Error Handling and Validation

Chapter 12 Sections 5 & 6

Types of Errors

- Expected errors

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

- Warnings

  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

Handling Errors

- Specific error messages and warnings are helpful during development
- They should never be displayed to end users because they can potentially provide security-risk information:

```
Warning: mysqli_connect(): (HY000/1045): Access
denied for user 'mjfstudent'@'localhost' (using
password: YES) in
/home/mferner/public_html/CIT410/db_tester1.php
on line 12
Access denied for user 'mjfstudent'@'localhost'
(using password: YES)
```

- Instead, end users should be provided with helpful but controlled feedback.

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."

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	

Add

Types of Input Validation

- **Required information**. Some data fields just cannot be left empty. For instance, the principal name of things or people is usually a required field. Other fields such as emails, phones, or passwords are typically required values.
- **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.

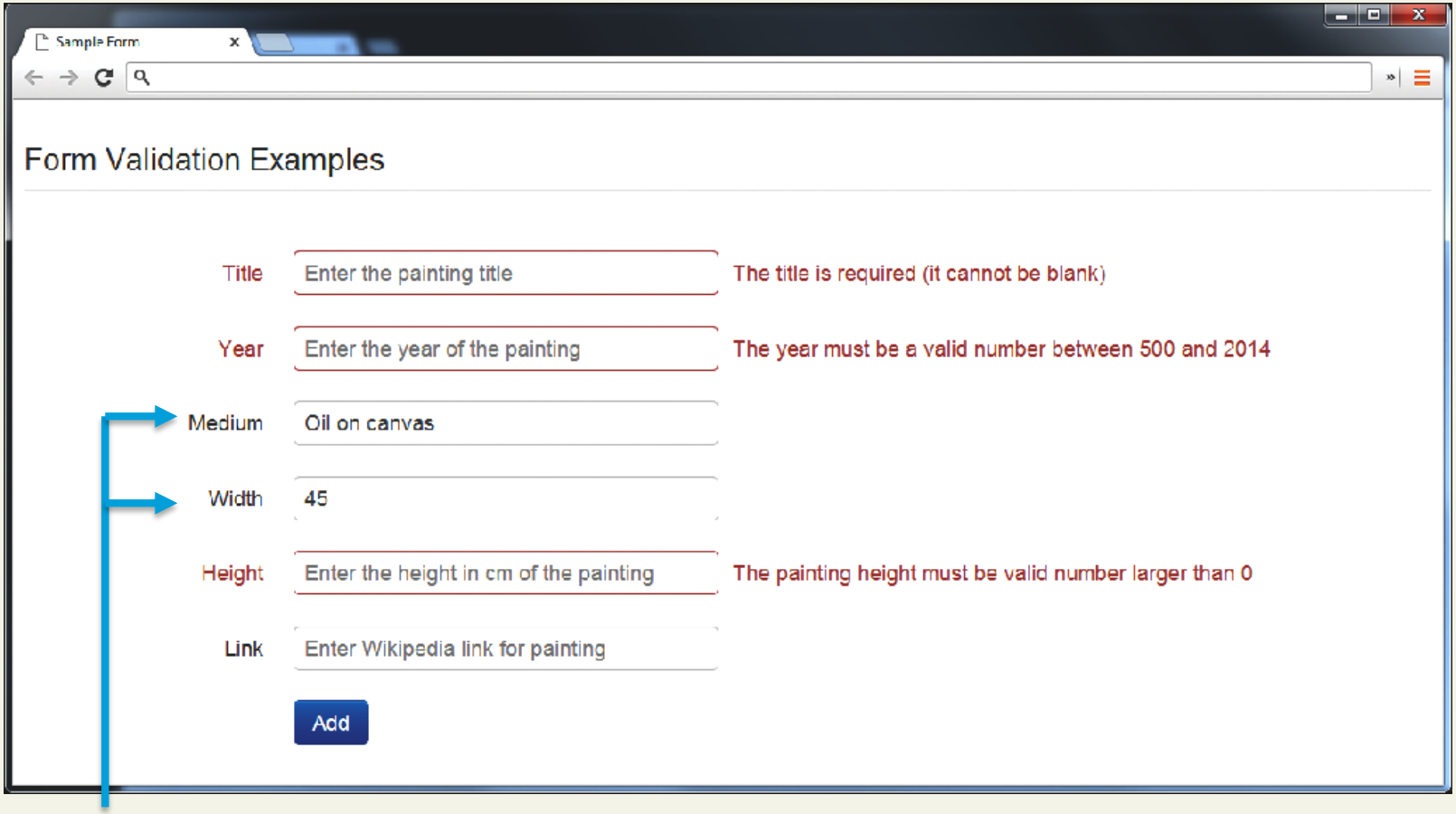
Types of Input Validation

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

Another example

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



Notice the correct input remains – this is called "sticky" input.

Reducing validation errors

- Use pop-up JavaScript alerts (or other popup) messages
- Provide textual hints to the user on the form itself
- Use tool tips to display context-sensitive help about the expected input
- Provide a JavaScript-based input mask when appropriate

Reducing validation errors

Static textual hints

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

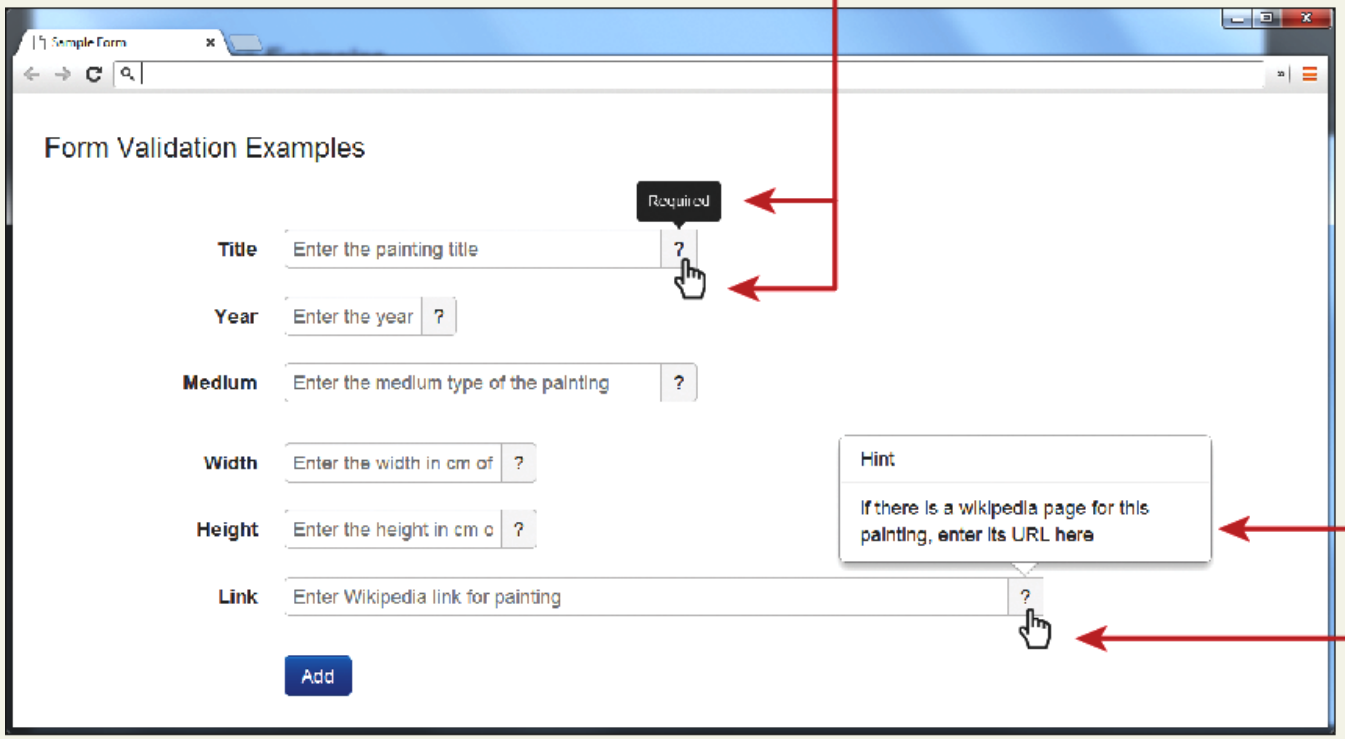
Placeholder text
(visible until user enters a value into field)

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

Reducing validation errors

Tool Tips and Pop-overs:

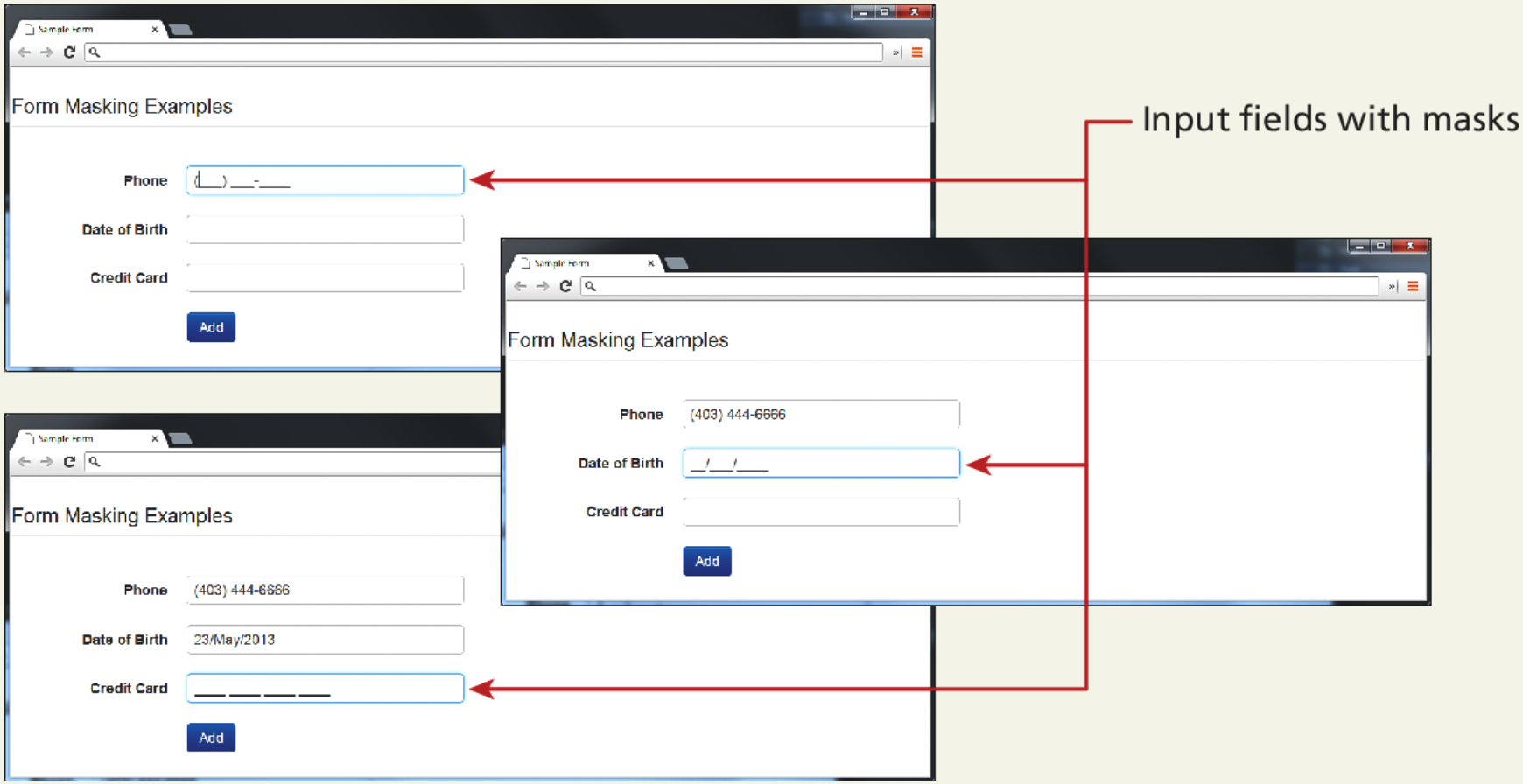
Pop-up tool tip
(appears when mouse
hovered over icon)



Pop-over

Reducing validation errors

JavaScript Mask



Reducing 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">
```

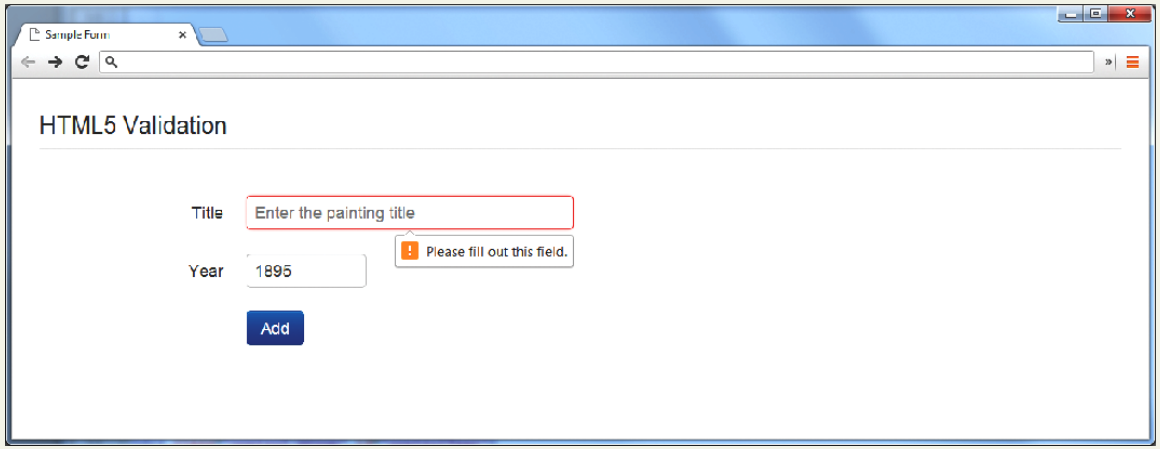
If you need a number, use the HTML5

```
<input type="number">
```

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*>

CAPTCHA

Completely **A**utomated **P**ublic **T**uring test to tell Computers and **H**umans **A**part

Automated form bots (often called **spam bots**) can flood a web application form with hundreds or thousands of bogus requests

This problem is generally solved by a test commonly referred to as a **CAPTCHA** which ask the user to enter a string of numbers and letters that are displayed in an obscured image that is difficult for a software bot to understand.

Where to Validate?

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

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

PHP Validation

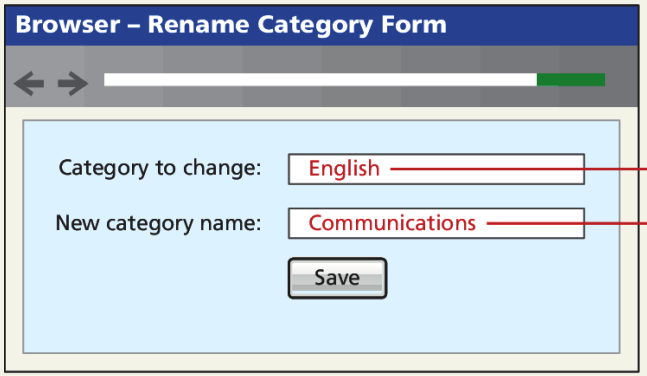
The only one you HAVE to do

No matter how good the HTML5 and JavaScript validation, client-side pre-validation 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.

Integrating User Data

Say, using an HTML form posted to the PHP script




```
<form method="post" action="rename.php">
  <input type="text" name="old" /><br/>
  <input type="text" name="new" /><br/>
  <input type="submit" />
</form>
```

$_POST['old']	$_POST['new']
English	Communications

```
UPDATE Categories SET CategoryName='English' WHERE CategoryName='Communications'
```

Integrating User Data

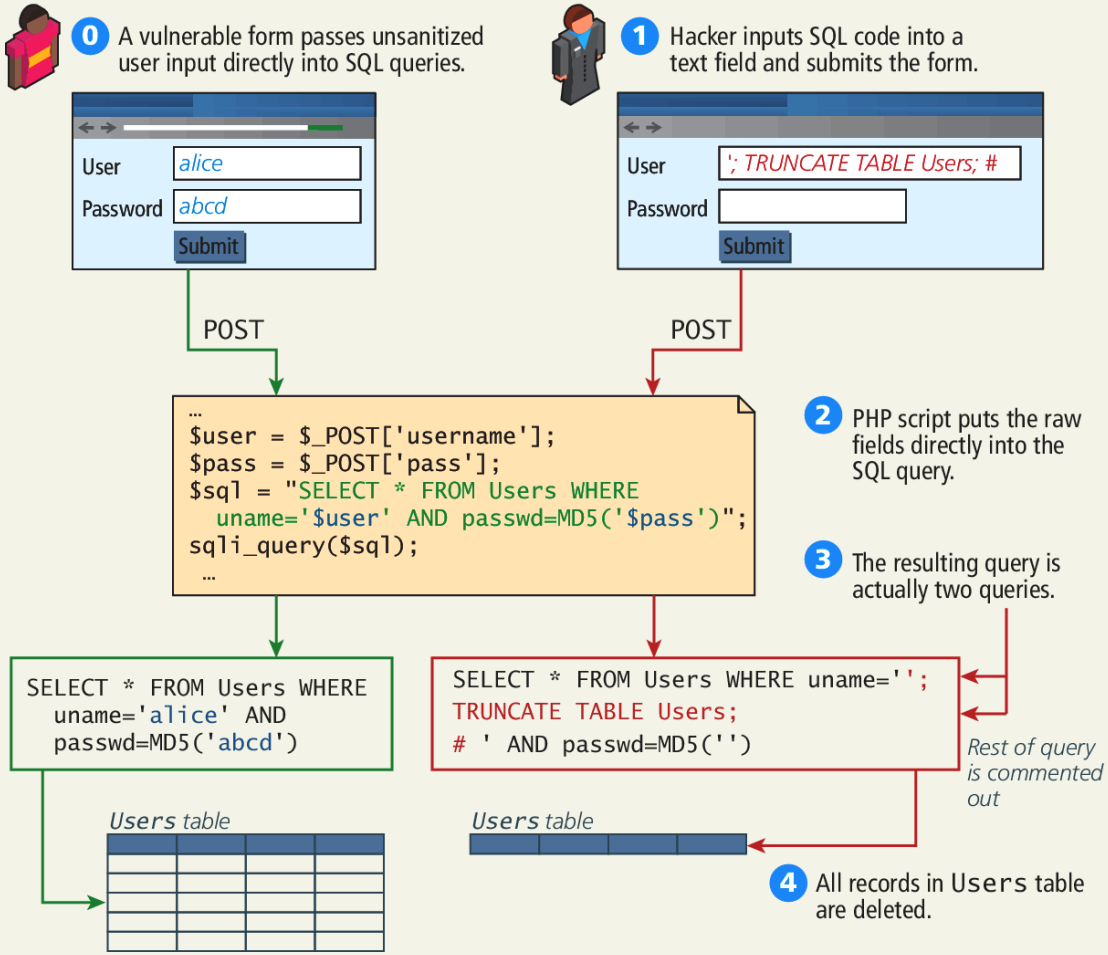
Not everyone is nice.

```
$from = $_POST['old'];
$to = $_POST['new'];
$sql = "UPDATE Categories SET CategoryName='$to' WHERE
        CategoryName='$from'";
```

While this does work, it opens our site to one of the most common web security attacks, the **SQL injection attack.**

SQL Injection Illustration

From Chapter 16



Defend against attack

Distrust user input

The SQL injection class of attack can be protected against by

- **Sanitizing** user input
- Using **Prepared Statements**

Form validation

- HTML5 will help with form validation:
  - Input types: email, date, tel, url, etc. See: http://www.w3schools.com/html/html5_form_input_types.asp
  - The *required* attribute. See: http://www.w3schools.com/html/html5_form_attributes.asp

- Useful functions:
- empty(*$var*)
- isset(*$var*)
- is_numeric (*$var)*

Form validation

- Useful functions:

Name	Description	Best use
empty(*$var*)	Returns TRUE if the variable has been set and is not NULL	Text input
isset(*$var*)	Returns TRUE if the variable hasn't been set, contains a NULL value, or contains an empty string.	Non-text input: radio buttons, check boxes, submit. etc.
is_numeric (*$var)*	Returns TRUE if the variable is a number or a string that can be converted to a number	

Form validation

- Useful functions for converting user-entered data for display:

Name	Description
htmlspecialchars(*$string*)	Converts certain HTML special characters (&, ', ", < and >) to their corresponding HTML entities. For example & becomes &
htmlentities(*$string*)	Converts all HTML characters that have corresponding HTML entities and returns the resulting string.

Validating data by type

Each data type that PHP supports has a corresponding function that checks if a variable is of that type.

- is_array()
- is_bool()
- is_float()
- is_int()
- is_null()
- is_numeric()
- is_resource()
- is_scalar()
- is_string()

The filter_input function

Name	Description
filter_input($type, $variable_name [, $filter)	Gets a value from a superglobal variable and optionally filters it. Returns the requested value on success, a FALSE value if the filter fails, or a NULL value if the requested value is not set.

The arguments: Name	Description
type	Specifies the superglobal variable to access. Common values include INPUT_GET, INPUT_POST, INPUT_COOKIE
variable_name	The name of the value to retrieve
filter	Optional. The constant for the filter to apply.

Common constants for filters

Name	Description
FILTER_VALIDATE_INT	Validates an integer value.
FILTER_VALIDATE_FLOAT	Validates a floating-point (double) value.
FILTER_VALIDATE_EMAIL	Validates an email address.
FILTER_VALIDATE_URL	Validates a URL.
FILTER_VALIDATE_BOOLEAN	Returns a TRUE value for "1", "true", "on", or "yes". Otherwise returns a FALSE value.
FILTER_SANITIZE_STRING	Removes tags and removes or encodes special characters from a string

Validation filters examples

```
$product_description = filter_input(INPUT_GET, 'product_description', FILTER_SANITIZE_STRING);
//Null if product_description has not been set in the $_GET array

$investment = filter_input(INPUT_POST, 'investment', FILTER_VALIDATE_FLOAT);
//Null if investment has not been set in the $_post array
//FALSE if 'investment' is not a valid float (double) value
```

- Considered a best practice to always use the filter_input functions when you use values from a superglobal array.

Prepared Statements

Better in general

A **prepared statement** is actually a way to improve performance for queries that need to be executed multiple times. When MySQL creates a prepared statement, it does something akin to a compiler in that it optimizes it so that it has superior performance for multiple requests. It also integrates sanitization into each user input automatically, thereby protecting us from SQL injection.

Prepared Statements

mysqli

```php
<?php
//Listing 11.17 Using a prepared statement (mysqli)
// retrieve parameter value from query string
$id = $_GET['id'];

// construct parameterized query
//notice the ? parameter - Don't put quotes around it!
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID=?";

// create a prepared statement
if ($statement = mysqli_prepare($connection, $sql)) {

  // Bind parameters s - string, b - blob, i - int
  mysqli_stmt_bind_param($statement, 'i', $id);

  // execute query
  mysqli_stmt_execute($statement);
  }
?>
```

Process Query Results

mysqli

```
<?php

//Listing 11.20 Looping through the result set (mysqli)
$sql = "select * from Categories order by CategoryName";

// run the query
if ($result = mysqli_query($connection, $sql)) {
  // fetch a record from result set into an associative array
  while($row = mysqli_fetch_assoc($result))
    {
      // the keys match the field names from the table
      echo $row['ID'] . " - " . $row['CategoryName'] ;
      echo "<br>";
    }
}
?>
```

Process Query Results

Mysqli – using prepared statements

```
<?php
//Listing 11.21 Looping through the result set (mysqli–using prepared statements)
$sql = "SELECT Title, CopyrightYear FROM Books WHERE ID=?";

if ($statement = mysqli_prepare($connection, $sql)) {
  mysqli_stmt_bind_param($statement, 'i', $id);
  mysqli_stmt_execute($statement);

  // bind result variables
  mysqli_stmt_bind_result($statement, $title, $year);
  // loop through the data
  while (mysqli_stmt_fetch($statement)) {
    echo $title . '-' . $year . '<br/>';
  }
}
?>
```

Things to Remember

1. Don't show system error messages to end users
2. Don't skip server-side validation – ever!