# Error Handling and Validation

Chapter 12

Randy Connolly and Ricardo Hoar

Fundamentals of Web Development

## Objectives

1 What are Errors and Exceptions?

2 PHP Error Reporting

3 PHP Error and Exception Handling

4 Regular Expressions

5 Validating User Input

6 Where to Perform Validation

Section 1 of 6

## WHAT ARE ERRORS AND EXCEPTIONS?

# Types of Errors

- **Expected errors**

  Things that you expect to go wrong. Bad user input, database connection, etc…

- **Programming errors**

  Using undeclared variables, wrong names of functions during calls, etc...

- **Warnings and Notices**

  Problems that generate a PHP warning message but will not halt the execution of the page

- **Fatal errors**

  are serious in that the execution of the page will terminate unless handled in some way

# Exceptions vs Errors

Not the same thing

- An **error** is some type of problem that generates a nonfatal warning message or that generates an error message that terminates the program's execution.
- An **exception** refers to objects that are of type Exception and which are used in conjunction with the object-oriented **try** . . . **catch** language construct for dealing with runtime errors.

Section 2 of 6

## PHP ERROR REPORTING

## PHP error reporting

Lots of control

PHP has a flexible and customizable system for reporting warnings and errors that can be set programmatically at runtime or declaratively at design-time within the **php.ini** file.

There are three main error reporting flags:

- **error_reporting**
- **display_errors**
- **log_errors**

# The error_reporting setting

What is an error?

The **error_reporting** setting specifies which type of errors are to be reported.

It can be set programmatically inside **any** PHP file:

```
error_reporting(E_ALL);
```

It can also be set within the **php.ini** file:

```
error_reporting = E_ALL
```

## The error_reporting setting

Some error reporting constants

| Constant Name | Value | Description |
|---|---|---|
| E_ALL | 8191 | Report all errors and warnings |
| E_ERROR | 1 | Report all fatal runtime errors |
| E_WARNING | 2 | Report all nonfatal runtime errors (that is, warnings) |
|  | 0 | No reporting |

## The display_errors setting

To show or not to show

The **display_error** setting specifies whether error messages should or should not be displayed in the browser.

It can be set programmatically via the **ini_set()** function:

```
ini_set('display_errors','0');
```

It can also be set within the **php.ini** file:

```
display_errors = Off
```

# The log_error setting

To record or not to record

The **log_error** setting specifies whether error messages should or should not be sent to the server error log.

It can be set programmatically via the **ini_set()** function:

```
ini_set('log_errors','1');
```

It can also be set within the **php.ini** file:

```
log_errors = On
```

## The log_error setting

Where to store.

The location to store logs in can be set programatically:

```
ini_set('error_log', '/restricted/my-errors.log');
```

It can also be set within the **php.ini** file:

```
error_log = /restricted/my-errors.log
```

# The log_error setting

error_log()

You can also programmatically send messages to the error log at any time via the **error_log()** function

```
$msg = 'Some horrible error has occurred!';

// send message to system error log (default)
error_log($msg,0);

// email message
error_log($msg,1,'support@abc.com','From: somepage.php@abc.com');

// send message to file
error_log($msg,3, '/folder/somefile.log');
```

Section 3 of 6

## PHP ERROR AND EXCEPTION HANDLING

## Procedural Error Handling

Recall connecting to a database, that there may be an error…

```
$connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME);

$error = mysqli_connect_error();
if ($error != null) {
   // handle the error
   ...
}
```

**LISTING 12.2** Procedural approach to error handling

# OO Exception Handling

Try, catch, finally

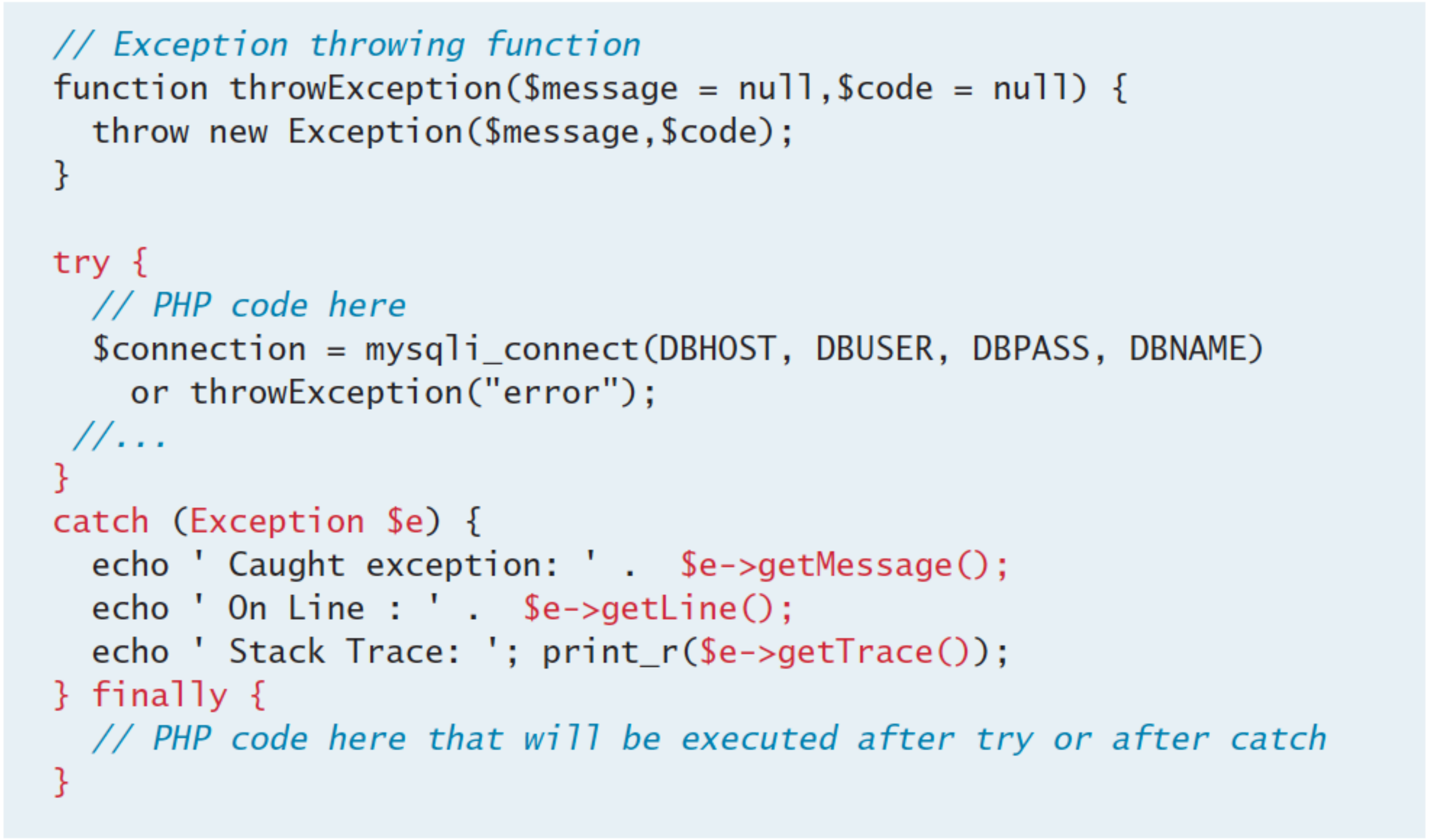
When a runtime error occurs, PHP *throws* an *exception*.

This exception can be *caught* and handled either by the function, class, or page that generated the exception or by the code that called the function or class.

If an exception is not caught, then eventually the PHP environment will handle it by terminating execution with an "Uncaught Exception" message.

# OO Exception Handling

Try, catch, finally



```php
// Exception throwing function
function throwException($message = null,$code = null) {
  throw new Exception($message,$code);
}

try {
  // PHP code here
  $connection = mysqli_connect(DBHOST, DBUSER, DBPASS, DBNAME)
    or throwException("error");
 //...
}
catch (Exception $e) {
  echo ' Caught exception: ' .  $e->getMessage();
  echo ' On Line : ' .  $e->getLine();
  echo ' Stack Trace: '; print_r($e->getTrace());
} finally {
  // PHP code here that will be executed after try or after catch
}
```

**LISTING 12.3** Example of try . . . catch block

## OO Exception Handling

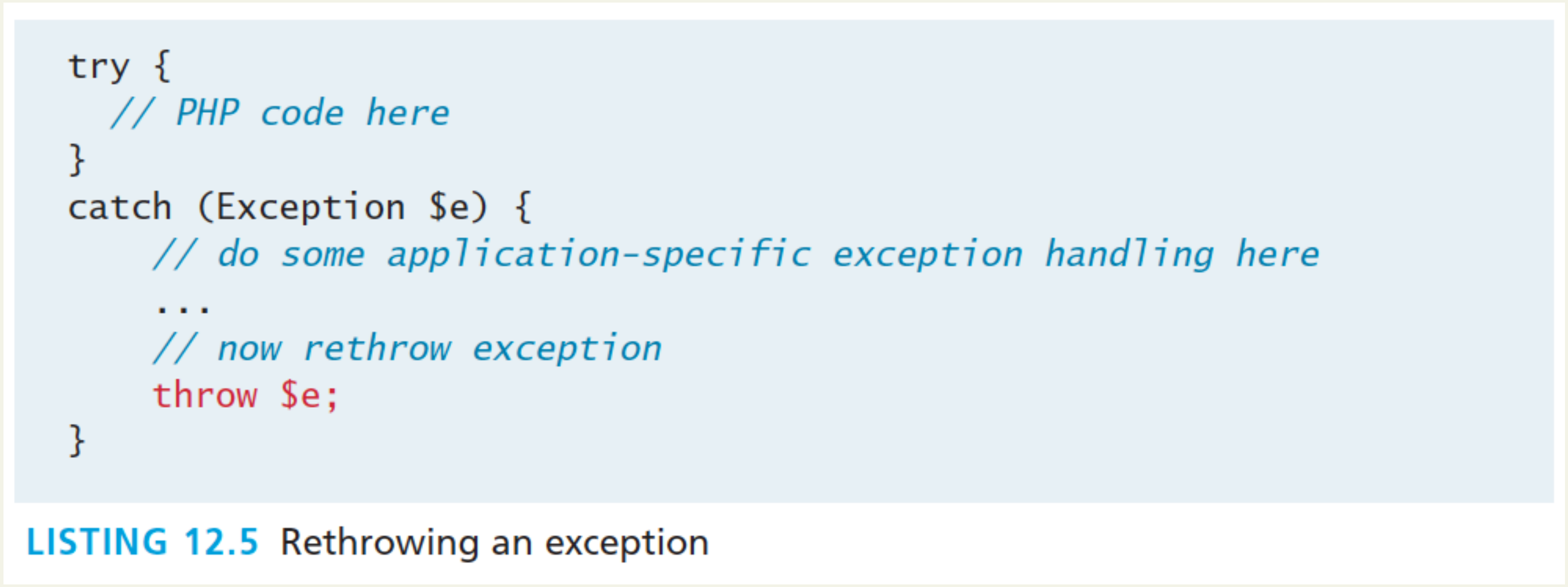
Finally

The **finally** block is optional. Any code within it will always be executed *after* the code in the try or in the catch blocks, even if that code contains a return statement.

The finally block is only available in PHP 5.5 and later

# Throw your own exception

Object oriented way of dealing with the unexpected



```
try {
  // PHP code here
}
catch (Exception $e) {
    // do some application-specific exception handling here
    ...
    // now rethrow exception
    throw $e;
}
```

**LISTING 12.5** Rethrowing an exception

## Custom Handlers

### Error and Exception Handlers

What should a custom error or exception handler do?

It should provide the *developer* with detailed information about the state of the application when the exception occurred, information about the exception, and when it happened.

It should hide any of those details from the regular end user, and instead provide the user with a generic message such as “Sorry but there was a problem”

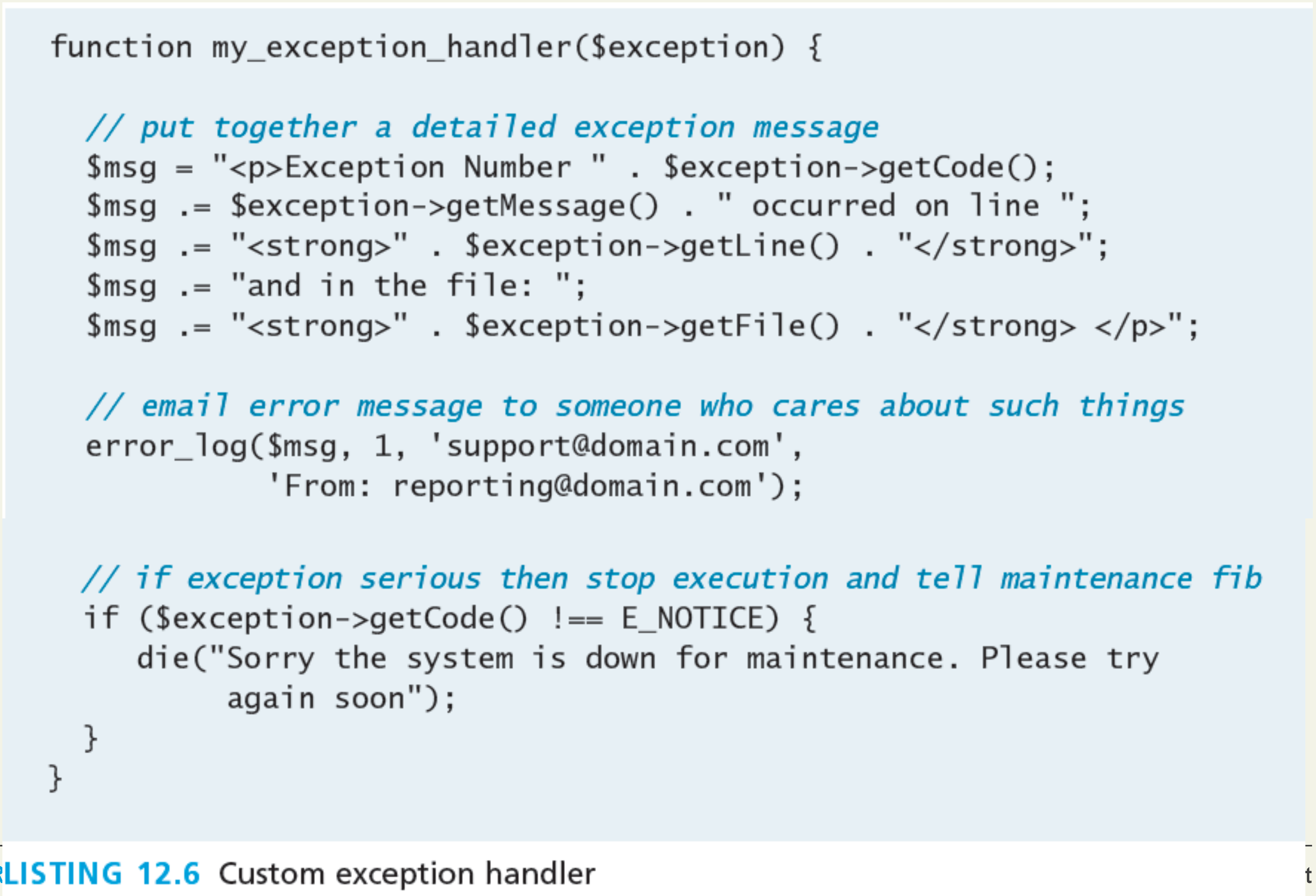
Once a handler function is defined, it must be registered, using the following code:

**set_exception_handler('my_exception_handler');**

the same can be done for errors using a similar function.

## Custom Handlers

Error and Exception Handlers



```
function my_exception_handler($exception) {

   // put together a detailed exception message
   $msg = "<p>Exception Number " . $exception->getCode();
   $msg .= $exception->getMessage() . " occurred on line ";
   $msg .= "<strong>" . $exception->getLine() . "</strong>";
   $msg .= "and in the file: ";
   $msg .= "<strong>" . $exception->getFile() . "</strong> </p>";

   // email error message to someone who cares about such things
   error_log($msg, 1, 'support@domain.com',
             'From: reporting@domain.com');

   // if exception serious then stop execution and tell maintenance fib
   if ($exception->getCode() !== E_NOTICE) {
      die("Sorry the system is down for maintenance. Please try
           again soon");
   }
}
```

LISTING 12.6 Custom exception handler

## Example 1

```php
<?php


// Using a non existing variable
echo $myvar;
echo "Hello!\n";

?>
```

```
$ php esempio1.php
Warning: Undefined variable $myvar in
/Users/vecchio/Documents/didattica/PWEB/php/errori/
esempio1.php on line 5
Hello!
```

## Example 2

```php
<?php

// Calling a method on a non-existing variable
echo $myvar->a_method();
echo "Hello\n";

?>
```

```
Warning: Undefined variable $myvar in
...errori/esempio2.php on line 4

Fatal error: Uncaught Error: Call to a member function
a_method() on null in ...errori/esempio2.php:4
Stack trace:
#0 {main}
  thrown in ..errori/esempio2.php on line 4
```

## Example 3

```php
<?php

// Turn off error reporting
ini_set('error_reporting', 0);

// Warning message is suppressed
echo $myvar;
// Error message is suppressed
echo $myvar->a_method();
// The following line is not executed because of fatal error
// produced by the previous line
echo "Hello\n";

?>
```

No error messages are shown

## Example 4

```php
<?php
function my_error_handler($errno, $errstr) {
  // My custom actions, here just printing
  echo "There was a problem. Error number: "
       . $errno . ", message: " . $errstr . "\n";
}
// Set a customized error handler
set_error_handler("my_error_handler", E_ALL);
// Turn off error reporting
ini_set('error_reporting', 0);
// Warning message is suppressed
echo $myvar;
// Error message is suppressed
echo $myvar->a_method();
// The following line is not executed because of fatal error
// produced by the previous line
echo "Hello\n";
?>
```

```
There was a problem. Error number: 2, message: Undefined variable $myvar
There was a problem. Error number: 2, message: Undefined variable $myvar
```

## Example 5

```
<?php
try {
  echo $myvar;
  echo $myvar->a_method();
  echo "Hello\n";
} catch (Exception $e) {
  echo "In the first catch block\n";
} catch (Error $e) {
  echo "In the second catch block\n";
  echo "Code: " . $e->getCode()
       . ", message: " . $e->getMessage()
       . ", line: " . $e->getLine()
       . ", file: " . $e->getFile() . "\n";
}
echo "Another hello!\n";
?>
```

```
Warning: Undefined variable $myvar in esempio5.php on line 3
Warning: Undefined variable $myvar in esempio5.php on line 4
In the second catch block
Code: 0, message: Call to a member function a_method() on null,
line: 4, file: esempio5.php
Another hello!
```

## Example 6

```php
<?php
try {
  echo $myvar;
  ( // <--
  echo "Hello\n";
} catch (Exception $e) {
  echo "In the first catch block\n";
} catch (Error $e) {
  echo "In the second catch block\n";
  echo "Code: " . $e->getCode()
       . ", message: " . $e->getMessage()
       . ", line: " . $e->getLine()
       . ", file: " . $e->getFile() . "\n";
}
echo "Another hello!\n";
?>
```

```
Parse error: syntax error,
unexpected token "echo" in
esempio6.php on line 6
```

# Checking user input

Notice that this parameter has no value.

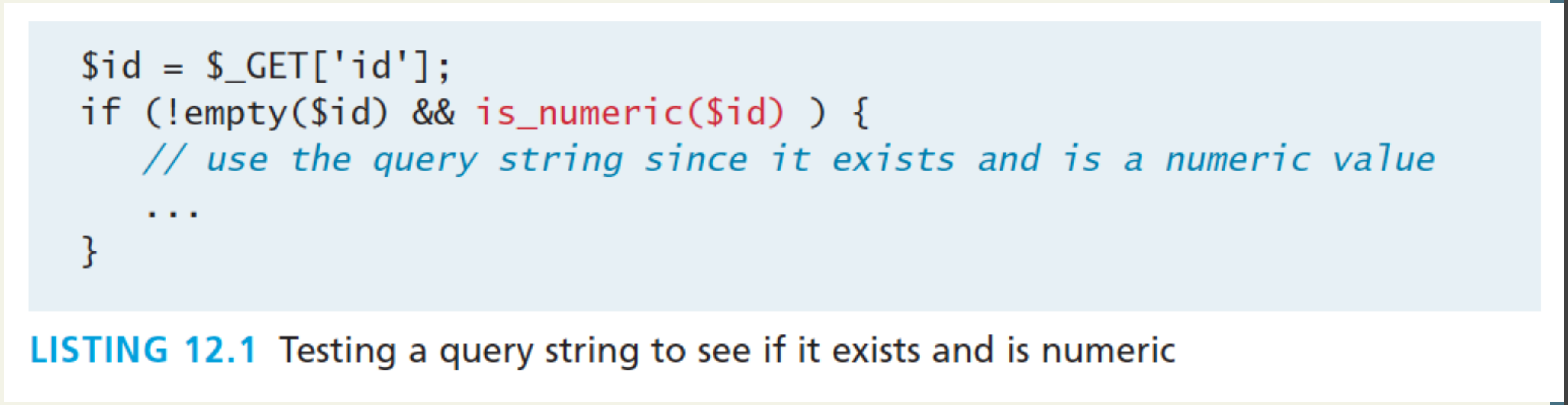
Example query string: `id=0&name1=&name2=smith&name3=%20`

This parameter's value is a space character (URL encoded).

| | | | |
|---|---|---|---|
| `isset($_GET['id'])` | returns | **true** | |
| `isset($_GET['name1'])` | returns | **true** | Notice that a missing value for a parameter is still considered to be `isset`. |
| `isset($_GET['name2'])` | returns | **true** | |
| `isset($_GET['name3'])` | returns | **true** | |
| `isset($_GET['name4'])` | returns | **false** | Notice that only a missing parameter name is considered to be not `isset`. |
| `empty($_GET['id'])` | returns | **true** | Notice that a value of zero is considered to be empty. This may be an issue if zero is a "legitimate" value in the application. |
| `empty($_GET['name1'])` | returns | **true** | |
| `empty($_GET['name2'])` | returns | **false** | |
| `empty($_GET['name3'])` | returns | **false** | Notice that a value of space is considered to be **not** empty. |
| `empty($_GET['name4'])` | returns | **true** | |

# Checking user input

Checking for a number



```
$id = $_GET['id'];
if (!empty($id) && is_numeric($id) ) {
   // use the query string since it exists and is a numeric value
   ...
}
```

**LISTING 12.1** Testing a query string to see if it exists and is numeric

Section 4 of 6

## REGULAR EXPRESSIONS

## Regular Expressions

A **regular expression** is a set of special characters that define a pattern.

They are a type of language that is intended for the matching and manipulation of text.

Regular expressions are a concise way to eliminate the conditional logic that would be necessary to ensure that input data follows a specific format.

## Regular Expressions

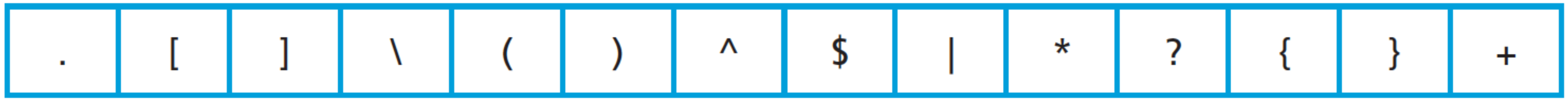
Syntax

A regular expression consists of two types of characters: **literals** and **metacharacters**.

- A **literal** is a character you wish to match in the target
- A **metacharacter** is a special symbol that acts as a command to the regular expression parser

# Regular Expressions

Characters with Special Meaning



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| . | [ | ] | \ | ( | ) | ^ | $ | \| | * | ? | { | } | + |

TABLE 12.2 Regular Expression Metacharacters (i.e., Characters with Special Meaning)

To use a metacharacter as a literal, you will need to escape it by prefacing it with a backslash (**\**)

# Regular Expressions

Table of typical patterns

| Expression | Description |
|---|---|
| ^ … $ | If used at the very start and end of the regular expression, it means that the entire string (and not just a substring) must match the rest of the regular expression contained between the ^ and the $ symbols. |
| \t | Matches a tab character. |
| \n | Matches a new line character. |
| . | Matches any character other than \n. |
| [qwerty] | Matches any single character of the set contained within the brackets. |
| [^qwerty] | Matches any single character not contained within the brackets. |
| [a-z] | Matches any single character within range of characters. |
| \w | Matches any word character. Equivalent to [a-zA-Z0-9]. |
| \W | Matches any nonword character. |
| \s | Matches any white-space character. |
| \S | Matches any nonwhite-space character. |
| \d | Matches any digit. |
| \D | Matches any nondigit. |
| * | Indicates zero or more matches. |
| + | Indicates one or more matches. |
| ? | Indicates zero or one match. |
| {n} | Indicates exactly n matches. |
| {n,} | Indicates n or more matches. |
| {n,m} | Indicates at least n but no more than m matches. |
| \| | Matches any one of the terms separated by the \| character. Equivalent to Boolean OR. |
| () | Groups a subexpression. Grouping can make a regular expression easier to understand. |

# Regular Expressions

Building one example

Consider a regular expression that would match a North American phone number without the area code.

A valid number contains three numbers, followed by a dash, followed by four numbers without any other character.

The regular expression for this would be:

```
^\d{3}–\d{4}$
```

# Regular Expressions

three numbers, followed by a dash, followed by four numbers

**^\d{3}–\d{4}$**

- The dash is here a literal character (can be used as a metacharacter in ranges but not here as it is not within **[]**); the rest are all metacharacters
- The **^** and **$** symbol indicate the beginning and end of the string, respectively
- The metacharacter **\d** indicates a digit, while the metacharacters **{3}** and **{4}** indicate three and four repetitions of the previous match (i.e., a digit), respectively

# Regular Expressions

three numbers, followed by a dash, followed by four numbers

A more sophisticated regular expression for a phone number would not allow the first digit in the phone number to be a zero ("0") or a one ("1").

The modified regular expression for this would be:

**^[2-9]\d{2}–\d{4}$**

## Regular Expressions

Any number (but 0,1), then 2 more, a dash and 4 more.

```
^[2-9]\d{2}–\d{4}$
```

- The **[2-9]** metacharacter indicates that the first character must be a digit within the range 2 through 9
- Since only two more numbers are needed the pattern **\d{3}** becomes **\d{2}**

# Regular Expressions

Allow a space, period, or dash in the number.

We can make our regular expression a bit more flexible by allowing either a single space (440 6061), a period (440.6061), or a dash (440-6061) between the two sets of numbers.

We can do this via the **[]** metacharacter:

**^[2-9]\d{2}[–\s\.]\d{4}$**

# Regular Expressions

Allow a space, period, or dash in the number.

```
^[2-9]\d{2}[–\s\.]\d{4}$
```

This expression indicates that the fourth character in the input must match one of the three characters contained within the square brackets

**–** matches a dash (since it is at the beginning of the character class [] there is no need to escape it), **\s** matches a white space, and **\.** matches a period

```
^[2-9]\d{2}[\–\s\.]\d{4}$
```

Same as above. In other cases, we must escape both **-** and **.**

# Regular Expressions

Allow multiple spaces

If we want to allow multiple spaces (but only a single dash or period) in our number:

**^[2-9]\d{2}[–\s\.]\s*\d{4}$**

The metacharacter sequence **\s*** matches zero or more white spaces.

# Regular Expressions

How about area code

To allow the area code to be

| | |
|---|---|
| • Surrounded by Brackets | (403) 440-6061 |
| • Separated with spaces | 403 440 6061 |
| • A Dash | 403-440-6061 |
| • A Period | 403.440.6061 |

```
^\(?\s*\d{3}\s*[\)–\.]?\s*[2-9]\d{2}\s*[–\.]\s*\d{4}$
```

## Regular Expressions

How about area code

**^\(?\s*\d{3}\s*[\)–\.]?\s*[2-9]\d{2}\s*[–\.]\s*\d{4}$**

The expression now matches

- zero or one "(" characters **\(?**
- zero or more spaces **\s***three digits **\d{3}**
- zero or more spaces **\s***
- either a ")" a "-", or a "." character **[\)-\.]?**
- zero or more spaces **\s***

# Regular Expressions

How about area code

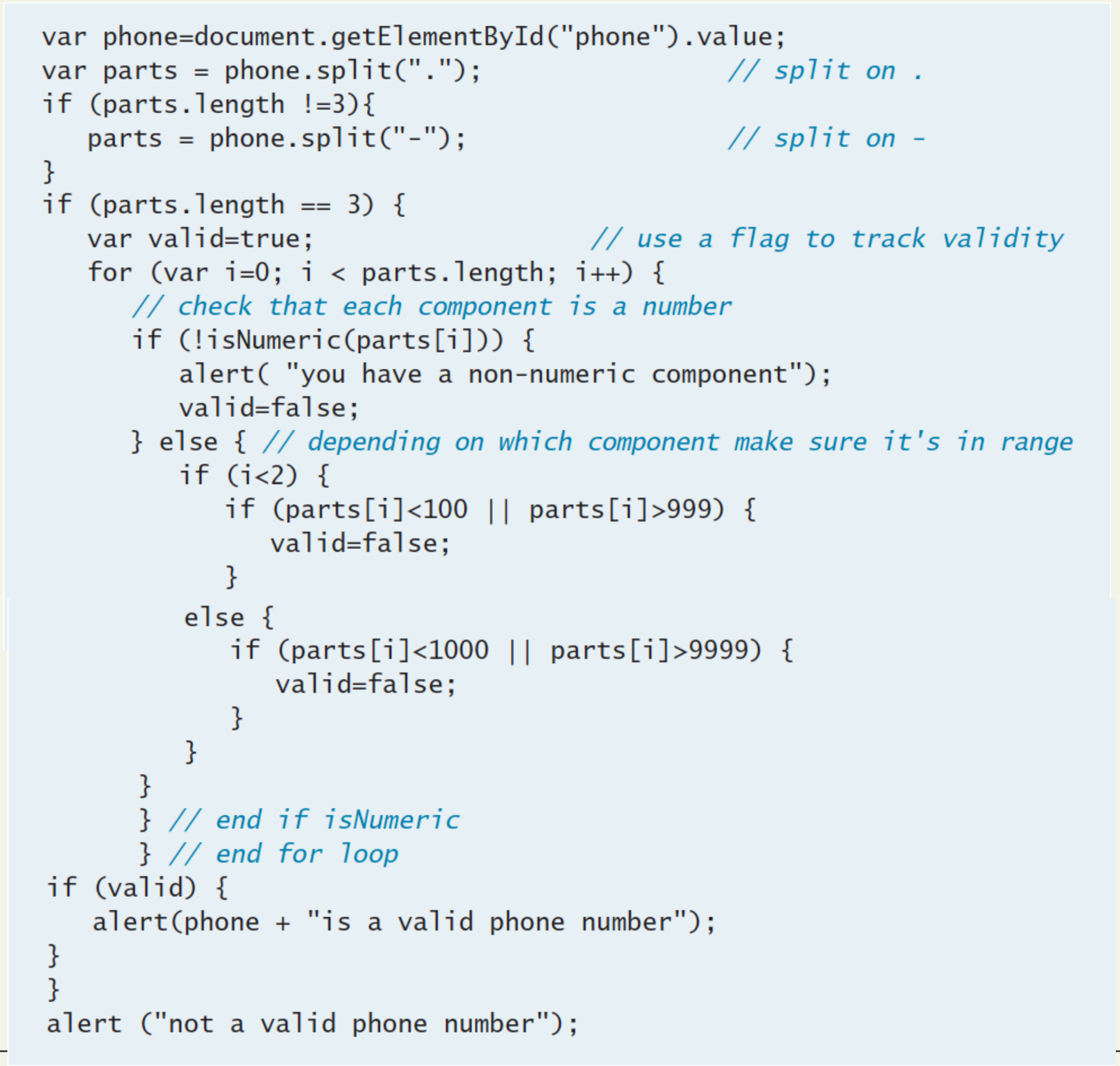
Finally, to make the area code optional we will group the area code by surrounding the area code subexpression within grouping metacharacters— which are "(" and ")"— and then make the group optional using the ? metacharacter.

```
^(\(?\s*\d{3}\s*[\)–\.]?\s*)?[2-9]\d{2}\s*[–\.]\s*\d{4}$
```

This may seem frightening, but compare to :

# Regular Expressions Alternative

`^(\(?\s*\d{3}\s*[\)–\.]?\s*)?[2-9]\d{2}\s*[–\.]\s*\d{4}$`



```
var phone=document.getElementById("phone").value;
var parts = phone.split(".");                      // split on .
if (parts.length !=3){
   parts = phone.split("-");                       // split on -
}
if (parts.length == 3) {
   var valid=true;                        // use a flag to track validity
   for (var i=0; i < parts.length; i++) {
      // check that each component is a number
      if (!isNumeric(parts[i])) {
         alert( "you have a non-numeric component");
         valid=false;
      } else { // depending on which component make sure it's in range
         if (i<2) {
            if (parts[i]<100 || parts[i]>999) {
               valid=false;
            }
         else {
            if (parts[i]<1000 || parts[i]>9999) {
               valid=false;
            }
         }
      }
      } // end if isNumeric
      } // end for loop
 if (valid) {
    alert(phone + "is a valid phone number");
 }
 }
 alert ("not a valid phone number");
```

LISTING 12.7 A phone number validation script without regular expressions

# Some Common Regular Expr.

| Regular Expression | Description |
|---|---|
| ^\S{0,8}$ | Matches 0 to 8 nonspace characters. |
| ^\w{8,16}$ | Simple password expression. The password must be at least 8 characters but no more than 16 characters long. |
| ^\d{5}(-\d{4})?$ | American zip code. |
| ^((0[1-9])\|(1[0-2]))\/(\d{4})$ | Month and years in format mm/yyyy. |
| ^(.+)@([^\.].*)\.([a-z]{2,})$ | Email validation based on current standard naming rules. |
| ^((http\|https)://)?([\w-] +\.)+[\w]+(/[\w- ./?]*)?$ | URL validation. After either http:// or https://, it matches word characters or hyphens, followed by a period followed by either a forward slash, word characters, or a period. |
| ^4\d{3}[\s\-]d{4}[\s\-] d{4}[\s\-]d{4}$ | Visa credit card number |
| ^5[1-5]\d{2}[\s\-]d{4}[\s\-] d{4}[\s\-]d{4}$ | Mastercard credit card number |

## Regex is everywhere

Including MySQL

MySQL also supports regular expressions through the REGEXP operator.

For instance, the following SQL statement matches all art works whose title contains one or more numeric digits:

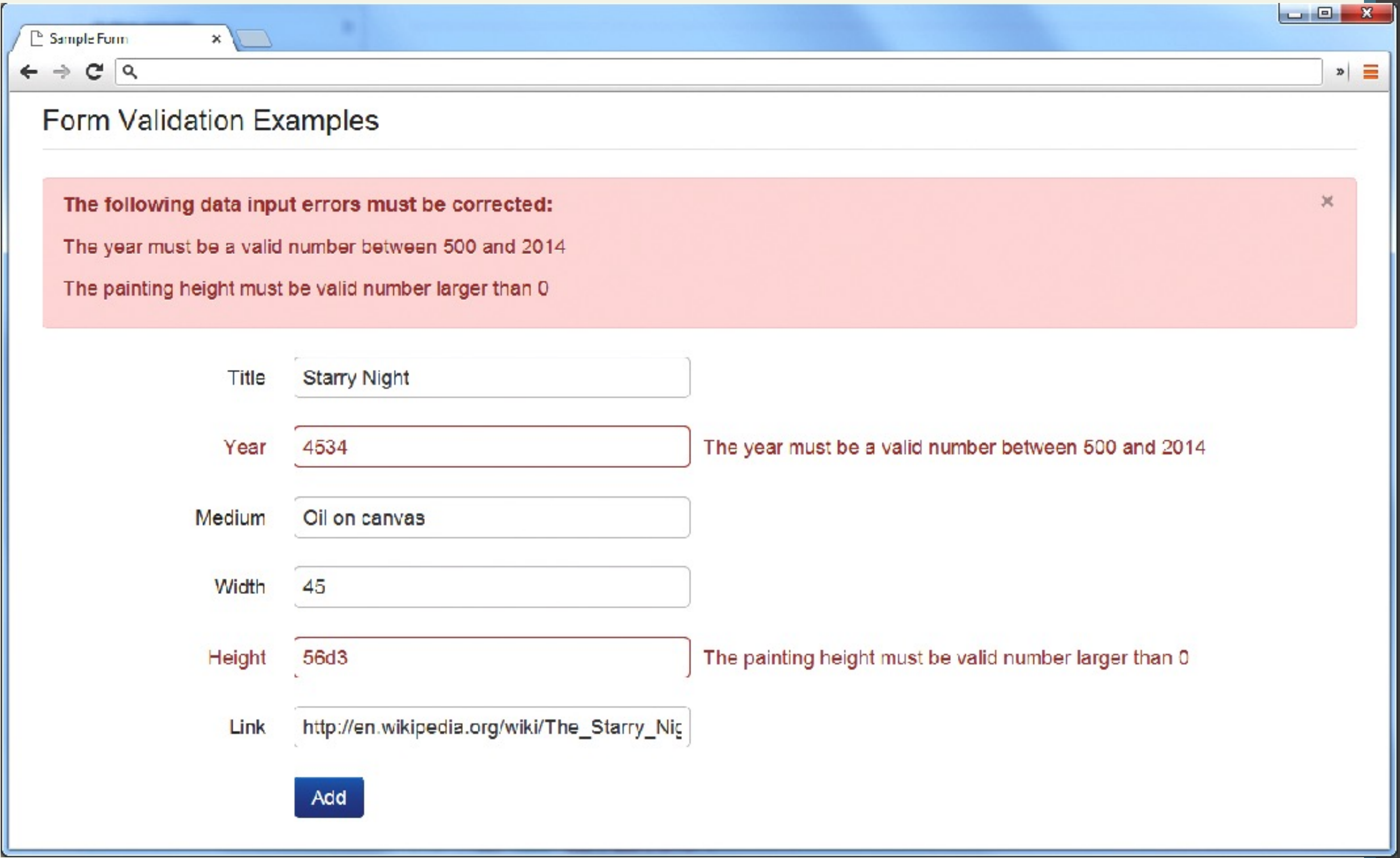
```
SELECT * FROM ArtWorks WHERE Title REGEXP '[0-9]+'
```

Section 5 of 6

## VALIDATING USER INPUT

# Notifying the User

What's wrong, where is it, and how to fix it.


Sample Form

Form Validation Examples

**The following data input errors must be corrected:**

The year must be a valid number between 500 and 2014

The painting height must be valid number larger than 0

| | | |
|---|---|---|
| Title | Starry Night | |
| Year | 4534 | The year must be a valid number between 500 and 2014 |
| Medium | Oil on canvas | |
| Width | 45 | |
| Height | 56d3 | The painting height must be valid number larger than 0 |
| Link | http://en.wikipedia.org/wiki/The_Starry_Nig | |

## Types of Input Validation

- **Required information**. Some data fields just cannot be left empty.
- **Correct data type**. Some input fields must follow the rules for its data type in order to be considered valid.
- **Correct format**. Some information, such as postal codes, credit card numbers, and social security numbers have to follow certain pattern rules.

## Types of Input Validation

Continued

- **Comparison**. Perhaps the most common example of this type of validation is entering passwords: most sites require the user to enter the password twice to ensure the two entered values are identical.
- **Range check**. Information such as numbers and dates have infinite possible values. However, most systems need numbers and dates to fall within realistic ranges.
- **Custom**. Some validations are more complex and are unique to a particular application

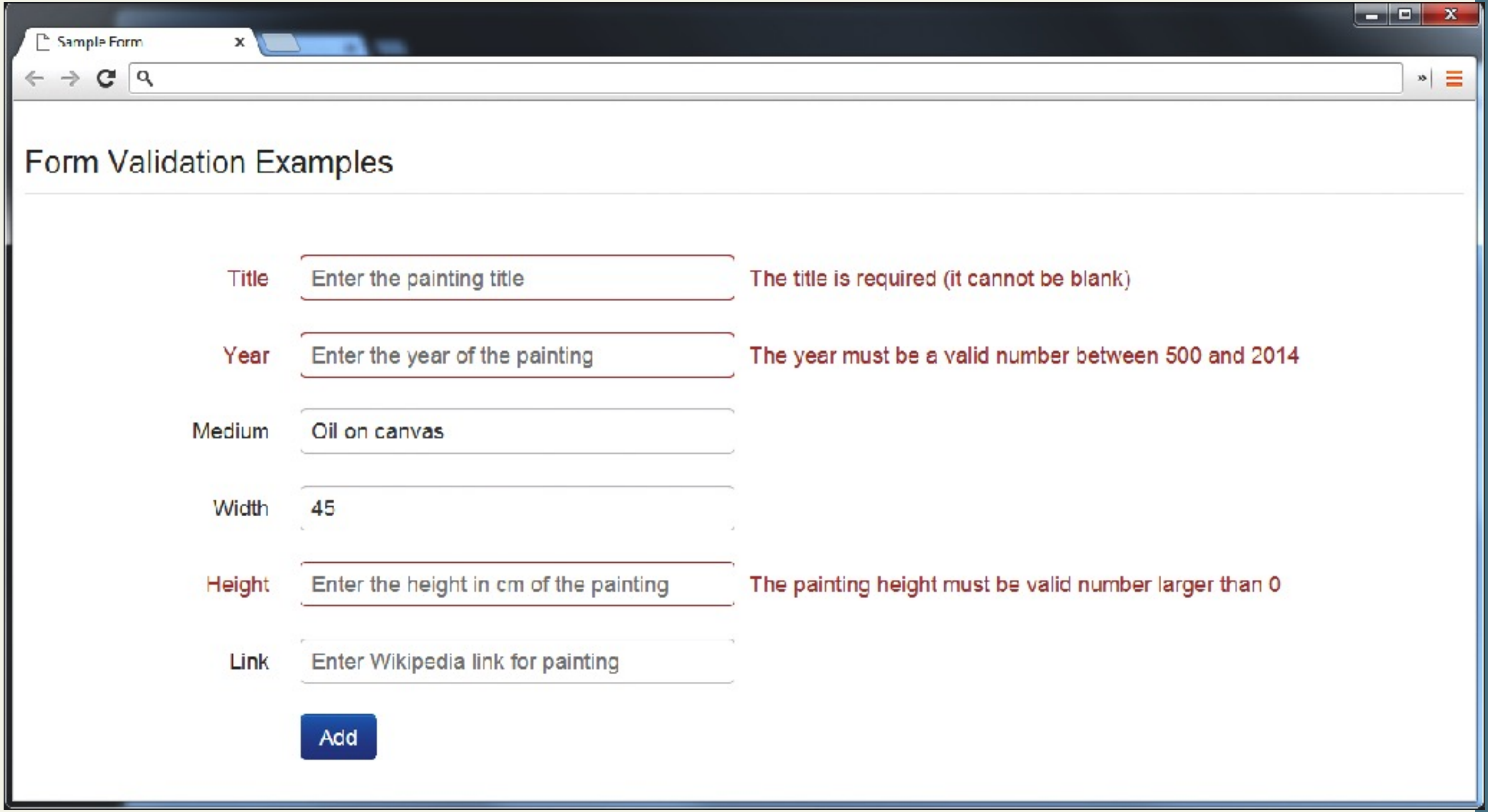
## Notifying the User

We found an error, now what?

- **What is the problem?** Users do not want to read lengthy messages to determine what needs to be changed. They need to receive a visually clear and textually concise message.
- **Where is the problem?** Some type of error indication should be located near the field that generated the problem.
- **If appropriate, how do I fix it?** For instance, don't just tell the user that a date is in the wrong format, tell him or her what format you are expecting, such as "The date should be in *yy/mm/dd* format."

## Another illustrative examples

What's wrong, where is it, and how to fix it.



Form Validation Examples

| | | |
|---|---|---|
| Title | Enter the painting title | The title is required (it cannot be blank) |
| Year | Enter the year of the painting | The year must be a valid number between 500 and 2014 |
| Medium | Oil on canvas | |
| Width | 45 | |
| Height | Enter the height in cm of the painting | The painting height must be valid number larger than 0 |
| Link | Enter Wikipedia link for painting | |

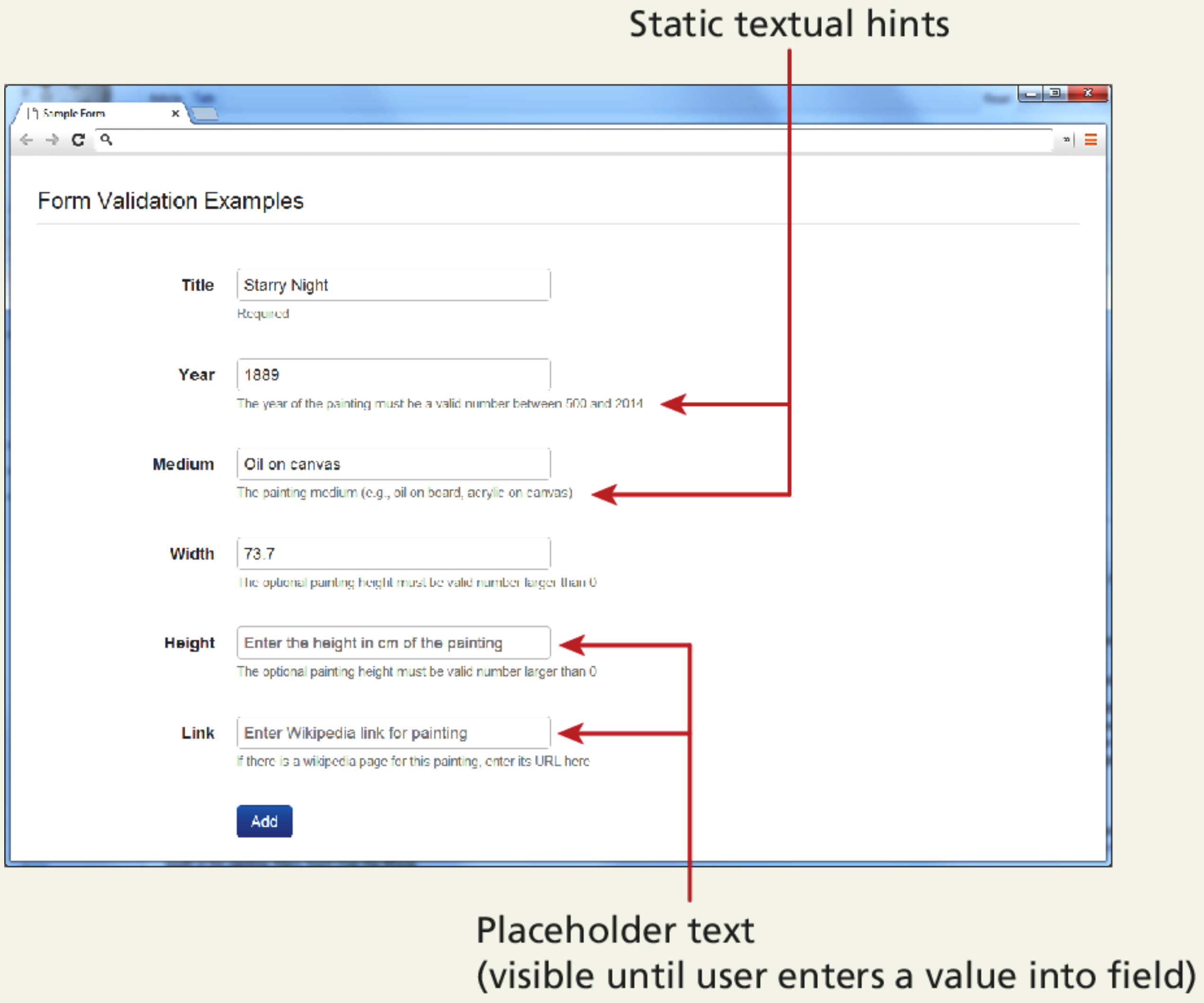
## How to reduce validation errors

An ounce of prevention is worth a pound of cure

- Using pop-up JavaScript alert (or other popup) messages
- Provide textual hints to the user on the form itself
- Using tool tips to display context-sensitive help about the expected input

# How to reduce validation errors

An ounce of prevention is worth a pound of cure



Form Validation Examples

| | |
|---|---|
| Title | Starry Night |
| | Required |
| Year | 1889 |
| | The year of the painting must be a valid number between 500 and 2014 |
| Medium | Oil on canvas |
| | The painting medium (e.g., oil on board, acrylic on canvas) |
| Width | 73.7 |
| | The optional painting height must be valid number larger than 0 |
| Height | Enter the height in cm of the painting |
| | The optional painting height must be valid number larger than 0 |
| Link | Enter Wikipedia link for painting |
| | If there is a wikipedia page for this painting, enter its URL here |

Add

```
<input type="text" … placeholder="Enter the height ...">
```

## How to reduce validation errors

HTML 5 input types

Many user input errors can be eliminated by choosing a better data entry type than the standard

**<input type="text">**

If you need to get a date from the user, use the HTML5

**<input type="date”>**

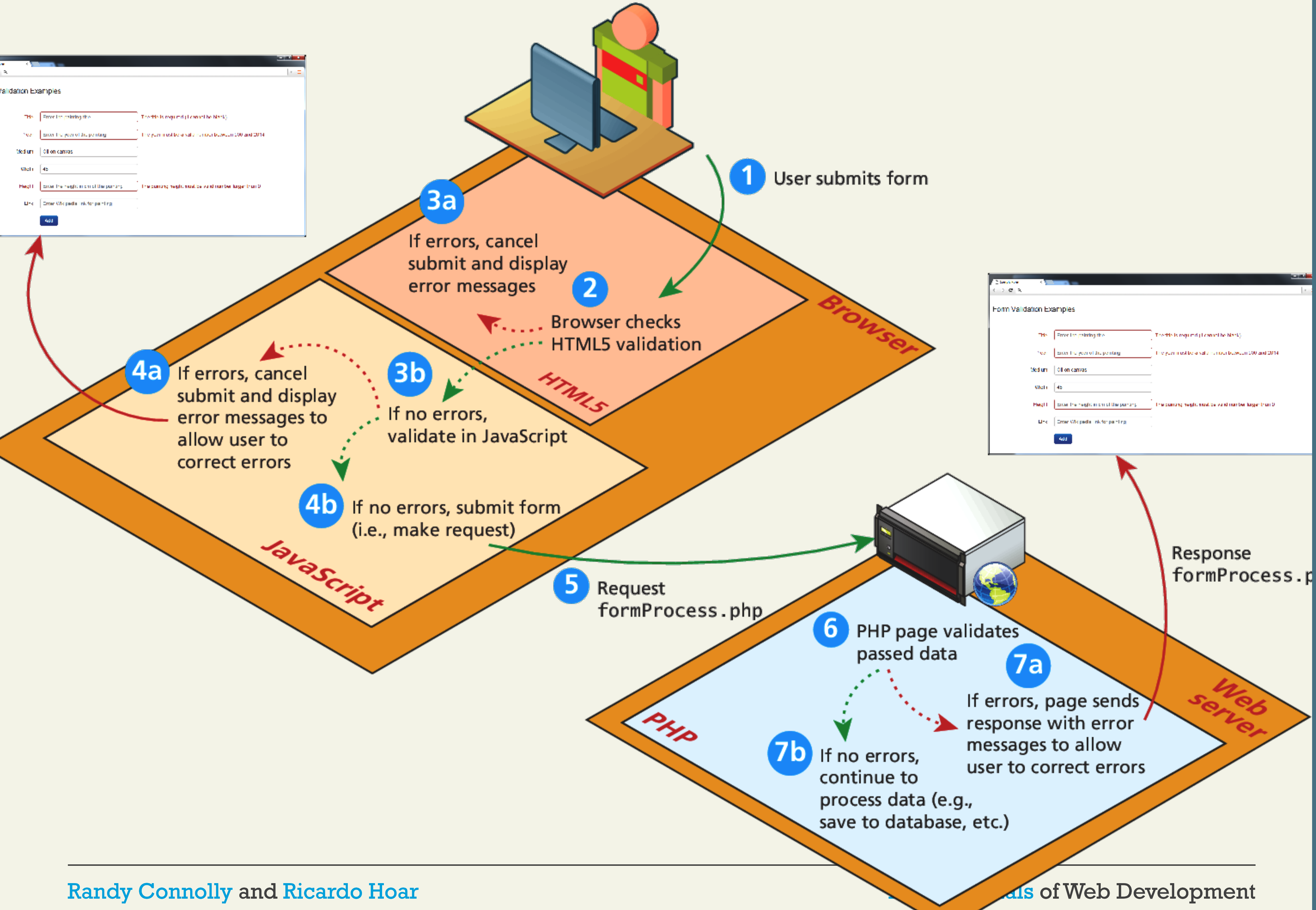
If you need a number, use the HTML5

**<input type="number">**

Section 6 of 6

## WHERE TO PERFORM VALIDATION

# Where to Validate?


1 User submits form
2 Browser checks HTML5 validation
3a If errors, cancel submit and display error messages
3b If no errors, validate in JavaScript
4a If errors, cancel submit and display error messages to allow user to correct errors
4b If no errors, submit form (i.e., make request)
5 Request formProcess.php
6 PHP page validates passed data
7a If errors, page sends response with error messages to allow user to correct errors
7b If no errors, continue to process data (e.g., save to database, etc.)
Response formProcess.p
Browser
HTML5
JavaScript
Web server
PHP
Form Validation Examples
Add

## Where to Validate?

So many places

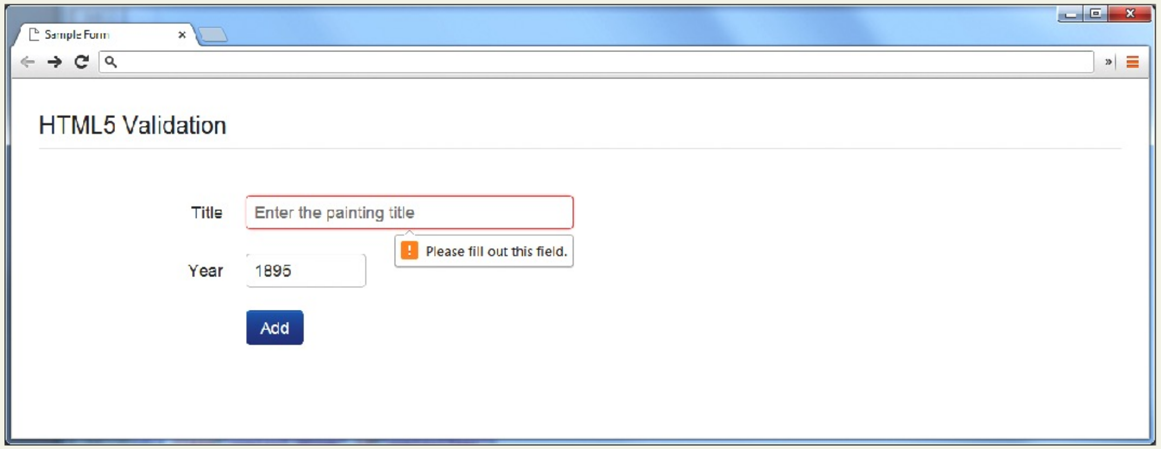
- Client-side using HTML5
- Client-Side using JavaScript
- **Server-Side using PHP**

While both client and server side validation is ideal, you must know that client-side scripts are not guaranteed to be executed. Therefore **you must always perform server-side validation**.

## HTML5 validation

Client-Side

The ***required*** attribute can be added to an input element, and browsers that support it will perform their own validation and message.



To disable HTML form validation

<form id="sampleForm" method=". . ." action=". . ." *novalidate*>

# JavaScript validation

Client-Side

Consider that we want to validate on a form submit.

```
function init() {
	var sampleForm = document.getElementById('sampleForm');
	sampleForm.onsubmit = validateForm;

}
 // call the init function once all the html has been loaded
window.onload = init;
```

# JavaScript validation

Client-Side

For instance, to check if the value in the form's password input element is between 9 and 17 characters (and the first is a letter), the JavaScript would be:

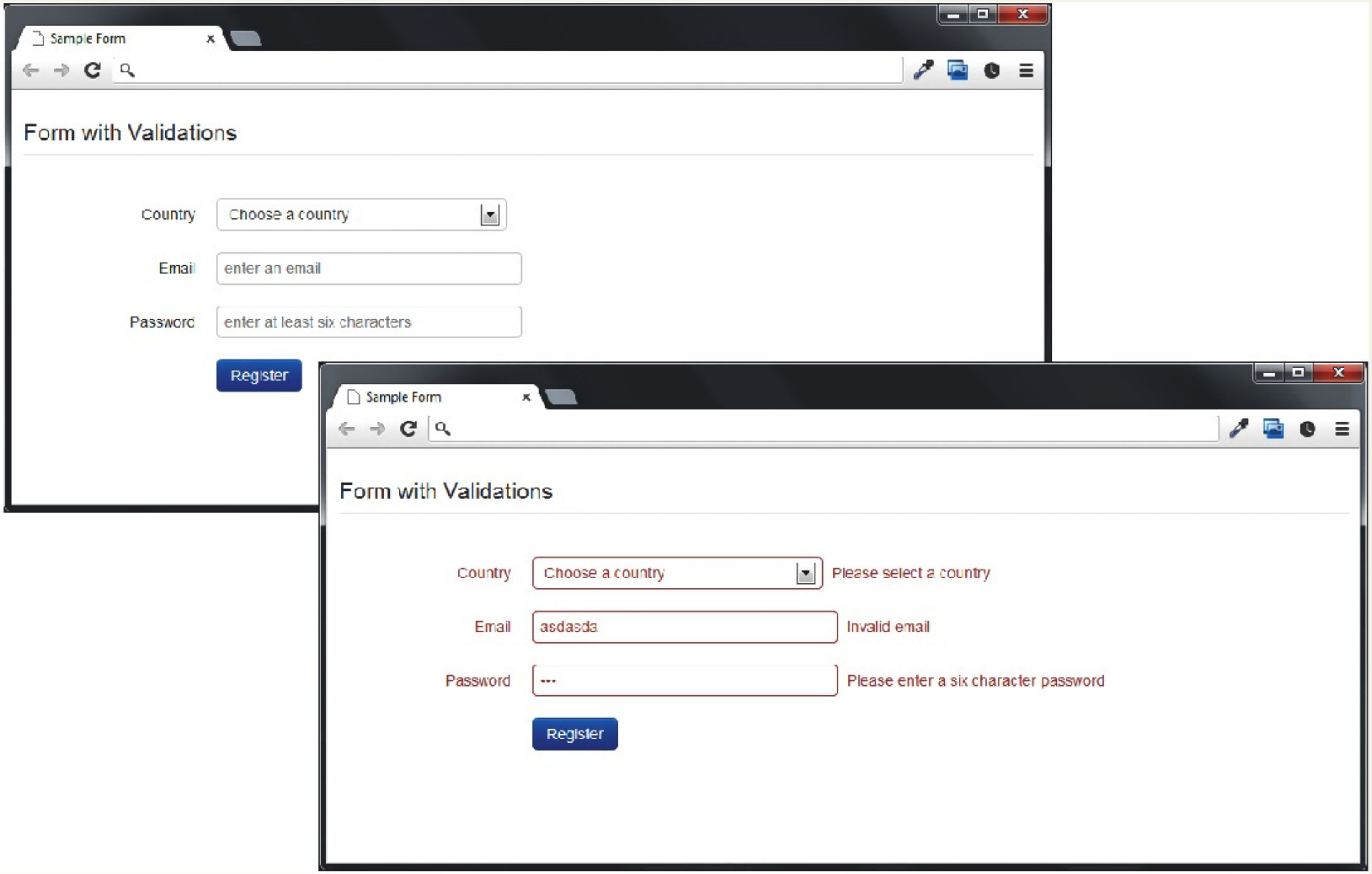
```
var passReg = /^[a-zA-Z]\w{8,16}$/;
if (! passReg.test(password.value)) {
    // provide some type of error message
}
```

What do we want to do when the JavaScript finds a validation error?

- Highlight errors by **adding CSS classes** to the input elements causing the error

# JavaScript validation

Client-Side



Sample Form

Form with Validations

| | |
|---|---|
| Country | Choose a country |
| Email | enter an email |
| Password | enter at least six characters |
| | Register |

Sample Form

Form with Validations

| | | |
|---|---|---|
| Country | Choose a country | Please select a country |
| Email | asdasda | Invalid email |
| Password | ••• | Please enter a six character password |
| | Register | |

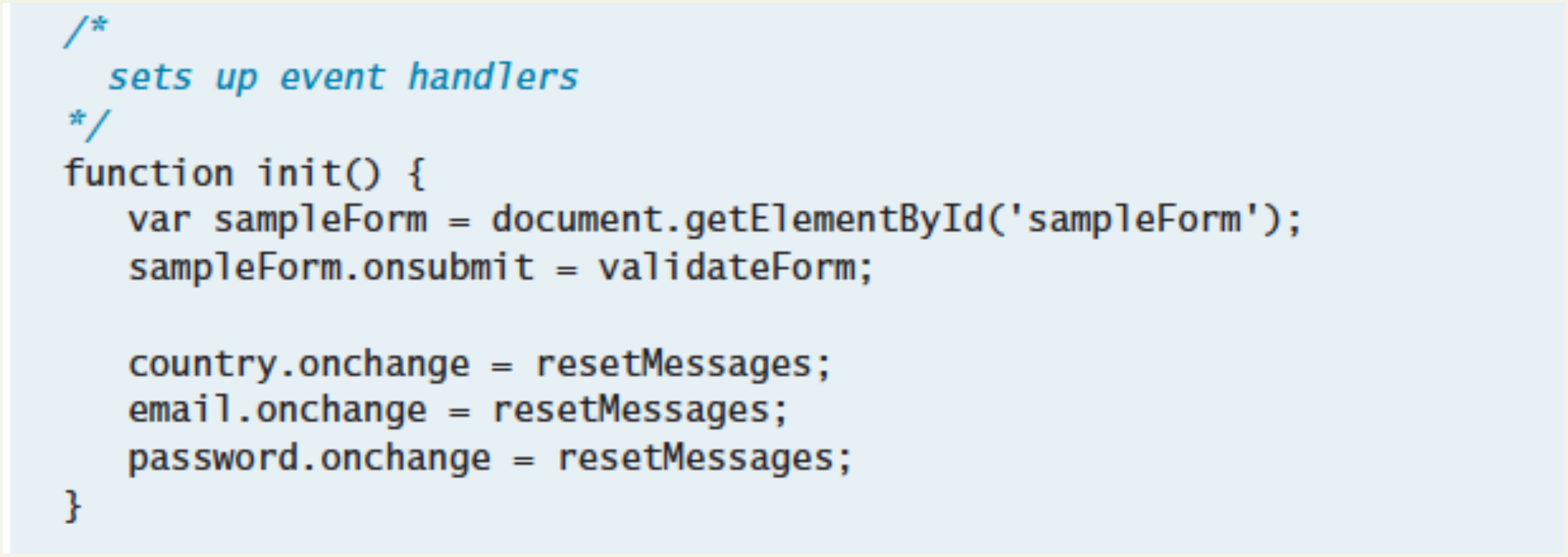
# JavaScript Code

Function to add an error message to a certain element (by id)

```
<script>
// we will reference these repeatedly
var country = document.getElementById('country');
var email = document.getElementById('email');
var password = document.getElementById('password');

/*
  Add passed message to the specified element
*/
function addErrorMessage(id, msg) {
   // get relevant span and div elements
   var spanId = 'error' + id;
   var span = document.getElementById(spanId);
   var divId = 'control' + id;
   var div = document.getElementById(divId);

   // add error message to error <span> element
   if (span) span.innerHTML = msg;
   // add error class to surrounding <div>
   if (div) div.className = div.className + " error";
}
```

# JavaScript Code

Set up the event handlers



```javascript
/*
  sets up event handlers
*/
function init() {
   var sampleForm = document.getElementById('sampleForm');
   sampleForm.onsubmit = validateForm;

   country.onchange = resetMessages;
   email.onchange = resetMessages;
   password.onchange = resetMessages;
}
```

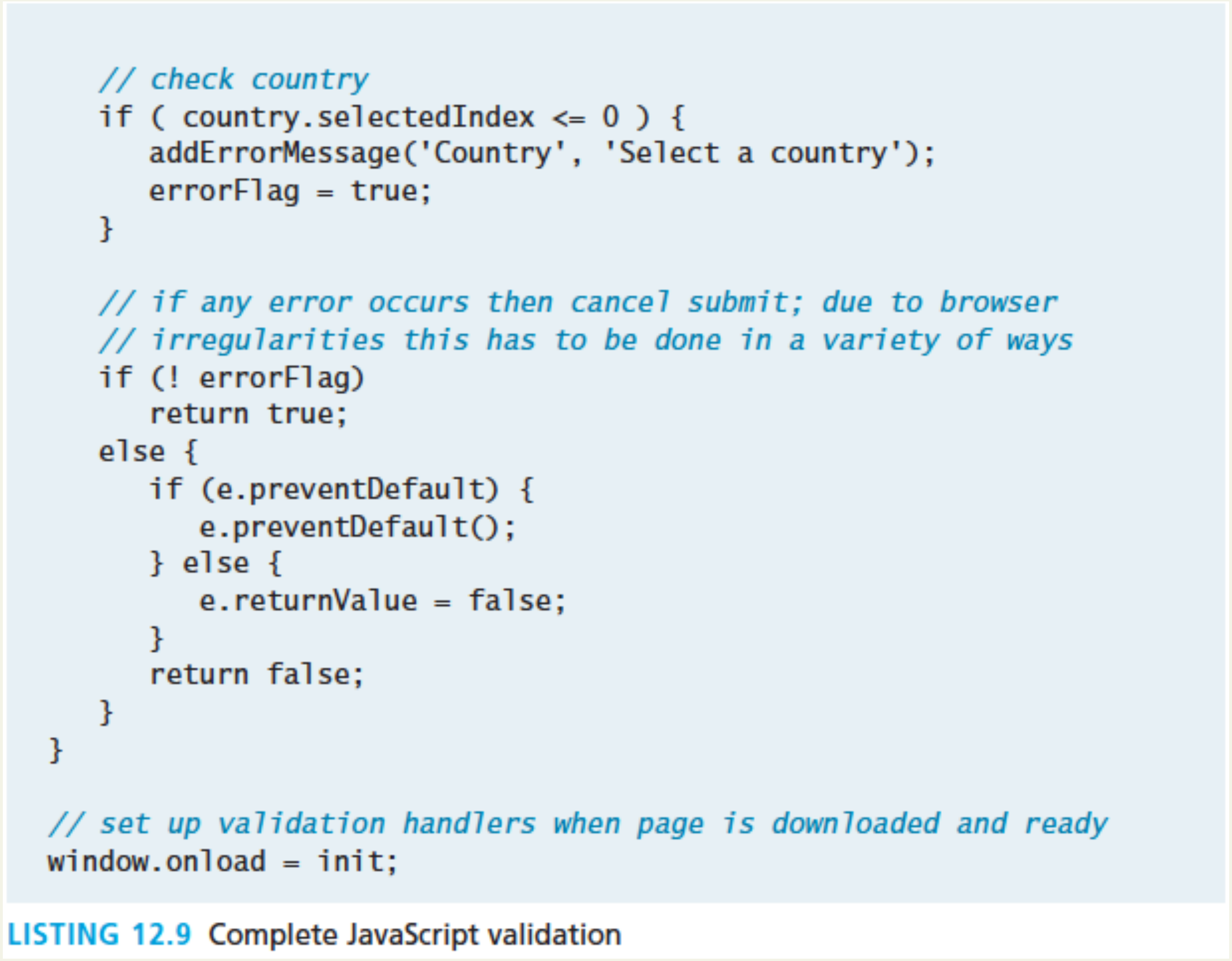
## JavaScript Code

The actual checks (part 1)

```
/*
  perform the validation checks
*/
function validateForm() {
   var errorFlag = false;

   // check email
   var emailReg = /(.+)@([^\.].*)\.([a-z]{2,})/;
   if (! emailReg.test(email.value)) {
      addErrorMessage('Email', 'Enter a valid email');
      errorFlag = true;
   }

   // check password
   var passReg = /^[a-zA-Z]\w{8,16}$/;
   if (! passReg.test(password.value)) {
      addErrorMessage('Password', 'Enter a password between 9-16
                       characters');
      errorFlag = true;
   }
```

# JavaScript Code

The actual checks (part 2)



```
    // check country
    if ( country.selectedIndex <= 0 ) {
       addErrorMessage('Country', 'Select a country');
       errorFlag = true;
    }

    // if any error occurs then cancel submit; due to browser
    // irregularities this has to be done in a variety of ways
    if (! errorFlag)
       return true;
    else {
       if (e.preventDefault) {
          e.preventDefault();
       } else {
          e.returnValue = false;
       }
       return false;
    }
 }

 // set up validation handlers when page is downloaded and ready
 window.onload = init;
```

LISTING 12.9 Complete JavaScript validation

## PHP Validation

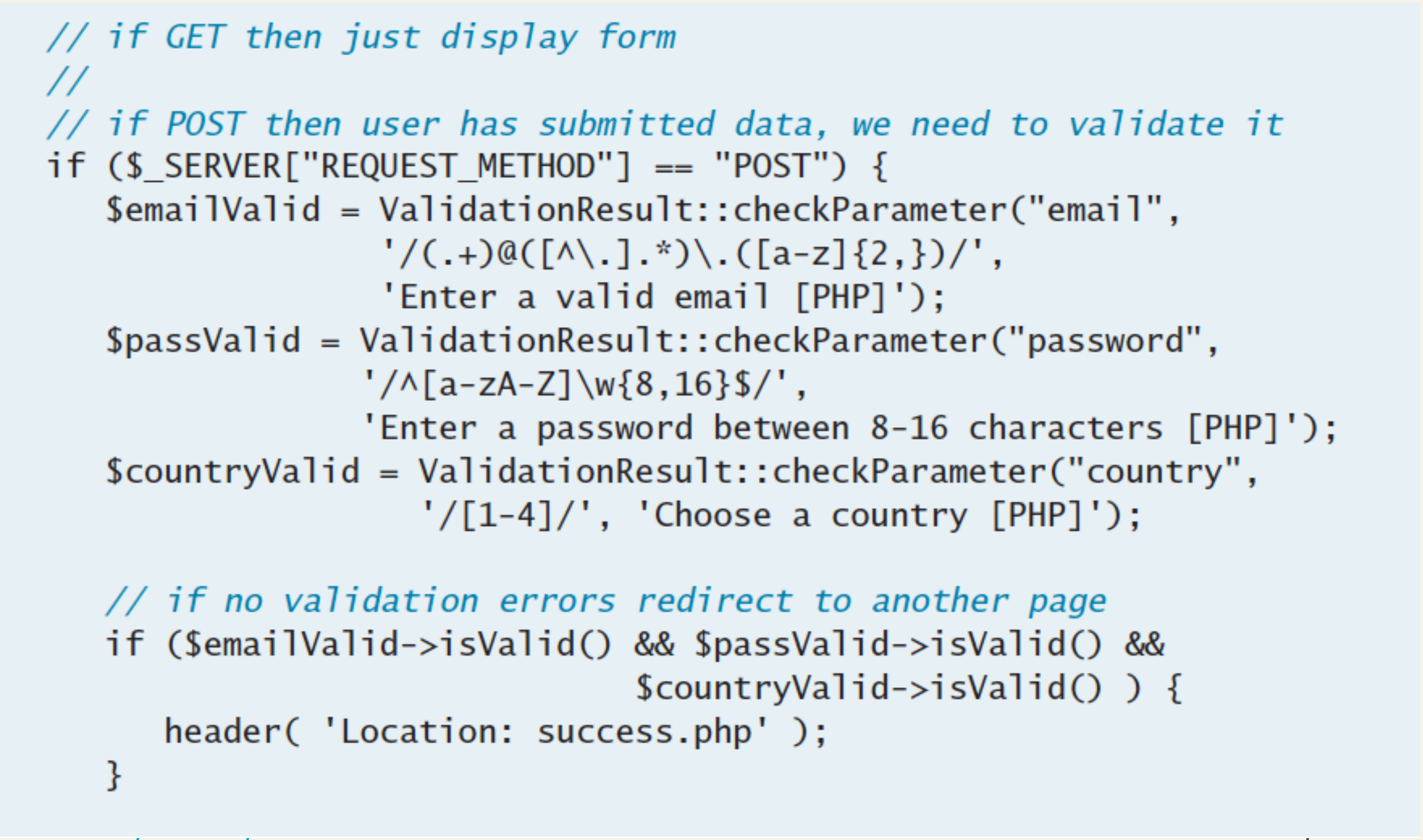
The only one you HAVE to do

No matter how good the HTML5 and JavaScript validation, client-side prevalidation can always be circumvented by hackers, or turned off by savvy users.

Validation on the server side using PHP is the most important form of validation and the only one that is absolutely essential.

# PHP Validation

An abridged example…



```
// if GET then just display form
//
// if POST then user has submitted data, we need to validate it
if ($_SERVER["REQUEST_METHOD"] == "POST") {
   $emailValid = ValidationResult::checkParameter("email",
                 '/(.+)@([^\.].*)\.([a-z]{2,})/',
                 'Enter a valid email [PHP]');
   $passValid = ValidationResult::checkParameter("password",
                '/^[a-zA-Z]\w{8,16}$/',
                'Enter a password between 8-16 characters [PHP]');
   $countryValid = ValidationResult::checkParameter("country",
                   '/[1-4]/', 'Choose a country [PHP]');

   // if no validation errors redirect to another page
   if ($emailValid->isValid() && $passValid->isValid() &&
                               $countryValid->isValid() ) {
      header( 'Location: success.php' );
   }
```

## A simple example

```
<!DOCTYPE html>
<head>
  <meta charset="utf-8">
  <title>Esempio validazione</title>
  <script src="./valid.js"></script>
  <link rel="stylesheet" href="./mystyle.css">
</head>
<body>
<form id="buy" action="order.php" method="get">
  <!-- First input is validated using HTML5 -->
  <label for="quante">How many eggs?</label>
  <input type="number" min="6" max="60"
   step="6" id="quante" name="ne" required>
  <br>
```

## A simple example

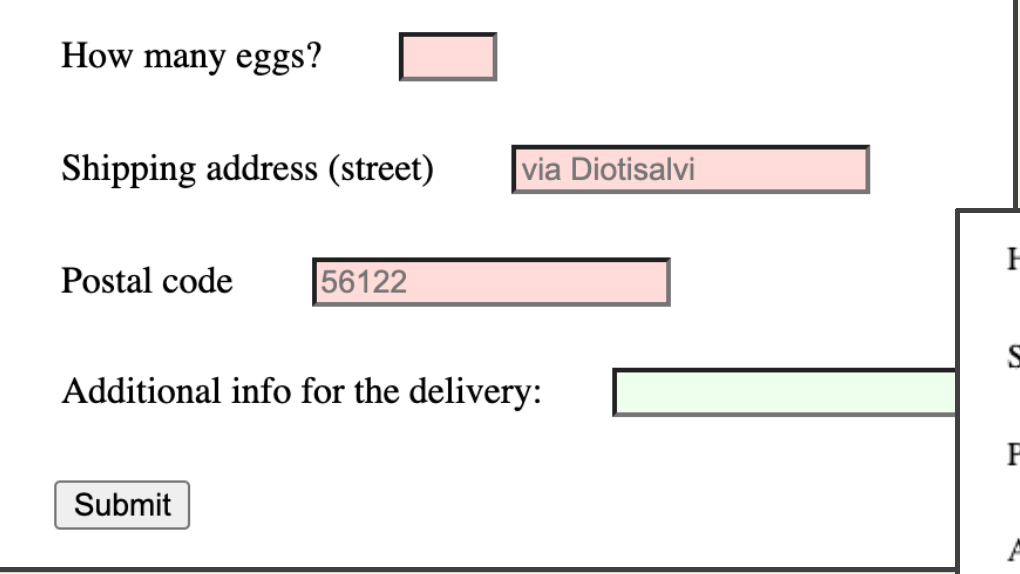
```
  <!-- Second input is validated using JS -->
  <label for="dove1">Shipping address (street)</label>
  <input type="text" id="dove1" name="sa"
         placeholder="via Diotisalvi" required>
  <br>
  <!-- Third input is validated using HTML5 -->
  <label for="dove2">Postal code</label>
  <input type="text" id="dove2" name="pc"
         pattern="^[0-9]{5}$" placeholder="56122" required>
  <br>
  <!-- The fourth field is not required and not validated -->
  <label for="note">Additional info for the delivery:</label>
  <input type="text" id="note"><br>
  <button type="submit">Submit</button>
</form>
</body>
```

# A simple example

```css
* {margin: 1em;}
/* :invalid is a pseudo class defined by HTML5
it is automatically applied when the constraints specified
by means of HTML5 attributes are not met
*/
input:invalid {background-color: rgba(255, 0, 0, 0.167);}
/* :valid and :required are two pseudoclasses, same as
before */
input:valid{background-color: rgba(152, 251, 152, 0.171);}
```

# A simple example

```
// valid.js
function validate(e) {
  // Only the second field is validate by means of JS code
  const sa = document.getElementById("dove1");
  const v = sa.value;
  const addr = ["via Diotisalvi", "via Mazzini",
                "Corso Italia"];
  if (!addr.includes(v)) {
    let msg =
      "We currently ship only to these addresses: " + addr;
    sa.setCustomValidity(msg);
  } else {
    sa.setCustomValidity("");
  }
}

function init() {
  const f = document.getElementById("dove1");
  f.addEventListener("input", validate);
}

window.onload = init;
```

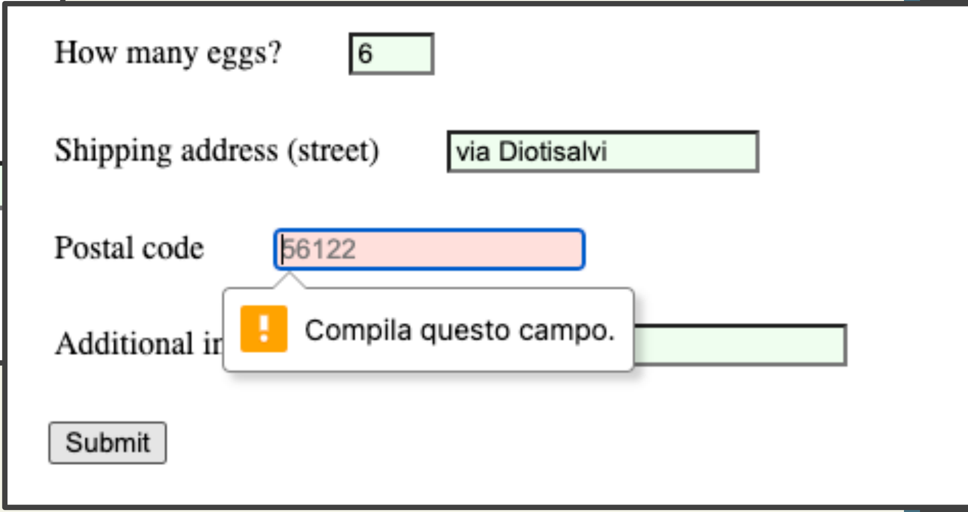

How many eggs?

Shipping address (street) via Diotisalvi

Postal code 56122

Additional info for the delivery:

Submit


How many eggs? 6

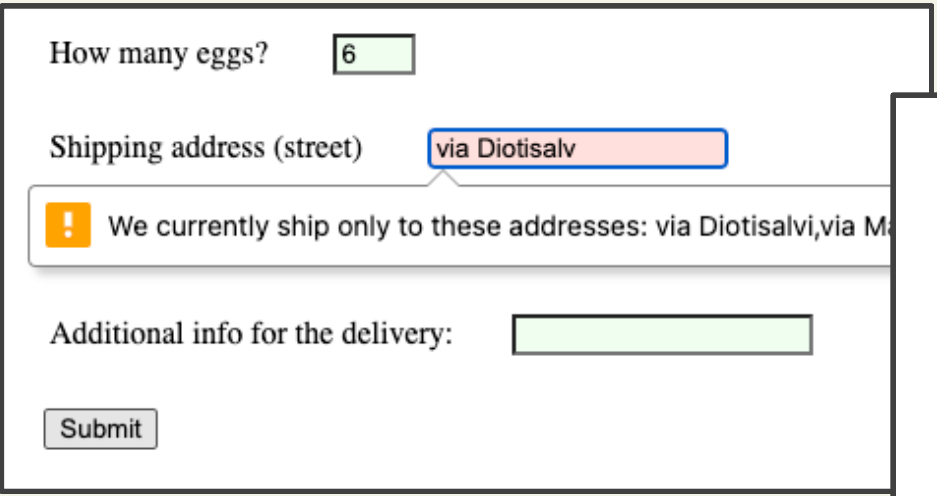
Shipping address (street) via Diotisalvi

Postal code 56122

Compila questo campo.

Additional ir

Submit

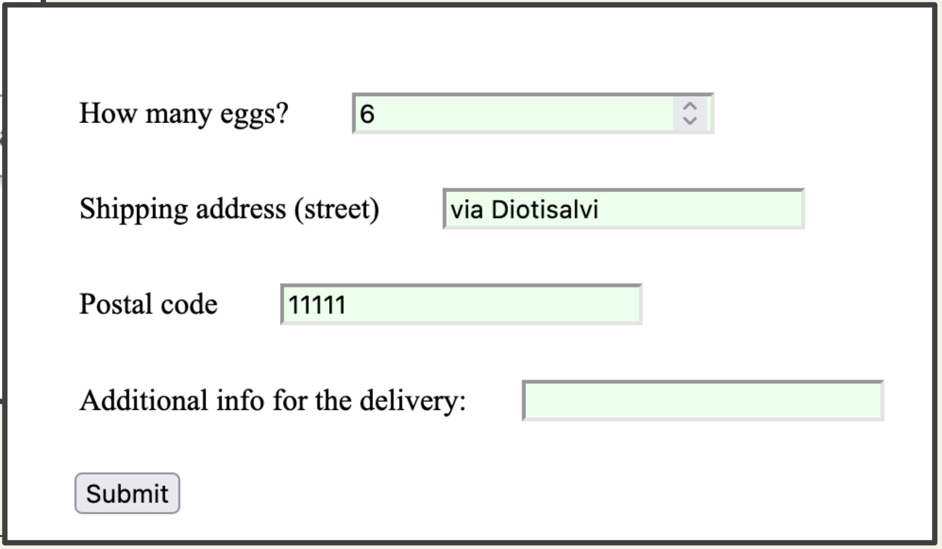

How many eggs? 6

Shipping address (street) via Diotisalv

We currently ship only to these addresses: via Diotisalvi,via M

Additional info for the delivery:

Submit


How many eggs? 6

Shipping address (street) via Diotisalvi

Postal code 11111

Additional info for the delivery:

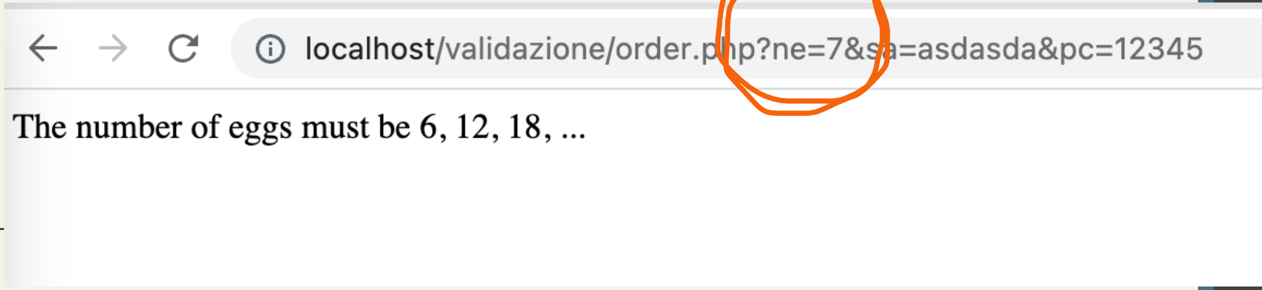
Submit

## A simple example

```php
<?php
if(empty($_GET["ne"]) ||
   empty($_GET["sa"]) ||
   empty($_GET["pc"])) {
  echo "One of the required values is missing";
  die();
}
$ne = $_GET["ne"];
$sa = $_GET["sa"];
$pc = $_GET["pc"];

if(!is_numeric($ne) || $ne % 6 != 0 ||
    $ne < 6 || $ne > 60) {
  echo "The number of eggs must be 6, 12, 18, ..., 60";
  die();
}
```

## A simple example

```
$addr = ["via Diotisalvi", "via Mazzini", "Corso Italia"];
if(!in_array($sa, $addr)) {
  echo "We currently ship only to these addresses: ";
  foreach($addr as $a)
    echo $a . " ";
  die();
}

if(!is_numeric($pc) || preg_match('/^[0-9]{5}$/', $pc)!=1) {
  echo "The postal code is not correct";
  die();
}

echo "Parameters are OK!";
?>
```



```
localhost/validazione/order.php?ne=7&sa=asdasda&pc=12345
```

```
The number of eggs must be 6, 12, 18, ...
```