Document Object Model DOM

OBJECTIVES

In this chapter you will learn:

- Study the Document Object Model (DOM)
- Understand the nature and structure of the DOM
- Access and change element attributes
- How to traverse elements in an HTML document.
- Add and remove content from the page
- Insert markup into a page using innerHTML
- How to change CSS styles dynamically
- To create JavaScript animations

What is the DOM?

- The **D**ocument **O**bject **M**odel (DOM) gives you access to all elements on a web page.
  - Your web browser builds a *model* of the web page (the *document*) that includes all *objects* on the page (tags, text, etc.)
  - Those objects are accessible via scripting languages in modern web browsers.

- The DOM is an API that allows programs to interact with HTML documents
  - W3C recommendations define standard DOM
  - With JavaScript, we can restructure an entire HTML document by dynamically adding, removing, changing, or reordering items on a page.
  - JavaScript gains access to all HTML elements through the DOM.

DOM Structure

- DOM documents have a treelike structure
  - When you load a document in your web browser, it creates several objects
  - These objects (nodes) exist in a hierarchy that reflects the structure of the HTML page

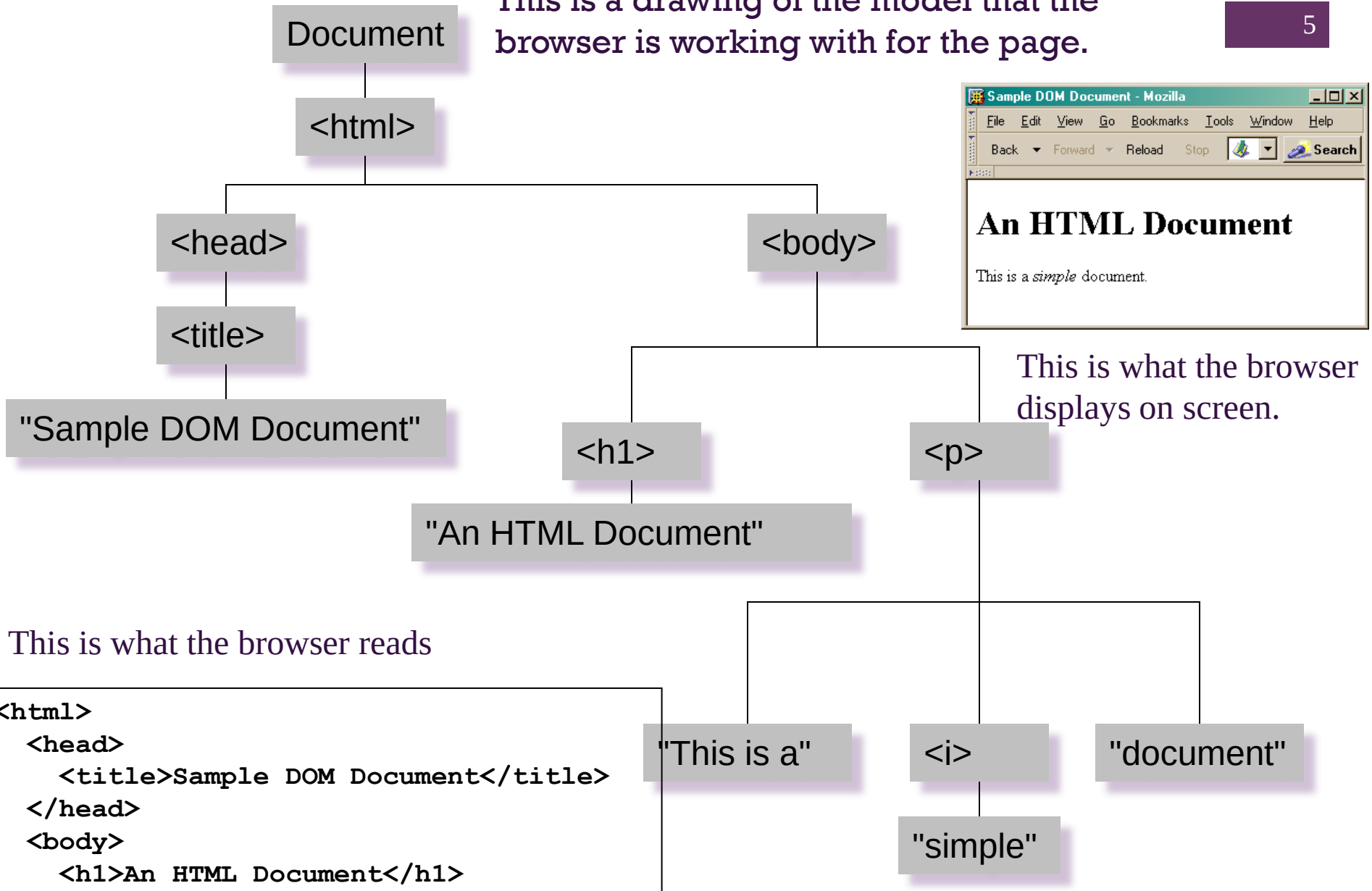
- DOM is an OO model
  - document elements (nodes) are objects with both data (properties) and operations (methods)
  - HTML elements -->objects
  - HTML element attributes --> properties

    <input type = "text" name = " address>

    would be represented as one object with two properties, type and name, with the values "text" and "address."

- The nodes in a document make up the page's DOM tree, which describes the relationships among elements
  - Nodes are related to each other through child-parent relationships
  - A node can have multiple children, but only one parent
  - Nodes with the same parent node are referred to as siblings
  - The HTML node in a DOM tree is called the root node because it has no parent

This is a drawing of the model that the browser is working with for the page.



This is what the browser displays on screen.

This is what the browser reads

```
<html>
  <head>
    <title>Sample DOM Document</title>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i>simple</i> document.
  </body>
</html>
```

Demonstration of a document's DOM tree.

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 12.1: domtree.html -->
6  <!-- Demonstration of a document's DOM tree. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>DOM Tree Demonstration</title>
10    </head>
11    <body>
12       <h1>An XHTML Page</h1>
13       <p>This page contains some basic XHTML elements. We use the Firefox
14          DOM Inspector and the IE Developer Toolbar to view the DOM tree
15          of the document, which contains a DOM node for every element in
16          the document.</p>
17       <p>Here's a list:</p>
18       <ul>
19          <li>One</li>
20          <li>Two</li>
21          <li>Three</li>
22       </ul>
23    </body>
24 </html>
```

HTML element

head element

title element

body element

h1 element

p element

p element

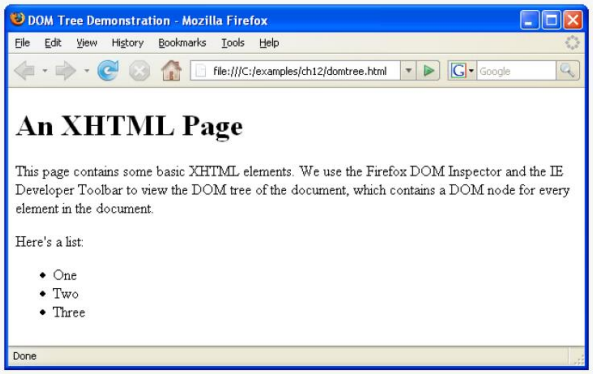
ul element

li element

li element

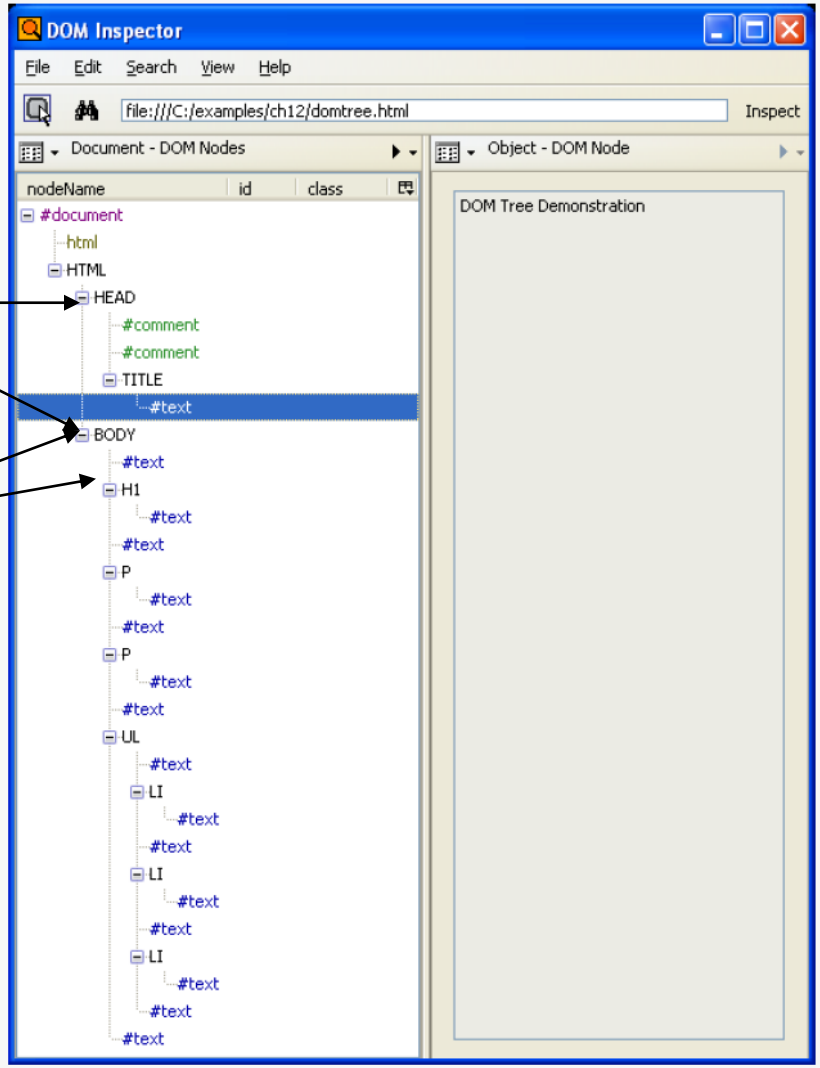
li element

a) The XHTML document is rendered in Firefox.



Demonstration of a document's DOM tree.

b) The Firefox DOM inspector displays the document tree in the left panel. The right panel shows information about the currently selected node.



HEAD and BODY nodes are siblings

The BODY node is the parent of the H1 node

Document Tree: Node

- There are many types of nodes in the DOM document tree, representing elements, text, comments, the document type declaration, etc.
  - Element Node – contains an HTML tag
  - Text Node – contains text
  - Text Nodes are contained in Element Nodes

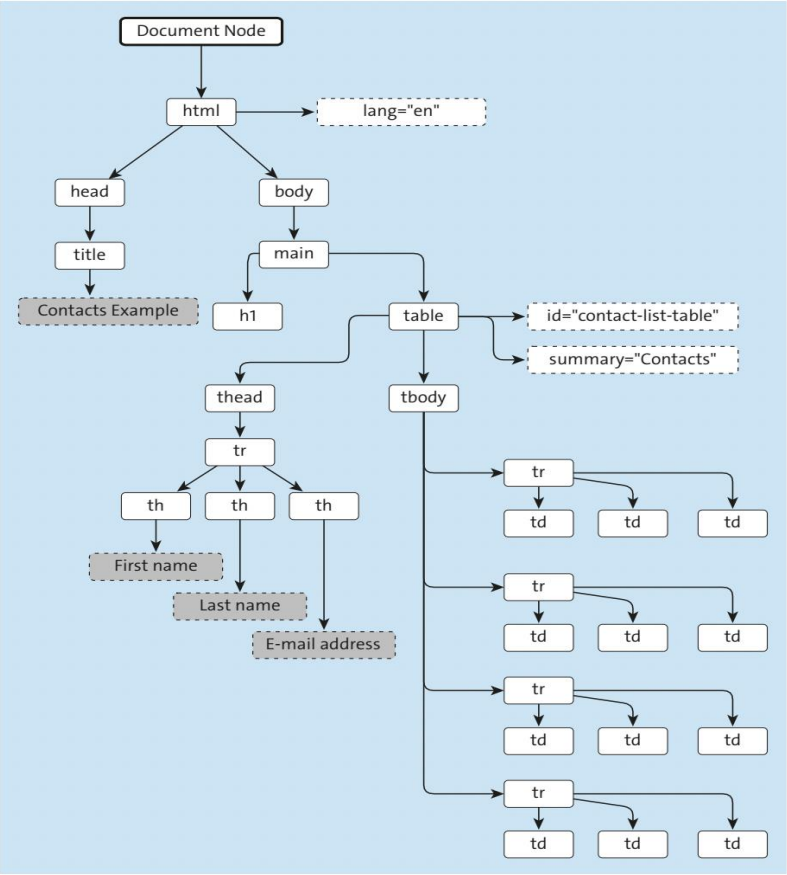


Figure 5.2 Example DOM Tree Structure

Adding Some Text To A Page

1. Create New Element Node

```
let newNode = document.createElement("p")
```

2. Create a Text Node

```
let newText = document.createTextNode("Some text.")
```

3. Attach the New Text Node to the New Element

```
newNode.appendChild(newText);
```

4. Find an Existing Element

```
let docElement = document.getElementById("thisLocation");
```

5. Append the New Element to the Existing Element

```
docElement.appendChild(newNode);
```

Putting the 5 Steps Together

```
<head>
<script language="javascript" type="text/javascript">
let myText;
myText = "This is new text to be added to the page dynamically.";
function addText(location) {
        let newNode;
         let newText;
         let docElement;
        newNode = document.createElement("p");
        newText = document.createTextNode(myText);
        newNode.appendChild(newText);
        docElement = document.getElementById(location);
        docElement.appendChild(newNode);
}
</script>
</head>
<body>
<p><a href="#" onclick="addText('thisLocation');">Click to add new text to the page</a></p>
<p id="thisLocation">New text will appear below here</p>
<p>Some further text in the page</p>
</body>
```

Remove a Node

- To remove a node, we use the element method

  removeChild(*name of node to be removed*)

- For example:

```
function remText(location) {
    let docElement;
    docElement = document.getElementById(location);
    docElement.removeChild(docElement.lastChild);
}
```

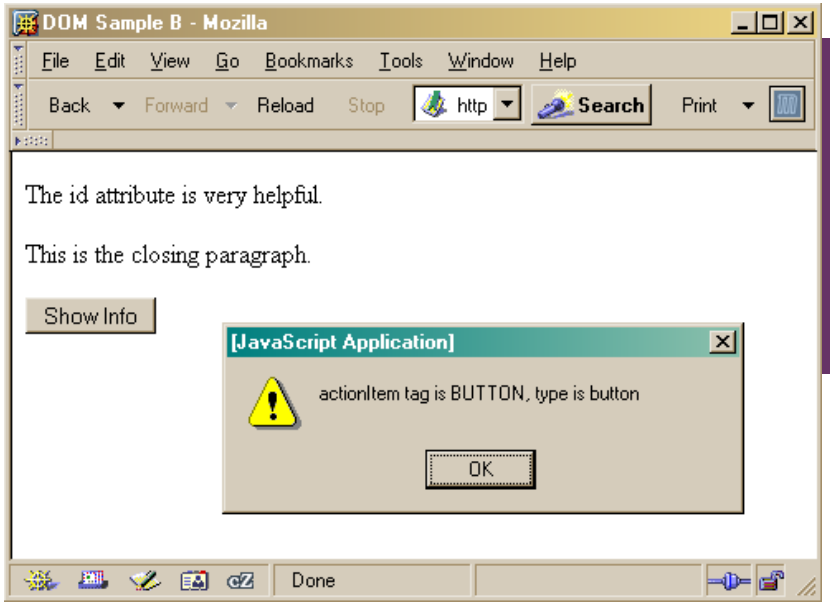
Selecting elements

- With id attribute
- With a specified name attribute
- With a specified tag name
- With a specified CSS class(es)
- Matching the specified CSS selector

Selecting Elements by ID

- **getElementById(…)**
  - This is a predefined function that makes use of the id that can be defined for any element in the page
  - An id must be unique in the page, so only one element is ever returned by this function
  - The argument to getElementById specifies which element is being requested

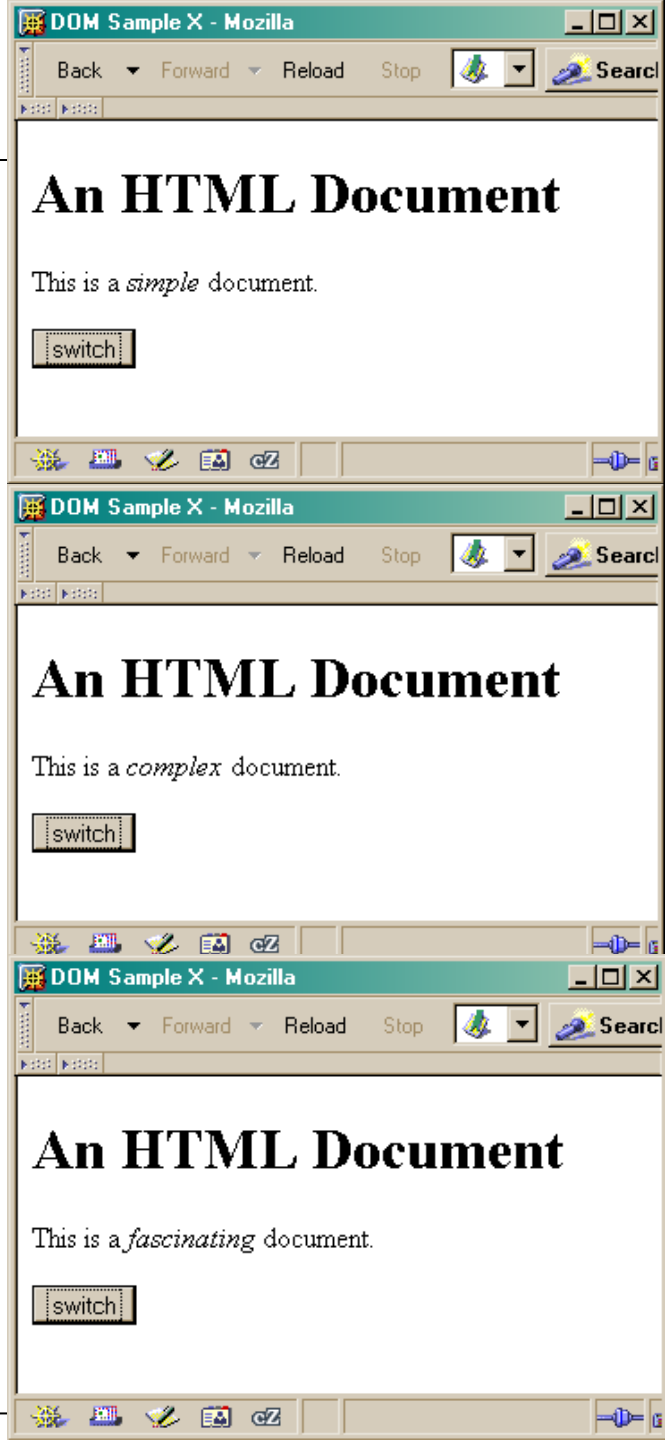
Some information about elements




```
<html>
  <head>
    <title>DOM Sample B</title>
    <script type="text/javascript">
    function showInfo() {
      let element = document.getElementById("actionItem");
      let buffer = element.id + " tag is " + element.tagName;
      buffer += ", type is "+element.type;
      alert(buffer);
    }
    </script>
  </head>
  <body>
    <p id="opener">The id attribute is very helpful.</p>
    <p id="closer">This is the closing paragraph.</p>
    <form>
    <button id="actionItem" type="button" onclick="showInfo()">Show
Info</button>
    </form>
  </body>
</html>
```

This is what the browser reads (dom3.html).

```
<html>
  <head>
    <title>DOM Sample 3</title>
    <script type="text/javascript">
    let switchCount = 0;
    let adjectives = ["simple","complex","fascinating","unique"];
    function switcher() {
                if (switchCount == (adjectives.length - 1))
                            switchCount = 0;
                else
                            switchCount++;
                let italicNode = document.getElementById("adjPhrase");
                italicNode.firstChild.nodeValue = adjectives[switchCount];
                }
    </script>
  </head>
  <body>
    <h1>An HTML Document</h1>
    <p>This is a <i id="adjPhrase">simple</i> document.
    <form>
    <button type="button" onclick="switcher()">switch</button>
    </form>
  </body>
</html>
```



Selecting Elements by TagName

- getElementsByTagName() allows you to work with groups of elements.
- Returned elements are in Document order
- This method returns “an array”

```
<script language="javascript" type="text/javascript">
theseElements = new Array();
theseElements = document.getElementsByTagName("li");
alert(theseElements.length);
for (i = 0; i < theseItems.length; i++) {
    alert(typeof theseItems[i]);
}
</script>
```

```
const tableCells = document.getElementsByTagName('td’);
 if(tableCells.length > 0) {// If at least one element is  found.
for(let i=0; i<tableCells.length; i++) {// Iterate all elements.
    const tableCell = tableCells[i];// Assign element to a variable.
     tableCell.style.fontFamily = 'Verdana';// Set new font.
    tableCell.style.fontSize = '9pt';// Set new font size.
    }
 }
```

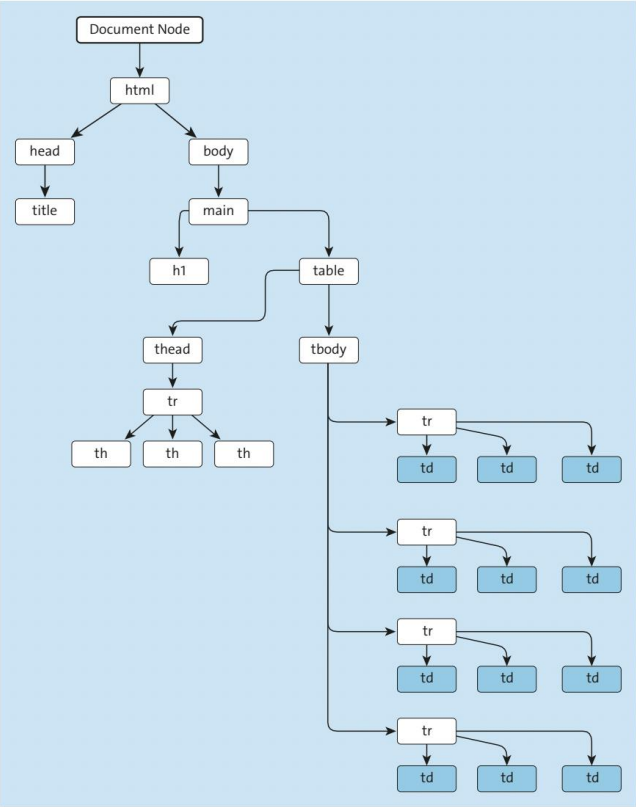


Figure 5.8 The getElementsByTagName() Method Selects Elements by Their Element Name

Getting all elements of a certain type

```
let allParas = document.getElementsByTagName("p");
for (let i = 0; i < allParas.length; i++) {
        allParas[i].style.backgroundColor = "yellow";
}
```

JS

```
<body>
        <p>This is the first paragraph</p>
        <p>This is the second paragraph</p>
        <p>You get the idea…</p>
</body>
```

HTML

Getting all elements with a CSS selector

```
let allParas = document.querySelectorAll("p");
for (let i = 0; i < allParas.length; i++) {
        allParas[i].style.backgroundColor = "yellow";
}
```

JS

```
<body>
        <p>This is the first paragraph</p>
        <p>This is the second paragraph</p>
        <p>You get the idea…</p>
</body>
```

HTML

Select the first element that matches a CSS selector

```
let onePara = document.querySelector ("p");
onePara.style.backgroundColor = "yellow";
```

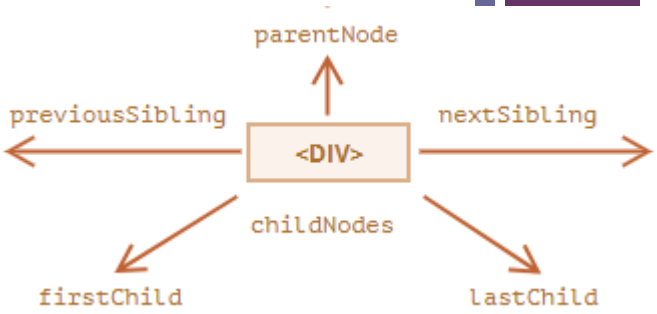
```
<body>
        <p>This is paragraph will be selected</p>
        <p>This is one won't</p>
        <p>Same and you get the idea...</p>
</body>
```

HTML

Walking the DOM

- The topmost tree nodes are available directly as document properties
  - document.documentElement
    - The topmost document node is document.documentElement. That's the DOM node of the <html> tag.
  - document.body
    - Another widely used DOM node is the <body> element – document.body.
  - document.head
    - The <head> tag is available as document.head.

Walking the DOM (2)



- Elements that are direct children:
  - childNodes
    - lists all child nodes, including text nodes.
  - children
    - only those children that are element nodes.
  - firstChild, lastchild
    - first and last child nodes
  - firstElementChild, lastElementChild
    - first and last element children
- Siblings are nodes that are children of the same parent
  - previousElementSibling, nextElementSibling
    - neighbor elements
- parentElement
  - Referencing the parent element

Tables

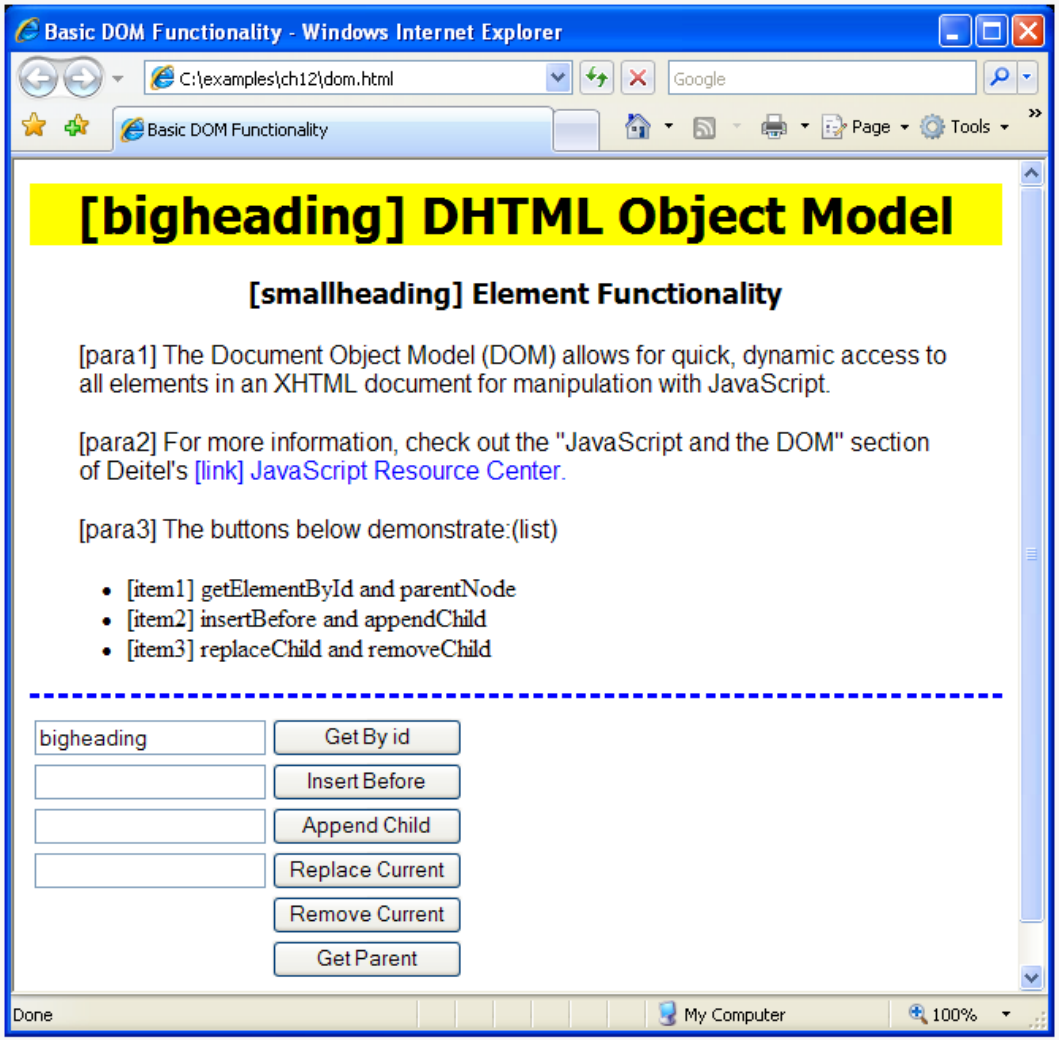
- table.rows – the collection of <tr> elements of the table
- table.caption/tHead/tFoot – references to elements <caption>, <thead>, <tfoot>
- table.tBodies – the collection of <tbody> elements
- tbody.rows – the collection of <tr> inside
- tr.cells – the collection of <td> and <th> cells inside the given <tr>
- tr.sectionRowIndex – the position (index) of the given <tr> inside the enclosing <thead>/<tbody>/<tfoot>.
- tr.rowIndex - the number of the <tr> in the table as a whole

Traversing and Modifying a DOM Tree

createElement	Creates a new DOM node, taking the tag name as an argument. Does not insert the element on the page.
createTextNode	• Creates a DOM node that can contain only text. Given a string argument, createTextNode inserts the string into the text node.
appendChild	• Called on a parent node to insert a child node (passed as an argument) after any existing children
parentNode	• property of any DOM node contains the node's parent
insertBefore	• Called on a parent with a new child and an existing child as arguments. The new child is inserted as a child of the parent directly before the existing child.
replaceChild	• Called on a parent, taking a new child and an existing child as arguments. The method inserts the new child into its list of children in place of the existing child.
removeChild	• Called on a parent with a child to be removed as an argument.

Basic DOM functionality.

a) This is the page when it first loads. It begins with the large heading highlighted.



Basic DOM functionality.

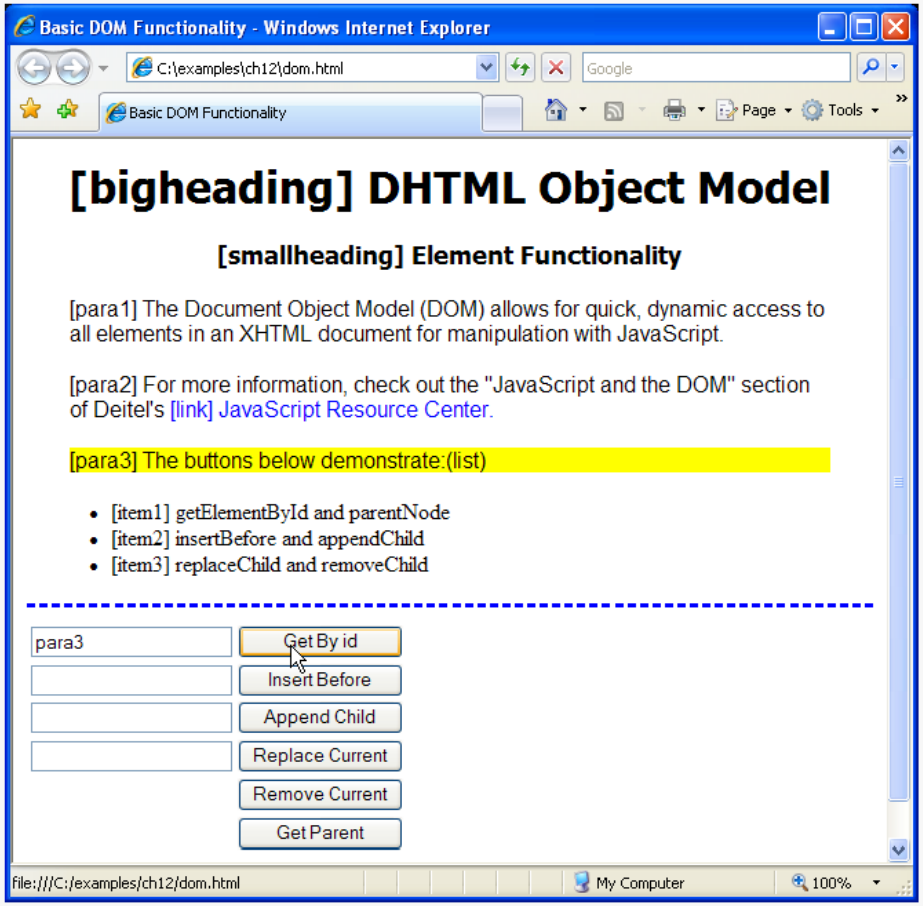
```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 12.2: dom.html -->
<!-- Basic DOM functionality. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Basic DOM Functionality</title>
      <style type = "text/css">
         h1, h3       { text-align: center;
                        font-family: tahoma, geneva, sans-serif }
         p            { margin-left: 5%;
                        margin-right: 5%;
                        font-family: arial, helvetica, sans-serif }
         ul           { margin-left: 10% }
         a            { text-decoration: none }
         a:hover      { text-decoration: underline }
         .nav         { width: 100%;
                        border-top: 3px dashed blue;
                        padding-top: 10px }
         .highlighted { background-color: yellow }
         .submit      { width: 120px }
      </style>
      <script type = "text/javascript">
         <!--
         var currentNode; // stores the currently highlighted node
         var idcount = 0; // used to assign a unique id to new elements

```

Creates a class to highlight text

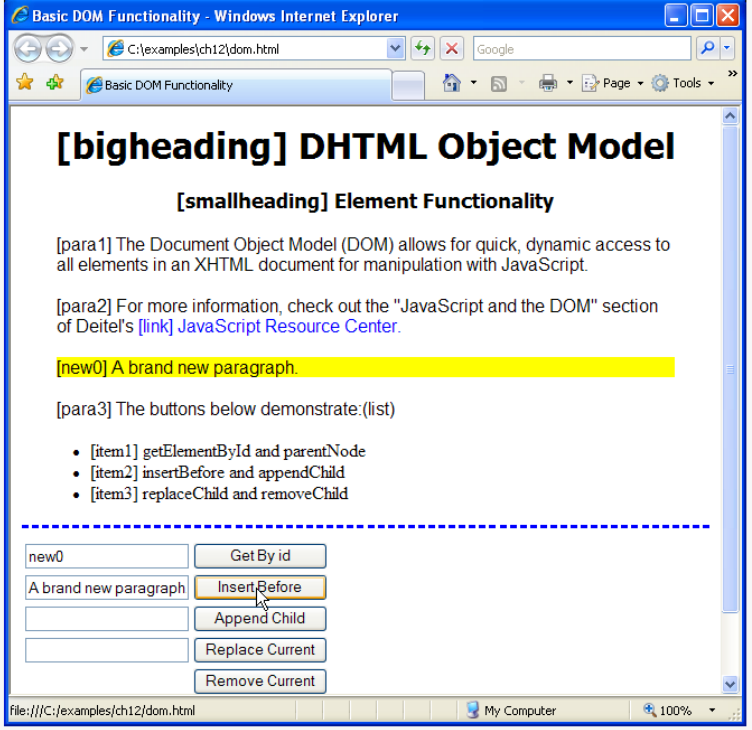
```
// get and highlight an element by its id attribute
function byId()
{
   var id = document.getElementById( "gbi" ).value;
   var target = document.getElementById( id );

   if ( target )
      switchTo( target );
} // end function byId
```

Calls function `switchTo` if the object can be found

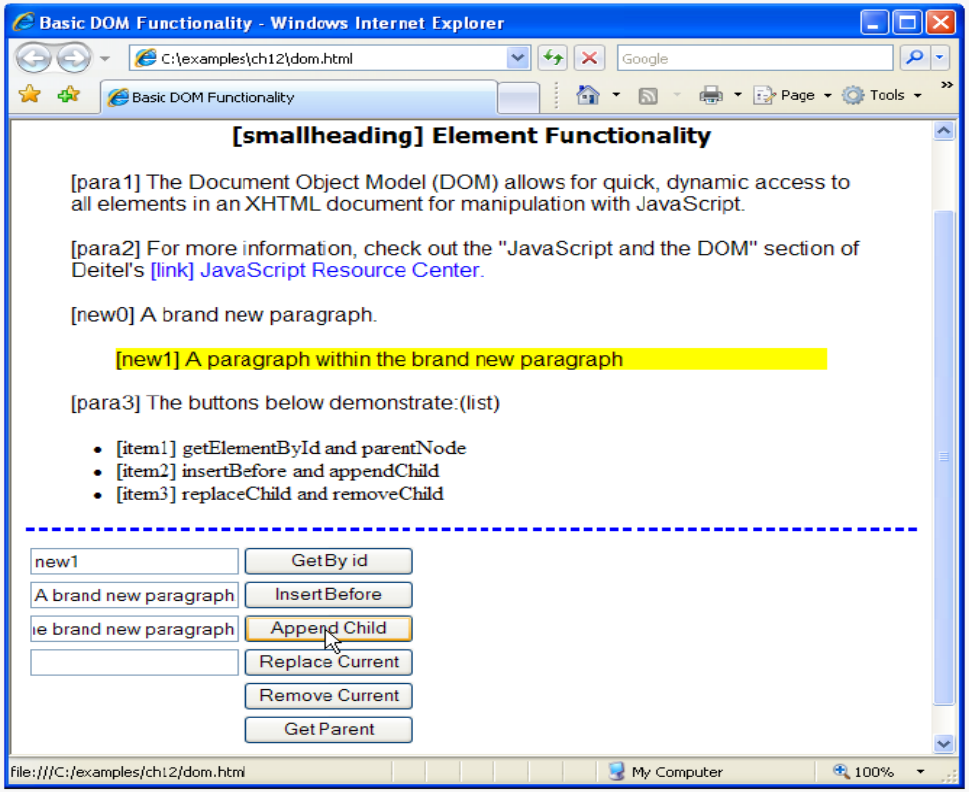
b) This is the document after using the Get By id button to select para3.



```
40          // insert a paragraph element before tl
41          // using the insertBefore method
42          function insert()
43          {
44             var newNode = createNewNode(
45                document.getElementById( "ins" ).value );
46             currentNode.parentNode.insertBefore( newNode, currentNode );
47             switchTo( newNode );
48          } // end function insert
49
50                                                          ent node
51
52
53
54
55
56
57
58
```



d) Using the Append Child button, a child paragraph is created.



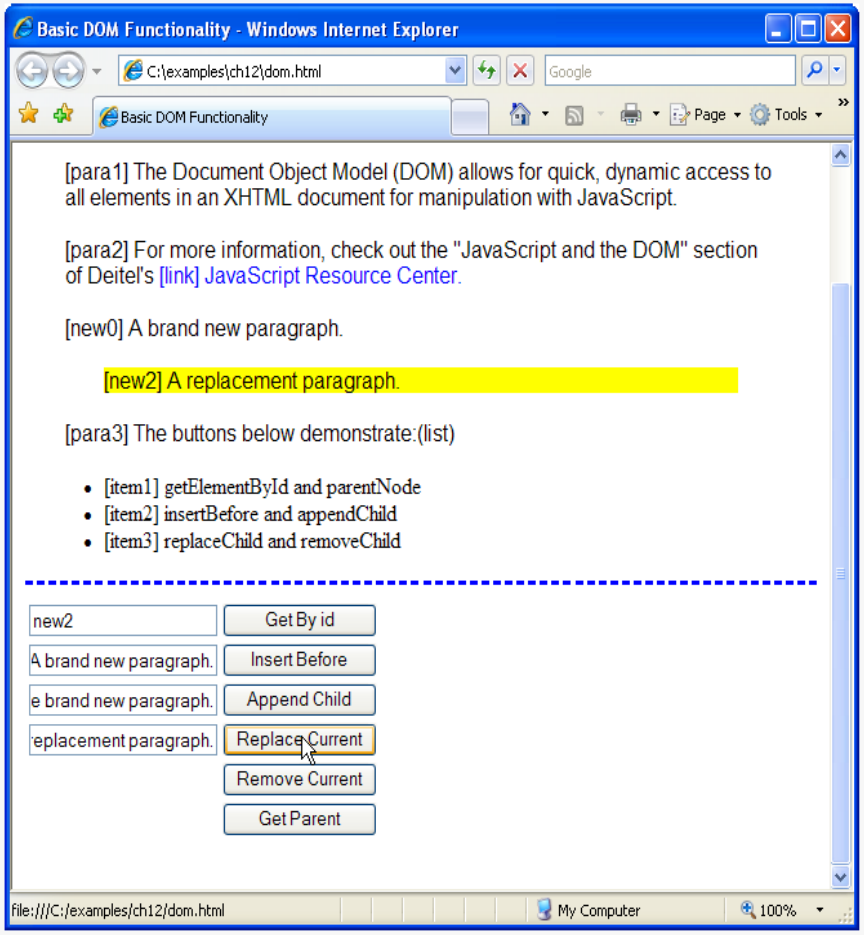

```
49
50      // append a paragraph node as the child of the current node
51      function appendNode()
52      {
53         var newNode = createNewNode(
54            document.getElementById( "append" ).value );
55         currentNode.appendChild( newNode );
56         switchTo( newNode );
57      } // end function appendNode
58
```

Inserts `newNode` as a child of the current node

```
// replace the currently selected node with a paragraph node
function replaceCurrent()
{
   var newNode = createNewNode(
      document.getElementById( "replace" ).value );
   currentNode.parentNode.replaceChild( newNode, currentNode );
   switchTo( newNode );
} // end function replaceCurrent
```

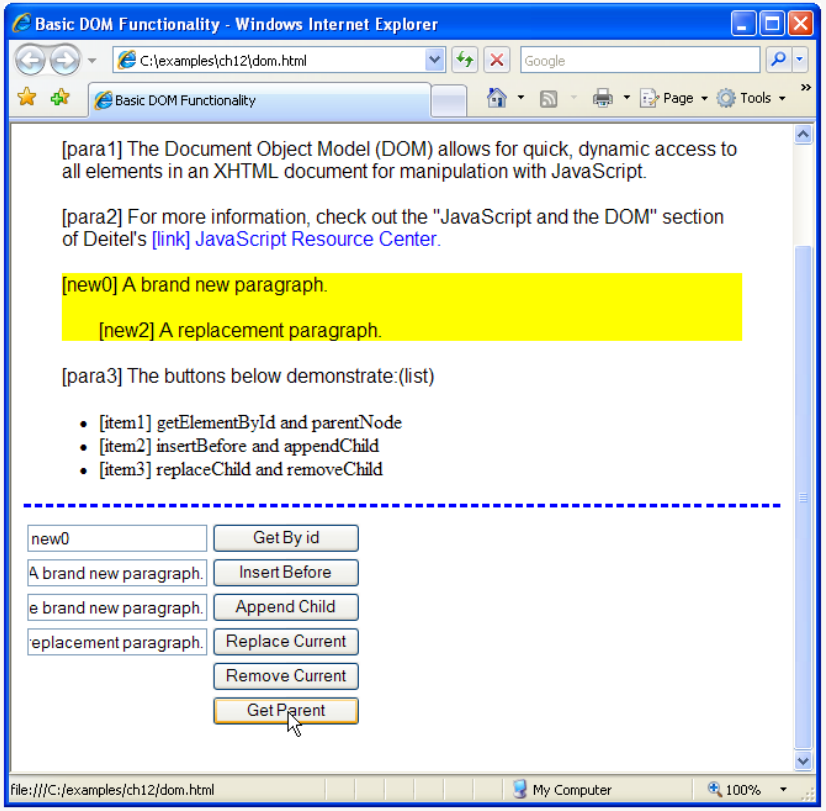
Gets the parent of `currentNode`, then inserts `newNode` into its list of children in place of `currentNode`

e) The selected paragraph is replaced with a new one.



```
// replace the currently selected node with a paragraph node
```

f) The Get Parent button gets the parent of the selected node.



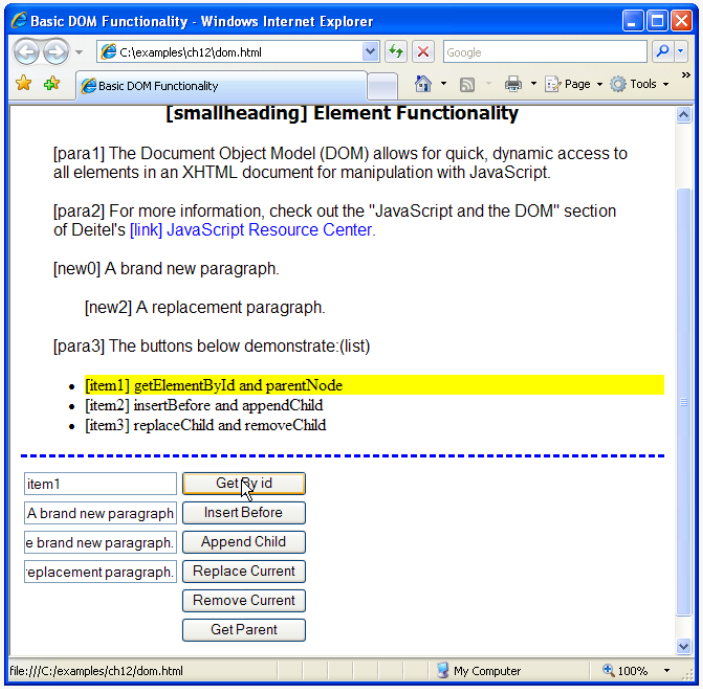

```
} // end function remove

// get and highlight the parent of the current node
function parent()
{
   var target = currentNode.parentNode;

   if ( target != document.body )
      switchTo( target );
   else
      alert( "No parent." );
} // end function parent
```

Gets the parent node

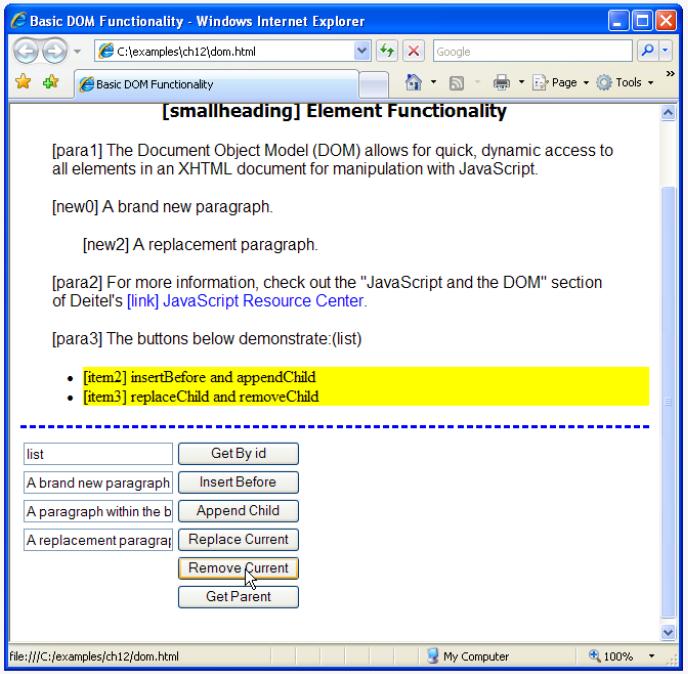
Makes sure the parent is not the `body` element

```
// remove the current node
function remove()
{
   if ( currentNode.parentNode == document.body )
      alert( "Can't remove a top-level element." );
   else
   {
      var oldNode = currentNode;
      switchTo( oldNode.parentNode );
      currentNode.removeChild( oldNode );
   }
} // end function remove
```

g) Now we select the first list item.



h) The Remove Current button removes the current node and selects its parent.



```
91
92          // helper function that returns a new paragraph node containing
93          // a unique id and the given text
94          function createNewNode( text )
95          {
96             var newNode = document.createElement( "p" );
97             nodeId = "new" + idcount;
98             ++idcount;
99             newNode.id = nodeId;
100            text = "[" + nodeId + "] " + text;
101            newNode.appendChild(document.createTextNode( text ) );
102            return newNode;
103         } // end function createNewNode
104
105         // helper function that switches to a new currentNode
106         function switchTo( newNode )
107         {
108            currentNode.className = ""; // remove old highlighting
109            currentNode = newNode;
110            currentNode.className = "highlighted"; // highlight new node
111            document.getElementById( "gbi" ).value = currentNode.id;
112         } // end function switchTo
113         // -->
114      </script>
115   </head>
116   <body onload = "currentNode = document.getElementById( 'bigheading' )">
117      <h1 id = "bigheading" class = "highlighted">
118         [bigheading] DHTML Object Model</h1>
119      <h3 id = "smallheading">[smallheading] Element Functionality</h3>
```

Creates (but does not insert) a new `p` node

Creates a unique `id` for the new node

Creates new text node with the contents of variable `text`, then inserts this node as a child of `newNode`

Changes `class` attribute to unhighlight old node

Highlights `currentNode`

Assigns `currentNode`'s `id` to the input field's `value` property

Basic DOM functionality.

```
120      <p id = "para1">[para1] The Document Object Model (DOM) allows for
121         quick, dynamic access to all elements in an XHTML document for
122         manipulation with JavaScript.</p>
123      <p id = "para2">[para2] For more information, check out the
124         "JavaScript and the DOM" section of Deitel's
125         <a id = "link" href = "http://www.deitel.com/javascript">
126            [link] JavaScript Resource Center.</a></p>
127      <p id = "para3">[para3] The buttons below demonstrate:(list)</p>
128      <ul id = "list">
129         <li id = "item1">[item1] getElementById and parentNode</li>
130         <li id = "item2">[item2] insertBefore and appendChild</li>
131         <li id = "item3">[item3] replaceChild and removeChild </li>
132      </ul>
133      <div id = "nav" class = "nav">
134         <form onsubmit = "return false" action = "">
135            <table>
136               <tr>
137                  <td><input type = "text" id = "gbi"
138                     value = "bigheading" /></td>
139                  <td><input type = "submit" value = "Get By id"
140                     onclick = "byId()" class = "submit" /></td>
141               </tr><tr>
142                  <td><input type = "text" id = "ins" /></td>
143                  <td><input type = "submit" value = "Insert Before"
144                     onclick = "insert()" class = "submit" /></td>
145               </tr><tr>
146                  <td><input type = "text" id = "append" /></td>
147                  <td><input type = "submit" value = "Append Child"
148                     onclick = "appendNode()" class = "submit" /></td>
149               </tr><tr>
```

Basic DOM functionality.

```
150                 <td><input type = "text" id = "replace" /></td>
151                 <td><input type = "submit" value = "Replace Current"
152                    onclick = "replaceCurrent()" class = "submit" /></td>
153              </tr><tr><td />
154                 <td><input type = "submit" value = "Remove Current"
155                    onclick = "remove()" class = "submit" /></td>
156              </tr><tr><td />
157                 <td><input type = "submit" value = "Get Parent"
158                    onclick = "parent()" class = "submit" /></td>
159              </tr>
160           </table>
161        </form>
162     </div>
163   </body>
164</html>
```

Insertion and removal

- node.append(...nodes or strings)
  - insert into node, at the end
- node.prepend(...nodes or strings)
  - insert into node, at the beginning
- node.before(...nodes or strings)
  - insert right before node
- node.after(...nodes or strings)
  - insert right after node
- node.replaceWith(...nodes or strings)
  - replace node
- node.remove()
  - remove the node

insertAdjacentHTML

- Given some HTML in html, elem.insertAdjacentHTML(where, html) inserts it depending on the value of where:
- "beforebegin“
  - insert html right before elem,
- "afterbegin“
  - insert html into elem, at the beginning,
- "beforeend“
  - insert html into elem, at the end,
- "afterend“
  - insert html right after elem

Attribute Nodes

- We can get at the attributes of an element through attribute nodes
- Attribute nodes, like text nodes are always contained in element nodes
- We shall look at methods: getAttribute() and setAttribute()
- Example:

```
function dispAttribs() {
   paras = document.getElementsByTagName("p");
   for (i = 0; i < paras.length; i++) {
         let messg = paras[i].getAttribute("className");
         alert(messg);
         }
}
```

- Add this to the bottom of the body:

```
<p onclick="dispAttribs();">Click here to see class attributes</p>
```

Setting Attribute Nodes

- Example:

```
function chngAttribs() {
   paras = document.getElementsByTagName("p");
   for (i = 0; i < paras.length; i++) {
         paras[i].setAttribute("className","jazz");
   }
}
```

- Add this to the bottom of the body:

```
<p onclick="chngAttribs();">Click here to change class
   attributes</p>
```

Removing Attribute Nodes

- Example:

```
function rmAttribs() {
   paras = document.getElementsByTagName("p");
   for (i = 0; i < paras.length; i++) {
         paras[i].removeAttribute("title")
   }
}
```

- Add this to the bottom of the body:

```
<p title="“hello" onclick=" rmAttribs();">Click here to
   remove title attributes</p>
```

Attributes

- attributes
  - is a collection of all attributes
  - E.g. elem.attributes

- has Attribute
  - Check for the existences of a specific attribute by name
  - E.g. elem.hasAttribute(name)

- Dataset
  - All attributes starting with “data-” are reserved for programmers’ use. They are available in the dataset property.
  - E.g., if an elem has an attribute named "data-about", it’s available as **elem.dataset.about**

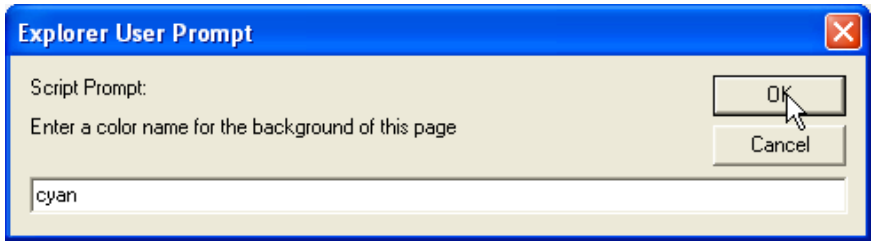
Dynamic Styles

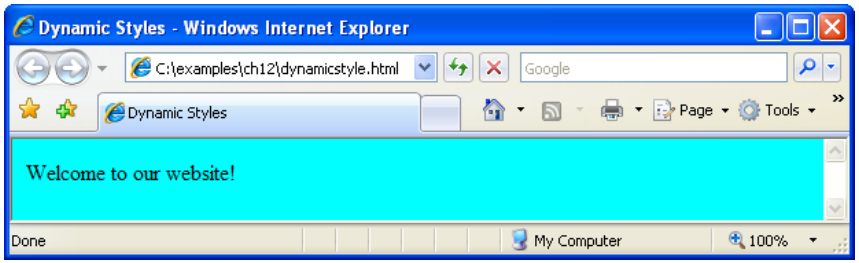
- An element's style can be changed dynamically
  - E.g., in response to user events
  - Can create many effects, including mouse hover effects, interactive menus, and animations
- body property of the document object
  - Refers to the body element in the HTML page
- style property
  - can access a CSS property in the format *node.style.styleproperty*.
- CSS property with a hyphen (-), such as background-color, is referred to as backgroundColor in JavaScript
  - Removing the hyphen and capitalizing the first letter of the following word is the convention for most CSS properties

```
1  <?xml version = "1.0" encoding = "utf-8"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 12.4: dynamicstyle.html -->
6  <!-- Dynamic styles. -->
7  <html xmlns = "http://www.w3.org/1999/xhtml">
8     <head>
9        <title>Dynamic Styles</title>
10       <script type = "text/javascript">
11          <!--
12          function start()
13          {
14             var inputColor = prompt( "Enter a color name for the " +
15                "background of this page", "" );
16             document.body.style.backgroundColor = inputColor;
17          } // end function start
18          // -->
19       </script>
20    </head>
21    <body id = "body" onload = "start()">
22       <p>Welcome to our website!</p>
23    </body>
24 </html>
```

Prompts the user for a color

Sets the `background-color` CSS property to the user's color





Using innerHTML

- All HTML in the tag is replaced when the innerHTML method is used
- innerHTML is not part of the DOM – so it may one day disappear – though it is universally recognised by browsers
- Tags within the innerHTML are not part of the DOM tree so they cannot be manipulated
- In the body of a blank HTML page insert a div tag:

  This will be replaced
- In the head of the page place this code:

```
window.onload = function() {
let testdiv = document.getElementById("test");
testdiv.innerHTML = "<p>Now we have inserted <em>this</em>
   instead!</p>";
}
```

There is also outer HTML

- Contains the full HTML of the element.
- A write operation into elem.outerHTML does not touch elem itself.
- Instead it gets replaced with the new HTML in the outer context.
- E.g. The variable can point to any element
  - elem.outerHTML = “ changed ”

textContent

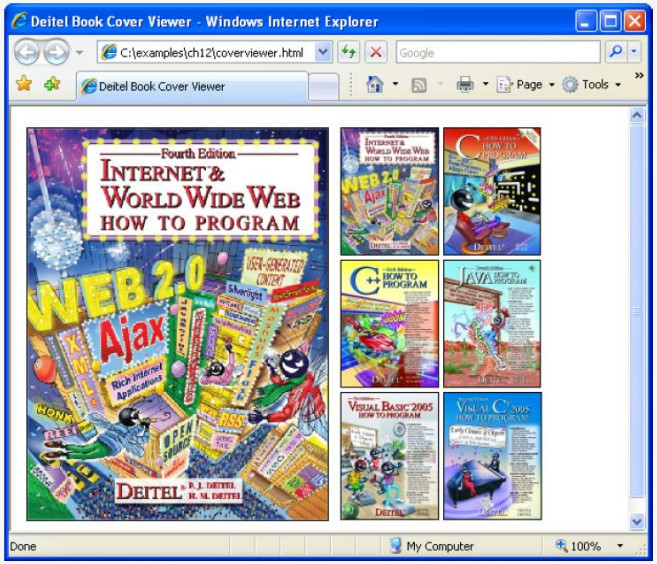
- The text inside the element: HTML minus all <tags>.
- Writing into it puts the text inside the element, with all special characters and tags treated exactly as text.
- Can safely insert user-generated text and protect from unwanted HTML insertions.
- E.g.
  - document.getElementById("p1").textContent = "Hello";

Dynamic Styles (Cont.)

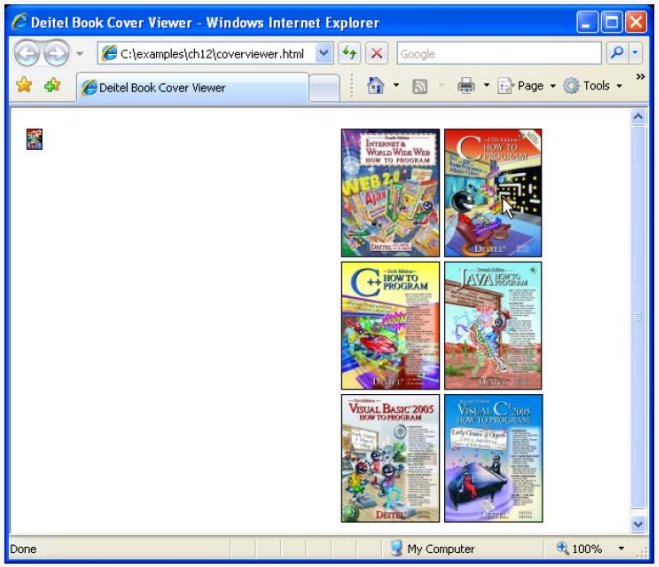
- setInterval method of the window object
  - Repeatedly executes a statement on a certain interval
  - Takes two parameters
    - A statement to execute repeatedly
    - An integer specifying how often to execute it, in milliseconds
  - Returns a unique identifier to keep track of that particular interval.
- window object's clearInterval method
  - Stops the repetitive calls of object's setInterval method
  - Pass to clearInterval the interval identifier that setInterval returned

Dynamic styles used for animation

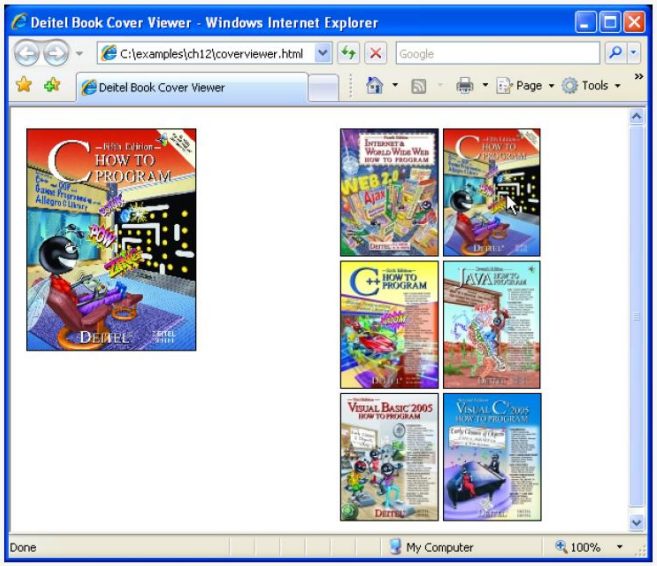
a) The cover viewer page loads with the cover of this book.



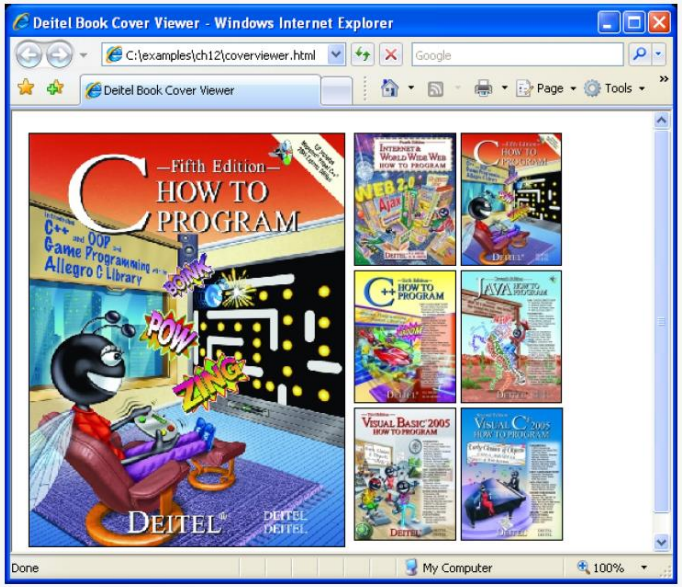
b) When the user clicks the thumbnail of C How to Program, the full-size image begins growing from the top-left corner of the window.



c) The cover continues to grow.



d) The animation finishes when the cover reaches its full size.



Dynamic styles used for animation

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 12.5: coverviewer.html -->
<!-- Dynamic styles used for animation. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Deitel Book Cover Viewer</title>
      <style type = "text/css">
         .thumbs   { width: 192px;
                     height: 370px;
                     padding: 5px;
                     float: left }
         .mainimg  { width: 289px;
                     padding: 5px;
                     float: left }
         .imgCover { height: 373px }
         img       { border: 1px solid black }
      </style>
      <script type = "text/javascript">
         <!--
         var interval = null; // keeps track of the interval
         var speed = 6; // determines the speed of the animation
         var count = 0; // size of the image during the animation

         // called repeatedly to animate the book cover
         function run()
         {
            count += speed;
```

Dynamic styles used for animation

```
31
32          // stop the animation when the image is large enough
33          if ( count >= 375 )
34          {
35             window.clearInterval( interval );
36             interval = null;
37          } // end if
38
39          var bigImage = document.getElementById( "imgCover" );
40          bigImage.style.width = .7656 * count + "px";
41          bigImage.style.height = count + "px";
42       } // end function run
43
44       // inserts the proper image into the main image area and
45       // begins the animation
46       function display( imgfile )
47       {
48          if ( interval )
49             return;
50
51          var bigImage = document.getElementById( "imgCover" );
52          var newNode = document.createElement( "img" );
53          newNode.id = "imgCover";
54          newNode.src = "fullsize/" + imgfile;
55          newNode.alt = "Large image";
56          newNode.className = "imgCover";
57          newNode.style.width = "0px";
58          newNode.style.height = "0px";
59          bigImage.parentNode.replaceChild( newNode, bigImage );
60          count = 0; // start the image at size 0
```

Stops the animation when the image has reached its full size

Keeps aspect ratio consistent

Sets properties for the new `img` node

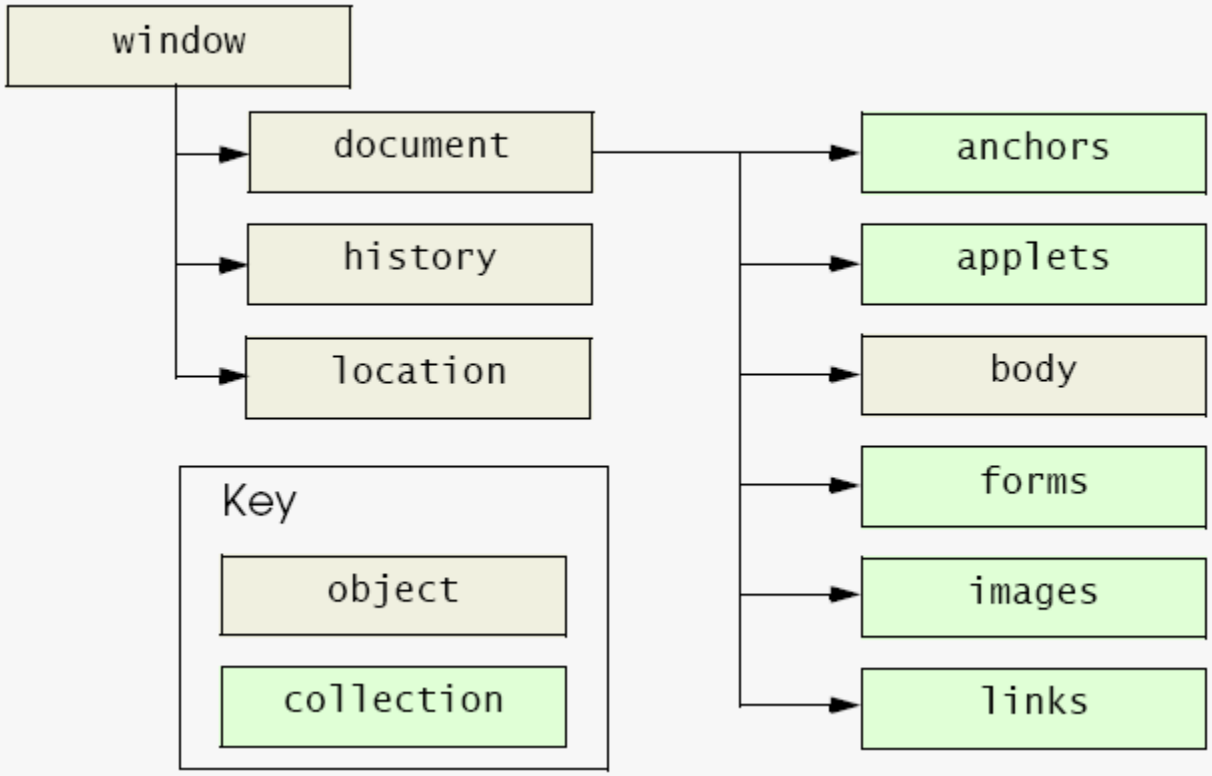
Swaps `newNode` for the old cover node

```
61            interval = window.setInterval( "run()", 10 ); // animate
62         } // end function display
63         // -->
64      </script>
65   </head>
66   <body>
67      <div id = "mainimg" class = "mainimg">
68         <img id = "imgCover" src = "fullsize/iw3htp4.jpg"
69            alt = "Full cover image" class = "imgCover" />
70      </div>
71      <div id = "thumbs" class = "thumbs" >
72         <img src = "thumbs/iw3htp4.jpg" alt = "iw3htp4"
73            onclick = "display( 'iw3htp4.jpg' )" />
74         <img src = "thumbs/chtp5.jpg" alt = "chtp5"
75            onclick = "display( 'chtp5.jpg' )" />
76         <img src = "thumbs/cpphtp6.jpg" alt = "cpphtp6"
77            onclick = "display( 'cpphtp6.jpg' )" />
78         <img src = "thumbs/jhtp7.jpg" alt = "jhtp7"
79            onclick = "display( 'jhtp7.jpg' )" />
80         <img src = "thumbs/vbhtp3.jpg" alt = "vbhtp3"
81            onclick = "display( 'vbhtp3.jpg' )" />
82         <img src = "thumbs/vcsharphtp2.jpg" alt = "vcsharphtp2"
83            onclick = "display( 'vcsharphtp2.jpg' )" />
84      </div>
85   </body>
86 </html>
```

Executes function `run` every 10 milliseconds

Dynamic styles used for animation

W3C Document Object Model ? BOM.



Objects and collections in the W3C Document Object Model ? BOM

Object or collection	Description
Objects	
`window`	Represents the browser window and provides access to the `document` object contained in the `window`. Also contains `history` and `location` objects.
`document`	Represents the XHTML document rendered in a window. The `document` object provides access to every element in the XHTML document and allows dynamic modification of the XHTML document. Contains several collections for accessing all elements of a given type.
`body`	Provides access to the `body` element of an XHTML document.
`history`	Keeps track of the sites visited by the browser user. The object provides a script programmer with the ability to move forward and backward through the visited sites.
`location`	Contains the URL of the rendered document. When this object is set to a new URL, the browser immediately navigates to the new location.

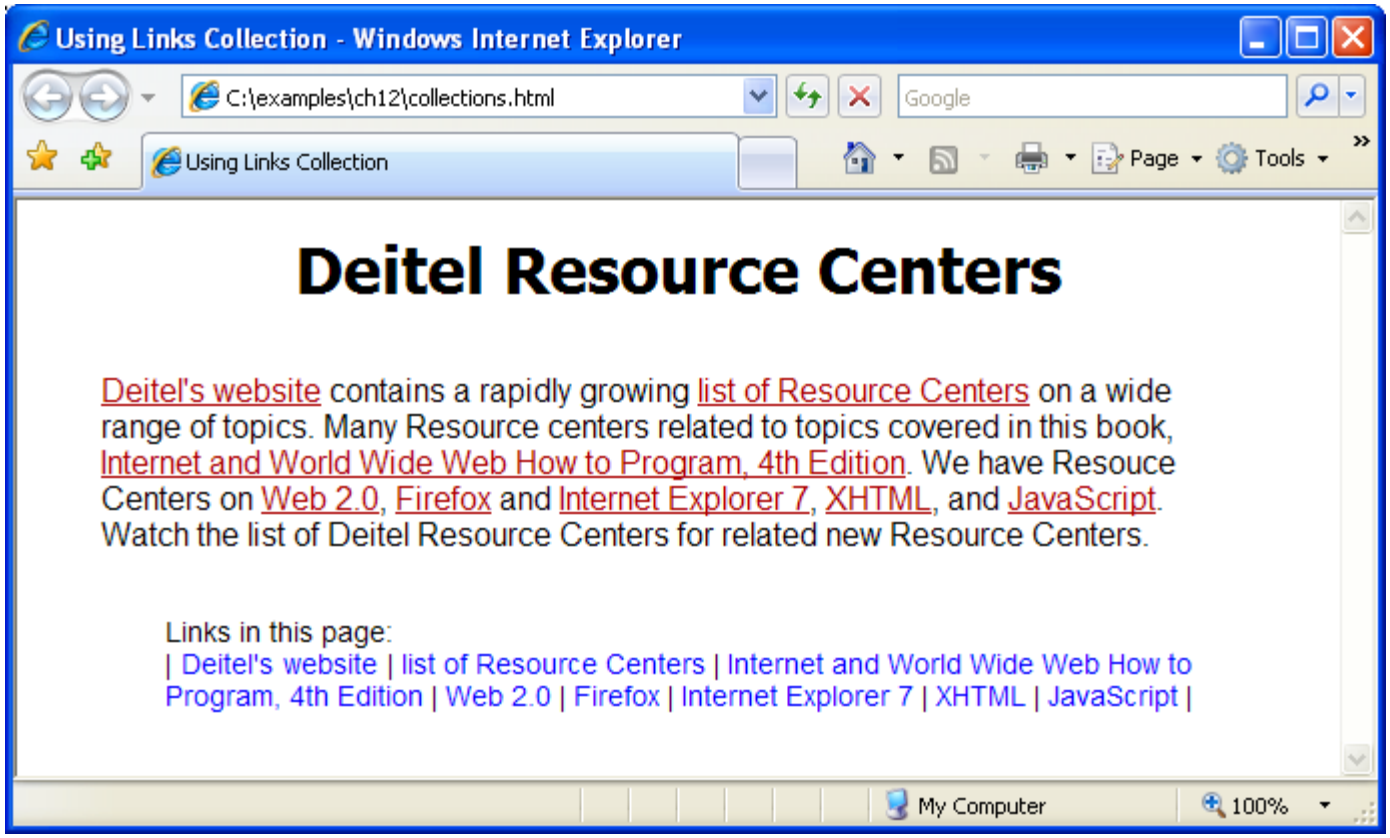
Objects and collections in the W3C Document Object Model ? BOM

Object or collection	Description
Collections	
`anchors`	Collection contains all the anchor elements (`a`) that have a `name` or `id` attribute. The elements appear in the collection in the order in which they were defined in the XHTML document.
`forms`	Contains all the `form` elements in the XHTML document. The elements appear in the collection in the order in which they were defined in the XHTML document.
`images`	Contains all the `img` elements in the XHTML document. The elements appear in the collection in the order in which they were defined in the XHTML document.
`links`	Contains all the anchor elements (`a`) with an `href` property. The elements appear in the collection in the order in which they were defined in the XHTML document.

DOM Collections

- Access items in a collection via square brackets
- item method of a DOM collection
  - Access specific elements in a collection, taking an index as an argument
- namedItem method
  - takes a name as a parameter and finds the element in the collection, if any, whose id attribute or name attribute matches it
- href property of a DOM link node
  - Refers to the link's href attribute
- Collections allow easy access to all elements of a single type in a page
  - Useful for gathering elements into one place and for applying changes across an entire page

Using the **links** collection



Using the **links** collection

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 12.3: collections.html -->
<!-- Using the links collection. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Using Links Collection</title>
      <style type = "text/css">
         body          { font-family: arial, helvetica, sans-serif }
         h1            { font-family: tahoma, geneva, sans-serif;
                         text-align: center }
         p             { margin: 5% }
         p a           { color: #aa0000 }
         .links        { font-size: 14px;
                         text-align: justify;
                         margin-left: 10%;
                         margin-right: 10% }
         .link a       { text-decoration: none }
         .link a:hover { text-decoration: underline }
      </style>
      <script type = "text/javascript">
         <!--
         function processlinks()
         {
            var linkslist = document.links; // get the document's links
            var contents = "Links in this page:\n<br />| ";

            // concatenate each link to contents
            for ( var i = 0; i < linkslist.length; i++ )
```

Stores the document's `links` collection in variable `linkslist`

Number of elements in the collection

Using the **links** collection.

```
32            {
33               var currentLink = linkslist[ i ];
34               contents += "<span class = 'link'>" +
35                  currentLink.innerHTML.link( currentLink.href ) +
36                  "</span> | ";
37            } // end for
38
39            document.getElementById( "links" ).innerHTML = contents;
40         } // end function processlinks
41         // -->
42      </script>
43   </head>
44   <body onload = "processlinks()">
45      <h1>Deitel Resource Centers</h1>
46      <p><a href = "http://www.deitel.com/">Deitel's website</a> contains
47         a rapidly growing
48         <a href = "http://www.deitel.com/ResourceCenters.html">list of
49         Resource Centers</a> on a wide range of topics. Many Resource
50         centers related to topics covered in this book,
51         <a href = "http://www.deitel.com/iw3htp4">Internet and World Wide
52         Web How to Program, 4th Edition</a>. We have Resouce Centers on
53         <a href = "http://www.deitel.com/Web2.0">Web 2.0</a>,
54         <a href = "http://www.deitel.com/Firefox">Firefox</a> and
55         <a href = "http://www.deitel.com/IE7">Internet Explorer 7</a>,
56         <a href = "http://www.deitel.com/XHTML">XHTML</a>, and
57         <a href = "http://www.deitel.com/JavaScript">JavaScript</a>.
58         Watch the list of Deitel Resource Centers for related new
59         Resource Centers.</p>
60      <div id = "links" class = "links"></div>
61   </body>
62 </html>
```

Stores the current link in `currentLink`

Uses the `link` method to create an anchor element with proper text and `href` attribute

Puts all links in one location by inserting them into an empty `div` element

The document's links

Backwards Compatibility

- Although most browsers fully support the DOM, some do not support it completely.
- Browser sniffing is too convoluted, so best to check for specific features
- Put this line of code at the beginning of a function

```
if (!document.getElementsByTagName) return false;
```

- So, if the browser does not support this method the function will stop