Form & Event Handling

Unit - III

HTML Elements

- Web pages would be very boring if you could not interact with or obtain information from the user, such as text, numbers, or dates. Luckily, with JavaScript this is possible.

- We can use this information within the web page, or it can be posted to the web server where we can manipulate it and store it in a database.

- This chapter concentrates on using the information within the web browser, which is called *client-side processing.*

HTML Elements

- Various interface elements like Buttons, textbox, textarea, radio button, checkboxes etc are the means by which a user interact with applications.

- We can include many of these types of elements in our webpage.
- When we have such an element inside our page, we can then tie code to its events.

- The elements discussed in this chapter are the common elements made available by HTML, and not ActiveX elements, Java Applets, or plug-ins.

HTML Forms

- Forms provide you with a way of grouping together HTML interaction elements with a common purpose.

- It's possible to have a number of separate forms in a single page but only one of the forms on a page can be submitted to the server at one time.

- To create a form, use the <form> and </form> tags to declare where it starts and where it ends. The

HTML Forms

- <form/> element has a number of attributes, such as:
    - the action attribute, which determines where the form is submitted to;
    - the method attribute, which determines how the information is submitted; and
    - the target attribute, which determines the frame to which the response to the form is loaded.
- Client-side scripting where programmer has no intention of submitting information to a server, these attributes are not necessary.

HTML Forms

- The *name* attribute in the *<form/>* element is used to make reference to the form.
- So, to create a blank form, the tags required would look something like this:

```
<form name="myForm">
</form>
```

- These tags create a Form object, which we can use to access the form. This object can be accessed in two ways.

HTML Forms

- First, the object can be accessed directly by using its name — in this case *document.myForm*.

- Alternatively, we can access the object through the document object’s forms collection property.
- For example, if it’s the first Form in the page, we can reference it using *document.forms[0]*.

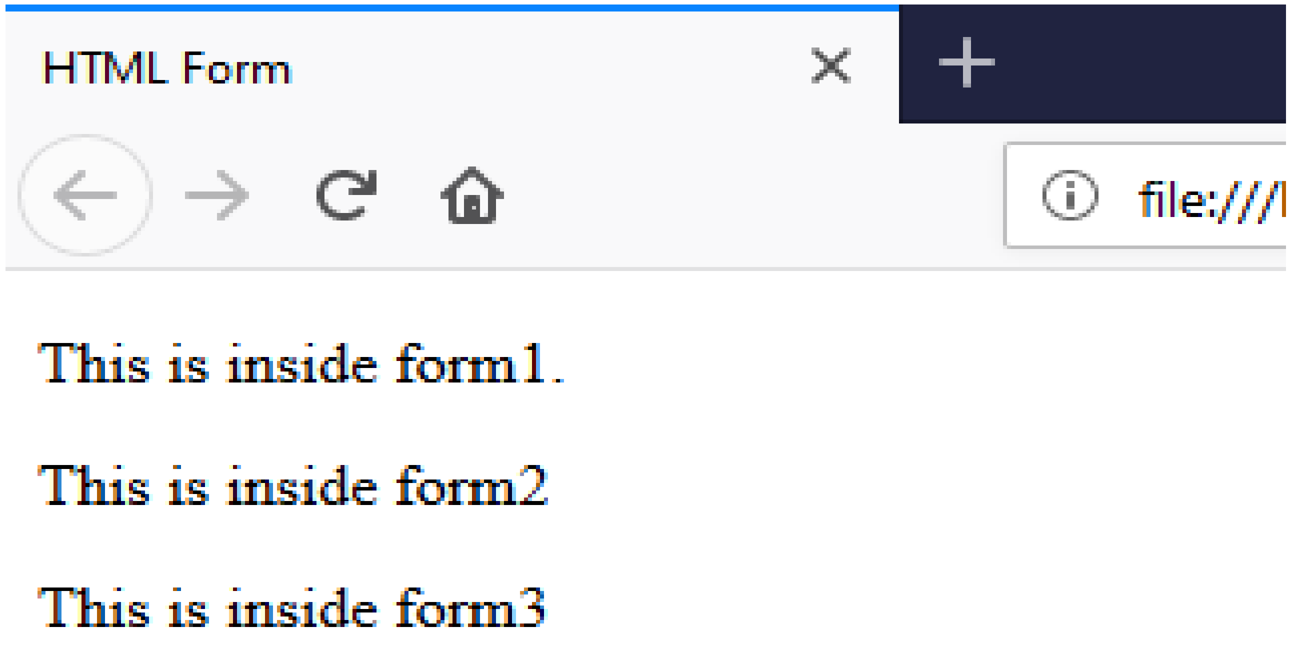
HTML Forms

- In this example we have a page with three forms on it. Using the forms collection, we access each Form object in turn and show the value of its name property in a message box.

```
<html>
  <head>
  <title>HTML Form</title>
  <script>
        function window_onload()
        {
                var numberForms = document.forms.length;
                var formIndex;
                for (formIndex = 0; formIndex < numberForms; formIndex++)
                {
                        alert(document.forms[formIndex].name);
                }
```

HTML Forms

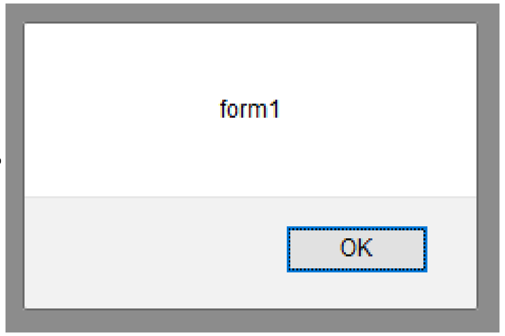
```
        }
  </script>
  </head>
  <body onload=”window_onload()“>
        <form name=“form1”>
               <p>This is iname=“form1nside
        </form>
        <form name=“form2”>
        <p>This is inside form2</p>
        </form>
        <form name=“form3”>
        <p>This is inside form3</p>
        </form>
  </body>
</html>
```

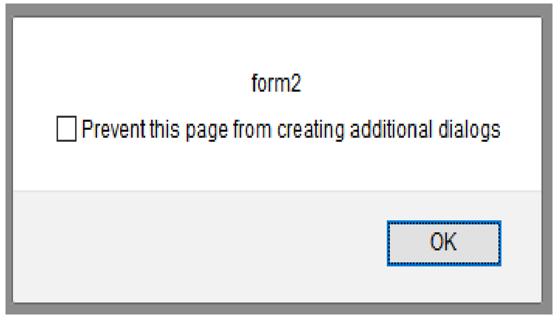


This is inside form1.

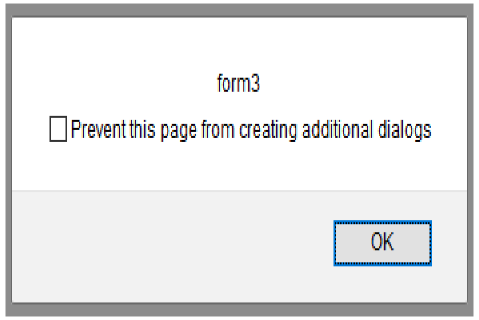
This is inside form2

This is inside form3


form1


form2

☐ Prevent this page from creating additional dialogs


form3

☐ Prevent this page from creating additional dialogs

Properties & Methods of Forms

- The HTML controls commonly found in forms also have corresponding objects.
- One way to access these is through the *elements* property of the Form object, another collection.
- The elements collection contains all the objects corresponding to the HTML interaction elements within the form.

Properties & Methods of Forms

- Being a collection, the elements property of the Form object has the *length* property, which tells you how many elements are in the form.

- The Form object also has the length property, which also gives you the number of elements in the form.

- E.g.

  document.myForm.length or
  document.myForm.elements.length

Properties & Methods of Forms

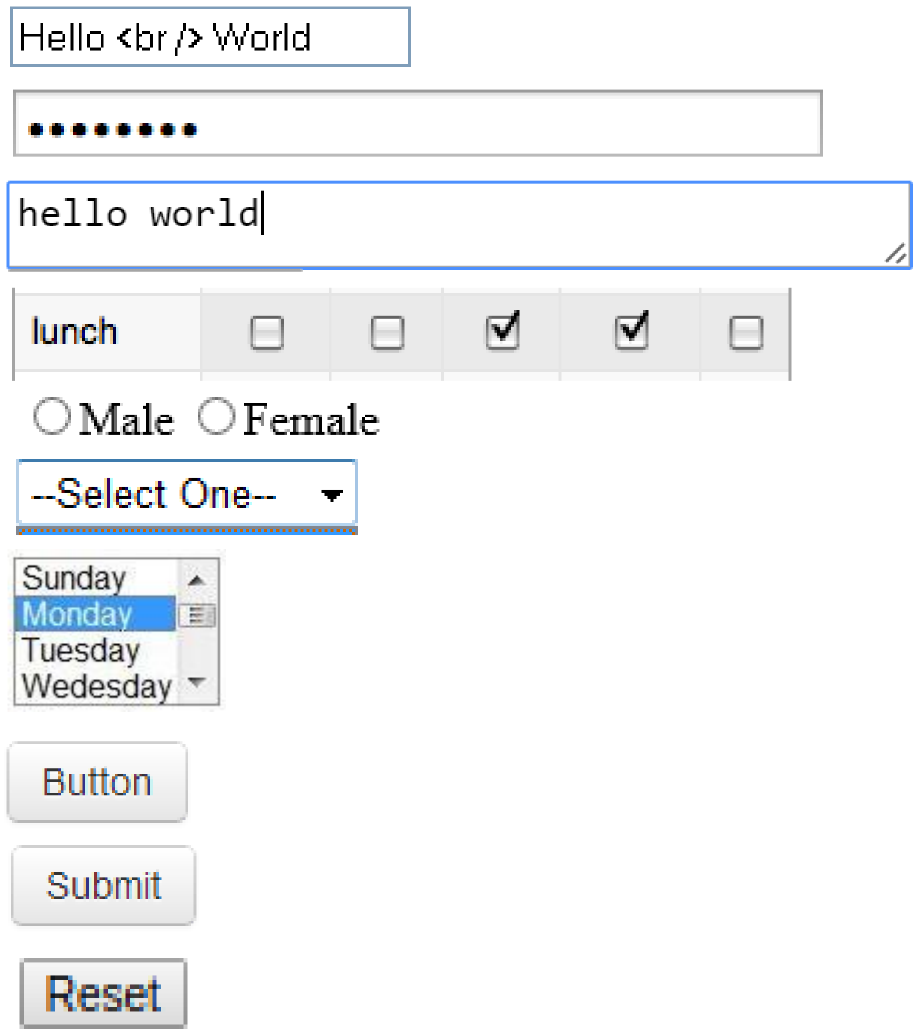
- When data is submitted from a form to a server, normally Submit button is used.
- However, the Form object also has the submit() method, which does nearly the same thing.
- The *submit()* method submits the form, but it does not fire the submit event of the Form object;
- thus, the *onsubmit* event handler is not called when submitting the form with submit().

Properties & Methods of Forms

- In addition to there being a Reset button, the Form object has the *reset()* method, which clears the form, or restores default values if these exist.

HTML Elements in Forms

- About 10 elements are commonly found within <form/> elements.
  - Text Box
  - Password
  - Text Area
  - Check Box
  - Radio Button
  - Drop Down List
  - List Box
  - Standard Button
  - Submit Button
  - Reset Button



Hello <br /> World

•••••••••

hello world

lunch ☐ ☐ ☑ ☑ ☐

○ Male ○ Female

--Select One--

Sunday
Monday
Tuesday
Wedesday

Button

Submit

Reset

HTML Elements in Forms

- Common Properties:
  - *name:* the value of this property can be used to refer that particular element in your script. Also, if you are sending the information in the form to a server, the element's name property is sent along with any value of the form element, so that the server knows what the value relates to.
  - *value:* which returns the value of the element. Also, setting the value of the value property enables you to put text inside the text box.
  - *type:* Sometimes it's useful to know what type of element you're dealing with, particularly where you're looping through the elements in a form using the elements collection property. This information can be retrieved by means of the type property, which each element's object has.

HTML Elements in Forms

- Common Methods:
  - *focus():* If an element is the center of the *focus*, any key presses made by the user will be passed directly to that element. For example, if a text box has focus, pressing keys will enter values into the text box. Also, if a button has the focus, pressing the Enter key will cause the button's *onclick* event handler code to fire, just as if a user had clicked the button with his mouse.
  - *blur(): Blur, which perhaps could be better called "lost focus," is the opposite of focus. If you want to remove* a form element from being the focus of the user's attention, you can use the blur() method. When used with a form element, the blur() method usually results in the focus shifting to the page containing the form.

HTML Elements in Forms

- Common Methods:
  - In addition to the focus() and blur() methods, all the form element's objects have the *onfocus* and *onblur* event handlers. These are fired when an element gets or loses the focus, respectively, due to user action or the focus() and blur() methods.

Standard Button Element

- The HTML element to create a button is <input/>.
- For example, to create a button called myButton, which has the words "Click Me" on its face, the <input/> element would need to be as follows:

```
<input type="button" name="myButton" value="Click Me" />
```

- The type attribute is set to button, and the value attribute is set to the text you want to appear on theface of the button.

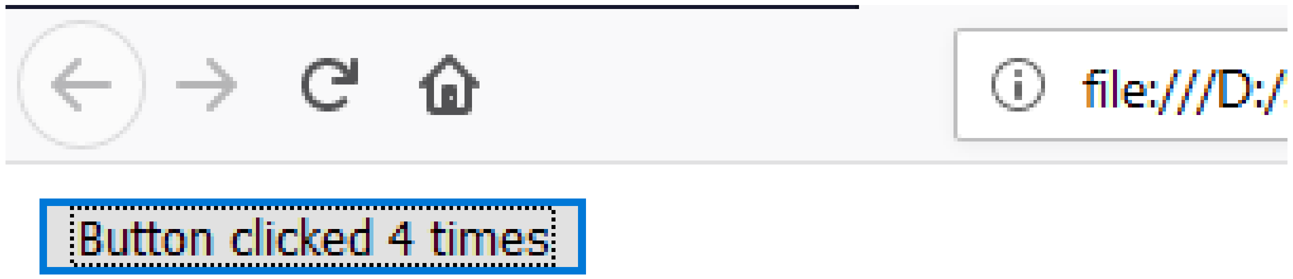
Standard Button Element

- What the button is really all about is the click event.
- All you need to do is define a function that you want to have executed when the button is clicked (say, button_onclick()) and then add the onclick event handler as an attribute of the <input/> element as follows:

```
<input type=”button” onclick=”button_onclick()” />
```

Standard Button Element

```
<html>
  <head>
  <title>Button Example</title>
  <script>
        var numberOfClicks = 0;
        function myButton_onclick()
        {       numberOfClicks++;
                document.form1.myButton.value = "Button clicked " +
                numberOfClicks + " times";
        }
  </script></head>
  <body><form name="form1">
                <input type="button" name="myButton" value="Button clicked 0 times"
  onclick="myButton_onclick()" />
  </form></body>
</html>
```



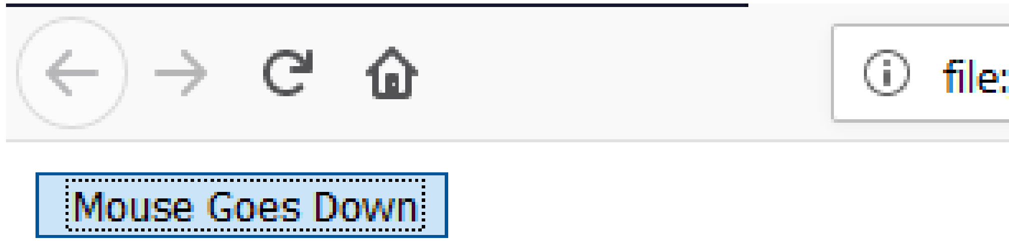
Standard Button Element

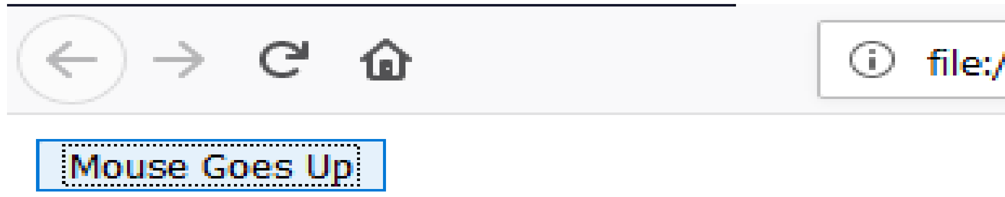
- Two less commonly used events supported by the Button object are the *mousedown* and *mouseup* events.

- *mouseup* and *mousedown* are triggered only when the mouse pointer is actually over the element in question.

Standard Button Element

```
<html>
  <head>
  <title>Mouseup & Mousedown</title>
  <script>
        function myButton_onmouseup()
        { document.form1.myButton.value = "Mouse Goes Up"
        }
        function myButton_onmousedown()
        { document.form1.myButton.value = "Mouse Goes Down"
        }
  </script></head>
  <body><form name="form1">
                <input type="button" name="myButton" value="Mouse Goes Up"
                onmouseup="myButton_onmouseup()"
        onmousedown="myButton_onmousedown()" />
  </form></body>
</html>
```





Standard Button Element

- For example, if you click and hold down the mouse button over your button, then move the mouse away from the button before releasing the mouse button, you'll find that the mouseup event does not fire and the text on the button's face does not change.
- In this instance it would be the document object's onmouseup event handler code that would fire, if you'd connected any code to it.

Submit & Reset Button Element

- Next additional types of button are Submit & Reset buttons.
- They are defined just as you do a standard button, except that the type attribute of the <input> tag is set to submit or reset rather than to button.
- E.g.:

```
<input type=”submit” value=”Submit” name=”submit1” />
<input type=”reset” value=”Reset” name=”reset1” />
```

Submit & Reset Button Element

- When the Submit button is clicked, the form data from the form that the button is inside gets sent to the server automatically, without the need for any script.
- When the Reset button is clicked, all the elements in a form are cleared and returned to their default values (the values they had when the page was first loaded).
- The Submit and Reset buttons have corresponding objects called Submit and Reset, which have exactly the same properties, methods, and events as a standard Button object.

Text Box Element

- The standard text box element enables users to enter a single line of text. This information can then be used in JavaScript code or submitted to a server for server-side processing.

- A text box is created by means of the <input/> element, much as the button is, but with the type attribute set to text. Again, you can choose not to include the value attribute, but if you do include it this value will appear inside the text box when the page is loaded.

Text Box Element

- It has two additional attributes, size and maxlength.
- The size attribute determines how many characters wide the text box is, and maxlength determines the maximum number of characters the user can enter in the box.
- Both attributes are optional and use defaults determined by the browser.

Text Box Element

- For example, to create a text box 10 characters wide, with a maximum character length of 15, and initially containing the words Hello World, your <input/> element would be as follows:

<input type="text" name="myTextBox" size="10" maxlength="15" value="Hello World" />

Text Box Element

- The Text object also has the *select()* method, which selects or highlights all the text inside the text box.

- This may be used if the user has entered an invalid value, and you can set the focus to the text box and select the text inside it. This then puts the user's cursor in the right place to correct the data and makes it very clear to the user where the invalid data is.

Text Box Element

- The Text object has the *onchange*, *onselect*, *onkeydown*, *onkeypress*, and *onkeyup* event handlers.
- The *onselect* event fires when the user selects some text in the text box.
- More useful is the *onchange* event, which fires when the element loses focus if (and only if) the value inside the text box is different from the value it had when it got the focus. This enables you to do things like validity checks that occur only if something has changed.

Text Box Element

- You can use the *readonly* attribute of the <input/> element or the *readOnly* property of the Text object to prevent the contents from being changed.

- *E.g.:*

```
<input type=”text” name=”txtReadonly” value=”Look but don’t change” onfocus=”document.form1.txtReadonly.blur()” readonly=”readonly”>
```

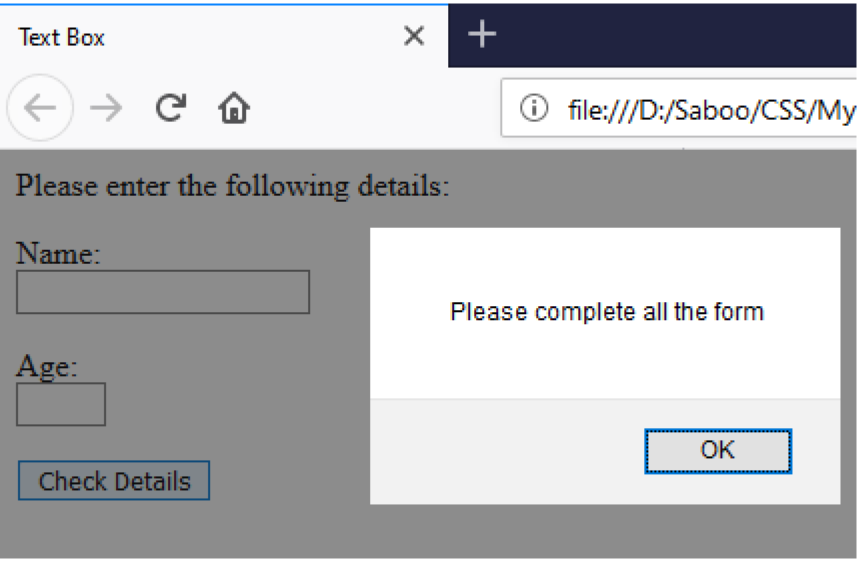
Text Box Element

```
<html>
  <head><title>Text Box</title>
  <script>
          function btnCheckForm_onclick()
          {
                    var myForm = document.form1;
                    if (myForm.txtAge.value == "" || myForm.txtName.value == "")
                    {         alert("Please complete all the form");
                              if (myForm.txtName.value == "")
                              { myForm.txtName.focus();
                              }
                              else
                              { myForm.txtAge.focus();
                              }
                    }
                    else
                    { alert("Thanks for completing the form " + myForm.txtName.value);
                    }
          }
```

Text Box Element

```
        function txtAge_onblur()
        {var txtAge = document.form1.txtAge;
                if (isNaN(txtAge.value) == true)
                {alert("Please enter a valid age");
                        txtAge.focus();
                        txtAge.select();
                }
        }
        function txtName_onchange()
        {alert("Hi " + document.form1.txtName.value);
        }
        </script></head>
  <body><form name="form1">
        Please enter the following details:
        <p>Name:<br>
        <input type="text" name="txtName" onchange="txtName_onchange()" /></p>
        <p>Age:<br>
        <input type="text" name="txtAge" onblur="txtAge_onblur()" size="3" maxlength="3" /></p>
        <p><input type="button" value="Check Details" name="btnCheckForm" onclick="btnCheckForm_onclick()"/></p>
  </form></body>
</html>
```

Text Box Element


Text Box

file:///D:/Saboo/CSS/My

Please enter the following details:

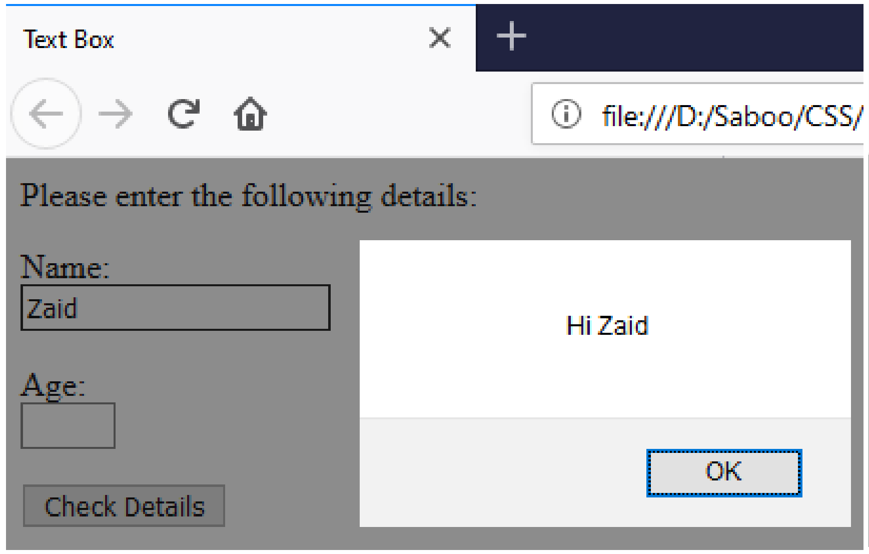
Name:

Age:

Check Details

Please complete all the form

OK


Text Box

file:///D:/Saboo/CSS/

Please enter the following details:

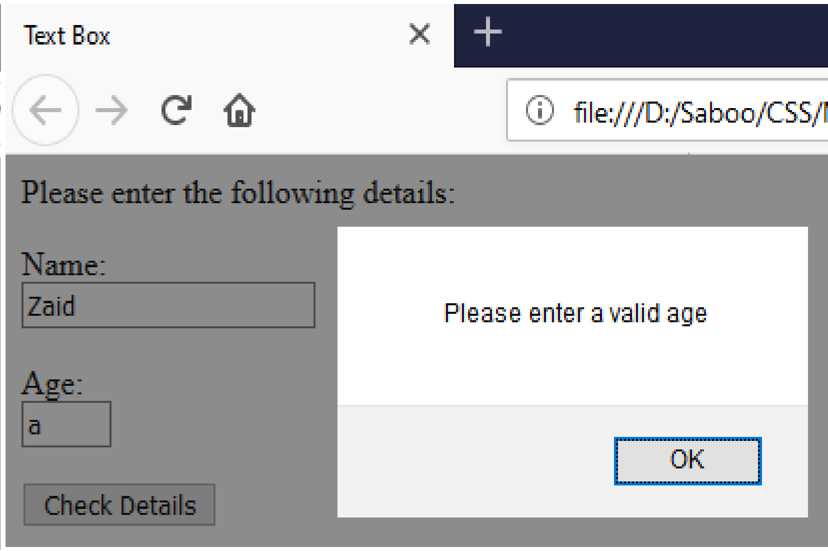
Name:
Zaid

Age:

Check Details

Hi Zaid

OK


Text Box

file:///D:/Saboo/CSS/

Please enter the following details:

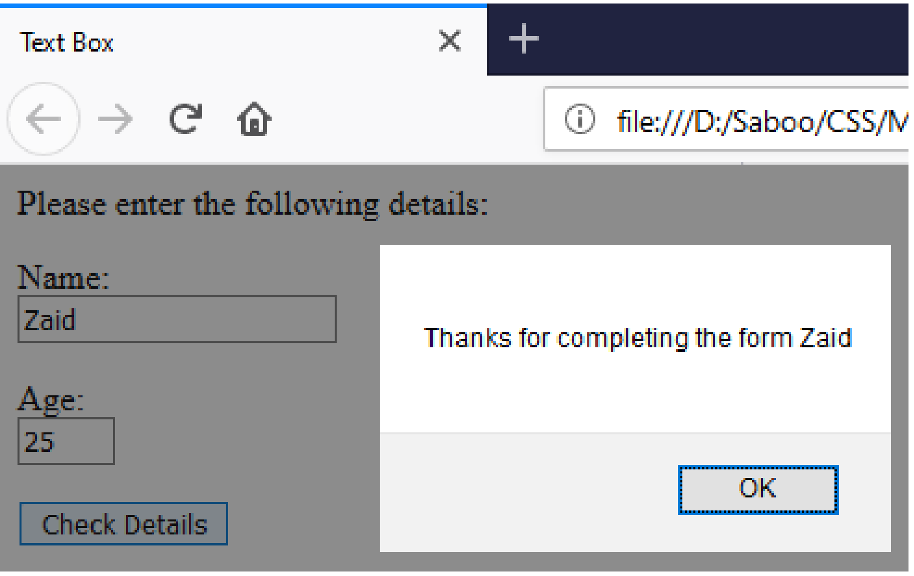
Name:
Zaid

Age:
a

Check Details

Please enter a valid age

OK


Text Box

file:///D:/Saboo/CSS/M

Please enter the following details:

Name:
Zaid

Age:
25

Check Details

Thanks for completing the form Zaid

OK

Password TextBox Element

- The only real purpose of the password box is to enable users to type in a password on a page and to have the password characters hidden, so that no one can look over the user’s shoulder and discover his or her password.

- However, this protection is visual only.

- When sent to the server, the text in the password is sent as plain text — there is no encryption or any attempt at hiding the text (unless the page is served over a secure connection from the server).

Password Text Box Element

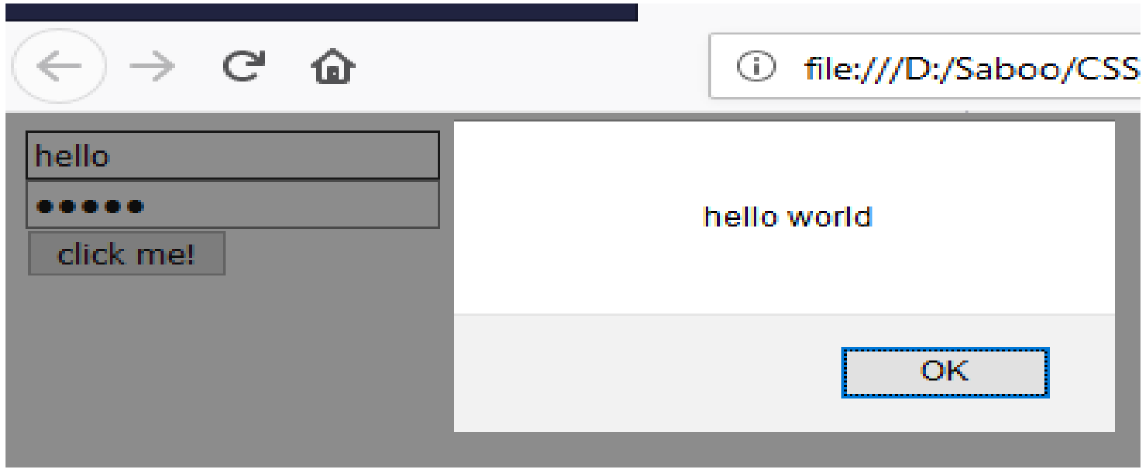
- Defining a password box is identical to defining a text box, except that the type attribute is password.

```
<input name=”password1” type=”password” />
```

- This form element creates an associated Password object, which is identical to the Text object in its properties, methods, and events.

Password Text Box Element

```
<html>
  <head><title>HTML Form</title><script>
        function calling()
        {       alert(text1.value+" "+text2.value);
        }
        function disp()
        {       alert(form1.text1.value+" "+form1.text2.value);
        }
  </script></head>
  <body><form name="form1">
                <input type="text" name="text1"/><br>
                <input type="password" name="text2"/><br>
                <input type="button" value="click me!" name="button1" onclick="disp()"/>
  </form></body>
</html>
```



hello world

Text Area Box Element

- The <textarea/> element allows multi-line input of text. Other than this, it acts very much like the text box element.
- However, unlike the text box, the textarea element has its own tag, the <textarea> tag.
- It also has two additional attributes: cols and rows. The cols attribute defines how many characters wide the text area will be, and the rows attribute defines how many character rows there will be.
- You set the textinside the element by putting it between the start and closing tags, rather than by using the value attribute.

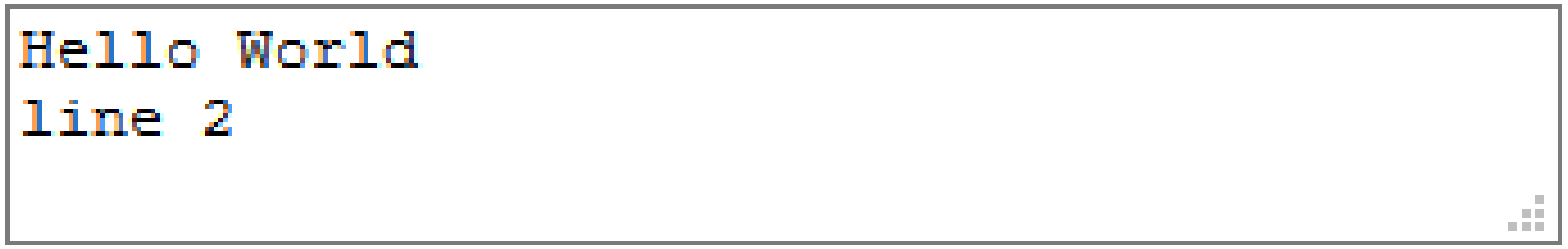
Text Area Box Element

- So if you want a <textarea/> element 40 characters wide by 20 rows deep with initial text Hello World on the first line and Line 2 on the second line, you define it as follows:

<textarea name="myTextArea" cols="40" rows="20">Hello World

Line 2

</textarea>



```
Hello World
line 2
```

Text Area Box Element

- The Textarea object created by the <textarea/> element has the same properties, methods, and events as the Text object you saw previously, except that the text area doesn’t have the maxlength attribute.

- Note that there is a value property even though the <textarea/> element does not have a value attribute. The value property simply returns the text between the <textarea> and </textarea> tags.

- The events supported by the Textarea object include the *onkeydown*, *onkeypress*, *onkeyup*, and *onchange* event handlers.

Check Box & Radio Button Element

- A check box enables the user to check and uncheck it.
- Radio buttons are basically a group of check boxes where only one can be checked at a time.
- Creating check boxes and radio buttons requires the <input/> element. Its *type* attribute is set to "*checkbox*" or "*radio*" to determine which box or button is created.
- To set a check box or a radio button to be checked when the page is loaded, you simply insert the attribute *checked* into the <input> tag and assign its value as *checked*.

Check Box & Radio Button Element

- E.g.
  <input type=“checkbox” name=“chkDVD” checked=“checked” value=“DVD” />

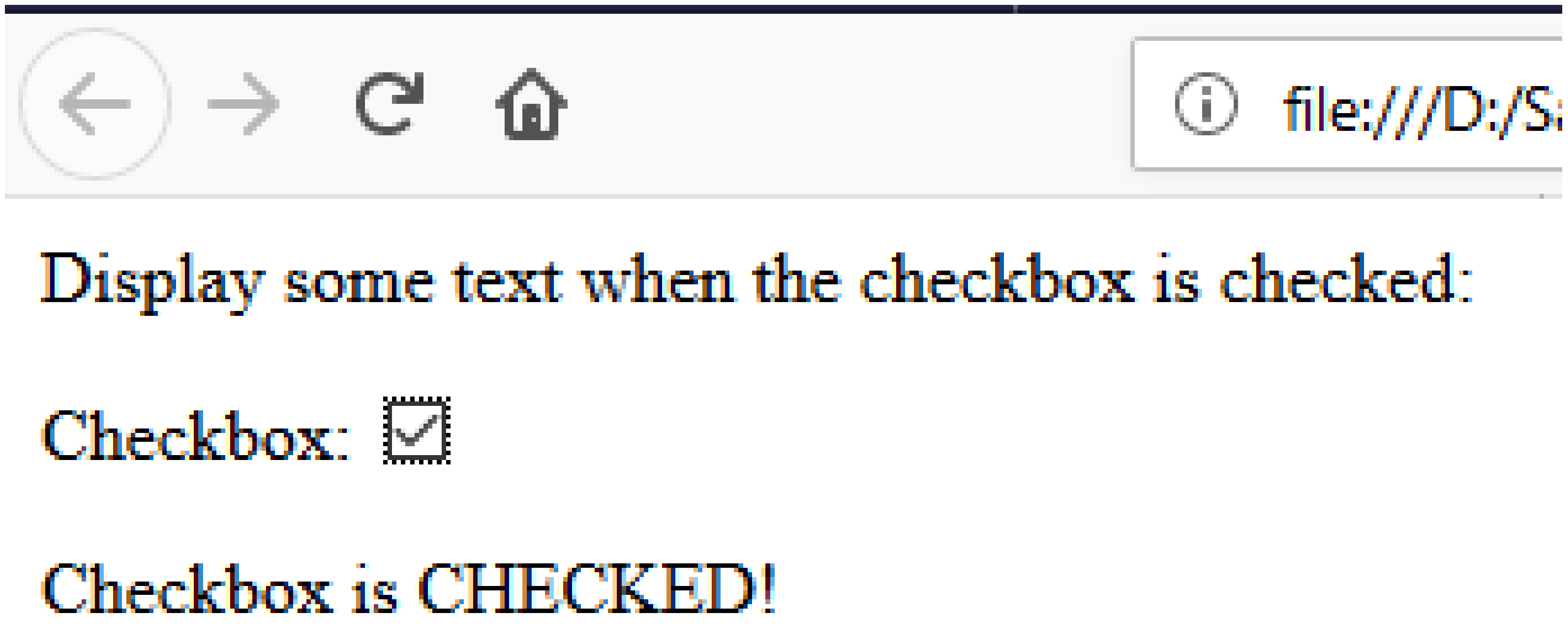
  <input type=“radio” name=“radCPUSpeed” checked=“checked” value=“1 GHz” />

- To create a group of radio buttons, you simply give each radio button the same name. This creates an array of radio buttons going by that name that you can access, as you would with any array, using its index.

Check Box & Radio Button Element

- Each check box has an associated Checkbox object, and each radio button in a group has a separate Radio object.
- For determining whether a user has actually checked or unchecked a check box, you need to use the checked property of the Checkbox object.
- This property returns true if the check box is currently checked and false if not.

Check Box & Radio Button Element



Display some text when the checkbox is checked:

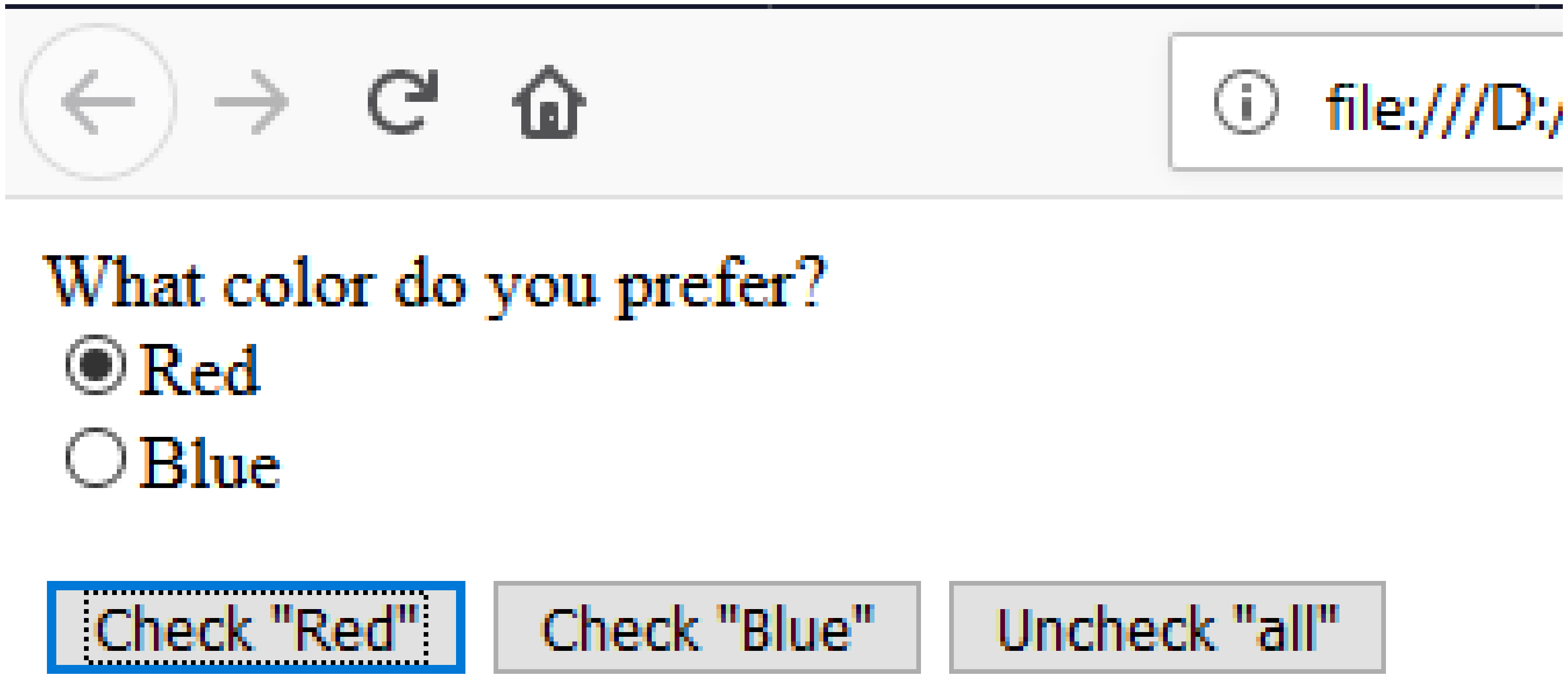
Checkbox: ✓

Checkbox is CHECKED!

Check Box & Radio Button Element

```
<html>
   <body>
            <p>Display some text when the checkbox is checked:</p>
            Checkbox: <input type="checkbox" id="myCheck"  onclick="myFunction()">
            <p id="text" style="display:none">Checkbox is CHECKED!</p>
            <script>
                     function myFunction() {
                       var checkBox = document.getElementById("myCheck");
                       var text = document.getElementById("text");
                       if (checkBox.checked == true){
                                  text.style.display = "block";
                       } else {
                                   text.style.display = "none";
                                 }
                     }
            </script>
   </body>
</html>
```

Check Box & Radio Button Element



file:///D:/

What color do you prefer?

● Red

○ Blue

Check Box & Radio Button Element

```
<html>
  <body><form>
              What color do you prefer?<br>
              <input type="radio" name="colors" id="red">Red<br>
              <input type="radio" name="colors" id="blue">Blue
            </form>
            <button onclick="checkred()">Check "Red"</button>
            <button onclick="checkblue()">Check "Blue"</button>
            <button onclick="uncheck()">Uncheck "all"</button>
            <script>
            function checkred() {
              document.getElementById("red").checked = true;     }
            function checkblue() {
              document.getElementById("blue").checked = true;   }
            function uncheck() {
              document.getElementById("red").checked = false;
              document.getElementById("blue").checked = false;  }
  </script></body>
</html>
```

Selection Boxes

- Although they look quite different, the drop-down list and the list boxes are actually both elements created with the <select> tag.

- The select element has one or more options in a list that you can select from; each of these options is defined by means of one or more <option/> elements inside the opening and closing <select> tags.

- The size attribute of the <select/> element is used to specify how many of the options are visible to the user.

Selection Boxes

- For example, to create a list box five rows deep and populate it with seven options, your HTML would look like this:

```
<select name="theDay" size="5">
        <option value="0" selected="selected">Monday</option>
        <option value="1">Tuesday</option>
        <option value="2">Wednesday</option>
        <option value="3">Thursday</option>
        <option value="4">Friday</option>
        <option value="5">Saturday</option>
        <option value="6">Sunday</option>
</select>
```

Selection Boxes

- Notice that the <option/> element for Monday also contains the attribute selected; this will make this option selected by default when the page is loaded.

- The values of the options have been defined as numbers, but text would be equally valid.

- To make it a *drop-down list*, we just need to *change the size* attribute in the <select/> element *to 1*, and presto, it's a drop-down list.

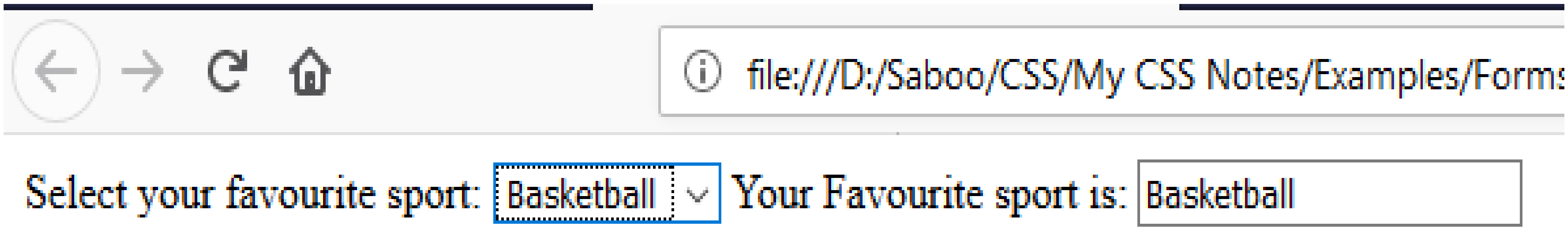
Selection Boxes

- If you want to let the user *choose more than one item* from a list at once, you simply need to *add the multiple attribute* to the <select/> definition.

Selection Boxes

```
<html>
   <script>
            function myfunction()
            {           var mylist=document.getElementById("myList");
                        document.getElementById("favourite").value=myList.options[myList.selectedIndex].text
            }</script>
   <body>           <form>
                        Select your favourite sport:
                        <select id="myList" onchange="myfunction()”>
                                    <option>Soccer</option>
                                    <option>Cricket</option>
                                    <option>Tennis</option>
                                    <option>Basketball</option>
                                    <option>Badminton</option>
                        </select>
                        Your Favourite sport is: <input type="text" id="favourite"                         size="20">
   </form></body>
</html>
```

Selection Boxes


file:///D:/Saboo/CSS/My CSS Notes/Examples/Forms

Select your favourite sport: Basketball Your Favourite sport is: Basketball

Mouse Events

Event	Description
click	Raised when the user clicks an HTML control.
dblclick	Raised when the user double-clicks an HTML control.
mousedown	Raised when the user presses a mouse button.
mousemove	Raised when the user moves the mouse pointer.
mouseout	Raised when the user moves the mouse pointer out from within an HTML control.
mouseover	Raised when the user moves the mouse pointer over an HTML control.
mouseup	Raised when the user releases the mouse button.

Keyboard Events

Event	Description
Keydown	Raised when the user presses a key on the keyboard.
Keypress	Raised when the user presses a key on the keyboard. This event will be raised continually until the user releases the key.
Keyup	Raised when the user releases a key that had been pressed.

Form Objects & elements

- The Form object represents a <form> elements in an HTML document.
- The elements property is an HTMLCollection that provides convenient access to all elements of the form.
- The *submit()* & *reset()* methods allow a form to be submitted or reset under program control.

Form Objects & elements

- Each form in a document is represented as an element of the *document.forms[]* array.
- The elements of a form (buttons, input fields, checkboxes, and so on) are collected in the arraylike object *Form.elements*.

Form Objects & elements

- Properties:
  - string **acceptCharset**: A list of one or more allowed character sets in which the form data may be encoded for submission.
  - string **action**: The URL to which the form should be submitted.
  - string **autocomplete**: The string “on” or “off”. If “on”, the browser can prefill form controls with saved values from a previous visit to the page.
  - string **method**: The HTTP method used to submit the form to the action URL. Either “get” or “post”.

Form Objects & elements

- Properties:
  - string **name**: The name of the form, as specified by the HTML name attribute. You can use the value of this property as a property name on the document object. The value of that document property will be this Form object.
  - boolean **noValidate**: true if the form is not to be validated before submission.
  - string **target**: The name of window or frame in which the document returned by form submission is to be displayed.

Form Objects & elements

- Methods:
  - boolean checkValidity(): In browsers that support form validation, this method checks the validity of each form control. It returns true if they are all valid. If any controls are not valid, it fires an invalid event on that control and then returns false.

  - void **reset()**: Reset all form elements to their default values.

  - void **Submit()**: Submit the form manually, without triggering a Submit event.

Changing Attribute value dynamically

- element.setAttribute():

  - Sets the value of an attribute on the specified element. If the attribute already exists, the value is updated; otherwise a new attribute is added with the specified name and value.

  - Syntax: *element.setAttribute(name, value);*

    where:
      name: Required. Specifies name of the attribute whose value is to be set
      value: Required. Specifies value to assign to the attribute.

Changing Attribute value dynamically

- element.getAttribute():

  - It is used to get the current value of an attribute.

  - Syntax: *element.getAttribute(name);*

    where:

    name:Required. Specifies name of the attribute whose value is to be fetched

Changing Attribute value dynamically

- Element.removeAttribute():

  - It is used to remove the specified attribute from an element.

  - Syntax: *element.removeAttribute(name);*

    where:
    name:Required. Specifies name of the attribute whose value is to be removed

Changing Attribute value dynamically

- Element.hasAttribute():

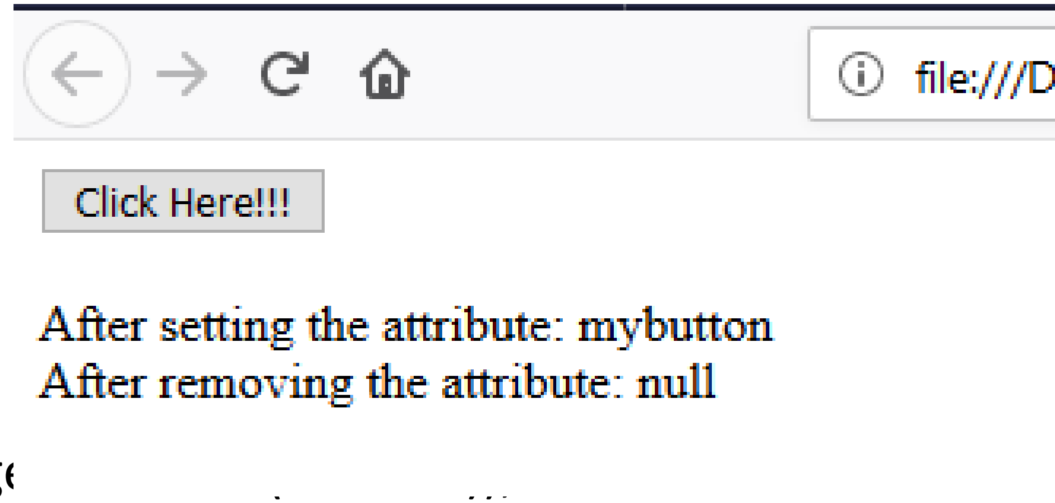
  - It returns true if the specifies attribute exists, otherwise it returns false.

  - Syntax: *element.hasAttribute(name);*

    where:
    name:Required. Specifies name of the attribute you want to check if exists.

Changing Attribute value dynamically

```
<html>
   <head><title>Changing attributes dynamically</title></head>
   <body><form name="form1">
            <input type="button" id="button1"><br>
            <script>
            var b=document.getElementById("button1");
            b.setAttribute("name","mybutton");
            b.setAttribute("value","Click Here!!!");
            document.write("<br>After setting the attribute: "+b.ge
            document.write("<br>Checking the attribute: "+b.has Attribute("name")); -True
            b.removeAttribute("name");
            document.write("<br>After removing the attribute: "+b.getAttribute("name"));
            document.write("<br>Checking the attribute: "+b.has Attribute("name")); -False
            </script></form>
   </body>
</html>
```



```
After setting the attribute: mybutton
After removing the attribute: null
```

Changing option list dynamically

- add():
  - This method is used to add an option to a drop down list.
  - By default new element is added at the end of the list.

  - Syntax: *selectObject.add(option, index)*

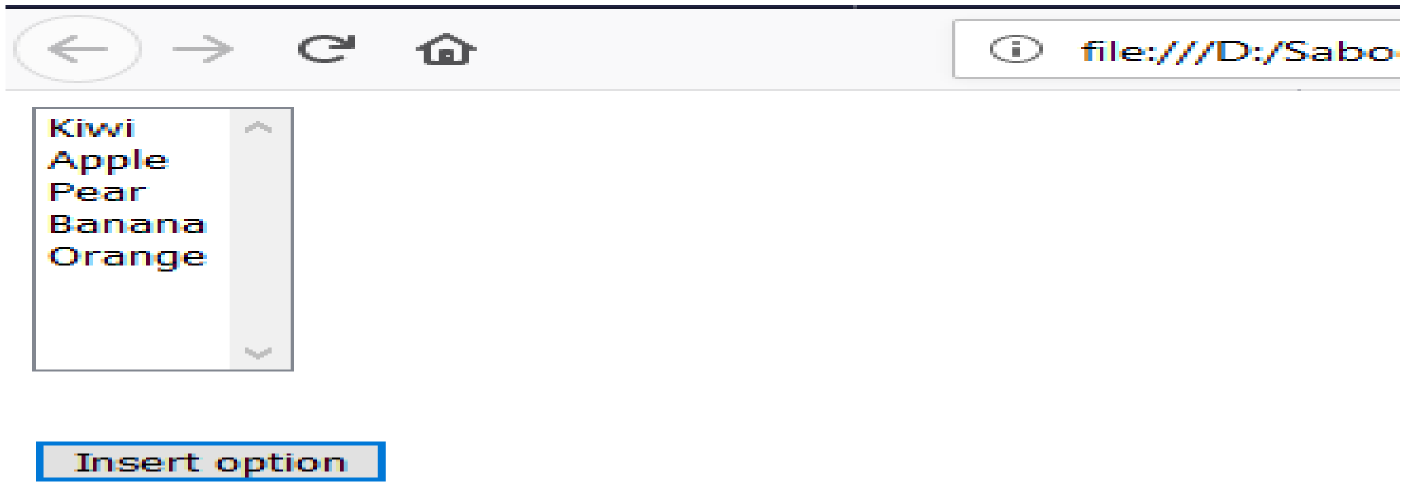
  where: option: Required. Specifies the option to add.
  Index : Optional. An integer that specifies the index position for where the new option should be inserted.

- *createElement()* method can be used to create an element node with the specified name which can be added into other existing HTML element.

  Syntax: *document.createElement(nodename);*

Changing option list dynamically

```
<html>
<body><form>
 <select id="mySelect" size="8">
  <option>Apple</option>
  <option>Pear</option>
  <option>Banana</option>
  <option>Orange</option>
 </select></form><br>
<button type="button" onclick="myFunction()">Insert option</button>
<script> function myFunction() {
 var x = document.getElementById("mySelect");
 var option = document.createElement("option");
 option.text = "Kiwi";
 x.add(option, x[0]); }
</script></body>
</html>
```



Changing option list dynamically

- remove():

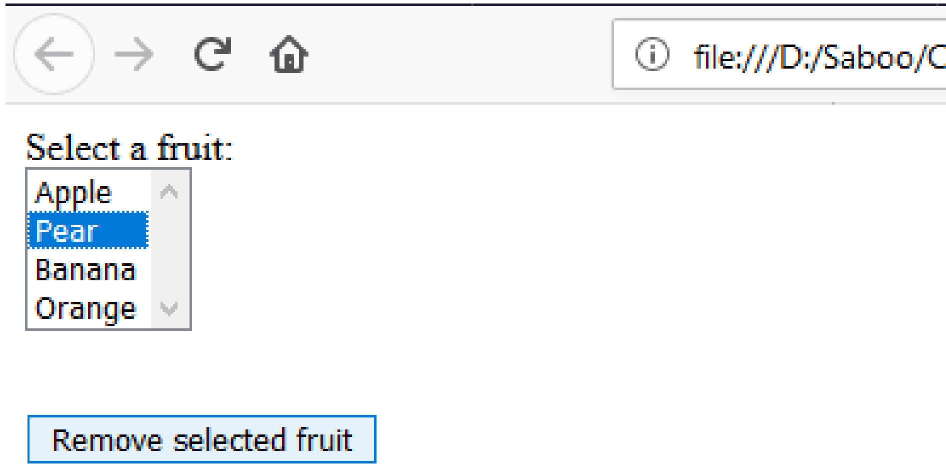
  - The remove() method is used to remove an option from a drop-down list.

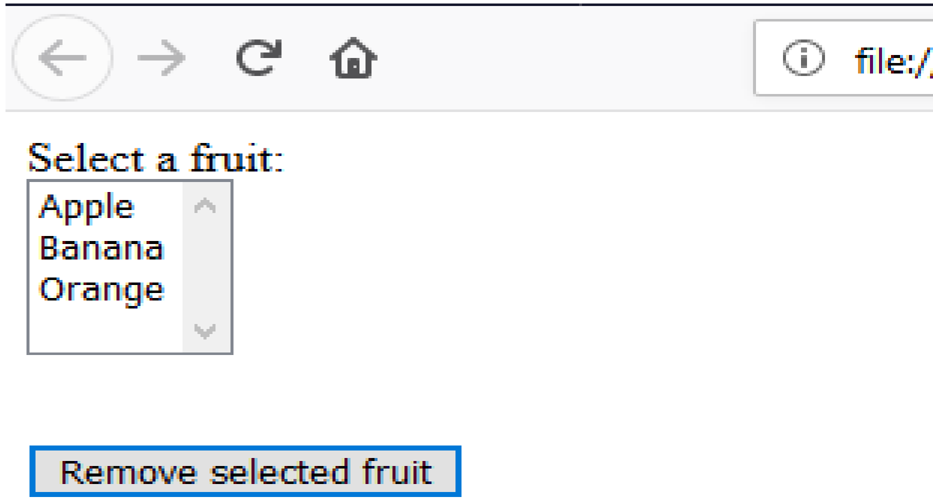
  - Syntax: *selectObject*.remove(*index*)

    where: Index : Required. Specifies the index of the option to remove. Index starts at 0

Changing option list dynamically

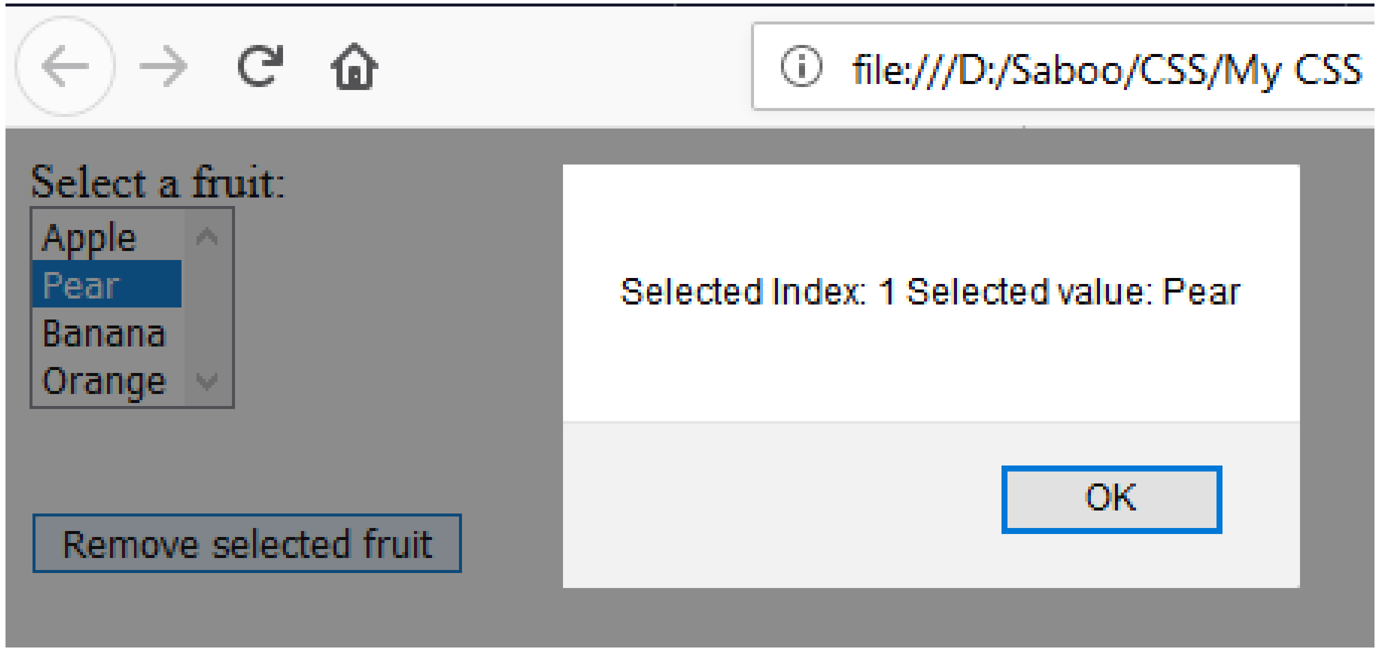
```
<html>
<body><form>
 Select a fruit:  <br>
 <select id="mySelect" size="4">
  <option>Apple</option>
  <option>Pear</option>
  <option>Banana</option>
  <option>Orange</option>
 </select></form><br>
<button onclick="myFunction()">Remove selected fruit</button>
<script> function myFunction() {
 var x = document.getElementById("mySelect");
 x.remove(x.selectedIndex); or x.remove(2); }
</script>
</body>
</html>
```





Evaluating list selection

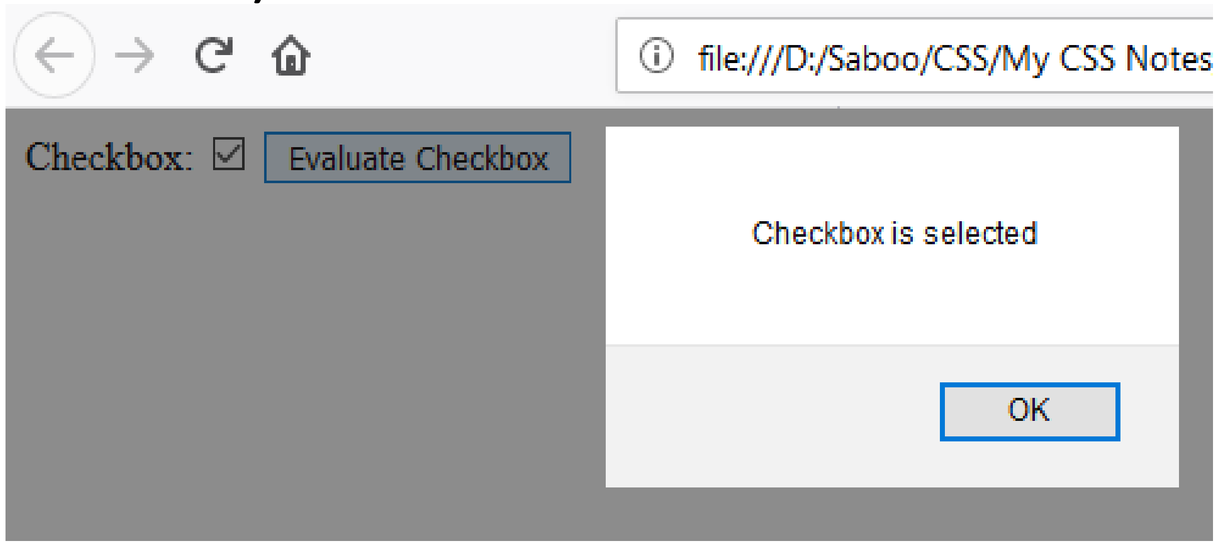
```
<html>
<body><form>
 Select a fruit:  <br>
 <select id="mySelect" size="4">
  <option>Apple</option>
  <option>Pear</option>
  <option>Banana</option>
  <option>Orange</option>
 </select></form><br>
<button onclick="myFunction()">Remove selected fruit</button>
<script> function myFunction() {
 var x = document.getElementById("mySelect");
 var index=x.selectedIndex;
 var val=x[index].value;
 alert("Selected Index: "+index+" Selected value: "+val); }
</script></body></html>
```



Selected Index: 1 Selected value: Pear

Evaluating checkbox selection

```
<html>
<body>
Checkbox: <input type="checkbox" id="myCheck">
<button onclick="check()">Evaluate Checkbox</button>
<script> function check() {
 if(document.getElementById("myCheck").checked)
   alert("Checkbox is selected");
 else
   alert("Checkbox is not selected"); }
</script>
</body></html>
```



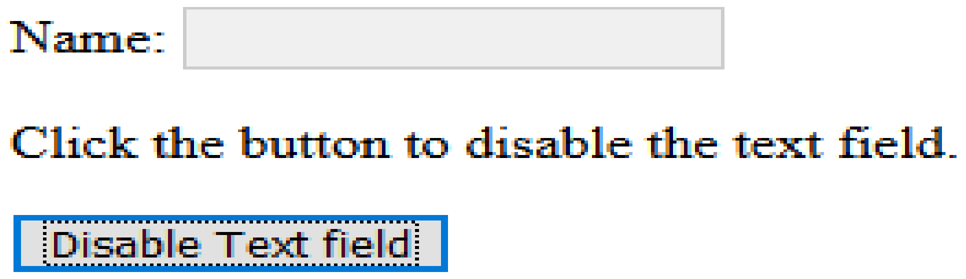
Checkbox is selected

Intrinsic Functions

- Following are some javascript functions that can be used with all the built-in Javascript objects.
  - isFinite(): Determines whether a value is finite, legal number.
  - isNaN(): Determines whether a value is an illegal number.
  - Number(): Converts an object value to a number.
  - parseFloat(): Parses a string and returns a floating point number.
  - parseInt(): Parses a string and returns an integer number.
  - String(): Converts an object value to a string.

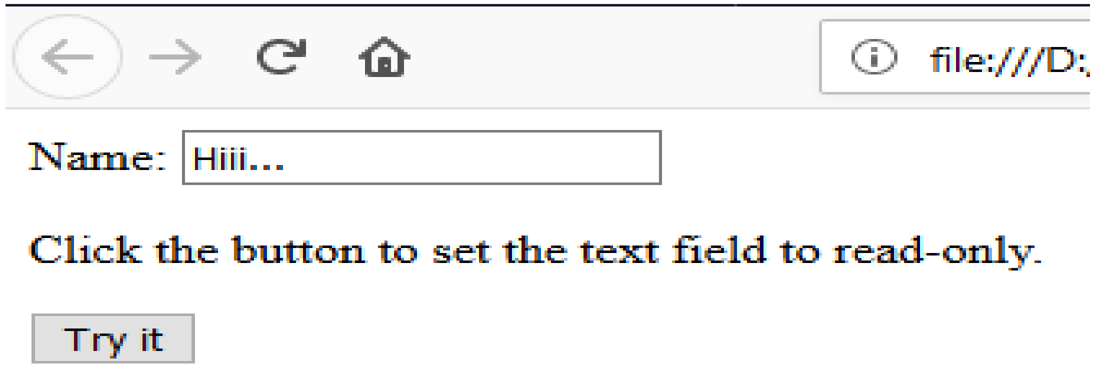
Disable an Element

```
<html>
<body>
Name: <input type="text" id="myText">
<p>Click the button to disable the text field.</p>
<button onclick="myFunction()">Disable Text field</button>
<script>
function myFunction() {
  document.getElementById("myText").disabled = true; }
</script></body>
</html>
```



Read Only Element

```
<html>
<body>
Name: <input type="text" id="myText">
<p>Click the button to set the text field to read-only.</p>
<button onclick="myFunction()">Try it</button>
<script>
function myFunction() {
  document.getElementById("myText").readOnly = true;
} </script></body>
</html>
```


file:///D:

Name: Hiii...

Click the button to set the text field to read-only.