

## *Chapter 5*

# Creating Dynamic Web Pages

Today, most of the Web pages, such as the registration and login pages, have some way of accepting input from a user. To design such Web pages, you need to use several elements, known as form elements, which enable the user to input values. The data entered using these form elements needs to be processed further upon submission. Every Web page should respond to end user actions, such as clicking a submit button, changing a value in a field, or making a selection from a list. All these actions are referred to as events and need to be handled appropriately. This dynamic functionality can be implemented using the JavaScript object model.

This chapter discusses how to design an HTML form for accepting the user input. In addition, it discusses the various types of browser and form objects.

## Objectives

In this chapter, you will learn to:

- Design an HTML form
- Manipulate the components of a Web page

## Designing an HTML Form

Consider the scenario of FoodExpress.com. This website facilitates customers to place online orders with various restaurants. For this, the customers need to provide various details, such as the name, contact number, and address, in an online form named as **OrderFoodOnline**, as shown in the following figure.

## OrderFoodOnline

Name:

Date:

email ID:

Contact Number:

Select Food: ☐ Non-vegetarian ☐ Vegetarian

Select Restaurant:

Drinks:

Soups:

Dishes:

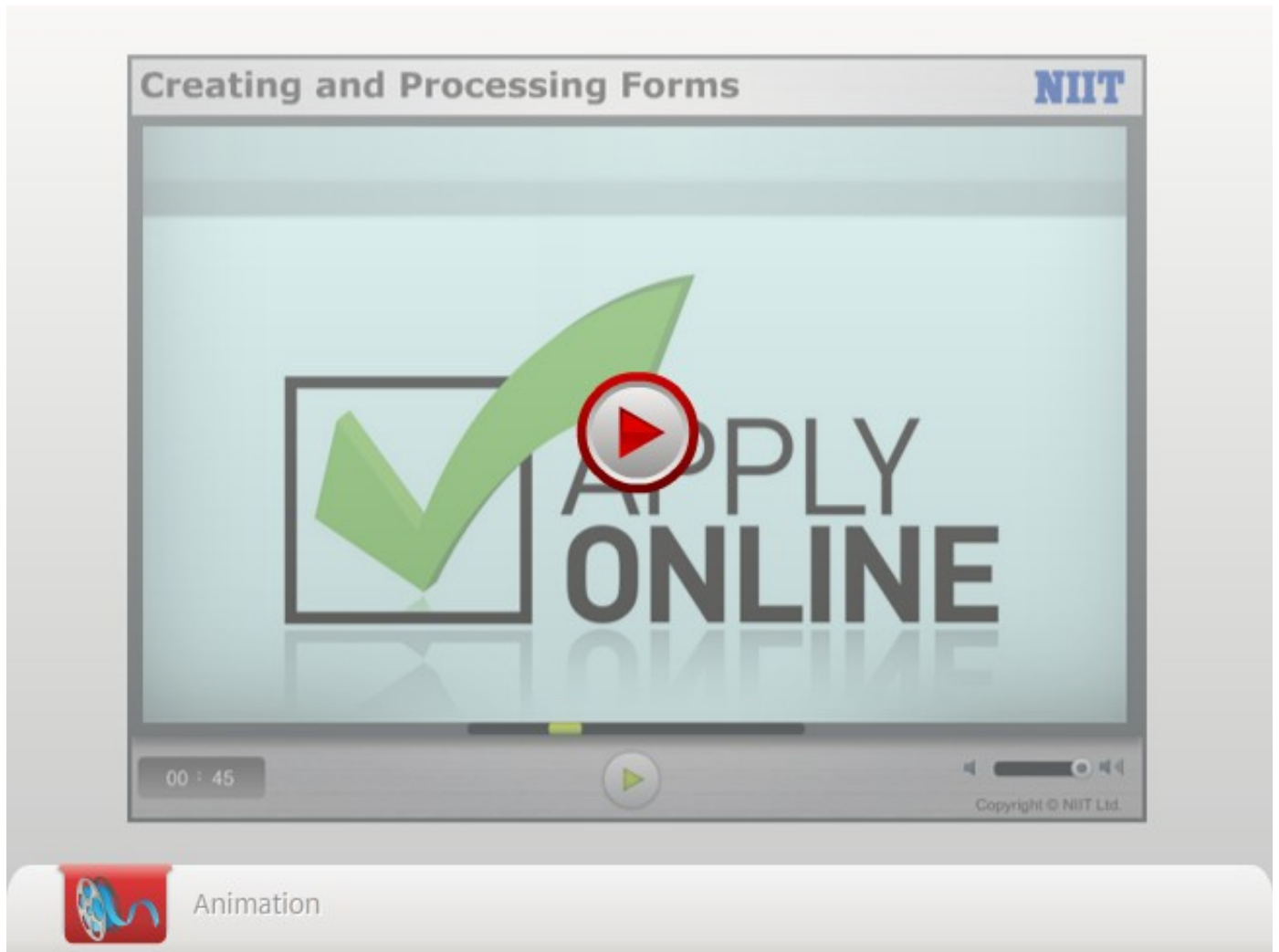
Order For: ☐ Take Away ☐ Home Delivery

Address:

State:

*The OrderFoodOnline Form*

Forms are interactive Web pages that are used to accept the user input. These forms consist of one or more types of input fields, such as text fields, check boxes, radio buttons, and the submit buttons. These input fields enable the user to fill and submit the information to a website. HTML supports various tags to create forms and input fields.



## Creating Forms

To create a form on a Web page, you need to use the `<FORM>` tag. The `<FORM>` tag helps you define a form. It has an opening `<FORM>` tag and a closing `</FORM>` tag. The following syntax is used to specify the `<FORM>` tag:

```
<FORM [attribute list]>
..
..
</FORM>
```

The `<FORM>` tag supports the following attributes:

- name
- ID
- action
- method
- autocomplete
- novalidate
- target

## name

The `name` attribute is used to specify a unique name for a form. It is used to uniquely identify a form in the `get` or `post` methods at the time of form submission. In addition, multiple forms can be present on a Web page. These forms can be differentiated using the `name` attribute. The following syntax is used to specify a name for a form:

```
<FORM name= "User defined name">... </FORM>
```

## ID

The `ID` attribute is used to specify a unique ID for the form element on a Web page. The `ID` attribute should be unique in the entire HTML document. Both the `ID` and the `name` attributes are used to uniquely identify a form element on a Web page. However, the ID given to a form element is used when you need to reference it with style sheets or scripts. You can assign a unique ID to a form element by using this attribute.

The following syntax is used to specify an ID for a form:

```
<FORM ID= "User defined ID">... </FORM>
```

## action

The `action` attribute specifies the URL of the page to which the contents of the form are submitted. If this attribute is missing, the URL of the document itself is assumed as the location for the form submission. The following syntax is used for the `action` attribute to specify the URL of the page:

```
<FORM action="filename or URL">
```

## method

The `method` attribute is used to specify the format in which the data will be submitted to the file or the URL specified in the `action` attribute. It can take either of the following values:

- `get`
- `post`

The default value of the `method` attribute is `get`.

### get

The `get` value appends the form data to the URL of the form as the name value pair at the time of form submission. Since the data is appended to the URL, it is always visible to the users. Therefore, it is not a correct approach when you need to submit the sensitive data. Also, the size of data that can be submitted using the `get` method is limited to only 3000 characters.

### post

The `post` value does not append the form data to the URL of the form when it is submitted. Therefore, the data is not shown in the URL and offers a secure way of submitting the data. Also, a large amount of data can be sent using the `post` method. This method can be used to send textual as well as image data.

## autocomplete

The `autocomplete` attribute is used to specify whether a form should have the autocomplete feature on or off. If it is on, the browser automatically completes the values in the fields based on the values that the user has entered before. The following code snippet is used to specify the `autocomplete` attribute for a form:

```
<FORM ID="fileID" autocomplete= "on">
```

## novalidate

The `novalidate` attribute specifies that the data in the form should not be validated by the browser at the time of data submission. It is an empty attribute that does not contain any value. The following code snippet is used to specify the `novalidate` attribute for a form:

```
<FORM ID="fileID" novalidate>
```

## target

The `target` attribute is used to specify the name of the frame or the window in which the response obtained after submitting the form needs to be displayed. The following syntax is used to specify the `target` attribute for a form:

```
<FORM target="_blank|_self|_parent|_top|frame_name">
```

The `target` attribute can have any one of the following values:

- `_blank`: Specifies that the response should be displayed in a new frame or window.
- `_self`: Specifies that the response should be displayed in the same frame.
- `_parent`: Specifies that the response should be displayed in the parent frame or window.
- `_top`: Specifies that the response should be displayed in the full body of the window.
- `frame_name`: Specifies that the response should be displayed in the specified frame.

In the BookYourHotel scenario, the following code is used to create the **OrderFoodOnline** form:

```
<FORM name= "OrderFoodOnline" ID= "Order_Food" method= "post">  
.....  
</FORM>
```

## Exploring Form Elements

You need to design the **OrderFoodOnline** form further so that it can be used to accept the user input.

For this, you need to add various input fields on the form. These fields can be added to a form by using the following tags:

- `<INPUT>`
- `<SELECT>`
- `<LABEL>`
- `<FIELDSET>`
- `<TEXTAREA>`
- `<DATALIST>`
- `<KEYGEN>`
- `<OUTPUT>`
- `<BUTTON>`

## <INPUT>

The `<INPUT>` tag is used to create input fields inside a form. These fields are used to accept input from the users. Input fields are of various types, such as text box, radio button, or check box. The type of input field is determined by the value of its `type` attribute. The `<INPUT>` tag has some important attributes, such as `type`, `value`, `name`, `ID`, `autocomplete`, `autofocus`, `form`, `required`, `pattern`, and `placeholder`.

### Defining the type Attribute

The `type` attribute of the `<INPUT>` tag defines the type of input field to be added on the form. The `type` attribute has the following values:

- `text`: Creates a single line editable text field. When the value of the `type` attribute is `text`, two additional attributes, `size` and `maxlength`, can also be specified. The `size` attribute is used for limiting the character width of the text field in the form. The `maxlength` attribute defines the maximum number of characters that can be typed in the text field. The following code snippet is used to create a text field:  
  
First Name: `<INPUT type="text" name="fname" size="20" maxlength="20">`  
Last Name: `<INPUT type="text" name="lname" size="20" maxlength="20">`  
  
In the preceding code snippet, two text fields, `fname` and `lname`, are created. Both the fields are 20 characters wide and can accept a maximum of 20 characters.
- `password`: Creates a password field, which will not display the characters being typed by the user. It hides the actual characters and shows the masked values for each character, such as `****`. The following code snippet is used to create a password field:  
  
`<INPUT type="password" name="accountpasswr">`
- `radio`: Creates a radio button, which lets the user select one of the options from a set of given options. When the value of the `type` attribute is `radio`, an additional attribute, `checked`, can also be specified. The `checked` attribute is used to specify that the radio button appears pre-selected when the page loads. The following code snippet is used to create a radio field:  
  
`<INPUT type="radio" name="Rating of Hotel" checked>`

```
5-Star<BR/>
```

```
<INPUT type="radio" name="Rating of Hotel"> Budgeted
```



*The checked attribute can also be used with the checkbox input field.*

In the preceding code snippet, two radio buttons `5-Star` and `Budgeted`, are created. The `checked` attribute is applied on the first radio button, `5-Star`. Therefore, it appears selected by default.

The radio buttons in a group are given the same name to ensure that the user is able to select only one radio button at a time.

The radio buttons are displayed as shown in the following figure.

*The Radio Buttons*

- checkbox: Creates a check box, which lets the user select one or more options from a set of given options. The following code snippet is used to create the check box fields:

```
<INPUT type="checkbox" name="Cuisine1" value="Continental Cuisine">
Continental Cuisine
```

```
<INPUT type="checkbox" name="Cuisine2" value="Chinese Cuisine">
Chinese Cuisine
```

In the preceding code snippet, two check boxes, `Continental Cuisine` and `Chinese Cuisine`, are created, as shown in the following figure.

*The Check Boxes*



*The value attribute will be discussed later in this chapter.*

- submit: Creates a submit button, which submits the form data to the location specified in the `action` attribute of the form. When the value of the `type` attribute is `submit`, some additional attributes, such as `formaction`, `formmethod`, `formtarget`, and `formnovalidate` can also be specified. The description of these attributes is given in the following list:
  - `formaction`: It is used to specify a URL where the form data would be submitted when the submit button is clicked. The URL specified in the `formaction` attribute of the submit button overrides the URL specified in the `action` attribute of the `<FORM>` tag. Therefore, the form is forcibly submitted to the URL specified in the `formaction` attribute of the submit button, instead of the URL specified in the `action` attribute of the form.
  - `formmethod`: It is used to specify the method, such as `get` and `post`, using which the form data will be sent to the file or URL specified in the `action` attribute of the

form. The value specified for the `formmethod` attribute of the submit button overrides the value of the `method` attribute of the `<FORM>` tag.

- `formtarget`: It is used to specify the name of the frame or the window in which the response would be displayed when the form is submitted. The value specified for the `formtarget` attribute of the submit button overrides the value of the `target` attribute of the `<FORM>` tag.
- `formnovalidate`: Every form is validated by default, unless you use the `novalidate` attribute with the `<FORM>` tag. The `formnovalidate` attribute of the submit button is used to force the form to behave like a form with the `novalidate` attribute.



*The `formaction`, `formmethod`, `formtarget`, and `formnovalidate` attributes can also be used with the `INPUT` type, image and the `<BUTTON>` tag.*

*The `INPUT` type, image and the `<BUTTON>` tag will be discussed later in this chapter.*

- `reset`: Creates a reset button, which clears the values entered by a user in the form fields. The following syntax is used to create a reset field:

```
<INPUT type="reset" name="reset">
```

- `url`: Adds a field that is used to enter the URL of a website. The value in this field is automatically validated for correctness when the form is submitted. The following syntax is used to create a URL field:

```
<INPUT type="url" name="locator">
```



*URL stands for Uniform Resource Locator. It is the address of a unique file or resource available on the web that is accessible through the Internet.*

*`www.w3schools.com` is an example of a URL specified in the address bar of a browser to connect to the home page of `w3schools.com` website.*

- `email`: Creates a field in an HTML form to accept the email address from the users. You can even use a normal text field to accept the email address. But every email address needs to be validated for its correct format, `<username>@<domainname>`. If you use the text field to accept an email address, such a validation is difficult to perform. However, with an email field, the email address is automatically validated upon submitting the form.

When the value of the `type` attribute is `email`, an additional attribute, `multiple`, can also be specified. The `multiple` attribute is used to specify that the user can enter more than one value in the field. The following code snippet is used to create an email field:

```
<INPUT type="email" name="email_id" multiple>
```

- `range`: Creates a slider control to enter a numeric value within a range. The default range of the slider is 0 to 100. When the value of the `type` attribute is `range`, additional attributes, such as `min`, `max`, `value`, and `step`, can also be specified. The `min` attribute is used to specify the minimum value in a range. The `max` attribute is used to specify the maximum value



in a range. The `value` attribute stores the current value associated with the range field. The `step` attribute is used to specify a number with which the `value` of the range field will increase or decrease as you move the slider. The following code snippet is used to create a range field:

```
<INPUT type="range" max="50" min="10" step="5" value="10">
```

The preceding code snippet creates a range with the default value 10, the minimum value, 10, and the maximum value, 50. The value will increase by 5 within the minimum and maximum limit when you use the slider to specify the numeric value. The range field is displayed, as shown in the following figure.



*The Range Field*

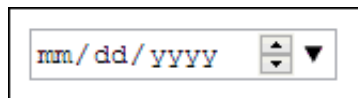


*The `min`, `max`, and `step` attributes can also be used with the input types, such as `number`, `date`, and `time`. The input types, such as `number`, `date`, and `time`, will be discussed later in this chapter.*

- `date`: Is used to define a date field in an HTML form. It allows a user to select a date. The following code snippet is used to create a date field:

```
<INPUT type="date" name="bday">
```

The preceding code snippet creates a date field, as shown in the following figure.

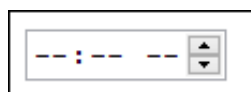


*The Date Field*

- `time`: Is used to define a time field in an HTML form. It allows a user to select time. The following code snippet is used to create a time field:

```
<INPUT type="time" name="usr_time">
```

The preceding code snippet creates a time field, as shown in the following figure.

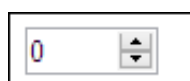


*The Time Field*

- `number`: Is used to create an input field for entering a numeric value. The following code snippet is used to create a number field:

```
<INPUT type="number" name="quantity" min="0" max="50">
```

The preceding code snippet creates a number field with a minimum value of 0 and a maximum value of 50. The number field is displayed, as shown in the following figure.



*The Number Field*

- `tel`: Is used to accept a telephone number from the user. The following code snippet is used

to create a telephone field:

```
<INPUT type="tel" name="usrtel">
```

- **image:** Is used to specify an image to be used as a `submit` button. When the value of the `type` attribute is `image`, some additional attributes, such as `height`, `width`, `alt`, and `src` can also be specified. The following list describes these attributes:
  - **height and width:** The `height` and `width` attributes specify the height and width for the image.
  - **alt:** The `alt` attribute is used to specify text for the alternative buttons to be displayed when the image specified in the `src` attribute could not be used.
  - **src:** The `src` attribute specifies the URL of the image that will be used as a submit button.

The following code snippet is used to create an image field:

```
<INPUT type="image" src="img_submit.gif" alt="submit" width="48" height="48" />
```

The preceding code displays the image to be used as the submit button, as shown in the following figure.



*The Image Used as the Submit Button*

## Defining the value Attribute

The `value` attribute specifies the value of the field. It behaves differently for different input types, as shown in the following list:

- For `button`, `reset`, and `submit` input types, the `value` attribute defines the text that appears on the face of the buttons.
- For `text` and `password` input types, the `value` attribute defines the default value for the fields.
- For `checkbox`, `radio`, and `image` input types, the `value` attribute defines the values associated with the input fields. All the radio buttons in a group have a common name. Therefore, the `value` attribute is used to differentiate the radio buttons in a group. In addition when a form is submitted, the selected radio button is identified by using its `value` attribute. Similarly, the user can select one or more check boxes and submit the form. All the selected check boxes can be identified by using the `value` attribute.

The following code snippet is used to specify the `value` attribute:

```
<INPUT type="text" name="fname" value="Enter Your First Name">
```

In the preceding code snippet, `Enter Your First Name`, is assigned to the `value` attribute of the input field.

## Defining the name Attribute

The `name` attribute specifies the name for the input field. It is used to identify the form field when the form data is submitted. The following code snippet is used to specify a name for the input field:

```
<INPUT type="text" name="Text1" />
```

In the preceding code, the name, `Text1`, is assigned to the text field.

## Defining the ID Attribute

The `ID` attribute provides a unique ID to the input field. This attribute is used to access the input fields in the CSS or JavaScript code.

The following code snippet is used to specify a unique ID to the input field:

```
<INPUT type="text" ID="value">
```

In the preceding code, the ID, `value`, is assigned to the input field.

## Defining the autocomplete Attribute

The `autocomplete` attribute is used to specify whether a form element should have the autocomplete feature on or off. If it is on, the browser automatically completes the values in the form fields based on the values that the user has entered earlier. The following code snippet is used to specify the `autocomplete` attribute for a form:

```
<INPUT type="email" name="email" autocomplete="on">
```

In the preceding code snippet, the `autocomplete` attribute is set to `on` for the email input field.

## Defining the autofocus Attribute

The `autofocus` attribute is used to ensure that the form element has a focus when a Web page loads. The following code snippet is used to specify an `autofocus` attribute for an input field:

```
<INPUT type="text" name="lname" autofocus>
```

In the preceding code snippet, the text field with the name, `lname`, will have the focus when the page loads.

## Defining the required Attribute

The `required` attribute is used to specify that an input field must not be left empty while submitting the form. The `required` attribute can be used with the input types, such as `text`, `tel`, `email`, `password`, `number`, `checkbox`, and `radio`. The following code snippet is used to specify the `required` attribute for the input field:

```
<INPUT type="text" name="username" required>
```

In the preceding code snippet, the `required` attribute ensures that the text field named `username` should not be left blank at the time of form submission.

## Defining the pattern Attribute

The `pattern` attribute is used to specify a regular expression against which the element's value will be checked. The `pattern` attribute can be used with the input types, such as `text`, `url`, `tel`, `email`, and `password`. The following table lists the expressions that can be applied on various input fields to create patterns.

<i><b>Expression</b></i>	<i><b>Description</b></i>
<code>[abc]</code>	<i>Finds the characters that are specified within the brackets.</i>
<code>[^abc]</code>	<i>Finds the characters that are not specified within the brackets.</i>
<code>[0-9]</code>	<i>Finds any digit from 0 to 9.</i>
<code>[A-Z]</code>	<i>Finds the character from uppercase A to uppercase Z.</i>
<code>[a-z]</code>	<i>Finds the character from lowercase a to lowercase z.</i>
<code>A-z</code>	<i>Finds the character from uppercase A to lowercase z.</i>

*The Expressions that can be Applied to Create Patterns*

The following code is used to specify the `pattern` attribute for the input field:

```
<INPUT type="text" name="country_code" pattern="[A-Za-z]{3}"
title="Three letter code">
```

In the preceding code, the value for the `pattern` attribute ensures that the text field accepts only three letter alphabets in capital or small letters. It will not allow the user to enter any number or special character in the text field.



*A regular expression is a set of characters, which is used to specify a pattern.*

## Defining the placeholder Attribute

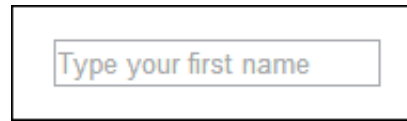
The `placeholder` attribute is used to specify a sample value for the input field that will be displayed until the user enters a value. The `placeholder` attribute can be used with the input types, such as `text`, `tel`, `email`, `password`, and `number`. The following code snippet is used to specify the `placeholder` attribute for an input field:

```
<INPUT type="text" placeholder="Type your first name" name="fname">
```

In the preceding code snippet, the text, `Type your First name here`, will appear in the text field

named `fname`. The text will appear when the page loads suggesting the user to enter the first name in the field.

The output derived by using the `placeholder` attribute with the text field is displayed, as shown in the following figure.



*The Output Derived by Using the Placeholder Attribute*

Consider the following code to create input fields, such as text, date, telephone number, email, and radio for the **OrderFoodOnline** Web page:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<STYLE>
body{
background-color:#FFEBCD;
}
.autostyle2{
color:red;
font-size:20px;
text-align:center;
}
</STYLE>
</HEAD>
<BODY bgcolor="#FFEBCD">
<FORM action = "registration.html">
<TABLE>
<TR class="autostyle2">
<TD colspan="4">OrderFoodOnline</TD></TR>
<TR>
<TD WIDTH="100%" COLSPAN="4">
</TD>
</TR>
<TR>
<TD> Name:</TD>
<TD><INPUT type="text" name="username" placeholder="Enter your name"
required></TD>
</TR>
<TR>
<TD>Date:</TD>
```

```

<TD><INPUT type="date" name="date" required></TD>

</TR>

<TR>

<TD>email ID:</TD>

<TD><INPUT type="email" name="usrmail" placeholder="Enter your email
ID" required></TD>

</TR>

<TR>

<TD>Contact Number:</TD>

<TD><INPUT type="tel" name="usrtel" placeholder="Enter your phone
number" required></TD>

</TR>

<TR><TD>Select Food:</TD>

<TD>Non-vegetarian<INPUT type="Radio" NAME="rd1">
Vegetarian<INPUT type="Radio" NAME="rd1" ></TD>

</TR>

</TABLE>

</FORM>

</BODY>

</HTML>

```

The preceding code creates a textbox to accept the name of the customer, a date field to accept the date, an email field to accept the validated email address of the customer, a telephone field to accept the telephone number, and radio buttons to accept the food preferences of the customer. The OrderFoodOnline form is displayed, as shown in the following figure.

*The OrderFoodOnline Form*

## <SELECT>

The <SELECT> tag is a container tag. It creates a drop-down list on the form. It has the following attributes:

- **multiple:** Is used to allow the user to select more than one value from the drop-down list by using the Ctrl key.
- **name:** Is used to specify a name of the selection list that will be used at the time of submitting

the form.

- `size`: Is used to specify the number of visible items in the selection or drop-down list. The default value is 1. If the value of this attribute is greater than 1, then the form field will be a list.
- `autofocus`: Is used to ensure that the focus is on the drop-down list when the page loads.
- `form`: Is used to specify the name of one or more forms to which the `<SELECT>` tag belongs.

The following code is used to create a `<SELECT>` tag:

```
<SELECT name= "5-Star_Hotels" size=1 multiple></SELECT>
```

The preceding code will create a drop-down list with the name, `5-Star_Hotels`, with the size, 1. The `multiple` attribute allows the user to select multiple items from the list.

However, the `<SELECT>` tag only creates a drop-down list. It does not embed list items in it. To specify the list items, you need to use the tags, `<OPTION>` and `<OPTGROUP>`, with the `<SELECT>` tag.

## Defining the `<OPTION>` Tag

It is always used within the `<SELECT>` tag and cannot be used as a standalone tag. It is used to create a list of options in the drop-down list and has the following attributes:

- `selected`: Is used to indicate that a particular option comes pre-selected when the page loads in the browser.
- `value`: Is used to indicate the value of the option to be sent on the form submission when that option is selected by the user.
- `disabled`: Is used to indicate that an option should be disabled when the page loads.

Consider the following code to create the drop-down lists in the **OrderFoodOnline** form:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
.....</TR>
<TR>
<TD>Select Restaurant:</TD>
<TD><SELECT>
<OPTION value="opt1">Select Your Restaurants</OPTION>
<OPTION value="opt2">La Figa</OPTION>
<OPTION value="opt3">Benihana</OPTION>
<OPTION value="opt4">Gallipoli</OPTION>
<OPTION value="opt5">Kings Road SteakHouse</OPTION>
</SELECT></TD>
</TR>
<TR>
<TD>Drinks:</TD>
<TD><SELECT>
```

```

<OPTION value="opt6">Select Your Drink</OPTION>
<OPTION value="opt7">Cappuccino</OPTION>
<OPTION value="opt8">Caffelatte</OPTION>
</SELECT></TD>
</TR>
<TR>
<TD> Soups:</TD>
<TD><SELECT>
<OPTION value="opt9">Select the Soup of Your Choice</OPTION>
<OPTION value="opt10">Minestrone</OPTION>
<OPTION value="opt11">Fonduta</OPTION>
<OPTION value="opt12">Pasta e fagioli</OPTION>
</SELECT></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>

```

In the preceding code, the drop-down lists, Drinks, Soups, and Restaurant, are created. The output derived by using the `<OPTION>` tag is displayed, as shown in the following figure.

*The Output Derived by Using the `<OPTION>` Tag*

## Defining the `<OPTGROUP>` Tag

The `<OPTGROUP>` tag is used to group the related options in one group. It is generally used when you have a long list of options and you want to group the related options in one to make it simpler. The `<OPTGROUP>` tag can have the following attributes:

- **disabled:** Is used to indicate that an option group should be shown disabled when the page loads.



- `label`: Is used to specify a label for the option group.

Consider the following code to create a drop-down list in the **OrderFoodOnline** Web page:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
.....</TR>
<TR>
<TD>Dishes:</TD>
<TD><SELECT>
<OPTION value="opt13">Select the Dishes of Your Choice</OPTION>
<OPTGROUP label="Italian">
<OPTION value="opt14">Pasta</OPTION>
<OPTION value="opt15">Fish</OPTION>
<OPTION value="opt16">Rice</OPTION>
</OPTGROUP>
<OPTGROUP label="Chinese">
<OPTION value="opt17">Chowmein</OPTION>
<OPTION value="opt18">Manchurian</OPTION>
<OPTION value="opt19">Water Chessnut Cake</OPTION>
</OPTGROUP>
</SELECT></TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

In the preceding code, the `<OPTGROUP>` tag is used to group the items in two categories, Italian and Chinese. Further, the `<OPTION>` tag is enclosed in between the `<OPTGROUP>` tag to specify the items in the categories.

The output derived by using the `<OPTGROUP>` tag is displayed, as shown in the following figure.

*The Output Derived by Using the <OPTGROUP> Tag*

## <LABEL>

The <LABEL> tag is used to define a label for the input fields. You can also define a label for the <OUTPUT> tag. The <LABEL> element does not render anything special for the user. However, it provides functionality for users in such a way that if they click on the text within the <LABEL> element, the corresponding field will automatically get selected. For this, the `for` attribute of the <LABEL> tag should be equal to the `ID` attribute of the related input field. The <LABEL> tag has the following attributes:

- `for`: Is used to bind the <LABEL> tag with the input field and should have the same value as the `ID` attribute of the input field.
- `form`: Is used to specify the name of one or more forms to which the <LABEL> tag belongs.

Consider the following code to create a label for the input fields in the **OrderFoodOnline** Web page:

```
<!DOCTYPE HTML>

<HTML>

<HEAD>

<STYLE>

body{

background-color:#FFEBD;
```

```
}  
.autostyle2{  
color:red;  
font-size:20px;  
text-align:center;  
}  
</STYLE>  
</HEAD>  
<BODY>  
    <FORM action = "registration.html">  
<TABLE>  
<TR class="autostyle2">  
<TD colspan="2">OrderFoodOnline</TD></TR>  
<TR>  
<TD WIDTH="100%" COLSPAN="2">  
</TD>  
</TR>  
<TR>  
<TD><LABEL for="name">Name:</LABEL></TD>  
<TD><INPUT type="text" name="username" ID="name" placeholder="Enter  
your  
name" required></TD>  
</TR>  
<TR>  
<TD><LABEL for="date">Date:</LABEL></TD>  
<TD><INPUT type="date" name="date" ID="date" required></TD>  
</TR>  
<TR>  
<TD><LABEL for="email">email ID:</LABEL></TD>  
<TD><INPUT type="email" name="usrmail" ID="email" placeholder="Enter  
your email ID" required></TD>  
</TR>  
<TR>  
<TD><LABEL for="number">Contact Number:</LABEL></TD>  
<TD><INPUT type="tel" name="usrtel" ID="number" placeholder="Enter  
your phone number" required></TD></TR>  
<TR><TD><LABEL for="food">Select Food:</LABEL></TD>  
<TD><LABEL for="nonveg">Non-vegetarian</LABEL>  
<INPUT type="radio" name="food" ID="nonveg">  
<LABEL for="veg">Vegetarian</LABEL>  
<INPUT type="radio" name="food" ID="veg"></TD>  
</TR>
```

```
<TR>
<TD><LABEL for="restro">Select Restaurant:</LABEL></TD>
<TD><SELECT>
<OPTION value="opt1">Select Your Restaurants</OPTION>
    <OPTION value="opt2">La Figa</OPTION>
    <OPTION value="opt3">Benihana</OPTION>
    <OPTION value="opt4">Gallipoli</OPTION>
    <OPTION value="opt5">Kings Road
SteakHouse</OPTION>
</SELECT></TD>
</TR>
<TR>
<TD><LABEL for="drinks">Drinks:</LABEL></TD>
<TD><SELECT>
<OPTION value="opt6">Select Your Drink</OPTION>
    <OPTION value="opt7">Cappuccino</OPTION>
    <OPTION value="opt8">Caffelatte</OPTION>
</SELECT></TD>
</TR>
<TR>
<TD><LABEL for="soups">Soups:</LABEL></TD>
<TD><SELECT>
<OPTION value="opt9">Select the Soup of Your Choice</OPTION>
    <OPTION value="opt10">Minestrone</OPTION>
    <OPTION value="opt11">Fonduta</OPTION>
    <OPTION value="opt12">Pasta e fagioli</OPTION>
</SELECT></TD>
</TR>
<TR>
<TD><LABEL for="dishes">Dishes:</LABEL></TD>
<TD><SELECT>
<OPTION value="opt13">Select the Dishes of Your Choice</OPTION>
<OPTGROUP label="Italian">
    <OPTION value="opt14">Pasta</OPTION>
    <OPTION value="opt15">Fish</OPTION>
    <OPTION value="opt16">Rice</OPTION>
</OPTGROUP>
<OPTGROUP label="Chinese">
    <OPTION value="opt17">Chowmin</OPTION>
    <OPTION value="opt18">Manchurian</OPTION>
    <OPTION value="opt19">Water Chessnut Cake</OPTION>
```

```

</OPTGROUP>
</SELECT></TD>

</TR>

<TR>

<TD><LABEL for="Order For:">Order For:</LABEL></TD>
<TD><LABEL for="pickup">Take Away</LABEL>
<INPUT type="radio" name="pickup" ID="pickup">
<LABEL for="home">Home Delivery</LABEL>
<INPUT type="radio" name="pickup" ID="home"></TD>

</TR>

</TABLE>

</FORM>

</BODY>

</HTML>

```

In the preceding code, the `<LABEL>` tag is used to label the input fields in the **OrderFoodOnline** Web page. You can select the input field by clicking on the label of the field. The output derived by using the `<LABEL>` tag is displayed, as shown in the following figure.

*The Output Derived by Using the `<LABEL>` Tag*

## **<FIELDSET>**

The `<FIELDSET>` tag is used to combine and group related fields in a form. It creates a box around the selected fields. You can also define the description for the fieldset by using the `<LEGEND>` tag. The `<LEGEND>` tag is used along with the `<FIELDSET>` tag to define a caption for the fieldset. It is the simplest way of organizing form elements along with their description in such a way that it is easier for a user to understand.

The `<FIELDSET>` tag can have the following attributes:

- `disabled`: The `disabled` attribute is used to indicate that a group of fields should be shown disabled when the page loads.
- `form`: The `form` attribute is used to specify the name of one or more forms to which the `<FIELDSET>` tag belongs.
- `name`: The `name` attribute is used to specify the name for the fieldset.

Consider the following code to group the drop-down lists inside a fieldset on the **OrderFoodOnline** Web page:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<FORM>
<FIELDSET>
Drinks:
<SELECT>
    <OPTION value="opt1">Select Your Drink</OPTION>
    <OPTION value="opt2">Cappuccino</OPTION>
    <OPTION value="opt3">Caffelatte</OPTION>
</SELECT>
Soups:
<SELECT>
    <OPTION value="opt4">Select the Soup of Your Choice</OPTION>
    <OPTION value="opt5">Minestrone</OPTION>
    <OPTION value="opt6">Fonduta</OPTION>
    <OPTION value="opt7">Pasta e fagioli</OPTION>
</SELECT>
Dishes:
<SELECT>
    <OPTION value="opt8">Select the Dishes of Your Choice.</OPTION>
    <OPTGROUP label="Italian">
        <OPTION value="opt9">Pasta</OPTION>
        <OPTION value="opt10">Fish</OPTION>
        <OPTION value="opt11">Rice</OPTION>
    </OPTGROUP>
    <OPTGROUP label="Chinese">
        <OPTION value="opt12">Chowmin</OPTION>
        <OPTION value="opt13">Manchurian</OPTION>
        <OPTION value="opt14">Water Chessnut Cake</OPTION>
    </OPTGROUP>
</SELECT>
```

```

</FIELDSET>

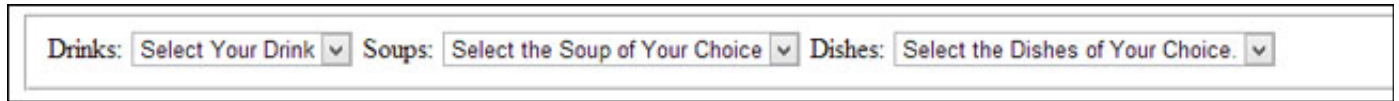
</FORM>

</BODY>

</HTML>

```

The preceding code groups the food items in one box. The output derived by using the `<FIELDSET>` tag is displayed, as shown in the following figure.



The screenshot shows a single rectangular box with a thin border. Inside the box, there are three dropdown menus arranged horizontally. The first is labeled 'Drinks:' followed by a dropdown menu with the text 'Select Your Drink'. The second is labeled 'Soups:' followed by a dropdown menu with the text 'Select the Soup of Your Choice'. The third is labeled 'Dishes:' followed by a dropdown menu with the text 'Select the Dishes of Your Choice'.

*The Output Derived by Using the `<FIELDSET>` Tag*

## <TEXTAREA>

The `<TEXTAREA>` tag creates a field in which the user can enter a large amount of text. The `<TEXTAREA>` tag has the following attributes:

- `rows`
- `cols`

### rows

The `rows` attribute helps to set the number of rows of text that will be visible without scrolling up or down in the field.

### cols

The `cols` attribute helps to set the number of columns of text that will be visible without scrolling right or left in the field.

Consider the following code to create a textarea in the **OrderFoodOnline** Web page to accept the users' address:

```

<!DOCTYPE HTML>

<HTML>

<HEAD>

.....</TR>

<TR>

<TD><LABEL for="Orderfor">Address:</LABEL></TD>

<TD><TEXTAREA rows="3" cols="16" ID="Orderfor">

Enter Your Address Here

</TEXTAREA></TD>

</TR></TABLE>

</FORM>

</BODY>

</HTML>

```

The output derived by using the `<TEXTAREA>` tag is displayed in the following figure.

*The Output Derived by Using the `<TEXTAREA>` Tag*

## **`<DATALIST>`**

The `<DATALIST>` tag is used to create a list of pre-defined options for an input field. It is used to provide an autocomplete feature on input fields so that the user can view the drop-down list of pre-defined options whenever they input data.

Consider the following code to create a datalist named **State** in the **OrderFoodOnline** Web page:

```
<!DOCTYPE HTML>

<HTML>

<HEAD>
.....</TR>
<TR>
<TD><LABEL for="state">State:</LABEL></TD>
<TD><INPUT list="stat" name="stat" ID="state">
<DATALIST ID="stat">
  <OPTION value="Alabama">
  <OPTION value="California">
  <OPTION value="Delaware">
  <OPTION value="Florida">
  <OPTION value="Hawaii">
</DATALIST><BR>
</TD>
```



```

</TR> </TABLE>

</FORM>

</BODY>

</HTML>

```

The preceding code creates a datalist so that whenever a user types the initial characters of the name of a state, the names matching the initials will appear in the drop-down list, as shown in the following figure.

**OrderFoodOnline**

Name:

Date:

email ID:

Contact Number:

Select Food: ☐ Non-vegetarian ☐ Vegetarian

Select Restaurant:

Drinks:

Soups:

Dishes:

Order For: ☐ Take Away ☐ Home Delivery

Address:

State:

*The Output Derived by Using the <DATALIST> Tag*

In the preceding figure, when a user types the character, c, California is prompted in the list.

## <KEYGEN>

The <KEYGEN> tag is used to specify a key-pair generated field in a form. Whenever the form is submitted, the private and public keys are generated, where the private key is stored locally, and the public key is sent to the server. As the public key is stored in the server, it can be used to authenticate a user in the future. The <KEYGEN> tag can have the following attributes:

- **autofocus:** Is used to specify that the <KEYGEN> tag automatically gets the focus when a Web page loads.
- **disabled:** Is used to indicate that a <KEYGEN> tag should be shown disabled when a page loads.
- **name:** Is used to specify a name for the <KEYGEN> tag.
- **keytype:** Is used to specify the security algorithm of the key. It accepts the name of various

security algorithms, such as `rsa`, `dsa`, and `ec`, as its value.

- `form`: Is used to specify the name of one or more forms to which the `<KEYGEN>` tag belongs.



*The `rsa`, `dsa`, and `ec` are the different security algorithms that are used for public key encryption.*

The `<KEYGEN>` tag can be created using the following code:

```
<KEYGEN name="key1" keytype="rsa">
```

The preceding code creates a key-pair generated field using the security algorithm, `rsa`.

## <OUTPUT>

The `<OUTPUT>` tag is used to represent the result of a calculation. The `<OUTPUT>` tag can have the following attributes:

- `for`: Is used to specify the relationship between the input fields used and the result generated for the calculation.
- `form`: Is used to specify the name of one or more forms to which the `<OUTPUT>` tag belongs.
- `name`: Is used to specify a name for the `<OUTPUT>` tag.

Consider the following code to get the result of the multiplication of two numbers by using the `<OUTPUT>` tag:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<FORM ONINPUT="mul.value=parseInt(val1.value)*parseInt(val2.value)">
Value1:<INPUT type="text" name="val1">
* Value2: <INPUT type="text" name="val2">
=<OUTPUT name="mul" for="val1 val2"></OUTPUT>
</FORM>
</BODY>
</HTML>
```

The preceding code creates two input fields, `val1` and `val2`, to accept user input. Further, the `ONINPUT` event executes on the form and multiplies the values present in the input fields, `val1` and `val2`. Then, the `<OUTPUT>` tag is used to display the result of the calculation. The output derived by using the `<OUTPUT>` tag is displayed, as shown in the following figure.

Value1:  \* Value2:  =200

*The Output Derived by Using the `<OUTPUT>` Tag*

## <BUTTON>

The <BUTTON> tag is used to create a button. However, unlike the input type, submit, you can specify a text or an image within the button by using the <P> or <IMG> tags. The <BUTTON> tag has the following attributes:

- **type**: Is used to specify the type of the button. It can accept the values, such as `button`, `submit`, and `reset`. The `button` value creates a simple push button, the `submit` value creates a button that submits the form data, and the `reset` value creates a button that reset the form fields.
- **name**: Is used to specify the name of the button.
- **form**: Is used to specify the name of one or more forms to which the button belongs.
- **autofocus**: Is used to specify that the button automatically gets the focus when the Web page loads.
- **disabled**: Is used to indicate that the button should be shown disabled when the Web page loads.

Consider the following code to create buttons, Submit and Reset, in the **OrderFoodOnline** Web page.

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<STYLE>
#button{
Margin-left:50px;
}
....
/HEAD
BODY
.....</TR>
<TR>
<TD>
<BUTTON ID="button" type="submit"><P>Submit</P></BUTTON></TD><TD>
<BUTTON type="reset"><P>Reset</P></BUTTON>
</TD>
</TR>
</TABLE>
</FORM>
</BODY>
</HTML>
```

The preceding code creates two buttons, Submit and Reset, as displayed in the following figure.

## OrderFoodOnline

Name:

Date:

email ID:

Contact Number:

Select Food: ☐ Non-vegetarian ☐ Vegetarian

Select Restaurant:

Drinks:

Soups:

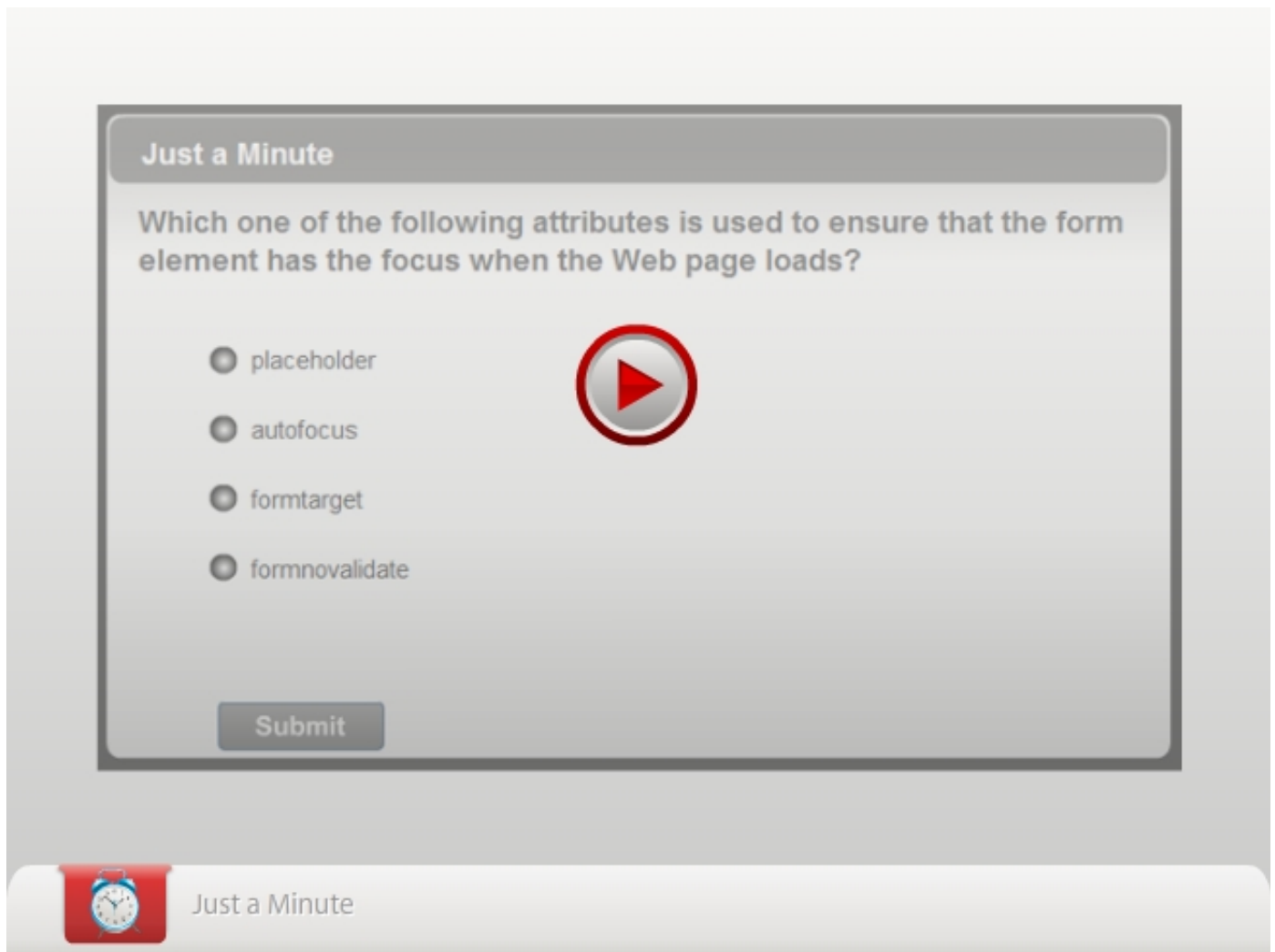
Dishes:

Order For: ☐ Take Away ☐ Home Delivery

Address:

State:

*The Output Derived by using the < BUTTON> Tag*



## Activity 5.1: Designing an HTML Form

### Manipulating the Components of a Web Page

The website of BookYourHotel.com enables the customers to access the online booking form for each hotel by clicking on the image of the hotel. In addition, a clock is displayed on the Web page, which is dynamically updated every second to display the current time to the users.

To implement such functionalities on the Web pages, JavaScript is used. It is an object-based language and treats every element, such as an image or a button inside a browser window as an object. Every object has a pre-defined set of attributes and events associated with it. JavaScript also provides various types of built-in objects, such as browser objects and form objects, which help to make a Web page dynamic and interactive.

### Working with Browser Objects

When a Web page is displayed in a browser, a user can access information not only about the current page but also additional details, such as information about the window and screen, the pages the user has visited in the past, or the version of the browser being used to view the document.

All this information is accessible by using browser objects. In addition, the browser parses the currently

displayed Web page into multiple objects, such as a window, screen, and a document. The objects that define the browser content and the browser itself are known as browser objects. Browser objects enable retrieval and manipulation of information about a browser, such as the window size, height, width, and name. These objects also enable access to information, such as browsing history and current version of a browser.

JavaScript defines the following browser objects on a Web page:

- `window`
- `document`
- `navigator`
- `screen`
- `history`
- `location`

Browser objects represent the browser environment and provide properties and methods for its access and manipulation. The browser environment refers to components, such as the window in which the document is displayed and the history list that contains information regarding the Web pages visited by a user. You can set the size, name, and default status of the browser window by using the `window` object. You can also manipulate URL of the document using the `location` object. Similarly, you can access the browser version and the browser name in which the Web page is displayed using the `navigator` object. This information can be useful for a correct display of a Web page in the browser.

## Using the window Object

The `window` object is one of the highest-level objects in the JavaScript object hierarchy. It represents a browser window, which displays the document. It can also be a combination of multiple frames. Each frame within a window is itself a window. The following table lists some of the properties of the `window` object.

<i>Properties</i>	<i>Description</i>	<i>Syntax</i>
<code>defaultStatus</code>	<i>Is a string value containing the default text that appears on the status bar of the window.</i>	<code>window.defaultStatus</code>
<code>document</code>	<i>Is a reference to the document displayed in the window.</i>	<code>window.document</code>
<code>frames[]</code>	<i>Is an array that represents all the frames in the window. You can refer to a particular frame by using the <code>frames[]</code> property.</i>	<code>window.frames[i]</code> , where <i>i</i> is the index of a particular frame in a window.
<code>frames.length</code>	<i>Is an integer value representing the number of frames in the window.</i>	<code>window.frames.length</code>

<i>name</i>	<i>Returns or sets a name for the window.</i>	<i>window.name</i>
<i>parent</i>	<i>Returns the parent window of the current window.</i>	<i>window.parent</i>
<i>self</i>	<i>Returns the current window.</i>	<i>window.self</i>
<i>top</i>	<i>Returns the topmost browser window. The topmost window is the current active window overlapping all the open windows.</i>	<i>window.top</i>
<i>status</i>	<i>Is a string value and is used to set the text on the status bar of the window.</i>	<i>window.status</i>

*The Properties of the window Object*

The following table lists some of the methods of the `window` object.

Methods	Description	Syntax	Example
<code>open()</code>	Opens a new browser window.	<p><code>window.open(URL, Name, specs, replace);</code></p> <p>where, <i>URL</i> is the address of the Web page to be displayed.</p> <p><i>Name</i> is the name of the window in which the Web page is to be displayed.</p> <p><i>specs</i> define the specifications of the window, such as its height and width, in the form of a comma-separated list of attributes.</p> <p><i>replace</i> is used to determine whether to replace the current document or create a new entry in the history list using <i>true</i> or <i>false</i> values, respectively.</p>	<p><code>window.open</code> <code>("index.html",</code> <code>"Home",</code> <code>"height=100,width=200",</code> <code>false);</code></p> <p>where, the URL is <code>index.html</code>.</p> <p>The Name of the window is <code>Home</code>.</p> <p>The height and width of the screen is 100 and 200, respectively.</p>
<code>close()</code>	Closes the current window.	<code>window.close();</code>	<p><code>window.close();</code></p> <p>Closes the current window.</p>
<code>alert()</code>	Displays a message in a dialog box.	<p><code>window.alert(messageText);</code></p> <p>where, <i>messageText</i> is the text to be displayed in the dialog box.</p>	<p><code>window.alert("Hello");</code></p> <p>Displays the message <code>Hello</code> in a dialog box.</p>
<code>setTimeout()</code>	Calls a function after a specified number of milliseconds.	<code>window.setTimeout(function, time)</code>	<p><code>var t=setTimeout</code> <code>("alertMessage()",3000);</code></p> <p>Calls the <code>alertMessage()</code> function after 3 seconds.</p>
<code>clearTimeout()</code>	Cancels a timer that is set using the <code>setTimeout()</code> method. It takes the <code>timerID</code> parameter returned by the <code>setTimeout()</code> method.	<code>window.clearTimeout(timerID)</code>	<p><code>t=setTimeout</code> <code>("timedCount()",1000);</code> <code>clearTimeout(t);</code></p> <p>where, <i>t</i> is the <code>timerID</code> parameter returned by the <code>setTimeOut()</code> method.</p>

### The Methods of the window Object



The window object is a default object when writing the JavaScript code. Therefore, it is not necessary to explicitly qualify the methods and properties of the window object using the dot operator.

Consider the following code snippet for opening a new window displaying the hotel bookings form, when a user clicks the hotel image:

```
function open_win()
```



```
{
    window.open("HotelBooking1.html","height=100,width=200");
}
```

In the preceding code snippet, when the function, `open_win()`, is executed, it opens the **purchase.html** Web page in a new window.

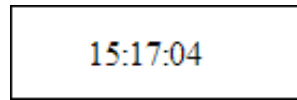
Consider the following code to display the clock on the Web page of the BookYourHotel.com website by using the `setTimeout()` method:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function clockTime()
{
    var todayDate=new Date();
    var hrs=todayDate.getHours();
    var mns=todayDate.getMinutes();
    var scs=todayDate.getSeconds();
    mns=check(mns);
    scs=check(scs);
    document.getElementById('displayTime').innerHTML=hrs+":"+mns+":"+scs;
    t=setTimeout('clockTime()',1000);
}
function check(t)
{
    if (t<10)
    {
        t="0" + t;
    }
    return t;
}
</SCRIPT>
</HEAD>
<BODY onload="clockTime()">
<DIV ID="displayTime"></DIV>
</BODY>
</HTML>
```

In the preceding code, the `clockTime()` function is called to display a clock on the Web page. The `clockTime()` function uses the `Date` object to get the current time in hours, minutes, and seconds. It updates time every second using the `setTimeout()` function. The `setTimeout()` function calls the `clockTime()` function after every second or 1000 milliseconds. The `clockTime()` function uses the `check(t)` function to check if the number of minutes or seconds is less than 10. If this is true, 0 is

displayed before the number of minutes or seconds. For example, if the current time is 14 hours 45 minutes and 3 seconds, the time is displayed as 14:45:03.

The output of the preceding code to display a clock on a Web page is displayed, as shown in the following figure.



*The Clock Displayed on a Web Page*

## Using the document Object

The `document` object is subordinate to the `window` object in the document object model hierarchy. The `document` object provides the properties and methods to work with many aspects of the current document, including information about anchors, forms, links, title, current location and URL, and the current colors. The HTML document that is loaded on the Web browser window acts as a `document` object.

The following table lists some of the properties of the `document` object.

<i><b>Properties</b></i>	<i><b>Description</b></i>
<code>alinkColor</code>	<i>Is a string value representing the color of an active link.</i>
<code>anchors[]</code>	<i>Is an array object containing references to all the anchor elements in a document.</i>
<code>bgColor</code>	<i>Is a string value representing the background color of the document.</i>
<code>cookie</code>	<i>Is a string value containing name/value pairs of data that will persist in the memory of the client computer until the Web browser is cleared or the expiry date is reached.</i>
<code>fgColor</code>	<i>Is a string value representing the text</i>

	<i>color of the document.</i>
<code>forms[]</code>	<i>Is an array object containing references to each form in the document. Form elements are contained within the form object.</i>
<code>linkColor</code>	<i>Is a string value representing the color of an unvisited link.</i>
<code>lastModified</code>	<i>Is a string value representing the date and time when the document was last modified.</i>
<code>links[]</code>	<i>Is an array object containing references of all the elements in the &lt;A&gt; tag and elements that use the &lt;AREA&gt; tag.</i>
<code>referrer</code>	<i>Is a string value representing the URL of the document from which the current document is accessed.</i>
<code>title</code>	<i>Is a string value representing the title of the document.</i>
<code>vlinkColor</code>	<i>Is a string value representing the color of the visited link.</i>

### *The Properties of the document Object*

Some of the widely used methods of the `document` object are:

- `write()`
- `writeln()`
- `getElementsByName()`

- `getElementsByTagName()`
- `getElementById()`

## `write()`

The `document.write()` method enables users to write text on a Web page. The following syntax is used to write the text , Hello!, using the `document.write()` method on a Web page:

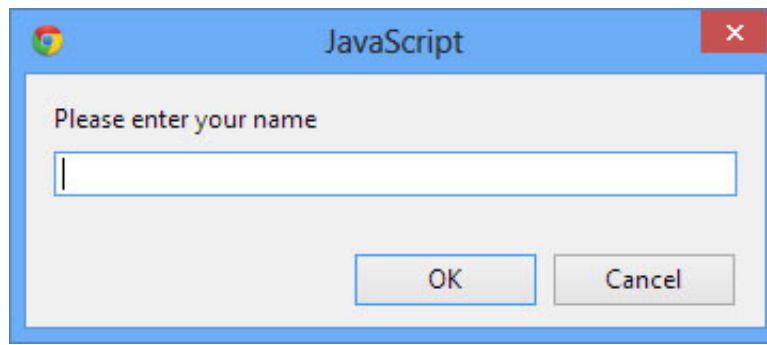
```
document.write("Hello!");
```

You can also add HTML elements to a Web page dynamically using the `document.write()` method.

For example, you can accept a name as input and then use the `document.write()` method to write it on a Web page. You can even dynamically specify the formatting of the content on the Web page, as shown in the following code:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<TITLE>JavaScript Write method Illustration</TITLE>
<STYLE>
body{
background-color:#DAA520;
}
</STYLE>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
{
var name= prompt("Please enter your name","");
document.write("<P>");
document.write("<I>");
document.write("<B>");
document.write("Welcome "+name);
document.write("</B>");
document.write("</I>");
document.write("</P>");
}
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the `prompt()` method is used to prompt the user to enter his/her name, as shown in the following figure.



*The Prompt Box*

Assuming that the user enters the name, George, in the prompt box, the output is displayed as shown in the following figure.



*The Output of Code Using the write() Method*

In the preceding code, the `document.write()` method is used in the `<SCRIPT>` tag to display the welcome message on the Web page.

## `writeln()`

The `writeln()` method also writes text on a Web page. The only difference is that the `writeln()` method appends a carriage return character to the end of the output. The carriage return character is used to write data on separate lines of the Web page. For example, consider the following code snippet:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<PRE>
<SCRIPT type="text/javascript">
document.writeln("Hi!");
document.writeln("Welcome to our site!");
document.write("Have a ");
document.write("great day");
</SCRIPT>
</PRE>
</BODY>
```

&lt;/HTML&gt;



*The `writeln()` method appends a newline character at the end of the text. However, HTML ignores newline when writing the output text. The `<PRE>` tag can be used to display the pre-formatted output on a Web page. When the `<SCRIPT>` tag is enclosed within the `<PRE>` tag, the newline appended using the `document.writeln()` method is rendered.*

When the preceding code snippet is executed, the message, **Hi!**, is displayed, and the carriage return character is added at the end of this message so that the next message is displayed in a new line.

The output of the preceding code snippet is displayed, as shown in the following figure.

```
Hi!
Welcome to our site!
Have a great day
```

*The Output Derived Using the `writeln()` Method*

## getElementsByTagName()

It is recommended that each element on a Web page should be specified by a unique name. However, multiple elements can also share the same name in a Web page. The `document.getElementsByTagName()` method is used to access all the elements with the specified name used in an HTML document. This method returns an array of all the elements with the specified name in the HTML document. The following syntax is used for the `getElementsByTagName()` method:

```
document.getElementsByTagName("name_of_the_element");
```

In the preceding syntax, `name_of_the_element`, specifies the name of the element(s) to be accessed.

Consider the following code:

```
<!DOCTYPE HTML>
<HTML>
<SCRIPT type="text/javascript">
function count()
{
var x=document.getElementsByTagName("link");
alert(x.length + "Hyperlinks");
}
</SCRIPT>
<BODY>
<A name="link" href="" >Link 1</A><BR/>
<A name="link" href="" >Link 2</A><BR/>
<A name="link" href="" >Link 3</A><BR/>
```

```
<BR/>
<INPUT type="button" value="Count" onclick="count()" />
</BODY>
</HTML>
```

In the preceding code, the `length` property is used to count the occurrences of all the elements that have their name specified as **link**. Therefore, the statement `x.length` will generate the value, 3, and display it in an alert box.

## getElementsByTagName()

Consider a scenario, where a user can customize the background color of all the text boxes on a Web page. A drop-down list is used to choose the background color. If the user selects the green color from a drop-down list, the background color of all the text boxes in the document changes to green. This can be achieved using the `document.getElementsByTagName()` method. The `document.getElementsByTagName()` method is used to access all the elements of the same type in the Web page. In the given scenario, the `document.getElementsByTagName()` method can be used to return an array of all the text boxes in the document which can then be manipulated using CSS to achieve the desired functionality. The following syntax is used for the `getElementsByTagName()` method:

```
document.getElementsByTagName("Tag_name");
```

In the preceding syntax, `Tag_name` specifies the tag name of the element(s) that have to be accessed.

Consider the following code snippet:

```
var x=document.getElementsByTagName("a");
alert(x.length + "Hyperlinks");
```

In the preceding code snippet, the number of anchor or `<A>` tags on the Web page is counted and displayed in an alert box. For example, if there are four `<A>` tags in a document, then the output of the preceding code will be 4 Hyperlinks.

## getElementById()

As you know, each HTML element used on a Web page has an optional attribute, `ID`, which uniquely identifies the element. The `document.getElementById()` method uses the ID of HTML elements to access and manipulate their content.

The following syntax is used to access an element using the `getElementById()` method:

```
document.getElementById("id_of_the_element");
```

In the preceding syntax, `id_of_the_element`, specifies the ID of the element that has to be accessed.

Consider the following code snippet to use a text box within the Web page:

```
<INPUT type="text" ID="text1"/>
```

To access the value of the text box inside the script, you need to use the following code snippet:

```
var name = document.getElementById("text1").value;
```

In the preceding code snippet, the value of the text box is fetched using `ID` of the text box specified as `text1`. The value is then stored in the `name` variable.



*The `value` property is used to retrieve the value of the elements that are specified using the `<INPUT>` tag.*

Using JavaScript, you can also change the content of the HTML elements, such as paragraphs, hyperlinks, and headers. To retrieve and change the text of the HTML elements that have both a start tag and an end tag, such as `<P>`, `<OPTION>`, and `<DIV>`, you can make use of the `innerHTML` property with the `document.getElementById()` method. Consider the following code snippet:

```
<!DOCTYPE HTML>

<HTML>

<SCRIPT type="text/javascript">
function replacetext()
{
document.getElementById("para1").innerHTML="Changed the content by
using the innerHTML property of getElementById() method";
}
</SCRIPT> <BODY>

<P ID="para1">A simple paragraph</P>

<INPUT type="button" value="Change text" onclick="replacetext()" />
</BODY>

</HTML>
```

In the preceding code snippet, when the **Change text** button is clicked, the `replacetext()` function is called using the `onclick` event. The `replacetext()` function replaces the text of the `<P>` element with the text, **Changed the content by using the `innerHTML` property of `getElementById()` method**.



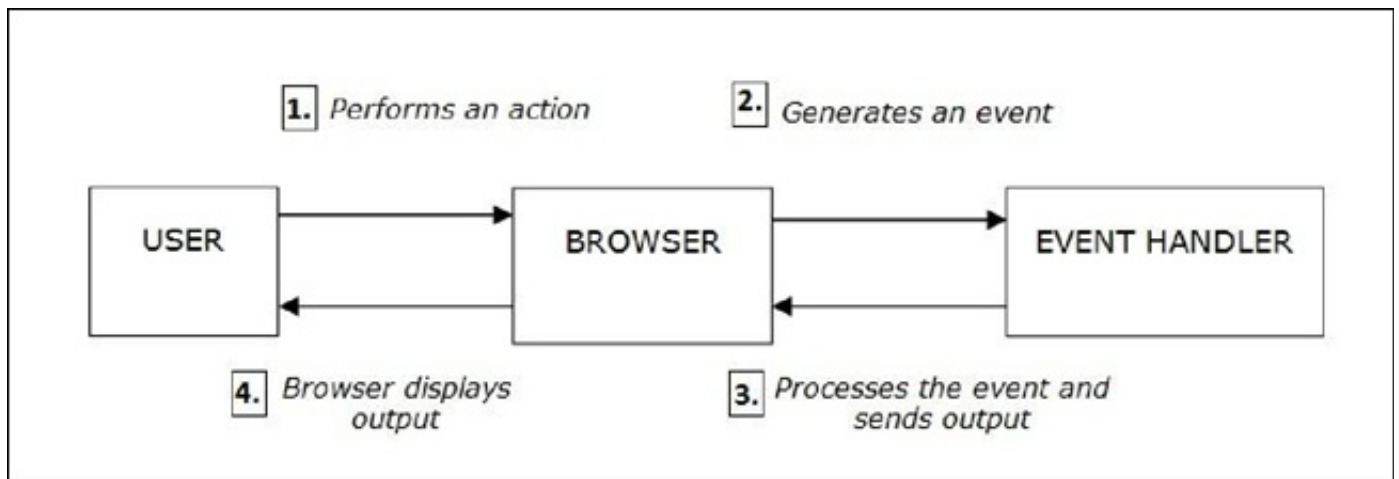
*Events are discussed in the next topic of this section.*

## Handling Events

In JavaScript, an event is an action that happens on a Web page, such as a mouse click or loading of a Web page. The Web browser waits for the event to occur, and when it occurs, performs the processing that has been programmed for the event. For example, when a user clicks the **OK** button on a form, the Web browser executes a part of the code, which is designed to execute on that event. This is called event handling. The function, which is executed in response to an event, is called an event handler.

The following figure displays the event handling process.





*The Event Handling Process*

The preceding figure depicts the process where a user interacts with the browser and an event is generated. This event is processed by the event handler and the output is sent to the browser. Further, the browser displays the output to the user.

The following table lists the various types of events supported in JavaScript.

<i><b>Event</b></i>	<i><b>Description</b></i>	<i><b>Example</b></i>
<i>onblur</i>	<i>Occurs when an element loses focus.</i>	<code>&lt;INPUT type="text" name="textbox1" onblur="show()" /&gt;</code> <i>where, show() is the function to be executed when the event occurs.</i>
<i>onchange</i>	<i>Occurs when the content of a field changes.</i>	<code>&lt;INPUT type="text" ID="fname" onchange="show()" /&gt;</code> <i>where, show() is the function to be executed when the event occurs.</i>
<i>onclick</i>	<i>Occurs when the user clicks an object.</i>	<code>&lt;BUTTON onclick="show()"&gt;Show Text&lt;/BUTTON&gt;</code> <i>where, show() is the function to be executed when the event occurs.</i>
<i>onfocus</i>	<i>Occurs when an element gets focus.</i>	<code>&lt;INPUT type="text" ID="firstname" onfocus="alert('hello')"; /&gt;</code>
<i>onkeydown</i>	<i>Occurs</i>	<code>&lt;INPUT type="text" onkeydown="alert('key is</code>

	<i>when any key of the keyboard is pressed.</i>	<code>down ') "/&gt;</code>
<code>onkeypress</code>	<i>Occurs when a character key of the keyboard is pressed or held down.</i>	<code>&lt;INPUT type="text" onkeypress="alert('key is pressed') "/&gt;</code>
<code>onkeyup</code>	<i>Occurs when a keyboard key is released.</i>	<code>&lt;INPUT type="text" onkeyup="alert('key is released') "/&gt;</code>
<code>onload</code>	<i>Occurs when a page or image has finished loading.</i>	<code>&lt;BODY onload="load()" &gt;</code> <i>where, load() is the function to be executed when the event occurs.</i>
<code>onmousedown</code>	<i>Occurs when the mouse button is pressed.</i>	<code>&lt;IMG src="image11.gif" onmousedown="alert('You clicked image11!') "&gt;</code>
<code>onmousemove</code>	<i>Occurs when the mouse is moved.</i>	<code>&lt;IMG src="image11.gif" onmousemove="alert('You moved your mouse over image11!') "&gt;</code>
<code>onmouseout</code>	<i>Occurs when the mouse is moved off an element.</i>	<code>&lt;P onmouseout="alert('Mouse moved out of the paragraph') "&gt; Move the mouse pointer out of my paragraph to display an alert box.&lt;/P&gt;</code>
<code>onmouseover</code>	<i>Occurs when the mouse moved over</i>	<code>&lt;P onmouseover="alert('moved over the paragraph') "&gt; Move the mouse pointer over me to display an alert box.&lt;/P&gt;</code>

	<i>an element.</i>	
<i>onmouseup</i>	<i>Occurs when the mouse button is released.</i>	<code>&lt;IMG src="image11.gif" onmouseup="alert('You clicked image11')" /&gt;</code>
<i>onselect</i>	<i>Occurs when a text is selected.</i>	Select text: <code>&lt;INPUT type="text" value="Select me!" onselect="alert('You have selected the text.')" /&gt;</code>
<i>onunload</i>	<i>Occurs when the user exits the page.</i>	<code>&lt;BODY onunload="alert('The page is unloaded')"&gt;</code>
<i>ondblclick</i>	<i>Occurs when the user double-clicks an object.</i>	<code>&lt;BUTTON ondblclick="show()" "&gt;Show Text&lt;/BUTTON&gt;</code> where, <i>show()</i> is the function to be executed when the event occurs.
<i>onerror</i>	<i>Occurs when an error occurs while loading a document or an image.</i>	<code>&lt;IMG src="image1.jpg" onerror="alert('Cannot load image.')"&gt;</code>

### *The Events Supported in JavaScript*

Consider the scenario of BookYourHotel.com website. The user registration page displays the registration form, which a user should fill to get registered on the website. The management wants to implement the following functionality on the registration page:

- If user leaves any text box blank, its background color should change to pink, otherwise, it should remain white.
- After the user fills the required details in the user registration form and clicks the **Register me** button, a confirm box should be displayed highlighting the registration information entered by the user.

Consider the following code to implement these functionalities:

```
<!DOCTYPE HTML>

<HTML> <HEAD>
```

```
<TITLE>REGISTRATION DETAILS</TITLE>

<STYLE>

h1,h3{
color: black;
font-size: 40px;
text-align:center;
}

table{
margin-left:650px;
border:2px solid white;
background-color:beige;
}

td{
padding:10px;
border:2px solid white;}

#button{
margin-left:740px;}

#button{
margin-left:740px;}

</STYLE>

<SCRIPT>

function show()
{
var fname = document.getElementById("txtbox1").value;
var lname = document.getElementById("txtbox2").value;
var age = document.getElementById("age").value;
var address = document.getElementById("address").value;
var gender=document.getElementById("gender").value;
confirm("You have entered:" + "\n Name : " + fname + " " + lname +
"\n Age : " + age + "\n Address : " + address + "\n Gender : " + gender
+ "\n\n Do you want to confirm these details ?");
}

function changeColor(val)
{
if((val.value=="") || (val.value==null))
{
val.style.background="pink";
}
else
{
val.style.background="#FFFFFF";
}
```

```

}
}
</SCRIPT>
</HEAD>
<BODY>
<H1>USER REGISTRATION DETAILS<HR/></H1>
<H3>Please fill the following details and get registered !!</H3><BR/>
<BR/>
<TABLE>
<TR>
<TD> First name: </TD>
<TD> <INPUT type="text" name="name1" ID="txtbox1"
onblur="changeColor(this)"/> </TD>
</TR> <TR>
<TD> Last name: </TD>
<TD><INPUT type="text" name="name2" ID="txtbox2"
onblur="changeColor(this)"/></TD>
</TR> <TR>
<TD> Age: </TD>
<TD> <INPUT type="text" name="age_box" ID="age"
onblur="changeColor(this)"/> </TD>
</TR> <TR>
<TD> Address:</TD>
<TD><TEXTAREA rows="5" name="address_box" ID="address"
onblur="changeColor(this)"></textarea></TD>
</TR> <TR>
<TD> Gender: </TD> <TD>
<SELECT name="Gender" ID="gender"> <OPTION value="Male">Male</OPTION>
<OPTION value="Female">Female</option> </SELECT> </TD>
</TR> </TABLE>
<BR/><INPUT ID="button" type="button" value="Register Me"
onclick="show()"/>
</BODY>
</HTML>

```



*The keyword, this, refers to the current objects, such as the text box and button, in the document.*

The Web page for the preceding code is displayed in the following figure.

## USER REGISTRATION DETAILS

**Please fill the following details and get registered !!**

First name:	<input type="text"/>
Last name:	<input type="text"/>
Age:	<input type="text"/>
Address:	<input type="text"/>
Gender:	<input type="text" value="Male"/>
<input type="button" value="Register Me"/>	

*The Browser Output*

The preceding output allows the user to enter details in the registration form. As the user fills a text box and presses the **Tab** key to move to the next one, the background color of the unfilled text box changes to yellow. The following figure displays the completed User Registration form.

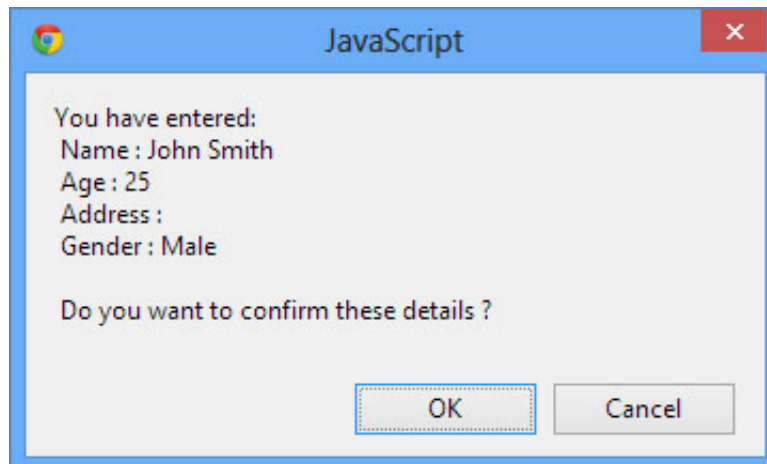
## USER REGISTRATION DETAILS

**Please fill the following details and get registered !!**

First name:	<input type="text" value="John"/>
Last name:	<input type="text" value="Smith"/>
Age:	<input type="text" value="25"/>
Address:	<input type="text"/>
Gender:	<input type="text" value="Male"/>
<input type="button" value="Register Me"/>	

*The Completed User Registration Form*

When the user clicks the **Register Me** button, a confirm dialog box is displayed, as shown in the following figure.



*A Confirm Dialog Box*

## Event Listeners

In JavaScript, you can handle user actions or events by calling specific methods known as event handlers. To handle each and every user action or event, a listener is notified. A listener is an object that waits for an event to occur and performs certain actions corresponding to it. However, for this, you need to register it with the event handler. The event handler function registered with the event of the field processes the event and provides an appropriate response to the listener. Then, the action is performed by the event listener.

To register a handler, you need to use the `addEventListener()` function. The function receives an event object that encapsulates the event information. The following syntax can be used to create an `addEventListener()` function:

```
addEventListener(type, listener[, useCapture]);
```

In the preceding syntax:

- `type`: Specifies a string representing the event type.
- `listener`: Specifies the object that receives a notification when an event occurs.
- `useCapture`: Is a Boolean variable, specifying whether an event needs to be captured or not. It is an optional parameter.

If the event handler captures an event, every time when the event occurs on the element, the event handler will be called. You can also remove a handler by calling the `removeEventListener()` function. However, the parameters passed to this function must be identical to the parameters passed to the `addEventListener()` function.

Consider the following code to understand the functionality of event listeners:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
```

```

function OnClick () {
alert ("A click event has occurred on the Submit button.");
}
function AddEventHandler () {
var Button = document.getElementById ("Button");
if (Button.addEventListener) {
Button.addEventListener ("click", OnClick, false);
}
}
function RemoveEventHandler () {
var Button = document.getElementById ("Button");
if (Button.removeEventListener) {
Button.removeEventListener ("click", OnClick, false);
}
}
</SCRIPT>
</HEAD>
<BODY>
Click on the Submit button when the 'click' event has a listener and
when it does not.<BR />
<BUTTON onclick="AddEventHandler();">Add a 'click' event listener to
the Submit button</BUTTON>
<BUTTON onclick="RemoveEventHandler();">Remove the event
listener</BUTTON>
<BR /><BR />
<BUTTON ID="Button">Submit</BUTTON>
</BODY>
</HTML>

```

## Using the navigator Object

The `navigator` object is one of the top-level objects in the JavaScript object hierarchy. Unlike the `window` or `document` objects, the `navigator` object is an independent object with its own set of properties and methods. An independent object does not have any child objects in its hierarchy.

You can use the `navigator` object to access information about the current browser, such as the browser name, version, and the user platform.

The following table lists the properties of the `navigator` object.

<i>Properties</i>	<i>Description</i>	<i>Syntax</i>
<code>appName</code>	<i>Specifies the</i>	<code>navigator.appName</code>



	<i>name of the Web browser.</i>	
<i>appVersion</i>	<i>Specifies information about the Web browser version, browser platform, and the country for which the Web browser is released.</i>	<i>navigator.appVersion</i>
<i>appName</i>	<i>Specifies the internal code name of the Web browser. For example, the internal code name returned for both, Microsoft Internet Explorer and Netscape browsers, is Mozilla.</i>	<i>navigator.appName</i>
<i>userAgent</i>	<i>Sends a string containing information, such as browser version, code name, and platform from the client to the server. The server uses this string to identify the client.</i>	<i>navigator.userAgent</i>
<i>platform</i>	<i>Specifies the operating system of the client machine.</i>	<i>navigator.platform</i>

*The Properties of the navigator Object*

The following table lists the methods associated with the `navigator` object.

<i><b>Methods</b></i>	<i><b>Description</b></i>	<i><b>Syntax</b></i>
<code>javaEnabled()</code>	<i>Returns a Boolean value that specifies whether JavaScript is enabled or disabled in the Web browser. For example, the value <code>True</code> indicates that JavaScript is enabled in the current Web browser.</i>	<code>navigator.javaEnabled()</code>
<code>taintEnabled()</code>	<i>Returns a Boolean value that specifies whether the security feature of the current Web browser is enabled or disabled. By default, the value returned is <code>False</code>.</i>	<code>navigator.taintEnabled()</code>

*The Methods of the navigator Object*

Consider the following code to use the `navigator` object:

```
<!DOCTYPE HTML>
<HTML>
<BODY>
<SCRIPT type="text/javascript">
document.write(navigator.appName);
document.write(navigator.appVersion);
</SCRIPT>
</BODY>
</HTML>
```

In the preceding code, the `appName` and `appVersion` properties of the `navigator` object are used to display the name and version of the browser, respectively.

## Using the screen Object

The `screen` object enables you to access details of the user's screen, such as its width, height, and resolution.

The following table lists the most commonly used properties of the `screen` object.

<i><b>Properties</b></i>	<i><b>Description</b></i>	<i><b>Syntax</b></i>
<i>height</i>	<i>Returns the total height of the screen.</i>	<i>screen.height</i>
<i>pixelDepth</i>	<i>Returns the color resolution of the screen.</i>	<i>screen.pixelDepth</i>
<i>width</i>	<i>Returns the width of the screen.</i>	<i>screen.width</i>
<i>availHeight</i>	<i>Returns the height of the screen without including the Windows taskbar.</i>	<i>screen.availHeight</i>
<i>availWidth</i>	<i>Returns the width of the screen without including the Windows taskbar.</i>	<i>screen.availWidth</i>
<i>colorDepth</i>	<i>Returns the bit depth of color palette that is used for displaying images.</i>	<i>screen.colorDepth</i>

*The Properties of the screen Object*

Consider the following code that illustrates the use of the `screen` object:

```
<!DOCTYPE HTML>

<HTML>

<BODY>

<SCRIPT type="text/javascript">
document.write("Height: " + screen.availHeight);
document.write("Width: " + screen.availWidth);
</SCRIPT>

</BODY>

</HTML>
```

The preceding code displays the height and width of the screen excluding the Window taskbar using the `availHeight` and `availWidth` properties.

## Using the history Object

The `history` object contains a list of all the pages that have been visited by the user.

The following table lists the properties of the `history` object.

<i>Properties</i>	<i>Description</i>	<i>Syntax</i>
<i>length</i>	<i>Is an integer value representing the number of links currently referenced by the <code>history</code> object in the current session.</i>	<i>history.length</i>

*The Properties of the history Object*

The following table lists the most commonly used methods of the `history` object.

<i>Methods</i>	<i>Description</i>	<i>Syntax</i>
<i>back()</i>	<i>Is used to move to the previous page.</i>	<i>history.back()</i>
<i>forward()</i>	<i>Is used to move to the next page.</i>	<i>history.forward()</i>
<i>go()</i>	<i>Is used to move back specific number of pages.</i>	<i>history.go(x)</i> <i>where x is the number of pages</i>

## *The Methods of the history Object*

Consider the following code to use the `history` object:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<SCRIPT type="text/javascript">
function show()
{
alert(history.length)
}
</SCRIPT>
</HEAD>
<BODY>
<INPUT type="button" value="Show History" onclick="show()" />
</BODY>
</HTML>
```

The preceding code displays the number of links referenced in the current browser session when a user clicks the button.

## Using the location Object

The `location` object contains information about the current URL. It can also be used to enable navigation to different URLs on the Internet. The most frequently used property of the `location` object is the `HREF` property. The `HREF` property specifies the exact URL of the current document.

The properties of the `location` object consist of various pieces of information that make up a complete URL. The following syntax is used for specifying the complete URL:

```
protocol://hostname:port/pathname/search#hash
```

The preceding syntax consists of the following attributes:

- `protocol`: Specifies the protocol, such as HTTP and FTP, which is used to transmit data over the Internet.
- `hostname`: Specifies the hostname of the URL, such as `http://www.silvertech.com`.
- `port`: Specifies the port number of the URL. It is optional. If the port number is not specified for the http request, the browser automatically connects to port 80.
- `pathname`: Specifies the path in the URL, such as `http://www.silvertech.com/careers.html`.
- `search`: Specifies the search string which is any portion of the URL followed by a question mark, such as `http://www.silvertech.com/careers.html/search?programmer`. The question mark is used for embedding arguments in the URL.
- `#hash`: Specifies the internal hyperlink of a Web page, such as `http://www.silvertech.com/careers.html#position1`.

The following table lists the most commonly used properties of the `location` object.

<i><b>Properties</b></i>	<i><b>Description</b></i>	<i><b>Syntax</b></i>
<i>hash</i>	<i>Specifies or returns the internal link anchor name. An internal link follows # in the URL.</i>	<i>location.hash</i>
<i>host</i>	<i>Specifies or returns the hostname: port portion of the URL.</i>	<i>location.host</i>
<i>hostname</i>	<i>Specifies or returns the hostname.</i>	<i>location.hostname</i>
<i>href</i>	<i>Specifies or returns the partial or full path of a file or a website.</i>	<i>location.href</i>
<i>port</i>	<i>Specifies or returns the port number.</i>	<i>location.port</i>
<i>protocol</i>	<i>Specifies or returns the protocol.</i>	<i>location.protocol</i>

*The Properties of the location Object*

The following table lists the most commonly used methods of the `location` object.

<i><b>Methods</b></i>	<i><b>Description</b></i>	<i><b>Syntax</b></i>
<i>assign()</i>	<i>Is used to load a new document.</i>	<i>location.assign(URL) ;</i>
<i>reload()</i>	<i>Is used to reload the current document.</i>	<i>location.reload() ;</i>
<i>replace()</i>	<i>Is used to replace the current</i>	<i>location.replace(newURL) ;</i>

	<i>document with another document.</i>	
--	--	--

### *The Methods of the location Object*

Consider the following code snippet to open the home page of XYZ.com website upon the click of a button:

```
<FORM>
<INPUT type = "button" VALUE = "visit us" onClick = "location.href =
'http://www.xyz.com'">
</FORM>
```

In the preceding code snippet, the `HREF` property of the `location` object is set to the URL of the XYZ website. As a result, when the user clicks the button, the home page of the XYZ website is displayed.

Consider the following code snippet to retrieve information regarding the current URL:

```
document.write(location.protocol);
```

In the preceding code snippet, the protocol of the current URL is retrieved using the `protocol` property of the `location` object.

## Working with Form Objects

Consider the scenario of BookYourHotel.com website. To purchase a product, a customer has to fill the purchase order form. The purchase order form requires entering the customer details, such as the name, gender, and address. This information is entered using the form objects, such as the text box and radio buttons. To access and process the content of the form objects, you need to refer to the form object.

The form object is a browser object, which acts as a container for several other objects that collect information from a user. The information collected using objects, such as the command buttons, radio buttons, text boxes, combo boxes, and check boxes, is submitted to the server for processing. Therefore, the form object enables you to create interactive Web pages.

The following objects of JavaScript are commonly used while working with Web forms:

- `form`
- `button`
- `checkbox`
- `text`
- `textarea`
- `radio`
- `select`

## form

A form accepts user inputs. When a Web page containing multiple forms is viewed in a browser, the browser creates a `form` object for every form on the Web page. These forms can also be accessed using an array. For example, the Web browser can access the first form on the Web page using `document.forms[0]`, and the second form can be accessed as `document.forms[1]`.

The methods associated with the `form` object are:

- `submit()`: Submits a form to the server for processing. The following syntax is used for the `submit()` method:

```
form1.submit()
```

In the preceding syntax, `form1` is the name of the form that will be submitted to the server on the occurrence of an event, such as clicking a hyperlink.

- `reset()`: Resets all the fields of the given form. The following syntax is used for the `reset()` method:

```
form1.reset()
```

In the preceding syntax, `form1` is the name of the form. The `reset()` method clears all the information entered in various fields of the form.

The events associated with the `form` object are:

- `onsubmit`: Occurs when a user clicks a button or hyperlink to submit the form to the server for processing.
- `onreset`: Occurs when a user opts for resetting the fields of the form by clicking the reset button or hyperlink.

## button

The `button` object refers to the HTML button. It allows you to perform several tasks based on user actions. You can write functions that are executed when a user clicks a specific button.

The methods associated with the `button` object are:

- `click()`: Simulates clicking of a button. For example, `myButton.click()`.
- `focus()`: Sets focus on a button.
- `blur()`: Removes focus from the button.

The event attributes associated with the `button` object are:

- `onclick`: Occurs when a user clicks a button.
- `onmousedown`: Occurs when a mouse button is pressed.
- `onmouseup`: Occurs when the mouse button is released.



## checkbox

The `checkbox` object refers to the HTML check box.

The methods associated with the `checkbox` object are:

- `click()`
- `focus()`
- `blur()`

The `checkbox` object can be referred inside a form using the following syntax:

```
document.form1.checkbox1
```

In the preceding syntax, `form1` is the name of the form and `checkbox` is the name or ID of the check box.

Consider the following code snippet to refer the `checkbox` object inside a form and perform validation on it:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<STYLE>
body{
background-color:pink;
}
</STYLE>
<SCRIPT>
function validate()
{
if(document.f1.check1.checked==false)
{
alert("Please select the check box");
return false
}
else
return true
}
</SCRIPT>
</HEAD>
<BODY>

<FORM name="f1" method="get" action="thanks.html" onsubmit="return
validate();" >

<INPUT type="checkbox" ID="check1" />Do you want to accept?

<INPUT type="submit" name="submit" value="Submit"></INPUT>
```

```
<INPUT type="reset" name="reset" value="Reset"></INPUT>
</FORM>
</BODY>
</HTML>
```

The preceding code snippet refers to the `checkbox` object inside a form and validates it before the user submits a form. It refers to the `checkbox` object using the `document.f1.check1` syntax. It checks whether the user has selected the check box or not by using the `document.f1.check1.checked==false` condition. If the user has not selected the check box, the `document.f1.check1.checked==false` condition evaluates to `false` and the message, **Please select the check box** is displayed.

## text

The `text` object refers to the HTML text box.

The methods associated with the `text` object are:

- `blur()`: Removes focus from a text field.
- `focus()`: Sets focus on a text field.
- `select()`: Selects a text field.

The event attributes associated with the `text` object are:

- `onfocus`: Fired when an element receives focus.
- `onchange`: Fired after the occurrence of the `blur` event, when the value of the text object is modified.
- `Onselect`: Fired when the user selects a part of the text within the text field.

The `text` object can be referred inside a form using the following syntax:

```
document.form1.text1
```

In the preceding syntax, `form1` is the name of the form and `text1` is the name or ID of the text box.

## textarea

The `textarea` object refers to the HTML textarea.

The `textarea` object has similar methods and event attributes as that of the `text` object.

## radio

The `radio` object refers to an HTML radio button.

The `radio` object can be referred inside a form using the following syntax:

```
document.form1.radio1
```

In the preceding syntax, `form1`, is the name of the form and `radio1` is the name or ID of the radio button.

Consider the following code snippet to refer the `radio` object inside a form and perform validation on it:

```
<!DOCTYPE HTML>

<HTML>

<HEAD>

<STYLE>

body{
background-color:pink;
}

</STYLE>

<SCRIPT>

function validate()
{
if((document.f1.M.checked==false) && (document.f1.F.checked==false))
{
alert("Please select the gender");
return false
}
else
return true
}

</SCRIPT>

</HEAD>

<BODY>

<FORM name="f1" method="get" action="thanks.html" onsubmit="return
validate();" >

Gender:

<INPUT type="radio" name="gender" ID="M" />Male
<INPUT type="radio" name="gender" ID="F" />Female
<INPUT type="submit" name="submit" value="Submit" align="center">
</INPUT>

<INPUT type="reset" name="reset" value="Reset"></INPUT>

</FORM>

</BODY>

</HTML>
```

The preceding code snippet refers to the `radio` object inside a form and validates it before the user submits the form. It checks whether the radio buttons, **Male** or **Female**, for indicating gender have been selected or not. If the user has not indicated gender, the `checked` property of the radio button returns `false` and **Please select the gender** message is displayed.

## select

The `select` object refers to the HTML drop-down list.

The methods associated with the `select` object are:

- `blur()`
- `focus()`

The events attributes associated with the `select` object are:

- `onchange`
- `onfocus`
- `onblur`

The `select` object can be referred inside a form using the following syntax:

```
document.form1.select1
```

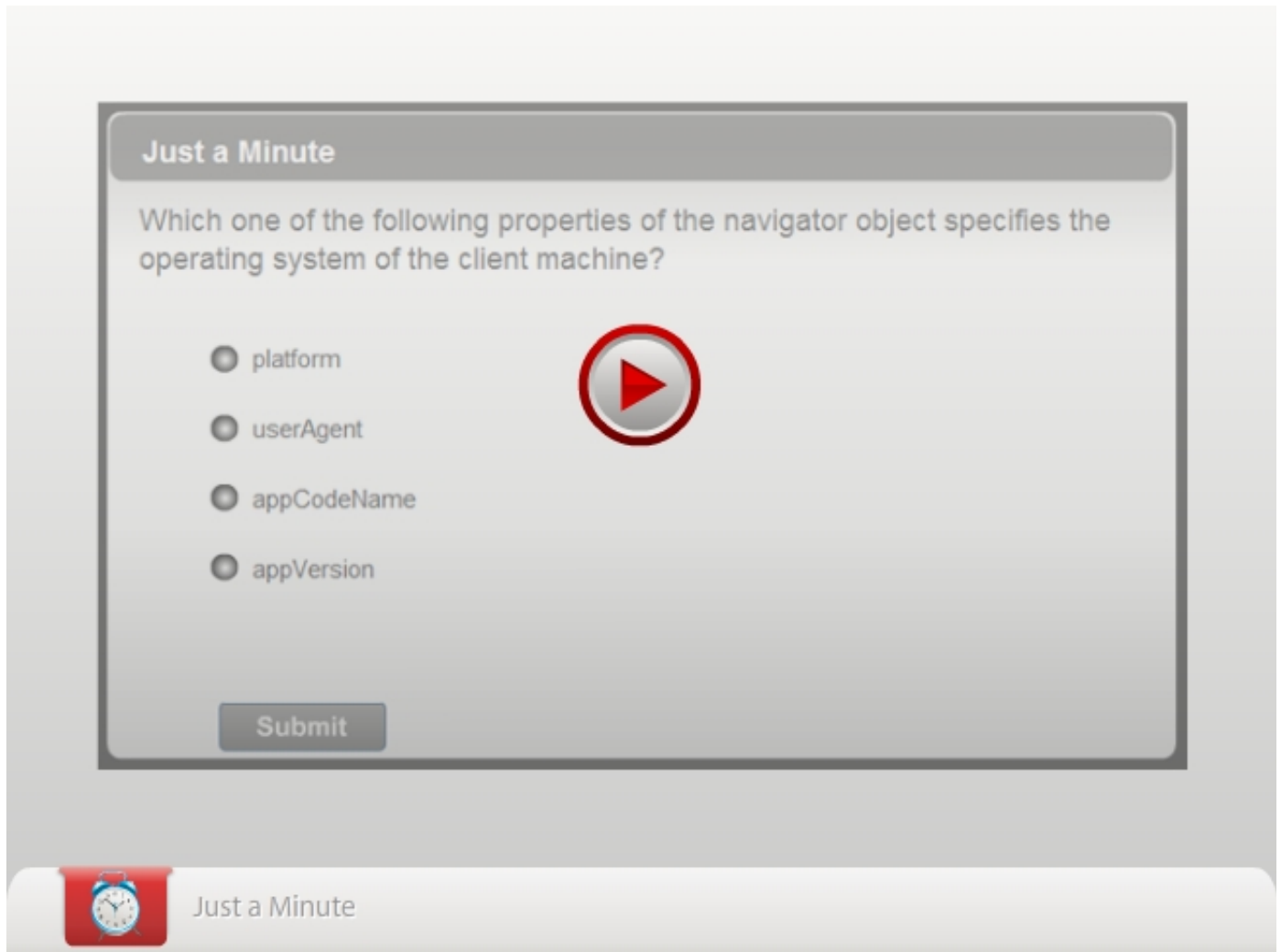
In the preceding syntax, `form1` is the name of the form and `select1` is the name or ID of the `select` object.

Consider the following code snippet to refer the `select` object inside a form and perform validation on it:

```
<!DOCTYPE HTML>
<HTML>
<HEAD>
<SCRIPT>
function validate()
{
if(document.f1.opt.value==0)
{
alert("Please select the country");
return false
}
else
return true
}
</SCRIPT>
</HEAD>
<BODY>
<FORM name="f1" method="get" action="thanks.html" onSubmit="return
validate();" >
Country:
<SELECT ID="opt" style="width:220">
```

```
<OPTION value=0>Select any country</OPTION>
<OPTION value="1">India</OPTION>
<OPTION value="2">USA</OPTION>
<OPTION value="3">UK</OPTION>
</SELECT>
<INPUT type="submit" name="submit" value="Submit">
<INPUT type="reset" name="reset" value="Reset">
</FORM>
</BODY>
</HTML>
```

The preceding code snippet refers to the `select` object inside a form and validates it before the user submits a form. It refers to the `select` object using the `document.f1.opt` statement. It checks whether a user has indicated his/her country or not using the `document.f1.opt.value==0` condition. If the user has not indicated his/her country, the **Please select the country** message is displayed.



## Activity 5.2: Manipulating the Components of a Web Page

### Summary

In this chapter, you learned that:

- To create a form on a Web page, you need to use the `<FORM>` tag.
- The `<FORM>` tag supports the following attributes:
  - `name`
  - `ID`
  - `action`
  - `method`
  - `autocomplete`
  - `novalidate`
  - `target`
- The fields can be added to a form by using the following tags:
  - `<INPUT>`
  - `<SELECT>`
  - `<LABEL>`
  - `<FIELDSET>`
  - `<TEXTAREA>`
  - `<DATAList>`
  - `<KEYGEN>`
  - `<OUTPUT>`
  - `<BUTTON>`
- JavaScript defines the following browser objects on a Web page:
  - `window`
  - `document`
  - `navigator`
  - `screen`
  - `history`
  - `location`
- Browser objects represent the browser environment and provide properties and methods for its access and manipulation.
- The form object is a browser object, which acts as a container for several other objects that collect information from a user.
- The following objects of JavaScript are commonly used while working with Web forms:
  - `form`
  - `button`
  - `checkbox`
  - `text`
  - `textarea`
  - `radio`

- `select`

## Reference Reading

### Designing an HTML Form

<b><i>Reference Reading: Books</i></b>	<b><i>Reference Reading: URLs</i></b>
<i>The Definitive Guide to HTML5 By Adam Freeman</i>	<a href="http://www.html5rocks.com/en/tutorials/forms/html5forms/">http://www.html5rocks.com/en/tutorials/forms/html5forms/</a> <a href="http://www.ohio.edu/technology/training/upload/html-advanced-reference-guide.pdf">http://www.ohio.edu/technology/training/upload/html-advanced-reference-guide.pdf</a>

### Manipulating the Components of a Web Page

<b><i>Reference Reading: Books</i></b>	<b><i>Reference Reading: URLs</i></b>
<i>The Definitive Guide to HTML5 By Adam Freeman</i>	<a href="http://www.w3schools.com/js/js_ex_browser.asp">http://www.w3schools.com/js/js_ex_browser.asp</a>