



## Module 4

# Creating Forms to Collect and Validate User Input

- Creating HTML5 Forms
- Validating User Input by Using HTML5 Attributes
- Validating User Input by Using JavaScript

# Lesson 1: Creating HTML5 Forms

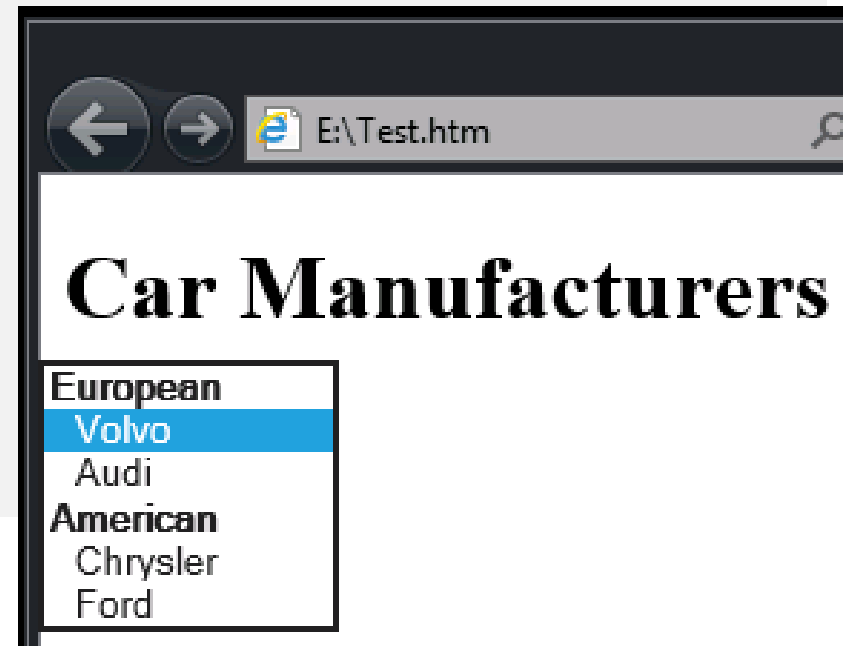
- Declaring a Form in HTML5
- HTML5 Input Types and Elements
- HTML5 Input Attributes

- Use an HTML5 form to gather user input:

```
<form name="userLogin" method="post" action="login.aspx">
  <fieldset>
    <legend>Enter your log in details:</legend>
    <div id="usernameField" class="field">
      <input id="uname" name="username" type="text"
        placeholder="First and Last Name" />
      <label for="uname">User's Name:</label>
    </div>
    <div id="passwordField" class="field">
      <input id="pwd" name="password" type="password"
        placeholder="Password" />
      <label for="pwd">User's Password:</label>
    </div>
  </fieldset>
  <input type="submit" value="Send" />
</form>
```

- HTML5 defines a wide range of new input types and elements, but not all are widely implemented

```
<select id="carManufacturer" name="carManufacturer">
  <optgroup label="European">
    <option value="volvo">Volvo</option>
    <option value="audi">Audi</option>
  </optgroup>
  <optgroup label="American">
    <option value="chrysler">
      Chrysler</option>
    <option value="ford">
      Ford</option>
  </optgroup>
</select>
```



- Input attributes modify the behavior of input types and forms to provide better feedback and usability:
  - autofocus
  - autocomplete
  - required
  - pattern
  - placeholder
  - many other input type-specific attributes

# Lesson 2: Validating User Input by Using HTML5 Attributes

- Principles of Validation
- Ensuring that Fields are Not Empty
- Validating Numeric Input
- Validating Text Input
- Styling Fields to Provide Feedback

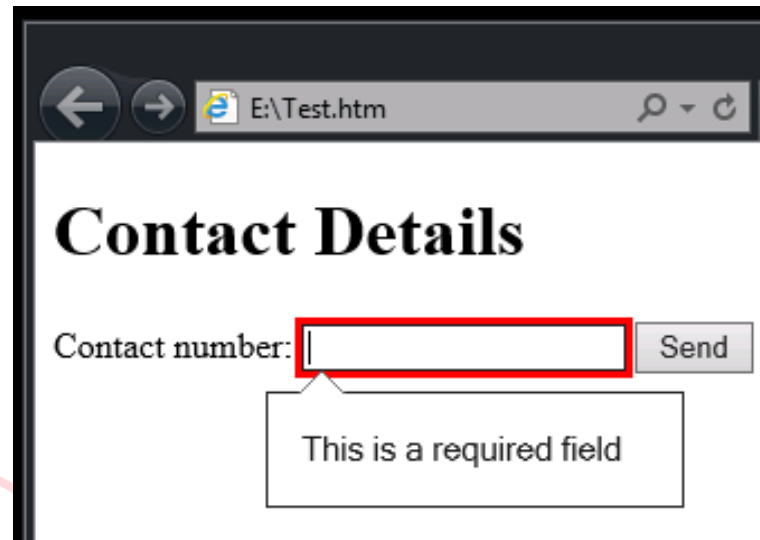
- User input can vary in accuracy, quality, and intent
- Client-side validation improves the user experience
- Server-side validation is still necessary



# Ensuring that Fields are Not Empty

- Use the **required** attribute to indicate mandatory fields
  - The browser checks that they are filled in before submitting the form

```
<input id="contactNo" name="contactNo" type="tel" placeholder="Enter your mobile number" required="required" />
```

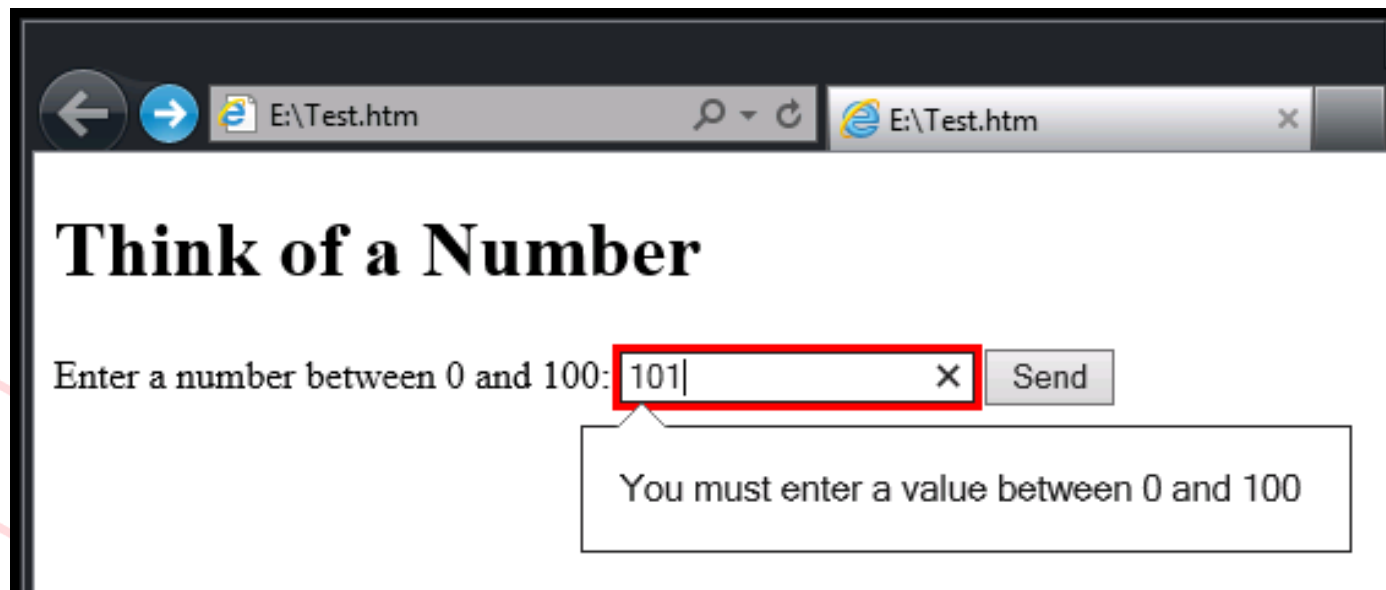


The screenshot shows a web browser window with the address bar displaying 'E:\Test.htm'. The page title is 'Contact Details'. Below the title, there is a form with the label 'Contact number:' followed by an empty text input field. The input field is highlighted with a red border. To the right of the input field is a 'Send' button. A tooltip message, 'This is a required field', is displayed below the input field, indicating that the field must be filled before the form can be submitted.

# Validating Numeric Input

- Use the **min** and **max** attributes to specify the upper and lower limit for numeric data

```
<input id="percentage" type="number" min="0" max="100" />
```

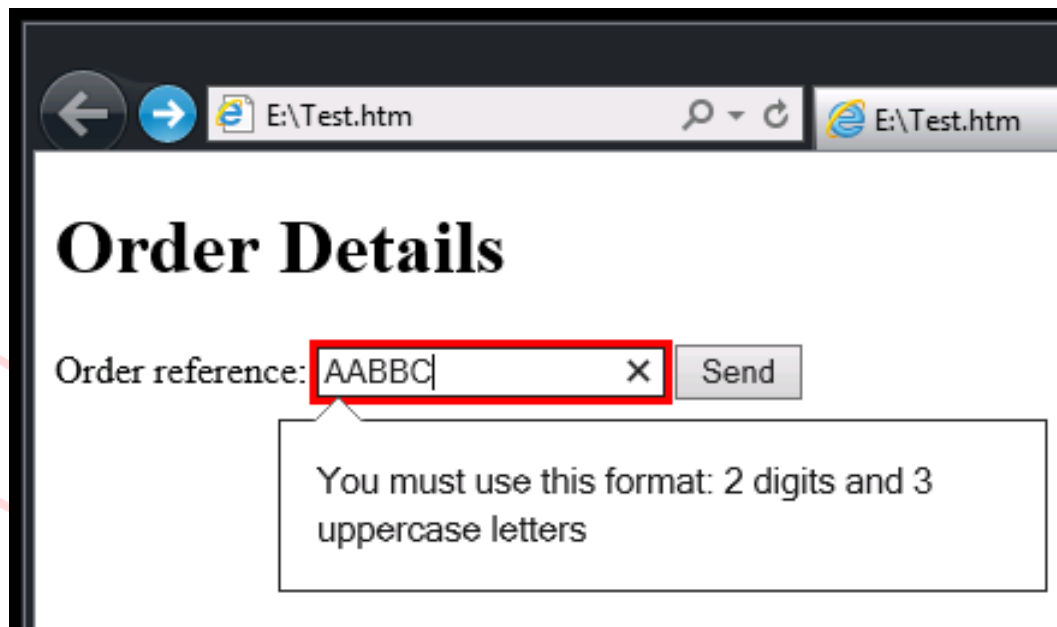


The screenshot shows a web browser window with the address bar displaying 'E:\Test.htm'. The page title is 'Think of a Number'. Below the title, there is a text input field with the placeholder text 'Enter a number between 0 and 100:'. The input field contains the value '101', which is highlighted with a red border. To the right of the input field is a 'Send' button. A validation error message is displayed in a box below the input field, stating 'You must enter a value between 0 and 100'.

# Validating Text Input

- Use the **pattern** attribute to validate text-based input by using a regular expression

```
<input id="orderRef" name="orderReference" type="text"  
  pattern="[0-9]{2}[A-Z]{3}" title="2 digits and 3 uppercase letters" />
```



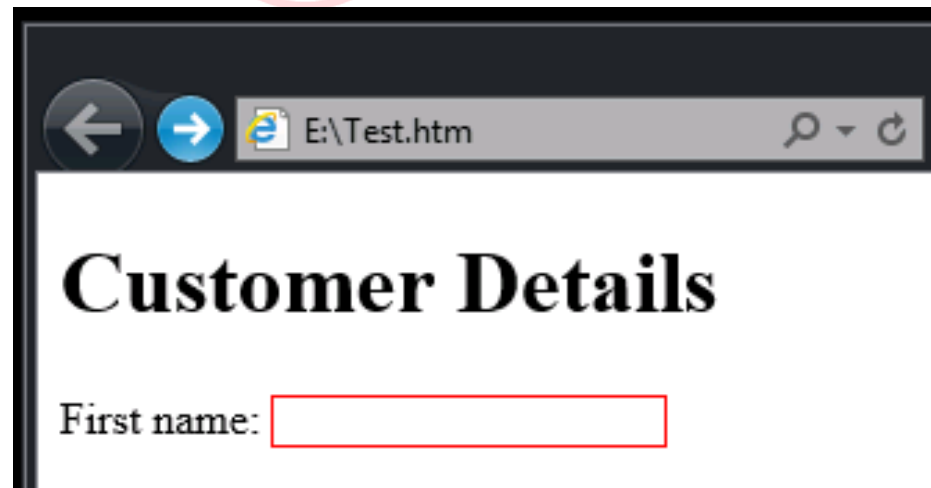
The screenshot shows a web browser window with the address bar displaying "E:\Test.htm". The page title is "Order Details". Below the title, there is a form with a label "Order reference:" followed by a text input field. The input field contains the text "AABBC" and has a red border around it. To the right of the input field is a "Send" button. A tooltip message is displayed below the input field, stating: "You must use this format: 2 digits and 3 uppercase letters".

# Styling Fields to Provide Feedback

Use CSS to style input fields

Use the **valid** and **invalid** pseudo-classes to detect fields that have passed or failed validation

```
input {  
  border: solid 1px;  
}  
input:invalid {  
  border-color: #f00;  
}  
input:valid {  
  border-color: #0f0;  
}
```



# Lesson 3: Validating User Input by Using JavaScript

- Handling Input Events
- Validating Input
- Ensuring that Fields are Not Empty
- Providing Feedback to the User
- Demonstration: Creating a Form and Validating User Input

- Catch the **submit** event to validate an entire form
  - Return true if the data is valid, false otherwise
  - The form is only submitted if the **submit** event handler returns true
- Catch the **input** event to validate individual fields on a character-by-character basis
  - If the data is not valid, display an error message by using the **setCustomValidity** function
  - If the data is valid, reset the error message to an empty string

- Use JavaScript code to emulate unsupported HTML5 input types and attributes in a browser:

```
<form id="scoreForm" ... onsubmit="return validateForm();" >
  <div id="scoreField" class="field" >
    <input id="score" name="score" type="number" />
  </div>
</form>
```

```
function isAnInteger( text ){
  var intTestRegex = /^s*(\+|-)?\d+s*$/;
  return String(text).search(intTestRegex) != -1;
}

function validateForm()
{
  if( ! isAnInteger(document.getElementById('score').value))
    return false;  /* No, it's not a number! Form validation fails */

  return true;
}
```

# Ensuring that Fields are Not Empty

Use JavaScript code to ensure that a required field does not contain only whitespace:

```
<form id="scoreForm" ... onsubmit="return validateForm();" >
  <div id="penaltiesField" class="field" >
    <input id="penalties" name="penalties" type="text" />
  </div>
</form>
```

```
function isSignificant( text ){
  var notWhitespaceTestRegex = /^[^\s]{1,}/;
  return String(text).search(notWhitespaceTestRegex) != -1;
}

function validateForm() {
  if( ! isSignificant(document.getElementById('penalties').value))
    return false;  /* No! Form validation fails */

  return true;
}
```



- Provide visual feedback to the user by defining styles and dynamically setting the class of an element:

```
.validatedFine {  
  border-color: #0f0;  
}  
.validationError {  
  border-color: #f00;  
}
```

```
function validateForm() {  
  var textbox = document.getElementById("penalties");  
  
  if( ! isSignificant(textBox.value)) {  
    textbox.className = "validationError";  
    return false;  /* No! Form validation fails */  
  }  
  textbox.className = "validatedFine";  
  return true;  
}
```

# Demonstration: Creating a Form and Validating User Input

In this demonstration, you will learn about the tasks that you will perform in the lab for this module.

# Lab: Creating a Form and Validating User Input

- Exercise 1: Creating a Form and Validating User Input by Using HTML5 Attributes
- Exercise 2: Validating User Input by Using JavaScript

Estimated Time: 60 minutes

Delegates who want to attend ContosoConf will be required to register and provide their details. You have been asked to add a page to the ContosoConf website that implements an attendee registration form.

The server-side code already exists to process the attendee data. However, the registration page performs very minimal validation that is not user friendly. You have decided to add client-side validation to the form to improve the accuracy of the registration data entered by attendees and to provide a better user experience.

# Module Review and Takeaways

- Review Question(s)