MODULE 6

Server Side Scripting using PHP

INTRODUCTION

* PHP is a powerful tool for making dynamic and interactive Web pages. PHP is the widely-used, free, and efficient alternative to competitors such as Microsoft's ASP.

What is PHP?

* PHP stands for PHP: Hypertext Preprocessor
* PHP is a server-side scripting language, like ASP
* PHP scripts are executed on the server
* PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
* PHP is an open source software
* PHP is free to download and use

PHP File

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

Why PHP?

* PHP runs on different platforms (Windows, Linux, Unix, etc.)
* PHP is compatible with almost all servers used today (Apache, IIS, etc.)
* PHP is FREE to download from the official PHP resource: [www.php.net](http://www.php.net/)
* PHP is easy to learn and runs efficiently on the server side

**Basic PHP Syntax**

A PHP scripting block always starts with ***<?php*** and ends with ***?>***. A PHP scripting block can be placed anywhere in the document. On servers with shorthand support enabled you can start a scripting block with <? and end with ?>.

|  |
| --- |
| <?php ?> |

* A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.
* Example e of a simple PHP script which sends the text "Hello World" to the browser:

|  |
| --- |
| * <html>  <body>  <?php  echo "Hello World";  ?>  </body>  </html> |

* Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.
* There are two basic statements to output text with PHP: ***echo*** and ***print***. In the example above we have used the echo statement to output the text "Hello World".
* Note: The file must have a .php extension. If the file has a .html extension, the PHP code will not be executed.

**Comments in PHP**

* In PHP, we use // to make a single-line comment or /\* and \*/ to make a large comment block.

|  |
| --- |
| * <html> <body> <?php //This is a comment /\* This is a comment block \*/ ?> </body> </html> |

**Variables in PHP**

* Variables are used for storing values, like text strings, numbers or arrays. When a variable is declared, it can be used over and over again in your script. All variables in PHP start with a ***$*** sign symbol.
* The correct way of declaring a variable in PHP:

|  |
| --- |
| $var\_name = value; |

* A variable containing a string, and a variable containing a number:

|  |
| --- |
| * <?php $txt="Hello World!"; $x=16; ?> |

* String Variables in PHP
* String variables are used for values that contain characters. After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

|  |
| --- |
| * <?php $txt="Hello World"; echo $txt; ?> |

* The output of the code above will be:

|  |
| --- |
| * Hello World |

**The Concatenation Operator**

* There is only one string operator in PHP. The concatenation operator (.)  is used to put two string values together. To concatenate two string variables together, use the concatenation operator:
* <?php  
  $txt1="Hello World!";  
  $txt2="What a nice day!";  
  echo $txt1 . " " . $txt2; ?>

|  |
| --- |
|  |

* The output of the code above will be:
  + Hello World! What a nice day!

|  |
| --- |
|  |

* If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string (a space character), to separate the two strings.

**Versions of PHP**

* PHP being HTML-centric - that is, PHP code was embedded inside HTML
* First Popular Version- PHP/FI 2.0 (1994) (Personal Home Page/ Form Interpreter)
  + The main issue with this version was that the PHP/FI parser was largely hand-written, and so users often met scripting errors that were technically not errors - they were just the PHP/FI parser screwing up
* PHP 3.0(1997)
  + the language extensible
  + speed was improved
  + the language was a lot stricter (Disadvantage)
  + some code that worked on PHP/FI would no longer work after upgrading.
  + execute while interpreting
* PHP 4.0(2000)
  + Zend Engine(Zeev and Andi) :- The engine took over the core of PHP, whereby all resources used in scripts (database connections, files, etc) are tracked automatically by the engine, and freed when no longer used to minimize memory usage and ensure there were no memory leaks.
  + complete web server abstraction
  + PHP runs on Apache 2, Microsoft's IIS, Zeus and more
  + compile first, execute later
  + PHP 4 compiled the entire script before executing it, it became possible to improve and store the compiled code before execution
  + Introduced multi-threading, which essentially allows particularly lengthy, but non-critical functions to be run independently from the main script process, further streamlining execution.
* PHP 5.0
  + Object Oriented Scripts
  + new error checking in the form of try/catch
  + interaction with XML documents, the flat-file database API SQLite, a new SOAP extension, MySQL Improved, and a lot more.
  + PHP 5.2.9 new release
* **PHP Operators**
  + Arithmetic Operators (+,-,\*,/,%,++,--)
  + Assignment Operators (+=,-=,\*=,/=,%=,.=,=)
  + Comparison Operators (==,!=,<=,>=,<,>,<>)
  + Logical Operators (&&,||,!)

**Conditional Statements**

* Conditional statements are used to perform different actions based on different conditions.

1. If Statement
   * if (condition) code to be executed if condition is true;

Example

* + <?php  
     $d=date("D");  
     if ($d=="Fri") echo "Have a nice weekend!";  
     ?>

1. If …Else statement
   * if (condition)  
       code to be executed if condition is true;  
     else  
       code to be executed if condition is false;

Example

<?php

$d=date("D");  
 if ($d=="Fri")  
   echo "Have a nice weekend!";  
 else  
   echo "Have a nice day!";  
 ?>

* If…Elseif…Else
  + if (condition)  
      code to be executed if condition is true;  
    elseif (condition)  
      code to be executed if condition is true;  
    else  
      code to be executed if condition is false;

Example

<? php

$d=date("D");  
if ($d=="Fri")  
  echo "Have a nice weekend!";  
elseif ($d=="Sun")  
  echo "Have a nice Sunday!";  
else  
  echo "Have a nice day!";

?>

**Switch Statement**

switch (n)  
{  
case label1:  
  code to be executed if n=label1;  
  break;  
case label2:  
  code to be executed if n=label2;  
  break;  
default:  
  code to be executed if n is different from both label1 and label2;  
}

example:

<?php  
switch ($x)  
{

case 1:  
  echo "Number 1";  
  break;  
case 2:  
  echo "Number 2";  
  break;  
case 3:  
  echo "Number 3";  
  break;  
default:  
  echo "No number between 1 and 3";

}  
?>

**PHP Loops**

* Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.
* In PHP, we have the following looping statements:
* ***while*** - loops through a block of code while a specified condition is true
* ***do...while*** - loops through a block of code once, and then repeats the loop as long as a specified condition is true
* ***for*** - loops through a block of code a specified number of times
* ***foreach*** - loops through a block of code for each element in an array

**The while Loop**

* The while loop executes a block of code while a condition is true.

### Syntax

|  |
| --- |
| while (condition)   {   code to be executed;   } |

### Example

* + <?php  
    $i=1;

echo "The numbers are… ";  
while($i<=5)  
  {

echo $i ;  
   $i++; } ?>

Output:

The numbers are 1 2 3 4 5

|  |
| --- |
|  |

**The do...while Statement**

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

### Syntax

|  |
| --- |
| * do   {   code to be executed;   } while (condition); |

### Example

<?php  
$i=1;

echo "The numbers are :";  
do  
  {  
  $i++;  
  echo $i;  
  } while ($i<=5); ?>

Output:

The numbers are: 2 3 4 5 6

**The for Loop**

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

### Syntax

|  |
| --- |
| * for (init; condition; increment)   {   code to be executed;   } |

Parameters:

* init: Mostly used to set a counter (but can be any code to be executed once at the beginning of the loop)
* condition: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
* increment: Mostly used to increment a counter (but can be any code to be executed at the end of the loop)

Note: Each of the parameters above can be empty, or have multiple expressions (separated by commas).

### Example

|  |
| --- |
| <?php  echo "The numbers are:"; for ($i=1; $i<=5; $i++)   {  echo  $i;   } ?> |

Output: The numbers are :1 2 3 4 5

**The foreach Loop**

* The foreach loop is used to loop through arrays.

### Syntax

|  |
| --- |
| * foreach ($array as $value)   {   code to be executed;   } |

* For every loop iteration, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop iteration, you'll be looking at the next array value.

### Example

* <?php  
  $x=array("one","two","three");  
  foreach ($x as $value)  
    {   echo $value . "<br />";   }  
  ?>

|  |
| --- |
|  |

Output:

one  
two  
three

**ARRAYS**

* A variable is a storage area holding a number or text. The problem is, a variable will hold only one value. An array is a special variable, which can store multiple values in one single variable. If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

|  |
| --- |
| * $cars1="Saab";  $cars2="Volvo";  $cars3="BMW"; |

* An array can hold all your variable values under a single name. And you can access the values by referring to the array name. Each element in the array has its own index so that it can be easily accessed.
* In PHP, there are three kind of arrays:
  + Numeric array - An array with a numeric index
  + Associative array - An array where each ID key is associated with a value
  + Multidimensional array - An array containing one or more arrays

**Numeric Arrays**

* A numeric array stores each array element with a numeric index. There are two methods to create a numeric array.
* In the following example the index are automatically assigned (the index starts at 0):
* $cars=array("Saab","Volvo","BMW","Toyota");
* In the following example we assign the index manually:
  + $cars[0]="Saab";  
    $cars[1]="Volvo";  
    $cars[2]="BMW";  
    $cars[3]="Toyota";

|  |
| --- |
|  |

### Example

* + <?php  
    $cars[0]="Saab";  
    $cars[1]="Volvo";  
    $cars[2]="BMW";  
    $cars[3]="Toyota";   
    echo $cars[0] . " and " . $cars[1] . " are Swedish cars.";  
    ?>

The code above will output:

* + Saab and Volvo are Swedish cars

**Associative Arrays**

* An associative array, each ID key is associated with a value. When storing data about specific named values, a numerical array is not always the best way to do it. With associative arrays we can use the values as keys and assign values to them.

### Example 1

$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);

|  |
| --- |
|  |

### Example 2

$ages['Peter'] = "32";  
$ages['Quagmire'] = "30";  
$ages['Joe'] = "34";

The ID keys can be used in a script:

|  |
| --- |
| <?php $ages['Peter'] = "32"; $ages['Quagmire'] = "30"; $ages['Joe'] = "34";  echo "Peter is " . $ages['Peter'] . " years old."; ?> |

The code above will output:

Peter is 32 years old.

**Multidimensional Arrays**

* In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

### Example

$families = array (   "Griffin"=>array   (“Peter”, “Lois”, “Megan”),

  "Quagmire"=>array  ( "Glenn"  ),

 "Brown"=>array ( "Cleveland", "Loretta", "Junior" ) );

|  |
| --- |
|  |

The array above would look like this if written to the output:

Array([Griffin] => Array  (  [0] => Peter  [1] => Lois  [2] => Megan )

[Quagmire] => Array  (  [0] => Glenn  )

[Brown] => Array  (  [0] => Cleveland  [1] => Loretta  [2] => Junior  )

**PHP Functions**

* A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.
* A function has two parts −
* Creating a PHP Function
* Calling a PHP Function
* A function will be executed by a call to the function. You may call a function from anywhere within a page.

Create a PHP Function

### Syntax

|  |
| --- |
| * function functionName() { code to be executed; } |

**PHP function guidelines:**

* Give the function a name that reflects what the function does
* The function name can start with a letter or underscore (not a number)

### Example

* <?php  
  function writeName()  
  {  
  echo "Kai Jim Refsnes";  
  }  
  echo "My name is ";  
  writeName();
* ?>

Output:

My name is Kai Jim Refsnes

**PHP Functions - Adding parameters**

* To add more functionality to a function, we can add parameters. A parameter is just like a variable. Parameters are specified after the function name, inside the parentheses.

### Example 1

* The following example will write different first names, but equal last name:

|  |
| --- |
| <?php function writeName($fname) { echo $fname . " Refsnes.<br />"; } echo "My name is "; writeName("Kai Jim"); echo "My sister's name is "; writeName("Hege"); echo "My brother's name is "; writeName("Stale"); ?> |

Output:

|  |
| --- |
| My name is Kai Jim Refsnes. My sister's name is Hege Refsnes. My brother's name is Stale Refsnes. |

### Example 2

* The following function has two parameters:
* <?php  
  function writeName($fname,$punctuation)  
  { echo $fname . " Refsnes" . $punctuation . "<br />"; }

echo "My name is "; writeName("Kai Jim",".");  
echo "My sister's name is "; writeName("Hege","!");  
echo "My brother's name is "; writeName("Ståle","?"); ?>

* Output:
* My name is Kai Jim Refsnes.  
  My sister's name is Hege Refsnes!  
  My brother's name is Ståle Refsnes?

**PHP Functions - Return values**

* To let a function return a value, use the return statement.

### Example

<?php

function add($x,$y)  
{  
$total=$x+$y;  
return $total;  
}  
echo "1 + 16 = " . add(1,16);  
?>

Output: 1 + 16 = 17

|  |
| --- |
|  |

**PHP Form Handling**

* The most important thing to notice, when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.

### Example

* The example below contains an HTML form with two input fields and a submit button:

|  |
| --- |
| <html> <body> <form action="welcome.php" method="post"> Name: <input type="text" name="fname" /> Age: <input type="text" name="age" /> <input type="submit" /> </form> </body></html> |

When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called "welcome.php":

"welcome.php" looks like this:

Welcome <?php echo $\_POST["fname"]; ?>!<br />  
You are <?php echo $\_POST["age"]; ?> years old.

Output could be something like this:

Welcome John!  
You are 28 years old.

**Form Validation**

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

**The $\_GET Function**

* The built-in $\_GET function is used to collect values from a form sent with method=" GET ".
* Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

### Example

<form action="welcome.php" method="get">  
Name: <input type="text" name="fname" />  
Age: <input type="text" name="age" />  
<input type="submit" />  
</form>

**When to use method="** GET **"?**

* When using method=" GET " in HTML forms, all variable names and values are displayed in the URL.
* Note: This method should not be used when sending passwords or other sensitive information!
* However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
* Note: The get method is not suitable for very large variable values. It should not be used with values exceeding 2000 characters.

**The $\_POST Function**

* The built-in $\_POST function is used to collect values from a form sent with method="POST".
* Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
* Note: However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post\_max\_size in the php.ini file).

### Example

<form action="welcome.php" method="post">  
Name: <input type="text" name="fname" />  
Age: <input type="text" name="age" />  
<input type="submit" />  
</form>

**When to use method="post"?**

* Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
* However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

**The PHP $\_REQUEST Function**

* The PHP built-in $\_REQUEST function contains the contents of both $\_GET, $\_POST, and $\_COOKIE.
* The $\_REQUEST function can be used to collect form data sent with both the GET and POST methods.

### Example

Welcome <?php echo $\_REQUEST["fname"]; ?>!<br />  
You are <?php echo $\_REQUEST["age"]; ?> years old

**The PHP Date() Function**

* The PHP date() function formats a timestamp to a more readable date and time.
* A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred.

|  |  |
| --- | --- |
| **Syntax date(format,timestamp)** | |
| Parameter | | Description | |
| format | | Required. Specifies the format of the timestamp | |
| timestamp | | Optional. Specifies a timestamp. Default is the current date and time | |

**PHP Date() - Format the Date**

* The required format parameter in the date() function specifies how to format the date/time.
* Here are some characters that can be used:
* d - Represents the day of the month (01 to 31)
* m - Represents a month (01 to 12)
* Y - Represents a year (in four digits)
* A list of all the characters that can be used in the format parameter.
* Other characters, like"/", ".", or "-" can also be inserted between the letters to add additional formatting:
* <?php  
  echo date("Y/m/d") . "<br />";  
  echo date("Y.m.d") . "<br />";  
  echo date("Y-m-d") ?>
* The output of the code above could be something like this:
* 2020/11/13  
  2020.11.13  
  2020-11-13

**PHP Date() - Adding a Timestamp**

* The optional timestamp parameter in the date() function specifies a timestamp. If you do not specify a timestamp, the current date and time will be used.
* The mktime() function returns the Unix timestamp for a date.
* The Unix timestamp contains the number of seconds between the Unix Epoch (January 1 1970 00:00:00 GMT) and the time specified.

### Syntax for mktime()

mktime(hour,minute,second,month,day,year,is\_dst)

* To go one day in the future we simply add one to the day argument of mktime():
* <?php  
  $tomorrow = mktime(0,0,0,date("m"),date("d")+1,date("Y"));  
  echo "Tomorrow is ".date("Y/m/d", $tomorrow);  
  ?>
* The output : Tomorrow is 2020/11/14

**What is a Cookie?**

* A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

**To Create a Cookie**

* The setcookie() function is used to set a cookie.
* Note: The setcookie() function must appear BEFORE the <html> tag.

### Syntax

|  |
| --- |
| * setcookie(name, value, expire, path, domain); |

### Example 1

* In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:
* <?php  
  setcookie("user", "Alex Porter", time()+3600);  
  ?>  
  <html>
* Note: The value of the cookie is automatically URL encoded when sending the cookie, and automatically decoded when received (to prevent URL encoding, use setrawcookie() instead).

### Example 2

You can also set the expiration time of the cookie in another way. It may be easier than using seconds.

|  |
| --- |
| * <?php $expire=time()+60\*60\*24\*30; setcookie("user", "Alex Porter", $expire); ?> <html> |

In the example above the expiration time is set to a month (60 sec \* 60 min \* 24 hours \* 30 days).

**Retrieve a Cookie Value**

The PHP $\_COOKIE variable is used to retrieve a cookie value.   
In the example below, we retrieve the value of the cookie named "user" and display it on a page:

* <?php  
  // Print a cookie  
  echo $\_COOKIE["user"];  
  // A way to view all cookies  
  print\_r($\_COOKIE);  
  ?>

|  |
| --- |
|  |

In the following example we use the isset() function to find out if a cookie has been set:

* <html><body>  
  <?php  
  if (isset($\_COOKIE["user"]))  
    echo "Welcome " . $\_COOKIE ["user"] . "!<br />";  
  else  
    echo "Welcome guest!<br />";  
  ?></body></html>

**Delete a Cookie**

When deleting a cookie you should assure that the expiration date is in the past.

* Delete example:

|  |
| --- |
| * <?php // set the expiration date to one hour ago setcookie("user", "", time()-3600); ?> |

**PHP Session Variables**

* When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.
* A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If you need a permanent storage you may want to store the data in a database.
* Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

**Starting a PHP Session**

* Before you can store user information in your PHP session, you must first start up the session.
* The session\_start() function must BEFORE the appear <html> tag:
* <?php session\_start(); ?>  
  <html><body>  
  </body></html>
* The code above will register the user's session with the server, allow you to start saving user information, and assign a UID for that user's session.

**Storing a Session Variable**

* <?php  
  session\_start();  
  // store session data  
  $\_SESSION['views']=1;  
  ?>
* <html>

<body>  
<?php  
//retrieve session data  
echo "Pageviews=". $\_SESSION['views'];  
?>

</body>

</html>

**Destroying a Session**

If you wish to delete some session data, you can use the unset() or the session\_destroy() function.

The **unset()** function is used to free the specified session variable:

* <?php  
  unset($\_SESSION['views']);  
  ?>

You can also completely destroy the session by calling the **session\_destroy()** function:

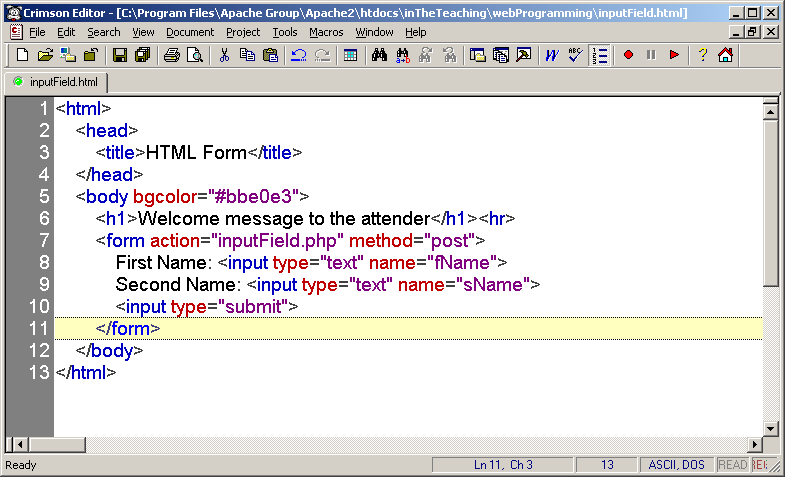
* <?php  
  session\_destroy();  
  ?>

session\_destroy() will reset your session and you will lose all your stored session data.

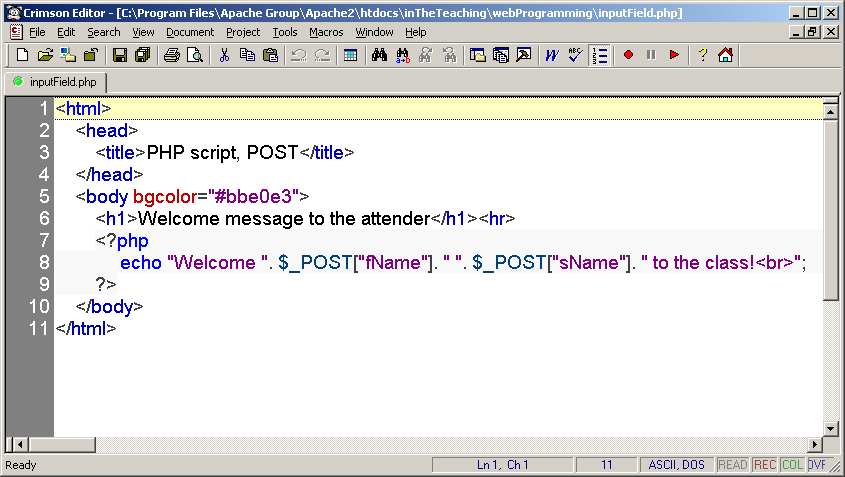
**Reading Data in Web Pages**

* A very common application of PHP is to have HTML form gather information from a website's visitor and then use PHP to do process that information.
* In the classroom PCs the configuration has been implemented so that each PC does have a Web server software (Wamp Server) of it’s own.
* All the HTML files must be saved to the directory C:\wamp\www\YourPersonalFolderName
* This means that http://localhost will display the home page of the local web site.
* The PHP $\_GET and $\_POST variables are used to retrieve information from forms, like user input.
* The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will automatically be available to your PHP scripts.
* The example below contains an HTML form with two text input fields and a submit button.
* Text fields are one line areas that allow the user to input text.
* The name setting adds an internal name to the field so the program that handles the form can identify the fields. The value setting defines what will appear in the box as the default value.
* When a visitor clicks a submit button, the form is sent to the address specified in the action setting of the <form> tag.
* The name setting adds an internal name to the button so the program that handles the form doesn't confuse the button with the other fields.

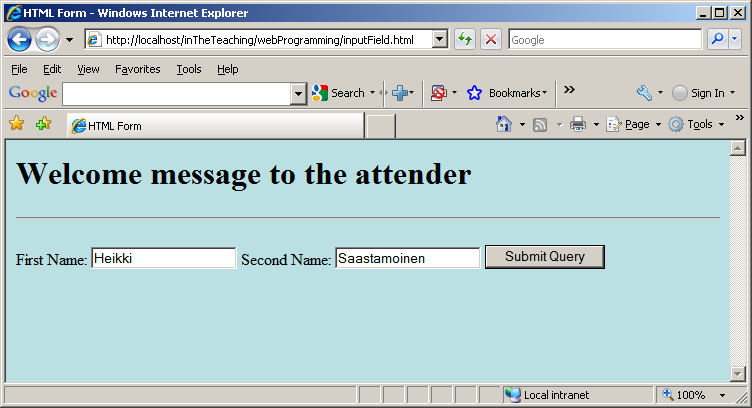
The value setting defines what is written on the button.

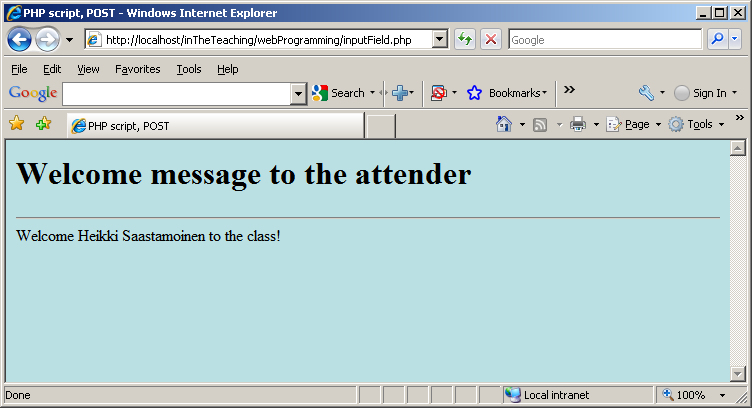


* When a user fills out the form above and click on the submit button, the form data is sent to a PHP file, called “inputField.php":
* " inputField.php" looks like this:



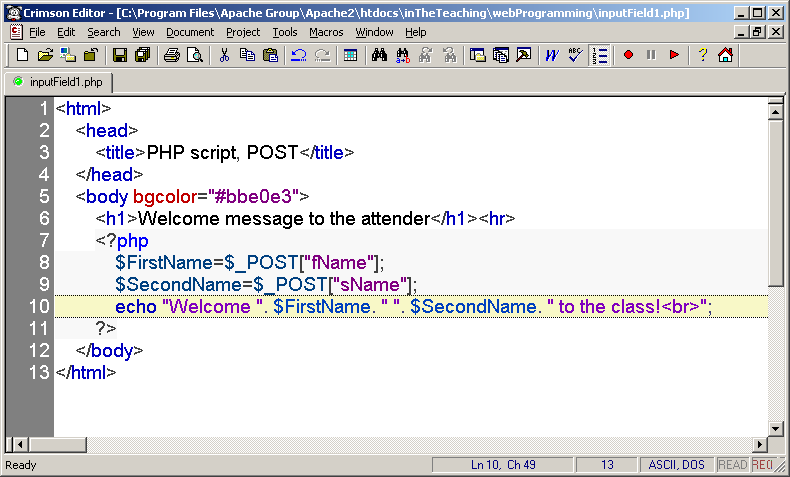
* Be sure to take notice the names of the form data names, as they represent the keys in the "$\_POST" associative array.
* Next the output could be something like this:





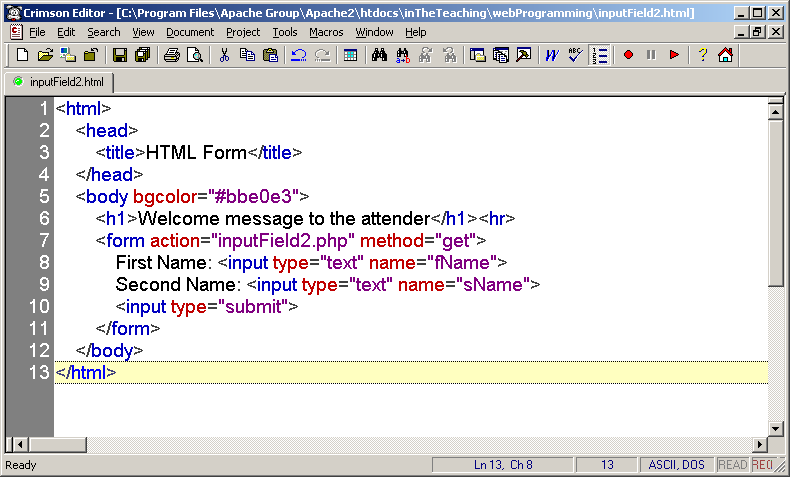
* The built-in $\_POST function is used to collect values from a form sent with method="post".
* Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
* Note: However, there is an 8 Mb max size for the POST method, by default (can be changed by setting the post\_max\_size in the php.ini file).
* When the user clicked the "Submit" button in the previous example, the URL looked like this:

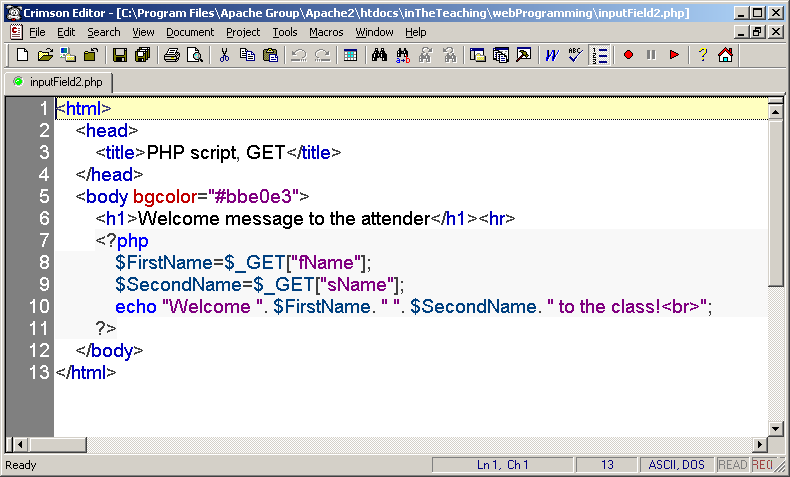
http://localhost/inTheTeaching/webProgramming/inputField.phpThe.

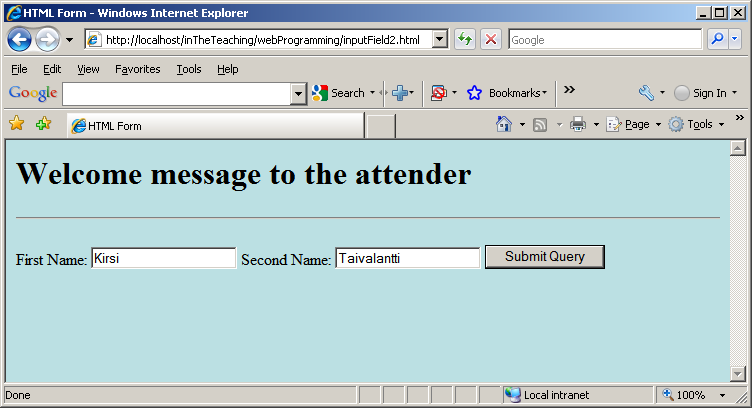
* “inputField.php" file was able to use the $\_POST function to collect form data (the names of the form fields will automatically be the keys in the $\_POST array).
* Another way to get this information would be to create two new variables and set them equal to the values that have been "posted“:
* 

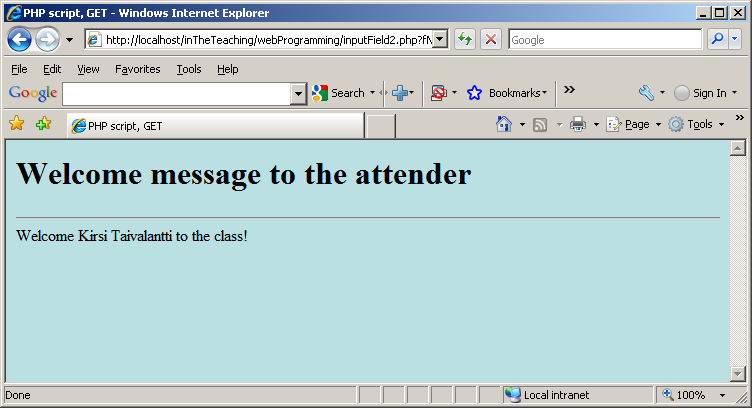
**When to use method="post"?**

* Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.
* However, because the variables are not displayed in the URL, it is not possible to bookmark the page.
* The built-in $\_GET function is used to collect values from a form sent with method="get".
* Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send (max. 100 characters).
* The get method passes the variables along to the “inputField2.php" web page by appending them onto the end of the URL.









* When the user clicked the "Submit" button, the URL sent to the server looked something like this:

http://localhost/inTheTeaching/webProgramming/inputField2.php?fName=Kirsi&sName=Taivalantti.

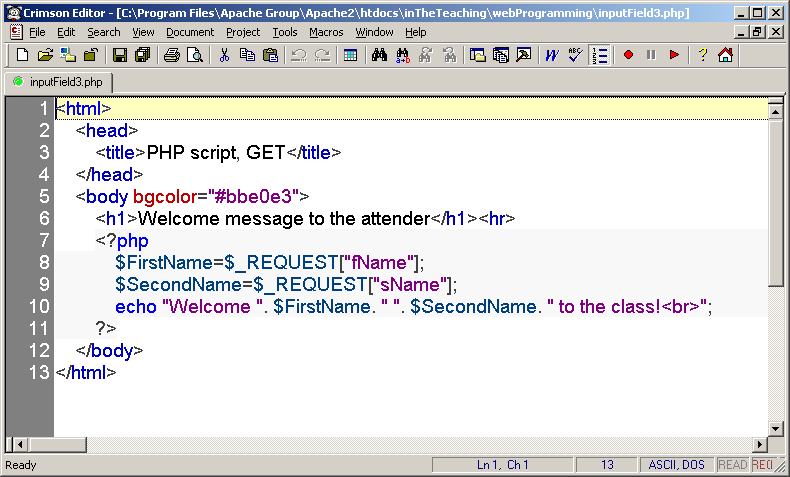
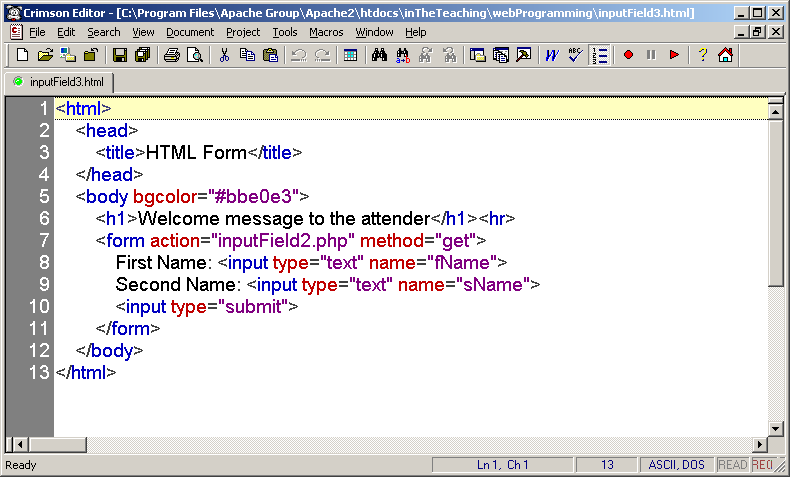
* The question mark "?" tells the browser that the following items are variables.
* The “inputField.php" file was able to use the $\_GET function to collect form data (the names of the form fields will automatically be the keys in the $\_GET array).

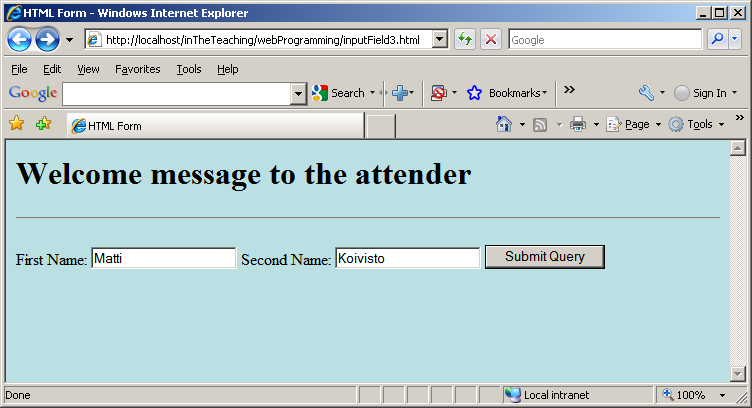
**When to use method="get"?**

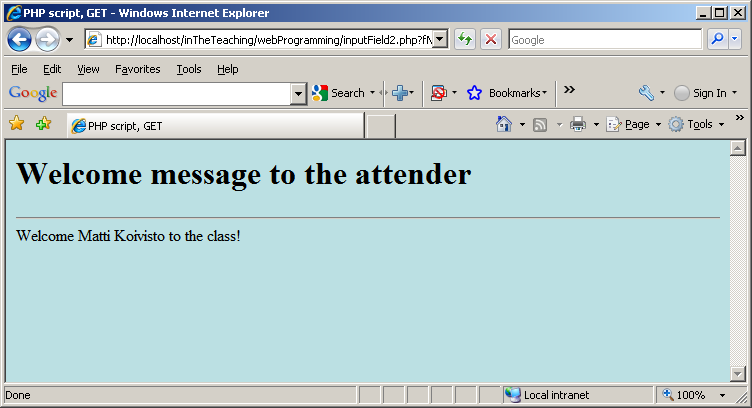
* When using method="get" in HTML forms, all variable names and values are displayed in the URL.
* Note**:** Because of the above behavior this method should not be used when sending passwords or other sensitive information!
* However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.
* Note**:** The get method is not suitable for large variable values; the value cannot exceed 100 characters.

**The PHP $\_REQUEST Function**

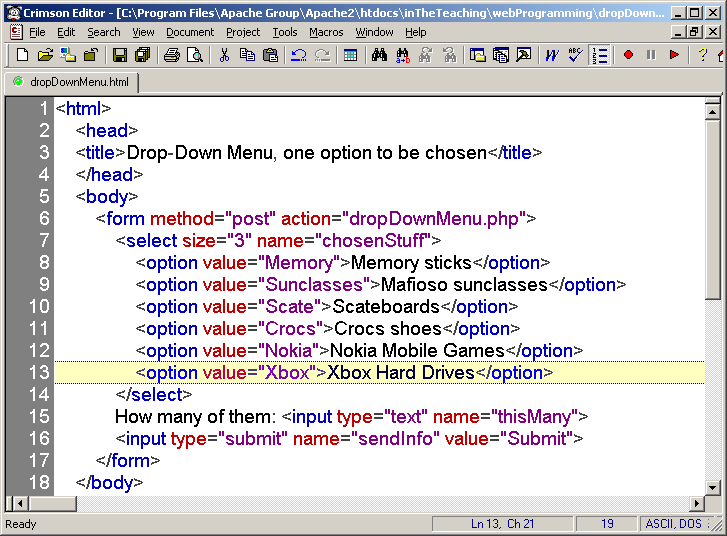
* The PHP built-in $\_REQUEST function contains the contents of both $\_GET, $\_POST, and $\_COOKIE.
* The $\_REQUEST function can be used to collect form data sent with both the GET and POST methods.

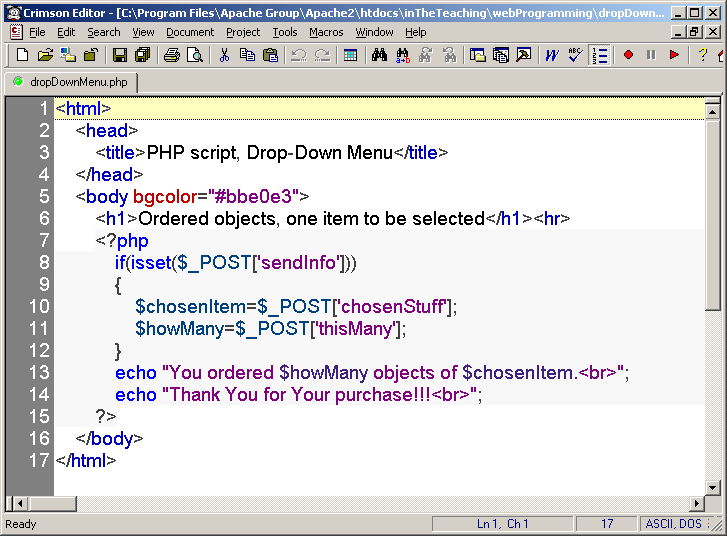


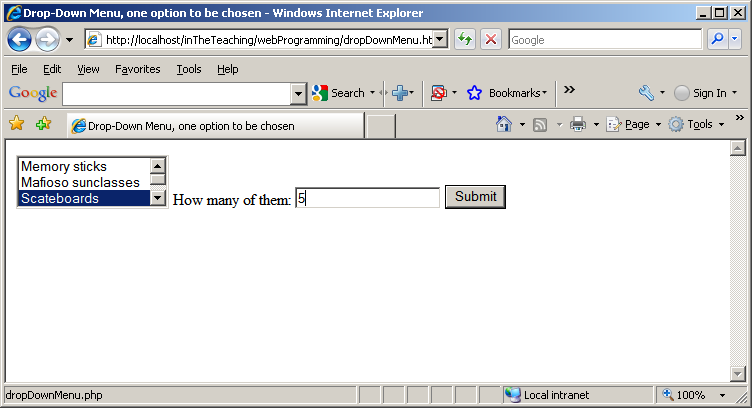


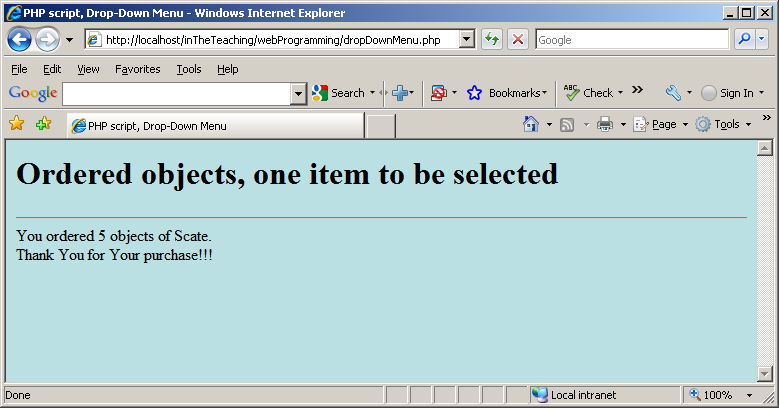


* Next example deals with Drop-Down Menu, only one item can be selected.
* The name setting adds an internal name to the field so the program that handles the form can identify the fields.  
  The value setting defines what will be submitted if the item is selected. This is not always the same as what it says in the menu.
* name = value information will be sent to the script (if the value attribute is used – otherwise the text displayed in the menu will be transmitted):



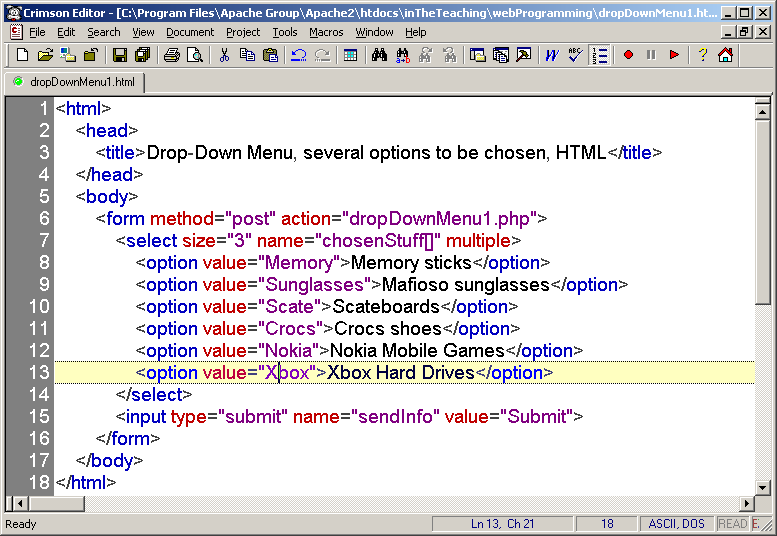


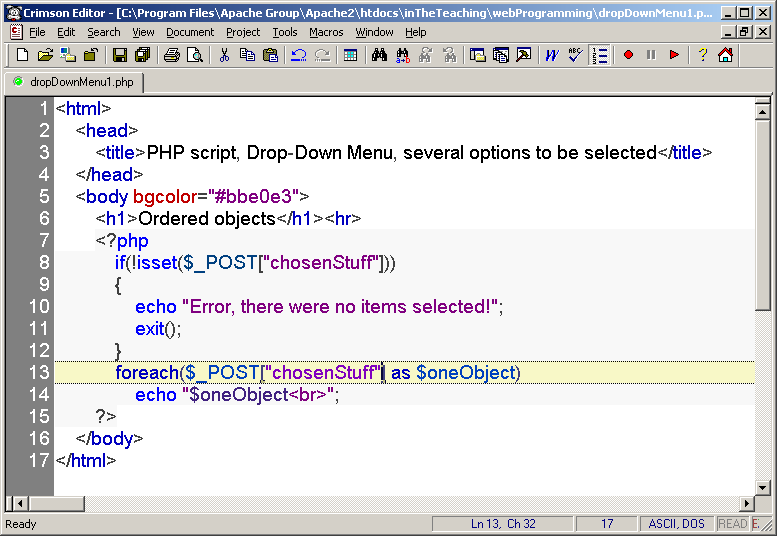


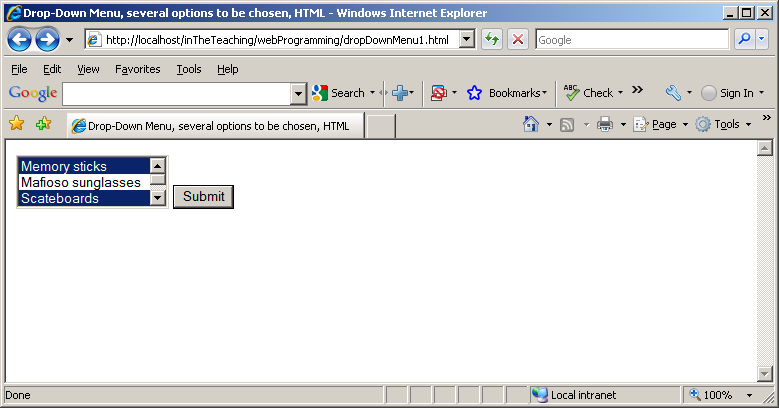


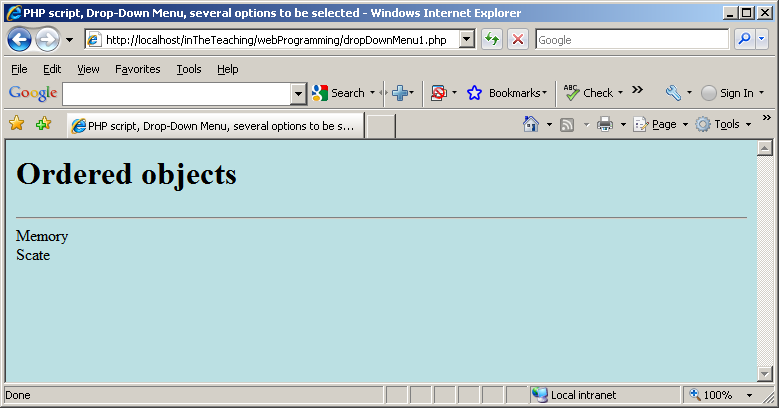
* Again an example with Drop-Down Menu, now several items can be selected.
* The multiple setting will allow for multiple selections if present.

Here we must give the control the name of an array, not just a name.



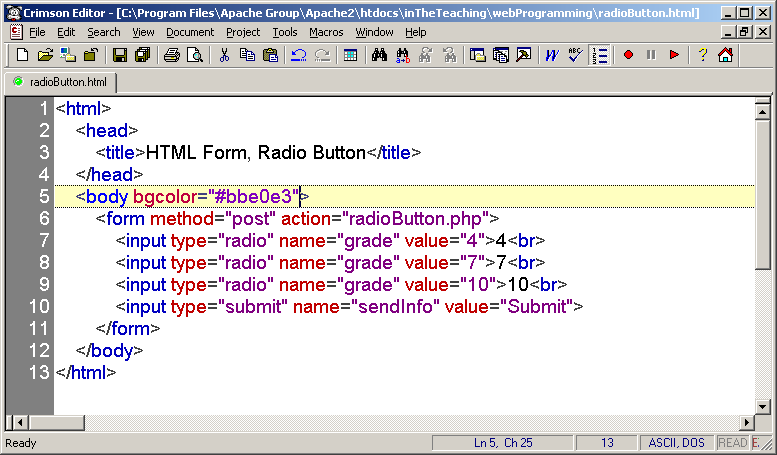


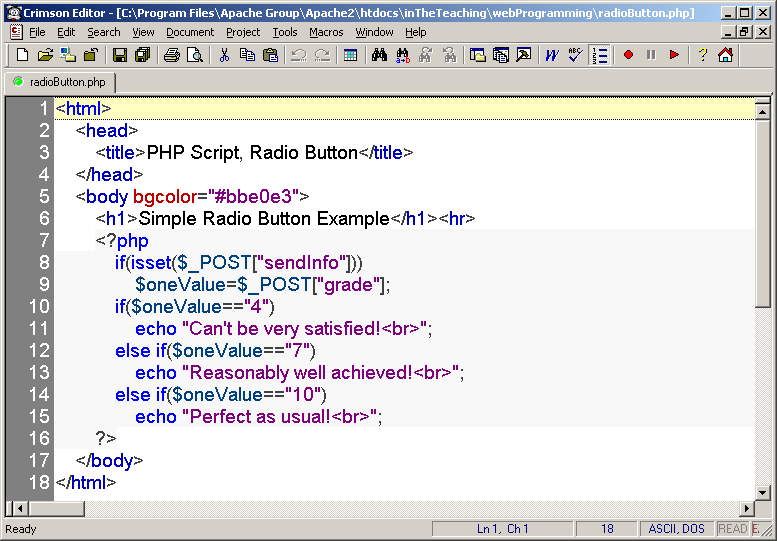


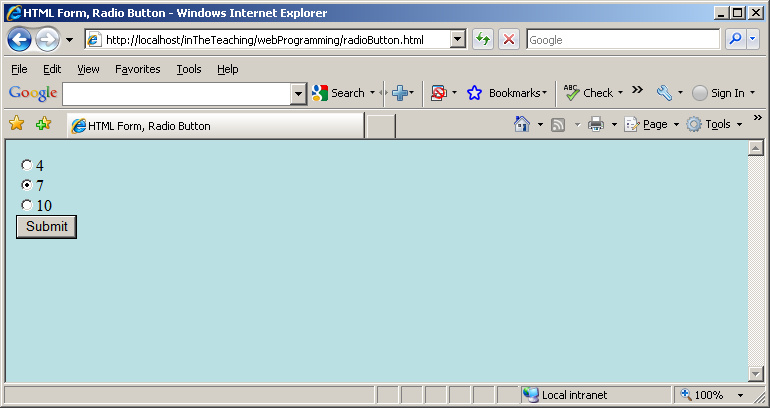


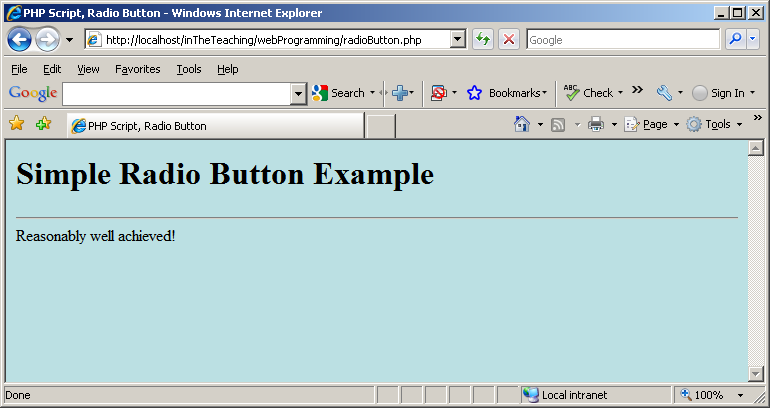
* Radio buttons are used when you want to let the visitor select one - and just one - option from a set of alternatives. If more options are to be allowed at the same time you should use check boxes instead.
* The name setting tells which group of radio buttons the field belongs to. When you select one button, all other buttons in the same group are unselected.  
  If you couldn't define which group the current button belongs to, you could only have one group of radio buttons on each page.

The value setting defines what will be submitted if checked.

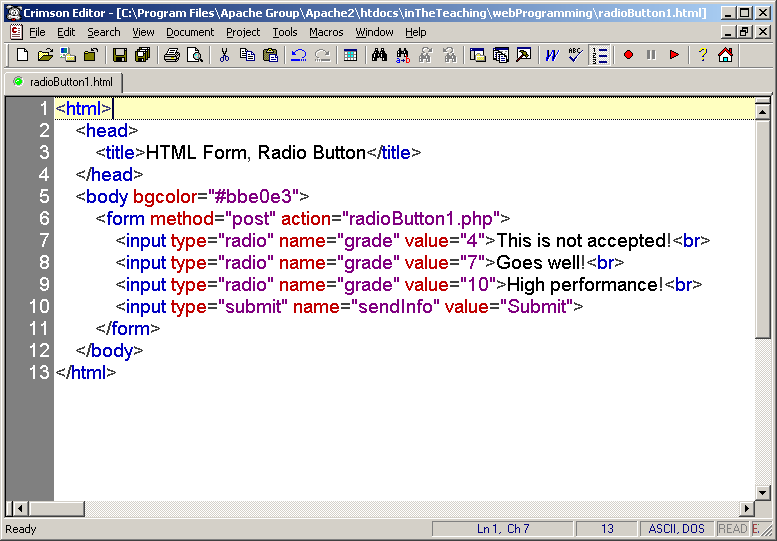


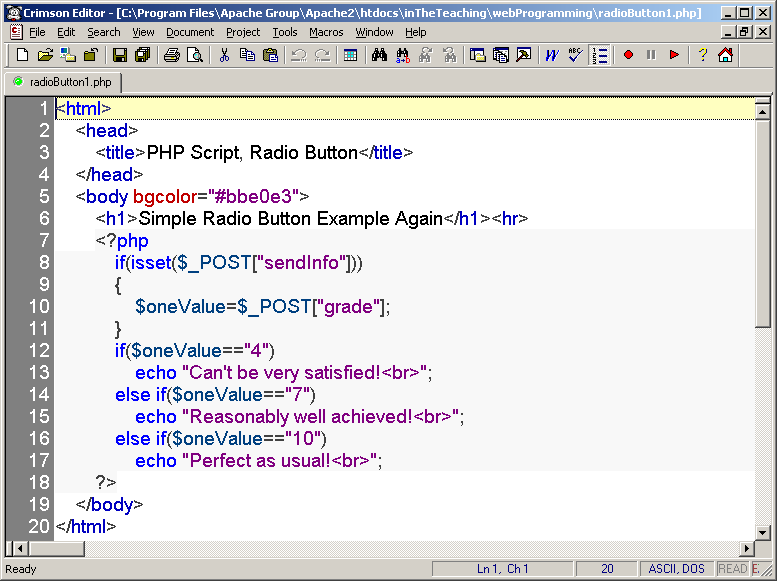


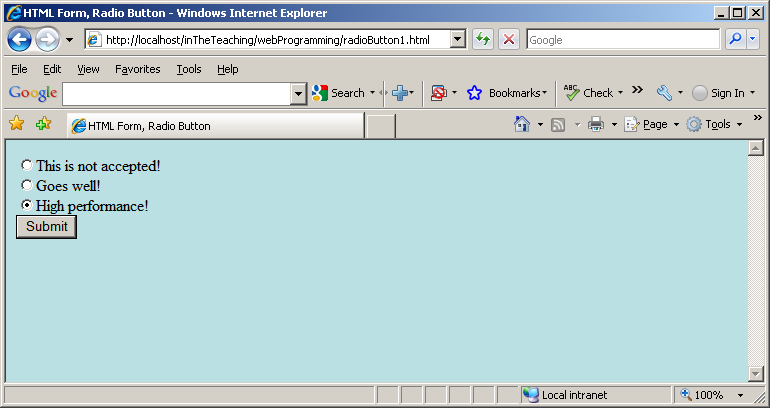


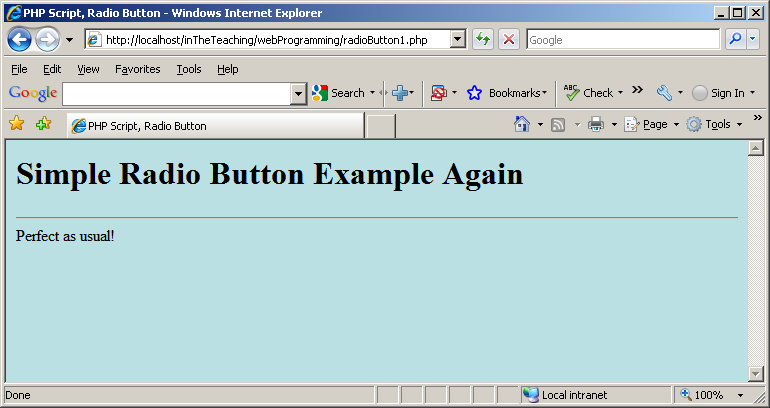


* Next example is only slightly modified compared to the previous one:

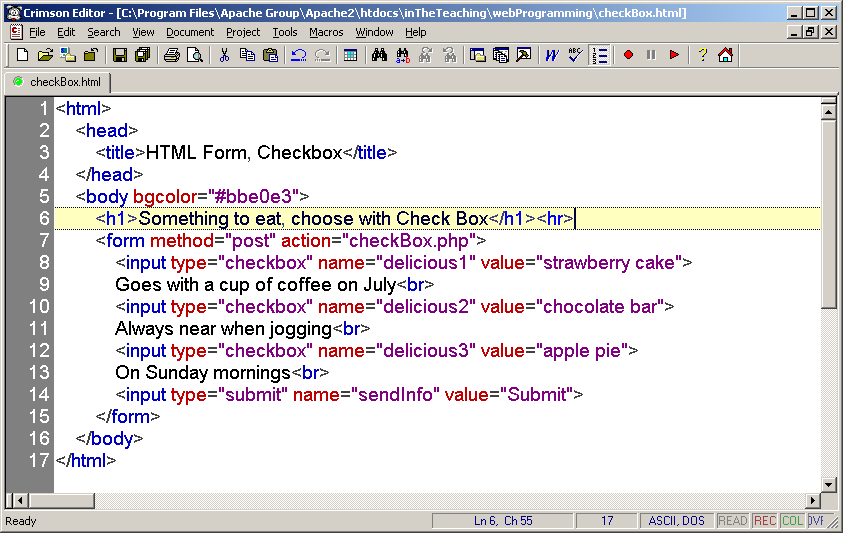


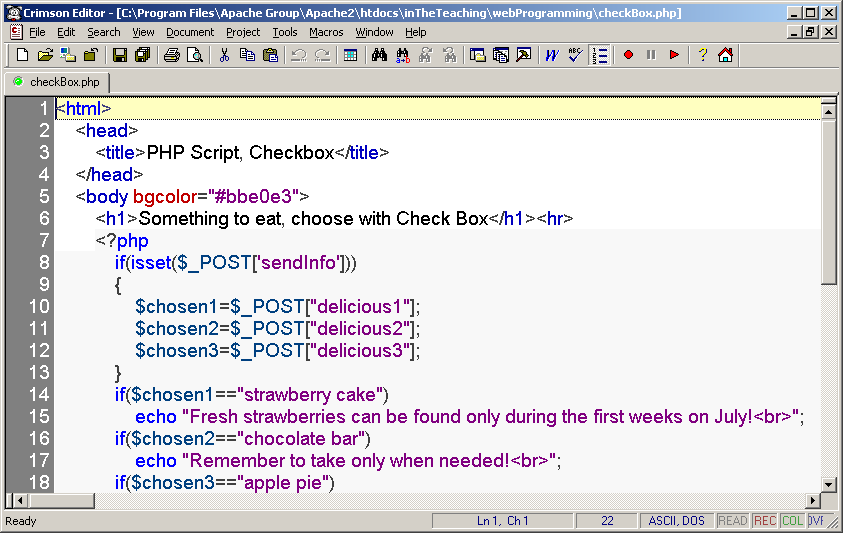


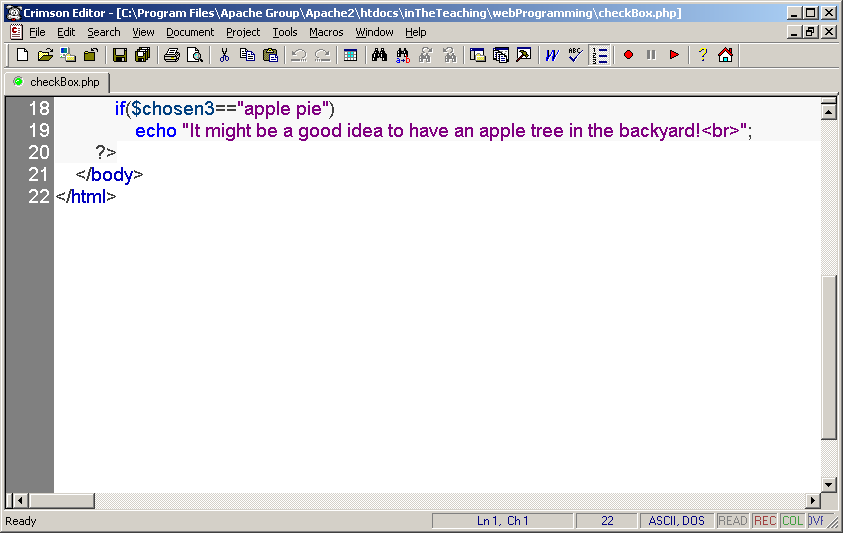


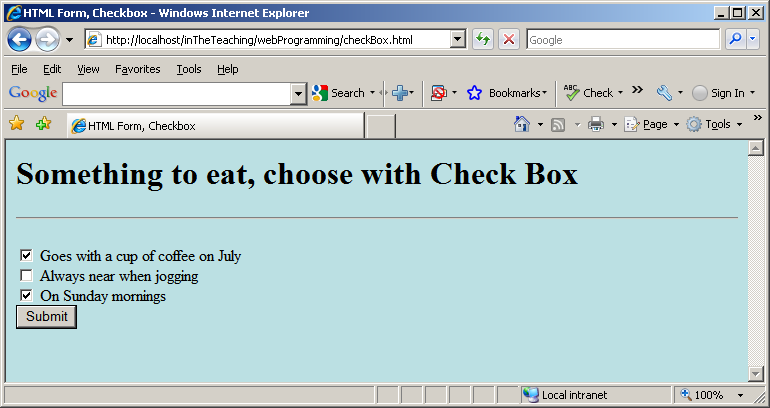


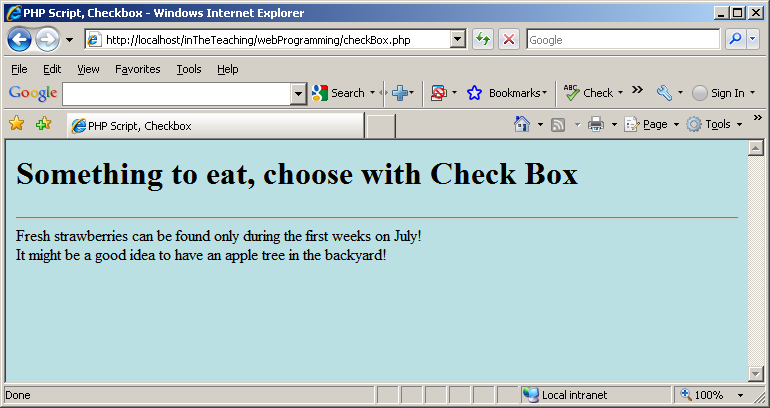
* Check boxes are used when you want to let the visitor select one or more options from a set of alternatives.
* The name setting adds an internal name to the field so the program that handles the form can identify the fields.
* The value setting defines what will be submitted if checked.



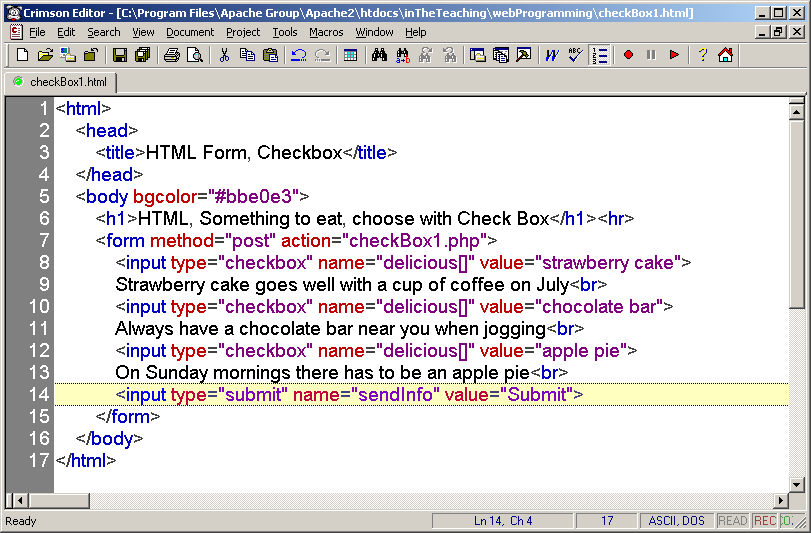


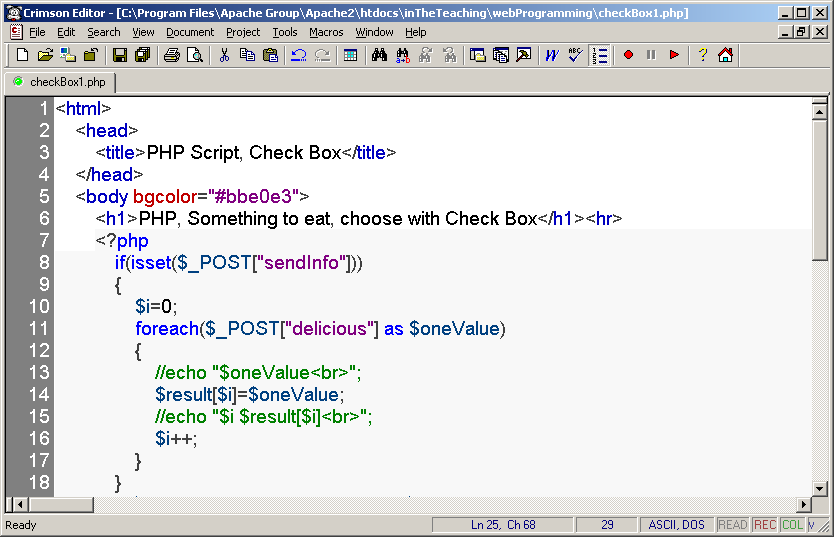


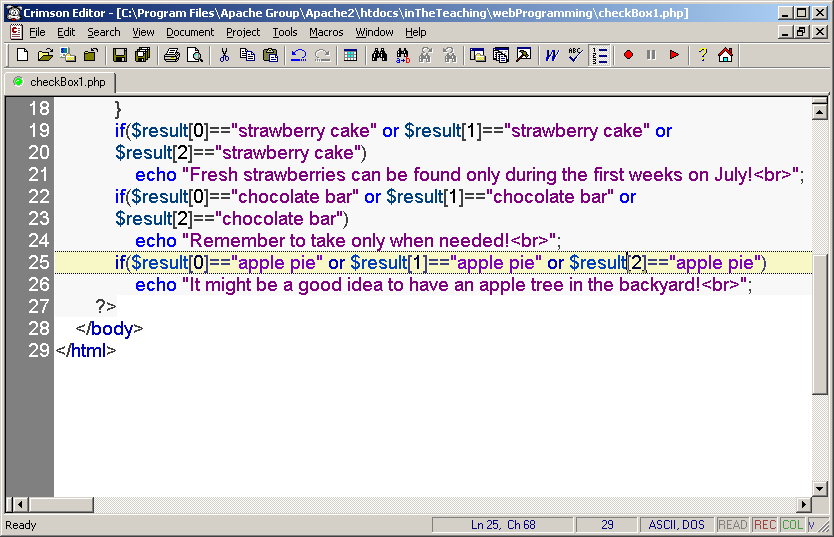


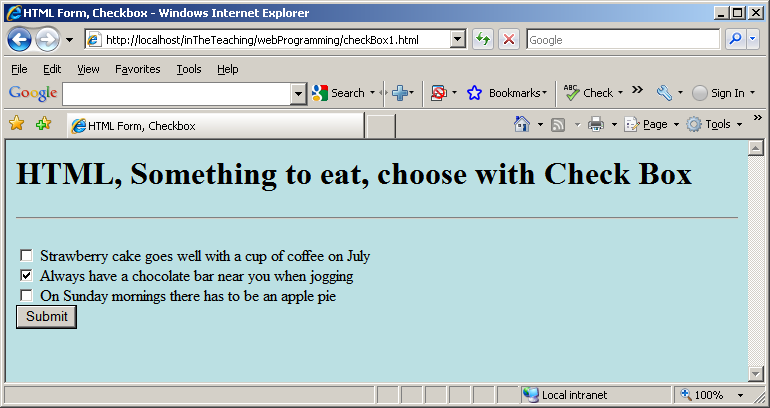


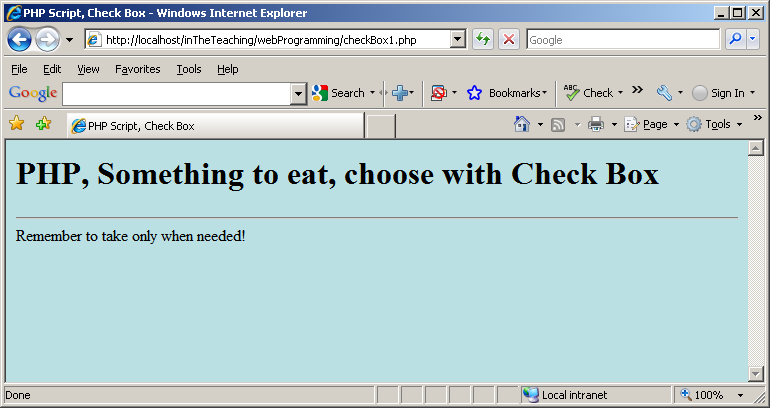
* A minor modification to the previous one, let us give the name setting the name of an array:





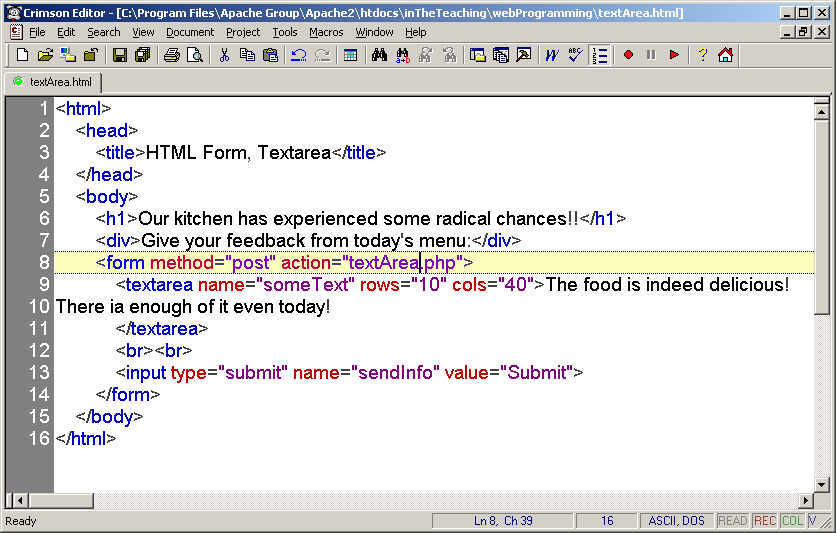


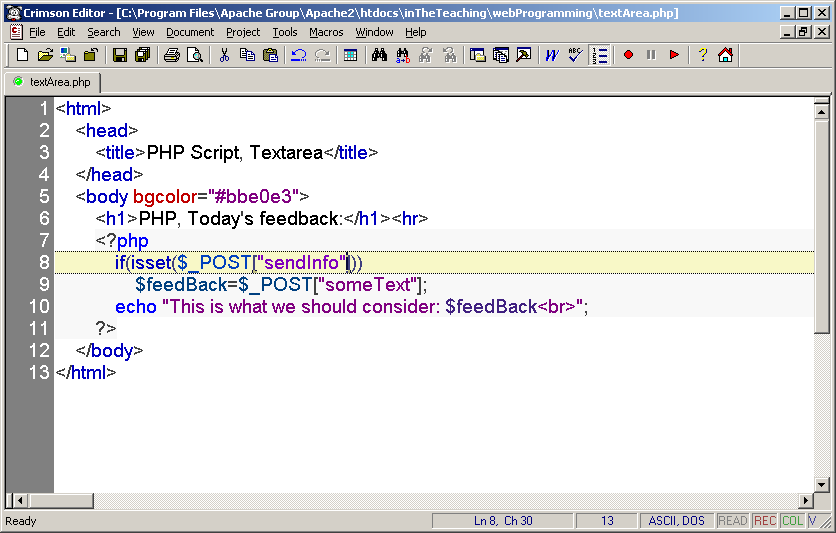




* Text areas are text fields that can span several lines.
* Unlike most other form fields, text areas are not defined with an <input> tag.
* Instead you enter a <textarea> tag where you want the text area to start and a closing </textarea> tag where you want the area to end.
* Everything written between these tags will be presented in the text area box.

The name setting adds an internal name to the field so the program that handles the form can identify the fields.





**Database Connectivity**

***HTML Code to create form***

<html>

<body>

<form name=”form1”method="post">

<table>

<tr> <td>Prouct No</td><td> <input type="text" name="txtpid"><td> </tr>

<tr> <td>Prouct Name</td><td><input type="text" name="txtpname"><td> </tr>

<tr> <td>Quntity</td><td> <input type="text" name="txtqty"><td> </tr>

<tr> <td>Unit Price</td><td> <input type="text" name="txtup"><td> </tr>

<tr><td><input type="submit" name="btn1" value="Save"></td>

<td><input type="submit" name="btn2" value="Update"></td> </tr>

<tr><td><input type="submit" name="btn3" value="Delete"></td>

<td><input type="submit" name="btn4" value="Show"></td></tr>

</form>

<?php

***//Code to insert command***

if(isset($\_POST['btn1']))

{

$pid=$\_POST['txtpid'];

$pname=$\_POST['txtpname'];

$qty=$\_POST['txtqty'];

$up=$\_POST['txtup'];

$db=MySql\_Connect("localhost","root");

MySql\_Selectdb("asd");

$q="insert into tblprdct values ('$pid','$pname','$qty','$up')";

if(MySql\_Query($q))

echo "new row added";

}

**//Code for Update command**

if(isset($\_POST['btn2']))

{

$pid=$\_POST['txtpid'];

$pname=$\_POST['txtpname'];

$qty=$\_POST['txtqty'];

$up=$\_POST['txtup'];

$db=MySql\_Connect("localhost","root");

MySql\_Selectdb("asd");

$q="update prdct set pnme='$pname',qty='$qty', up='$up' where pno='$pid'";

if(MySql\_Query($q))

echo "row updated";

}

**//Code for Delete command**

if(isset($\_POST['btn3']))

{

$pid=$\_POST['txtpid'];

$db=MySql\_Connect("localhost","root");

MySql\_Selectdb("asd");

$q="delete from prdct where pno='$pid'";

if(MySql\_Query($q))

echo "row deleted";

}

**//Code for Select command**

if(isset($\_POST['btn4']))

{

$db=MySql\_Connect("localhost","root");

MySql\_Selectdb("asd");

echo "<table border=1>";

echo "<tr>";

echo "<th>Product No </th>" ;

echo "<th> Product Name </th>" ;

echo "<th> Quantity </th>" ;

echo"<th>Unit Price</th>"

echo "</tr>";

$q="Select \* from prdct ";

$r=MySql\_Query($q);

while ($row=MySql\_fetch\_array($r))

{

echo "<tr>";

echo "<td>".$row['pno']."</td>";

echo "<td>".$row[1]."</td>";

echo "<td>".$row[2]."</td>";

echo "</tr>";

} }

?> </body> </html>