**Form Helper**

The Form Helper file contains functions that assist in working with forms.

* [Loading this Helper](https://codeigniter.com/user_guide/helpers/form_helper.html#loading-this-helper)
* [Escaping field values](https://codeigniter.com/user_guide/helpers/form_helper.html#escaping-field-values)
* [Available Functions](https://codeigniter.com/user_guide/helpers/form_helper.html#available-functions)

[**Loading this Helper**](https://codeigniter.com/user_guide/helpers/form_helper.html#id1)

This helper is loaded using the following code:

$this**->**load**->**helper('form');

[**Escaping field values**](https://codeigniter.com/user_guide/helpers/form_helper.html#id2)

You may need to use HTML and characters such as quotes within your form elements. In order to do that safely, you’ll need to use [common function](https://codeigniter.com/user_guide/general/common_functions.html) html\_escape().

Consider the following example:

$string **=** 'Here is a string containing "quoted" text.';

**<**input type**=**"text" name**=**"myfield" value**=**"<?php echo $string; ?>" **/>**

Since the above string contains a set of quotes, it will cause the form to break. The [html\_escape()](https://codeigniter.com/user_guide/general/common_functions.html#html_escape) function converts HTML special characters so that it can be used safely:

**<**input type**=**"text" name**=**"myfield" value**=**"<?php echo html\_escape($string); ?>" **/>**

**Note**

If you use any of the form helper functions listed on this page, the form values will be automatically escaped, so there is no need to call this function. Use it only if you are creating your own form elements.

[**Available Functions**](https://codeigniter.com/user_guide/helpers/form_helper.html#id3)

The following functions are available:

**form\_open([*$action = ''*[, *$attributes = ''*[, *$hidden = array()*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$action** (*string*) – Form action/target URI string * **$attributes** (*array*) – HTML attributes * **$hidden** (*array*) – An array of hidden fields’ definitions |
| **Returns:** | An HTML form opening tag |
| **Return type:** | string |

Creates an opening form tag with a base URL **built from your config preferences**. It will optionally let you add form attributes and hidden input fields, and will always add the *accept-charset* attribute based on the charset value in your config file.

The main benefit of using this tag rather than hard coding your own HTML is that it permits your site to be more portable in the event your URLs ever change.

Here’s a simple example:

**echo** form\_open('email/send');

The above example would create a form that points to your base URL plus the “email/send” URI segments, like this:

**<**form method**=**"post" accept**-**charset**=**"utf-8" action**=**"http://example.com/index.php/email/send"**>**

**Adding Attributes**

Attributes can be added by passing an associative array to the second parameter, like this:

$attributes **=** **array**('class' **=>** 'email', 'id' **=>** 'myform');

**echo** form\_open('email/send', $attributes);

Alternatively, you can specify the second parameter as a string:

**echo** form\_open('email/send', 'class="email" id="myform"');

The above examples would create a form similar to this:

**<**form method**=**"post" accept**-**charset**=**"utf-8" action**=**"http://example.com/index.php/email/send" class**=**"email" id**=**"myform"**>**

**Adding Hidden Input Fields**

Hidden fields can be added by passing an associative array to the third parameter, like this:

$hidden **=** **array**('username' **=>** 'Joe', 'member\_id' **=>** '234');

**echo** form\_open('email/send', '', $hidden);

You can skip the second parameter by passing any falsy value to it.

The above example would create a form similar to this:

**<**form method**=**"post" accept**-**charset**=**"utf-8" action**=**"http://example.com/index.php/email/send"**>**

**<**input type**=**"hidden" name**=**"username" value**=**"Joe" **/>**

**<**input type**=**"hidden" name**=**"member\_id" value**=**"234" **/>**

**form\_open\_multipart([*$action = ''*[, *$attributes = array()*[, *$hidden = array()*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$action** (*string*) – Form action/target URI string * **$attributes** (*array*) – HTML attributes * **$hidden** (*array*) – An array of hidden fields’ definitions |
| **Returns:** | An HTML multipart form opening tag |
| **Return type:** | string |

This function is absolutely identical to [form\_open()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_open) above, except that it adds a *multipart* attribute, which is necessary if you would like to use the form to upload files with.

**form\_hidden(*$name*[, *$value = ''*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$name** (*string*) – Field name * **$value** (*string*) – Field value |
| **Returns:** | An HTML hidden input field tag |
| **Return type:** | string |

Lets you generate hidden input fields. You can either submit a name/value string to create one field:

form\_hidden('username', 'johndoe');

*// Would produce: <input type="hidden" name="username" value="johndoe" />*

… or you can submit an associative array to create multiple fields:

$data **=** **array**(

'name' **=>** 'John Doe',

'email' **=>** 'john@example.com',

'url' **=>** 'http://example.com'

);

**echo** form\_hidden($data);

*/\**

*Would produce:*

*<input type="hidden" name="name" value="John Doe" />*

*<input type="hidden" name="email" value="john@example.com" />*

*<input type="hidden" name="url" value="http://example.com" />*

*\*/*

You can also pass an associative array to the value field:

$data **=** **array**(

'name' **=>** 'John Doe',

'email' **=>** 'john@example.com',

'url' **=>** 'http://example.com'

);

**echo** form\_hidden('my\_array', $data);

*/\**

*Would produce:*

*<input type="hidden" name="my\_array[name]" value="John Doe" />*

*<input type="hidden" name="my\_array[email]" value="john@example.com" />*

*<input type="hidden" name="my\_array[url]" value="http://example.com" />*

*\*/*

If you want to create hidden input fields with extra attributes:

$data **=** **array**(

'type' **=>** 'hidden',

'name' **=>** 'email',

'id' **=>** 'hiddenemail',

'value' **=>** 'john@example.com',

'class' **=>** 'hiddenemail'

);

**echo** form\_input($data);

*/\**

*Would produce:*

*<input type="hidden" name="email" value="john@example.com" id="hiddenemail" class="hiddenemail" />*

*\*/*

**form\_input([*$data = ''*[, *$value = ''*[, *$extra = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*array*) – Field attributes data * **$value** (*string*) – Field value * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML text input field tag |
| **Return type:** | string |

Lets you generate a standard text input field. You can minimally pass the field name and value in the first and second parameter:

**echo** form\_input('username', 'johndoe');

Or you can pass an associative array containing any data you wish your form to contain:

$data **=** **array**(

'name' **=>** 'username',

'id' **=>** 'username',

'value' **=>** 'johndoe',

'maxlength' **=>** '100',

'size' **=>** '50',

'style' **=>** 'width:50%'

);

**echo** form\_input($data);

*/\**

*Would produce:*

*<input type="text" name="username" value="johndoe" id="username" maxlength="100" size="50" style="width:50%" />*

*\*/*

If you would like your form to contain some additional data, like JavaScript, you can pass it as a string in the third parameter:

$js **=** 'onClick="some\_function()"';

**echo** form\_input('username', 'johndoe', $js);

Or you can pass it as an array:

$js **=** **array**('onClick' **=>** 'some\_function();');

**echo** form\_input('username', 'johndoe', $js);

**form\_password([*$data = ''*[, *$value = ''*[, *$extra = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*array*) – Field attributes data * **$value** (*string*) – Field value * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML password input field tag |
| **Return type:** | string |

This function is identical in all respects to the [form\_input()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_input) function above except that it uses the “password” input type.

**form\_upload([*$data = ''*[, *$value = ''*[, *$extra = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*array*) – Field attributes data * **$value** (*string*) – Field value * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML file upload input field tag |
| **Return type:** | string |

This function is identical in all respects to the [form\_input()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_input) function above except that it uses the “file” input type, allowing it to be used to upload files.

**form\_textarea([*$data = ''*[, *$value = ''*[, *$extra = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*array*) – Field attributes data * **$value** (*string*) – Field value * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML textarea tag |
| **Return type:** | string |

This function is identical in all respects to the [form\_input()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_input) function above except that it generates a “textarea” type.

**Note**

Instead of the *maxlength* and *size* attributes in the above example, you will instead specify *rows* and *cols*.

**form\_dropdown([*$name = ''*[, *$options = array()*[, *$selected = array()*[, *$extra = ''*]]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$name** (*string*) – Field name * **$options** (*array*) – An associative array of options to be listed * **$selected** (*array*) – List of fields to mark with the *selected* attribute * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML dropdown select field tag |
| **Return type:** | string |

Lets you create a standard drop-down field. The first parameter will contain the name of the field, the second parameter will contain an associative array of options, and the third parameter will contain the value you wish to be selected. You can also pass an array of multiple items through the third parameter, and CodeIgniter will create a multiple select for you.

Example:

$options **=** **array**(

'small' **=>** 'Small Shirt',

'med' **=>** 'Medium Shirt',

'large' **=>** 'Large Shirt',

'xlarge' **=>** 'Extra Large Shirt',

);

$shirts\_on\_sale **=** **array**('small', 'large');

**echo** form\_dropdown('shirts', $options, 'large');

*/\**

*Would produce:*

*<select name="shirts">*

*<option value="small">Small Shirt</option>*

*<option value="med">Medium Shirt</option>*

*<option value="large" selected="selected">Large Shirt</option>*

*<option value="xlarge">Extra Large Shirt</option>*

*</select>*

*\*/*

**echo** form\_dropdown('shirts', $options, $shirts\_on\_sale);

*/\**

*Would produce:*

*<select name="shirts" multiple="multiple">*

*<option value="small" selected="selected">Small Shirt</option>*

*<option value="med">Medium Shirt</option>*

*<option value="large" selected="selected">Large Shirt</option>*

*<option value="xlarge">Extra Large Shirt</option>*

*</select>*

*\*/*

If you would like the opening <select> to contain additional data, like an id attribute or JavaScript, you can pass it as a string in the fourth parameter:

$js **=** 'id="shirts" onChange="some\_function();"';

**echo** form\_dropdown('shirts', $options, 'large', $js);

Or you can pass it as an array:

$js **=** **array**(

'id' **=>** 'shirts',

'onChange' **=>** 'some\_function();'

);

**echo** form\_dropdown('shirts', $options, 'large', $js);

If the array passed as $options is a multidimensional array, then form\_dropdown() will produce an <optgroup> with the array key as the label.

**form\_multiselect([*$name = ''*[, *$options = array()*[, *$selected = array()*[, *$extra = ''*]]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$name** (*string*) – Field name * **$options** (*array*) – An associative array of options to be listed * **$selected** (*array*) – List of fields to mark with the *selected* attribute * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML dropdown multiselect field tag |
| **Return type:** | string |

Lets you create a standard multiselect field. The first parameter will contain the name of the field, the second parameter will contain an associative array of options, and the third parameter will contain the value or values you wish to be selected.

The parameter usage is identical to using [form\_dropdown()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_dropdown) above, except of course that the name of the field will need to use POST array syntax, e.g. foo[].

**form\_fieldset([*$legend\_text = ''*[, *$attributes = array()*]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$legend\_text** (*string*) – Text to put in the <legend> tag * **$attributes** (*array*) – Attributes to be set on the <fieldset> tag |
| **Returns:** | An HTML fieldset opening tag |
| **Return type:** | string |

Lets you generate fieldset/legend fields.

Example:

**echo** form\_fieldset('Address Information');

**echo** "<p>fieldset content here</p>\n";

**echo** form\_fieldset\_close();

*/\**

*Produces:*

*<fieldset>*

*<legend>Address Information</legend>*

*<p>fieldset content here</p>*

*</fieldset>*

*\*/*

Similar to other functions, you can submit an associative array in the second parameter if you prefer to set additional attributes:

$attributes **=** **array**(

'id' **=>** 'address\_info',

'class' **=>** 'address\_info'

);

**echo** form\_fieldset('Address Information', $attributes);

**echo** "<p>fieldset content here</p>\n";

**echo** form\_fieldset\_close();

*/\**

*Produces:*

*<fieldset id="address\_info" class="address\_info">*

*<legend>Address Information</legend>*

*<p>fieldset content here</p>*

*</fieldset>*

*\*/*

**form\_fieldset\_close([*$extra = ''*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$extra** (*string*) – Anything to append after the closing tag, *as is* |
| **Returns:** | An HTML fieldset closing tag |
| **Return type:** | string |

Produces a closing </fieldset> tag. The only advantage to using this function is it permits you to pass data to it which will be added below the tag. For example

$string **=** '</div></div>';

**echo** form\_fieldset\_close($string);

*// Would produce: </fieldset></div></div>*

**form\_checkbox([*$data = ''*[, *$value = ''*[, *$checked = FALSE*[, *$extra = ''*]]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*array*) – Field attributes data * **$value** (*string*) – Field value * **$checked** (*bool*) – Whether to mark the checkbox as being *checked* * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML checkbox input tag |
| **Return type:** | string |

Lets you generate a checkbox field. Simple example:

**echo** form\_checkbox('newsletter', 'accept', **TRUE**);

*// Would produce: <input type="checkbox" name="newsletter" value="accept" checked="checked" />*

The third parameter contains a boolean TRUE/FALSE to determine whether the box should be checked or not.

Similar to the other form functions in this helper, you can also pass an array of attributes to the function:

$data **=** **array**(

'name' **=>** 'newsletter',

'id' **=>** 'newsletter',

'value' **=>** 'accept',

'checked' **=>** **TRUE**,

'style' **=>** 'margin:10px'

);

**echo** form\_checkbox($data);

*// Would produce: <input type="checkbox" name="newsletter" id="newsletter" value="accept" checked="checked" style="margin:10px" />*

Also as with other functions, if you would like the tag to contain additional data like JavaScript, you can pass it as a string in the fourth parameter:

$js **=** 'onClick="some\_function()"';

**echo** form\_checkbox('newsletter', 'accept', **TRUE**, $js);

Or you can pass it as an array:

$js **=** **array**('onClick' **=>** 'some\_function();');

**echo** form\_checkbox('newsletter', 'accept', **TRUE**, $js);

**form\_radio([*$data = ''*[, *$value = ''*[, *$checked = FALSE*[, *$extra = ''*]]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*array*) – Field attributes data * **$value** (*string*) – Field value * **$checked** (*bool*) – Whether to mark the radio button as being *checked* * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML radio input tag |
| **Return type:** | string |

This function is identical in all respects to the [form\_checkbox()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_checkbox) function above except that it uses the “radio” input type.

**form\_label([*$label\_text = ''*[, *$id = ''*[, *$attributes = array()*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$label\_text** (*string*) – Text to put in the <label> tag * **$id** (*string*) – ID of the form element that we’re making a label for * **$attributes** (*mixed*) – HTML attributes |
| **Returns:** | An HTML field label tag |
| **Return type:** | string |

Lets you generate a <label>. Simple example:

**echo** form\_label('What is your Name', 'username');

*// Would produce: <label for="username">What is your Name</label>*

Similar to other functions, you can submit an associative array in the third parameter if you prefer to set additional attributes.

Example:

$attributes **=** **array**(

'class' **=>** 'mycustomclass',

'style' **=>** 'color: #000;'

);

**echo** form\_label('What is your Name', 'username', $attributes);

*// Would produce: <label for="username" class="mycustomclass" style="color: #000;">What is your Name</label>*

**form\_submit([*$data = ''*[, *$value = ''*[, *$extra = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*string*) – Button name * **$value** (*string*) – Button value * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML input submit tag |
| **Return type:** | string |

Lets you generate a standard submit button. Simple example:

**echo** form\_submit('mysubmit', 'Submit Post!');

*// Would produce: <input type="submit" name="mysubmit" value="Submit Post!" />*

Similar to other functions, you can submit an associative array in the first parameter if you prefer to set your own attributes. The third parameter lets you add extra data to your form, like JavaScript.

**form\_reset([*$data = ''*[, *$value = ''*[, *$extra = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*string*) – Button name * **$value** (*string*) – Button value * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML input reset button tag |
| **Return type:** | string |

Lets you generate a standard reset button. Use is identical to form\_submit().

**form\_button([*$data = ''*[, *$content = ''*[, *$extra = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*string*) – Button name * **$content** (*string*) – Button label * **$extra** (*mixed*) – Extra attributes to be added to the tag either as an array or a literal string |
| **Returns:** | An HTML button tag |
| **Return type:** | string |

Lets you generate a standard button element. You can minimally pass the button name and content in the first and second parameter:

**echo** form\_button('name','content');

*// Would produce: <button name="name" type="button">Content</button>*

Or you can pass an associative array containing any data you wish your form to contain:

$data **=** **array**(

'name' **=>** 'button',

'id' **=>** 'button',

'value' **=>** 'true',

'type' **=>** 'reset',

'content' **=>** 'Reset'

);

**echo** form\_button($data);

*// Would produce: <button name="button" id="button" value="true" type="reset">Reset</button>*

If you would like your form to contain some additional data, like JavaScript, you can pass it as a string in the third parameter:

$js **=** 'onClick="some\_function()"';

**echo** form\_button('mybutton', 'Click Me', $js);

**form\_close([*$extra = ''*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$extra** (*string*) – Anything to append after the closing tag, *as is* |
| **Returns:** | An HTML form closing tag |
| **Return type:** | string |

Produces a closing </form> tag. The only advantage to using this function is it permits you to pass data to it which will be added below the tag. For example:

$string **=** '</div></div>';

**echo** form\_close($string);

*// Would produce: </form> </div></div>*

**set\_value(*$field*[, *$default = ''*[, *$html\_escape = TRUE*]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$field** (*string*) – Field name * **$default** (*string*) – Default value * **$html\_escape** (*bool*) – Whether to turn off HTML escaping of the value |
| **Returns:** | Field value |
| **Return type:** | string |

Permits you to set the value of an input form or textarea. You must supply the field name via the first parameter of the function. The second (optional) parameter allows you to set a default value for the form. The third (optional) parameter allows you to turn off HTML escaping of the value, in case you need to use this function in combination with i.e. [form\_input()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_input) and avoid double-escaping.

Example:

**<**input type**=**"text" name**=**"quantity" value**=**"<?php echo set\_value('quantity', '0'); ?>" size**=**"50" **/>**

The above form will show “0” when loaded for the first time.

**Note**

If you’ve loaded the [Form Validation Library](https://codeigniter.com/user_guide/libraries/form_validation.html) and have set a validation rule for the field name in use with this helper, then it will forward the call to the [Form Validation Library](https://codeigniter.com/user_guide/libraries/form_validation.html)’s own set\_value() method. Otherwise, this function looks in $\_POST for the field value.

**set\_select(*$field*[, *$value = ''*[, *$default = FALSE*]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$field** (*string*) – Field name * **$value** (*string*) – Value to check for * **$default** (*string*) – Whether the value is also a default one |
| **Returns:** | ‘selected’ attribute or an empty string |
| **Return type:** | string |

If you use a <select> menu, this function permits you to display the menu item that was selected.

The first parameter must contain the name of the select menu, the second parameter must contain the value of each item, and the third (optional) parameter lets you set an item as the default (use boolean TRUE/FALSE).

Example:

**<**select name**=**"myselect"**>**

**<**option value**=**"one" **<?**php **echo** set\_select('myselect', 'one', **TRUE**); **?>** >One</option>

<option value**=**"two" **<?php** **echo** set\_select('myselect', 'two'); **?>** >Two</option>

<option value**=**"three" **<?php** **echo** set\_select('myselect', 'three'); **?>** >Three</option>

</select>

**set\_checkbox(*$field*[, *$value = ''*[, *$default = FALSE*]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$field** (*string*) – Field name * **$value** (*string*) – Value to check for * **$default** (*string*) – Whether the value is also a default one |
| **Returns:** | ‘checked’ attribute or an empty string |
| **Return type:** | string |

Permits you to display a checkbox in the state it was submitted.

The first parameter must contain the name of the checkbox, the second parameter must contain its value, and the third (optional) parameter lets you set an item as the default (use boolean TRUE/FALSE).

Example:

**<**input type**=**"checkbox" name**=**"mycheck" value**=**"1" **<?**php **echo** set\_checkbox('mycheck', '1'); **?>** />

<input type**=**"checkbox" name**=**"mycheck" value**=**"2" **<?php** **echo** set\_checkbox('mycheck', '2'); **?>** />

**set\_radio(*$field*[, *$value = ''*[, *$default = FALSE*]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$field** (*string*) – Field name * **$value** (*string*) – Value to check for * **$default** (*string*) – Whether the value is also a default one |
| **Returns:** | ‘checked’ attribute or an empty string |
| **Return type:** | string |

Permits you to display radio buttons in the state they were submitted. This function is identical to the [set\_checkbox()](https://codeigniter.com/user_guide/helpers/form_helper.html#set_checkbox) function above.

Example:

**<**input type**=**"radio" name**=**"myradio" value**=**"1" **<?**php **echo** set\_radio('myradio', '1', **TRUE**); **?>** />

<input type**=**"radio" name**=**"myradio" value**=**"2" **<?php** **echo** set\_radio('myradio', '2'); **?>** />

**Note**

If you are using the Form Validation class, you must always specify a rule for your field, even if empty, in order for the set\_\*() functions to work. This is because if a Form Validation object is defined, the control for set\_\*() is handed over to a method of the class instead of the generic helper function.

**form\_error([*$field = ''*[, *$prefix = ''*[, *$suffix = ''*]]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$field** (*string*) – Field name * **$prefix** (*string*) – Error opening tag * **$suffix** (*string*) – Error closing tag |
| **Returns:** | HTML-formatted form validation error message(s) |
| **Return type:** | string |

Returns a validation error message from the [Form Validation Library](https://codeigniter.com/user_guide/libraries/form_validation.html), associated with the specified field name. You can optionally specify opening and closing tag(s) to put around the error message.

Example:

*// Assuming that the 'username' field value was incorrect:*

**echo** form\_error('myfield', '<div class="error">', '</div>');

*// Would produce: <div class="error">Error message associated with the "username" field.</div>*

**validation\_errors([*$prefix = ''*[, *$suffix = ''*]])**

|  |  |
| --- | --- |
| **Parameters:** | * **$prefix** (*string*) – Error opening tag * **$suffix** (*string*) – Error closing tag |
| **Returns:** | HTML-formatted form validation error message(s) |
| **Return type:** | string |

Similarly to the [form\_error()](https://codeigniter.com/user_guide/helpers/form_helper.html#form_error) function, returns all validation error messages produced by the [Form Validation Library](https://codeigniter.com/user_guide/libraries/form_validation.html), with optional opening and closing tags around each of the messages.

Example:

**echo** validation\_errors('<span class="error">', '</span>');

*/\**

*Would produce, e.g.:*

*<span class="error">The "email" field doesn't contain a valid e-mail address!</span>*

*<span class="error">The "password" field doesn't match the "repeat\_password" field!</span>*

*\*/*

**form\_prep(*$str*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$str** (*string*) – Value to escape |
| **Returns:** | Escaped value |
| **Return type:** | string |

Allows you to safely use HTML and characters such as quotes within form elements without breaking out of the form.

**Note**

If you use any of the form helper functions listed in this page the form values will be prepped automatically, so there is no need to call this function. Use it only if you are creating your own form elements.

**Note**

This function is DEPRECATED and is just an alias for [common function](https://codeigniter.com/user_guide/general/common_functions.html) html\_escape() - please use that instead.