Web Forms

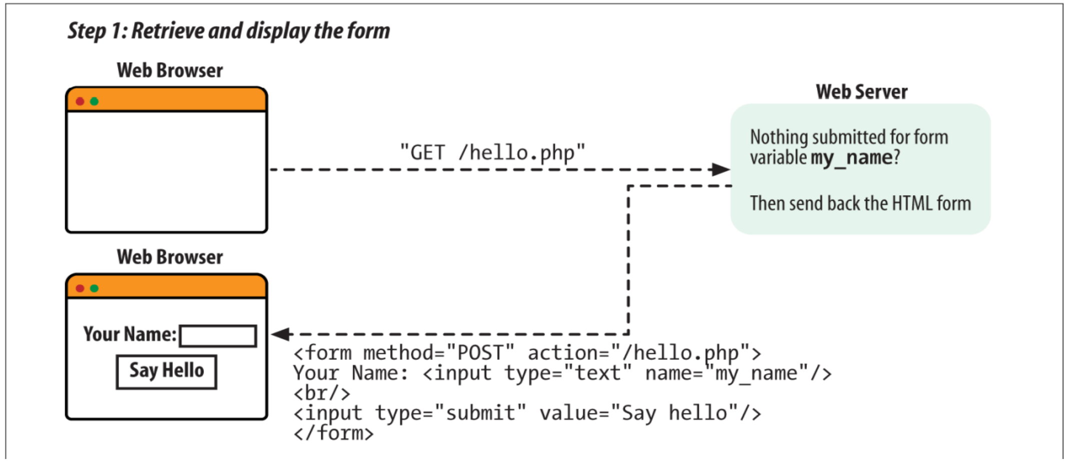
Chapter 7

Web Forms

- Form processing is an essential component of almost any web application.
- Users communicate with the server via web forms, Examples:
  - signing up for a new account
  - searching in a website
- Web Forms are Implemented in 2 steps:

1. Using HTML, we construct a form with appropriate elements in it, such as text boxes, checkboxes, and buttons.
2. Writing Server-side scripts (e.g. PHP) to process the user submitted information.

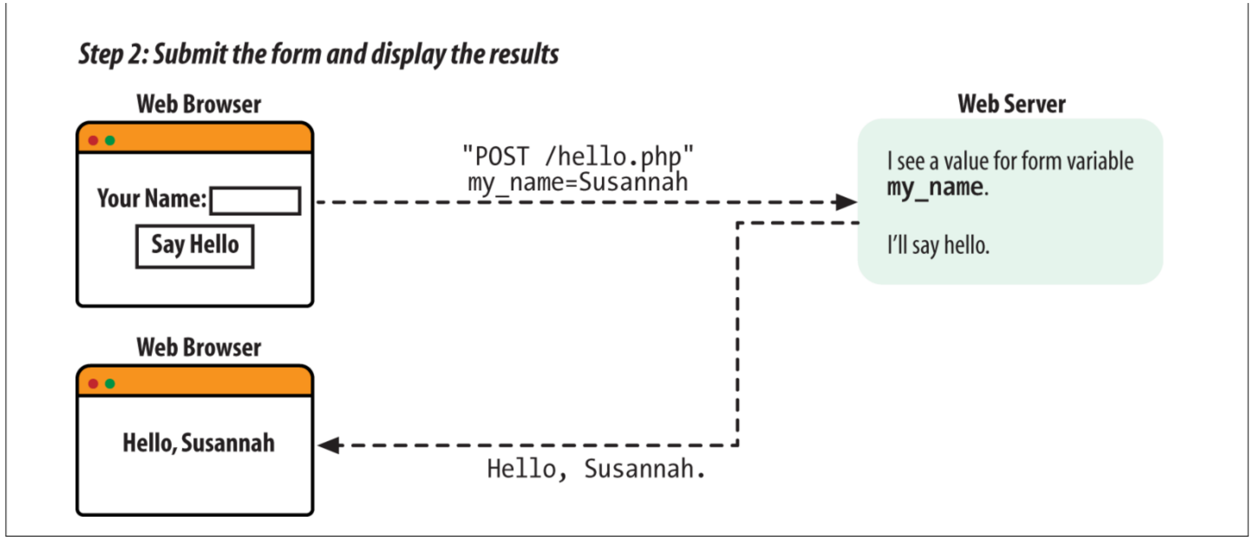
Review: Client-Server Communication



Step 1: Retrieve and display the form

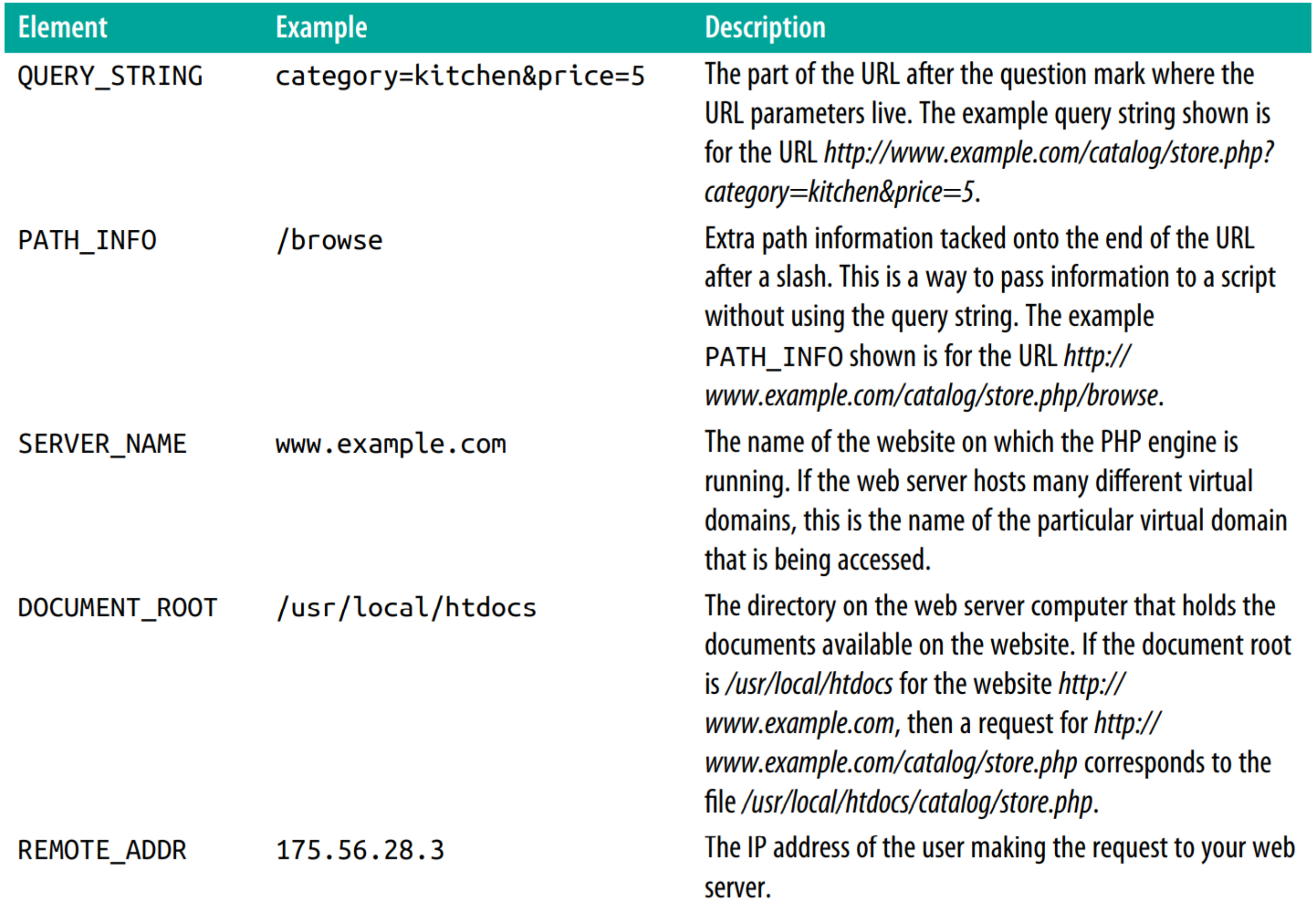
Web Browser
"GET /hello.php"
Web Server
Nothing submitted for form variable **my_name**?
Then send back the HTML form
Web Browser
Your Name:
Say Hello

```
<form method="POST" action="/hello.php">
Your Name: <input type="text" name="my_name"/>
<br/>
<input type="submit" value="Say hello"/>
</form>
```



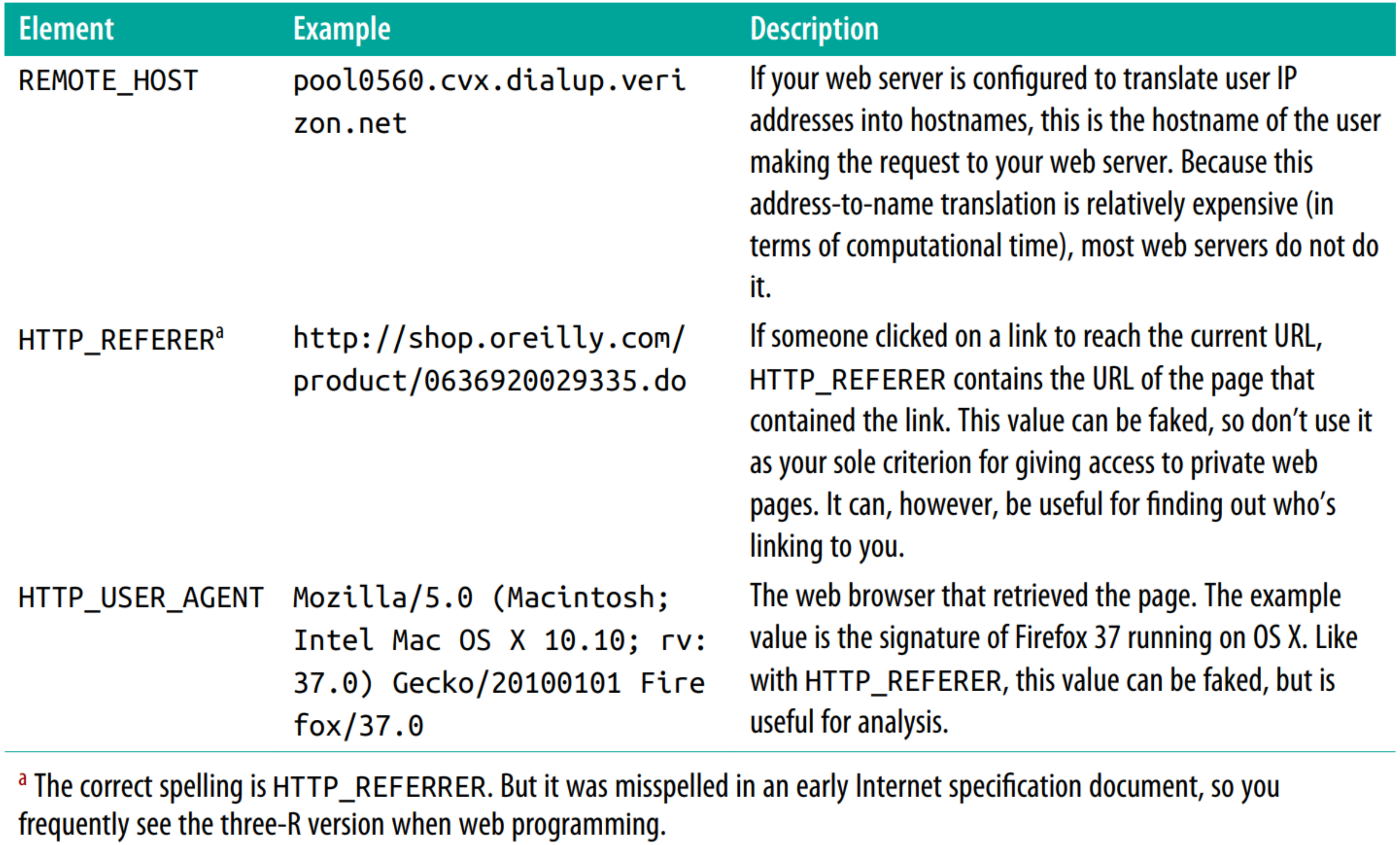
Step 2: Submit the form and display the results

Useful Server Variables



Element	Example	Description
QUERY_STRING	category=kitchen&price=5	The part of the URL after the question mark where the URL parameters live. The example query string shown is for the URL *http://www.example.com/catalog/store.php?category=kitchen&price=5*.
PATH_INFO	/browse	Extra path information tacked onto the end of the URL after a slash. This is a way to pass information to a script without using the query string. The example PATH_INFO shown is for the URL *http://www.example.com/catalog/store.php/browse*.
SERVER_NAME	www.example.com	The name of the website on which the PHP engine is running. If the web server hosts many different virtual domains, this is the name of the particular virtual domain that is being accessed.
DOCUMENT_ROOT	/usr/local/htdocs	The directory on the web server computer that holds the documents available on the website. If the document root is */usr/local/htdocs* for the website *http://www.example.com*, then a request for *http://www.example.com/catalog/store.php* corresponds to the file */usr/local/htdocs/catalog/store.php*.
REMOTE_ADDR	175.56.28.3	The IP address of the user making the request to your web server.

Useful Server Variables

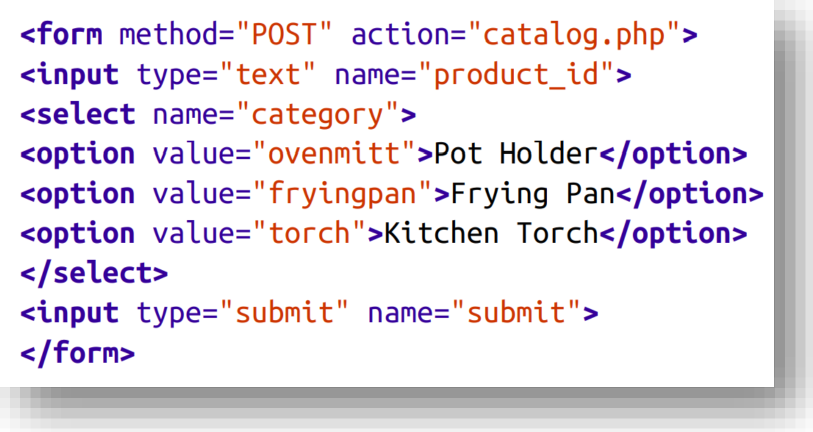


Element	Example	Description
REMOTE_HOST	pool0560.cvx.dialup.verizon.net	If your web server is configured to translate user IP addresses into hostnames, this is the hostname of the user making the request to your web server. Because this address-to-name translation is relatively expensive (in terms of computational time), most web servers do not do it.
HTTP_REFERER[a]	http://shop.oreilly.com/product/0636920029335.do	If someone clicked on a link to reach the current URL, HTTP_REFERER contains the URL of the page that contained the link. This value can be faked, so don't use it as your sole criterion for giving access to private web pages. It can, however, be useful for finding out who's linking to you.
HTTP_USER_AGENT	Mozilla/5.0 (Macintosh; Intel Mac OS X 10.10; rv:37.0) Gecko/20100101 Firefox/37.0	The web browser that retrieved the page. The example value is the signature of Firefox 37 running on OS X. Like with HTTP_REFERER, this value can be faked, but is useful for analysis.

[a] The correct spelling is HTTP_REFERRER. But it was misspelled in an early Internet specification document, so you frequently see the three-R version when web programming.

Accessing Form Parameters

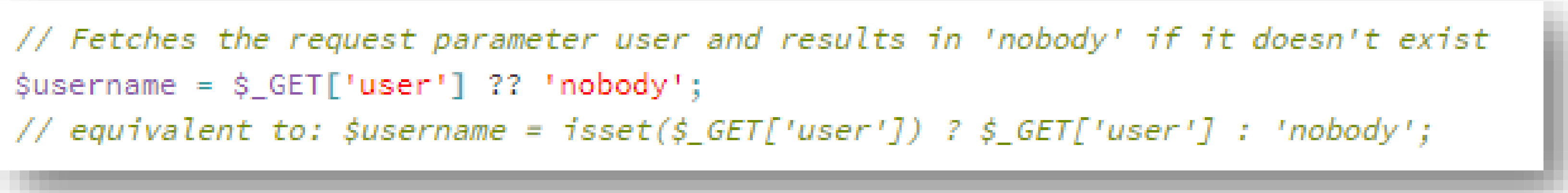
- At every request, the PHP engine sets up some auto-global arrays that contain the values of parameters submitted in a form or passed in the URL.
- URL and form parameters from GET method forms are put into $_GET.
- Form parameters from POST method forms are put into $_POST.
- For Example:
  - The URL http://www.example.com/catalog.php?product_id=21&category=fryingpan puts two values into $_GET:
    - *$_GET['product_id'] is set to 21*
    - *$_GET['category'] is set to fryingpan*
  - the same values could be put into $_POST if the form method was post



```
<form method="POST" action="catalog.php">
<input type="text" name="product_id">
<select name="category">
<option value="ovenmitt">Pot Holder</option>
<option value="fryingpan">Frying Pan</option>
<option value="torch">Kitchen Torch</option>
</select>
<input type="submit" name="submit">
</form>
```

null coalesce operator (??)

- The coalesce, or ?? operator returns the result of its first operand if it exists and is not NULL, or else its second operand

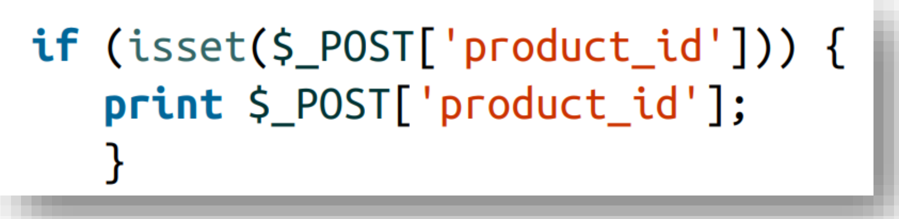


```php
// Fetches the request parameter user and results in 'nobody' if it doesn't exist
$username = $_GET['user'] ?? 'nobody';
// equivalent to: $username = isset($_GET['user']) ? $_GET['user'] : 'nobody';
```

- Null coalesce introduced in PHP 7.0, for older versions use ternary operator (? :) and isset()

  - It is equivalent to: `$username = isset($_GET['user']) ? $_GET['user'] : 'nobody’;`

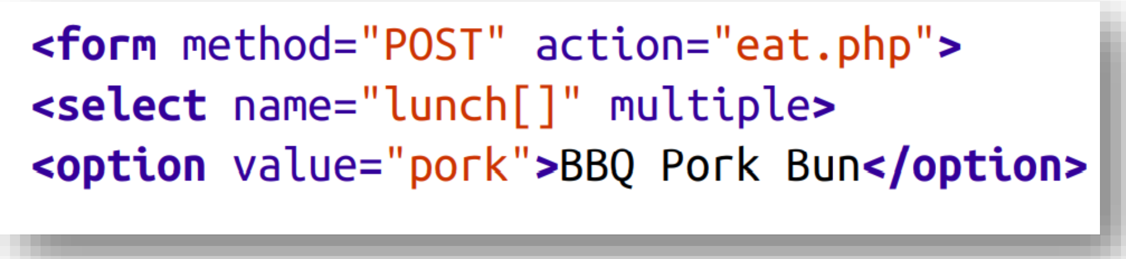
- OR



```php
if (isset($_POST['product_id'])) {
   print $_POST['product_id'];
   }
```

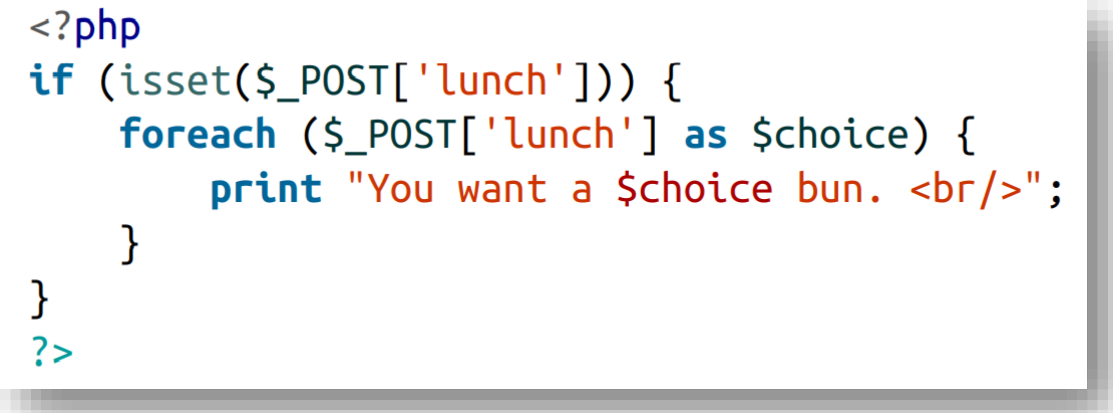
Element with multiple values

- A form element with multiple values its name must end with []



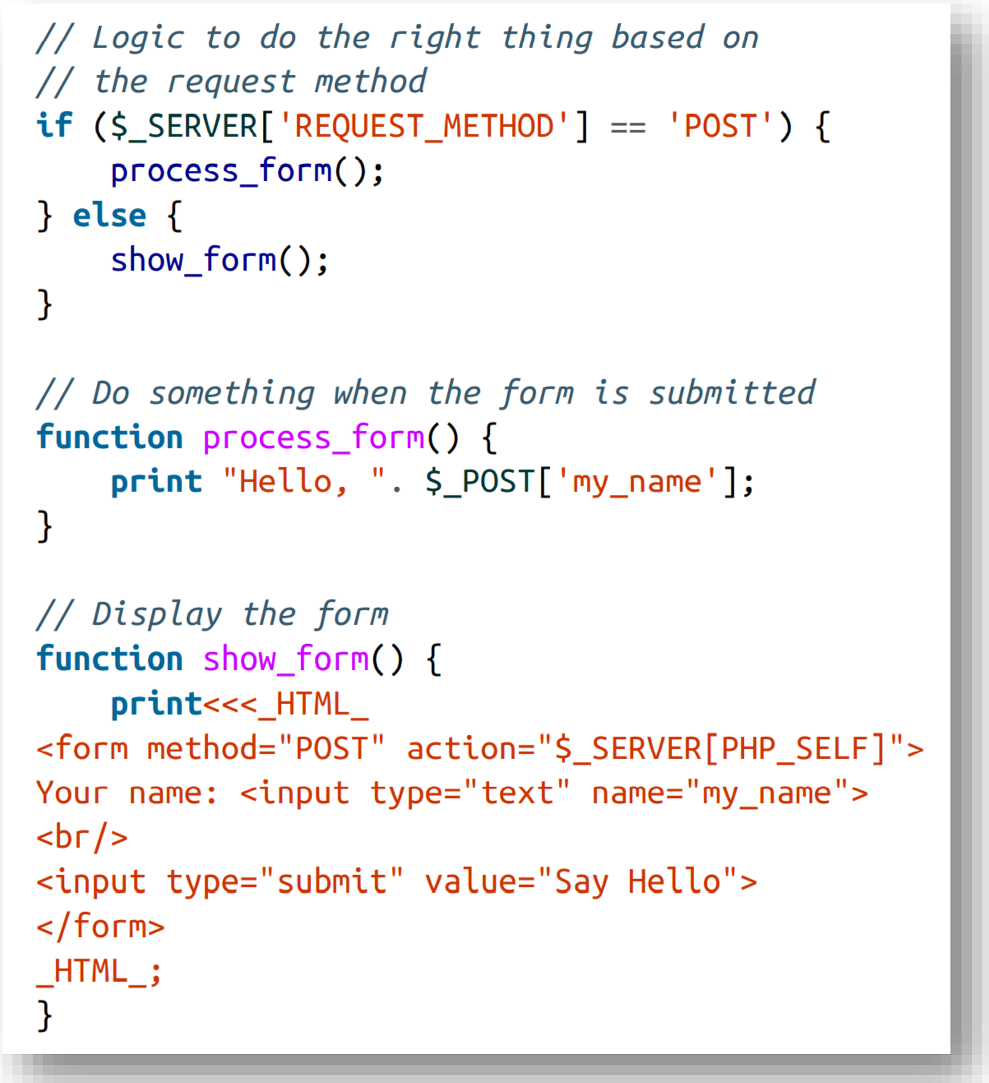
```html
<form method="POST" action="eat.php">
<select name="lunch[]" multiple>
<option value="pork">BBQ Pork Bun</option>
```

- If the form above is submitted with 2 values selected, then $_POST['lunch'] becomes a two-element array



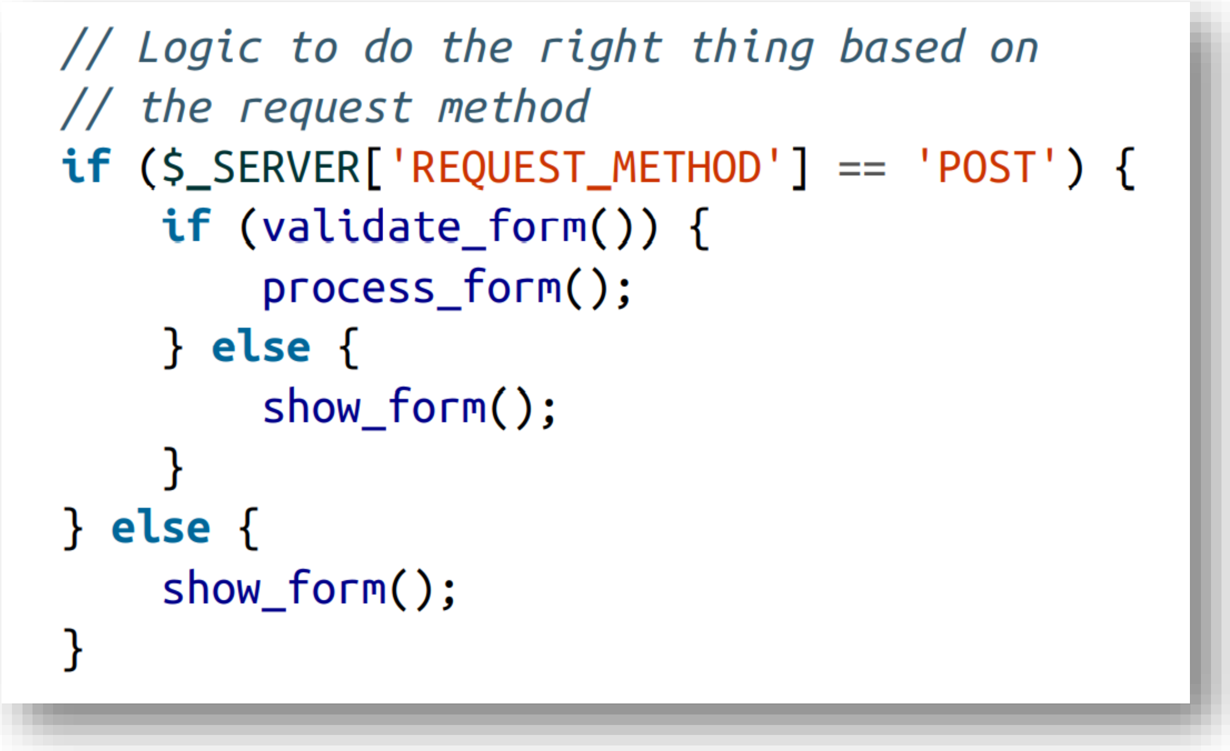
```php
<?php
if (isset($_POST['lunch'])) {
    foreach ($_POST['lunch'] as $choice) {
        print "You want a $choice bun. <br/>";
    }
}
?>
```

Form Processing with Functions

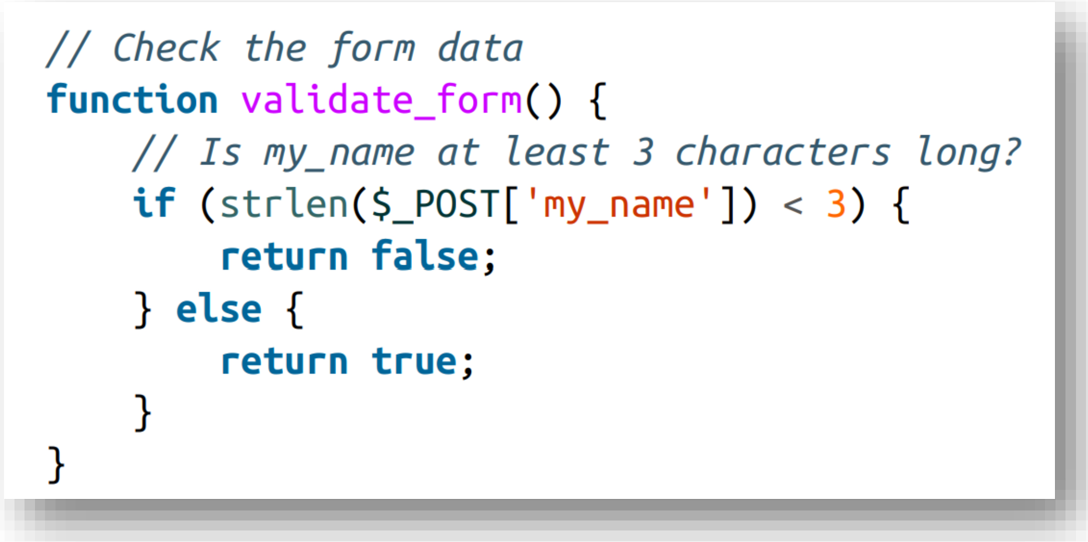


```
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    process_form();
} else {
    show_form();
}

// Do something when the form is submitted
function process_form() {
    print "Hello, ". $_POST['my_name'];
}

// Display the form
function show_form() {
    print<<<_HTML_
<form method="POST" action="$_SERVER[PHP_SELF]">
Your name: <input type="text" name="my_name">
<br/>
<input type="submit" value="Say Hello">
</form>
_HTML_;
}
```

Validating Data



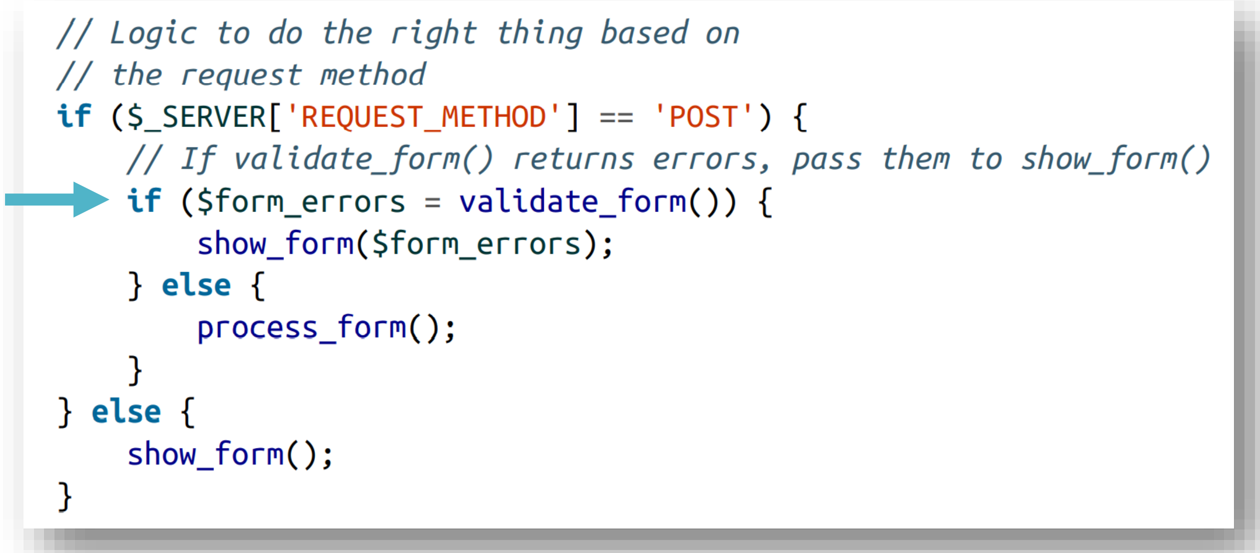
```
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    if (validate_form()) {
        process_form();
    } else {
        show_form();
    }
} else {
    show_form();
}
```



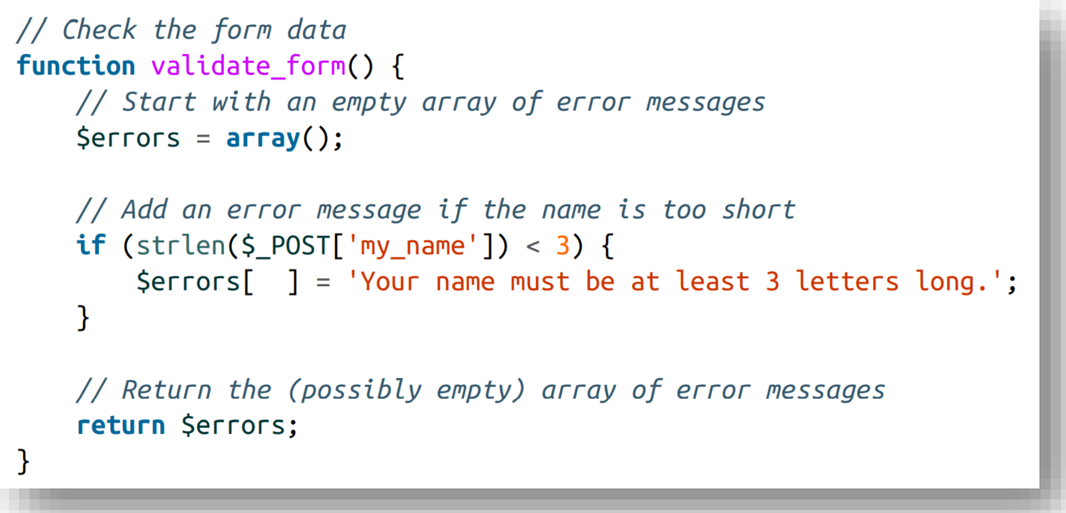
```
// Check the form data
function validate_form() {
    // Is my_name at least 3 characters long?
    if (strlen($_POST['my_name']) < 3) {
        return false;
    } else {
        return true;
    }
}
```

Validating Data

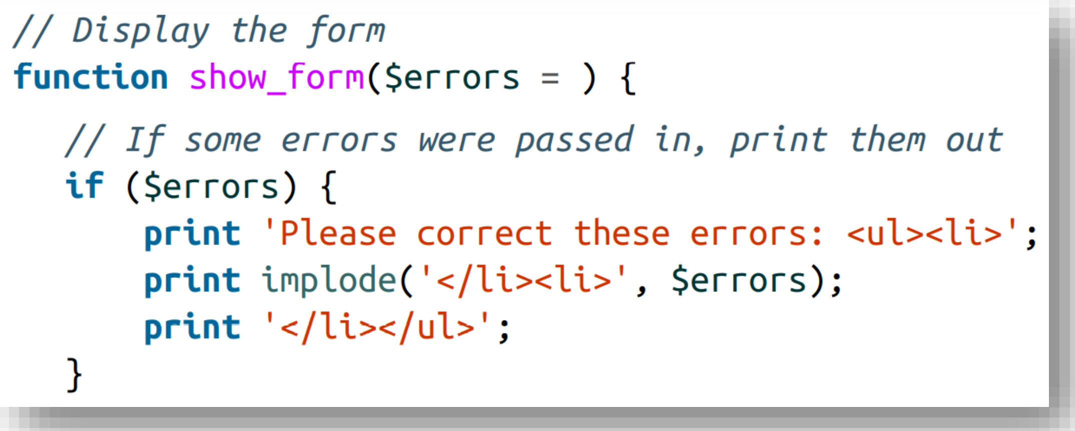
- Ideally, when validation fails, you should explain the error to the user, if appropriate, redisplay the errornous value entered in the form element.



```
// Logic to do the right thing based on
// the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // If validate_form() returns errors, pass them to show_form()
    if ($form_errors = validate_form()) {
        show_form($form_errors);
    } else {
        process_form();
    }
} else {
    show_form();
}
```



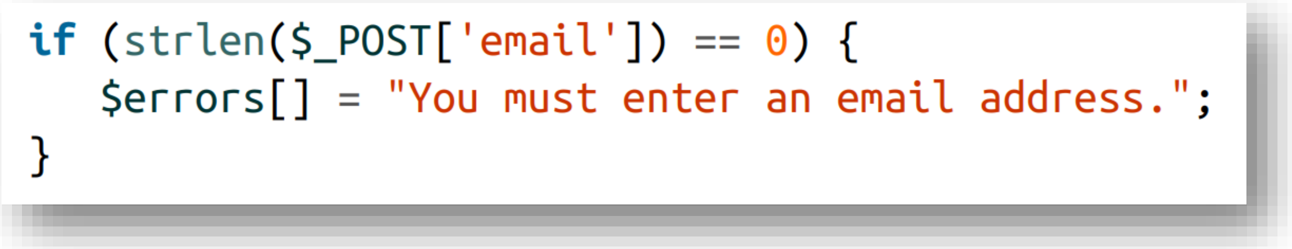
```
// Check the form data
function validate_form() {
    // Start with an empty array of error messages
    $errors = array();

    // Add an error message if the name is too short
    if (strlen($_POST['my_name']) < 3) {
        $errors[  ] = 'Your name must be at least 3 letters long.';
    }

    // Return the (possibly empty) array of error messages
    return $errors;
}
```



```
// Display the form
function show_form($errors = ) {

   // If some errors were passed in, print them out
   if ($errors) {
       print 'Please correct these errors: <ul><li>';
       print implode('</li><li>', $errors);
       print '</li></ul>';
   }
```

Validating Data

- The code in previous slide takes advantage of the fact that an empty array evaluates to false.
- Required Elements
- To check for required elements, check the element’s length with strlen()

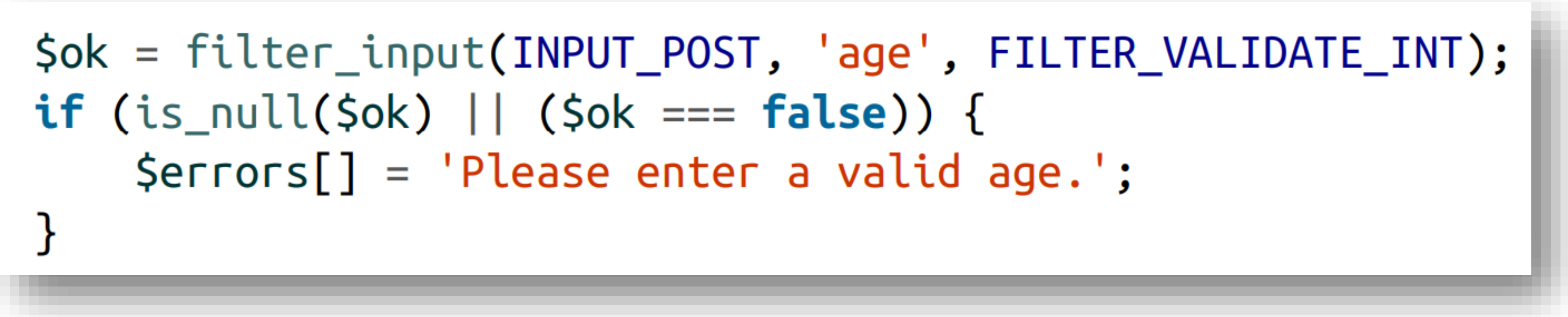


```
if (strlen($_POST['email']) == 0) {
   $errors[] = "You must enter an email address.";
}
```

- Note:
- A if (! $_POST['quantity']) treats a value that evaluates to false as an error. Using strlen() lets users enter a value such as 0 into a required element.

Validating Numeric Values

- use filter_input() function with an appropriate filter, to check for numeric of string
  - The FILTER_VALIDATE_INT checks for integers
  - The FILTER_VALIDATE_FLOAT checks for floating-point numbers



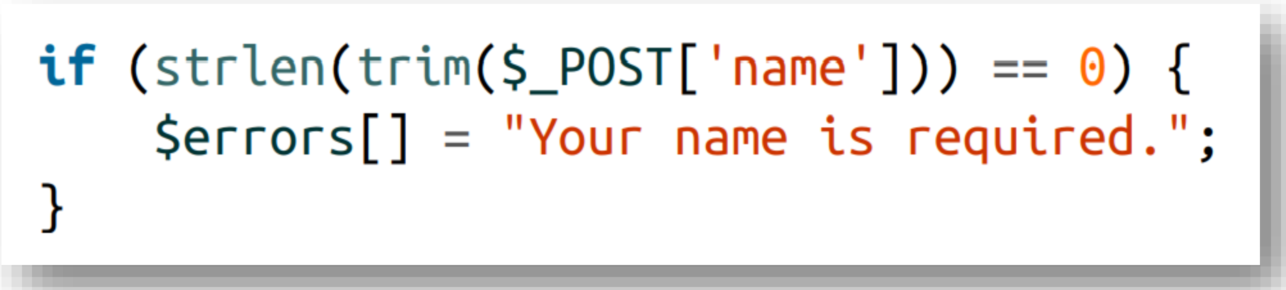
```
$ok = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
if (is_null($ok) || ($ok === false)) {
    $errors[] = 'Please enter a valid age.';
}
```

- INPUT_POST is a predefined constant which refers to the elements of $_POST array.

- If the specified input element is valid, filter_input() returns the value.
- If the specified input element is missing, it returns null.

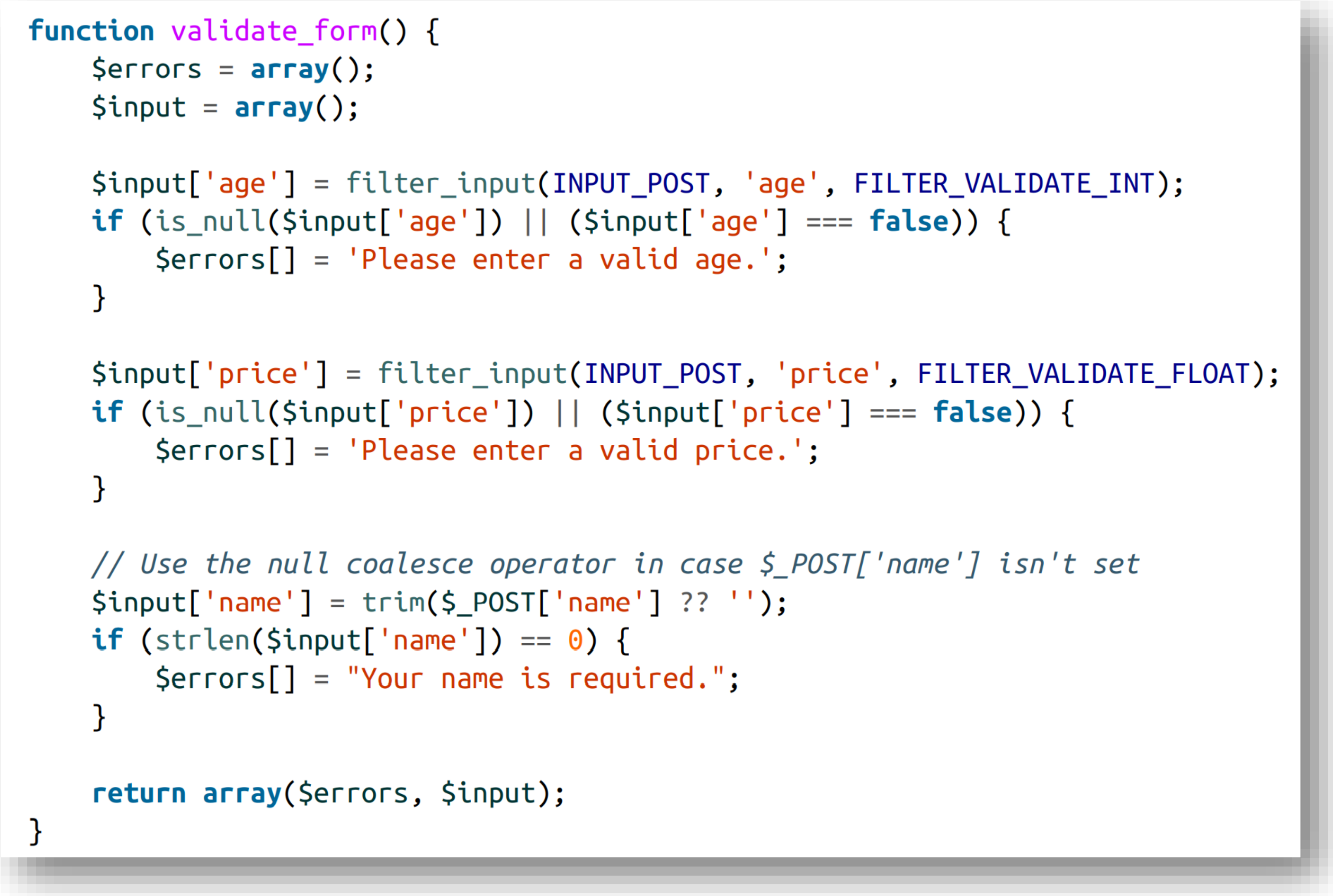
Validating Strings

- Use trim() to remove leading and trailing whitespaces.
- You can combine it with strlen() to disallow an entry of just whitespace characters.



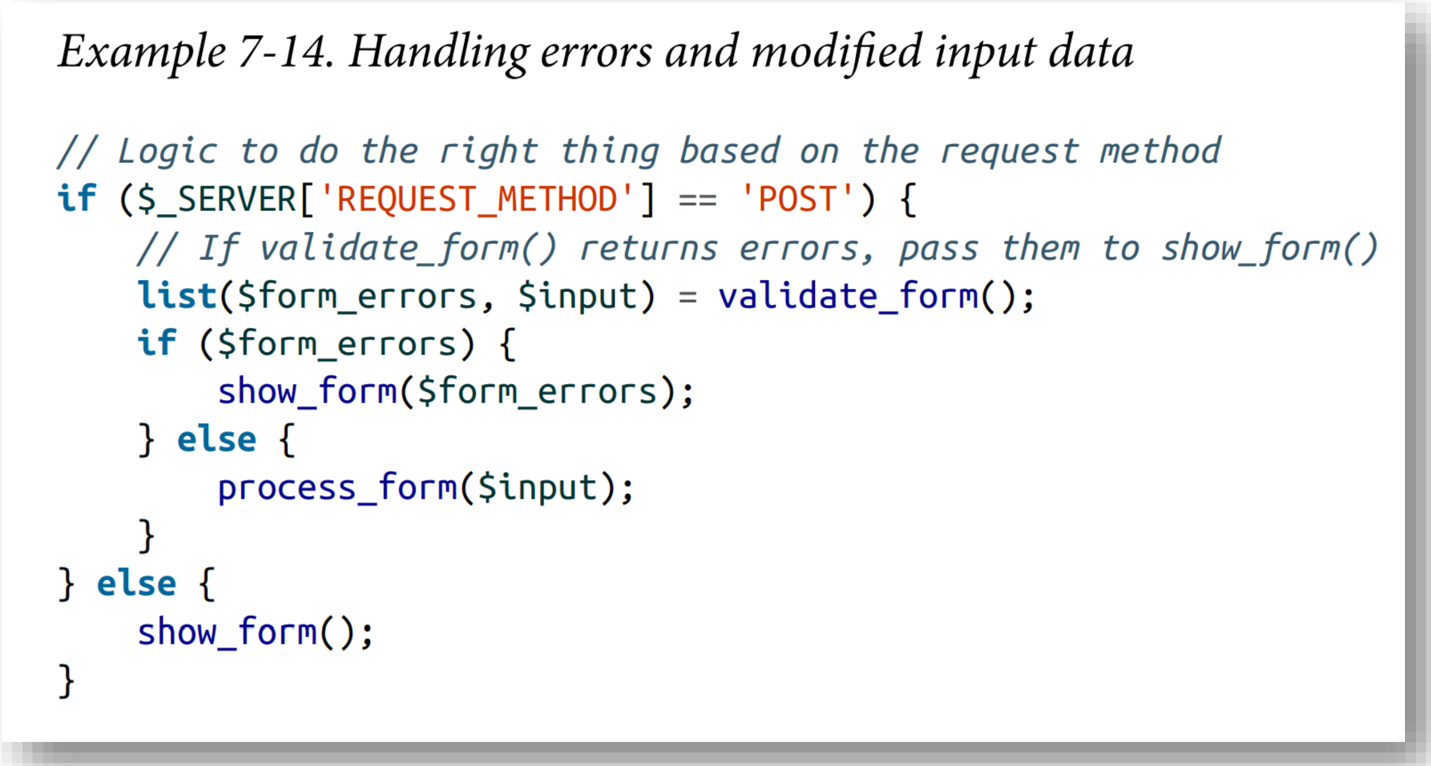
```
if (strlen(trim($_POST['name'])) == 0) {
    $errors[] = "Your name is required.";
}
```

Comprehensive Validation



```
function validate_form() {
    $errors = array();
    $input = array();

    $input['age'] = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT);
    if (is_null($input['age']) || ($input['age'] === false)) {
        $errors[] = 'Please enter a valid age.';
    }

    $input['price'] = filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
    if (is_null($input['price']) || ($input['price'] === false)) {
        $errors[] = 'Please enter a valid price.';
    }

    // Use the null coalesce operator in case $_POST['name'] isn't set
    $input['name'] = trim($_POST['name'] ?? '');
    if (strlen($input['name']) == 0) {
        $errors[] = "Your name is required.";
    }

    return array($errors, $input);
}
```

Comprehensive Validation



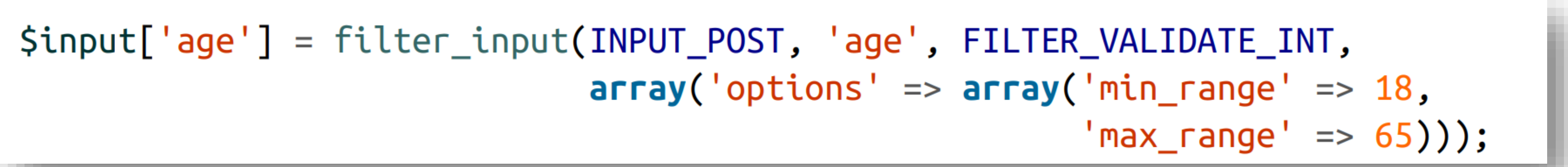
Example 7-14. Handling errors and modified input data

```
// Logic to do the right thing based on the request method
if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    // If validate_form() returns errors, pass them to show_form()
    list($form_errors, $input) = validate_form();
    if ($form_errors) {
        show_form($form_errors);
    } else {
        process_form($input);
    }
} else {
    show_form();
}
```

- list($form_errors, $input) puts the first element of that returned array into the $form_errors variable and the second element into $input.
- Makes the code more readable

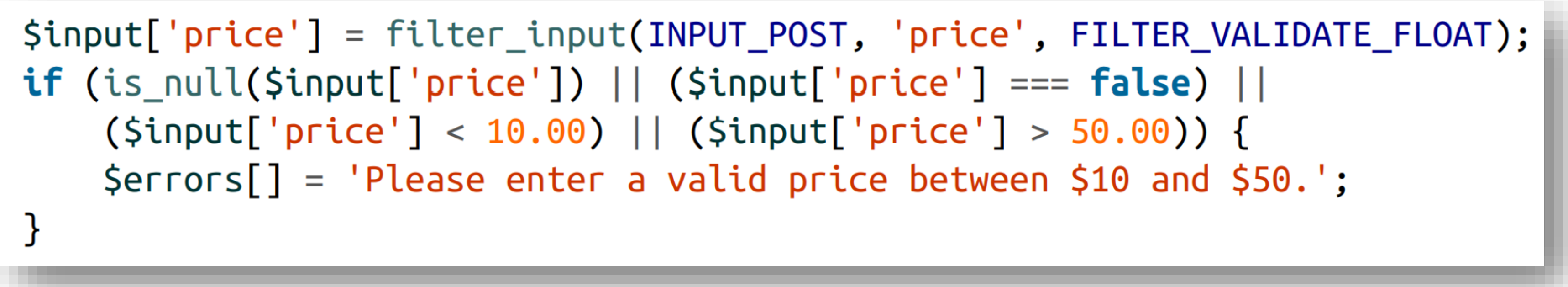
Number Range

- To check whether an integer falls within a certain range, use the min_range and max_range options of the FILTER_VALIDATE_INT filter.



```
$input['age'] = filter_input(INPUT_POST, 'age', FILTER_VALIDATE_INT,
                             array('options' => array('min_range' => 18,
                                                      'max_range' => 65)));
```

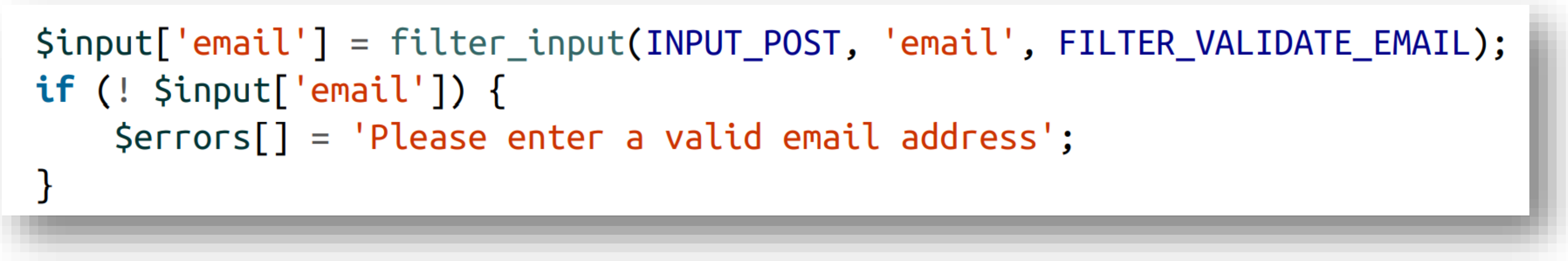
- The FILTER_VALIDATE_FLOAT filter doesn’t support the min_range and max_range options, so you need to do the comparisons yourself



```
$input['price'] = filter_input(INPUT_POST, 'price', FILTER_VALIDATE_FLOAT);
if (is_null($input['price']) || ($input['price'] === false) ||
    ($input['price'] < 10.00) || ($input['price'] > 50.00)) {
    $errors[] = 'Please enter a valid price between $10 and $50.';
}
```

Email Addresses

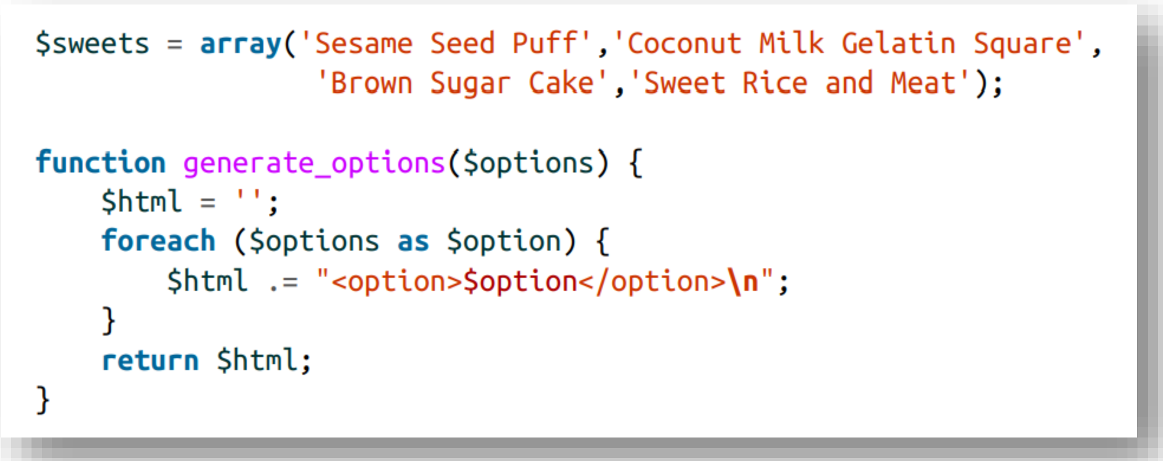
- The FILTER_VALIDATE_EMAIL filter checks strings against the rules for valid email addresse syntax



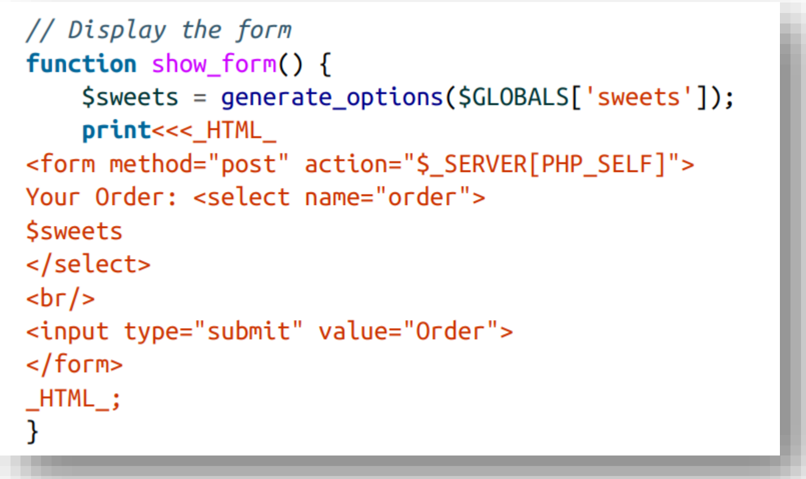
```
$input['email'] = filter_input(INPUT_POST, 'email', FILTER_VALIDATE_EMAIL);
if (! $input['email']) {
    $errors[] = 'Please enter a valid email address';
}
```

<select> Menu

- Although a user can’t submit an off-menu value using common browsers, BUT an attacker can construct a request containing any arbitrary value without using a browser.
- To simplify display and validation of <select> menus:
  - put the menu choices in an array.
  - Then, iterate through that array to display the <select> menu inside the show_form() function.

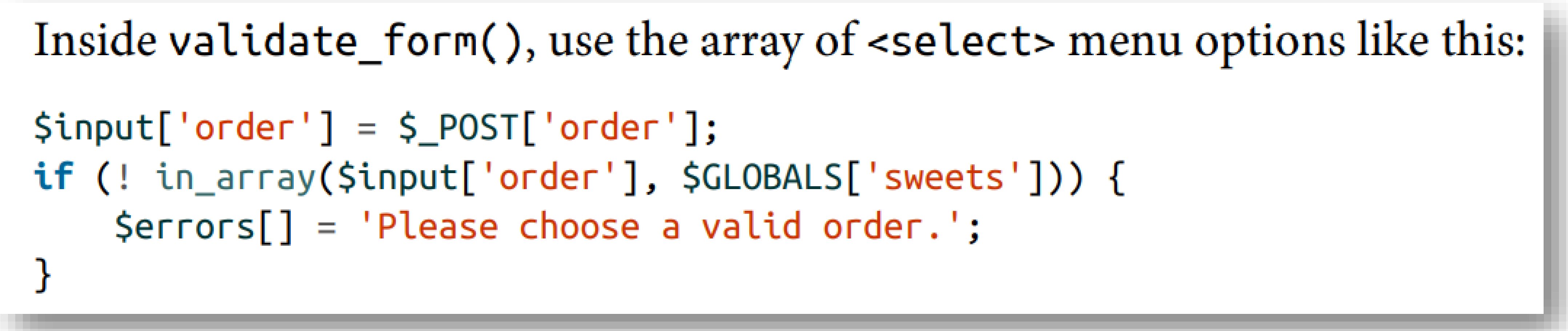


```php
$sweets = array('Sesame Seed Puff','Coconut Milk Gelatin Square',
                'Brown Sugar Cake','Sweet Rice and Meat');

function generate_options($options) {
    $html = '';
    foreach ($options as $option) {
        $html .= "<option>$option</option>\n";
    }
    return $html;
}
```



```php
// Display the form
function show_form() {
    $sweets = generate_options($GLOBALS['sweets']);
    print<<<_HTML_
<form method="post" action="$_SERVER[PHP_SELF]">
Your Order: <select name="order">
$sweets
</select>
<br/>
<input type="submit" value="Order">
</form>
_HTML_;
}
```

<select> Menu

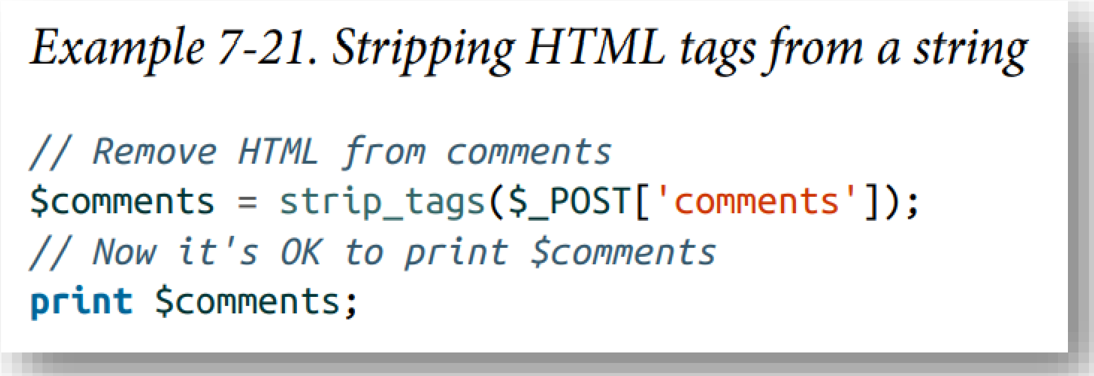


Inside `validate_form()`, use the array of `<select>` menu options like this:

```
$input['order'] = $_POST['order'];
if (! in_array($input['order'], $GLOBALS['sweets'])) {
    $errors[] = 'Please choose a valid order.';
}
```

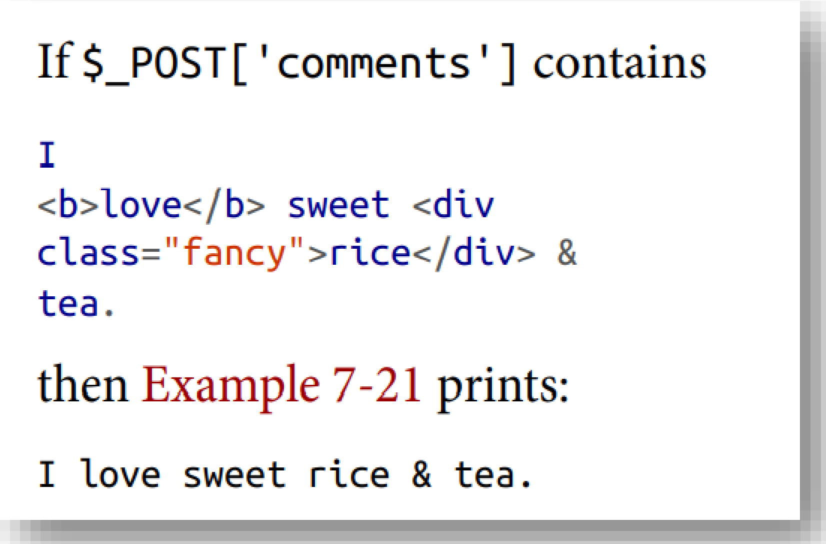
HTML and JavaScript

- Submitted form data that contains HTML or JavaScript can cause big problems.
- Use strip_tags() to remove HTML tags from an input data



Example 7-21. Stripping HTML tags from a string

```
// Remove HTML from comments
$comments = strip_tags($_POST['comments']);
// Now it's OK to print $comments
print $comments;
```



If `$_POST['comments']` contains

```
I
<b>love</b> sweet <div
class="fancy">rice</div> &
tea.
```

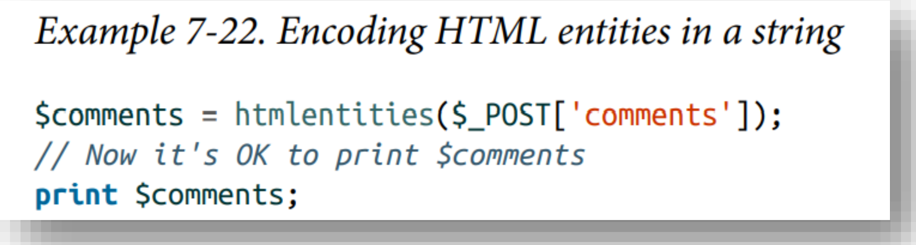
then Example 7-21 prints:

```
I love sweet rice & tea.
```

HTML and JavaScript

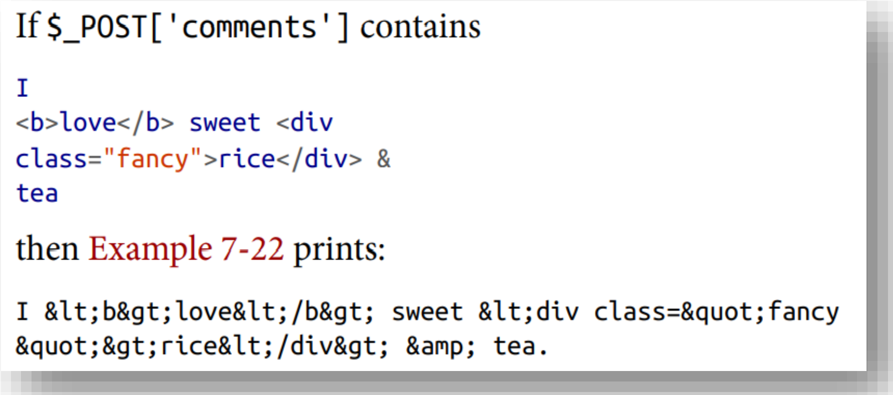
- The strip_tags() behaves poorly with mismatched < and > characters.
- For example, it turns I <3 Monkeys to I because of < character

- Encoding instead of stripping the tags often gives better results.
- Conversts the special characters to HTML entities
- (, &, and ")
  - < to &let;
  - > to &gt;
  - & to &amp;
  - " to &quote;



Example 7-22. Encoding HTML entities in a string

```
$comments = htmlentities($_POST['comments']);
// Now it's OK to print $comments
print $comments;
```



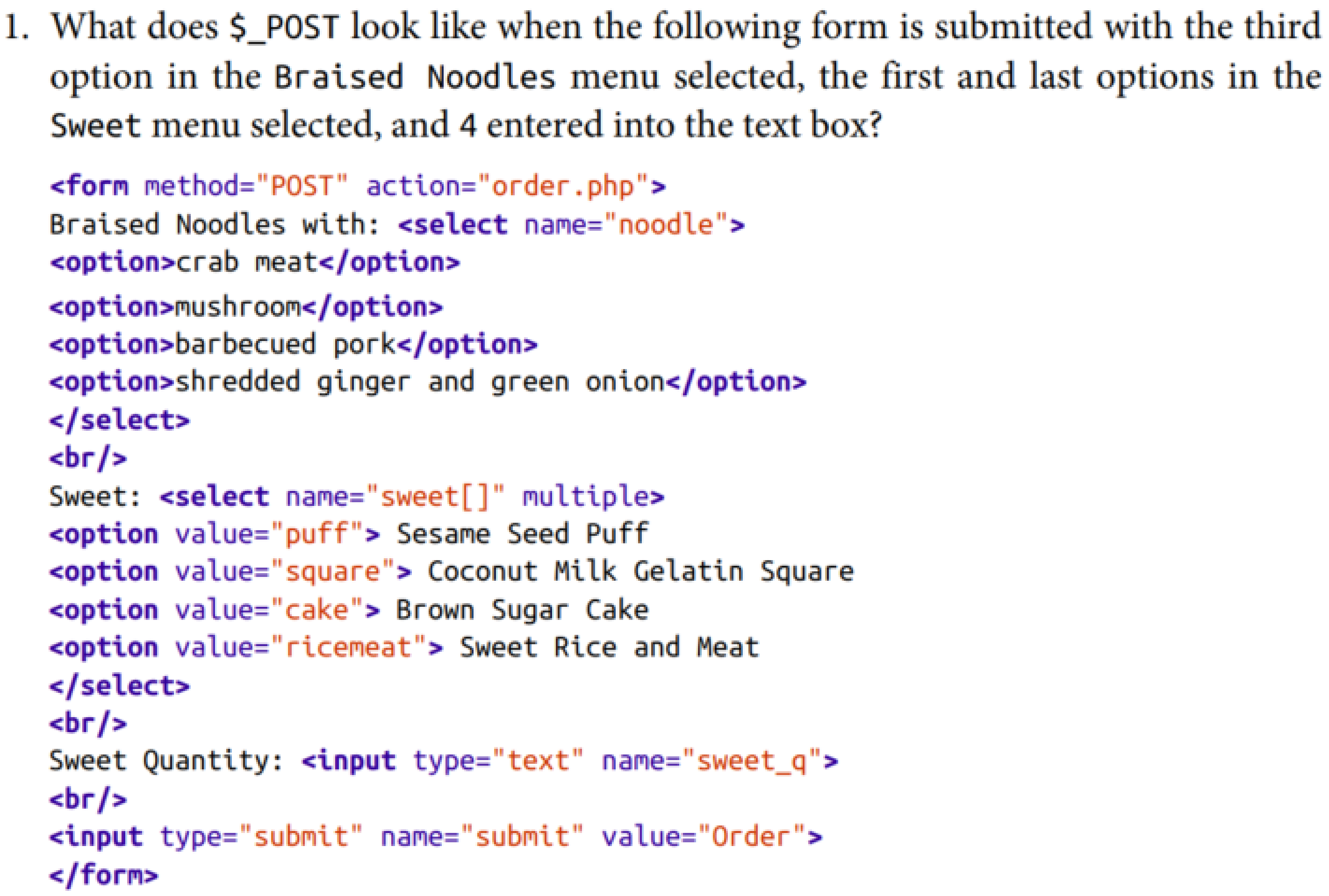
If `$_POST['comments']` contains

```
I
<b>love</b> sweet <div
class="fancy">rice</div> &
tea
```

then Example 7-22 prints:

```
I &lt;b&gt;love&lt;/b&gt; sweet &lt;div class=&quot;fancy
&quot;&gt;rice&lt;/div&gt; &amp; tea.
```

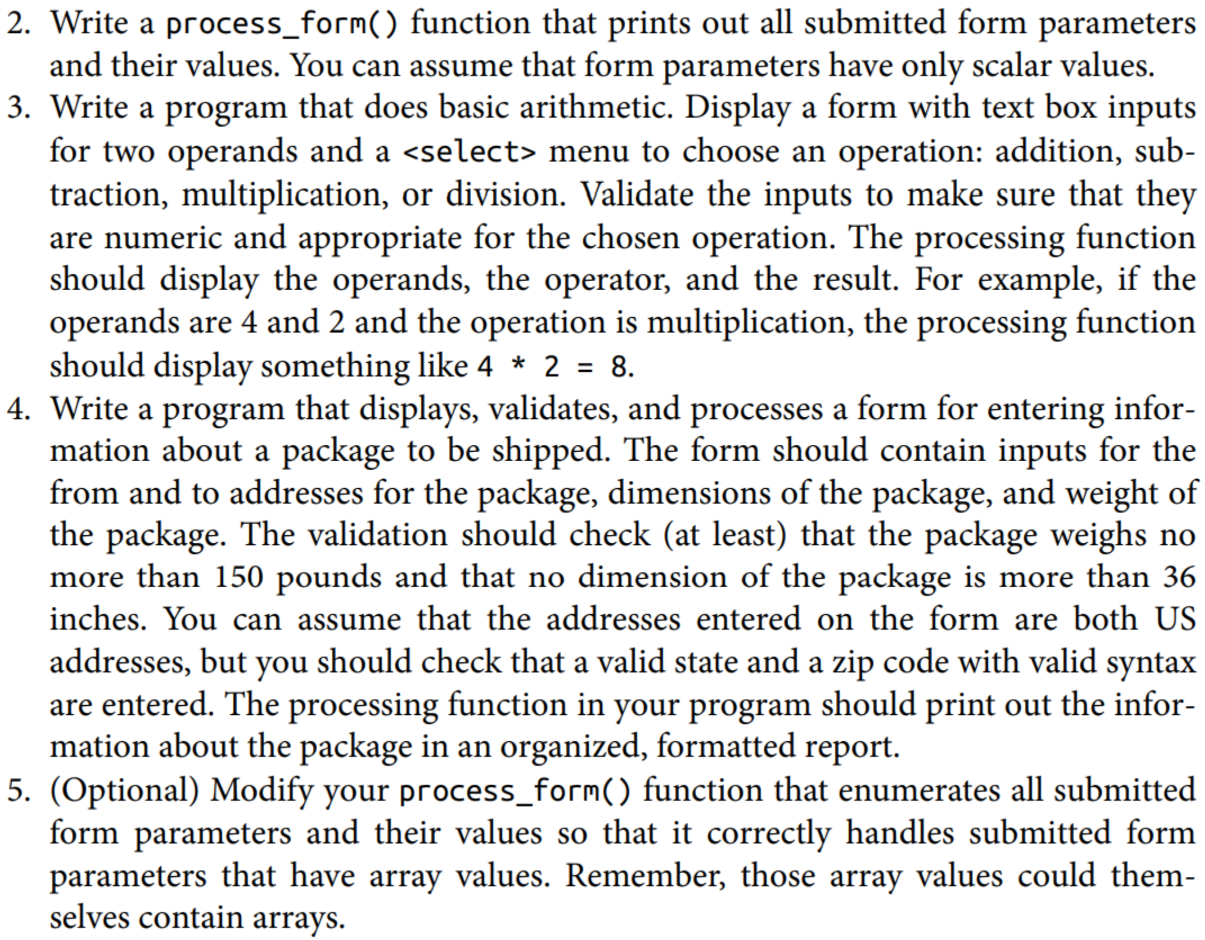
Exercises



1. What does $_POST look like when the following form is submitted with the third option in the Braised Noodles menu selected, the first and last options in the Sweet menu selected, and 4 entered into the text box?

```
<form method="POST" action="order.php">
Braised Noodles with: <select name="noodle">
<option>crab meat</option>
<option>mushroom</option>
<option>barbecued pork</option>
<option>shredded ginger and green onion</option>
</select>
<br/>
Sweet: <select name="sweet[]" multiple>
<option value="puff"> Sesame Seed Puff
<option value="square"> Coconut Milk Gelatin Square
<option value="cake"> Brown Sugar Cake
<option value="ricemeat"> Sweet Rice and Meat
</select>
<br/>
Sweet Quantity: <input type="text" name="sweet_q">
<br/>
<input type="submit" name="submit" value="Order">
</form>
```

Exercises



2. Write a `process_form()` function that prints out all submitted form parameters and their values. You can assume that form parameters have only scalar values.
3. Write a program that does basic arithmetic. Display a form with text box inputs for two operands and a `<select>` menu to choose an operation: addition, subtraction, multiplication, or division. Validate the inputs to make sure that they are numeric and appropriate for the chosen operation. The processing function should display the operands, the operator, and the result. For example, if the operands are 4 and 2 and the operation is multiplication, the processing function should display something like `4 * 2 = 8`.
4. Write a program that displays, validates, and processes a form for entering information about a package to be shipped. The form should contain inputs for the from and to addresses for the package, dimensions of the package, and weight of the package. The validation should check (at least) that the package weighs no more than 150 pounds and that no dimension of the package is more than 36 inches. You can assume that the addresses entered on the form are both US addresses, but you should check that a valid state and a zip code with valid syntax are entered. The processing function in your program should print out the information about the package in an organized, formatted report.
5. (Optional) Modify your `process_form()` function that enumerates all submitted form parameters and their values so that it correctly handles submitted form parameters that have array values. Remember, those array values could themselves contain arrays.