HTML5, CSS3, and JavaScript 6th Edition

Tutorial 12
Working with Document Nodes and Style Sheets

Carey

Objectives

- Explore nodes and the node tree
- Create element and text nodes
- Append nodes to a web document
- Work with the properties and methods of element nodes
- Create attribute nodes

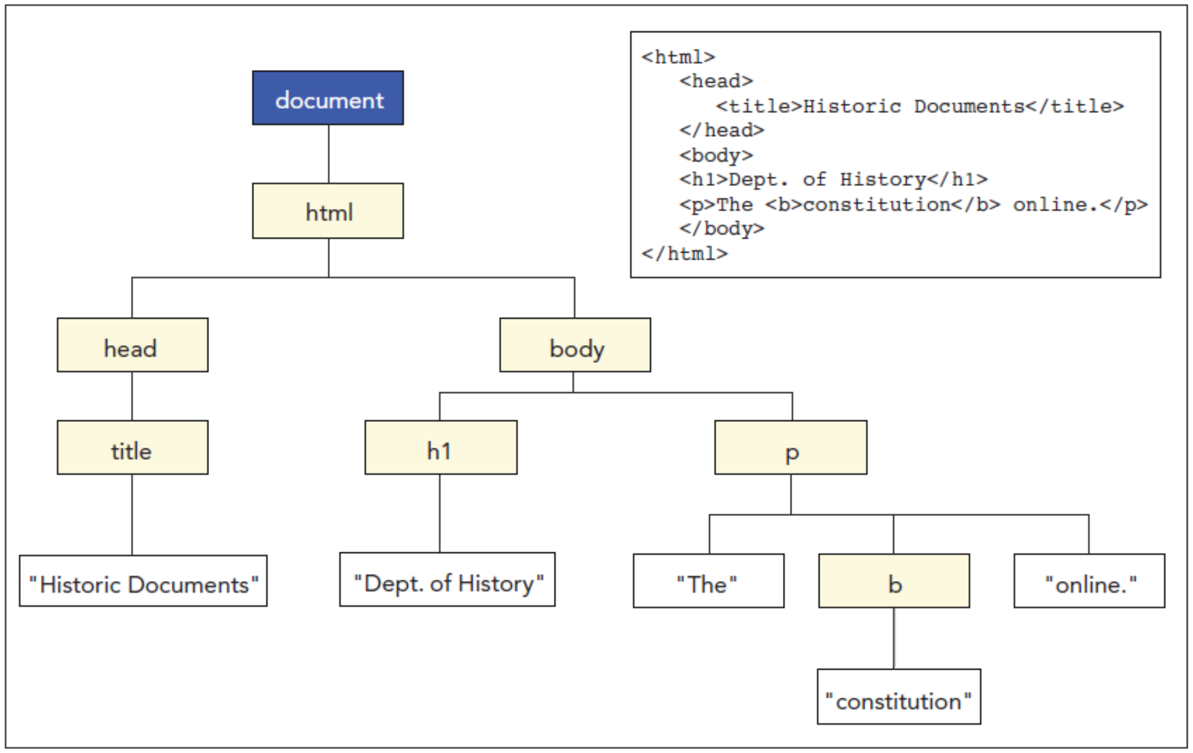
Objectives (continued)

- Add attribute nodes to page elements
- Create external and embedded style sheets with JavaScript
- Add style sheets to a web document
- Create a style rule for an embedded style sheet
- Enable and disable style sheets

Nodes and Document Structure

- **Node:** Any object within an HTML file
- **Node tree:** Hierarchical structure of nodes

Figure 12-1 A sample node tree



```
<html>
   <head>
      <title>Historic Documents</title>
   </head>
   <body>
   <h1>Dept. of History</h1>
   <p>The <b>constitution</b> online.</p>
   </body>
</html>
```

© 2016 Cengage Learning

Nodes and Document Structure (continued 1)

- Nodes in the node tree have a familial relationship
- Each node can be a parent, child, and/or sibling of other nodes
- Node relationship is indicated using the following expression:

```
    node.relationship
```

  where *`node`* is a currently selected node and *`relationship`* is the relationship of another node to the current node

Nodes and Document Structure (continued 2)

- **Root node** is the base node from which all other nodes originate
- Root node is an elementary node in the hierarchy of a node tree
- Each node can contain one or more **child nodes** to transfer the required information

Nodes and Document Structure (continued 3)

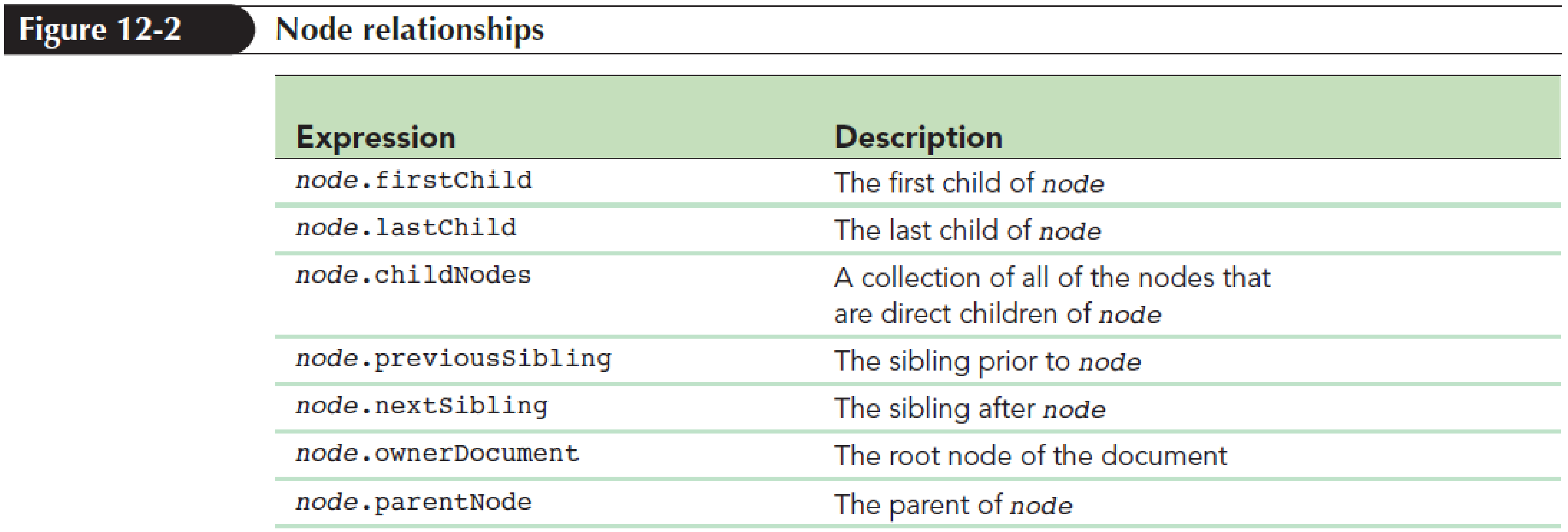


Figure 12-2 Node relationships

Expression	Description
`node.firstChild`	The first child of *node*
`node.lastChild`	The last child of *node*
`node.childNodes`	A collection of all of the nodes that are direct children of *node*
`node.previousSibling`	The sibling prior to *node*
`node.nextSibling`	The sibling after *node*
`node.ownerDocument`	The root node of the document
`node.parentNode`	The parent of *node*

Nodes and Document Structure (continued 4)

Figure 12-3 Element node relationships

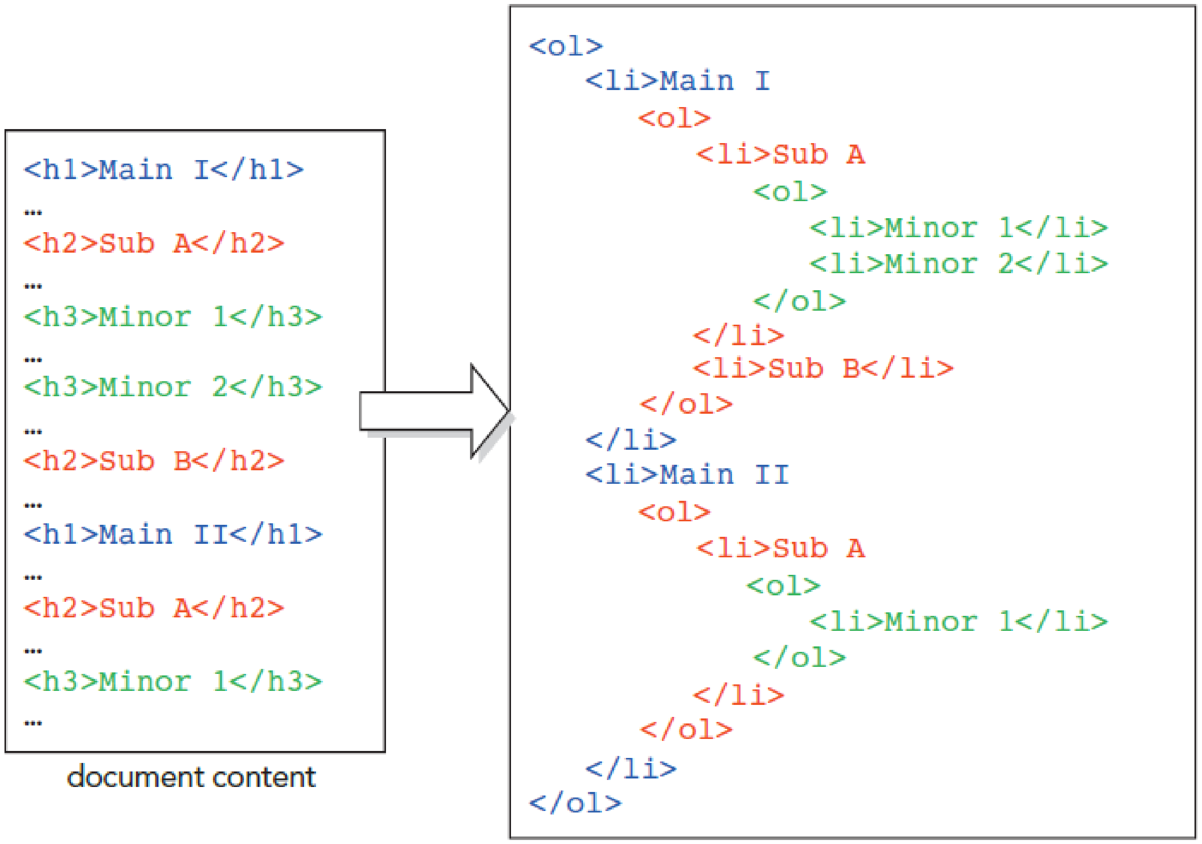
Property	Description
`node.children`	A collection of all of the element nodes contained within *node*
`node.firstElementChild`	The first element within *node*
`node.lastElementChild`	The last element within *node*
`node.nextElementSibling`	The next sibling element to *node*
`node.previousElementSibling`	The previous sibling element to *node*
`node.childElementCount`	The number of child elements within *node*
`node.children.length`	The number of child elements within *node*

Restructuring the Node Tree

- Working with nodes is better than writing HTML code through the `innerHTML` property
- Node tree can be rearranged to form a new structure
- Restructuring a node tree helps in ordering an HTML content with more accuracy

Restructuring the Node Tree (continued 1)

Figure 12-5 Document headings restructured as an outline



```
<h1>Main I</h1>
…
<h2>Sub A</h2>
…
<h3>Minor 1</h3>
…
<h3>Minor 2</h3>
…
<h2>Sub B</h2>
…
<h1>Main II</h1>
…
<h2>Sub A</h2>
…
<h3>Minor 1</h3>
…
```

document content

```
<ol>
   <li>Main I
      <ol>
         <li>Sub A
            <ol>
               <li>Minor 1</li>
               <li>Minor 2</li>
            </ol>
         </li>
         <li>Sub B</li>
      </ol>
   </li>
   <li>Main II
      <ol>
         <li>Sub A
            <ol>
               <li>Minor 1</li>
            </ol>
         </li>
      </ol>
   </li>
</ol>
```

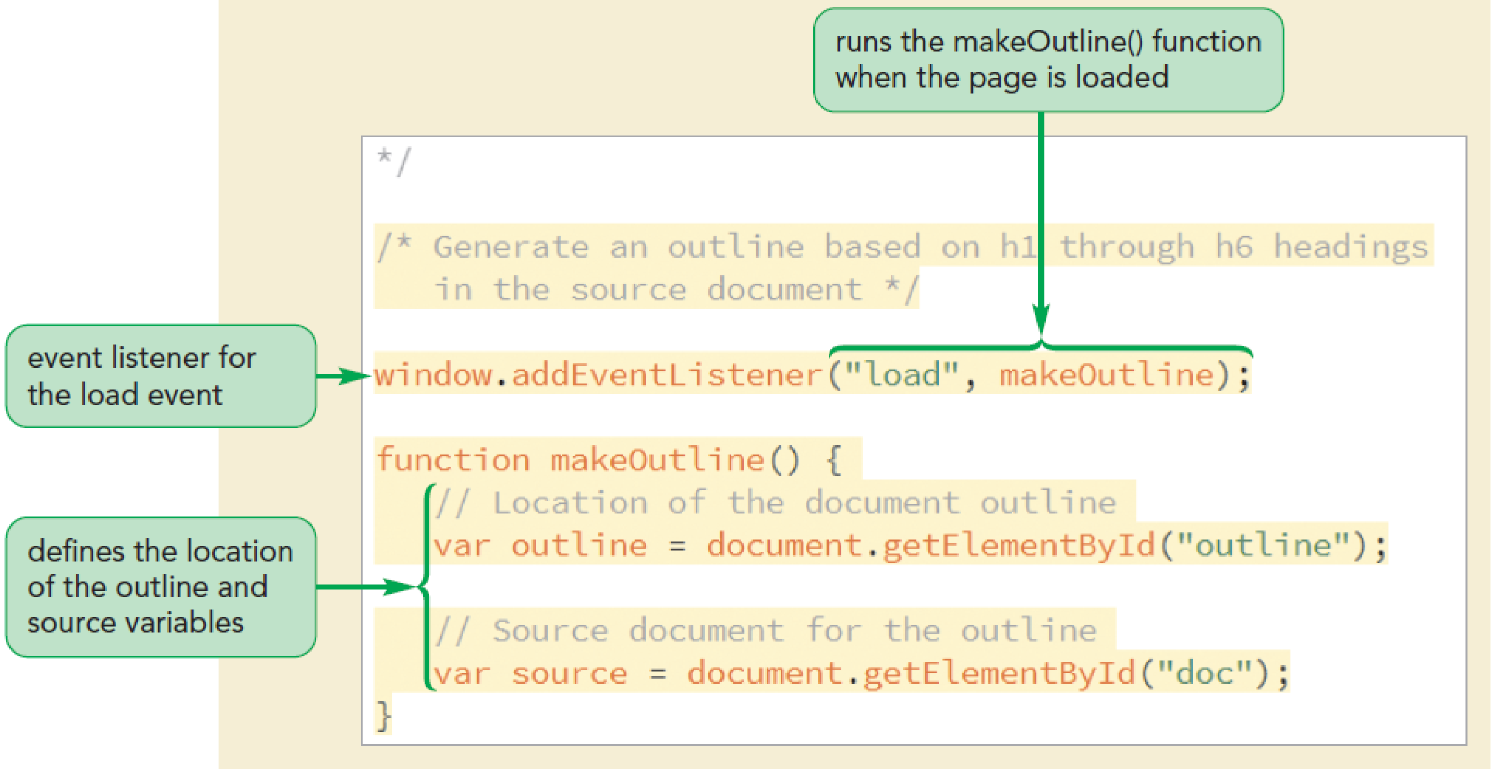
nested list generated by the outlining function

© 2016 Cengage Learning

Restructuring the Node Tree (continued 2)

- makeOutline() function is used to generate the code for the document outline

Figure 12-6 Adding the makeOutline() function


runs the makeOutline() function when the page is loaded
event listener for the load event
defines the location of the outline and source variables

```
*/

/* Generate an outline based on h1 through h6 headings
   in the source document */

window.addEventListener("load", makeOutline);

function makeOutline() {
   // Location of the document outline
   var outline = document.getElementById("outline");

   // Source document for the outline
   var source = document.getElementById("doc");
}
```

Creating and Appending Nodes

- JavaScript's document object model supports several methods to create nodes of different types

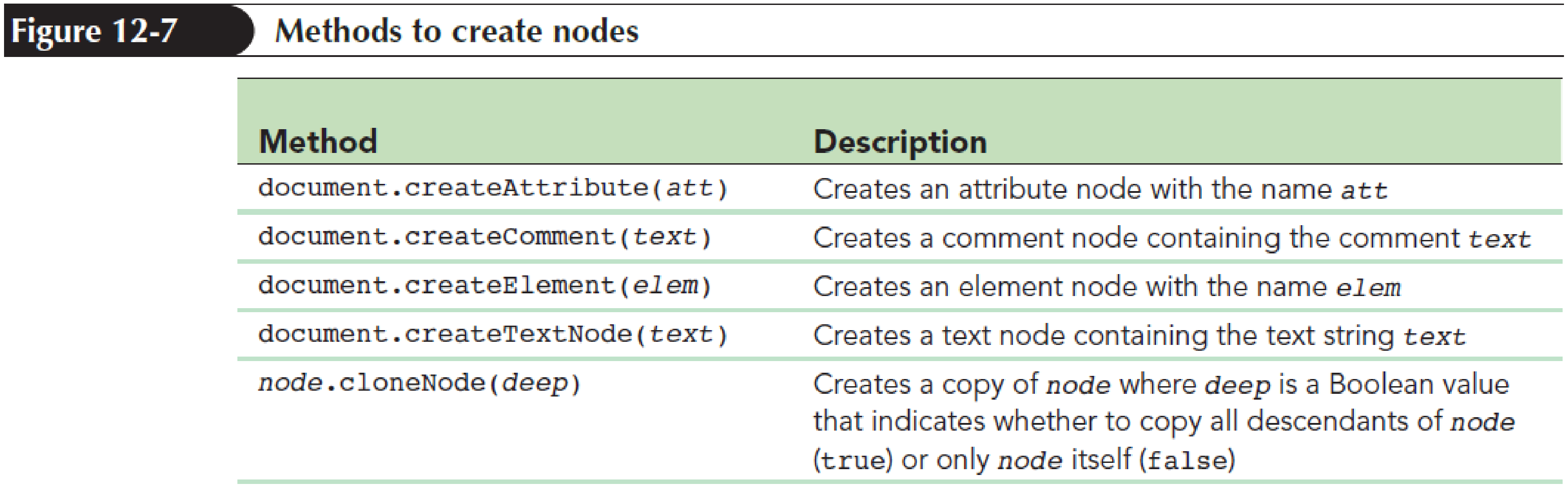


Figure 12-7 Methods to create nodes

Method	Description
document.createAttribute(*att*)	Creates an attribute node with the name *att*
document.createComment(*text*)	Creates a comment node containing the comment *text*
document.createElement(*elem*)	Creates an element node with the name *elem*
document.createTextNode(*text*)	Creates a text node containing the text string *text*
node.cloneNode(*deep*)	Creates a copy of *node* where *deep* is a Boolean value that indicates whether to copy all descendants of *node* (true) or only *node* itself (false)

Creating and Appending Nodes (continued 1)

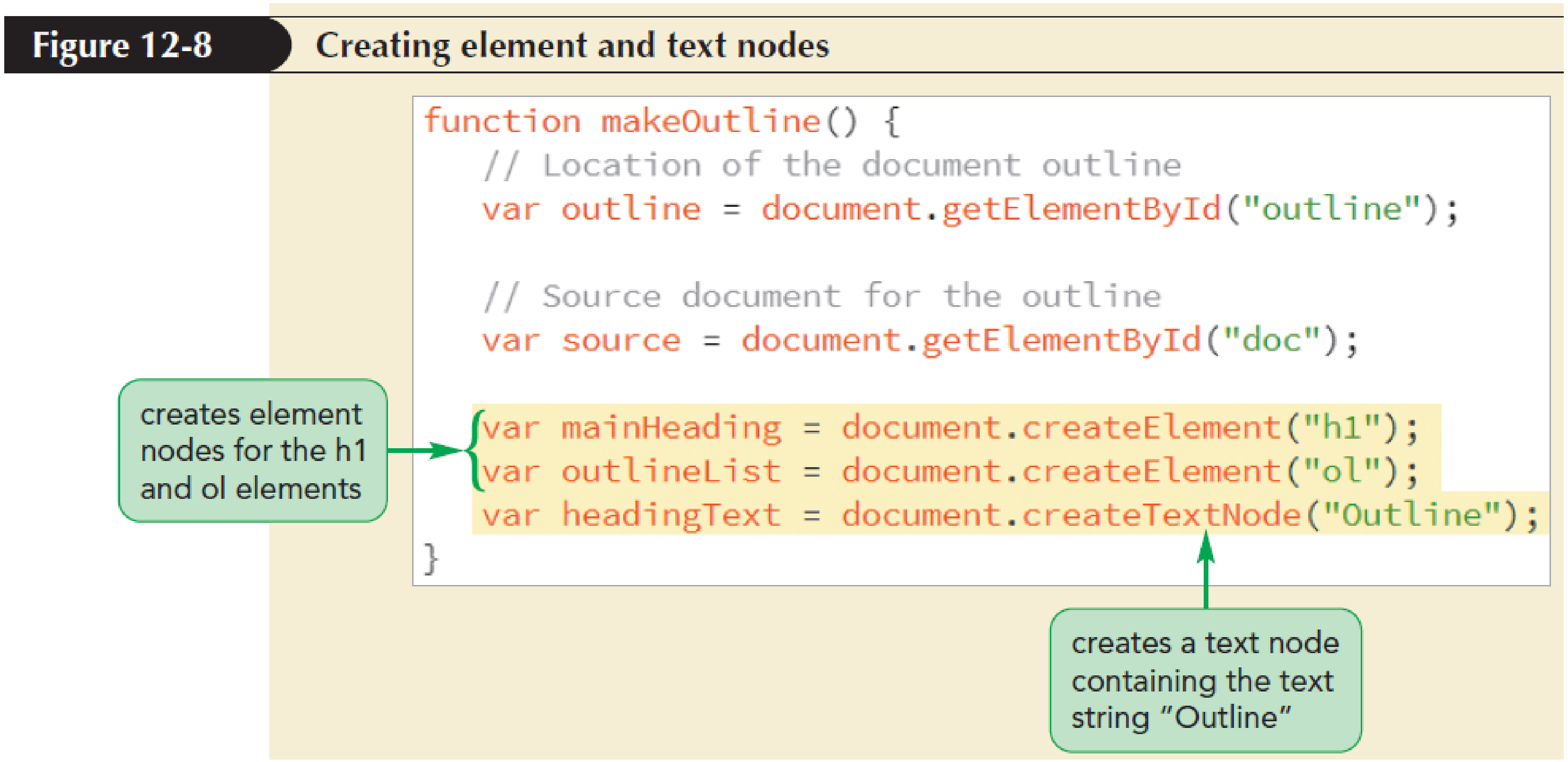


Figure 12-8 Creating element and text nodes

```
function makeOutline() {
   // Location of the document outline
   var outline = document.getElementById("outline");

   // Source document for the outline
   var source = document.getElementById("doc");

   var mainHeading = document.createElement("h1");
   var outlineList = document.createElement("ol");
   var headingText = document.createTextNode("Outline");
}
```

creates element nodes for the h1 and ol elements
creates a text node containing the text string "Outline"

Creating and Appending Nodes (continued 2)

- Nodes that are created are added to the computer memory as a **document fragment**
- Document fragment is not a part of the document’s node tree until it is appended to a node within the tree

Creating and Appending Nodes (continued 3)

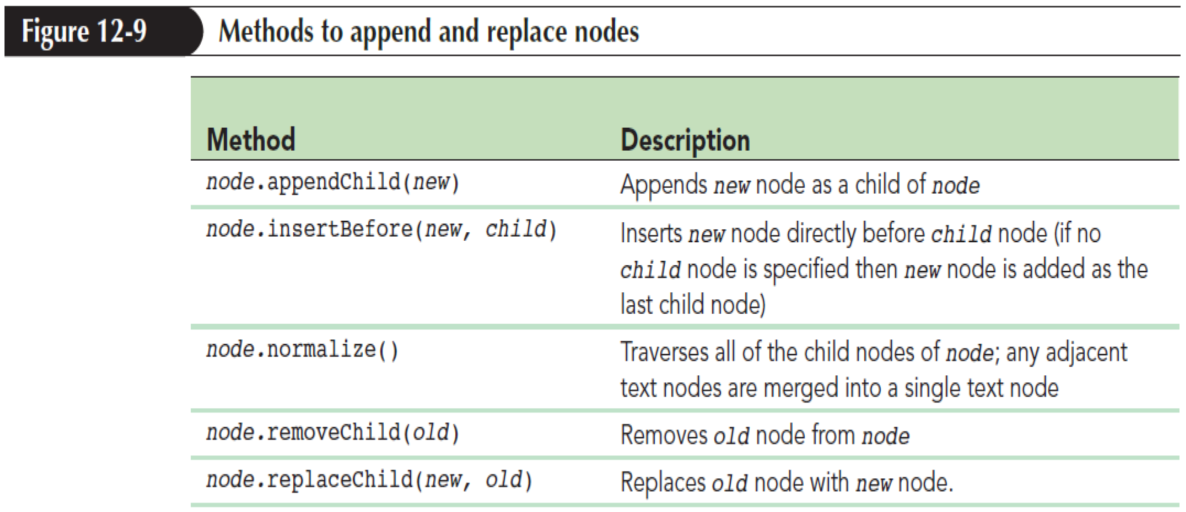


Figure 12-9 Methods to append and replace nodes

Method	Description
node.appendChild(*new*)	Appends *new* node as a child of *node*
node.insertBefore(*new*, *child*)	Inserts *new* node directly before *child* node (if no *child* node is specified then *new* node is added as the last child node)
node.normalize()	Traverses all of the child nodes of *node*; any adjacent text nodes are merged into a single text node
node.removeChild(*old*)	Removes *old* node from *node*
node.replaceChild(*new*, *old*)	Replaces *old* node with *new* node.

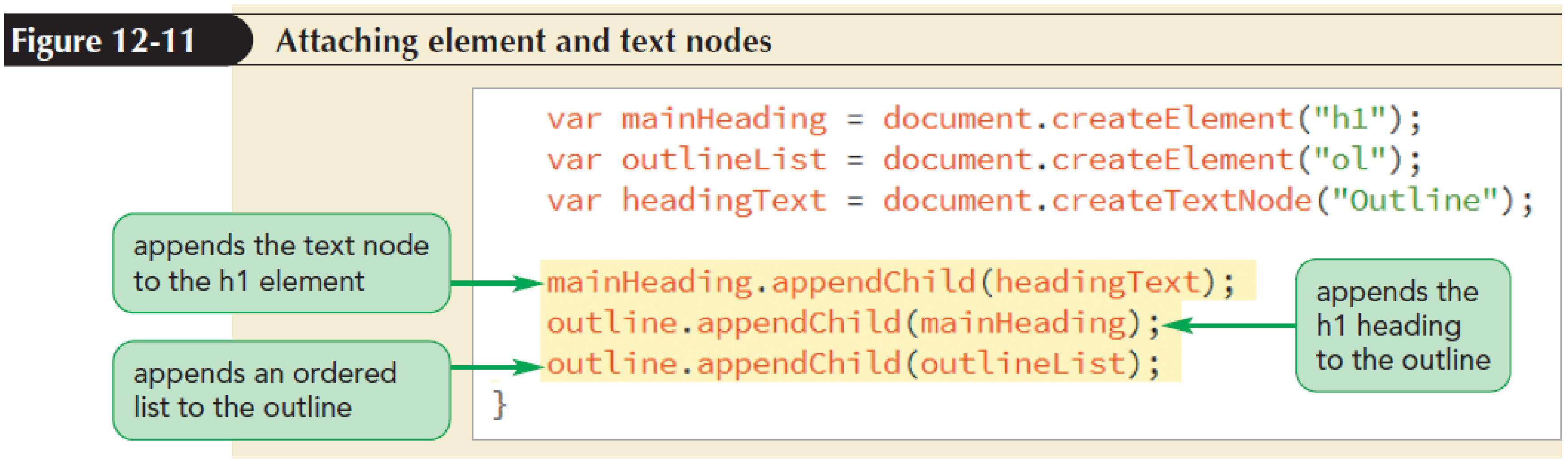


Figure 12-11 Attaching element and text nodes

```
   var mainHeading = document.createElement("h1");
   var outlineList = document.createElement("ol");
   var headingText = document.createTextNode("Outline");

   mainHeading.appendChild(headingText);
   outline.appendChild(mainHeading);
   outline.appendChild(outlineList);
}
```

appends the text node to the h1 element
appends an ordered list to the outline
appends the h1 heading to the outline

Working with Node Types, Names, and Values

- Text for each entry in the outline list should match the text of a heading in the source article
- To maintain uniformity throughout a document, outlining app does the following:
  - Loops through the child nodes of the source article

Working with Node Types, Names, and Values (continued)

- Tests whether each child node represents an h1 through h6 element node
- Extracts the element's text if it is a heading and creates a list item containing that same text string
- Appends the list item as a new child of the outline's ordered list

Looping through the Child Nodes Collection

- Use a counter variable starting with a value of 0 and increase by 1 for each node, up to the length of the `childNodes` collection

```
for (var i = 0;
        i <node.childNodes.length;
        i++)
{
        Commands
}
```

Looping through the Child Nodes Collection (continued 1)

- Object reference for child nodes in the `for` loop is

  `node.childNodes[i]`

  where *node* is the parent node of the child nodes collection, and *i* is the value of the counter variable in the `for` loop

Looping through the Child Nodes Collection (continued 2)

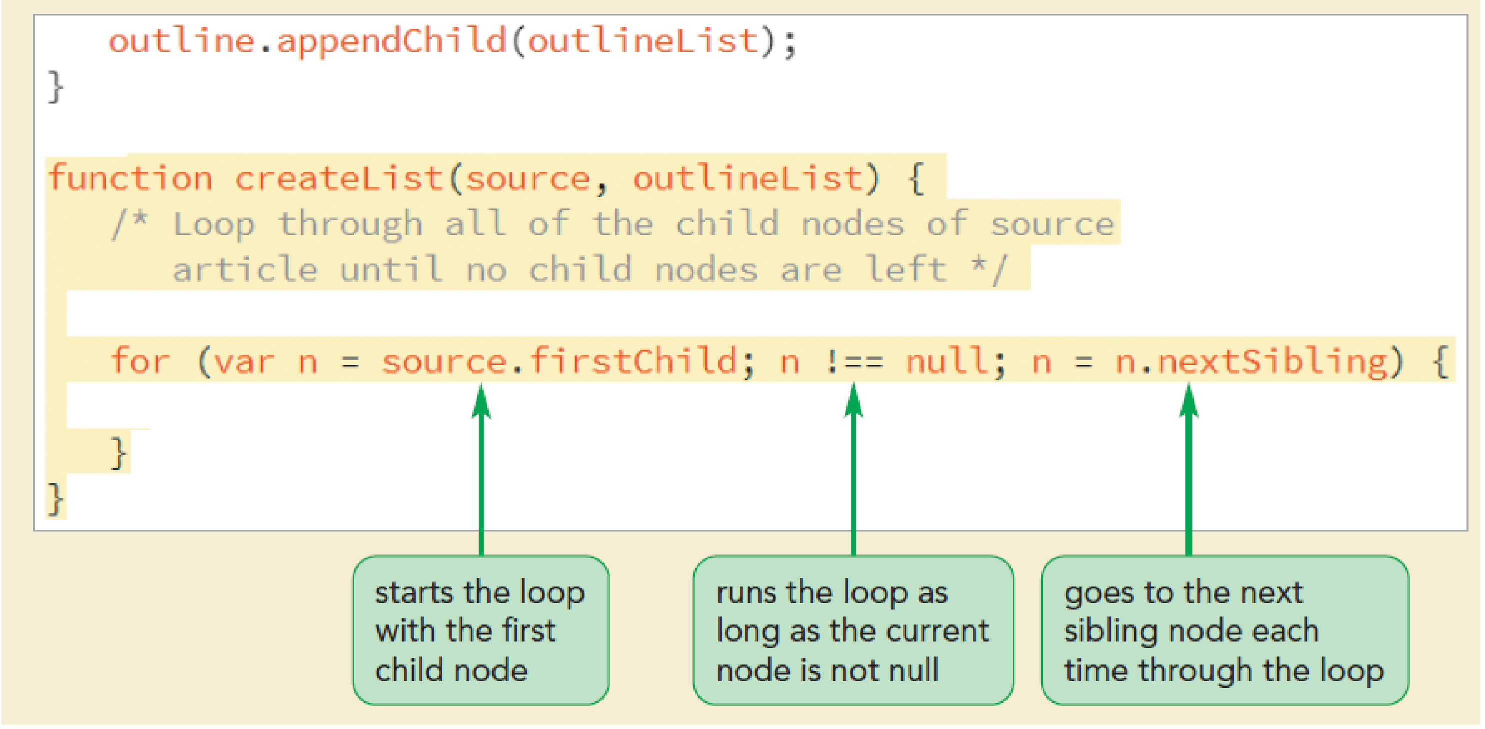
- Use familial references, starting with the first child of the parent node and traverse each subsequent sibling until no siblings remain

```
for (var n = node.firstChild; n !==
null; n = n.nextSibling) {
        commands
}
```

- Object reference for child nodes in the `for` loop is *n* variable defined within the `for` loop expression

Looping through the Child Nodes Collection (continued 3)

Figure 12-14 Using sibling nodes in a for loop



```
   outline.appendChild(outlineList);
}

function createList(source, outlineList) {
   /* Loop through all of the child nodes of source
      article until no child nodes are left */

   for (var n = source.firstChild; n !== null; n = n.nextSibling) {

   }
}
```

starts the loop with the first child node
runs the loop as long as the current node is not null
goes to the next sibling node each time through the loop

Node Properties

- The node has the following properties:
  - *node*`.nodeType`: It indicates the type of node
  - *node*`.nodeName`: It indicates the node name
  - *node*`.nodeValue`: It indicates the node value for a *node* in the document
- To check whether a node refers to an element, attribute, text string, comment, document, or any other type of node, use

  node`.nodeType`

Node Properties (continued 1)

- To retrieve the name of a node within a document, use

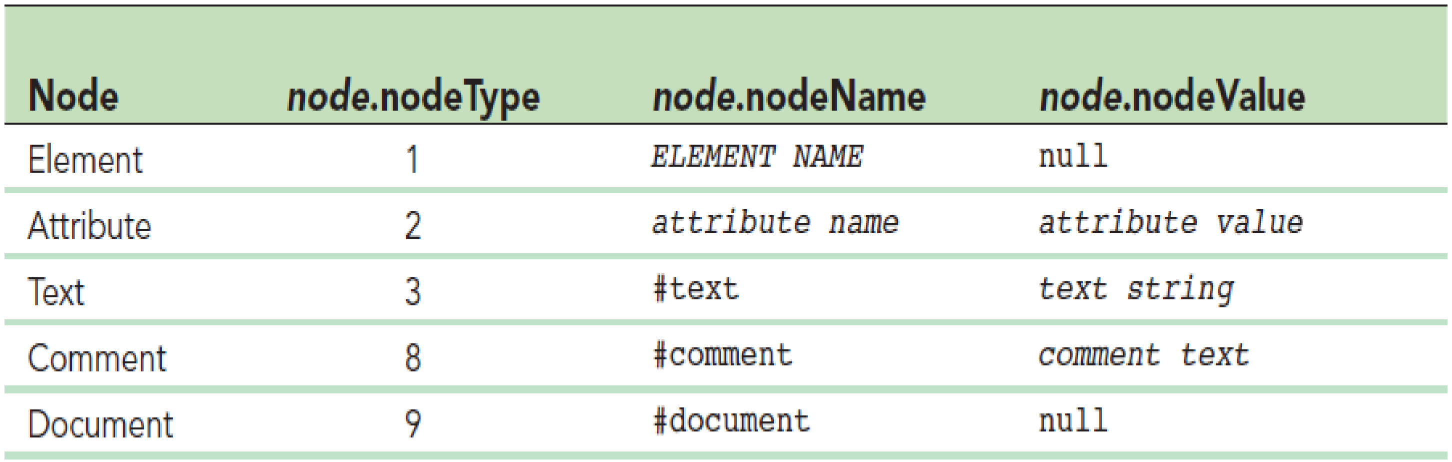
  `node``.nodeName`

- To retrieve a node's value, use

  `node``.nodeValue`

Node Properties (continued 2)

Figure 12-15 Node types, names, and values



Node	***node*.nodeType**	***node*.nodeName**	***node*.nodeValue**
Element	1	*ELEMENT NAME*	null
Attribute	2	*attribute name*	*attribute value*
Text	3	#text	*text string*
Comment	8	#comment	*comment text*
Document	9	#document	null

Node Properties (continued 3)

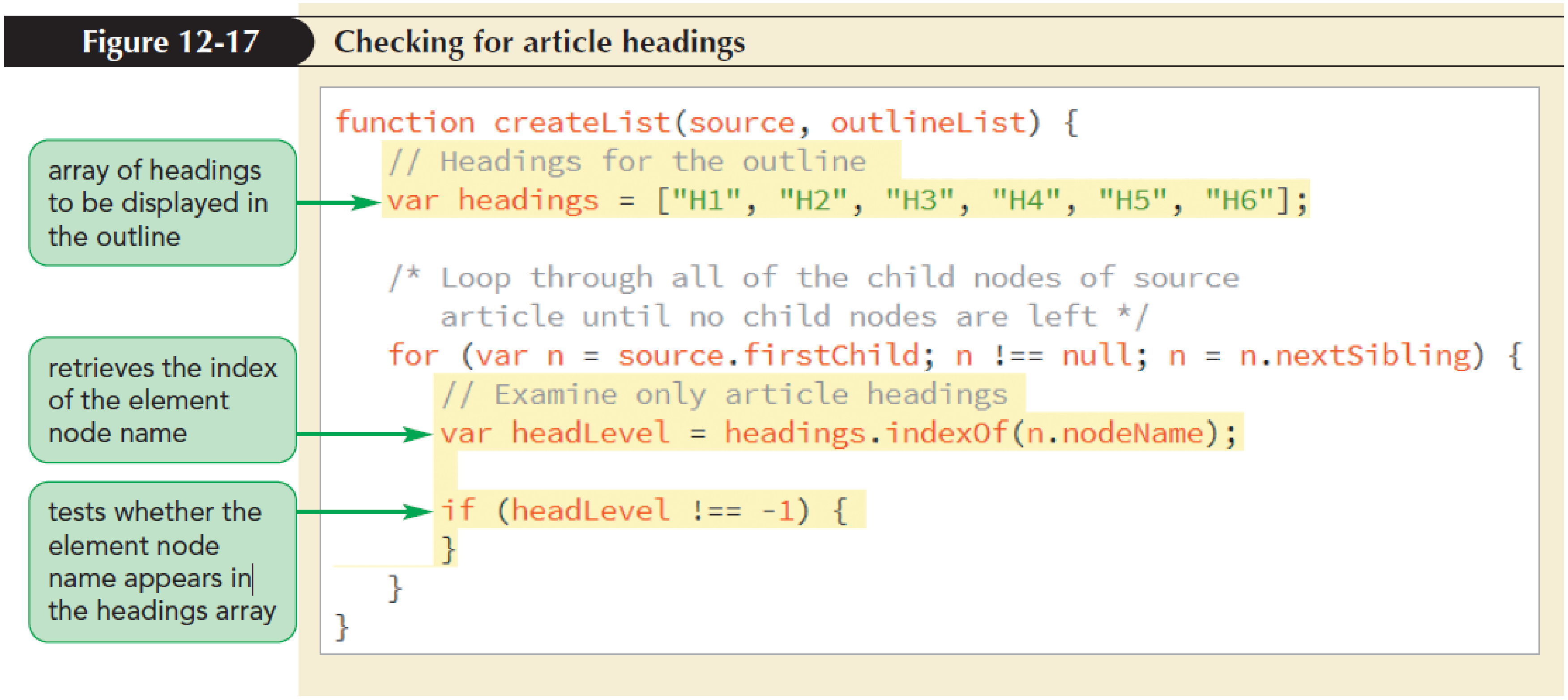


Figure 12-17 Checking for article headings

```
function createList(source, outlineList) {
   // Headings for the outline
   var headings = ["H1", "H2", "H3", "H4", "H5", "H6"];

   /* Loop through all of the child nodes of source
      article until no child nodes are left */
   for (var n = source.firstChild; n !== null; n = n.nextSibling) {
      // Examine only article headings
      var headLevel = headings.indexOf(n.nodeName);

      if (headLevel !== -1) {
      }
   }
}
```

array of headings to be displayed in the outline
retrieves the index of the element node name
tests whether the element node name appears in the headings array

Node Properties (continued 4)

- Text node can be referenced within an element in order to access the text contained within the element
- Example:

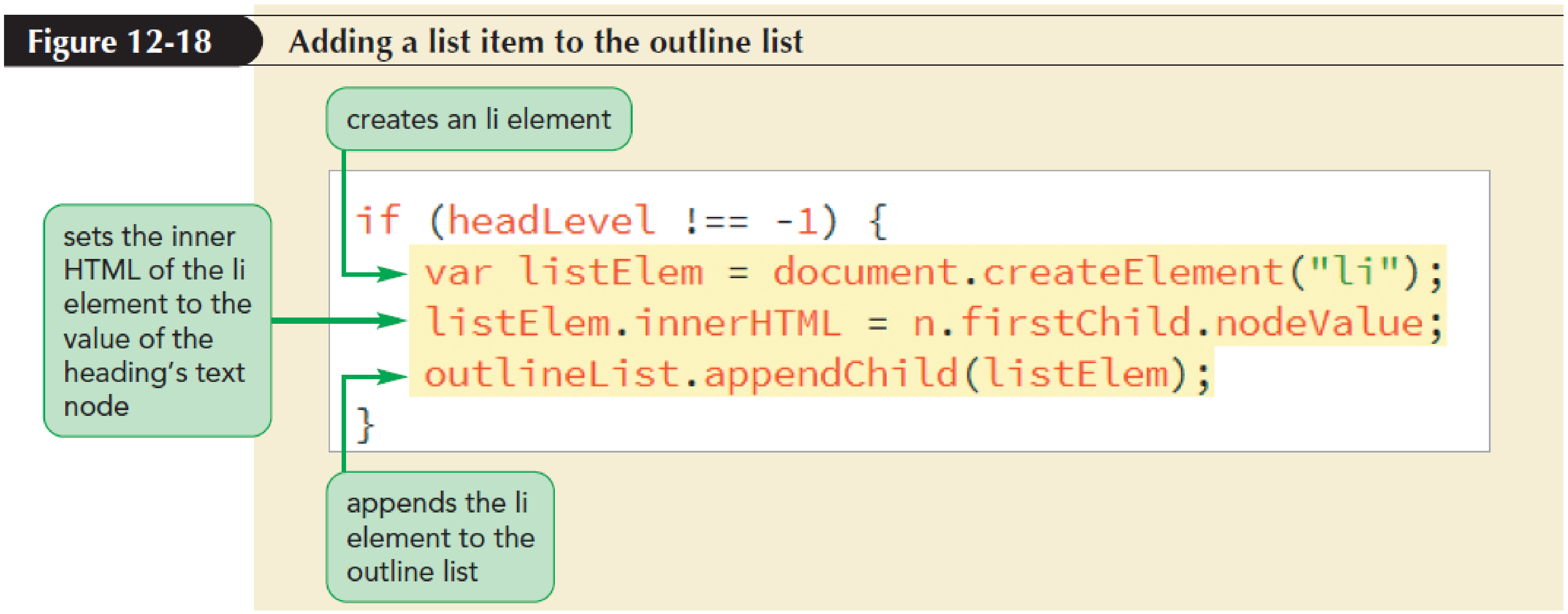
```
<h1 id=”title”>History Online</h1>
```

Text of the above `h1` heading is obtained by

```
document.getElementById(“title”).
firstChild.nodeValue;
```

Node Properties (continued 5)

Figure 12-18 Adding a list item to the outline list

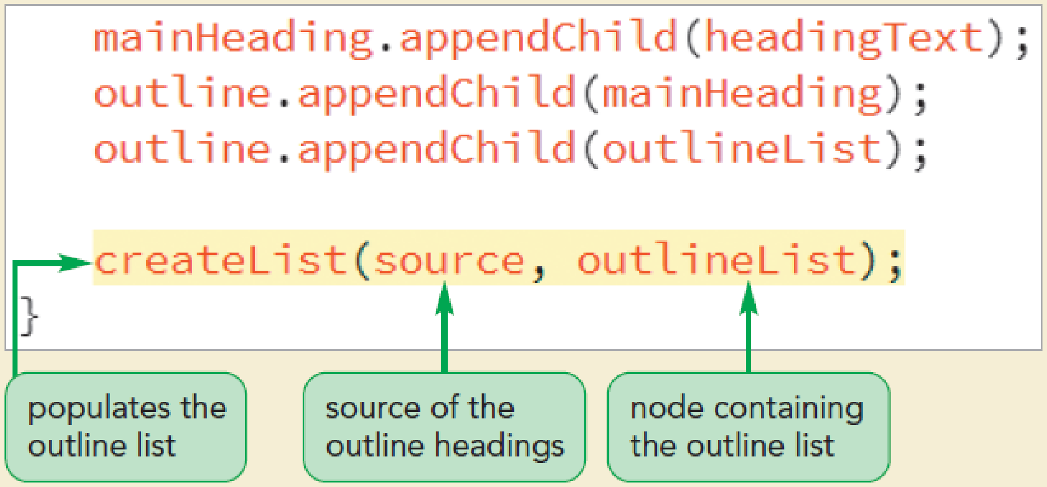


```
if (headLevel !== -1) {
   var listElem = document.createElement("li");
   listElem.innerHTML = n.firstChild.nodeValue;
   outlineList.appendChild(listElem);
}
```

creates an li element
sets the inner HTML of the li element to the value of the heading's text node
appends the li element to the outline list

Node Properties (continued 6)

Figure 12-19 Displaying the outline

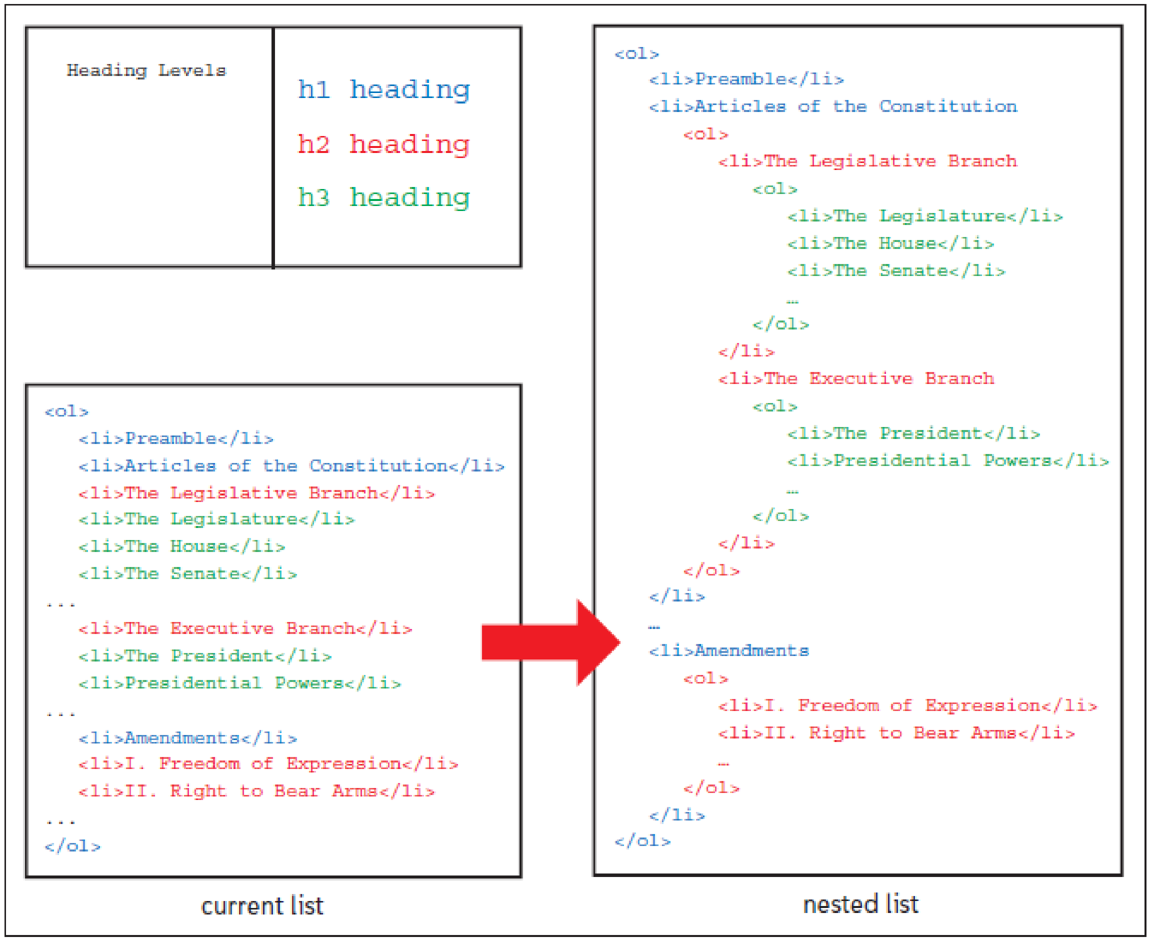


```
   mainHeading.appendChild(headingText);
   outline.appendChild(mainHeading);
   outline.appendChild(outlineList);

   createList(source, outlineList);
}
```

populates the outline list
source of the outline headings
node containing the outline list

Creating a Nested List

Figure 12-21 Creating a nested list



```
<ol>
   <li>Preamble</li>
   <li>Articles of the Constitution</li>
   <li>The Legislative Branch</li>
   <li>The Legislature</li>
   <li>The House</li>
   <li>The Senate</li>
...
   <li>The Executive Branch</li>
   <li>The President</li>
   <li>Presidential Powers</li>
...
   <li>Amendments</li>
   <li>I. Freedom of Expression</li>
   <li>II. Right to Bear Arms</li>
...
</ol>
```

current list

```
<ol>
   <li>Preamble</li>
   <li>Articles of the Constitution
      <ol>
         <li>The Legislative Branch
            <ol>
               <li>The Legislature</li>
               <li>The House</li>
               <li>The Senate</li>
               …
            </ol>
         </li>
         <li>The Executive Branch
            <ol>
               <li>The President</li>
               <li>Presidential Powers</li>
               …
            </ol>
         </li>
      </ol>
   </li>
   …
   <li>Amendments
      <ol>
         <li>I. Freedom of Expression</li>
         <li>II. Right to Bear Arms</li>
         …
      </ol>
   </li>
</ol>
```

nested list

© 2016 Cengage Learning

Creating a Nested List (continued 1)

- In a nested list, lower-level headings are nested within upper-level headings
- All `h1` heading are placed at the top most level of the table of contents, all `h2` headings are placed at the next lower level, and so forth
- `indexOf()`function is used to determine the level of each list item

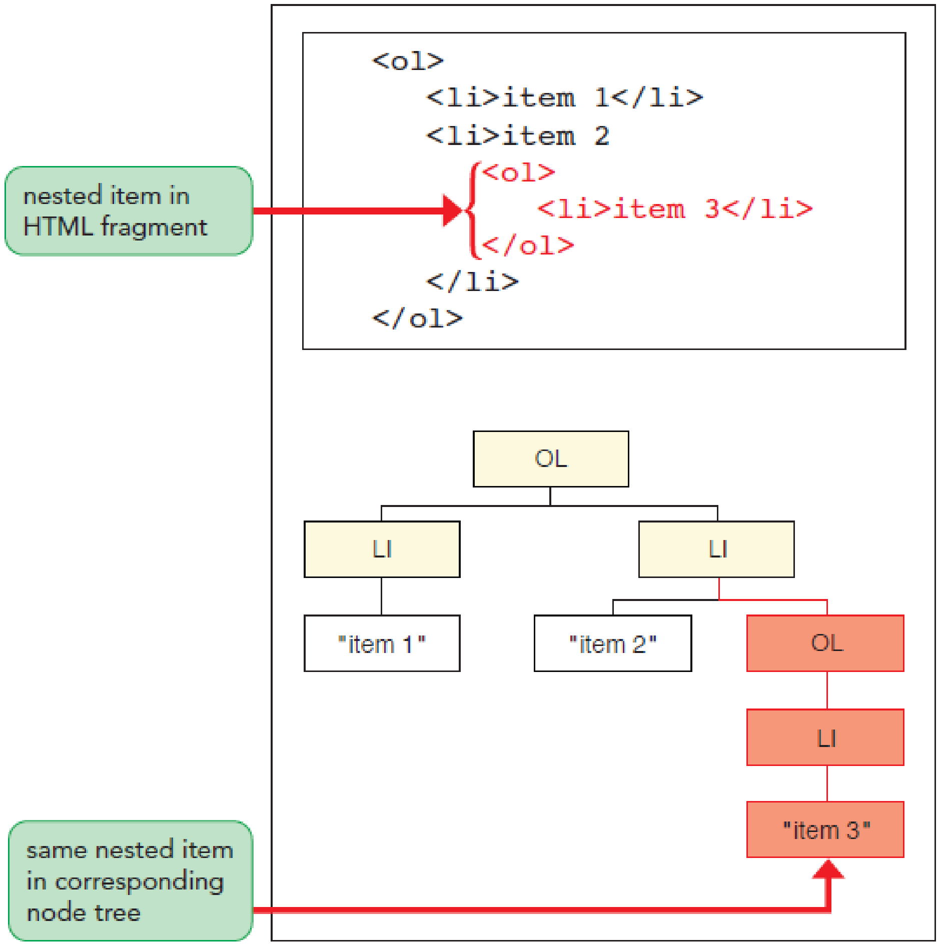
Creating a Nested List (continued 2)

- To test the three possibilities for the comparison of the index numbers, use:

```
if (headLevel === prevLevel) {
   // Append the list item to the
current list
} else if (headLevel > prevLevel) {
   // Start a new nested list
} else {
   // Append the entry to a higher list
}
```

Creating a Nested List (continued 3)

Figure 12-25 Appending a nested list



```
<ol>
   <li>item 1</li>
   <li>item 2
      <ol>
         <li>item 3</li>
      </ol>
   </li>
</ol>
```

nested item in HTML fragment
OL
LI
LI
"item 1"
"item 2"
OL
LI
"item 3"
same nested item in corresponding node tree

© 2016 Cengage Learning

Creating a Nested List (continued 4)

- In JavaScript, following commands are used to append a nested list:

```
var nestedList =
document.createElement(“ol”);
NestedList.appendChild(listElem);
outlineList.lastChild.appendChild(ne
stedList);
outlineList = nestedList;
```

Creating a Nested List (continued 5)

- Use the following command to calculate the difference between the index number of the previous heading and the current heading:

```
var levelUp = prevLevel - headLevel;
```

- Use the `parentNode` property twice for each difference in level to move up the node tree, as shown below:

```
for (var i = 1; i <= levelUp; i++) {
outlineList =
outlineList.parentNode.parentNode;}
```

Creating a Nested List (continued 6)

```
for (var i = 1; i <= levelUp; i++)
{
outlineList =
outlineList.parentNode.parentNode;
}
```

- Use the following command to append a list item to a higher-order list:

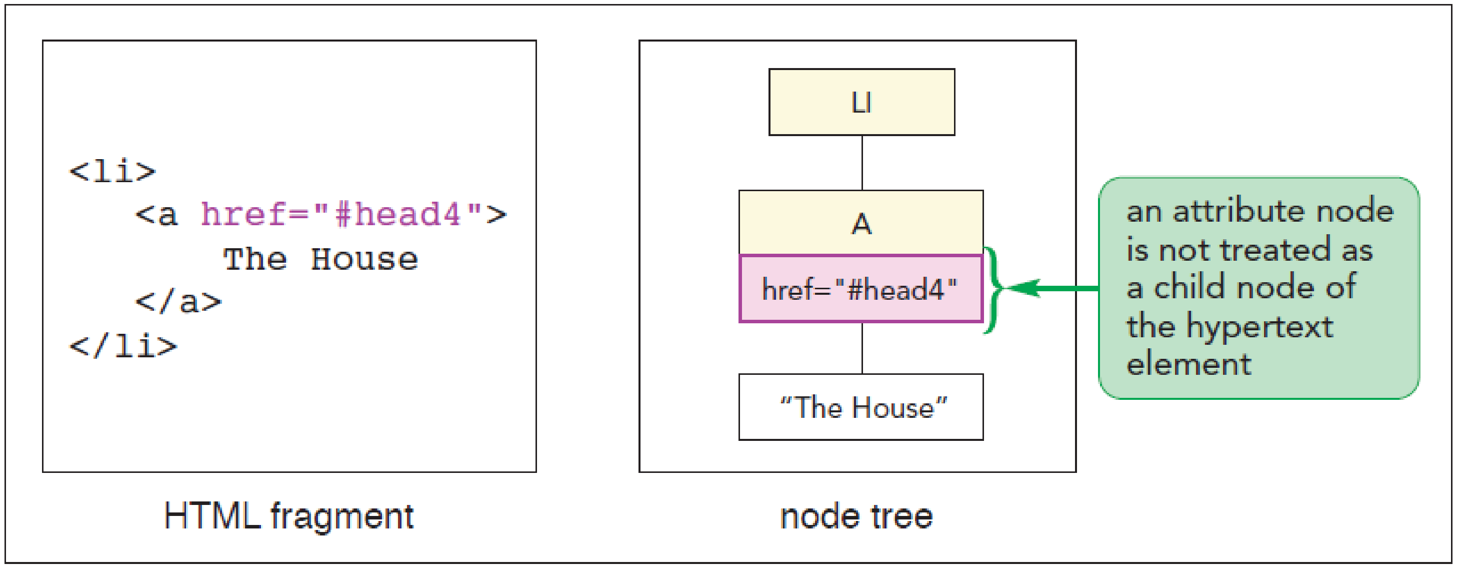
```
outlineList.appendChild(listElem);
```

Working with Attribute Nodes

- Attribute node contains an attribute that can be attached to an element node
- Attribute nodes are not part of the node tree because they are not the child or parent of any node
- Attribute nodes cannot be placed in a node hierarchy

Working with Attribute Nodes (continued 1)

Figure 12-31 Attribute nodes



```
<li>
   <a href="#head4">
       The House
   </a>
</li>
```

HTML fragment
LI
A
href="#head4"
"The House"
node tree
an attribute node is not treated as a child node of the hypertext element

© 2016 Cengage Learning

Working with Attribute Nodes (continued 2)

- Document object model supports several methods to create, attach, and set the values of attributes

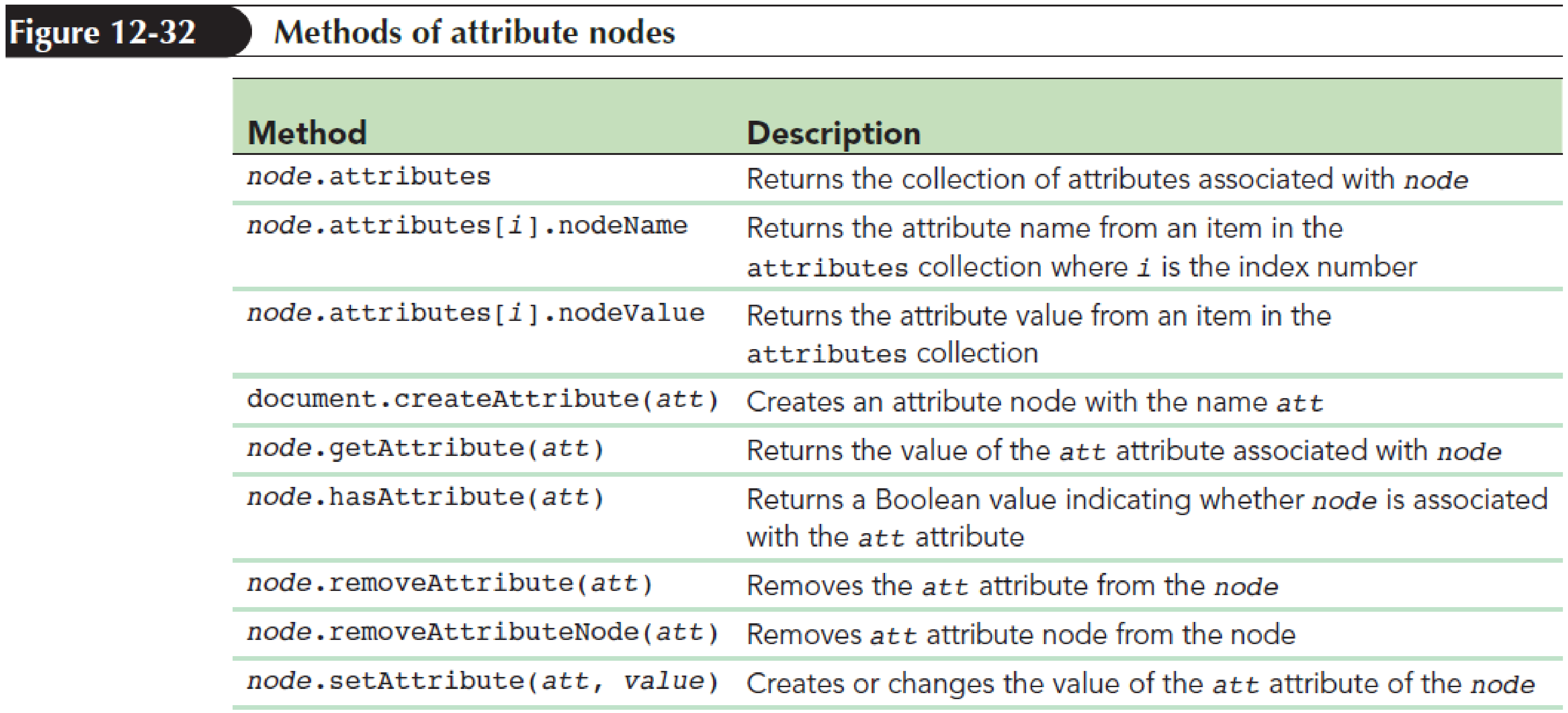


Figure 12-32 Methods of attribute nodes

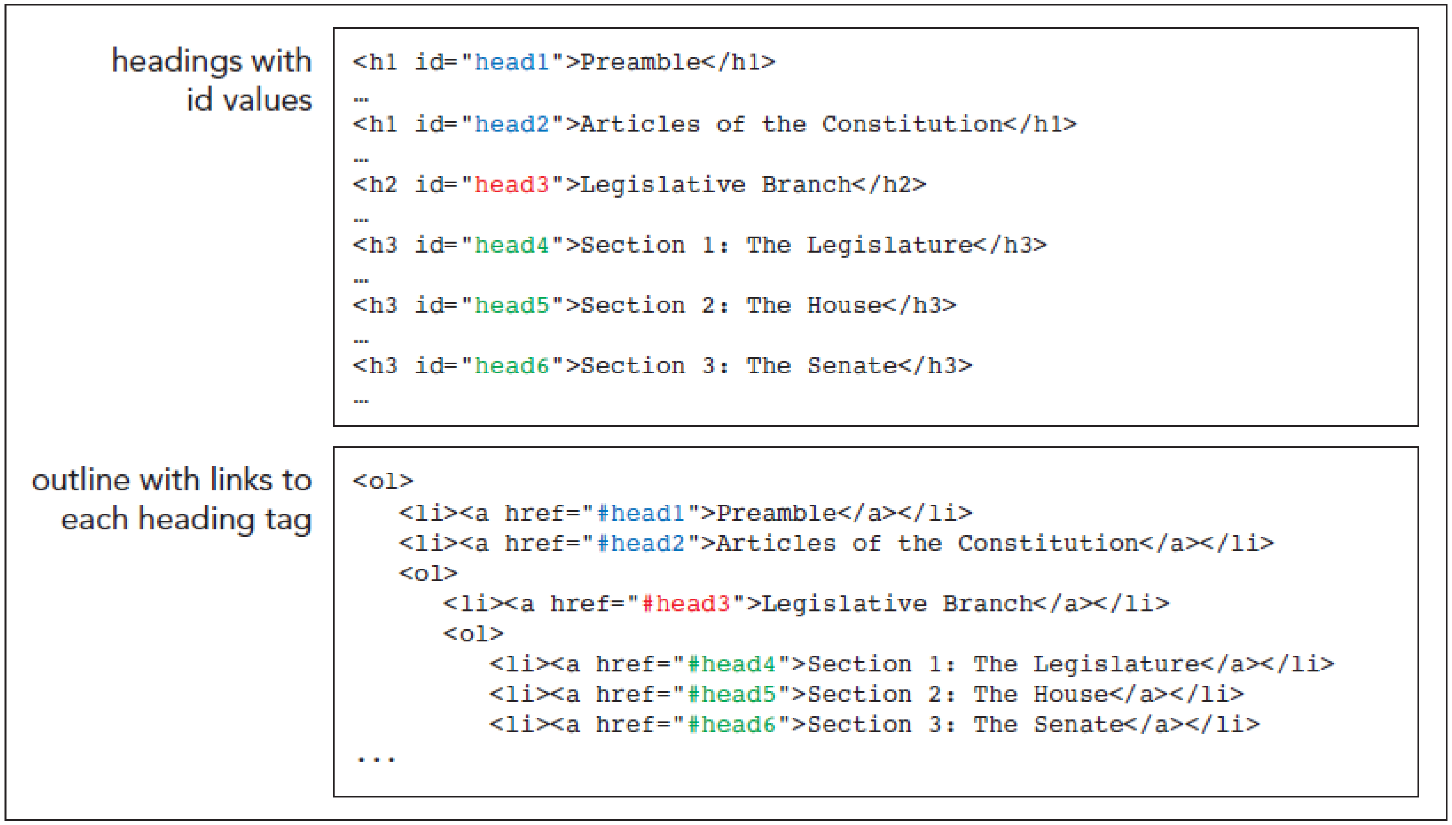
Method	Description
node.attributes	Returns the collection of attributes associated with *node*
node.attributes[*i*].nodeName	Returns the attribute name from an item in the attributes collection where *i* is the index number
node.attributes[*i*].nodeValue	Returns the attribute value from an item in the attributes collection
document.createAttribute(*att*)	Creates an attribute node with the name *att*
node.getAttribute(*att*)	Returns the value of the *att* attribute associated with *node*
node.hasAttribute(*att*)	Returns a Boolean value indicating whether *node* is associated with the *att* attribute
node.removeAttribute(*att*)	Removes the *att* attribute from the *node*
node.removeAttributeNode(*att*)	Removes *att* attribute node from the node
node.setAttribute(*att*, *value*)	Creates or changes the value of the *att* attribute of the *node*

Creating Heading IDs

- Creating hypertext links involves two steps,
  - Adding an ID attribute to each heading (if it doesn’t already have one) to mark its location within the web page.
  - Changing the content of each list item to a hypertext link pointing to the corresponding heading

Creating Heading IDs (continued 1)

Figure 12-33 Linking list items to their headings



headings with id values

```
<h1 id="head1">Preamble</h1>
…
<h1 id="head2">Articles of the Constitution</h1>
…
<h2 id="head3">Legislative Branch</h2>
…
<h3 id="head4">Section 1: The Legislature</h3>
…
<h3 id="head5">Section 2: The House</h3>
…
<h3 id="head6">Section 3: The Senate</h3>
…
```

outline with links to each heading tag

```
<ol>
   <li><a href="#head1">Preamble</a></li>
   <li><a href="#head2">Articles of the Constitution</a></li>
   <ol>
      <li><a href="#head3">Legislative Branch</a></li>
      <ol>
         <li><a href="#head4">Section 1: The Legislature</a></li>
         <li><a href="#head5">Section 2: The House</a></li>
         <li><a href="#head6">Section 3: The Senate</a></li>
...
```

© 2016 Cengage Learning

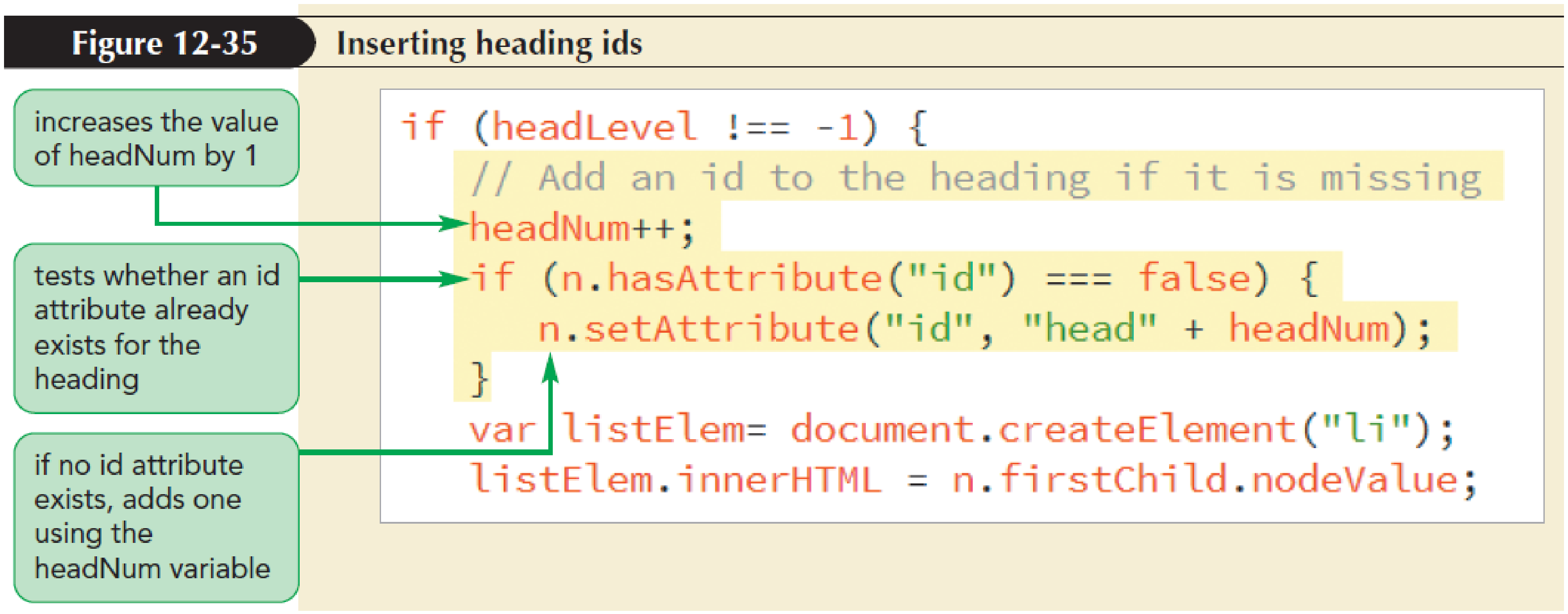
Creating Heading IDs (continued 2)

- Counter variable `headNum` is created to maintain the uniqueness of each ID
- `headNum` is set with initial value = 0, and it increases by 1 for each heading found in the source document
- Use the following code to create the `headNum` variable:

```
    var headNum = 0;
```

Creating Heading IDs (continued 3)

Figure 12-35 Inserting heading ids



```
if (headLevel !== -1) {
   // Add an id to the heading if it is missing
   headNum++;
   if (n.hasAttribute("id") === false) {
      n.setAttribute("id", "head" + headNum);
   }
   var listElem= document.createElement("li");
   listElem.innerHTML = n.firstChild.nodeValue;
```

increases the value of headNum by 1
tests whether an id attribute already exists for the heading
if no id attribute exists, adds one using the headNum variable

Looping through the Attributes Collection

- To examine each HTML attribute associated with an element, use the following code:

```
node.attributes
```

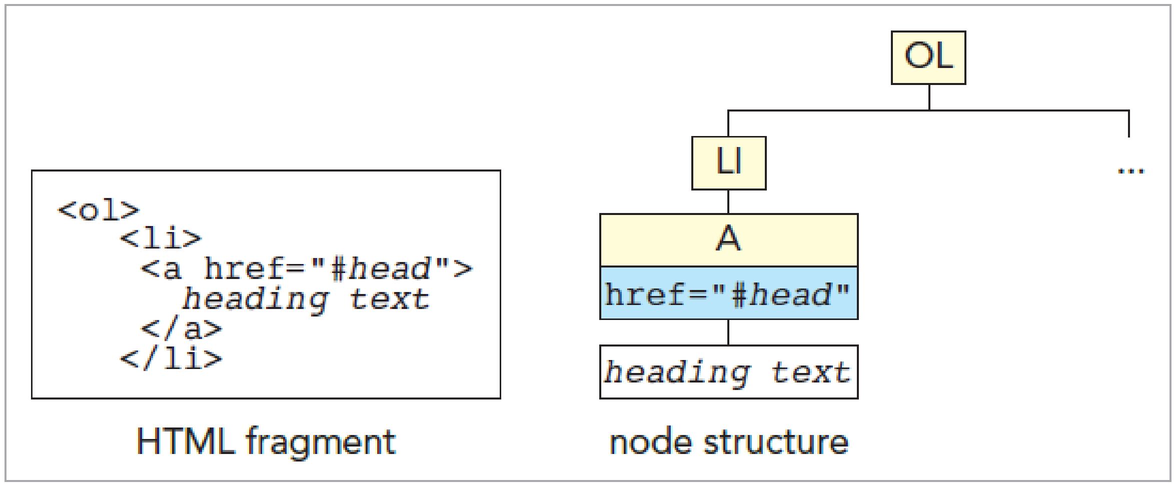
- Named node map is an unordered list of nodes that can be referenced by their names
- Named node map refers to the names of attributes in an attributes collection
- Index numbers are also used to refer to the attributes

Creating Hypertext Links

- Each list item can be changed to a hypertext link pointing to the corresponding heading using the `href` attribute
- Value of the `href` attribute for each list item is `#id` where `id` is the ID value assigned to the corresponding heading

Creating Hypertext Links (continued 1)

Figure 12-37 Node structure of the linked list items

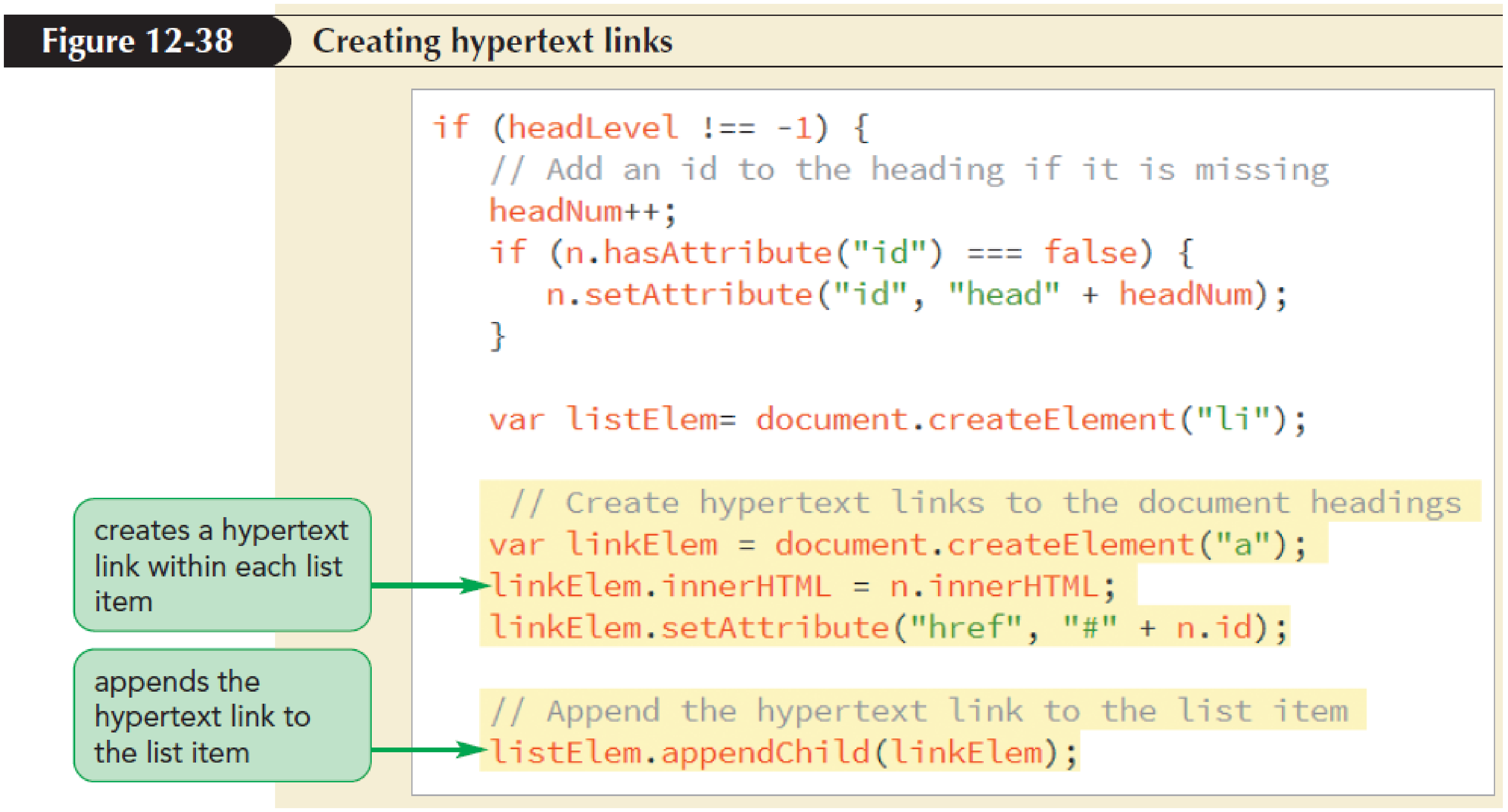


```
<ol>
   <li>
     <a href="#head">
       heading text
     </a>
   </li>
```

© 2016 Cengage Learning

Creating Hypertext Links (continued 2)

Figure 12-38 Creating hypertext links



```
if (headLevel !== -1) {
   // Add an id to the heading if it is missing
   headNum++;
   if (n.hasAttribute("id") === false) {
      n.setAttribute("id", "head" + headNum);
   }

   var listElem= document.createElement("li");

    // Create hypertext links to the document headings
   var linkElem = document.createElement("a");
   linkElem.innerHTML = n.innerHTML;
   linkElem.setAttribute("href", "#" + n.id);

   // Append the hypertext link to the list item
   listElem.appendChild(linkElem);
```

creates a hypertext link within each list item
appends the hypertext link to the list item

Working with Style Sheets

- Inline styles are created by modifying the `style` property of various page elements
- In inline styles, the `style` property is applied to specific elements in a document
- JavaScript can be used to create and modify style sheets that can be applied to en entire document

The styleSheets Collection

- Both external and embedded style sheets are part of the following object collection:

  ```
  document.styleSheets
  ```

- Arrangement of style sheets within a collection follows their order of declaration within a document head
- First style sheet declared in a document is referenced as `document.styleSheets[0];`

The styleSheets Collection (continued 1)

- Subsequent style sheets would be referenced as `document.styleSheets[1]`, `document.styleSheets[2]`, and so forth
- Total number of style sheets in a document is given by the `document.styleSheets.length` property
- Last style sheet in a collection can be referred using the following property:

```
document.styleSheets[document.styleS
heets.length-1]
```

The styleSheet Object

- Each style sheet in a styleSheets collection is a styleSheet object

Figure 12-41 styleSheet object properties

Property	Description
sheet.cssRules	The object collection of style rules within the style sheet
sheet.disabled	A Boolean value indicating whether the style sheet is disabled (`true`) or enabled (`false`)
sheet.href	The URL of the style sheet file or an empty text string for an embedded style sheet (read-only)
sheet.media	A text string containing the list of media types associated with the style sheet (read-only)
sheet.ownerNode	The `embed` or `link` element node that created the style sheet (read-only)
sheet.parentStyleSheet	The styleSheet object containing the style sheet inserted via the `@import` rule (read-only)
sheet.title	The title of the style sheet (read-only)
sheet.type	The MIME type of the style sheet (read-only)

The styleSheet Object (continued 1)

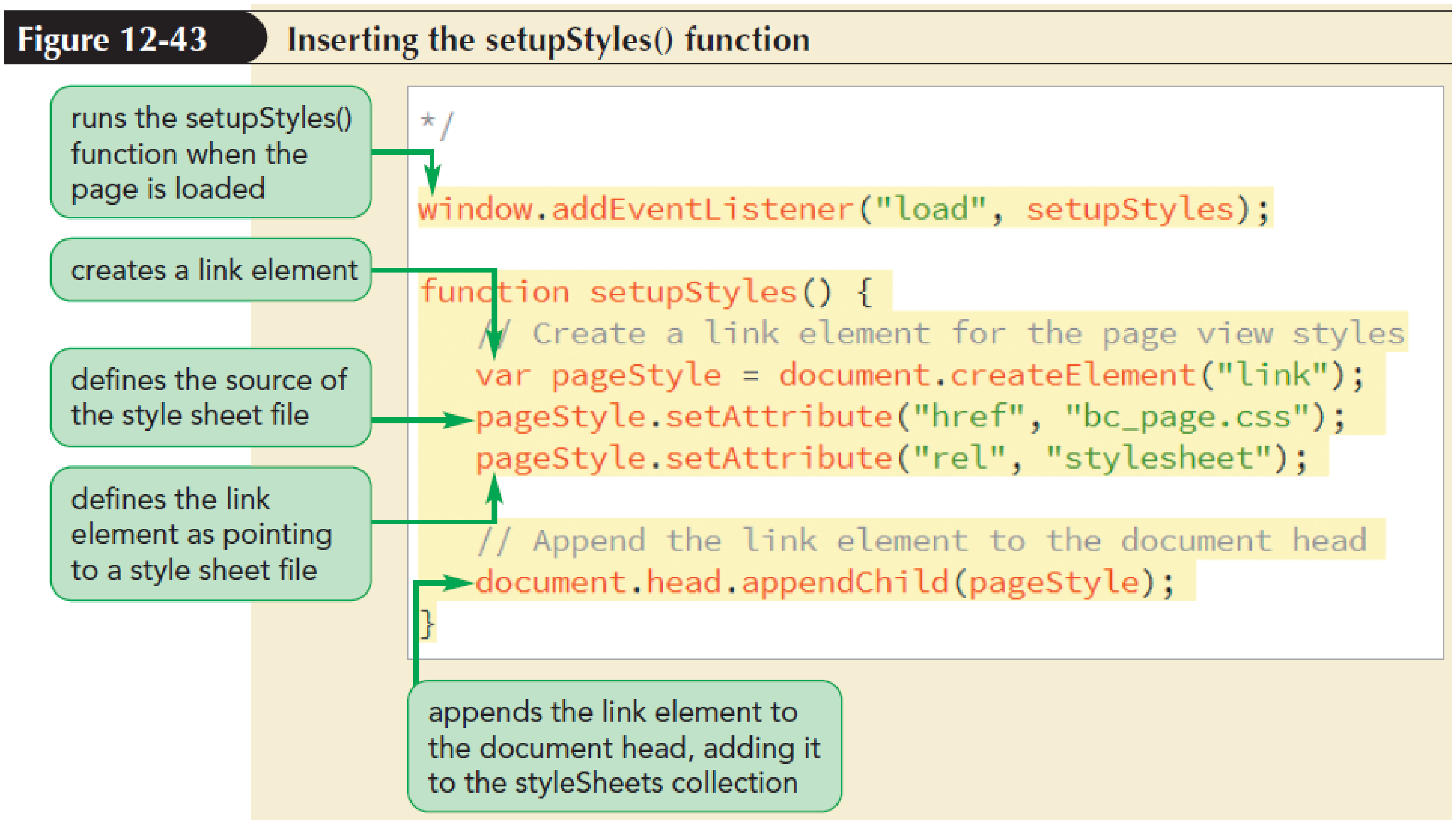
- To reference a specific style sheet, use the following code:

  ```
      document.styleSheets[index]
  ```

  where *`index`* is the index number of the style sheet, starting with 0 for the first style sheet
- **Style sheet switcher** is an application used to choose among different style sheets to display a web document

The styleSheet Object (continued 2)

Figure 12-43 Inserting the setupStyles() function

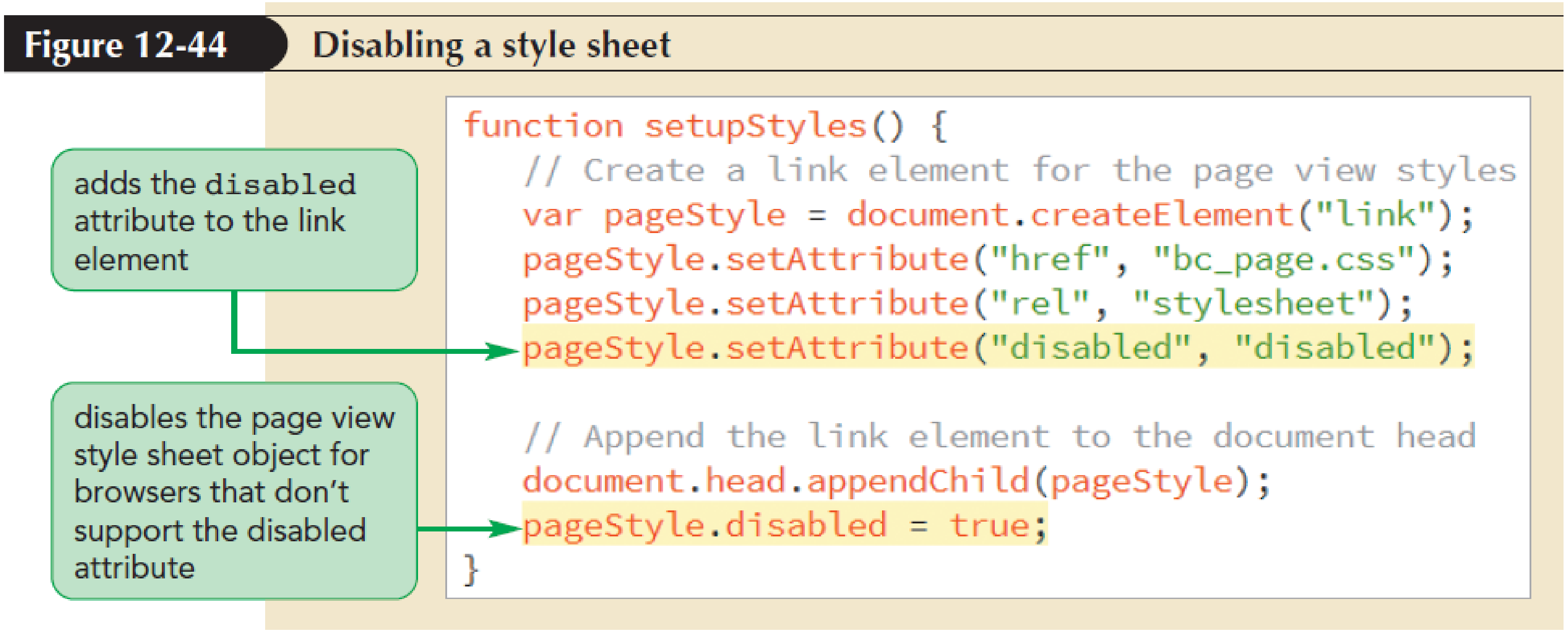


```
*/

window.addEventListener("load", setupStyles);

function setupStyles() {
   // Create a link element for the page view styles
   var pageStyle = document.createElement("link");
   pageStyle.setAttribute("href", "bc_page.css");
   pageStyle.setAttribute("rel", "stylesheet");

   // Append the link element to the document head
   document.head.appendChild(pageStyle);
}
```

runs the setupStyles() function when the page is loaded
creates a link element
defines the source of the style sheet file
defines the link element as pointing to a style sheet file
appends the link element to the document head, adding it to the styleSheets collection

The styleSheet Object (continued 3)

Figure 12-44 Disabling a style sheet


adds the disabled attribute to the link element
disables the page view style sheet object for browsers that don't support the disabled attribute

```
function setupStyles() {
   // Create a link element for the page view styles
   var pageStyle = document.createElement("link");
   pageStyle.setAttribute("href", "bc_page.css");
   pageStyle.setAttribute("rel", "stylesheet");
   pageStyle.setAttribute("disabled", "disabled");

   // Append the link element to the document head
   document.head.appendChild(pageStyle);
   pageStyle.disabled = true;
}
```

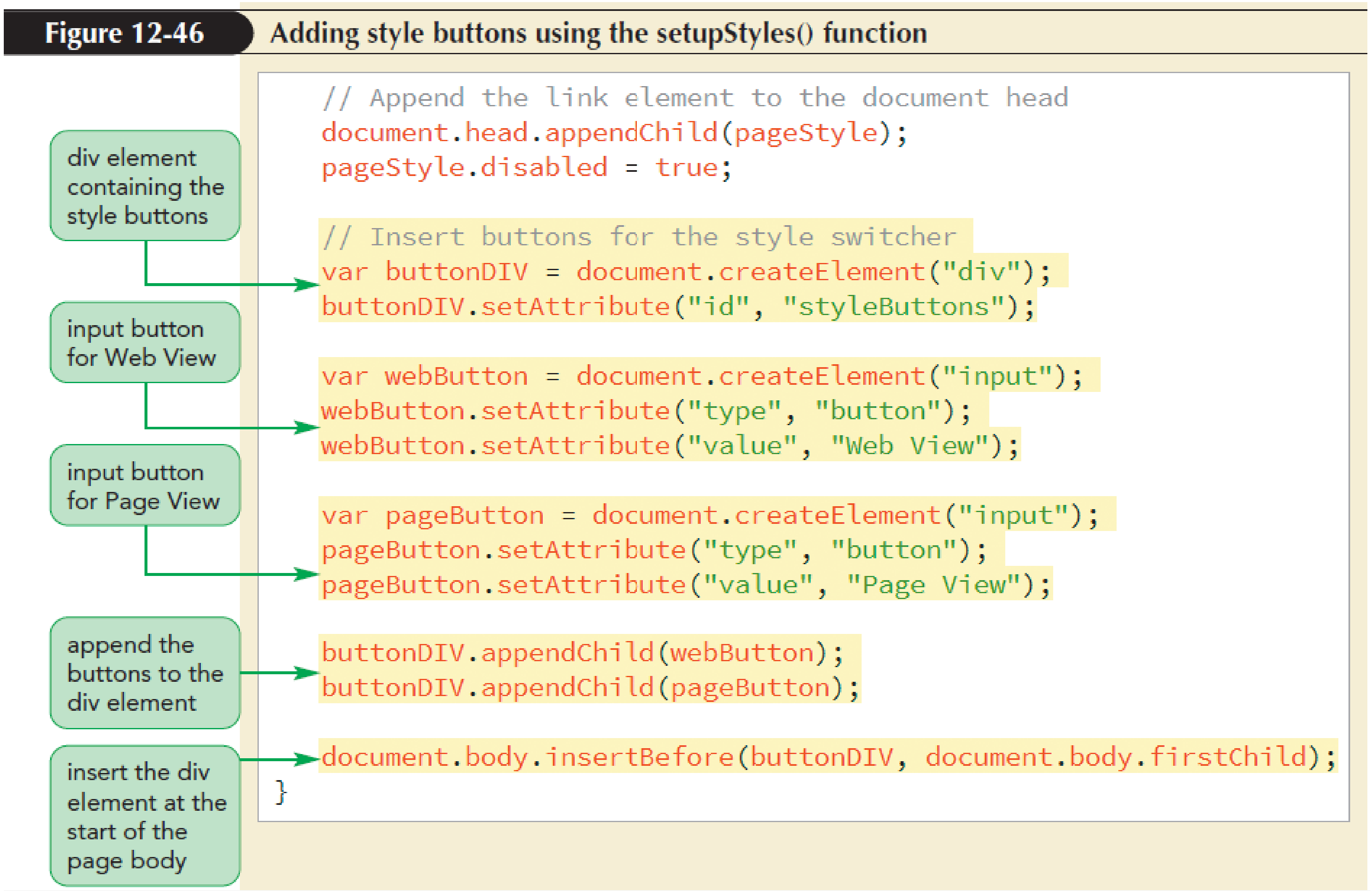
The styleSheet Object (continued 4)

- Form buttons are used to switch between the two views of a page
- HTML code for the form buttons is as follows:

```
<div id=”styleButtons”>
<input type=”button” value=”Web
View”  />
<input type=”button” value=”Page
View” />
</div>
```

The styleSheet Object (continued 5)

Figure 12-46 Adding style buttons using the setupStyles() function



```
    // Append the link element to the document head
    document.head.appendChild(pageStyle);
    pageStyle.disabled = true;

    // Insert buttons for the style switcher
    var buttonDIV = document.createElement("div");
    buttonDIV.setAttribute("id", "styleButtons");

    var webButton = document.createElement("input");
    webButton.setAttribute("type", "button");
    webButton.setAttribute("value", "Web View");

    var pageButton = document.createElement("input");
    pageButton.setAttribute("type", "button");
    pageButton.setAttribute("value", "Page View");

    buttonDIV.appendChild(webButton);
    buttonDIV.appendChild(pageButton);

    document.body.insertBefore(buttonDIV, document.body.firstChild);
}
```

div element containing the style buttons
input button for Web View
input button for Page View
append the buttons to the div element
insert the div element at the start of the page body

Working with Style Sheet Rules

- Style rules within the sheet are part of a following object collection:

  ```
  stylesheet.cssRules
  ```

  where stylesheet is a reference to a sheet within the `document.styleSheets` object collection

Working with Style Sheet Rules (continued 1)

- Example: If the first style sheet contains the following rules:

```
<style id=”hStyles”>
    h1 {color: red;}
    h2 {color: blue;}
</style>
```

then the following object reference

```
document.styleSheets[0].cssRules[1]
```

points to the second style rule

```
h2 {color: blue;}
```

Style Rule Objects

- Each individual rule within a cssRules collection is treated as an object

Figure 12-48 cssRule object properties

Property	Description
rule.cssText	The contents of *rule* as a text string (read-only)
rule.parentRule	The style rule containing *rule* as a parent (read-only)
rule.parentStyleSheet	The style sheet containing *rule* (read-only)
rule.selectorText	The text of the selector for *rule*
rule.style.*property*	The value of a specific style *property* within *rule*
rule.type	An integer representing the type of *rule* where 0=unknown, 1=style rule, 2=@charset, 3=@import, 4=@media, 5=@font-face, and 6=@page (read-only)

Adding and Removing Style Rules

- To add a new rule at the end of a style sheet, use `*sheet*.cssRules.length` as the index value
- Once a rule has been removed, all subsequent rules move up in the cssRules collection and their index values change subsequently

Adding and Removing Style Rules (continued 1)

- Use the following code to insert a new style rule into a style sheet:

  ```
  sheet.insertRule(rule, index)
  ```

  where *sheet* is the style sheet, *rule* is the text of the style rule, and *index* is the index value of the new rule

- Use the following code to remove a rule from a style sheet:

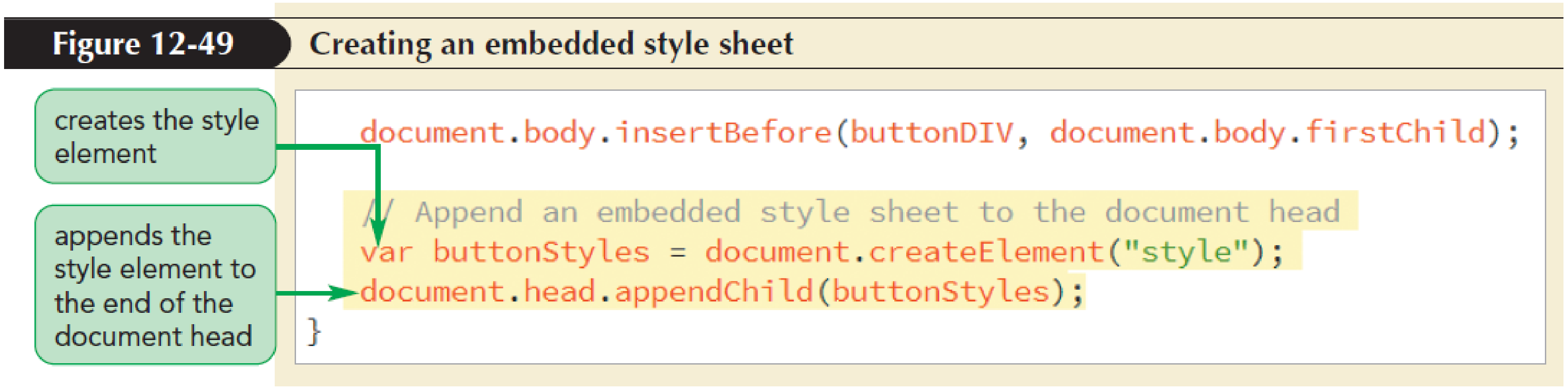
  ```
  sheet.deleteRule(index)
  ```

  where *index* is the index value of the rule

Adding and Removing Style Rules (continued 2)

- JavaScript can be used to append an embedded style sheet to the end of a document head

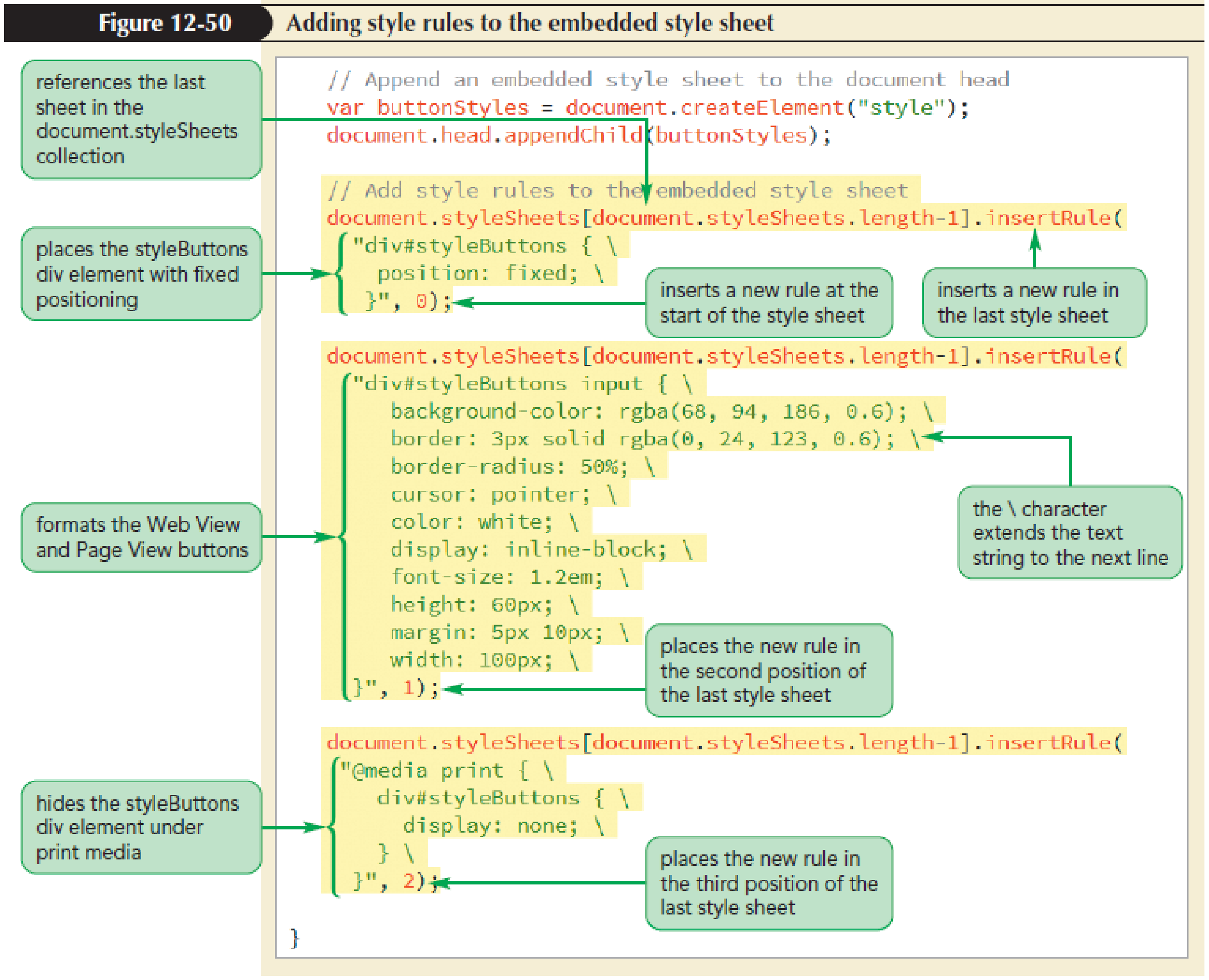
Figure 12-49 Creating an embedded style sheet


creates the style element
appends the style element to the end of the document head

```
   document.body.insertBefore(buttonDIV, document.body.firstChild);

   // Append an embedded style sheet to the document head
   var buttonStyles = document.createElement("style");
   document.head.appendChild(buttonStyles);
}
```

Adding and Removing Style Rules (continued 3)

Figure 12-50 Adding style rules to the embedded style sheet



```
// Append an embedded style sheet to the document head
var buttonStyles = document.createElement("style");
document.head.appendChild(buttonStyles);

// Add style rules to the embedded style sheet
document.styleSheets[document.styleSheets.length-1].insertRule(
 "div#styleButtons { \
   position: fixed; \
 }", 0);

document.styleSheets[document.styleSheets.length-1].insertRule(
 "div#styleButtons input { \
    background-color: rgba(68, 94, 186, 0.6); \
    border: 3px solid rgba(0, 24, 123, 0.6); \
    border-radius: 50%; \
    cursor: pointer; \
    color: white; \
    display: inline-block; \
    font-size: 1.2em; \
    height: 60px; \
    margin: 5px 10px; \
    width: 100px; \
 }", 1);

document.styleSheets[document.styleSheets.length-1].insertRule(
"@media print { \
   div#styleButtons { \
     display: none; \
   } \
 }", 2);

}
```

references the last sheet in the document.styleSheets collection
places the styleButtons div element with fixed positioning
inserts a new rule at the start of the style sheet
inserts a new rule in the last style sheet
formats the Web View and Page View buttons
the \ character extends the text string to the next line
places the new rule in the second position of the last style sheet
hides the styleButtons div element under print media
places the new rule in the third position of the last style sheet

Adding and Removing Style Rules (continued 4)

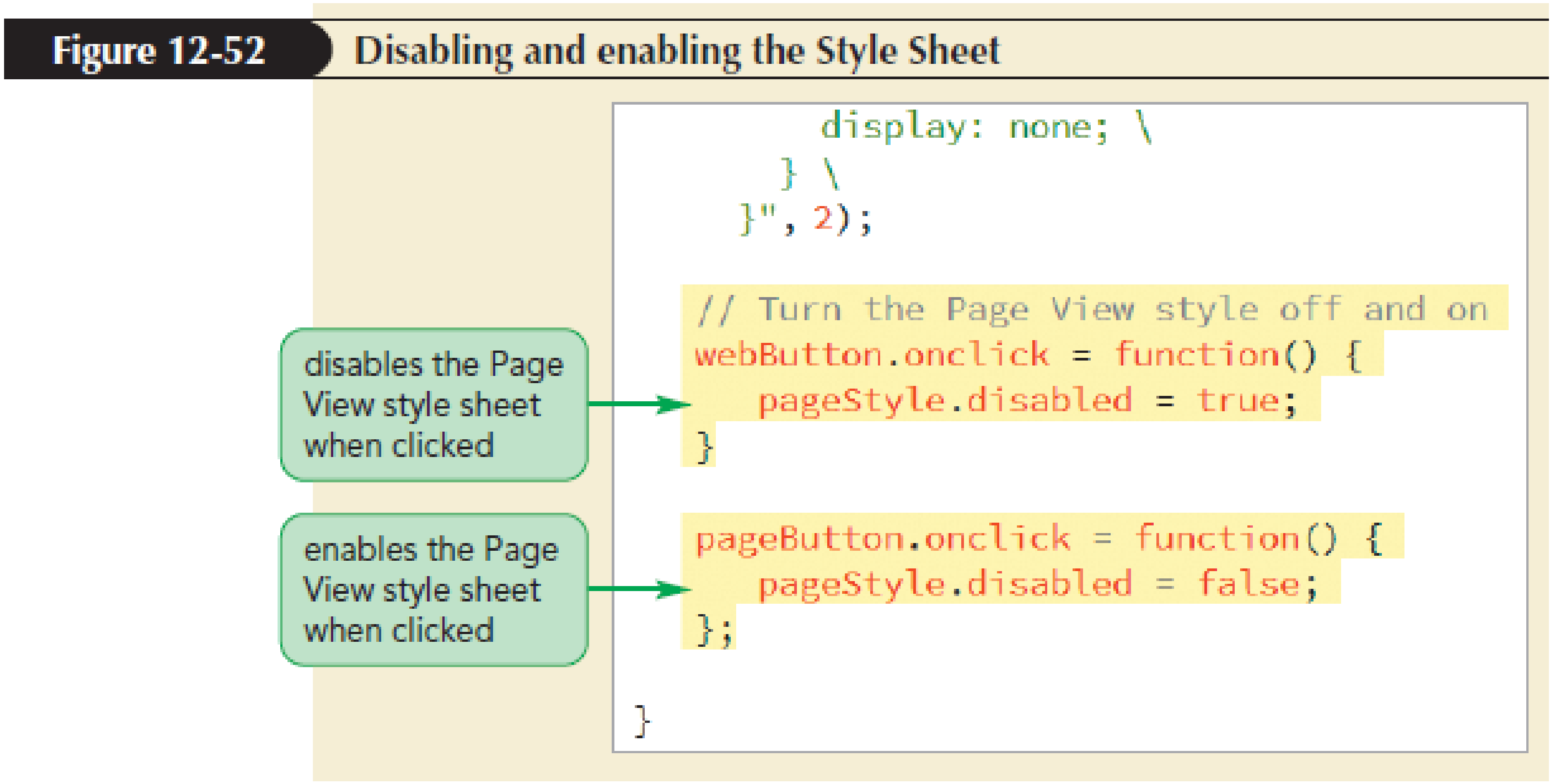


Figure 12-52 Disabling and enabling the Style Sheet

```
          display: none; \
        } \
      }", 2);

   // Turn the Page View style off and on
   webButton.onclick = function() {
      pageStyle.disabled = true;
   }

   pageButton.onclick = function() {
      pageStyle.disabled = false;
   };

}
```

disables the Page View style sheet when clicked
enables the Page View style sheet when clicked

Exploring Calculated Styles

- Final appearance of any page element is a **calculated style** based on all the styles found within the following:
  - External style sheets
  - Embedded sheets
  - Inline styles
  - Styles built into the browser

Exploring Calculated Styles (continued 1)

- Following `getComputedStyle()` method can be used to retrieve the calculated styles:

  ```
  document.getComputedStyle(object,
  pseudo)
  ```

  where *`object`* is a page object, and *`pseudo`* is a text string containing a pseudo-element that is applied to the page object
- If no pseudo-element is used, set the *`pseudo`* value to null

Exploring Calculated Styles (continued 2)

- To retrieve the calculated styles for the first paragraph of the current document, use:

```
var p1 =
document.getElementsByTagName(“p”)[0
];
var p1Styles =
document.getComputedStyle(p1, null);
```

Exploring Calculated Styles (continued 3)

- To retrieve the calculated value of a specific style, use:

  `styleDeclaration.getPropertyValue`
  `(`*`styleText`*`)`

  or

  `styleDeclaration.style`

  where *`styleDeclaration`* is a CSSStyleDeclaration object, *`styleText`* is a text string, and *`style`* is the JavaScript name of a style property

Exploring Calculated Styles (continued 4)

- Calculated styles are read-only values and cannot be changed using JavaScript
- Values are expressed in absolute units
- Calculated property that measures a size is expressed only in pixels