CS 85: PHP Programming

Santa Monica College Computer Science & Information Systems Dept.

Module 5: Forms

Web Forms

Processing Forms

Submitting Forms

POST & GET Methods

* filter\_var()

Forms are, by far, the most common way of interacting with PHP. As we mentioned before, it is recommended that you have basic knowledge of HTML and CSS. If you don't, just head to the HTML Wikibook (https://en.wikibooks.org/wiki/HTML) or W3School HTML Tutorial (<https://www.w3schools.com/html/>) for a refresher.

# HyperText Markup Language/Forms

HTML forms are an easy way to gather data from the end users. Forms are commonly used for account registration, e-commerce site, guestbook site, content management systems, search boxes, etc. User data can be entered using a variety of input types, from textboxes, drop down menu, radio button controls, etc. Processing the user input requires a server-side scripting language such as PHP or in some cases when limited interaction is to be provided within a single page a client-side scripting language such as JavaScript. In this course we will focus on server side processing using PHP.

# Form Setup

To create a form the HTML tag <form> with the attributes action and method is specified as follows:

<form method="post" action="action.php"> <!-- Your form here --> </form>

Once the user clicks "Submit", the form body is sent to the PHP script action.php for processing. The values entered in all fields in the form are stored in the associative array variables $\_GET or $\_POST, depending on the method used to submit the form.

The difference between the GET and POST methods is that the GET submits all the values in the URL appended to the action URL an as query string parameters. While POST submits values transparently through HTTP headers.

$\_GET and $\_POST are Superglobal arrays discussed in the next section.

Here is a simple form. Notice how it only contains HTML code and is saved with a .html file extension.

<!DOCTYPE **html**>  
<**html**>  
<**body**>  
  
<**h2**>HTML Forms</**h2**>  
  
<**form id="sampleform" action="action\_page.php" method="get"**>  
 <**fieldset**>  
 <**legend**>Personal Information</**legend**>  
 <**label for="firstname"**>First name</**label**>  
 <**input type="text" name="firstname" id="firstname" value="Mickey"**>  
 <**br**>  
 <**label for="lastname"**>Last name</**label**>  
 <**input type="text" name="lastname" id="lastname" value="Mouse"**>  
 <**br**><**br**>  
 <**input type="submit" value="Submit"**>  
 </**fieldset**>  
</**form**>  
  
<**p**>If you click the "Submit" button, the form-data will be sent to a page called "action\_page.php".</**p**>  
  
</**body**>  
</**html**>

Form Explanation:

id

The name of the form or control.

action

The URL of a server-side script which can process the data.

method

The method used to send the information. Two methods are supported, POST and GET. POST is the preferred method except for simple searches which generally use GET. Use with server-side languages.

fieldset

Form controls are normally contained in a fieldset element. Complex forms may have multiple fieldsets. Fieldsets can contain other fieldsets.

legend

Each fieldset begins with a legend element. The content of the element is used as a title placed in the border of the fieldset.

label for=""

A label for is a single form control. The value of the for attribute must match the id attribute of a form control in the same form.

input type="" name ="" id=""

Various types of input controls. Supported types are - submit, text, password, checkbox, radio, reset, file, hidden, image and button. The name Attribute is used by the server to identify which piece of data was entered in a given box on the form. The id attribute is used to match an input with its label. The name and id attributes normally have identical values for text inputs but different values for checkbox and radio inputs.

select

There is also a SELECT element for drop down lists and a TEXTAREA element for multi-line text input.

This simple example uses <br /> tags to force newlines between the different controls. A real-world form would use more structured markup to layout the controls neatly.

# Superglobals

PHP has special built-in variables of data type associative array that store server, script, form, session information. Associative arrays elements are referenced with a string instead of an integer index value. For example $GLOBAL[“varNume”];. These variables are available at all scopes of your script. Meaning the variable is accessible in user defined functions or PHP include/required files without specifically stating ***global $variable*** to access their values. These built-in variables are known as Superglobals or Autoglobals variables.

The PHP Superglobal variables are:

* $GLOBALS
* $\_SERVER : A variable which holds information about headers, paths, and script locations.
* $\_REQUEST : Array of all the elements in the $\_COOKIE, $POST, $\_GET array
* $\_POST : Array of values input into a form using the POST method
* $\_GET : Array of values input into a form using the GET method
* $\_FILES
* $\_ENV
* $\_COOKIE
* $\_SESSION

# $GLOBALS

$GLOBALS is a PHP Superglobal variable which can be called to access global variables anywhere in the PHP script. PHP stores all global variables in an associative array called $GLOBALS[“string”]. The string in $GLOBAL[] is the name of the variable that holds the variable value.

<!DOCTYPE **html**>  
<**html**>  
<**body**>  
  
**<?php**$x = 75;  
$y = 25;  
  
**function** addition() {  
 $GLOBALS[**'z'**] = $GLOBALS[**'x'**] + $GLOBALS[**'y'**];  
}  
  
addition();  
**echo** $z;  
**?>**</**body**>  
</**html**>

# $\_SERVER

$\_SERVER is a PHP super global variable which holds information about the web server, the PHP script executing and PHP engine being used. The $\_SERVER[] variable can be useful to determine if either the POST or GET method was used to submit a form or to create a form that calls back to itself.

|  |  |
| --- | --- |
| $\_SERVER['PHP\_SELF'] | Returns the filename of the currently executing script |
| $\_SERVER['REQUEST\_METHOD'] | Returns the request method used to access the page (such as POST) |
| $\_SERVER['HTTP\_HOST'] | Returns the Host header from the current request |
| $\_SERVER['SCRIPT\_FILENAME'] | Returns the absolute pathname of the currently executing script |

The example below shows how to use some of the elements in $\_SERVER:

<!DOCTYPE **html**>  
<**html**>  
<**body**>  
  
**<?php  
echo** $\_SERVER[**'PHP\_SELF'**];  
**echo "<br>"**;  
**echo** $\_SERVER[**'SERVER\_NAME'**];  
**echo "<br>"**;  
**echo** $\_SERVER[**'HTTP\_HOST'**];  
**echo "<br>"**;  
**echo** $\_SERVER[**'HTTP\_REFERER'**];  
**echo "<br>"**;  
**echo** $\_SERVER[**'HTTP\_USER\_AGENT'**];  
**echo "<br>"**;  
**echo** $\_SERVER[**'SCRIPT\_NAME'**];  
**?>**</**body**>  
</**html**>

# Method

Method is the <form> element attribute that defines how the form will be submitted to the server. The method attribute can only be GET or POST. The difference between the GET and POST methods is that GET method submits all the values in the URL appended to the action URL an as query string parameters. The POST method submits values transparently through HTTP headers. A general rule of thumb is if you are submitting sensitive data, use POST. POST forms usually provide more security

When the form is submitted to the server for processing, the form body is sent to the PHP script for processing. All fields in the form are stored in either the Superglobal array $\_GET or $\_POST, depending on the method used to submit the form. The form input fields name will be Superglobal array $\_GET or $\_POST key and the user enter data will be the be Superglobal array value.

**GET Method**

As explained, any data sent via a PHP form using the GET method, will converted into values/data and be added into a query string within URL known as URL Encoding. This URL encoding hold both action script link separated by a ? and the encoded form field value/data separated by the & character.

**http://www.asite.com/action.php?name=mary&email=mary@gmail.com&contact=1234567896**

Here is an example of an HTML form using the method=”get”.

Side Note:

Remember the htmlspecialchars() function?  
The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

Filename: welcome\_form.html

<!DOCTYPE **html**>  
<**html**>  
 <**title**>HTML GET Method</**title**>  
<**body**>  
  
<**h2**>HTML Forms</**h2**>  
  
<**form action="welcome\_get.php" method="get"**>  
 Name: <**input type="text" name="name"**><**br**>  
 E-mail: <**input type="text" name="email"**><**br**>

Contact: <**input type="text" name="contact"**><**br**>  
 <**input type="submit"**>  
</**form**>  
  
<**p**>If you click the "Submit" button, the form-data will be sent to a page called "welcome\_get.php".</**p**>  
  
</**body**>  
</**html**>

Filename: welcome\_get.php

<!DOCTYPE **html**>  
<**html**>  
 <**title**>HTML GET Method</**title**>  
<**body**>  
  
<**h2**>HTML Forms</**h2**>  
  
**<?php  
if**( $\_GET[**"name"**] || $\_GET[**"email"**] || $\_GET[**"contact"**])  
{  
 **echo "Welcome: "**. $\_GET[**'name'**]. **"<br />"**;  
 **echo "Your Email is: "**. $\_GET[**"email"**]. **"<br />"**;  
 **echo "Your Mobile No. is: "**. $\_GET[**"contact"**];  
}  
**?>**</**body**>  
</**html**>

Within the same PHP script an if/else statement can be used to determine if the user has already filled in the form and clicked submit. If this event has already occurred then the $\_GET associative array will have a value for each of the input fields (ie **$\_GET["name"]**).

<**input type="submit" name="submit" value="Submit"**>  
**<?php  
if** (**isset**($\_GET[**"submit"**])) {  
 *// process the form contents...*}  
**echo** $\_SERVER[**'SCRIPT\_NAME'**];  
**?>**

**Example POST Method**

Now let’s take a look at an example of a PHP form using the POST method of storing user inputted data. User submitted data in forms using the POST method are not transmitted to the server in the URL but in an associated Superglobal array $\_POST embedded in the HTML header. Each form input field name in the form will automatically become a key in the $\_POST Superglobal array and the user inputted data will be the values of the $\_POST Superglobal array that can be accessed by the action PHP script $\_POST[“fieldName”].

Form using the POST method also has no limit to the amount of data being transmitted to the server. The POST method is the method required for file transfers or binary input to the server for server side processing. For example, uploading images to Facebook to be added to your profile. All done via file upload form and processed on the server side by PHP scripts. Developers prefer POST for sending form data.

Filename:postsample.html  
<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**>HTML FORM POST</**title**>  
</**head**>  
<**body**><**form action="postaction.php" method="post"**>  
 <**label for="name"**>Name: </**label**>  
 <**input type="text" name="name"**></**input**><**br**/>  
 <**label for="email"**>Email: </**label**>  
 <**input type="text" name="email"**></**input**><**br**/>  
 <**label for="contact"**>Contact: </**label**>  
 <**input type="text" name="contact"**></**input**><**br**/>  
 <**input type="submit" name="submit" value="Submit"**></**input**>  
</**form**></**body**>  
</**html**>

**Filename:postaction.html**

**<?php***// comment: checking if form has already been submitted with all fields completed***if** (**isset**($\_POST[**"name"**]) && **isset**($\_POST[**"email"**]) && **isset**($\_POST[**"contact"**])) {  
 **echo "Welcome: "**. $\_POST[**'name'**]. **"<br />"**;  
 **echo "Your Email is: "**. $\_POST[**"email"**]. **"<br />"**;  
 **echo "Your Mobile No. is: "**. $\_POST[**"contact"**];  
}  
**?>**

Within the same PHP script, a condition statement has been defined to check if the $\_POST associative array has entries for name, email and contact. If these array entries exist, then print out “Welcome …. “.

**<?php***//checking if form has already been submitted with all fields completed***if** (**isset**($\_POST[**"name"**]) && **isset**($\_POST[**"email"**]) && **isset**($\_POST[**"contact"**])) {  
 **echo "Welcome: "**. $\_POST[**'name'**]. **"<br />"**;  
 **echo "Your Email is: "**. $\_POST[**"email"**]. **"<br />"**;  
 **echo "Your Mobile No. is: "**. $\_POST[**"contact"**];  
}  
**?>**

**Example: POST Login**

File: loginForm.html

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**>Login Page</**title**>  
</**head**>  
<**body**>  
*<!-- File enterlogin.html -->*<**form method="post" action="login.php"**>  
 Please log in.<**br**/>  
 Username: <**input name="username" type="text"** /><**br** />  
 Password: <**input name="password" type="password"** /><**br**/>  
 <**input name="submit" type="submit"** />  
</**form**>  
</**body**>  
</**html**>

This form should appear similar to the image below.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Please log in. | | | Username: |  | | Password: |  | |  | submit | |  |

The script used to process the loginForm.html file is login.php as defined by the action attribute in the form tag.

**Filename:login.php**

**<?php**$checkUsername = $\_POST[**'username'**];  
$checkedPassword = $\_POST[**'password'**];  
  
**if**($checkUsername == **"spoom"** && $checkedPassword == **"apassword"**) {  
 **echo**(**"Welcome, Spoom."**);  
}  
**else** {  
 **echo**(**"You're not Spoom!"**);  
}  
**?>**

# Form Validation

It is always recommended that ALL USER INPUT BE VALIDATED before that data is used in any way. By validating the user input you are protecting your script and the server your site is being host on from malicious attacks. A simple example of malicious code being entered is if a hacker entered ***<script>location.href('http://www.abadsite.com')</script>*** in a form field of a guestbook. A guest book normally display all guest entered greeting. Now with the malicious being display on the guestbook comment page, all future guestbook visitors will automatically be redirected to [www.abadsite.com](http://www.abadsite.com) simple because the malicious JavaScript code has been entered form field and the PHP did not validate/check if the user input is validate. This type of cyber attacked is called Cross-Site Scripting (XSS).

Now let’s take a look at the sample login script, but now user input is being checked for special characters with the PHP built-in function htmlspecialchars().

**Filename:login.php**

**<?php**$checkUsername = $\_POST[**'username'**];  
$checkedPassword = $\_POST[**'password'**];  
  
**if**($checkUsername == **"spoom"** && $checkedPassword == **"apassword"**) {  
 **echo**(**"Welcome, Spoom."**);  
}  
**else** {  
 **echo**(**"You're not Spoom!"**);  
}  
**?>**

Now notice how the below code uses htmlspecialchars()in an attempt to clear any malicious HTML code.

**Filename:login.php**

**<?php**$checkUsername = *htmlspecialchars*($\_POST[**'username'**]);  
$checkedPassword = *htmlspecialchars*($\_POST[**'password'**]);  
  
**if**($checkUsername == **"spoom"** && $checkedPassword == **"apassword"**) {  
 **echo**(**"Welcome, Spoom."**);  
}  
**else** {  
 **echo**(**"You're not Spoom!"**);  
}  
**?>**

Let's take a look at that script a little closer.

$\_POST[**'username'**]  
$\_POST[**'password'**]

As you can see, $\_POST is an array, with keys matching the names of each field in the form. For backward compatibility, you can also refer to them numerically, but you generally shouldn't as this method is much clearer.

# Validating Forms

Let’s take a closer look at form validation. There are a few techniques for validating user input into a form. The simplest methods is using the PHP's htmlspecialchars() function which attempts to replace any HTML tags such as < > with the proper HTML code such as &lt and &gt.

Now the code is much safer and can prevents possible attackers from exploiting our code by injecting HTML or Javascript code. If anyone attempts to enter code such as <script>location.href('http://www.abadsite.com')</script> into a form field, what will be used after using the htmlspecialchars() function is &lt;script&gt;location.href('http://www.abadsite.com')&lt;/script&gt;.

Now if you are expecting specific type of data entered by your users, you can attempt to validate even further and check for a character pattern of the user inputted data to ensure it is in fact the input being expected. Let check user input for an email address and user input for a date. We can start by removing any unnecessary character from the variable and then remove any slashes.

If you are expecting a large amount of user input, it would be best to create a function that only validate user input. Here is an example of such a function. Notice how this function take any data passed to it and removed end of line, removes any slashed and then checks for any embedded HTML code and replace it with HTML accepted characters for example < is &lt

|  |
| --- |
| <?php  function checkInput($data)  {  $data = trim($data);  $data = stripslashes($data);  $data = htmlspecialchars($data);  return $data;  }  ?> |

Let’s pass a user inputted field into our checkInput function.

|  |
| --- |
| $checkUsername = checkInput($\_POST['username']); |

# Validating Email and Special Strings

PHP has a built in regular expression function called preg\_match() that can be used to check for validate input strings.

# Validate Email Address

It is common to create an account on a website and be required to enter an email address. But there is no way a company can know if that email address actually exists but it can check that the email string entered does follow a normal email address format which it must contain a @ symbol and have a domain name such as gmail.com. By using preg\_match() your code can search for an @ symbol and a . between the @ and the end of the email address. Read more on preg\_match <http://php.net/manual/en/function.preg-match.php>

|  |
| --- |
| $emailaddress = htmlspecialchars($\_POST['emailaddress']);  if (!preg\_match("/([\w\-]+\@[\w\-]+\.[\w\-]+)/",$email))  {  echo "E-mail address not valid";  } |

The above code would make the e-mail address required.

# Validate URL address

Just like we can use preg\_match() to check an email address format, we can use preg\_match() to check the format of a URL entered into a PHP form field.

|  |
| --- |
| $myURL = htmlspecialchars($\_POST['myWebsite']);  if (!preg\_match("/^(https?:\/\/+[\w\-]+\.[\w\-]+)/i",$myURL))  {  echo “URL address not valid";  } |

# Validate Other

There will be many specifically formatted fields in PHP that will need to be validate for security reasons such as social security number, phone number, street address. Here we can take a quick look at some such cases.

# Digits 0-9 only

Checking if $age is a number

|  |
| --- |
| if (preg\_match("/\D/",$age))  {  echo "Please enter numbers only for Age";  } |

# Letters a-z and A-Z only

Here we are checking if $string is made of letters a-z and A-Z only. No spaces, digits or any other characters is allowed.

|  |
| --- |
| if (preg\_match("/[^a-zA-Z]/",$string))  {  echo "Please enter letters a-z and A-Z only!";  } |

# Anything but whitespace

This code will show an error if $text contains of any whitespace characters (space, tab, newline):

|  |
| --- |
| if (preg\_match("/\s/",$text))  {  echo "Please do not enter any spaces, tabs or new lines!";  } |

# User Data

Once the user submitted data has been validated, what will the data be used for? The possibilities are endless. Web forms are normally made with a specific purpose. Common form uses include shopping carts, account creation, guestbook, survey, Canvas quizzes. One of the most widely used form are the account registration form. The account information such as first name, last name, email address, username, password and phone number is normally validated and then entered into a back end database to be called in the future to authenticate accounts. Another common form is the email form. The email form collects the user’s name and email address to then email information related to the company directly to the user inbox.

# HTML form code

Let's use the form we started with in this tutorial and just add a few more fields to make it more interesting. In this example we will make fields "Your name", "Subject", "E-mail" and "Comments" required, all others optional. We will mark required field labels bold so the visitor knows which fields he/she has to fill in.

HTML code below it into a plain text file and save it as **contact.html.** Notice how it only contains HTML code and is saved with a .html file extension.

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**title**>Contact Form</**title**> </**head**> <**body**>  <**p**>Required fields are <**b**>bold</**b**></**p**>  <**form action="contact.php" method="post"**>  <**p**><**b**>Username:</**b**> <**input type="text" name="username"** /><**br** />  <**b**>Password:</**b**> <**input type="password" name="password"** /><**br** />  <**b**>E-mail:</**b**> <**input type="text" name="email"** /><**br** />  <**b**>Your Website:</**b**> <**input type="text" name="website"** /><**br** />  </**p**>   <**p**>Do you like this website?  <**input type="radio" name="likeit" value="Yes" checked="checked"** /> Yes  <**input type="radio" name="likeit" value="No"** /> No  <**input type="radio" name="likeit" value="Not sure"** /> Not sure</**p**>   <**p**>How did you find us?  <**select name="how"**>  <**option value=""**> -- Please select -- </**option**>  <**option**>Google</**option**>  <**option**>Yahoo</**option**>  <**option**>Link from a website</**option**>  <**option**>Word of mouth</**option**>  <**option**>Other</**option**>  </**select**>   <**p**><**b**>Your comments:</**b**><**br** />  <**textarea name="comments" rows="10" cols="40"**>Text Area</**textarea**></**p**>   <**p**><**input type="submit" value="Send it!"**></**p**>  </**form**>  </**body**> </**html**> |

**Thank you page**

We could include the response in the PHP script (as shown before), but keeping it in an outside file makes the script itself less complicated and the response page easier to edit and customize.

HTML code below it into a plain text file and save it as **thanks.html**

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**title**>Thank you Page</**title**> </**head**> <**body**>  <**p**><**b**>Your message was sent</**b**></**p**>  <**p**>Your message was successfully sent!  Thank you for contacting us, we will reply  to your inquiry as soon as possible!</**p**>  </**body**> </**html**> |

# PHP form script

This script is just a summary of topics covered in this tutorial. Included are some comments to explain what is happening. PHP code below it into a plain text file and save it as **contact.php**

Change the default "you@domain.com" recipient address inside the code to your own e-mail address (the one you wish to receive form results to)!

|  |
| --- |
| **<?php**  /\* Set e-mail recipient \*/  $myemail = **"**you@domain.com**";**  /\* Check all form inputs using check\_input function \*/  $yourname = check\_input($\_POST[**'**username**'**])**;**  $email = check\_input($\_POST[**'**email**'**])**;**  $website = check\_input($\_POST[**'**website**'**])**;**  $likeit = check\_input($\_POST[**'**likeit**'**])**;**  $how\_find = check\_input($\_POST[**'**how**'**])**;**  $comments = check\_input($\_POST[**'**comments**'**])**;**  $subject = **"**Contact Info**";**  /\* If e-mail is not valid show error message \*/  if (!preg\_match(**"**/([\w\-]+\@[\w\-]+\.[\w\-]+)/**"**, $email))  {  show\_error(**"**E-mail address not valid**"**)**;**  }  /\* If URL is not valid set $website to empty \*/  if (!preg\_match(**"**/^(https?:\/\/+[\w\-]+\.[\w\-]+)/i**"**, $website))  {  $website = **'';**  }  /\* Let's prepare the message for the e-mail with multi-line string \*/  $message = **"**Hello!  Your contact form has been submitted by:  Name: **$**yourname  E-mail: **$**email  URL: **$**website  Like the website? **$**likeit  How did he/she find it? **$**how\_find  Comments:  **$**comments  End of message  **";**  /\* Send the message using mail() function \*/  mail($myemail, $subject, $message)**;**  /\* Redirect visitor to the thank you page \*/  header(**'**Location: thanks.html**'**)**;**  exit()**;**  /\* Functions we used \*/  **function** check\_input**(**$data**)**  {  $data = trim($data)**;**  $data = stripslashes($data)**;**  $data = htmlspecialchars($data)**;**  if (strlen($data) == 0)  {  show\_error(**"**Empty Field**"**)**;**  }  return $data**;**  }  **function** show\_error**(**$myError**)**  {  **?>**  <html>  <body>  <b>Please correct the following error:</b><br />  **<?php** echo $myError**;** **?>**  </body>  </html>  **<?php**  exit()**;**  }  **?>** |

# Single Page Form

For simple forms that only requires a minimal amount of processing, it can be easiest to create a single PHP file that both displays the form and process the form. This form can be organized using condition statements to determine if the form has or has not being submitted. If the form has not been submitted, display the form. If the form has been submitted process the form.

By using the $\_SERVER["PHP\_SELF"] super global variable, the file name of the currently executing script is returned. That can be as the value for the action attribute in the form tag.

<**form method="post" action="<?php echo** $\_SERVER[**"PHP\_SELF"**];**?>"**>

This form will then send the form data to itself for processing instead of being directed to secondary file. For security purposing, it is recommended to pass the $\_SERVER["PHP\_SELF"] value to htmlspecialchars() to convert any special characters to HTML entities.

The $\_SERVER["PHP\_SELF"] variable can be used by hackers!

If PHP\_SELF is used in on a form, then a user can enter a slash (/) and then some Cross Site Scripting (XSS) commands to execute. This is a security vulnerability. Cross-site scripting (XSS) is a type of computer security vulnerability typically found in Web applications. XSS enables attackers to inject client-side script into Web pages viewed by other users.

For example this string can be entered in the URL: <http://www.example.com/test_form.php/%22%3E%3Cscript%3Ealert('hacked')%3C/script%3E>

This URL will then be translated to:   
 <form method="post" action="test\_form.php/"><script>alert('hacked')</script>   
Using the <script> tag, this line of code will execute a JavaScript alert() popup .

To avoid this security vulnerability, use the htmlspecialchars() function. For example:

<**form method="post" action="<?php echo** *htmlspecialchars*($\_SERVER[**"PHP\_SELF"**]);**?>"**>

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**>HTML FORM POST</**title**>  
</**head**>  
<**body**>  
*<!-- comment: action attribute set to self -->*<**form action="<?php echo** *htmlspecialchars*($\_SERVER[**'PHP\_SELF'**]); **?>" method="post"**>  
 <**label for="name"**>Name: </**label**>  
 <**input type="text" name="name"**></**input**><**br**/>  
 <**label for="email"**>Email: </**label**>  
 <**input type="text" name="email"**></**input**><**br**/>  
 <**label for="contact"**>Contact: </**label**>  
 <**input type="text" name="contact"**></**input**><**br**/>  
 <**input type="submit" name="submit" value="Submit"**></**input**>  
</**form**>  
  
**<?php***// comment: checking if form has already been submitted with all fields completed***if** (**isset**($\_POST[**"name"**]) && **isset**($\_POST[**"email"**]) && **isset**($\_POST[**"contact"**])) {  
 **echo "Welcome: "**. $\_POST[**'name'**]. **"<br />"**;  
 **echo "Your Email is: "**. $\_POST[**"email"**]. **"<br />"**;  
 **echo "Your Mobile No. is: "**. $\_POST[**"contact"**];  
}  
**?>**</**body**>  
</**html**>

Sample Code: Single PHP script using function and condition to display and process the form:

<!DOCTYPE **html**>  
<**html**>  
<**head**>  
 <**title**>Single PHP File Form</**title**>  
</**head**>  
<**body**>  
  
**<?php**$states = **array**(**'AL'**, **'AK'**, **'AZ'**, **'AR'**, **'CA'**, **'CO'**, **'CT'**, **'DC'**, **'DE'**, **'FL'**, **'GA'**, **'HI'**, **'ID'**, **'IL'**, **'IN'**, **'IA'**,   
**'KS'**, **'KY'**, **'LA'**, **'ME'**, **'MD'**, **'MA'**, **'MI'**, **'MN'**, **'MS'**, **'MO'**, **'MT'**, **'NE'**, **'NV'**, **'NH'**, **'NJ'**, **'NM'**, **'NY'**, **'NC'**, **'ND'**,   
**'OH'**, **'OK'**, **'OR'**, **'PA'**, **'RI'**, **'SC'**, **'SD'**, **'TN'**, **'TX'**, **'UT'**, **'VT'**, **'VA'**, **'WA'**, **'WV'**, **'WI'**, **'WY'**);  
  
  
**function** validate\_form() {  
 $error = **array**();  
  
 **if** (!*in\_array*($\_POST[**"from\_state"**], $GLOBALS[**'states'**])) {  
 $error[] = **"Please select a valid state for the From address."**; *//Add to Error List* }  
 $fromzip = $\_POST[**"from\_zip"**];  
 **if** (!*preg\_match*(**"/^[0-9]{5}([- ]?[0-9]{4})?$/"**, $fromzip)) {  
 $error[] = **"Enter a valid zip code for the From address"**;  
 }  
 **if** (!*in\_array*($\_POST[**"from\_state"**], $GLOBALS[**'states'**])) {  
 $error[] = **"Please select a valid state for the To address."**; *//Add to Error List* }  
 $tozip = $\_POST[**"to\_zip"**] ;  
 **if** (!*preg\_match*(**"/^[0-9]{5}([- ]?[0-9]{4})?$/"**, $tozip)) {  
 $error[] = **"Enter a valid zip code for the To address"**;  
 }  
 **if** ($\_POST[**'weight'**] > 150) {  
 $error[] = **"Too much weight"**;  
 }  
 **if** (($\_POST[**'height'**] > 36) || ($\_POST[**'width'**] > 36) || ($\_POST[**'depth'**] > 36)) {  
 $error[] = **"Too big"**;  
 }  
 **if**(**empty**($error)) {  
 process\_form();  
 }  
 **return** $error;  
} *// end validate\_form()*

**function** process\_form() {  
 **print "Hi "** . $\_POST[**'from\_name'**] . **", your package is ready<br>"**;  
 **print "Your box's size is "** . $\_POST[**'height'**] . **"in"** . **"x"** . $\_POST[**'width'**] . **"in"** . **"x"** . $\_POST[**'depth'**] . **"in"** . **"<br>"**;  
 **print "Your box weighs "** . $\_POST[**'weight'**] . **"lbs"** . **"<br>"**;  
 **print "Would you like to ship another package?"** . **"<br>"**. **"<br>"**;  
} *// end process\_form()*

**function** print\_form($error = **array**()) {  
**print '  
<form action="'** . *htmlspecialchars*($\_SERVER[**'PHP\_SELF'**]) . **'" method="POST">  
<table>  
<tr><th>From:</th><td></td></tr>  
<tr><td>Name:</td>  
<td><input name="from\_name" type="text" /></td></tr>  
<tr><td>Address 1:</td>  
<td><input name="from\_address1" type="text" /></td></tr>  
<tr><td>Address 2:</td>  
<td><input name="from\_address2" type="text" /></td></tr>  
<tr><td>City:</td>  
<td><input name="from\_city" type="text" /></td></tr>  
<tr><td>State:</td>  
<td><input name="from\_state" type="text">  
<tr><td>ZIP:</td>  
<td><input name="from\_zip" type="text" /></td></tr>  
<tr><th>To:</th><td></td></tr>  
<tr><td>Name:</td>  
<td><input name="to\_name" type="text" /></td></tr>  
<tr><td>Address 1:</td>  
<td><input name="to\_address1" type="text" /></td></tr>  
<tr><td>Address 2:</td>  
<td><input name="to\_address2" type="text" /></td></tr>  
<tr><td>City:</td>  
<td><input name="to\_city" type="text" /></td></tr>  
<tr><td>State:</td>  
<td><input name="to\_state" type="text">  
<tr><td>ZIP:</td>  
<td><input name="to\_zip" type="text" /></td></tr>  
<tr><th>Package:</th><td></td></tr>  
<tr><td>Weight:</td>  
<td><input name="weight" type="text" /></td></tr>  
<tr><td>Height:</td>  
<td><input name="height" type="text" /></td></tr>  
<tr><td>Width:</td>  
<td><input name="width" type="text" /></td></tr>  
<tr><td>Depth:</td>  
<td><input name="depth" type="text" /></td></tr>  
<tr><td colspan="2" align="center">  
<input value="Ship!" type="submit" /></td></tr>  
</table>  
</form> '**;  
 **foreach** ($error **as** $each\_error) {  
 **print** $each\_error . **"<br>"**;  
 }  
} *// end print\_form()*

*// Has the form been submitted***if** ($\_SERVER[**'REQUEST\_METHOD'**] === **'POST'**) {  
 **if** (($errors=validate\_form())==0) {  
 validate\_forms();  
 }  
 **else** {  
 print\_form($errors);  
 }  
}  
**else** {  
 print\_form();  
}  
  
**?>**</**body**>  
</**html**>

# Sticky Forms

When submitting forms, it is commonly for users to submit the form incorrectly. The user is either entering invalid formatted data or leaving a field empty. Once the user submits the form, it is normally for an error message to appear stating why the form was submitted incorrect. But the fields they did enter correctly is gone. Making the user reenter all the data again. Sticky forms, are form specifically coded to reenter the valid text the user did enter when prompting the user to correct the incorrectly entered fields. Notice the HTML input attribute value is set to a PHP echo statement:

<**input type="text" name="fullname" value="<?php echo** $fullname; **?>"**>

By using multiple PHP escape tags, the user entered data can be entered back into the form.

**Filename: stickyform.php**

**<?php**$fullname = **""**;  
$email = **""**;  
$text = **""**;  
**if**(**isset**($\_POST[**'fullname'**]) && $\_POST[**'fullname'**] != **""**) {  
 $fullname = $\_POST[**'fullname'**];  
}  
**if**(**isset**($\_POST[**'email'**]) && $\_POST[**'email'**] != **""**) {  
 $email = $\_POST[**'email'**];  
}  
**if**(**isset**($\_POST[**'feedback'**]) && $\_POST[**'feedback'**] != **""**) {  
 $text= $\_POST[**'feedback'**];  
}  
**?>**<**html**>  
<**h1**>Customer Feedback</**h1**>  
<**p1**>Please tell us what you think</**p1**><**br**><**br**>  
  
<**form method='POST' action='<?php echo** $\_SERVER[**'PHP\_SELF'**];**?>'** >  
 <**p1**>Your name:</**p1**><**br**>  
 <**input type="text" name="fullname" value="<?php echo** $fullname; **?>"**><**br**><**br**>  
  
 <**p1**>Your email address:</**p1**><**br**>  
 <**input type="text" name="email" value="<?php echo** $email;**?>"**><**br**><**br**>  
  
 <**p1**>Your feedback:</**p1**><**br**>  
 <**textarea rows="5" cols="50" name="feedback"**>**<?php echo** $text;**?>** </**textarea**><**br**><**br**>  
  
 <**input type="submit" Value="Send Feedback"**><**br**><**br**>  
 **<?php** *error\_reporting*(***E\_ALL***);  
 **?>**</**form**>  
</**html**>