–59)	44
`hours`	Hours of the day (0–23)	17
`mday`	Day of the month (1–31)	3
`wday`	Day of the week (0–6)	0
`mon`	Month of the year (1–12)	2
`year`	Year (four digits)	2008
`yday`	Day of year (0–365)	33
`weekday`	Day of the week (name)	Sunday
`month`	Month of the year (name)	February
`0`	Timestamp	1202082293

Converting a Timestamp with getdate()

```
<?php
$date_array = getdate(); // no argument passed so today's date will be used
foreach ($date_array as $key => $val) {
        echo "$key = $val<br/>";
}
?>

<hr/>

<?php
echo "<p>Today's date:
".$date_array['mon']."/".$date_array['mday']."/".$date_array['year']."</p>";
?>
```

A Simple HTML Form

```
<!DOCTYPE html>
<html>
<head>
<title>A simple HTML form</title>
</head>
<body>
<form method="post" action="send_simpleform.php">
  <p><label for="user">Name:</label><br/>
  <input type="text" id="user" name="user"></p>
  <p><label for="message">Message:</label><br/>
  <textarea id="message" name="message" rows="5" cols="40"></textarea></p>
  <button type="submit" name="submit" value="send">Send Message</button>
</form>
</body>
</html>
```

send_simpleform.php

```
<!DOCTYPE html>
<html>
<head>
<title>A simple response</title>
</head>
<body>
<p>Welcome, <strong><?php echo $_POST['user']; ?></strong>!</p>
<p>Your message is:
<strong><?php echo $_POST['message']; ?></strong></p>
</body>
</html>
```

Superglobals: $_POST

- In addition to global variables of your own creation, PHP has several predefined variables called **superglobals**. These variables are always present, and their values are available to all your scripts. Each of the following superglobals is actually an array of other variables:

$_GET contains any variables provided to a script through the GET method.

$_POST contains any variables provided to a script through the POST method.

$_COOKIE contains any variables provided to a script through a cookie.

$_FILES contains any variables provided to a script through file uploads.

$_SERVER contains information such as headers, file paths, and script locations.

$_ENV contains any variables provided to a script as part of the server environment.

$_REQUEST contains any variables provided to a script via GET/POST/COOKIE.

$_SESSION contains any variables that are currently registered in a session.

$_POST

$HTTP_POST_VARS [deprecated]

(PHP 4 >= 4.1.0, PHP 5)

$_POST -- $HTTP_POST_VARS [deprecated] — HTTP POST variables

Description

Report a bug

An associative array of variables passed to the current script via the HTTP POST method.

$HTTP_POST_VARS contains the same initial information, but is not a superglobal. (Note that *$HTTP_POST_VARS* and *$_POST* are different variables and that PHP handles them as such)

Changelog

Report a bug

Version	Description
4.1.0	Introduced *$_POST* that deprecated *$HTTP_POST_VARS*.

Examples

Report a bug

An HTML Form with Check Boxes, pt. 1

```
<!DOCTYPE html>
<html>
<head>
<title>An HTML form with checkboxes</title>
</head>
<body>
<form action="send_formwithcb.php" method="POST">
<p><label for="name">Name:</label><br/>
<input type="text" id="name" name="user" /></p>
```

An HTML Form with Check Boxes, pt. 2

```
<fieldset>
<legend>Select Some Products:</legend><br/>
<input type="checkbox" id="Apples"
      name="products[]" value="Apples">
      <label for="Apples">Apples</label><br/>

<input type="checkbox" id="Oranges"
      name="products[]" value="Oranges">
      <label for="Oranges">Oranges</label><br/>

<input type="checkbox" id="Bananas"
      name="products[]" value="Bananas">
      <label for="Bananas">Bananas</label>
</fieldset>
```

An HTML Form with Check Boxes, pt. 3

```
<button type="submit" name="submit" value="submit">Submit Form</button>
</form>
</body>
</html>
```

send_formwithcb.php, part 1

```
<!DOCTYPE html>
<html>
<head>
<title>Reading checkboxes</title>
</head>
<body>
<p>Welcome, <strong><?php echo $_POST['user']; ?></strong>!</p>
```

send_formwithcb.php, part 2

```
<p>Your product choices are:
<?php
if (!empty($_POST['products'])) {
        echo "<ul>";
        foreach ($_POST['products'] as $value) {
                echo "<li>$value</li>";
        }
        echo "</ul>";
} else {
        echo "None";
}
?>
</body>
</html>
```

The <ul> & Tags

- The <ul> tag defines an unordered (bulleted) list.
  - Use the <ul> tag together with the <li> tag to create unordered lists.

- The <li> tag defines a list item.
  - The <li> tag can be used in ordered lists(<ol>) and unordered lists (<ul>).

A Number Guessing Game, part 1

```
<!DOCTYPE html>
<html>
<head>
<title>A PHP number guessing script</title>
</head>
<body>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<p><label for="guess">Type your guess here:</label><br/>
<input type="text" id="guess" name="guess" /></p>
<button type="submit" name="submit" value="submit">Submit</button>
</form>
</body>
</html>
```

Superglobals: $_SERVER

- In addition to global variables of your own creation, PHP has several predefined variables called **superglobals**. These variables are always present, and their values are available to all your scripts. Each of the following superglobals is actually an array of other variables:

$_GET contains any variables provided to a script through the GET method.

$_POST contains any variables provided to a script through the POST method.

$_COOKIE contains any variables provided to a script through a cookie.

$_FILES contains any variables provided to a script through file uploads.

$_SERVER contains information such as headers, file paths, and script locations.

$_ENV contains any variables provided to a script as part of the server environment.

$_REQUEST contains any variables provided to a script via GET/POST/COOKIE.

$_SESSION contains any variables that are currently registered in a session.

'*PHP_SELF*'

The filename of the currently executing script, relative to the document root. For instance, *$_SERVER['PHP_SELF']* in a script at the address *http://example.com/test.php/foo.bar* would be */test.php/foo.bar*. The __FILE__ constant contains the full path and filename of the current (i.e. included) file. If PHP is running as a command-line processor this variable contains the script name since PHP 4.3.0. Previously it was not available.

'argv'

Array of arguments passed to the script. When the script is run on the command line, this gives C-style access to the command line parameters. When called via the GET method, this will contain the query string.

'argc'

Contains the number of command line parameters passed to the script (if run on the command line).

'*GATEWAY_INTERFACE*'

What revision of the CGI specification the server is using; i.e. '***CGI/1.1***'.

'*SERVER_ADDR*'

The IP address of the server under which the current script is executing.

'*SERVER_NAME*'

A Number Guessing Game, Part 2

```
<?php
$num_to_guess = 42;
if (!isset($_POST['guess'])) {
    $message = "Welcome to the guessing machine!";
} elseif (!is_numeric($_POST['guess'])) { // is not numeric
     $message = "I don't understand that response.";
} elseif ($_POST['guess'] == $num_to_guess) { // matches!
     $message = "Well done!";
} elseif ($_POST['guess'] > $num_to_guess) {
    $message = $_POST['guess']." is too big! Try a smaller number.";
} elseif ($_POST['guess'] < $num_to_guess) {
     $message = $_POST['guess']." is too small! Try a larger number.";
} else  { // some other condition
     $message = "I am terribly confused.";
}
?>
```

A Number Guessing Game, Part 1 Revised

```
<!DOCTYPE html>
<html>
<head>
<title>A PHP number guessing script</title>
</head>
<body>
<h1><?php echo $message; ?></h1>
<form action="<?php echo $_SERVER['PHP_SELF']; ?>" method="POST">
<p><label for="guess">Type your guess here:</label><br/>
<input type="text" id="guess" name="guess" /></p>
<button type="submit" name="submit" value="submit">Submit</button>
</form>
</body>
</html>
```

Cookies and Sessions

- PHP contains numerous functions for managing and keeping track of user information, including both simple **cookies** and all-encompassing user **sessions**.
  - You can use cookies within your PHP scripts to store small bits of information about a user.
  - A cookie is a small amount of data stored by the user's browser in compliance with a request from a server or script.
  - A single host can request that up to 20 cookies to be stored by a user's browser. The size of an individual cookie is limited to 4KB.
  - After a cookie is set, only the originating host can read the data.
    - Users can configure their browser to notify them upon receipt of all cookies, or even to refuse all cookie requests.

Setting and Printing a Cookie Value

```
<?php
setcookie("vegetable", "artichoke", time()+3600, "/", “.yourdomain.com", 0);

if (isset($_COOKIE['vegetable'])) {
        echo "<p>Hello again! You have chosen: ".$_COOKIE['vegetable']."</p>";
} else {
        echo "<p>Hello. This may be your first visit.</p>";
}
?>
```

Network Functions

setcookie

Change language: English

Edit Report a Bug

(PHP 4, PHP 5)

setcookie — Send a cookie

Description

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

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

Once the cookies have been set, they can be accessed on the next page load with the `$_COOKIE` or `$HTTP_COOKIE_VARS` arrays. Note, superglobals such as `$_COOKIE` became available in PHP 4.1.0. Cookie values also exist in `$_REQUEST`.

Parameters

All the arguments except the **name** argument are optional. You may also replace an argument with an empty string ("") in order to skip that argument. Because the **expire** argument is integer, it cannot be skipped with an empty string, use a zero (*0*) instead.

» RFC 6265 provides the normative reference on how each **setcookie()** parameter is interpreted.

name

Setting and Printing a Cookie Value

setcookie("vegetable", "artichoke", time()+3600, "/", ".yourdomain.com", 0);

- The cookie name is set to "vegetable" on line 2, and the cookie value to "artichoke".
- The time() function gets the current timestamp and adds 3600 to it (3,600 seconds in an hour). This total represents the expiration date.
- The code defines a path of "/", which means that a cookie should be sent for any page within this server environment.
- The domain argument is set to ".yourdomain.com" (you should make the change relevant to your own domain, or leave it blank if you are working on **localhost**).
- Finally, the code passes 0 to setcookie(), signaling that cookies can be sent in an unsecure environment.

Accessing Cookies

- A PHP script has access to the cookie in the environment variable HTTP_COOKIE or as part of the $_COOKIE superglobal variable, which you may access three different ways:

```
echo $_SERVER['HTTP_COOKIE']; // will print "vegetable=artichoke"

echo getenv('HTTP_COOKIE'); // will print "vegetable=artichoke"

echo $_COOKIE['vegetable']; // will print "artichoke"
```

Superglobals: $_COOKIE

- In addition to global variables of your own creation, PHP has several predefined variables called **superglobals**. These variables are always present, and their values are available to all your scripts. Each of the following superglobals is actually an array of other variables:

$_GET contains any variables provided to a script through the GET method.

$_POST contains any variables provided to a script through the POST method.

$_COOKIE contains any variables provided to a script through a cookie.

$_FILES contains any variables provided to a script through file uploads.

$_SERVER contains information such as headers, file paths, and script locations.

$_ENV contains any variables provided to a script as part of the server environment.

$_REQUEST contains any variables provided to a script via GET/POST/COOKIE.

$_SESSION contains any variables that are currently registered in a session.

Deleting a Cookie

- To delete a cookie, you could call setcookie() with the name argument only:

```
setcookie("vegetable");
```

- This approach does not always work, so you should not rely on it.
- Instead, to delete a cookie, it is safest to set the cookie with a date that you are sure has already expired:

```
setcookie("vegetable", "", time()-60, "/", ".yourdomain.com", 0);
```

- Also make sure that you pass setcookie() the same path, domain, and secure parameters as you did when originally setting the cookie.

The Anatomy of a Cookie

- A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK
Date: Wed, 9 Apr 2014 10:50:58 GMT
Server: Apache/2.2.21 (Unix) PHP/5.4.0
Set-Cookie: vegetable=artichoke; path=/;domain=.yourdomain.com
Connection: close
Content-Type: text/html
```

The Anatomy of a Cookie

- A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK
Date: Wed, 9 Apr 2014 10:50:58 GMT
Server: Apache/2.2.21 (Unix) PHP/5.4.0
Set-Cookie: vegetable=artichoke; path=/;domain=.yourdomain.com
Connection: close
Content-Type: text/html
```

What HTTP feature is that?

Keep Alive

- Some HTTP/1.0 implementations supported a **Keep-Alive** request header
- The Keep-Alive header was sent if the client was capable of using the open connection.
- Reuse of a connection is a standard feature in HTTP/1.1.
- A client can specify in a request header that it wants its connection closed after the first response
- A server can indicate in a response header that it will be closing the connection once it has completed its current response.

Requests [edit]

Field name	Description	Example
Accept	Content-Types that are acceptable for the response	`Accept: text/plain`
Accept-Charset	Character sets that are acceptable	`Accept-Charset: utf-8`
Accept-Encoding	Acceptable encodings. See HTTP compression.	`Accept-Encoding: gzip, deflate`
Accept-Language	Acceptable human languages for response	`Accept-Language: en-US`
Accept-Datetime	Acceptable version in time	`Accept-Datetime: Thu, 31 May 2007 20:35:00 GMT`
Authorization	Authentication credentials for HTTP authentication	`Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`
Cache-Control	Used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain	`Cache-Control: no-cache`
Connection	What type of connection the user-agent would prefer	`Connection: keep-alive`
Cookie	an HTTP cookie previously sent by the server with `Set-Cookie` (below)	`Cookie: $Version=1; Skin=new;`
Content-Length	The length of the request body in octets (8-bit bytes)	`Content-Length: 348`
Content-MD5	A Base64-encoded binary MD5 sum of the	

Responses [edit]

Field name	Description	Example
Access-Control-Allow-Origin	Specifying which web sites can participate in cross-origin resource sharing	`Access-Control-Allow-Origin: *`
Accept-Ranges	What partial content range types this server supports	`Accept-Ranges: bytes`
Age	The age the object has been in a proxy cache in seconds	`Age: 12`
Allow	Valid actions for a specified resource. To be used for a *405 Method not allowed*	`Allow: GET, HEAD`
Cache-Control	Tells all caching mechanisms from server to client whether they may cache this object. It is measured in seconds	`Cache-Control: max-age=3600`
Connection	Options that are desired for the connection[21]	`Connection: close`
Content-Encoding	The type of encoding used on the data. See HTTP compression.	`Content-Encoding: gzip`
Content-Language	The language the content is in	`Content-Language: da`
	The length of the response body in	

Cookies: A Word of Caution

- If your application relies heavily on cookies and users have cookies disabled, your application won't work.
  - However, you can do your part to warn users that cookies are coming, by announcing your intention to use cookies, and also by checking that cookies are enabled before doing anything "important" with your application.

Sessions

- Session functions provide a unique identifier to a user, which can then be used to store and acquire information linked to that ID.

- When a visitor accesses a session-enabled page, either a new identifier is allocated or the user is re-associated with one that was already established in a previous visit.

- Any variables that have been associated with the session become available to your code through the $_SESSION superglobal.

Superglobals: $_SESSION

- In addition to global variables of your own creation, PHP has several predefined variables called **superglobals**. These variables are always present, and their values are available to all your scripts. Each of the following superglobals is actually an array of other variables:

$_GET contains any variables provided to a script through the GET method.

$_POST contains any variables provided to a script through the POST method.

$_COOKIE contains any variables provided to a script through a cookie.

$_FILES contains any variables provided to a script through file uploads.

$_SERVER contains information such as headers, file paths, and script locations.

$_ENV contains any variables provided to a script as part of the server environment.

$_REQUEST contains any variables provided to a script via GET/POST/COOKIE.

$_SESSION contains any variables that are currently registered in a session.

Starting or Resuming a Session

```
<?php
session_start();
echo "<p>Your session ID is ".session_id().".</p>";
?>
```

«session_set_save_handler session_status»

view this page in Brazilian Portuguese [edit] Last updated: Fri, 29 Mar 2013

session_start

(PHP 4, PHP 5)

session_start — Start new or resume existing session

Description

Report a bug

bool **session_start** (void)

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

When **session_start()** is called or when a session auto starts, PHP will call the open and read session save handlers. These will either be a built-in save handler provided by default or by PHP extensions (such as SQLite or Memcached); or can be custom handler as defined by session_set_save_handler(). The read callback will retrieve any existing session data (stored in a special serialized format) and will be unserialized and used to automatically populate the $_SESSION superglobal when the read callback returns the saved session data back to PHP session handling.

To use a named session, call session_name() before calling **session_start()**.

When session.use_trans_sid is enabled, the **session_start()** function will register an internal output handler for URL rewriting.

If a user uses ***ob_gzhandler*** or similar with ob_start(), the function order is important for proper output. For example, ***ob_gzhandler*** must be registered before starting the session.

«session_get_cookie_params session_is_registered»

view this page in Brazilian Portuguese

[edit] Last updated: Fri, 29 Mar 2013

session_id

(PHP 4, PHP 5)

session_id — Get and/or set the current session id

Description

Report a bug

string **session_id** ([string *$id*])

session_id() is used to get or set the session id for the current session.

The constant `SID` can also be used to retrieve the current name and session id as a string suitable for adding to URLs. See also Session handling.

Parameters

Report a bug

id

Storing Variables in a Session

```
<?php
session_start();
$_SESSION['product1'] = "Apples
$_SESSION['product2'] = "Oranges";
echo "The products have been registered.";
?>
```

Accessing Variables in a Session

```
<?php
session_start();
?>
<p>Your chosen products are:</p>
<ul>
<li><?php echo $_SESSION['product1']; ?></li>
<li><?php echo $_SESSION['product2']; ?></li>
</ul>
```

Destroying Sessions

- You can use **session_destroy**() to end a session, erasing all session variables.
- The session_destroy() function requires no arguments.
- You should have an established session for this function to work as expected.
- The following code fragment resumes a session and abruptly destroys it:

```
session_start();
session_destroy();
```

Destroying Sessions

```
<?php
session_start();
echo "<p>Your session ID is ".session_id().".</p>";
session_destroy();
echo "<p>Your session ID is ".session_id().".</p>";
?>
```

«session_decode session_encode»

view this page in Brazilian Portuguese [edit] Last updated: Fri, 29 Mar 2013

session_destroy

(PHP 4, PHP 5)

session_destroy — Destroys all data registered to a session

Description

Report a bug

bool **session_destroy** (void)

session_destroy() destroys all of the data associated with the current session. It does not unset any of the global variables associated with the session, or unset the session cookie. To use the session variables again, session_start() has to be called.

In order to kill the session altogether, like to log the user out, the session id must also be unset. If a cookie is used to propagate the session id (default behavior), then the session cookie must be deleted. setcookie() may be used for that.

Return Values

Report a bug

Files: Creation & Deletion

- If a file does not yet exist, you can create it with the touch() function:

  touch('myfile.txt');

- Given a string representing a file path, touch() attempts to create an empty file of that name.

- If the file already exists, its contents is not disturbed, but the modification date is updated to reflect the time at which the function executed.

- You can remove an existing file with the unlink() function:

  unlink('myfile.txt');

- As did the touch() function, unlink() accepts a file path.

Opening Files: fopen()

- Before you can work with a file, you must first open it for reading or writing, or for performing both tasks.
- PHP provides the fopen() function for doing so. The **fopen()** function returns a file resource you use later to work with the open file:
  - To open a file for reading, you use the following:

    $fp = fopen(“test.txt”, “r”);
  - You use the following to open a file for writing:

    $fp = fopen(“test.txt”, “w”);
  - To open a file for *appending* (that is, to add data to the end of a file), you use this:

    $fp = fopen(“test.txt”, “a”);

Reading Files: fgets()

- When you open a file for reading, you might want to access it line by line. To read a line from an open file, you can use the **fgets**() function.
  - It requires the file resource returned from fopen() as its argument, and an integer as a second argument, which specifies the number of bytes that the function should read if it doesn't first encounter a line end or the end of the file.
  - The fgets() function reads the file until it reaches a newline character ("\n"), the number of bytes specified in the length argument, or the end of the file, whichever comes first:

```
$line = fgets($fp, 1024); // where $fp is the file resource returned by fopen()
```

Writing Files: fwrite() & fputs()

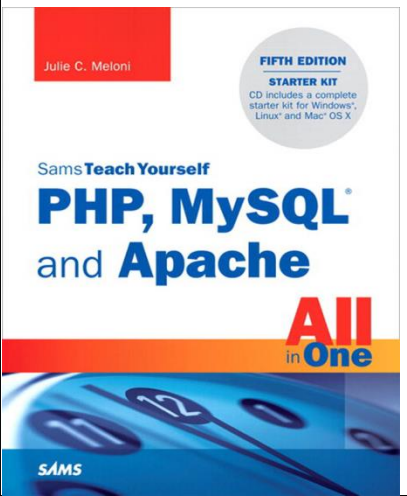
- The fwrite() function accepts a file resource and a string, and then writes the string to the file.
- The fputs() function works in exactly the same way:

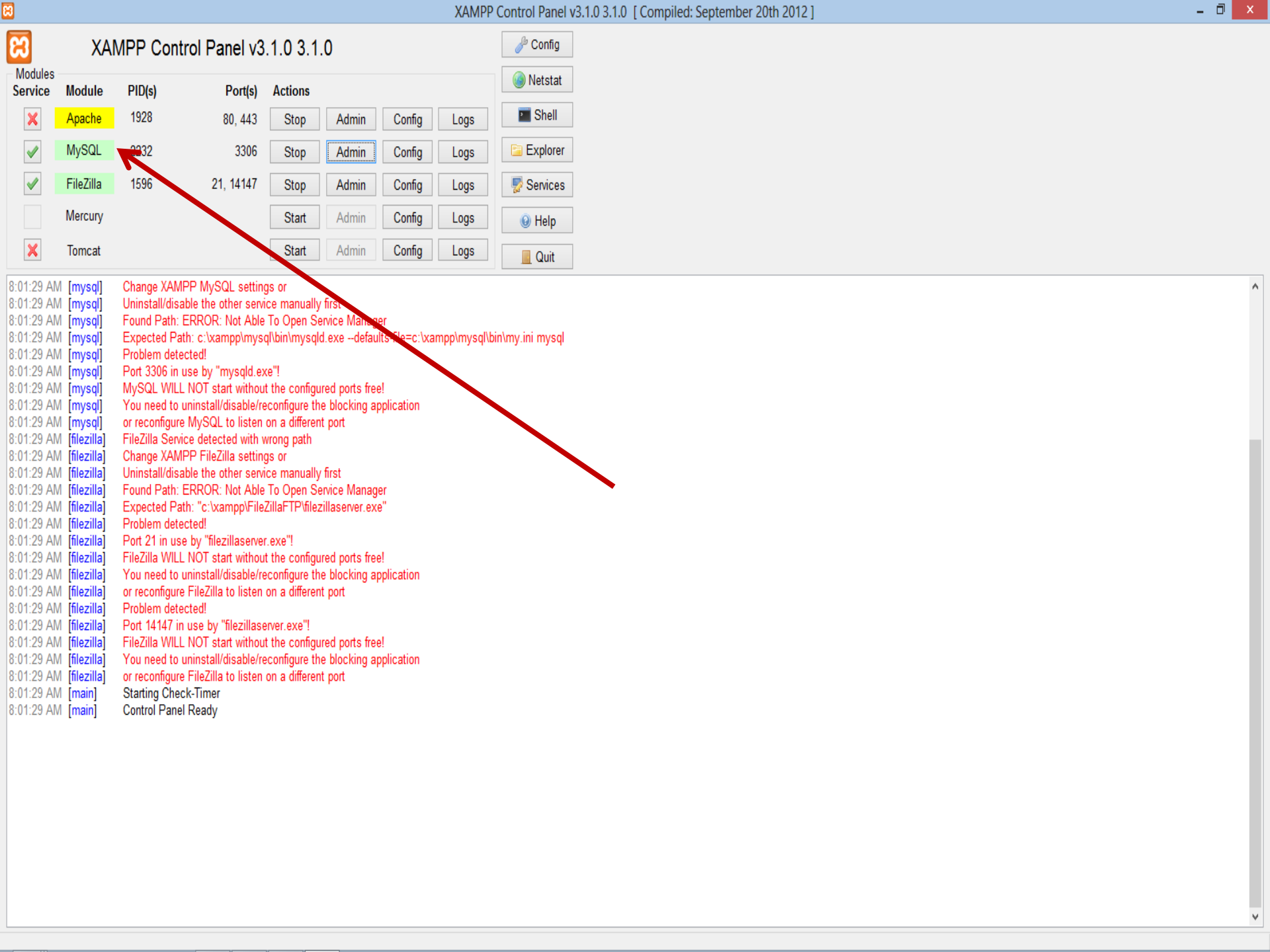
```
fwrite($fp, “hello world”);
fputs($fp, “hello world”);
```

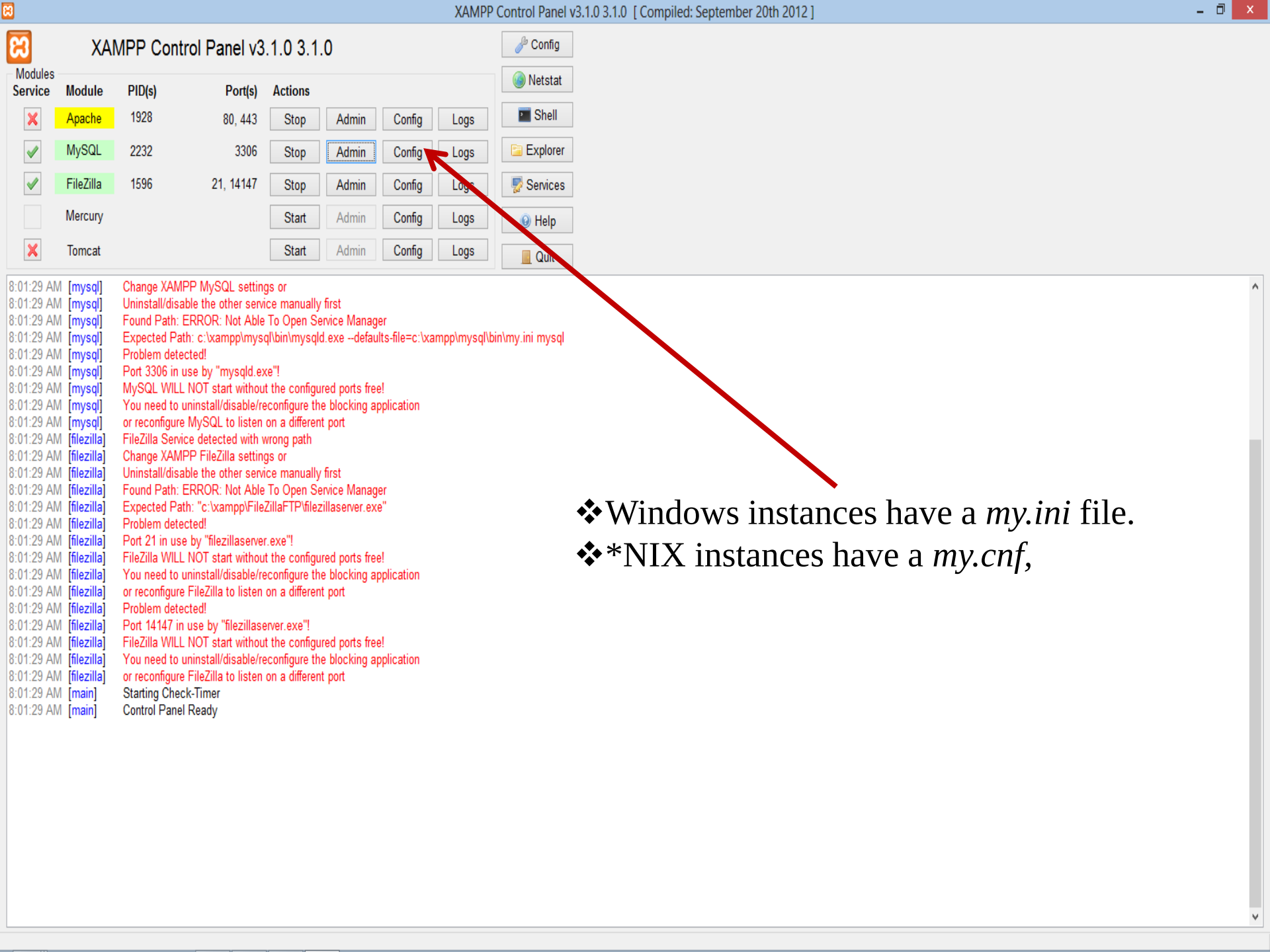
Filesystem Functions

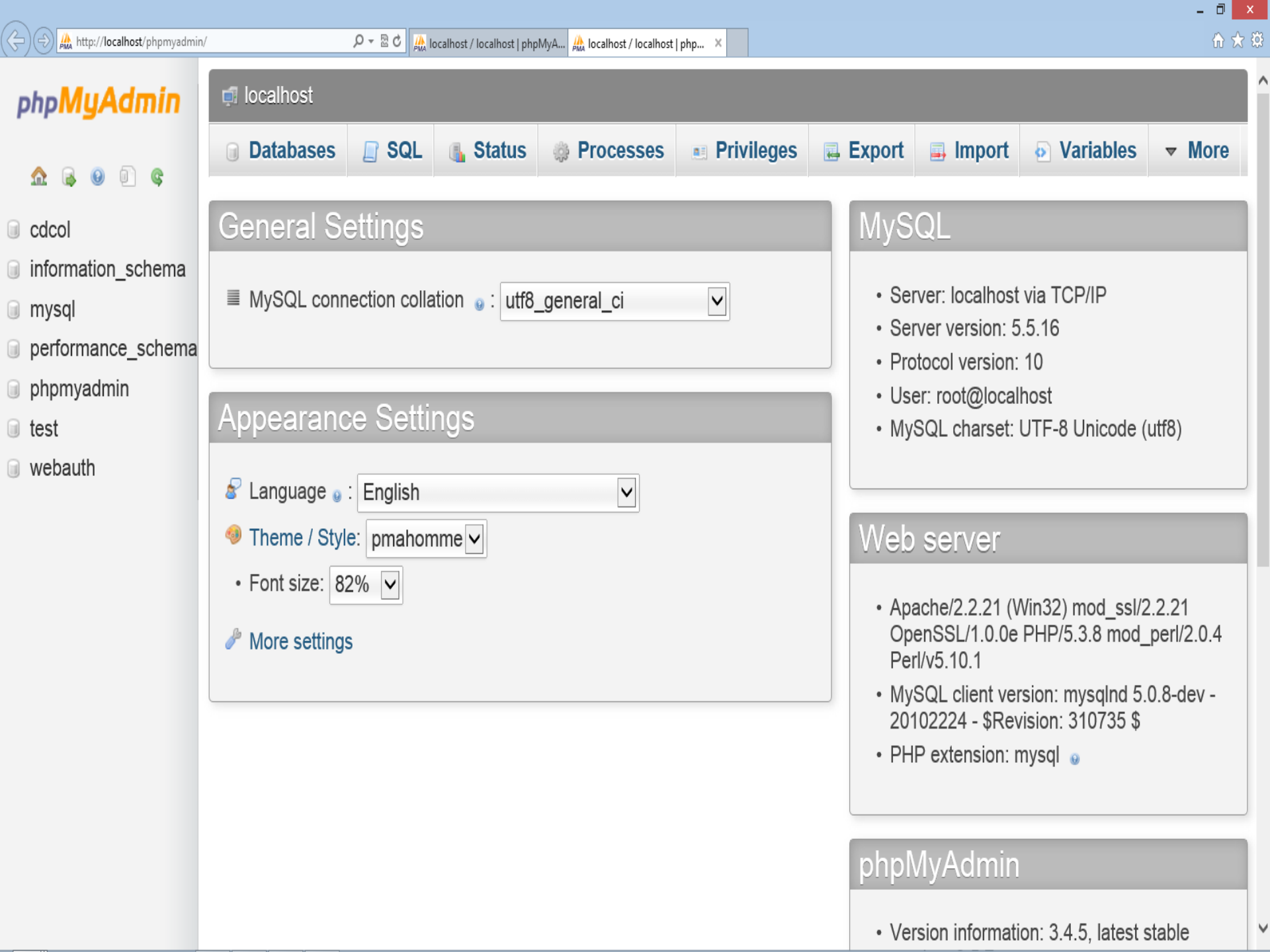
- chown — Changes file owner
- clearstatcache — Clears file status cache
- copy — Copies file
- delete — See unlink or unset
- dirname — Returns parent directory's path
- disk_free_space — Returns available space on filesystem or disk partition
- disk_total_space — Returns the total size of a filesystem or disk partition
- diskfreespace — Alias of disk_free_space
- fclose — Closes an open file pointer
- feof — Tests for end-of-file on a file pointer
- fflush — Flushes the output to a file
- fgetc — Gets character from file pointer
- fgetcsv — Gets line from file pointer and parse for CSV fields
- fgets — Gets line from file pointer
- fgetss — Gets line from file pointer and strip HTML tags
- file_exists — Checks whether a file or directory exists
- file_get_contents — Reads entire file into a string
- file_put_contents — Write a string to a file
- file — Reads entire file into an array
- fileatime — Gets last access time of file
- filectime — Gets inode change time of file
- filegroup — Gets file group
- fileinode — Gets file inode
- filemtime — Gets file modification time

PHP/MySQL Integration




XAMPP Control Panel v3.1.0 3.1.0 [Compiled: September 20th 2012]
XAMPP Control Panel v3.1.0 3.1.0
Modules
Service
Module
PID(s)
Port(s)
Actions
Apache
1928
80, 443
Stop
Admin
Config
Logs
MySQL
3932
3306
Stop
Admin
Config
Logs
FileZilla
1596
21, 14147
Stop
Admin
Config
Logs
Mercury
Start
Admin
Config
Logs
Tomcat
Start
Admin
Config
Logs
Config
Netstat
Shell
Explorer
Services
Help
Quit
8:01:29 AM [mysql] Change XAMPP MySQL settings or
8:01:29 AM [mysql] Uninstall/disable the other service manually first
8:01:29 AM [mysql] Found Path: ERROR: Not Able To Open Service Manager
8:01:29 AM [mysql] Expected Path: c:\xampp\mysql\bin\mysqld.exe --defaults-file=c:\xampp\mysql\bin\my.ini mysql
8:01:29 AM [mysql] Problem detected!
8:01:29 AM [mysql] Port 3306 in use by "mysqld.exe"!
8:01:29 AM [mysql] MySQL WILL NOT start without the configured ports free!
8:01:29 AM [mysql] You need to uninstall/disable/reconfigure the blocking application
8:01:29 AM [mysql] or reconfigure MySQL to listen on a different port
8:01:29 AM [filezilla] FileZilla Service detected with wrong path
8:01:29 AM [filezilla] Change XAMPP FileZilla settings or
8:01:29 AM [filezilla] Uninstall/disable the other service manually first
8:01:29 AM [filezilla] Found Path: ERROR: Not Able To Open Service Manager
8:01:29 AM [filezilla] Expected Path: "c:\xampp\FileZillaFTP\filezillaserver.exe"
8:01:29 AM [filezilla] Problem detected!
8:01:29 AM [filezilla] Port 21 in use by "filezillaserver.exe"!
8:01:29 AM [filezilla] FileZilla WILL NOT start without the configured ports free!
8:01:29 AM [filezilla] You need to uninstall/disable/reconfigure the blocking application
8:01:29 AM [filezilla] or reconfigure FileZilla to listen on a different port
8:01:29 AM [filezilla] Problem detected!
8:01:29 AM [filezilla] Port 14147 in use by "filezillaserver.exe"!
8:01:29 AM [filezilla] FileZilla WILL NOT start without the configured ports free!
8:01:29 AM [filezilla] You need to uninstall/disable/reconfigure the blocking application
8:01:29 AM [filezilla] or reconfigure FileZilla to listen on a different port
8:01:29 AM [main] Starting Check-Timer
8:01:29 AM [main] Control Panel Ready


XAMPP Control Panel v3.1.0 3.1.0 [Compiled: September 20th 2012]
XAMPP Control Panel v3.1.0 3.1.0
Modules
Service Module PID(s) Port(s) Actions
Apache 1928 80, 443 Stop Admin Config Logs
MySQL 2232 3306 Stop Admin Config Logs
FileZilla 1596 21, 14147 Stop Admin Config Logs
Mercury Start Admin Config Logs
Tomcat Start Admin Config Logs
Config
Netstat
Shell
Explorer
Services
Help
Quit
8:01:29 AM [mysql] Change XAMPP MySQL settings or
8:01:29 AM [mysql] Uninstall/disable the other service manually first
8:01:29 AM [mysql] Found Path: ERROR: Not Able To Open Service Manager
8:01:29 AM [mysql] Expected Path: c:\xampp\mysql\bin\mysqld.exe --defaults-file=c:\xampp\mysql\bin\my.ini mysql
8:01:29 AM [mysql] Problem detected!
8:01:29 AM [mysql] Port 3306 in use by "mysqld.exe"!
8:01:29 AM [mysql] MySQL WILL NOT start without the configured ports free!
8:01:29 AM [mysql] You need to uninstall/disable/reconfigure the blocking application
8:01:29 AM [mysql] or reconfigure MySQL to listen on a different port
8:01:29 AM [filezilla] FileZilla Service detected with wrong path
8:01:29 AM [filezilla] Change XAMPP FileZilla settings or
8:01:29 AM [filezilla] Uninstall/disable the other service manually first
8:01:29 AM [filezilla] Found Path: ERROR: Not Able To Open Service Manager
8:01:29 AM [filezilla] Expected Path: "c:\xampp\FileZillaFTP\filezillaserver.exe"
8:01:29 AM [filezilla] Problem detected!
8:01:29 AM [filezilla] Port 21 in use by "filezillaserver.exe"!
8:01:29 AM [filezilla] FileZilla WILL NOT start without the configured ports free!
8:01:29 AM [filezilla] You need to uninstall/disable/reconfigure the blocking application
8:01:29 AM [filezilla] or reconfigure FileZilla to listen on a different port
8:01:29 AM [filezilla] Problem detected!
8:01:29 AM [filezilla] Port 14147 in use by "filezillaserver.exe"!
8:01:29 AM [filezilla] FileZilla WILL NOT start without the configured ports free!
8:01:29 AM [filezilla] You need to uninstall/disable/reconfigure the blocking application
8:01:29 AM [filezilla] or reconfigure FileZilla to listen on a different port
8:01:29 AM [main] Starting Check-Timer
8:01:29 AM [main] Control Panel Ready
❖Windows instances have a *my.ini* file.
❖*NIX instances have a *my.cnf*,


http://localhost/phpmyadmin/
localhost / localhost | phpMyA...
localhost / localhost | php...
phpMyAdmin
cdcol
information_schema
mysql
performance_schema
phpmyadmin
test
webauth
localhost
Databases
SQL
Status
Processes
Privileges
Export
Import
Variables
More
General Settings
MySQL connection collation : utf8_general_ci
Appearance Settings
Language : English
Theme / Style: pmahomme
Font size: 82%
More settings
MySQL
Server: localhost via TCP/IP
Server version: 5.5.16
Protocol version: 10
User: root@localhost
MySQL charset: UTF-8 Unicode (utf8)
Web server
Apache/2.2.21 (Win32) mod_ssl/2.2.21 OpenSSL/1.0.0e PHP/5.3.8 mod_perl/2.0.4 Perl/v5.10.1
MySQL client version: mysqlnd 5.0.8-dev - 20102224 - $Revision: 310735 $
PHP extension: mysql
phpMyAdmin
Version information: 3.4.5, latest stable

Documentation Library
Table of Contents
MySQL 5.6 Manual
MySQL 5.5 Manual
MySQL 5.1 Manual
MySQL 5.0 Manual
MySQL 3.23/4.0/4.1 Manual
Search manual:
Go

MySQL 5.5 Reference Manual

Including MySQL Cluster NDB 7.2 Reference Guide

Abstract

This is the MySQL™ Reference Manual. It documents MySQL 5.5 through 5.5.32, as well as MySQL Cluster releases based on version 7.2 of NDBCLUSTER through 5.5.30-ndb-7.2.13.

MySQL 5.5 features. This manual describes features that are not included in every edition of MySQL 5.5; such features may not be included in the edition of MySQL 5.5 licensed to you. If you have any questions about the features included in your edition of MySQL 5.5, refer to your MySQL 5.5 license agreement or contact your Oracle sales representative.

For release notes detailing the changes in each release, see the MySQL 5.5 Release Notes.

For legal information, see the Legal Notices.

General	Administrators	MySQL Enterprise	Developers & Functionality	Connectors & APIs	HA/Scalability
Tutorial	Installation &	MySQL	MySQL	Connectors and	» HA/Scalability

Preface and Legal Notices »

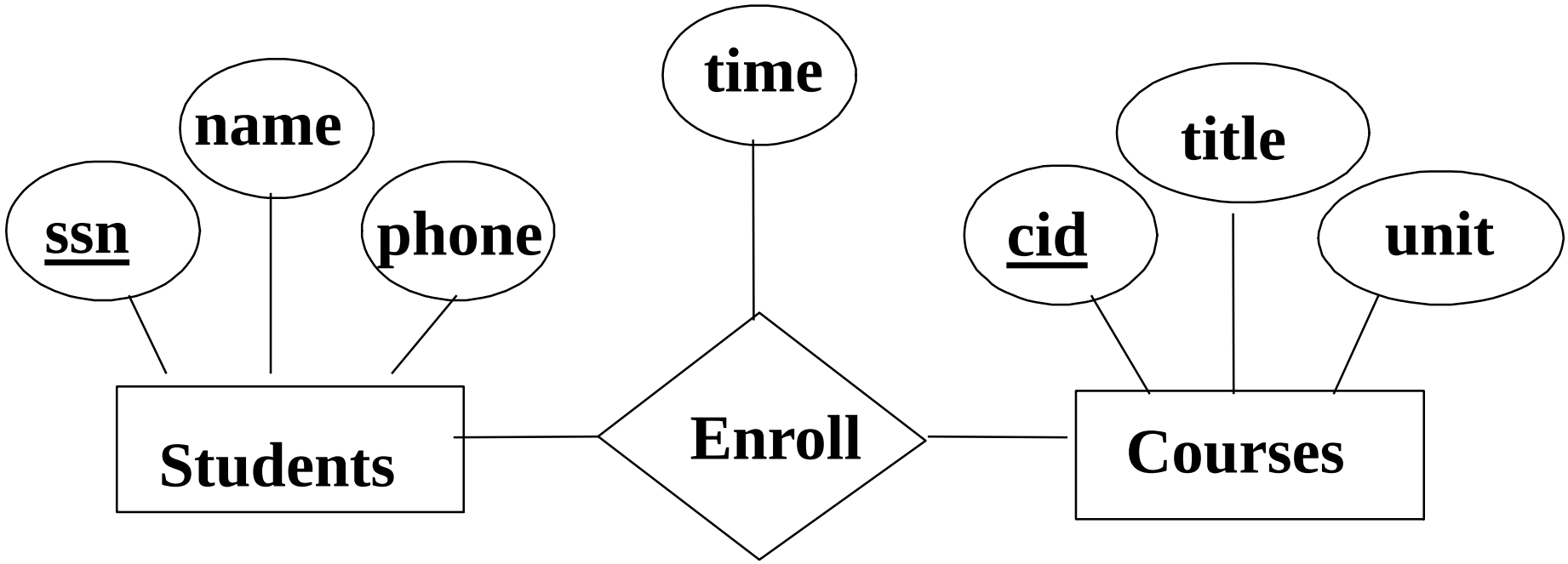
Section Navigation [Toggle]

- MySQL 5.5 Reference Manual
- Preface and Legal Notices
- 1 General Information
- 2 Installing and Upgrading MySQL
- 3 Tutorial
- 4 MySQL Programs
- 5 MySQL Server Administration
- 6 Security
- 7 Backup and Recovery
- 8 Optimization
- 9 Language Structure
- 10 Globalization
- 11 Data Types
- 12 Functions and Operators
- 13 SQL Statement Syntax
- 14 Storage Engines
- 15 High Availability and Scalability
- 16 Replication
- 17 MySQL Cluster NDB 7.2

Database Design

- **Database design** is the process of producing a detailed **data model** (e.g. ER Model) of a database.
  - This term can also be used to describe many different parts of the design of an overall database system.
  - Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data.
  - The process of doing database design generally consists of the following two steps:
    - Determine the **relationships** between the different data elements.
    - Superimpose a **logical structure** upon the data on the basis of these relationships.

An Example of ER Model



ER Model

- **Entity**: Real-world object distinguishable from other objects. E.g. Students, Courses.

- An entity has multiple **attributes**. E.g. Students have ssn, name, phone.

- Entities have **relationships** with each other. E.g. Students enroll Courses.

Relationships

- There are three types of data relationships
  - One-to-one
  - One-to-many
  - Many-to-many

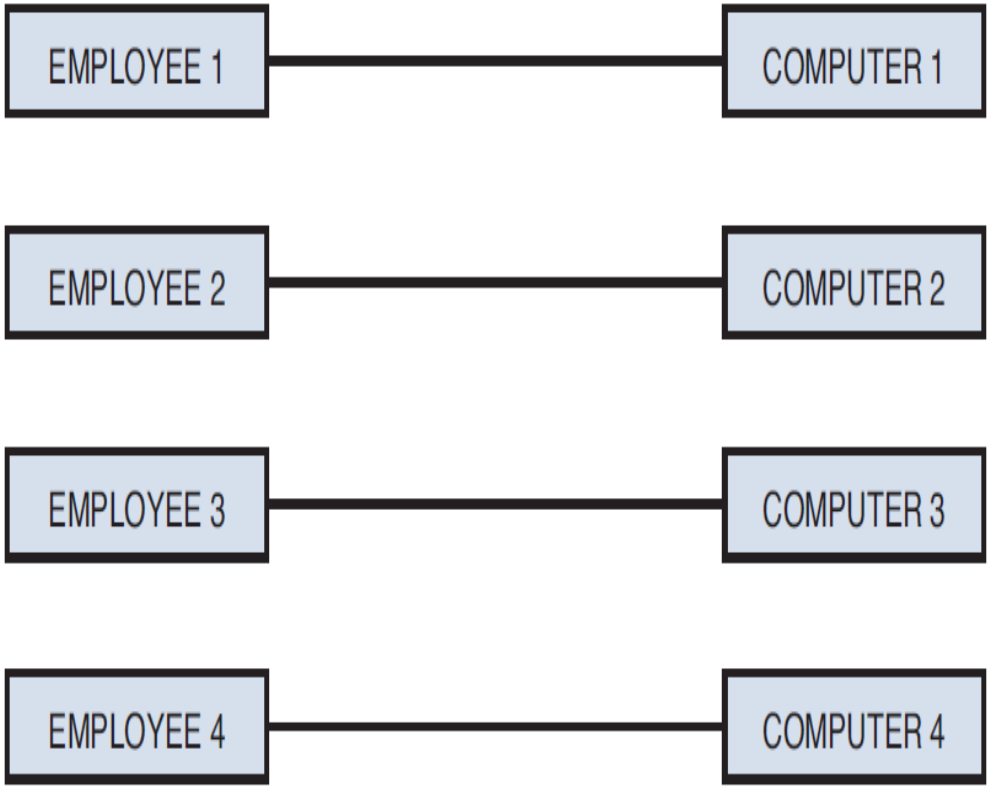
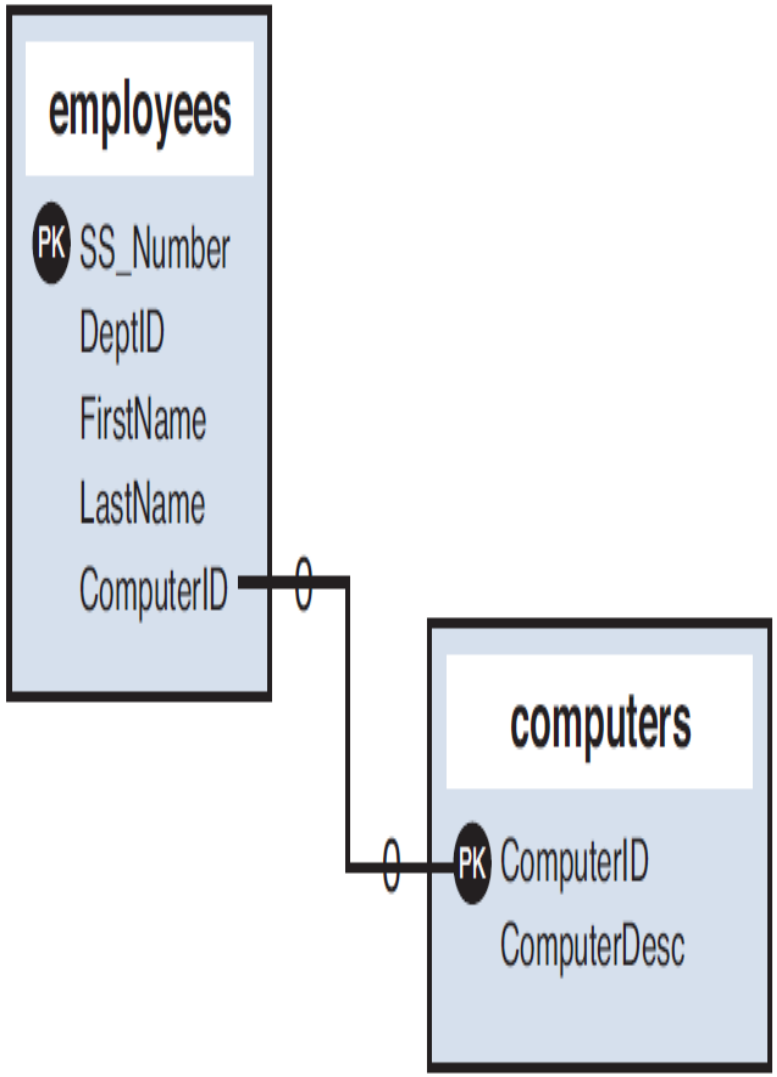


FIGURE 15.2
One computer is assigned to each employee.

The `employees` and `computers` tables in your database would look something like Figure 15.3, which represents a one-to-one relationship.

FIGURE 15.3
One-to-one relationship in the data model.



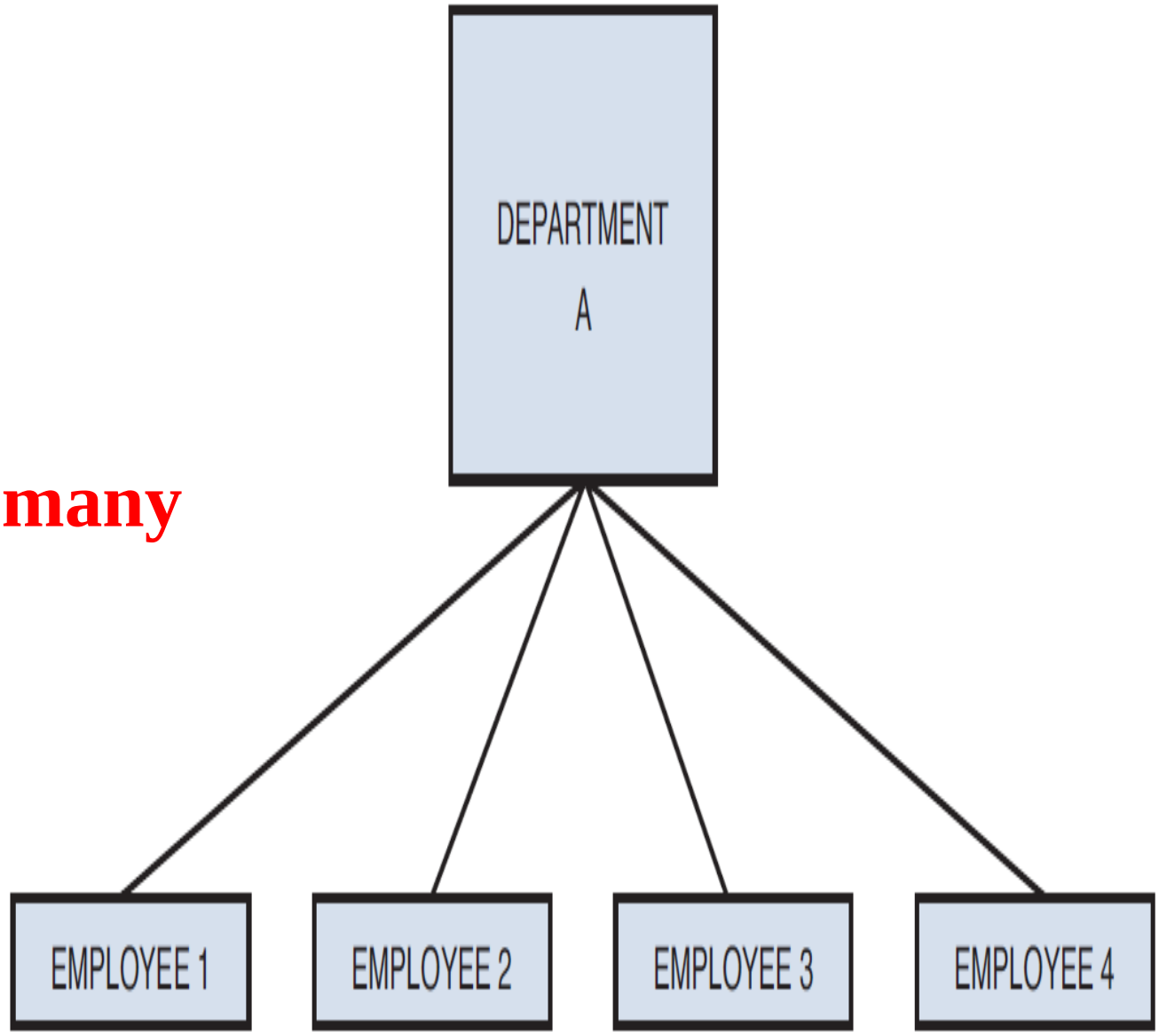
Primary Key

- In a relational database, a **Primary Key** is a key that uniquely defines the characteristics of each row (also known as record or tuple).
  - For example, a birthday could be shared by many people and so it would not be a prime candidate for the Primary Key.
  - A social security number or Driver's License number would be ideal since it correlates to one single data value.

FIGURE 15.4

One department contains many employees.

One-to-many



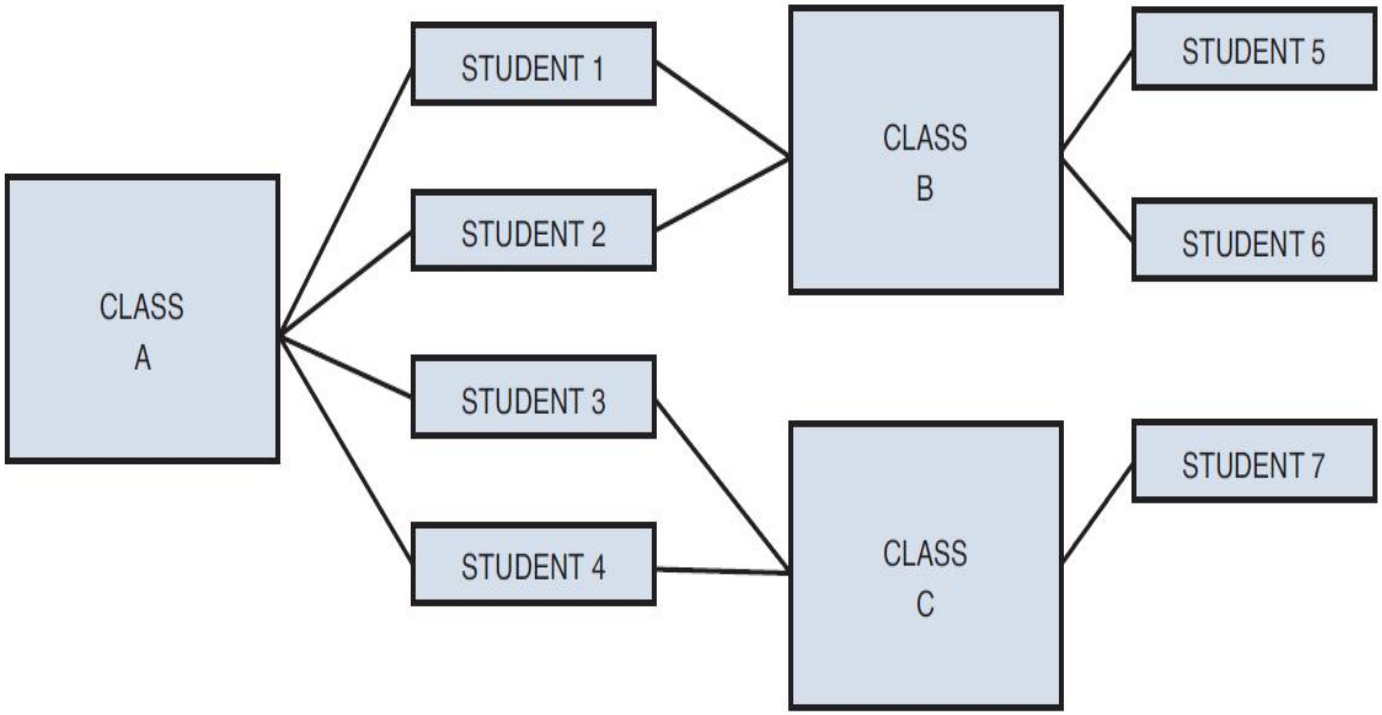


FIGURE 15.5
Students take classes, and classes contain students.

As you can see, this sort of relationship does not present an easy method for relating tables. Your tables could look like Figure 15.6, seemingly unrelated.

Many-to-many

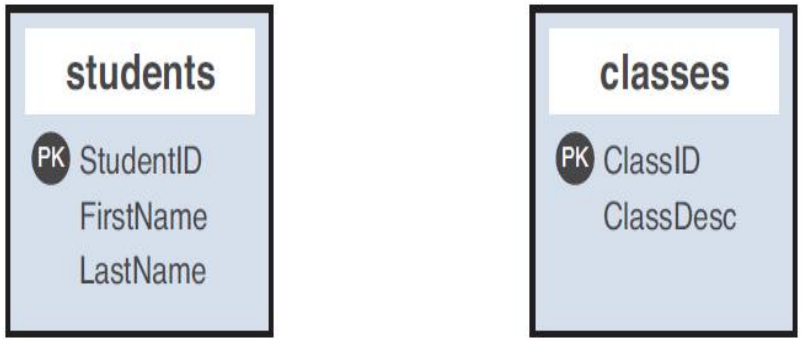


FIGURE 15.6
The `students` table and the `classes` table, unrelated.

FIGURE 15.7
The `students_classes_map` table acts as an intermediary.

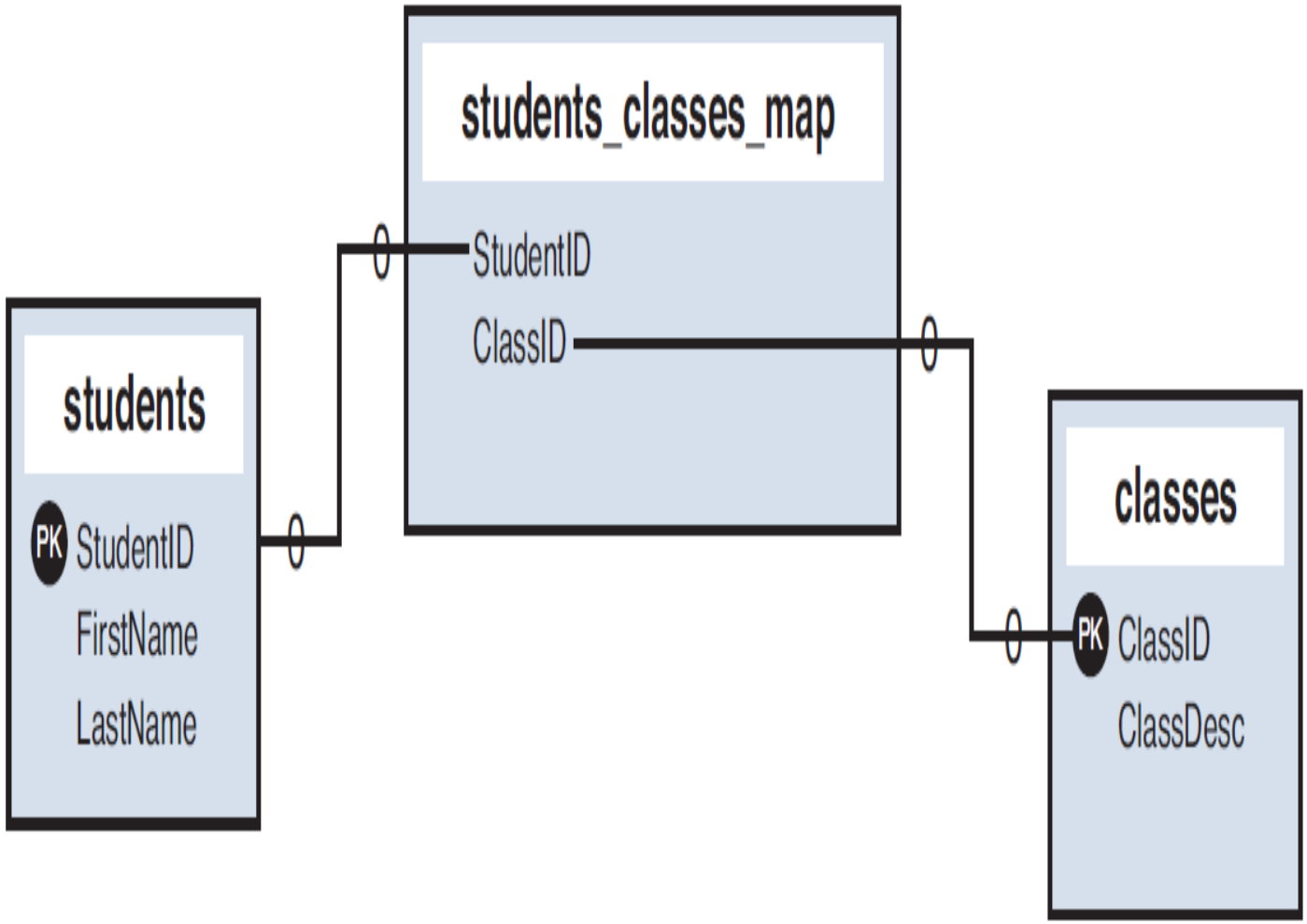


FIGURE 15.8
The `students_classes_map` table populated with data.

STUDENTID	CLASSID
STUDENT 1	CLASS A
STUDENT 2	CLASS A
STUDENT 3	CLASS A
STUDENT 4	CLASS A
STUDENT 5	CLASS B
STUDENT 6	CLASS B
STUDENT 7	CLASS C
STUDENT 1	CLASS B
STUDENT 2	CLASS B
STUDENT 3	CLASS C
STUDENT 4	CLASS C

Normalization

- Normalization is simply a set of rules that will ultimately make your life easier.
- It is the art of organizing your database in such a way that your tables are appropriately related, and are flexible enough for future growth.
- The sets of rules used in normalization are called **normal forms**.
  - If your database design follows the first set of rules, it is considered in the first normal form.
  - If the first three sets of rules of normalization are followed, your database is said to be in the third normal form.

Problems with the Flat Table

- In the students-and-courses database, assume that you have the following fields in your flat table:
  - **StudentName**: The name of the student.
  - **CourseID1:** The ID of the first course taken by the student.
  - **CourseDescription1:** The description of the first course taken by the student.
  - **CourseInstructor1:** The instructor of the first course taken by the student.
  - **CourseID2**—The ID of the second course taken by the student.
  - **CourseDescription2**—The description of the second course taken by the student.
  - **CourseInstructor2:** The instructor of the second course taken by the student.
  - …

First Normal Form

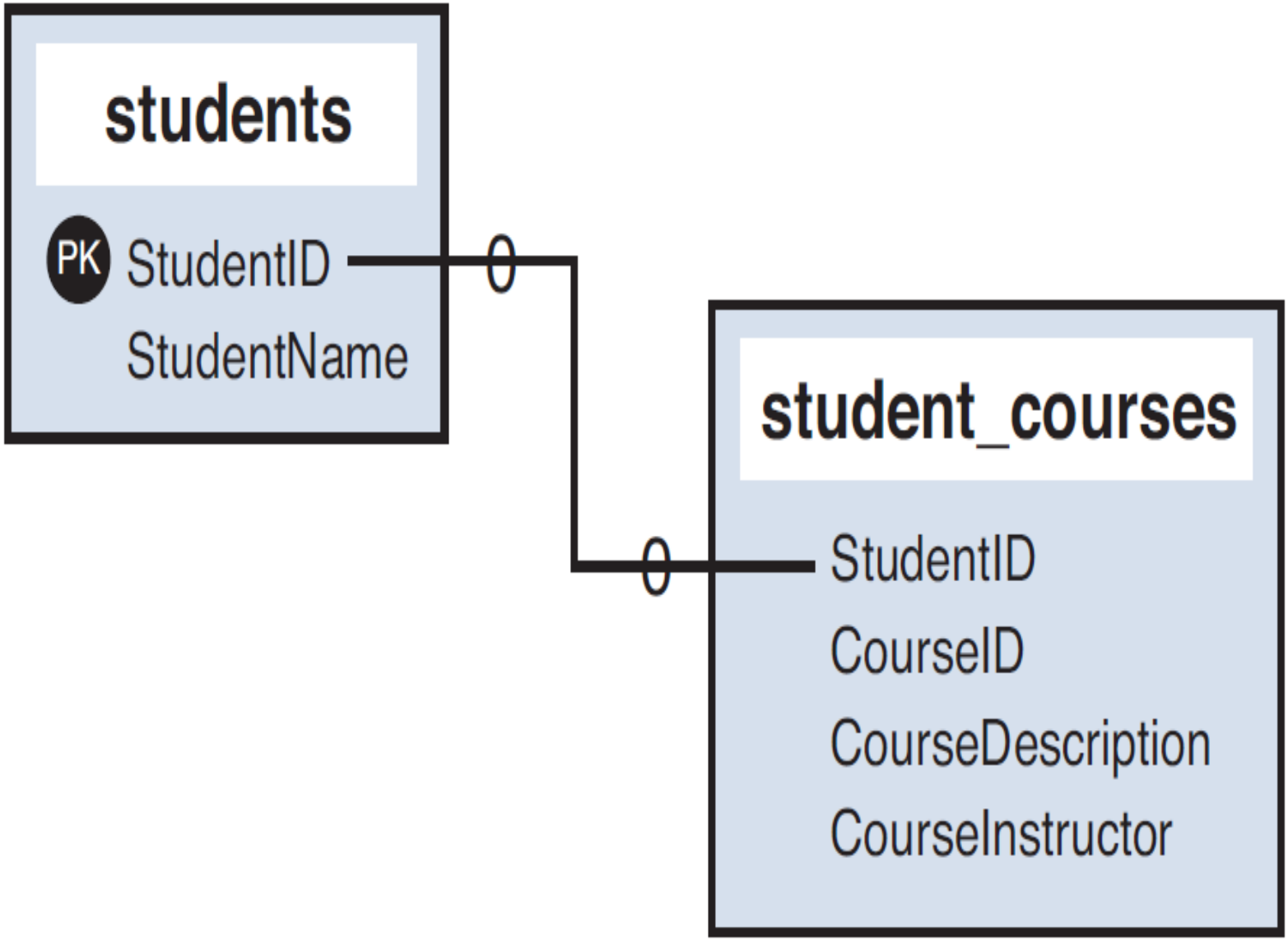
- The rules for the first normal form are as follows:

  Eliminate repeating information.

  Create separate tables for related data.

- If you think about the flat table design with many repeated sets of fields for the students-and-courses database, you can identify two distinct topics: **students** and **courses**.

- Taking your students-and-courses database to the first normal form means that you create two tables: one for students and one for courses.

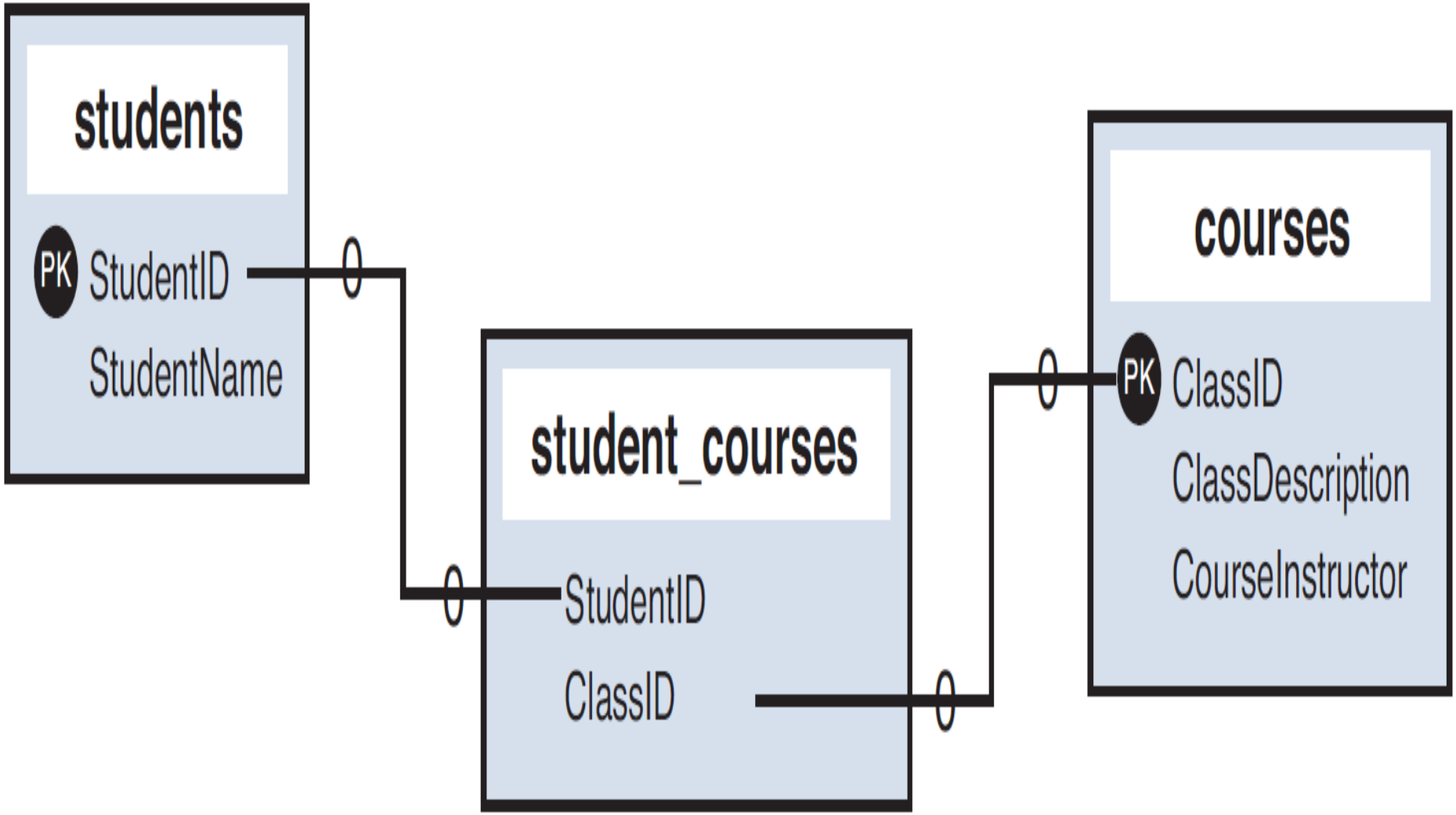

students
PK
StudentID
StudentName
student_courses
StudentID
CourseID
CourseDescription
CourseInstructor

Second Normal Form

- The rule for the second normal form is as follows:

  No non-key attributes depend on a portion of the primary key.

- In plain English, this means that if fields in your table are not entirely related to a primary key, you have more work to do.
  - In the students-and-courses example, you need to break out the courses into their own table and modify the students_courses table.
  - CourseID, CourseDescription, and CourseInstructor can become a table called courses with a primary key of CourseID.
  - The students_courses table should then just contain two fields: StudentID and CourseID.

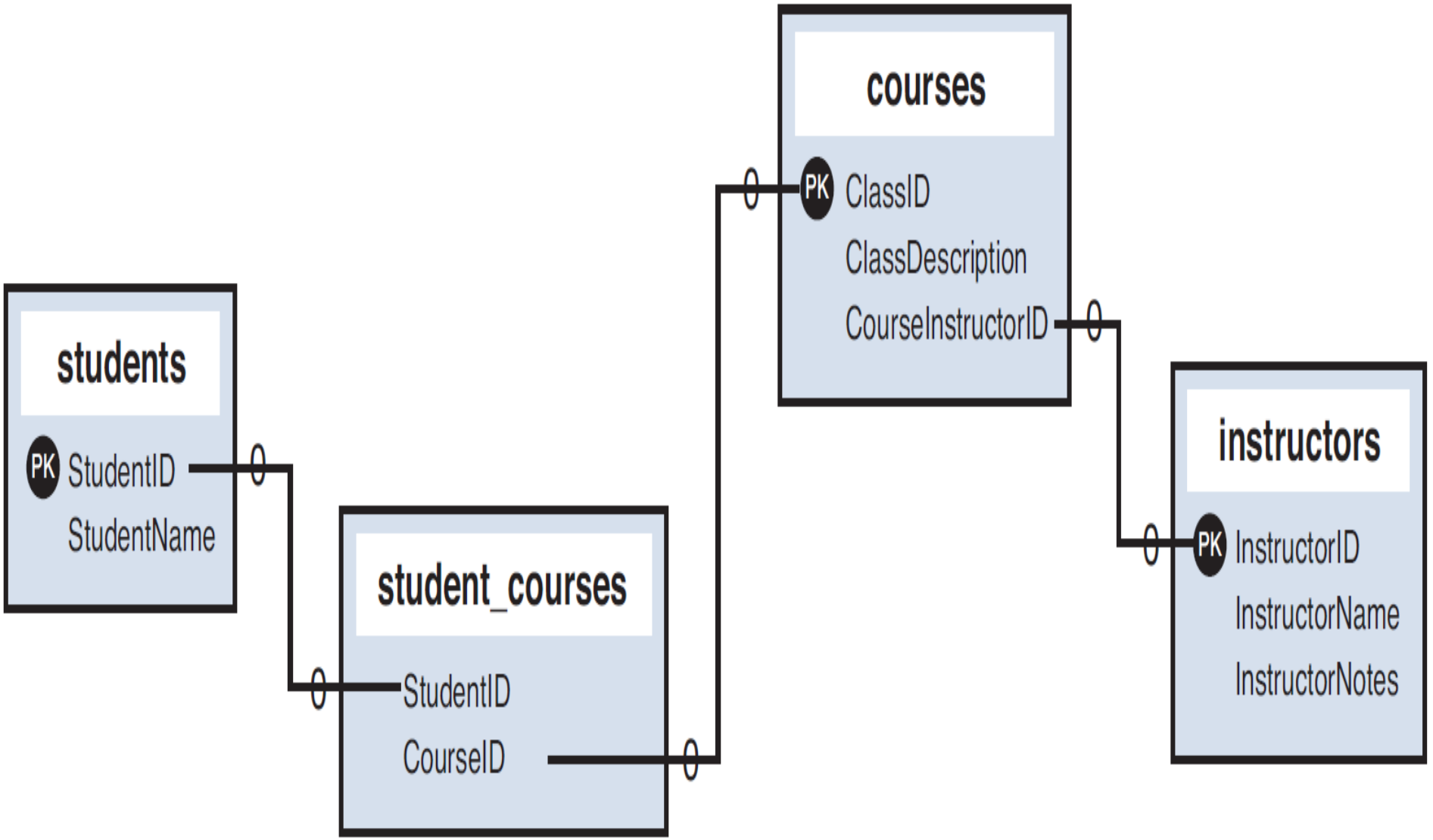

students
PK
StudentID
StudentName
student_courses
StudentID
ClassID
courses
PK
ClassID
ClassDescription
CourseInstructor

Third Normal Form

- The rule for the third normal form is as follows:

  No attributes depend on other non-key attributes.

- This rule simply means that you need to look at your tables and see whether you have more fields that can be broken down further and that are not dependent on a key.
  - Think about removing repeated data and you'll find your answer: instructors.
  - Inevitably, an instructor teaches more than one class. However, CourseInstructor is not a key of any sort.
  - So, if you break out this information and create a separate table purely for the sake of efficiency and maintenance, that is the third normal form.


courses
PK
ClassID
ClassDescription
CourseInstructorID
students
PK
StudentID
StudentName
student_courses
StudentID
CourseID
instructors
PK
InstructorID
InstructorName
InstructorNotes

Other Normal Forms

- There are more than three normal forms. Additional forms are:
  - The Boyce-Codd normal form
  - Fourth normal form
  - Fifth normal form
  - Join-Projection normal form.
- These forms are not often followed in practical application development because the benefits of doing so are outweighed by the cost in man-hours and database efficiency.
- For more information, please see http://en.wikipedia.org/wiki/Database_normalization#Normal_forms.

SQL

- **SQL** is a special-purpose programming language designed for managing data held in a relational database management systems (RDBMS).
  - SQL consists of a data definition language and a data manipulation language.
  - SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standards (ISO) in 1987.
    - But different vendors do not perfectly follow the standard. Hence, code is not completely portable among different database systems.
  - The scope of SQL includes schema/table creation, data insert, delete, query, update and and modification, and data access control.

Table Creation

- The generic table-creation syntax is:

  CREATE TABLE table_name (column_name column_type);

- The following table-creation example creates a generic grocery_inventory table with fields for ID, item name, item description, item price, and quantity:

```
CREATE TABLE grocery_inventory (
id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
item_name VARCHAR (50) NOT NULL,
item_desc TEXT,
item_price FLOAT NOT NULL,
curr_qty INT NOT NULL
);
```

INSERT

- After you have created some tables, you use the SQL command INSERT for adding new records to these tables. The basic syntax of INSERT is

  INSERT INTO table_name (column list) VALUES (column values);

- Using the grocery_inventory table as an example, you have five fields: id, item_name, item_desc, item_price, and curr_qty. To insert a complete record, you could use either of these statements:

  INSERT INTO grocery_inventory
  (id, item_name, item_desc, item_price, curr_qty)
  VALUES ('1', 'Apples', 'Beautiful, ripe apples.', '0.25', 1000);

  INSERT INTO grocery_inventory VALUES ('2', 'Bunches of Grapes', 'Seedless grapes.', '2.99', 500);

SELECT

- The most basic SELECT syntax looks like this:

  SELECT expressions_and_columns FROM table_name
  [WHERE some_condition_is_true]
  [ORDER BY some_column]
  [LIMIT offset, rows]

- One handy expression is the * symbol, which stands for *everything*. So, to select everything (all rows, all columns) from the grocery_inventory table, your SQLstatement would be:

  SELECT * FROM grocery_inventory;

  SELECT id, item_name, curr_qty FROM grocery_inventory;

WHERE

- From the example SELECT syntax, you see that WHERE is used to specify a particular condition:

```
SELECT expressions_and_columns FROM table_name
[WHERE some_condition_is_true]
```

- An example is to retrieve all the records for items with a quantity of 500:

```
SELECT * FROM grocery_inventory WHERE curr_qty = 500;
```

Ordering SELECT Results

- Results of SELECT queries are ordered as they were inserted into the table and should not be relied on as a meaningful ordering system.
- If you want to order results a specific way, such as by date, ID, name, and so on, specify your requirements using the ORDER BY clause.
- In the following statement, the intention is a result set ordered alphanumerically by item_name:

```
SELECT id, item_name, curr_qty FROM grocery_inventory
ORDER BY item_name;
```

SELECT Multiple Tables

- You are not limited to selecting only one table at a time.
- When you select from more than one table in one SELECT statement, you are really joining the tables together.

```
SELECT * FROM fruit, color;

SELECT fruitname, colorname FROM fruit, color
WHERE fruit.id = color.id;
```

Using DELETE

- The basic DELETE syntax is as follows:

  DELETE FROM table_name

  [WHERE some_condition_is_true]

  [LIMIT rows]

- Notice that no column specification is used in the DELETE command
- When you use DELETE, the entire record is removed.

Mysql vs Mysqli

- Before MySQL 4.1.3 (2005), the **mysql_*** family of functions were standard.
  - You might still find code examples using them.
- But since MySQL 4.1.3, the database system includes functionality necessitating new communications methods in PHP, all encompassed in the **mysqli_*** family of functions.
  - All code in this class uses the mysqli_* family of functions.
- For more information, see the PHP Manual chapter "MySQL Improved Extension," at http://www.php.net/mysqli.

MySQL Improved Extension

mysqlconnect.php

```
<?php
$mysqli = mysqli_connect("localhost", "root", "", "testDB");

if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
} else {
    printf("Host information: %s\n", mysqli_get_host_info($mysqli));
}
?>
```

«mysqli::commit
mysqli::$connect_error»

view this page in Brazilian Portuguese

[edit] Last updated: Fri, 05 Apr 2013

mysqli::$connect_errno

mysqli_connect_errno

(PHP 5)

mysqli::$connect_errno -- mysqli_connect_errno — Returns the error code from last connect call

Description

Report a bug

Object oriented style

string *$mysqli->connect_errno*;

Procedural style

int **mysqli_connect_errno** (void)

Returns the last error code number from the last call to mysqli_connect().

Note:

mysqlconnect_modified.php

- Although the connection closes when the script finishes its execution, it is a good practice to close the connection explicitly.

```
<?php
$mysqli = mysqli_connect(" localhost", "root", "", "testDB");

if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
} else {
    printf("Host information: %s\n", mysqli_get_host_info($mysqli));
    mysqli_close($mysqli);
}
?>
```

Executing Queries

```
<?php
$mysqli = mysqli_connect(“localhost", "root", "", " testDB");
if (mysqli_connect_errno()) {
        printf("Connect failed: %s\n", mysqli_connect_error());
        exit();
} else {
        $sql = "CREATE TABLE testTable (id INT NOT NULL PRIMARY KEY
AUTO_INCREMENT, testField VARCHAR (75))";
        $res = mysqli_query($mysqli, $sql);
        if ($res === TRUE) {
                echo "Table testTable successfully created.";
        } else {
                printf("Could not create table: %s\n", mysqli_error($mysqli));
        }
        mysqli_close($mysqli);
}
?>
```

mysqli_query

(PHP 5)

mysqli::query -- mysqli_query — Performs a query on the database

Description

Report a bug

Object oriented style

mixed **mysqli::query** (string *$query* [, int *$resultmode* = *MYSQLI_STORE_RESULT*])

Procedural style

mixed **mysqli_query** (mysqli *$link* , string *$query* [, int *$resultmode* = *MYSQLI_STORE_RESULT*])

Performs a ***query*** against the database.

For non-DML queries (not INSERT, UPDATE or DELETE), this function is similar to calling mysqli_real_query() followed by either mysqli_use_result() or mysqli_store_result().

Note:

In the case where you pass a statement to **mysqli_query()** that is longer than ***max_allowed_packet*** of the server, the returned error codes are different depending on whether you are using MySQL Native Driver (***mysqlnd***) or MySQL Client Library (***libmysqlclient***). The behavior is as follows:

- ***mysqlnd*** on Linux returns an error code of 1153. The error message means "got a packet bigger than ***max_allowed_packet*** bytes".
- ***mysqlnd*** on Windows returns an error code 2006. This error message means "server has gone away".
- ***libmysqlclient*** on all platforms returns an error code 2006. This error message means "server has gone away".

Insert Data with PHP

```
<?php
$mysqli = mysqli_connect("localhost", “root", "", “testDB");
if (mysqli_connect_errno()) {
        printf("Connect failed: %s\n", mysqli_connect_error());
        exit();
} else {
        $sql = "INSERT INTO testTable (testField) VALUES ('some value')";
        $res = mysqli_query($mysqli, $sql);
        if ($res === TRUE) {
                echo "A record has been inserted.";
        } else {
                printf("Could not insert record: %s\n", mysqli_error($mysqli));
        }
        mysqli_close($mysqli);
}
?>
```

Retrieving Data with PHP

```
<?php
$mysqli = mysqli_connect("localhost", "root", "", "testDB ");
if (mysqli_connect_errno()) {
        printf("Connect failed: %s\n", mysqli_connect_error());
        exit();
} else {
        $sql = "SELECT * FROM testTable";
        $res = mysqli_query($mysqli, $sql);

        if ($res) {
                $number_of_rows = mysqli_num_rows($res);
                printf("Result set has %d rows.\n", $number_of_rows);
        } else {
                printf("Could not retrieve records: %s\n", mysqli_error($mysqli));
        }
        mysqli_free_result($res);
        mysqli_close($mysqli);
}
?>
```

mysqli_result::$num_rows

mysqli_num_rows

(PHP 5)

mysqli_result::$num_rows -- mysqli_num_rows — Gets the number of rows in a result

Description

Report a bug

Object oriented style

int *$mysqli_result->num_rows*;

Procedural style

int **mysqli_num_rows** (mysqli_result *$result*)

Returns the number of rows in the result set.

The behaviour of **mysqli_num_rows()** depends on whether buffered or unbuffered result sets are being used. For unbuffered result sets, **mysqli_num_rows()** will not return the correct number of rows until all the rows in the result have been retrieved.

Parameters

Report a bug

result

Procedural style only: A result set identifier returned by mysqli_query(), mysqli_store_result() or mysqli_use_result().

Retrieving Data with PHP

```
<?php
$mysqli = mysqli_connect("localhost", "root", "", “testDB");
if (mysqli_connect_errno()) {
        printf("Connect failed: %s\n", mysqli_connect_error());
        exit();
} else {
        $sql = "SELECT * FROM testTable";
        $res = mysqli_query($mysqli, $sql);
        if ($res) {
                while ($newArray = mysqli_fetch_array($res, MYSQLI_ASSOC)) {
                        $id  = $newArray['id'];
                        $testField = $newArray['testField'];
                        echo "The ID is ".$id." and the text is: ".$testField."<br/>";
                }
        } else {
                printf("Could not retrieve records: %s\n", mysqli_error($mysqli));
        }
        mysqli_free_result($res);
        mysqli_close($mysqli);
}
?>
```

mysqli_fetch_array

(PHP 5)

mysqli_result::fetch_array -- mysqli_fetch_array — Fetch a result row as an associative, a numeric array, or both

Description

Report a bug

Object oriented style

mixed **mysqli_result::fetch_array** ([int *$resulttype* = *MYSQLI_BOTH*])

Procedural style

mixed **mysqli_fetch_array** (mysqli_result *$result* [, int *$resulttype* = *MYSQLI_BOTH*])

Returns an array that corresponds to the fetched row or **NULL** if there are no more rows for the resultset represented by the ***result*** parameter.

mysqli_fetch_array() is an extended version of the mysqli_fetch_row() function. In addition to storing the data in the numeric indices of the result array, the **mysqli_fetch_array()** function can also store the data in associative indices, using the field names of the result set as keys.

Note: Field names returned by this function are ***case-sensitive***.

Note: This function sets NULL fields to the PHP **NULL** value.

If two or more columns of the result have the same field names, the last column will take precedence and overwrite the earlier data. In order to access multiple columns with the same name, the numerically indexed version of the row must be used.

Parameters

Report a bug

Additional MySQL Functions in PHP

- More than 100 MySQL-specific functions are available through the MySQLi interface in PHP.
  - Most of these functions are simply alternative methods of retrieving data or are used to gather information about the table structure in question.
  - For a complete list of functions, with practical examples, visit the MySQLi section of the PHP Manual at http://www.php.net/mysqli.