Instructor: Michael Kremer, Ph.D.

Class 6

Professional Program: Data Administration and Management

JAVASCRIPT AND JQUERY: AN INTRODUCTION (WEB PROGRAMMING, X452.1)

AGENDA

14. Window Object
15. Scripting Documents

Window Object

14.

14.1 PROPERTIES AND METHODS

- Window object is the global object for client-side JavaScript programs

Timers

- setTimeout() and setInterval() allow you to register a function to be invoked once or repeatedly after a specified amount of time has elapsed:
  - setTimeout() method of the Window object schedules a function to run after a specified number of milliseconds elapses
  - setInterval() is like setTimeout() except that the specified function is invoked repeatedly at intervals of the specified number of milliseconds
  - setTimeout() and setInterval() returns a value that can be passed to clearTimeout() to cancel the execution of the scheduled function

Example:

```
setInterval(updateClock, 60000); // Call updateClock() every 60 seconds
```

14.1 PROPERTIES AND METHODS

Error Handling

- onerror property (= event handler) of a Window object is invoked when an uncaught exception propagates all the way up
- Assign a function to this property → function is invoked whenever a JavaScript error occurs: function is the error handler
- For historical reasons, the onerror event handler of the Window object is invoked with three string arguments:
- Return value:
  - False → error was handled, no further action required. (In Firefox must return true)
  - True → unhandled error, browser display's message
- onerror handler is a holdover from the early days of JavaScript: It is rarely used in modern code, good for quick testing

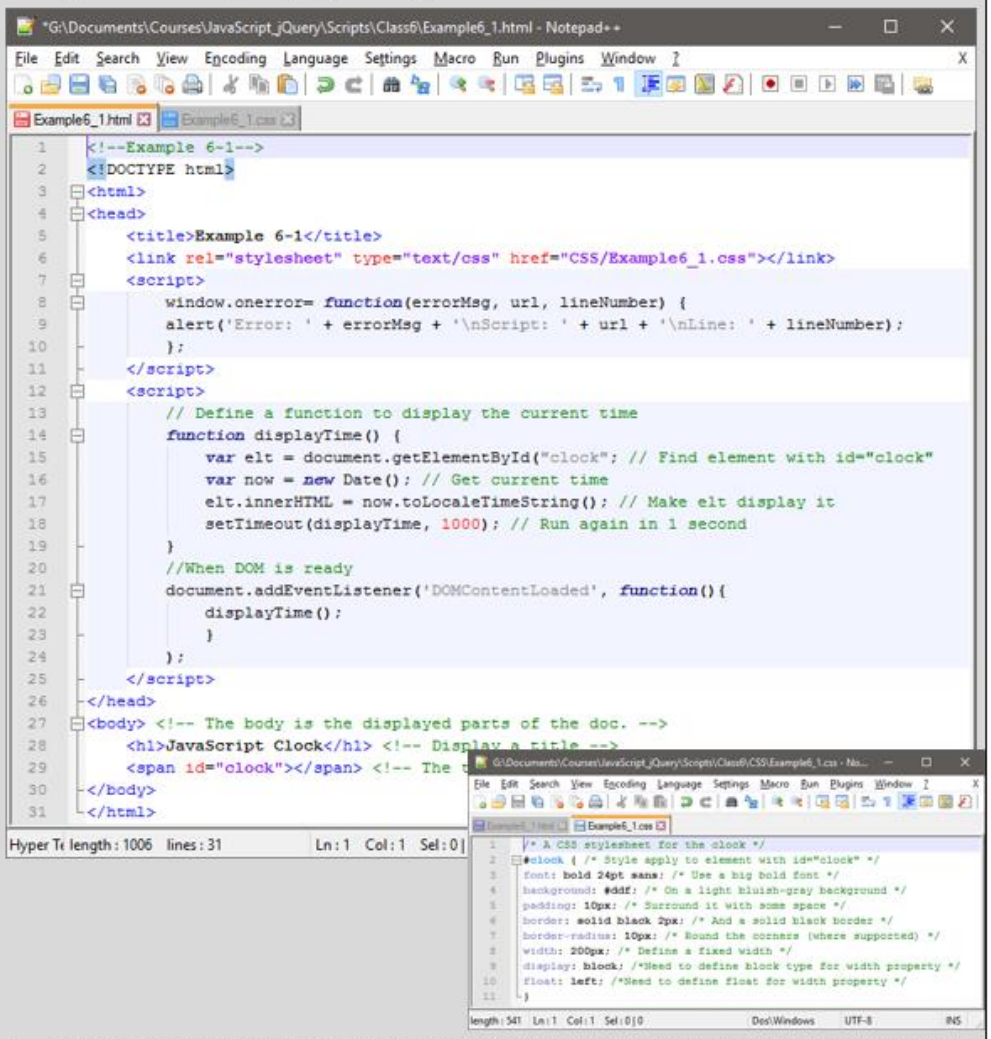
The three string arguments are:

- The first argument to window.onerror is a message describing the error.
- The second argument is a string that contains the URL of the JavaScript code that caused the error.
- The third argument is the line number within the document where the error occurred.

14.1 PROPERTIES AND METHODS

Example 6-1: JavaScript Timers and Error Handling

1. Type the following html page in notepad++ or any other text editor.

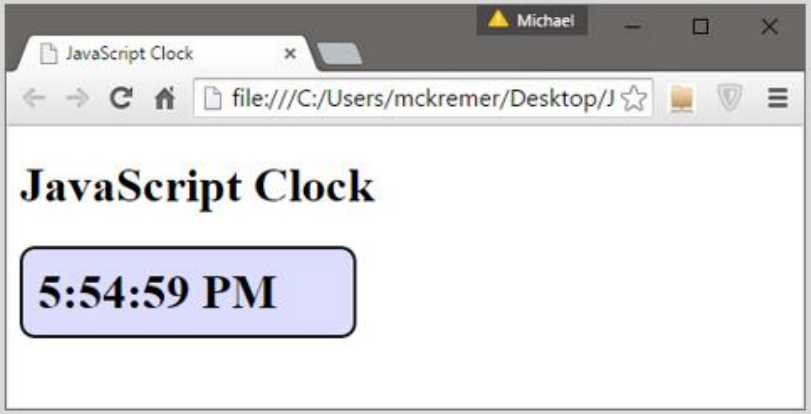


2. Save the page as Example6_1.html and Example6_1.css (in folder CSS).(Make sure not to save it as .txt file)

14.1 PROPERTIES AND METHODS

Example 6-1 (continued): JavaScript Timers and Error Handling

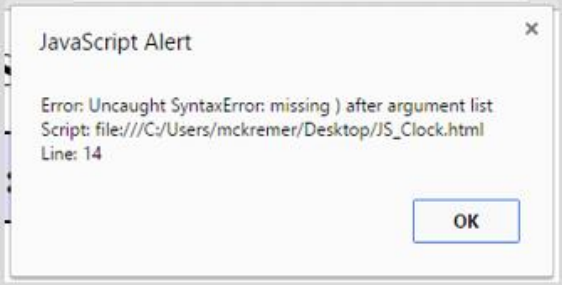
3. Double-click on the file to display it in a web browser.



4. Now edit the html file to generate a syntax error. In the JavaScript function, remove the closing parentheses of the getElementById method.

```
13  function displayTime() {
14      var elt = document.getElementById("clock"; // Find element with id="clock"
15      var now = new Date(); // Get current time
16      elt.innerHTML = now.toLocaleTimeString(); // Make elt display it
17      setTimeout(displayTime, 1000); // Run again in 1 second
18  }
```

5. Run the html file again.



6. Notice the alert dialog box displaying detailed information about the error.

14.2 BROWSER LOCATION AND NAVIGATION

- Window location property refers to location object (current URL)
- Also defines methods to load a new document
- Document object also has location property referring to the same object

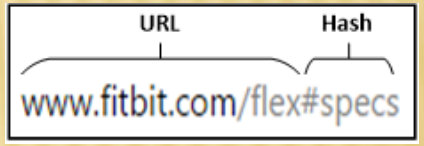
The location property of the Document object also refers to the Location object:

window.location === document.location // always true

- document.URL → URL when document was first loaded

URLs

- href property of location object refers to textual URL
- Other properties: protocol, host, hostname, port, pathname, search, and hash



- Known as “URL decomposition” properties, and they are also supported by Link objects (created by <a> and <area> elements in HTML documents)

14.2 BROWSER LOCATION AND NAVIGATION

Loading New Documents

- assign() method: load and display document with URL specified
- replace() method: similar, removes current URL from history and displays new document with URL specified
- For redirects, use replace:

Example:

```
// If the browser does not support the XMLHttpRequest object
// redirect to a static page that does not require it.
if (!XMLHttpRequest) location.replace("staticpage.html");
```

- reload() method: reloads the current document
- More traditional way of reloading, simply assign URL to location
- Navigate to a new page: Assign URL into location object

Example:

```
location = "http://ucb-access.org";
```

14.2 BROWSER LOCATION AND NAVIGATION

- Assignment of relative URLs to location are resolved against the current URL:

Example:

```
location = "page2.html"; // Load the next page
```

- Bare fragment identifier is a special kind of relative URL → browser does not load a new document but simply scroll to display a new section of the document.

Example:

```
location = "#top"; // Jump to the top of the document
```

- URL decomposition properties of the Location object are writable, and setting them changes the location URL and also causes the browser to load a new document

Example:

```
location.search = "?page=" + (pagenum+1); // load the next page
```

14.2 BROWSER LOCATION AND NAVIGATION

Browsing History

- Length property of the History object specifies the number of elements in the browsing history list, but for security reasons scripts are not allowed to access the stored URLs
- back() and forward() methods, go() takes an integer to navigate to skip pages in history object

Example:

```
history.go(-2); // Go back 2, like clicking the Back button twice
```

- For child windows (such as <iframe>elements), the browsing histories of the child windows are chronologically interleaved with the history of the main window
- History management before HTML 5 was complex and not easy. An application managing its own history (maybe because of Ajax functionality) had to employ complex JavaScript code
- HTML 5 offers History API!

14.3 BROWSER AND SCREEN INFORMATION

- Need to detect the current browser or the OS the browser is running in

Navigator Object

- Navigator object contains browser vendor and version number information
- Was used for “browser-sniffing”: → this approach is problematic because it requires constant tweaking as new browsers and new versions of existing browsers are introduced
- Rather perform feature testing!
- Browser sniffing is still needed to work around certain bugs in browsers

- appName
  The full name of the web browser. In IE, this is “Microsoft Internet Explorer”. In Firefox, this property is “Netscape”. For compatibility with existing browser sniffing code, other browsers often report the name “Netscape” as well.
- appVersion
  This property typically begins with a number and follows that with a detailed string that contains browser vendor and version information. The number at the start of this string is often 4.0 or 5.0 to indicate generic compatibility with fourth- and fifth generation browsers. There is no standard format for the appVersion string, so parsing it in a browser-independent way is not possible.
- userAgent
  The string that the browser sends in its USER-AGENTHTTP header. This property typically contains all the information in appVersion and may contain additional details as well. Like appVersion, there is no standard format. Since this property contains the most information, browser-sniffing code typically uses it.
- Platform
  A string that identifies the operating system (and possibly the hardware) on which the browser is running

14.3 BROWSER AND SCREEN INFORMATION

- For new browsers, navigator.appName and navigator.appVersion had to be tweaked in order for web sites to work in new browsers → loose meaning
- For "modern" browser sniffing, use navigator.userAgent.

Example 6-2: Navigator Object

1. Open Firefox Scratchpad (see Example 1-1).
2. Type the following code:

*Scratchpad

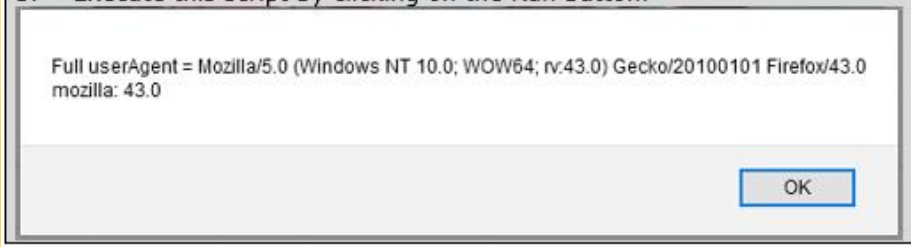
File Edit View Execute Help

Open File... Save Save As... Run Inspect Display Pretty Print

```
1  // Define browser.name and browser.version for client sniffing, using code
2  // derived from jQuery 1.4.1. Both the name and number are strings, and both
3  // may differ from the public browser name and version. Detected names are:
4  //
5  // "webkit": Safari or Chrome; version is WebKit build number
6  // "opera": the Opera browser; version is the public version number
7  // "mozilla": Firefox or other gecko-based browsers; version is Gecko version
8  // "msie": IE; version is public version number
9  //
10 // Firefox 3.6, for example, returns: { name: "mozilla", version: "1.9.2" }.
11 var browser = (function() {
12   var s = navigator.userAgent.toLowerCase();
13   var match = /(webkit)[ \/]([\w.]+)/.exec(s) || //Chrome
14         /(opera)(?:.*version)?[ \/]([\w.]+)/.exec(s) || //Opera
15         /(msie) ([\w.]+)/.exec(s) ||  //Internet Explorer
16         !/compatible/.test(s) && /(mozilla)(?:.*? rv:([\w.]+))?/.exec(s) || //Mozilla
17         [];
18   return { name: match[1] || "", version: match[2] || "0" };
19 }());
20 alert("Full userAgent = " + navigator.userAgent + "\n" +browser.name + ": " + browser.version);
21
```

Line 21, Col 1

3. Execute this script by clicking on the Run button.



14.3 BROWSER AND SCREEN INFORMATION

- Standardized and widely implemented nonstandard Navigator properties include:

- onLine
  The navigator.onLine property (if it exists) specifies whether the browser is currently connected to the network. Applications may want to save state locally while they are offline.
- Geolocation
  A Geolocation object that defines an API for determining the user's geographical location.
- javaEnabled()
  A nonstandard method that should return true if the browser can run Java applets.
- cookiesEnabled()
  A nonstandard method that should return true if the browser can store persistent cookies. May not return the correct value if cookies are configured on a site-by-site basis

Screen Object

- Screen property of a Window object refers to a Screen object → information about the size of the user's display and the number of colors available on it
- availWidth and availHeight properties specify the display size that is actually available (excluding i.e. desktop taskbar)
- colorDepth property specifies the bits-per-pixel value of the screen (values are 16,24,32)
- window.screen property and the Screen object to which it refers are both nonstandard but widely implemented:
  - Use them to determine whether your web app is running in a small form factor device such as a netbook computer or mobile phone

14.4 DIALOG BOXES

- Window object provides 3 methods for displaying simple dialog boxes:
  - Although easy to use, use them sparingly → most users find those dialog boxes disruptive to their browsing experience.
  - Commonly used for debugging

- alert() displays a message to the user and waits for the user to dismiss the dialog.
- confirm() displays a message, waits for the user to click an OK or Cancel button and returns a Boolean value.
- prompt() displays a message, waits for the user to enter a string, and returns that string.

Example:

```
do {
        var name = prompt("What is your name?"); // Get a string
        var correct = confirm("You entered '" + name + "'.\n" + // Get a Boolean
        "Click Okay to proceed or Cancel to re-enter.");
} while(!correct)
alert("Hello, " + name); // Display a plain message
```

- confirm() and prompt() methods block → do not return until the user dismisses the dialog boxes they display
- In most browsers alert() method also blocks and waits for the user to dismiss the dialog box, but this is not required

Note: The messages displayed by alert(), confirm(), and prompt()are plain text, not HTML-formatted text. You can format these dialog boxes only with spaces, newlines, and punctuation characters.

Warning: The showModalDialog() method is deprecated in Firefox and Chrome removed support from it. Use alternatives from JS frameworks of jQuery.

14.5 MULTIPLE FRAMES AND WINDOWS

- Single web browser window on your desktop may contain several tabs or if tabs are disabled, multiple browser windows that are open
- Each tab/window is an independent browsing context → own Window object
- Scripts running in one tab/window usually have no way of even knowing that the other tabs/windows exist
- Exception: Script in one window or tab can open new windows or tabs, and when a script does this, the windows can interact with one another and with one another's documents (subject to the constraints of the same-origin policy)
- <iframe> creates a nested browsing context represented by a Window object of its own
- <frameset> and <frame> (deprecated) also create nested contexts, and also own a separate window object
- Nested browsing contexts are not isolated from one another the way independent tabs usually are

14.5 MULTIPLE FRAMES AND WINDOWS

Opening Windows

- open() method of the Window object
- Window.open() loads a specified URL into a new or existing window and returns the Window object that represents that window →four arguments:
  - URL of the document to display, if left blank → URL about:blank
  - String that specifies a window name: uses existing window or creates a new window
  - Comma-separated list of size and features attributes for the new window to be opened (if omitted → defaults are used)
  - Boolean value that indicates whether the URL specified as the first argument should replace the current entry in the window's browsing history (true) or create a new entry in the window's browsing history (false)

14.5 MULTIPLE FRAMES AND WINDOWS

- Third argument is nonstandard, HTML5 specification insists that browsers be able to ignore it

Example:

```
var w = window.open("smallwin.html", "smallwin",
"width=400,height=350,status=yes,resizable=yes");
```

- Browsers include restrictions on the features you can specify (security), not allowed to specify a window that is too small or is positioned offscreen
- Return value of the open() method is the Window object that represents the named or newly created window

Example:

```
var w = window.open(); // Open a new, blank window.
w.alert("About to visit http://example.com"); // Call its alert() method
w.location = "http://example.com"; // Set its location property
```

- In windows created with the window.open() method, the opener property refers back to the Window object of the script that opened it

Example:

```
w.opener !== null; // True for any window w created by open()
w.open().opener === w; // True for any window w
```

- Pop-up blocking: window.open generally works only if initiated by the user (button click)

14.5 MULTIPLE FRAMES AND WINDOWS

Closing Windows

- If you create a Window object w, you can close it with:

Example:

```
w.close();
```

- JavaScript code running within that window itself can close it with:

Example:

```
window.close();
```

- Explicit use of the window identifier to distinguish the close() method of the Window object from the close() method of the Document object
- In general, close only those windows that your own JavaScript code has created →Other windows cannot be closed or a prompt is displayed to the user in order to cancel the request
- close() method of a Window object that represents a frame rather than a top-level window or tab does nothing: it is not possible to close a frame/tab (delete iframe instead)

14.5 MULTIPLE FRAMES AND WINDOWS

Example 6-3: Opening and Closing Windows

1. Type the following html page in notepad++ or any other text editor.

G:\Documents\Courses\JavaScript_jQuery\Scripts\Class6\Example6_3.html - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

Example6_3.html | Example6_3a.html

```
1  <!--Example 6-3-->
2  <!DOCTYPE html>
3  <html>
4  <head>
5          <title>Example 6-3</title>
6          <script> //JavaScript to open new window
7          document.addEventListener("DOMContentLoaded", function() {
8              var button = document.getElementById("btnWinOpen");
9              button.onclick = function(){
10                 if(confirm("Open new Window?"))
11                 {
12                     var win = window.open("Example6_3a.html","Course",
13                     "height=600,width=1200,menubar=no,resizable=no,scrollbars=yes");
14                     win.alert("Navigating to course web site")
15                 }
16             }
17         });
18         </script>
19 </head>
20 <body> <!-- The body is the displayed parts of the doc. -->
21     <h1>Opening and Closing Windows</h1> <!-- Display a title -->
22 <p>Below is a button that opens the course web site (ucb-access.org)</p>
23 <input type="button" id="btnWinOpen" value="Course Web Site" />
24 </body>
25 </html>
```

Hyp length : 824 lines : 25 | Ln : 1 Col : 1 Sel : 0 | 0 | Dos\Windows | UTF-8 | INS

2. Save the page as Example6_3.html.(Make sure not to save it as .txt file)

14.5 MULTIPLE FRAMES AND WINDOWS

Example 6-3(continued): Opening and Closing Windows

3. Type another html page and save it as Example6_7a.html.

G:\Documents\Courses\JavaScript_jQuery\Scripts\Class6\Example6_3a.html - Notepad++

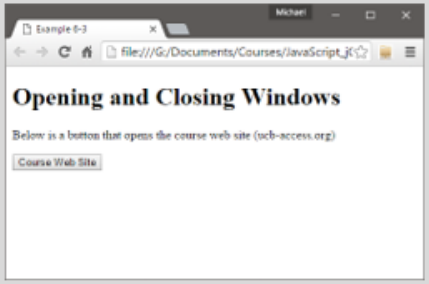
File Edit Search View Encoding Language Settings Macro Run Plugins Window ? X

Example6_3.html | Example6_3a.html

```
1  <!--Example 6-3a-->
2  <!DOCTYPE html>
3  <html>
4  <head>
5          <title>Example 6-3a</title>
6          <script> //JavaScript to add close event to button btnClose
7          document.addEventListener("DOMContentLoaded", function() {
8              var button = document.getElementById("btnClose");
9              button.onclick = function(){ //Add event to button btnClose
10                 window.close(); //Close current window
11             }
12         });
13         </script>
14 </head>
15 <body> <!-- The body is the displayed parts of the doc. -->
16     <h1>Opening and Closing Windows</h1> <!-- Display a title -->
17 <p>Below is the course web site in an iframe</p>
18 <iframe src="http://ucb-access.org" width="100%" height="400px" ></iframe>
19 <input type="button" id="btnClose" value="Close Window" />
20 </body>
21 </html>
```

length : 737 lines : 21 | Ln : 1 Col : 1 Sel : 0 | 0 | Dos\Windows | UTF-8 | INS

4. Open page Example6_3.html in a browser.
5. Click on the Course Web Site button.
6. Click on OK to confirm. Notice now that a second browser window opened. Also notice the alert dialog box!
7. Click on the Close Window button.

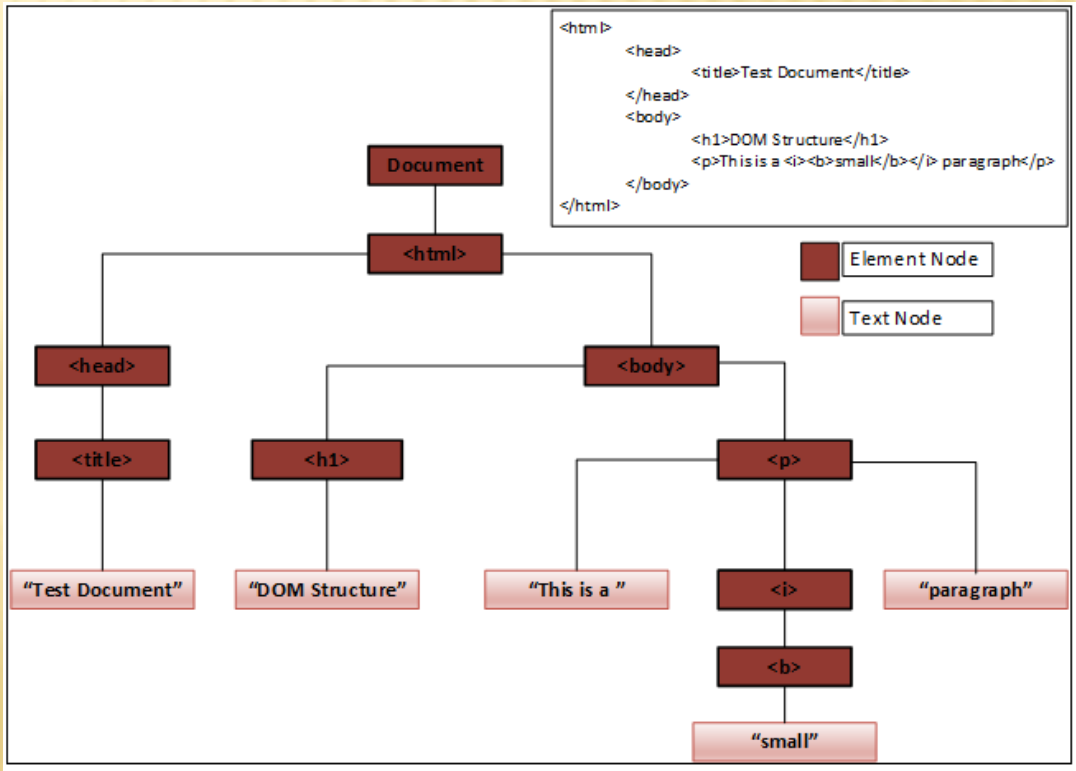


Scripting Documents

15.

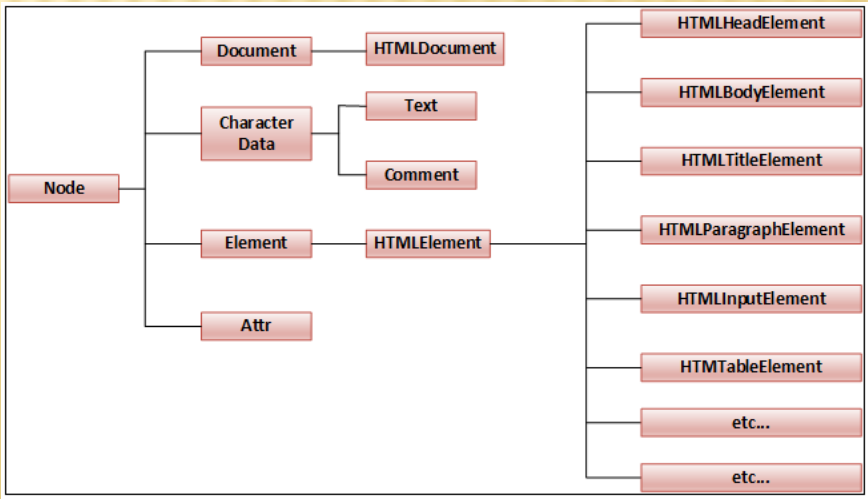
15.1 OVERVIEW OF THE DOM

- Client-side JavaScript exists to turn static HTML documents into interactive web applications
- Every Window object has a document property that refers to a Document object → content of the window
- Document Object Model, or DOM, is fundamental API for representing and manipulating the content of HTML and XML
- API is not particularly complicated, but there are a number of architectural details you need to understand



15.1 OVERVIEW OF THE DOM

- Tree representation of an HTML document contains nodes:
  - HTML tags or elements
  - Strings of text (also html comments).
- Terminology:
  - Node directly above a node is the parent of that node
  - Nodes one level directly below another node are the children of that node
  - Nodes at the same level, and with the same parent, are siblings
  - Nodes below deeper than children are descendants
  - Nodes above higher than parents are ancestors
- 3 different nodes, document , HTML element (head, body, title, etc.), and text
- Type hierarchy:



15.2 SELECTING DOCUMENT ELEMENTS

- In order to manipulate elements of the document, obtain or select the Element objects that refer to document elements
- The DOM defines a number of ways to select elements:

- with a specified id attribute
- with a specified name attribute
- with the specified tag name
- with the specified CSS class or classes
- matching the specified CSS selector

Selecting Elements by ID

- Any HTML element can have an id attribute. The value of this attribute must be unique within the document

Example:

```
var section1 = document.getElementById("section1");
```

Selecting Elements by Name

- HTML name attribute was originally intended to assign names to form elements
- Value of a name attribute does not have to be unique!
- Name attribute is only valid on a handful of HTML elements, including forms, form elements, <iframe>, and <img>elements

Example:

```
var radiobuttons = document.getElementsByName("favorite_color");
```

15.2 SELECTING DOCUMENT ELEMENTS

- getElementsByName() is defined by the HTMLDocument class, not the Document class
- Returns a NodeList object that behaves like a read-only array of Element objects

Example:

Read-only array like object containing the Element objects for all elements in a document.

```
var spans = document.getElementsByTagName("span");
```

Selecting Elements by Type

- Select all HTML or XML elements of a specified type (or tag name) using getElementsByTagName() method of the Document object
- Returns a NodeList object (in document order)
- HTML tags are case-insensitive!
- Wildcard argument "*" → all elements
- Element class defines same method, but finds only descendants of the element that it is invoked on

Example:

```
var firstparagraph = document.getElementsByTagName("p")[0];
```

Example:

Find all elements inside the first <p>element of a document:

```
var firstparagraph = document.getElementsByTagName("p")[0];
var firstParagraphSpans = firstparagraph.getElementsByTagName("span");
```

15.2 SELECTING DOCUMENT ELEMENTS

Selecting Elements by CSS Class

- Class attribute of an HTML element is a space-separated list of zero or more identifiers
- Class is a reserved word in JavaScript, so client-side JavaScript uses the className property to hold the value of the HTML class attribute!
- getElementsByClassName() can be invoked on both HTML documents and HTML elements, returns NodeList object

Example:

```
// Find all elements that have "warning" in their class attribute
var warnings = document.getElementsByClassName("warning");
// Find all descendants of the element named "log" that have the class
// "error" and the class "fatal"
var log = document.getElementById("log");
var fatal = log.getElementsByClassName("fatal error");
```

- getElementsByClassName() follows the matching algorithm used by stylesheets:
  - If the document is rendered in quirks (old) mode, the method performs a case-insensitive string comparison
  - In normal(strict) mode, the comparison is case sensitive

15.2 SELECTING DOCUMENT ELEMENTS

Selecting Elements by CSS Selectors

- CSS stylesheets have a very powerful syntax, known as selectors, for describing elements or sets of elements within a document
- Along with the standardization of CSS3 selectors, another W3C standard, known as "Selectors API" defines JavaScript methods for obtaining the elements that match a given selector

Example:

```
#nav // An element with id="nav"
div // Any <div> element
.warning // Any element with "warning" in its class attribute
```

More generally, elements can be selected based on attribute values.

Example:

```
p[lang="fr"] // A paragraph written in French: <p lang="fr">
*[name="x"] // Any element with a name="x" attribute
```

These basic selectors can be combined.

Example:

```
span.fatal.error // Any <span> with "warning" and "fatal" in its class
span[lang="fr"].warning // Any warning in French
```

Selectors can also specify document structure.

Example:

```
#log span // Any <span> descendant of the element with id="log"
#log>span // Any <span> child of the element with id="log"
body>h1:first-child // The first <h1> child of the <body>
```

Selectors can be combined to select multiple elements or multiple sets of elements.

Example:

```
div, #log // All <div> elements plus the element with id="log"
```

15.2 SELECTING DOCUMENT ELEMENTS

- Key to this API is the Document method querySelectorAll():
  - Single string argument containing a CSS selector and returns a NodeList that represents all elements in the document that match the selector
  - NodeList returned by querySelectorAll() is not live! Does not update when document structure has changed
  - If no elements match, querySelectorAll() returns an empty NodeList
  - If the selector string is invalid, querySelectorAll() throws an exception
- Other method querySelector():
  - Returns only the first (in document order) matching element or null if there is no matching element

Example:

```
//Sets background of p elements having a div as parent
var x = document.querySelectorAll("div > p");
for (var i = 0; i < x.length; i++) {
   x[i].style.backgroundColor = "red";
}
```

- The two methods are also defined on Elements
- jQuery library uses this kind of CSS selector-based query as its central programming paradigm

15.3 TRAVERSING THE DOM

- Two ways of traversing the DOM:
  - Document can be conceptualized as a tree of Node objects (includes all nodes, such as text and even comment nodes)
  - Documents can be traversed as trees of Element objects (includes only element nodes)

Documents as Tree of Nodes

- parentNode
  The Node that is the parent of this one, or nullfor nodes like the Document object that have no parent.
- childNodes
  A read-only array-like object (a NodeList) that is a live representation of a Node's child nodes.
- firstChild, lastChild
  The first and last child nodes of a node, or null if the node has no children.
- nextSibling, previousSibling
  The next and previous sibling node of a node. Two nodes with the same parent are siblings. Their order reflects the order in which they appear in the document. These properties connect nodes in a doubly linked list.
- nodeType
  The kind of node this is. Document nodes have the value 9. Element nodes have the value 1. Text nodes have the value 3. Comments nodes are 8 and DocumentFragment nodes are 11.
- nodeValue
  The textual content of a Text or Comment node.
- nodeName
  The tag name of an Element, converted to uppercase.

Example:

Using these Node properties, the second child node of the first child of the Document can be referred to with expressions like these:
document.childNodes[0].childNodes[1]
document.firstChild.firstChild.nextSibling

15.3 TRAVERSING THE DOM

Example:

Suppose the document in question is the following:
<html><head><title>Test</title></head><body>Hello World!</body></html>
Then the second child of the first child is the <body>element.
It has a nodeType of 1 and a nodeName of "BODY".

Note: This API is extremely sensitive to variations in the document text. If the document is modified by inserting a single newline between the <html>and the <head>tag, for example, the Text node that represents that newline becomes the first child of the first child, and the second child is the <head> element instead of the <body>body.

Documents as Tree of Elements

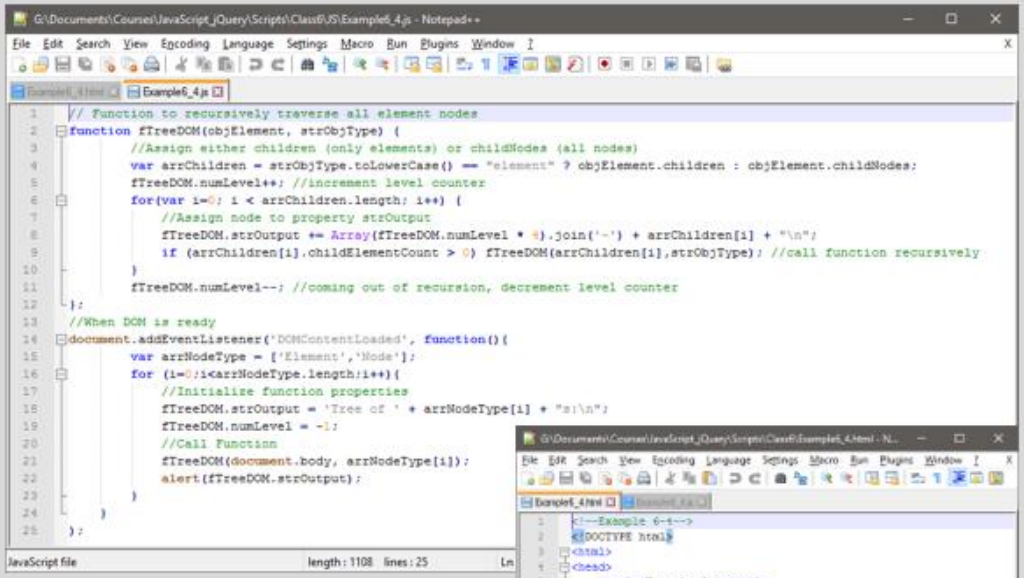
- Ignores Text and Comment nodes that are also part of the document
- First Part: Children property of Element objects returns NodeList, but contains only Element nodes
- Text and Comment nodes cannot have children, which means that the Node.parentNode property described above never returns a Text or Comment node → always returns an element
- Second part of an element-based document traversal API is Element properties that are analogs to the child and sibling properties of the Node object:

- firstElementChild, lastElementChild
  Like firstChild and lastChild, but for Element children only.
- nextElementSibling, previousElementSibling
  Like nextSibling and previousSibling, but for Element siblings only.
- childElementCount
  The number of element children. Returns the same value as children.length

15.3 TRAVERSING THE DOM

Example 6-4: Documents as tree of nodes/tree of elements

1. Type the following html page in notepad++ or any other text editor.



2. Save the page as Example6_4.html, .js. (Make sure not to save it as .txt file)
3. Double-click on the html file and view it in a browser.

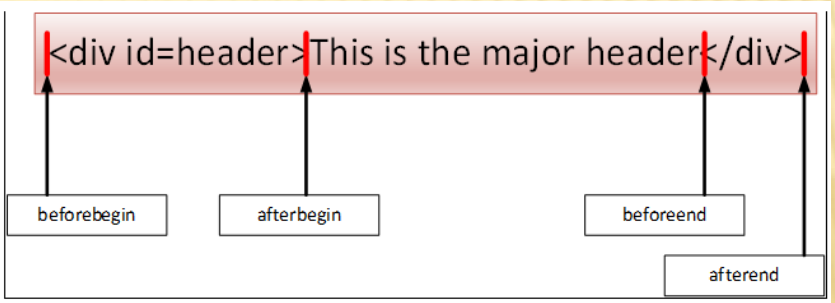
15.4 ELEMENT CONTENT

- What is content:
  - The content is the HTML string "This is a simple document."
  - The content is the plain-text string "This is a simple document."
  - The content is a Text node, an Element node that has a Text node child, and another Text node.

Element Content as HTML

- Reading the innerHTML property of an Element returns the content of that element as a string of markup
- HTML5 says that innerHTML should work on Document nodes as well as Element nodes, but this is not universally supported yet
- outerHTML returns the string of HTML or XML markup that includes the opening and closing tags of the element on which you queried it

15.4 ELEMENT CONTENT



- Standardized by HTML5: insertAdjacentHTML() method:

Syntax:

element.**insertAdjacedHTML**(*position*, *text*)

- Insert string of arbitrary HTML markup "adjacent" to the specified element:
  - First argument: "beforebegin", "afterbegin", "beforeend" or "afterend".
  - Second argument: string of markup

Element Content as Plain Text

- To query the content of an element as plain text, or to insert plaintext into a document → textContent property

Example:

```
var pElement = document.getElementsByTagName("p")[0]; // First <p> in the document
var text = pElement.textContent; // Text is "This is a simple document."
pElement.textContent = "Hello World!"; // Alter paragraph content
```

15.4 ELEMENT CONTENT

- textContent (all except IE) and innerText (IE) properties are similar enough that you can usually use them interchangeably.

Example:

```
/**
* With one argument, return the textContent or innerText of the element.
* With two arguments, set the textContent or innerText of element to value.
*/
function textContent(element, value) {
        var content = element.textContent; // Check if textContent is defined
        if (value === undefined) { // No value passed, so return current text
        if (content !== undefined) return content;
        else return element.innerText;
        }
else { // A value was passed, so set text
        if (content !== undefined) element.textContent = value;
        else element.innerText = value;
        }
}
```

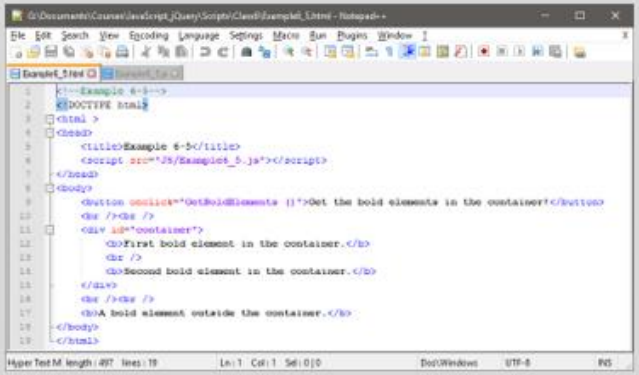
Element Content as Text Nodes

- Another way to work with the content of an element is as a list of child nodes, each of which may have its own set of children
- Usually the Text nodes that are of interest
- Next example shows a textContent() function that recursively traverses the children of an element and concatenates the text of all the Text node descendants
- In order to understand the code, recall that the nodeValue property (defined by the Node type) holds the content of a Text node

15.4 ELEMENT CONTENT

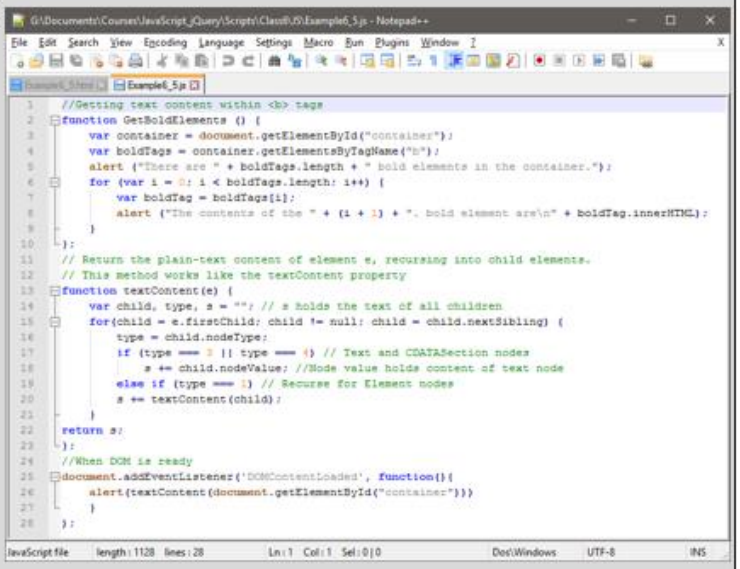
Example 6-5: Getting Text Content from a Node

1. Type the following html page in notepad++ or any other text editor.



```
<!--Example 6-5-->
<!DOCTYPE html>
<html >
<head>
    <title>Example 6-5</title>
    <script src="JS/Example6_5.js"></script>
</head>
<body>
    <button onclick="GetBoldElements ()">Get the bold elements in the container!</button>
    <br /><br />
    <div id="container">
        <b>First bold element in the container.</b>
        <br />
        <b>Second bold element in the container.</b>
    </div>
    <br /><br />
    <b>A bold element outside the container.</b>
</body>
</html>
```

2. Save the page as Example6_5.html.(Make sure not to save it as .txt file)
3. Create the following JavaScript file and save it as Example6_5.js (JS Folder).

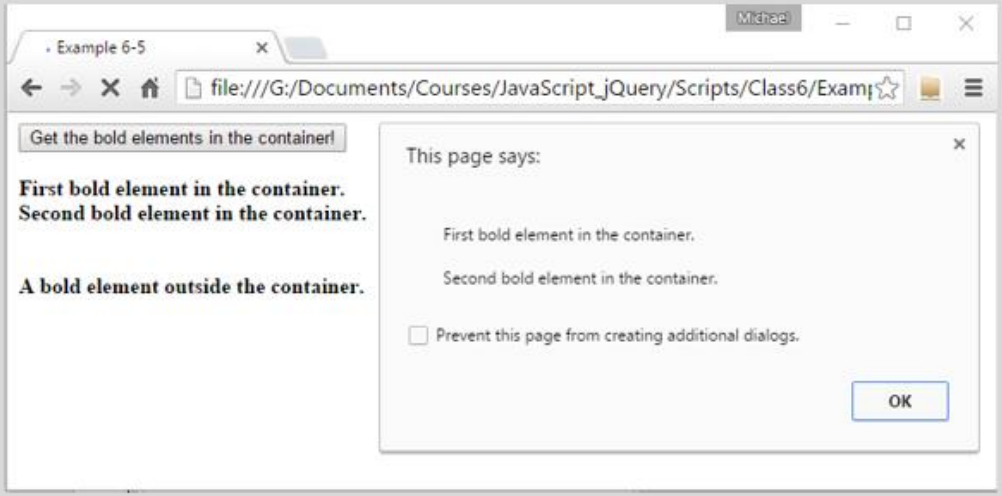


```
//Getting text content within <b> tags
function GetBoldElements () {
    var container = document.getElementById("container");
    var boldTags = container.getElementsByTagName("b");
    alert ("There are " + boldTags.length + " bold elements in the container.");
    for (var i = 0; i < boldTags.length; i++) {
        var boldTag = boldTags[i];
        alert ("The contents of the " + (i + 1) + ". bold element are\n" + boldTag.innerHTML);
    }
};
// Return the plain-text content of element e, recursing into child elements.
// This method works like the textContent property
function textContent(e) {
    var child, type, s = ""; // s holds the text of all children
    for(child = e.firstChild; child != null; child = child.nextSibling) {
        type = child.nodeType;
        if (type === 3 || type === 4) // Text and CDATASection nodes
            s += child.nodeValue; //Node value holds content of text node
        else if (type === 1) // Recurse for Element nodes
        s += textContent(child);
    }
return s;
};
//When DOM is ready
document.addEventListener('DOMContentLoaded', function(){
    alert(textContent(document.getElementById("container")))
    }
);
```

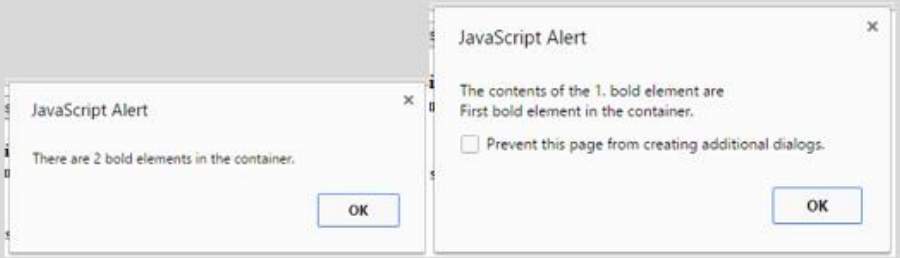
15.4 ELEMENT CONTENT

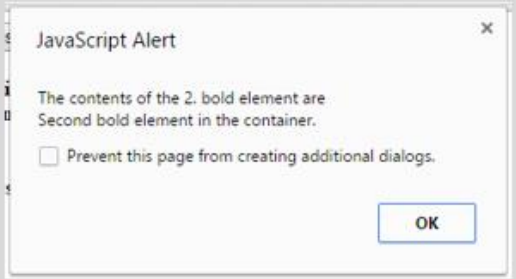
Example 6-5 (continued): Getting Text Content from a Node

4. Double-click on the html file and view it in a browser.



5. Now click on the button.





15.5 MANIPULATING DOCUMENT NODES

- Besides changing/creating document content and traversing the DOM → alter a document at the level of individual nodes:
  - Document type defines methods for creating Element and Text objects
  - Node type defines methods for inserting, deleting, and replacing nodes in the tree

Creating Nodes

- createElement() method of the Document object
- Pass tag name of element as the method argument →tag name is case-insensitive for HTML documents and case sensitive for XML documents

Example:

```
var newElement = document.CreateElement("p") //New Element
var newnode = document.createTextNode("text node content"); //New text node
```

- cloneNode() method that returns a new copy of the node

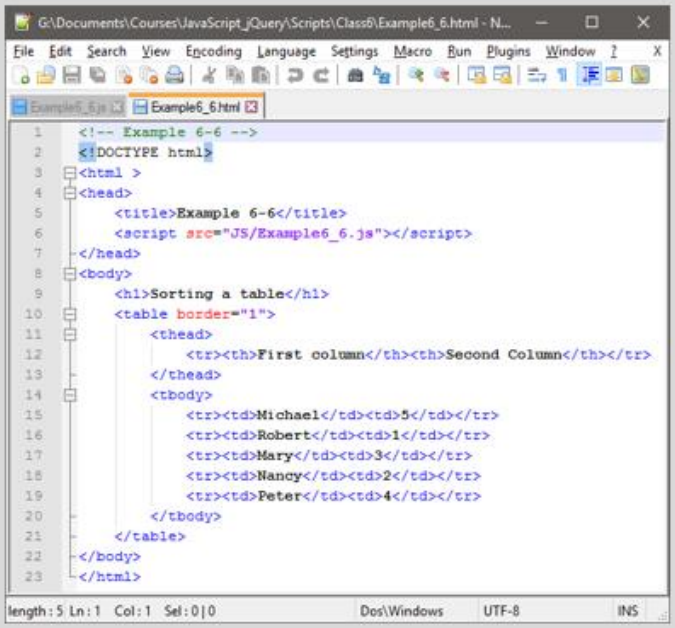
Inserting Nodes

- Node methods appendChild() (becomes last child) or insertBefore() (Specify node to perform insertBefore())

15.5 MANIPULATING DOCUMENT NODES

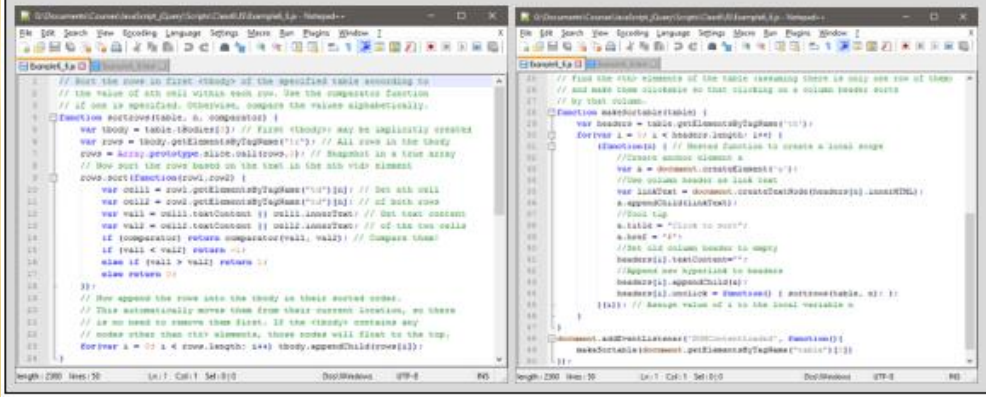
Example 6-6: Creating Hyperlink Table header Elements to Sort a Table

1. Type the following html page in notepad++ or any other text editor.




```
<!-- Example 6-6 -->
<!DOCTYPE html>
<html >
<head>
    <title>Example 6-6</title>
    <script src="JS/Example6_6.js"></script>
</head>
<body>
    <h1>Sorting a table</h1>
    <table border="1">
        <thead>
            <tr><th>First column</th><th>Second Column</th></tr>
        </thead>
        <tbody>
            <tr><td>Michael</td><td>5</td></tr>
            <tr><td>Robert</td><td>1</td></tr>
            <tr><td>Mary</td><td>3</td></tr>
            <tr><td>Nancy</td><td>2</td></tr>
            <tr><td>Peter</td><td>4</td></tr>
        </tbody>
    </table>
</body>
</html>
```

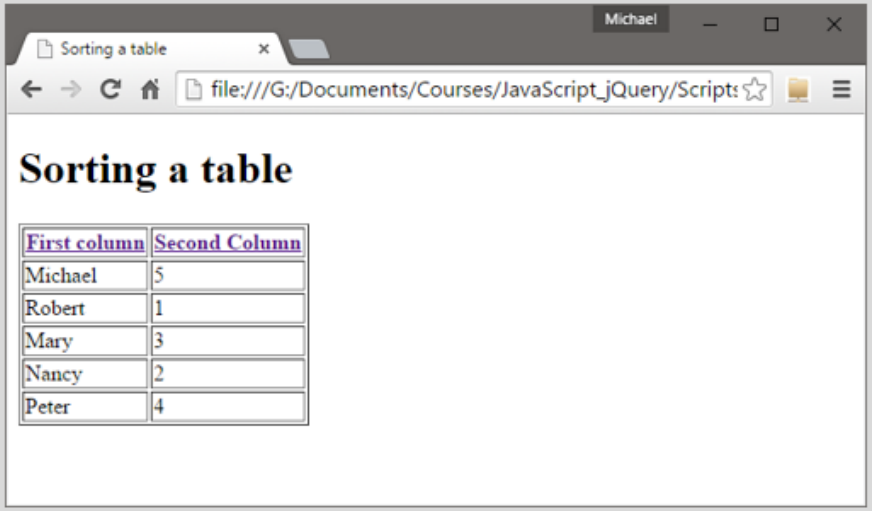
2. Save the page as Example6_6.html.(Make sure not to save it as .txt file).
3. Create the following JavaScript file and save it as Example6_6.js (JS Folder).



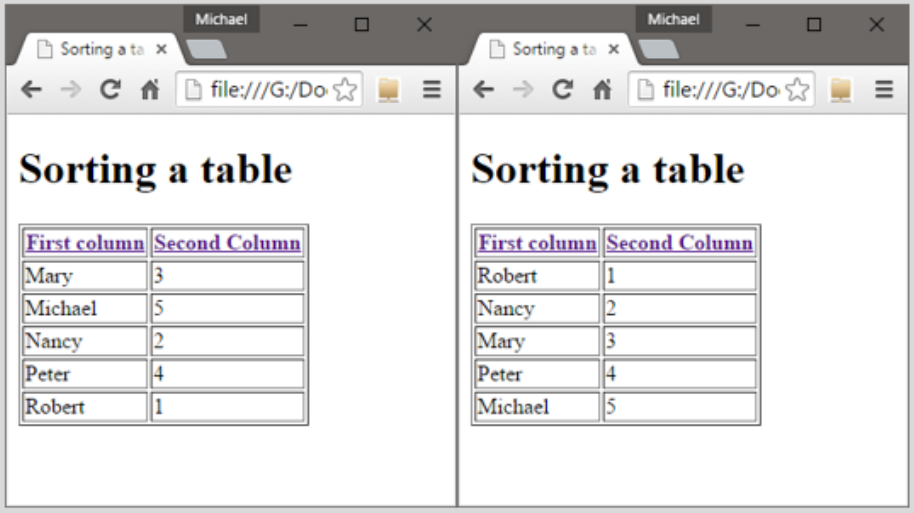
15.5 MANIPULATING DOCUMENT NODES

Example 6-6 (continued): Creating Hyperlink Table header Elements to Sort a Table

4. Double-click on the html file and run it in a browser.



5. Click on both column headers to sort the table by the respective column.



15.5 MANIPULATING DOCUMENT NODES

Removing and Replacing Nodes

- removeChild() method removes a node from the document tree
- This method is not invoked on the node to be removed but (as the "child" part of its name implies) on the parent of that node

Syntax:

```
n.parentNode.removeChild(n);
```

- replaceChild() removes one child node and replaces it with a new one:
  - Invoke this method on the parent node, passing the new node as the first argument and the node to be replaced as the second argument

Example:

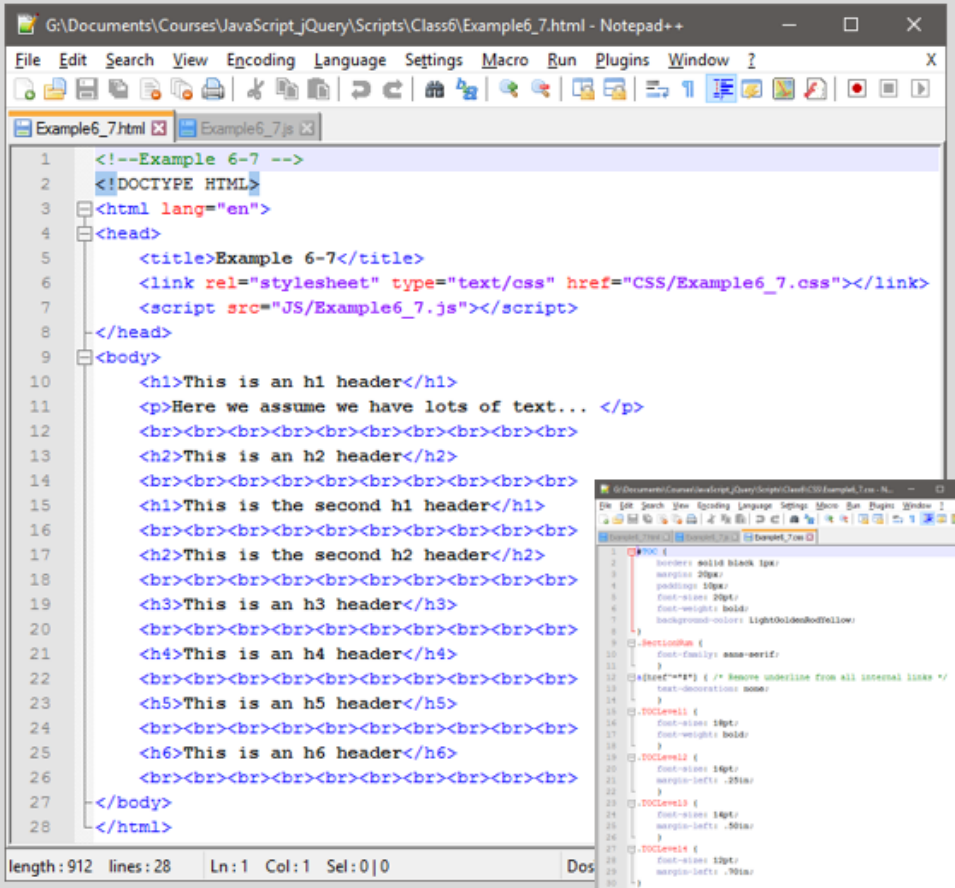
To replace the node n with a string of text:

```
n.parentNode.replaceChild(document.createTextNode("[ REDACTED ]"), n);
```

15.5 MANIPULATING DOCUMENT NODES

Example 6-7: Generating table of Contents

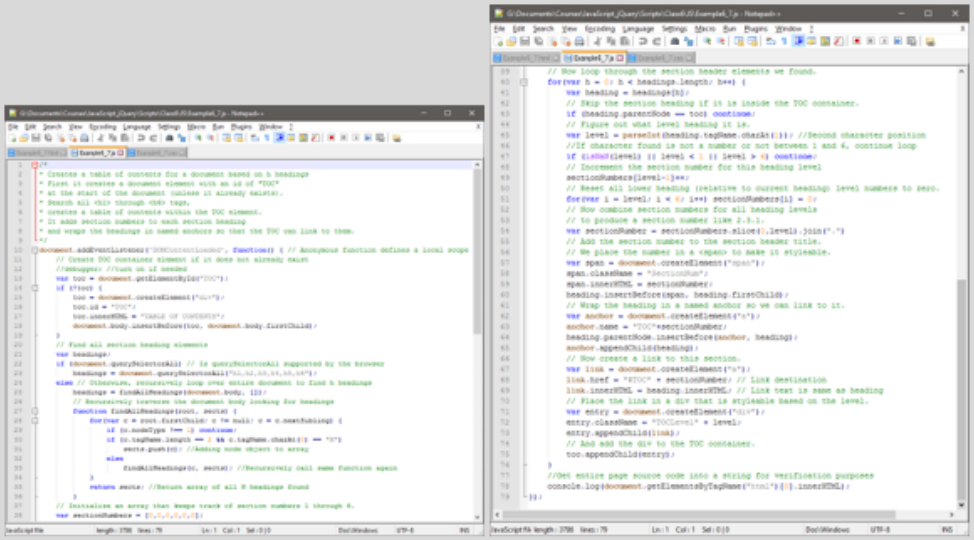
1. Type the following html, JavaScript and CSS page in notepad++ or any other text editor.



2. Save the pages as Example6_7.html and Example6_7.css (in CSS Folder).(Make sure not to save it as .txt file)
3. Create the following JavaScript file and save it as Example6_7.js (in JS Folder).

15.5 MANIPULATING DOCUMENT NODES

Example 6-7 (continued): Generating table of Contents



4. Double-click on the html page and run it in your browser. It should look like shown below:

5. Turn on the Developer Tools and notice the output in the console (last JS statement)

15.6 MANIPULATING HTML FORMS

- HTML <form>element, and the various form input elements, such as <input>, <select>, and <button>, have an important place in client-side programming
- For server-side processing, submit button is necessary
- For client-side, submit is not necessary, but maybe useful

Selecting Form/Form Elements

- Use standard methods to select elements in a form

Example:

```
var fields = document.getElementById("address").getElementsByTagName("input");
```

- With querySelectorAll(), you might select all radio buttons, or all elements with the same name, from a form

Example:

```
// All radio buttons in the form with id "shipping"
document.querySelectorAll('#shipping input[type="radio"]');
// All radio buttons with name "method" in form with id "shipping"
document.querySelectorAll('#shipping input[type="radio"][name="method"]');
```

15.6 MANIPULATING HTML FORMS

- <form> with a name="address“ attribute can be selected:

- window.address // Brittle: do not use
- document.address // Only works for forms with name attribute
- document.forms.address // Explicit access to a form with name or id
- document.forms[n] // Brittle: n is the form's numerical position

- document.forms is an HTMLCollection object that allows form elements to be selected by numerical order, by id, or by name
- Form objects themselves act like HTMLCollections of form elements and can be indexed by name or number
- Name attribute: groups of related checkboxes and mandatory for mutually exclusive groups of radioboxes to share a value of the name attribute.

- document.forms.address[0]
- document.forms.address.street
- document.address.street // only for name="address", not id="address"

- document.forms.address.elements[0]
- document.forms.address.elements.street

Example:

```
<form name="shipping">
<fieldset><legend>Shipping Method</legend>
<label><input type="radio" name="method" value="1st">First-class</label>
<label><input type="radio" name="method" value="2day">2-day Air</label>
<label><input type="radio" name="method" value="overnite">Overnight</label>
</fieldset>
</form>
```

With this form, you might refer to the array of radio button elements like this:

```
var methods = document.forms.shipping.elements.method;
```

15.6 MANIPULATING HTML FORMS

- To determine which shipping method the user has selected, we would loop through the form elements in the array and check the checked property of each:

Example:

```
var shipping_method;
for(var i = 0; i < methods.length; i++)
        if (methods[i].checked) shipping_method = methods[i].value;
```

Form/Elements Properties

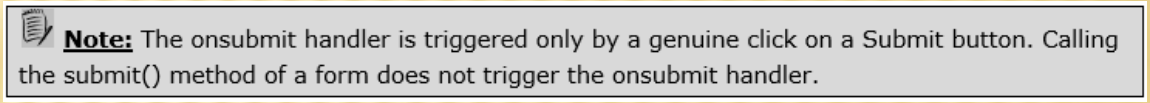
- elements[] array described above is the most interesting property of a Form object
- action, encoding, method, and target properties to control how form data is submitted to the web server and where the results are displayed.
- Client-side JavaScript can set the value of these properties, but they are only useful when the form is actually submitted to a server-side program
- JavaScript Form object supports two methods, submit() and reset() (same as HTML submit and reset)

15.6 MANIPULATING HTML FORMS

- All (or most) form elements have the following properties in common:
  - type: Read-only string that identifies the type of the form element
  - form: Read-only reference to Form object in which element is contained
  - name: Read-only string specified by HTML name attribute
  - value: Read/write string that specifies the “value” contained or represented by form element

Form/Elements Event Handlers

- Each Form element has an onsubmit event handler to detect form submission and an onreset event handler to detect form resets.



- The onsubmit handler is invoked just before the form is submitted; it can cancel the submission by returning false

15.6 MANIPULATING HTML FORMS

- onreset event is invoked just before the form is reset, and it can prevent the form elements from being reset by returning false
- Like the onsubmit handler, onreset is triggered only by a genuine Reset button:
  - Calling the reset() method of a form does not trigger onreset.

Example:

```
<form...
onreset="return confirm('Really erase ALL input and start over?')">
...
<button type="reset">Clear and Start Over</button>
</form>
```

- Form elements typically fire a click or change event when the user interacts with them, and you can handle these events by defining an onclick or onchange event handler
- change event occurs when the user changes the value represented by the element:
  - Happens when user enters text in a text field or selects an option from a drop-down list.
- In event handler, this keyword refers to the document element that triggered the event

15.6 MANIPULATING HTML FORMS

SELECT/OPTION Element

- Browsers typically render Select elements in drop-down menus, but if you specify a size attribute with a value greater than 1, they will display the options in a (possibly scrollable) list instead
- Type property:

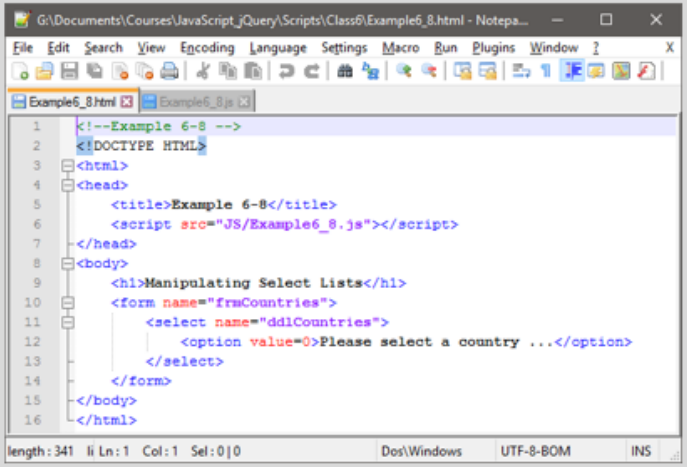
  - If the <select>element has the multiple attribute, the user is allowed to select multiple options, and the type property of the Select object is "select-multiple".
  - If the multiple attribute is not present, only a single item can be selected, and the type property is "select-one".

- When the user selects or deselects an option, the Select element triggers its onchange event handler
- Type=select-one: read/write selectedIndex property specifies which one of the options is currently selected
- Type=select-multiple: Loop through options[] array for selected
- Manipulate options array to dynamically add/delete options

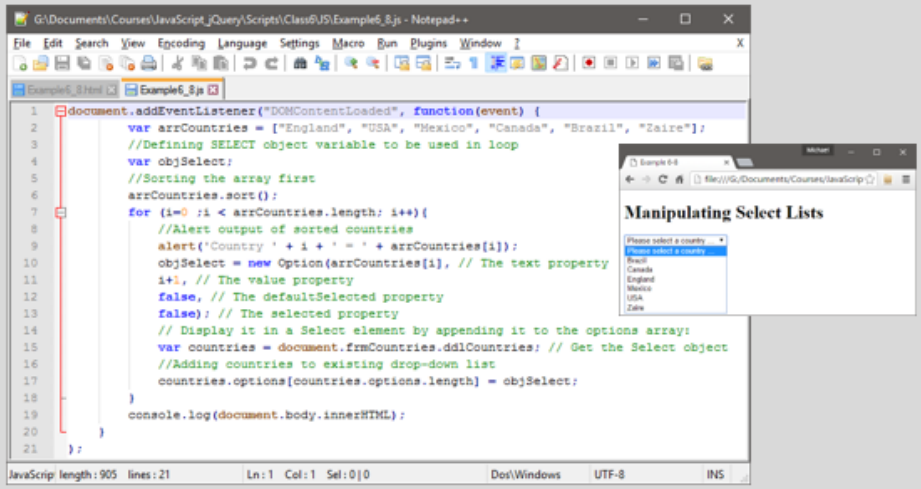
15.6 MANIPULATING HTML FORMS

Example 6-8: Populating Drop-Down List

1. Type the following html page in notepad++ or any other text editor.



2. Save the page as Example6_8.html (Make sure not to save it as .txt file).
3. Create the following JavaScript page:



4. Save the page as Example6_8.js (in JS Folder). Execute the html file.

15.7 DOCUMENT PROPERTIES

- Additional document properties of interest:

- Referrer property: contains the URL of document from which the user linked to the current document

cookie:
A special property that allows JavaScript programs to read and write HTTP cookies.

domain
A property that allows mutually trusted web servers within the same Internet domain to relax same-origin policy security restrictions on interactions between their web pages.

lastModified
A string that contains the modification date of the document.

location
This property refers to the same Location object as the locationproperty of the Window object.

referrer
The URL of the document containing the link, if any, that brought the browser to the current document. This property has the same content as the HTTP Referer header, but it is spelled with a double r.

title
The text between the <title>and <title>tags for this document.

URL
The URL of the document as a read-only String rather than as a Location object. The value of this property is the same as the initial value of location.href, but it is not dynamic like the Location object is. If the user navigates to a new fragment identifier within the document, for example, location.href will change, but document.URL will not.

Example:

```
if (document.referrer.indexOf("http://www.google.com/search?") == 0) {
        var args = document.referrer.substring(ref.indexOf("?")+1).split("&");
        for(var i = 0; i < args.length; i++) {
                if (args[i].substring(0,2) == "q=") {
                document.write("<p>Welcome Google User. ");
                document.write("You searched for: " +
                unescape(args[i].substring(2)).replace('+', ' ');
                break;
                }
        }
}
```