CIS 4004: Web Based Information Technology
Summer 2014

Introduction To JavaScript – Part 3 – More On Events

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
http://www.cs.ucf.edu/courses/cis4004/sum2014

Department of Electrical Engineering and Computer Science
University of Central Florida

JavaScript – Part 3 – More On Events

- To make your web application respond to user actions on the page, you need to do three things:
  - Decide which events should be monitored (listened for).
  - Set up the event handlers that trigger functions when events occur.
  - Write the functions that provide the appropriate responses to the events.
- As you've seen in the previous JavaScript notes, an event is issued as the result of some specific activity – usually user activity, but sometimes browser activity such as a page load – and that you handle the event with an event handler.

JavaScript – Part 3 – More On Events

- An event handler is always the name of the event preceded by "`on`"; for example, the event `click` is handled by the `onclick` event handler.

- The event handler causes a function to run, and the function provides the response to the event.

- The tables on the next two pages lists some of the more commonly used event handlers. For a more complete listing see: http://www.w3.org/TR/DOM-Level-3-Events/

JavaScript – Part 3 – More On Events

Event Category	Event Triggered When…	Event Handler
Browser Events	Page completes loading	`onload`
	Page is removed from browser window	`onunload`
	JavaScript throws an error	`onerror`
Mouse Events	User clicks over an element	`onclick`
	User double-clicks over an element	`obdblclick`
	The mouse button is pressed down over an element	`onmousedown`
	The mouse button is released over an element	`onmouseup`
	The mouse pointer moves onto an element	`onmouseover`
	The mouse pointer leaves an element	`onmouseout`

JavaScript – Part 3 – More On Events

Event Category	Event Triggered When…	Event Handler
Keyboard Events	A key is pressed	`onkeydown`
	A key is released	`onkeyup`
	A key is pressed and released	`onkeypress`
Form Events	The element receives focus	`onfocus`
	The element loses focus	`onblur`
	The user selects type in text or text area field	`onselect`
	User submits a form	`onsubmit`
	User resets a form	`onreset`
	Field loses focus and content has changed since receiving focus	`onchange`

Inline Event Handlers

- An event handler can be utilized inline by attaching the event handler directly to an element, such as:

  ```
  <input type="text" onblur="doValidate()" />
  ```

  In this case, a form text field has the JavaScript function `doValidate()` associated with its `blur` event – the function will be called when the user moves the cursor out of the field by pressing Tab or clicks elsewhere. The function could then check if the user typed something in the field or not.

- While inline event handlers have been used for a number of years, they are not ideal as they mix the HTML and the JavaScript, and these should be separate. In a modern web application – in the interests of accessibility, maintainability, and reliability – you want to keep JavaScript and CSS out of your HTML markup.

The Handler As An Object Property

- The example below illustrates the way that we’ve mostly thus far have utilized the event handlers in our JavaScript examples (see JavaScript – Part 2 notes).

```
var clickableImage = document.getElementById(“dog_pic”);

clickableImage.onclick = showLargeImage;
```

- In this example, first the object representing the HTML element with the `id` = “dog_pic” is assigned to the variable `clickableImage`. The event handler `onclick` is assigned as a property of the object, using a function name as the `onclick` property’s value. The function `showLargeImage` will run when the user click on the element with the `id` = “dog_pic”.

The Handler As An Object Property

- The technique shown on the previous page has the desirable property of keeping the JavaScript out of the markup since this would appear only in an external JavaScript file and not in the markup.
- However, there are a couple of rather serious drawbacks to this approach.
- First, only one event at a time can be assigned using this technique, because only one value can exist for a property at any given time. You can't assign another event to the `onclick` property without overwriting this one, and for the same reason, another event that was previously assigned is overridden by this one.

The Handler As An Object Property

- Second, when the user click on this element and the function is called, the function has to be hard-coded with the name of the object so that it knows which element to work on.

```
function showLargeImage() {

   thePicture = document.getElementById(“dog_pic”);

   //do something with the picture

}
```

- If you change the object that is the source of the event, you will also need to modify the function.

The Handler As An Object Property

- For the two reasons just explained, the "handler as an object property" technique is suitable for use only when you just want to assign one event to one object, such as running an initial `onload` function once the page is first loaded.

- This technique does not really provide a robust solution for use throughout a RIA (Rich Interface Application, i.e. web pages with user interaction often incorporating AJAX – later this semester), where events commonly get assigned and removed from objects as the application runs.

- In almost every such case, the best way to manage events is to use event listeners.

Event Listeners

- Event listeners were introduced with the DOM model and provide comprehensive event registration.

- An event listener does what its name suggests: After being attached to an object (a node in the DOM), it then listens patiently for its event to occur. When it "hears" its event, it then calls its associated function in the same manner as the " handler as an object property" method but with two important distinctions.

Event Listeners

- First, *an event listener passes an event object containing information about its triggering event to the function it calls*.
- Within the function, you can read this object's properties to determine the target element, the type of event that occurred – such as `click`, `focus`, `mousedown` – and other details about the event.
- This capability can reduce coding considerably, because you can write very flexible functions for key tasks, such as handling clicks, that provide variations in the response depending on the calling object and triggering event. Otherwise, you would have to write a separate, and probably very similar, function for every type of event you need to handle.

Event Listeners

- Second, *you can attach multiple event listeners to a single object.*
- As a result, you don't have to worry when adding one listener that you are overwriting another that was added earlier, as you would when assigning an event as an object property.
- Both W3C-compliant browsers and Microsoft browsers enable event handlers, they differ in how those handlers are attached to element and in the way they provide access to the event object.
- We'll focus on the W3C approach, which will be the de facto standard in the future. I'll show you both techniques as well as a work around that will enable the JavaScript to determine which browser the user is using.

Event Listeners – W3C Technique

- The W3C technique for adding/registering an event handler is the method `addEventListener()` which takes three arguments:
  - The first is the name of the event for which you are registering the handler.
  - The second is the function that will be called to handle the event.
  - The third is either "true" or "false". Typically, this will be false. When true is used this relates to event bubbling (covered later).
- An example is shown on the next page.

Event Listeners – W3C Technique

- Example:

```
emailField=document.getElementById(“email”);

emailField.addEventListener(‘focus’, doHighlight, false);

email.Field.addEventListener(‘blur’, doValidate, false);
```

Get the object

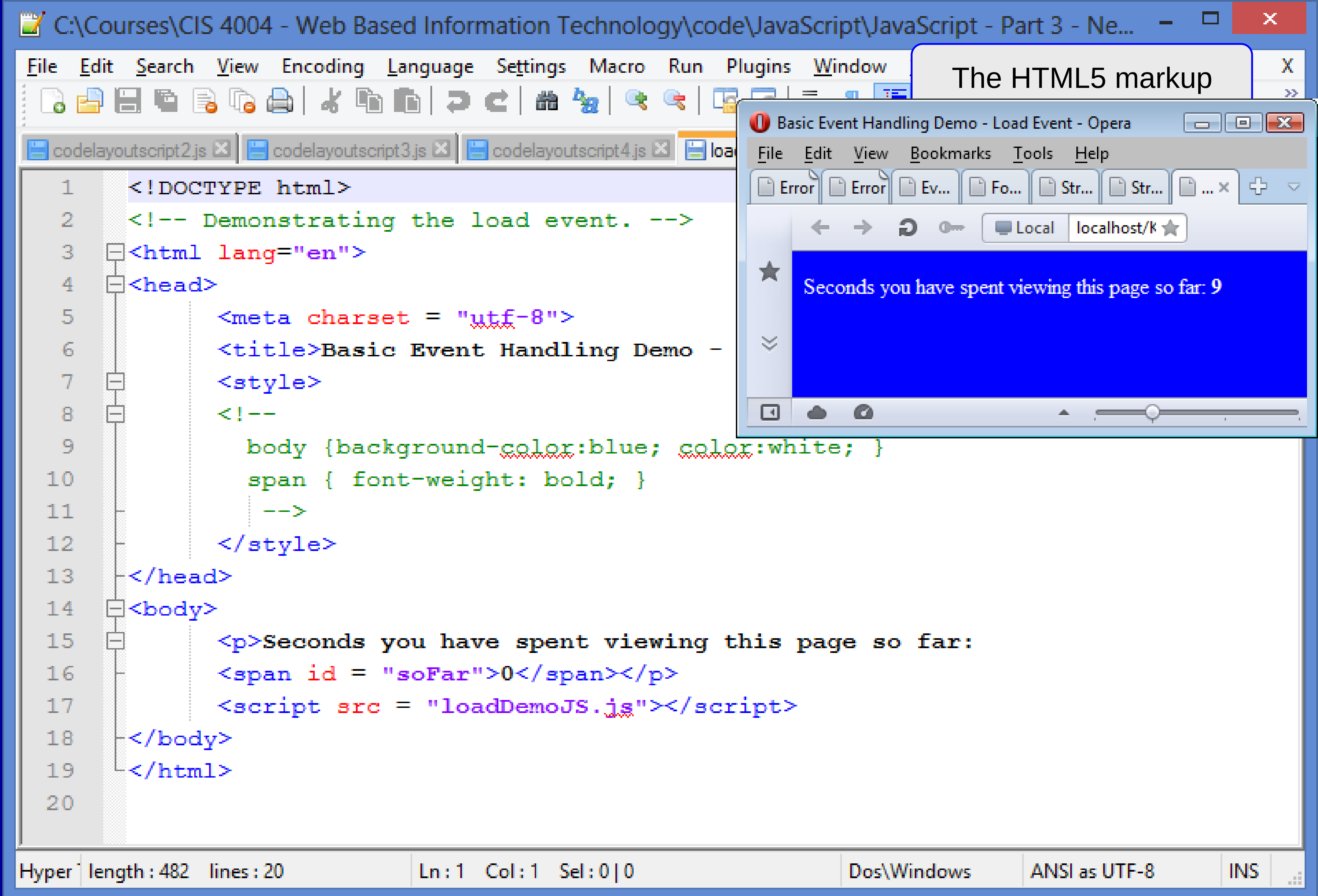
Add a focus listener

Add a blur listener

- The function `doHighlight` would be called when the cursor moves into the field, and the function `doValidate` would be called when the cursor moves out of the field.

- As many event listeners as you would like can be attached to an object in this fashion.

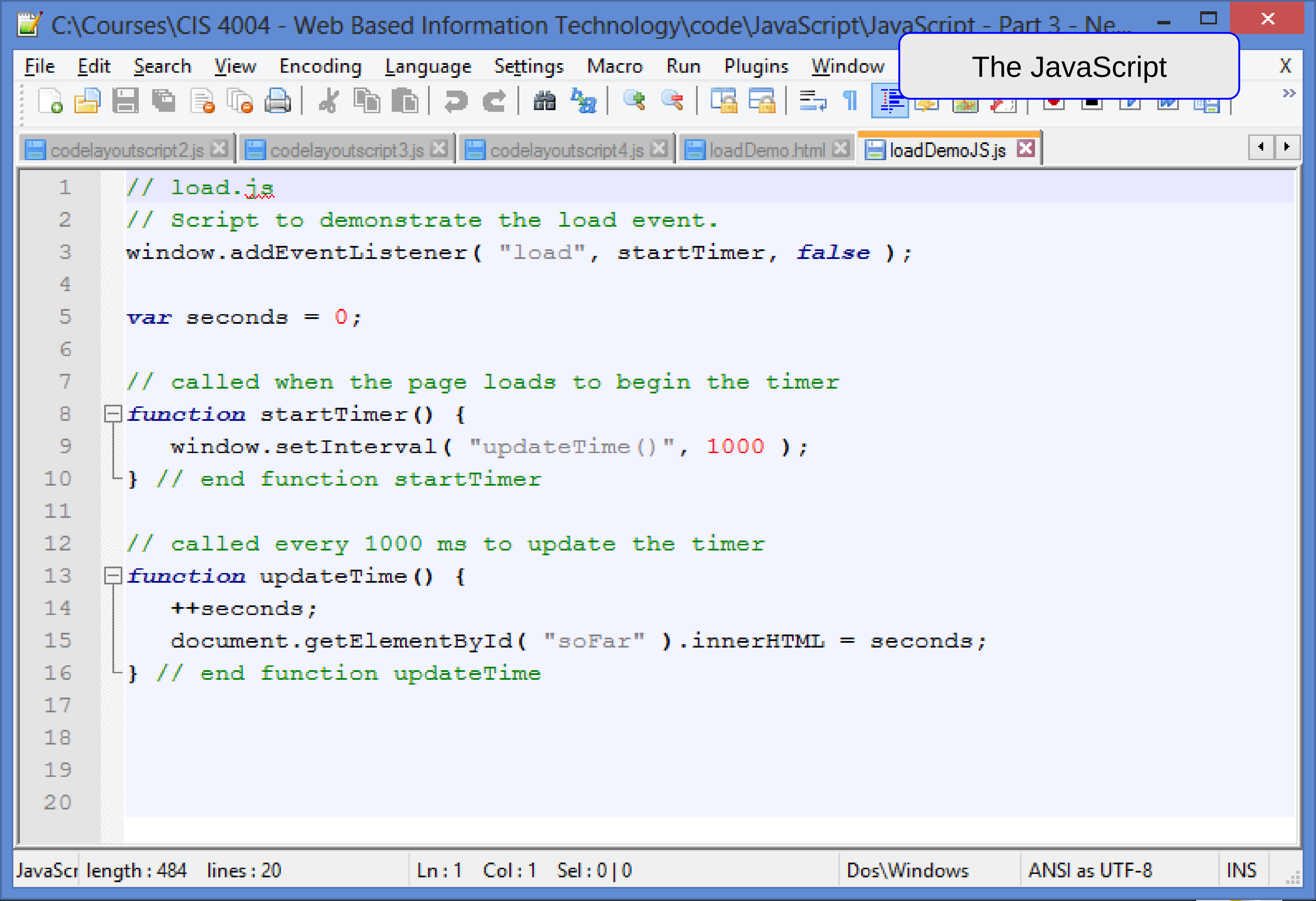
- The next two pages illustrate simple event handler registration.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - Ne...
File Edit Search View Encoding Language Settings Macro Run Plugins Window X
codelayoutscript2.js
codelayoutscript3.js
codelayoutscript4.js
loa
Basic Event Handling Demo - Load Event - Opera
File Edit View Bookmarks Tools Help
Error
Error
Ev...
Fo...
Str...
Str...
...
Local
localhost/k
Hyper length : 482 lines : 20
Ln : 1 Col : 1 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

The HTML5 markup

```
<!DOCTYPE html>
<!-- Demonstrating the load event. -->
<html lang="en">
<head>
      <meta charset = "utf-8">
      <title>Basic Event Handling Demo –
      <style>
      <!--
        body {background-color:blue; color:white; }
        span { font-weight: bold; }
         -->
      </style>
</head>
<body>
      <p>Seconds you have spent viewing this page so far:
      <span id = "soFar">0</span></p>
      <script src = "loadDemoJS.js"></script>
</body>
</html>
```

Seconds you have spent viewing this page so far: **9**



The JavaScript

```javascript
// load.js
// Script to demonstrate the load event.
window.addEventListener( "load", startTimer, false );

var seconds = 0;

// called when the page loads to begin the timer
function startTimer() {
   window.setInterval( "updateTime()", 1000 );
} // end function startTimer

// called every 1000 ms to update the timer
function updateTime() {
   ++seconds;
   document.getElementById( "soFar" ).innerHTML = seconds;
} // end function updateTime
```

Event Listeners – W3C Technique

- Event listeners can be removed (unregistered) in a similar manner by using the `removeEventListener` method.

- Example:

Get the object

Remove the focus listener

```
emailField=document.getElementById(“email”);

emailField.removeEventListener(‘focus’, doHighlight, false);

email.Field.removeEventListener(‘blur’, doValidate, false);
```

Remove the blur listener

Event Listeners – Microsoft Technique

- Microsoft's event registration model is slightly different than the W3C technique.

W3C: `emailField.addEventListener('focus', doHighlight, false);`

Microsoft: `emailField.attachEvent('onfocus', doHighlight);`

- Similary, Microsoft's event listener removal is also slightly different than the W3C technique.

W3C: `emailField.removeEventListener('focus', doHighlight, false);`

Microsoft: `emailField.detachEvent('onfocus', doHighlight);`

Adding Event Listeners

- For the time being, at least until IE either disappears or becomes W3C-compliant (not likely!), you will need to write your JavaScript to add event listeners in the correct format for the browser being used by your visitor.

- Fortunately, John Resig (the guy who developed jQuery) has written a couple of helper functions that will allow your JavaScript to determine the correct event model to use.

- The next two pages illustrate these two functions and I will also place them on the course web page for you to download and use. From a JavaScript perspective the functions are a little complex, so don’t worry if you don’t fully understand how they work. Remember that this is the beauty of “black boxing”.

John Resig’s `addEvent` Helper Function

```
function addEvent( obj, type, fn ) {
  if ( obj.attachEvent ) {
    obj['e'+type+fn] = fn;
    obj[type+fn] = function(){obj['e'+type+fn]  (
window.event );}
    obj.attachEvent( 'on'+type, obj[type+fn] );
  } else
    obj.addEventListener(type, fn, false);
}
```

John Resig’s `removeEvent` Helper Function

```
function removeEvent( obj, type, fn ) {
  if ( obj.detachEvent ) {
    obj['e'+type+fn] = fn;
    obj.detachEvent( 'on'+type, obj[type+fn] );
    obj[type+fn] = null;
  } else
    obj.removeListener(type, fn, false);
}
```

Using John Resig's Helper Functions

- Black boxing means that you don't need to understand how John Resig's functions work, just know what they do and how to use them.

- If you want to add an event listener to the email field form in the previous example, all you would need to do is call the `addEvent` helper function like this:

```
addEvent(emailField, ‘focus’, doHighLight);
```

- The three arguments are the element, the event, and the function to call when the element receives that event. Resig's function then takes care of formatting the event registration appropriately for the browser on which it is running. I'll use Resig's functions from this point on.

The First Event: `load`

- Typically, the first thing you want JavaScript to do is set up the initial state of the page so its ready for use by the visitor.

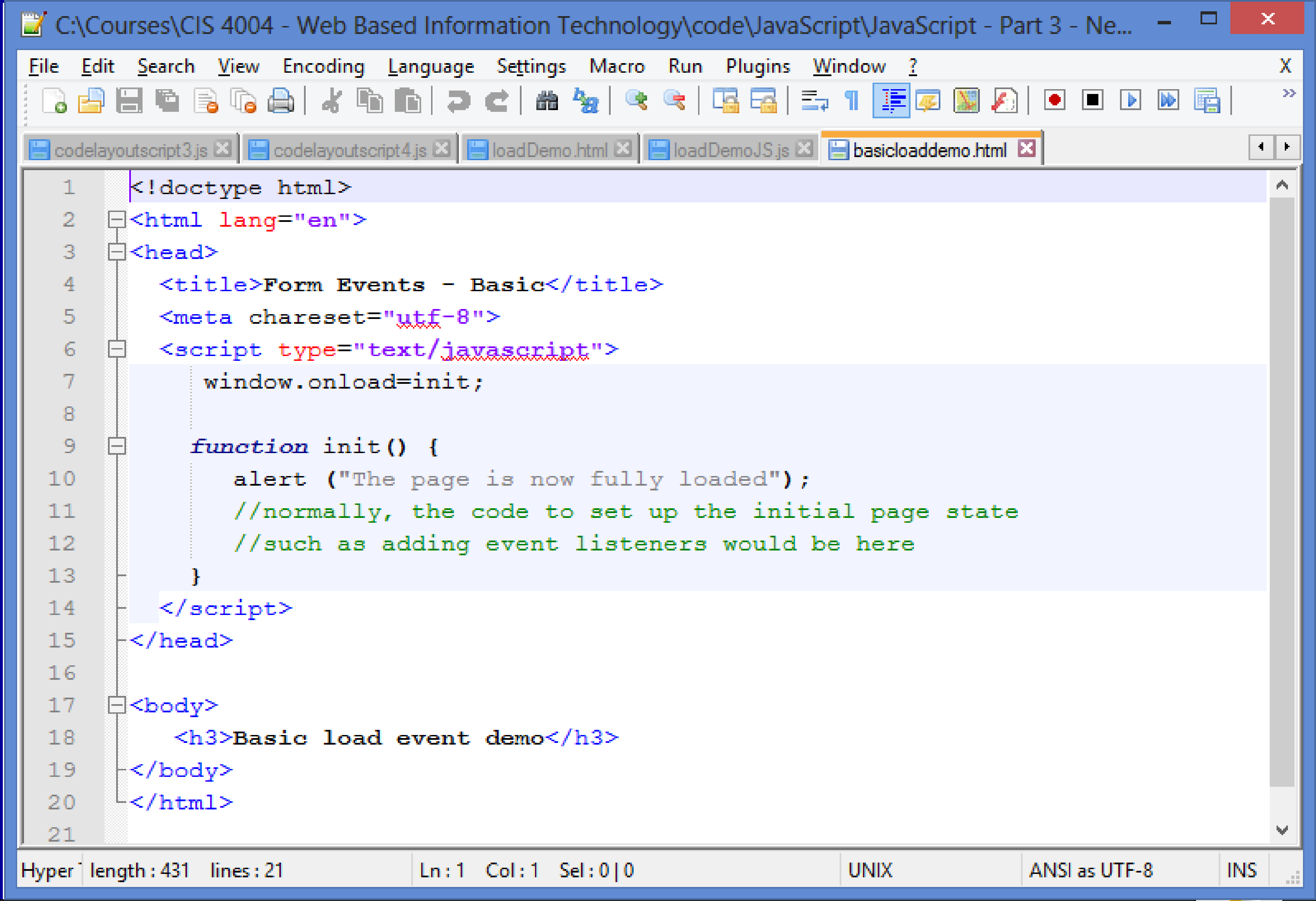
- A very common part of this initialization process is to attach event listeners to the elements in the DOM that will respond to user actions, and you cannot do that until the DOM has loaded into the browser.

- For example, you might want to attach `blur` events to the text fields of a form so you can detect when the user click or tabs away from them. You can then immediately validate the text the user entered.

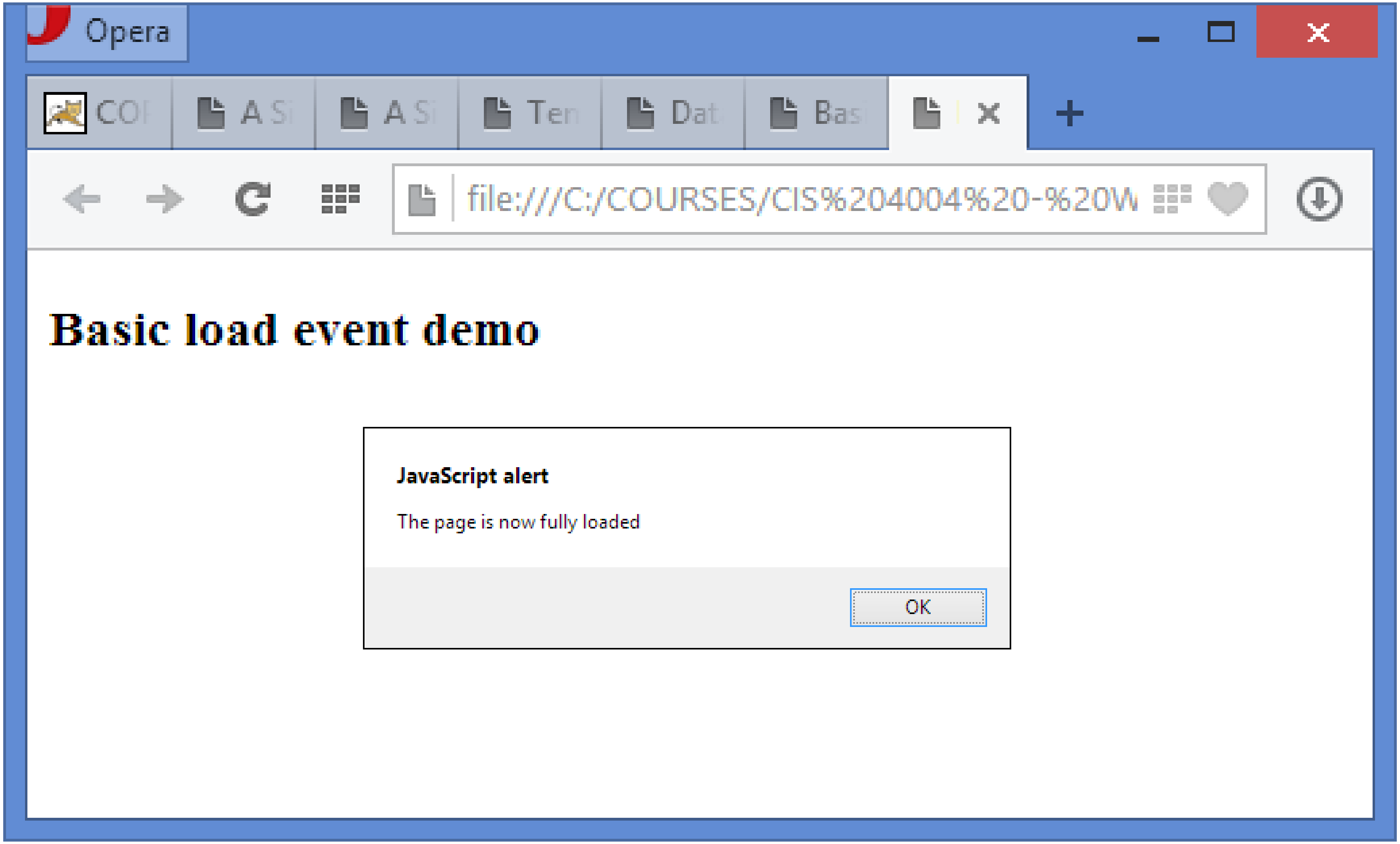
- To help you ensure that you are working with a DOM that actually exists, a load event is issued when the page is entirely loaded.

The First Event: `load`

- You can use the `onload` event handler to detect this event and trigger the JavaScript functions that will set up the page state for the user.
- The example on the next page illustrates this technique.
- Notice that in the JavaScript that the first line calls the `init` function; there are no parentheses after the `init` function name. You would normally add parentheses after a function name because you would want the function to run immediately at that point in the code.
- However, because you are setting up an event that will call the function at a later time, you don't do that here.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - Ne...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
codelayoutscript3.js
codelayoutscript4.js
loadDemo.html
loadDemoJS.js
basicloaddemo.html
Hyper length : 431 lines : 21
Ln : 1 Col : 1 Sel : 0 | 0
UNIX
ANSI as UTF-8
INS

```
<!doctype html>
<html lang="en">
<head>
  <title>Form Events - Basic</title>
  <meta chareset="utf-8">
  <script type="text/javascript">
     window.onload=init;

    function init() {
       alert ("The page is now fully loaded");
       //normally, the code to set up the initial page state
       //such as adding event listeners would be here
    }
  </script>
</head>

<body>
   <h3>Basic load event demo</h3>
</body>
</html>

```



Basic load event demo

JavaScript alert

The page is now fully loaded

The First Event: `load`

- If you wrote `window.onload= init();` the function would run immediately (setting the `onload` property to the result of the function) and not wait for the page load event to be sent.

- By omitting the parentheses when you assign the `init` function to the `onload` property, the function does not run immediately, instead, it runs when the load event occurs after the page is fully loaded.

- Also note that `onload` is a method of the `window` object, so you must always precede it with `window`, for it to work.

- Note too, that any JavaScript statement not enclosed in a function and just "loose" on the page runs as soon as it loads. For this reason, it's very unusual to place any JavaScript except the `onload` event assignment outside of a function.

Adding Event Listeners on Page Load

- After all of the previous discussion, we'll now look at a simple example of event listeners that are added to an element when the page loads.
- In this example case, when the `onload` event handler calls the `init` function, it will add event listeners to a text field.
- As a result of the functions called by these event listeners, the text field will highlight (its background will be set to green) when the field receives focus; it will unhighlight (the default white background be restored) when the focus is removed.
- We'll develop this example in a systematic manner which might help you with the techniques you can use in developing your own projects.

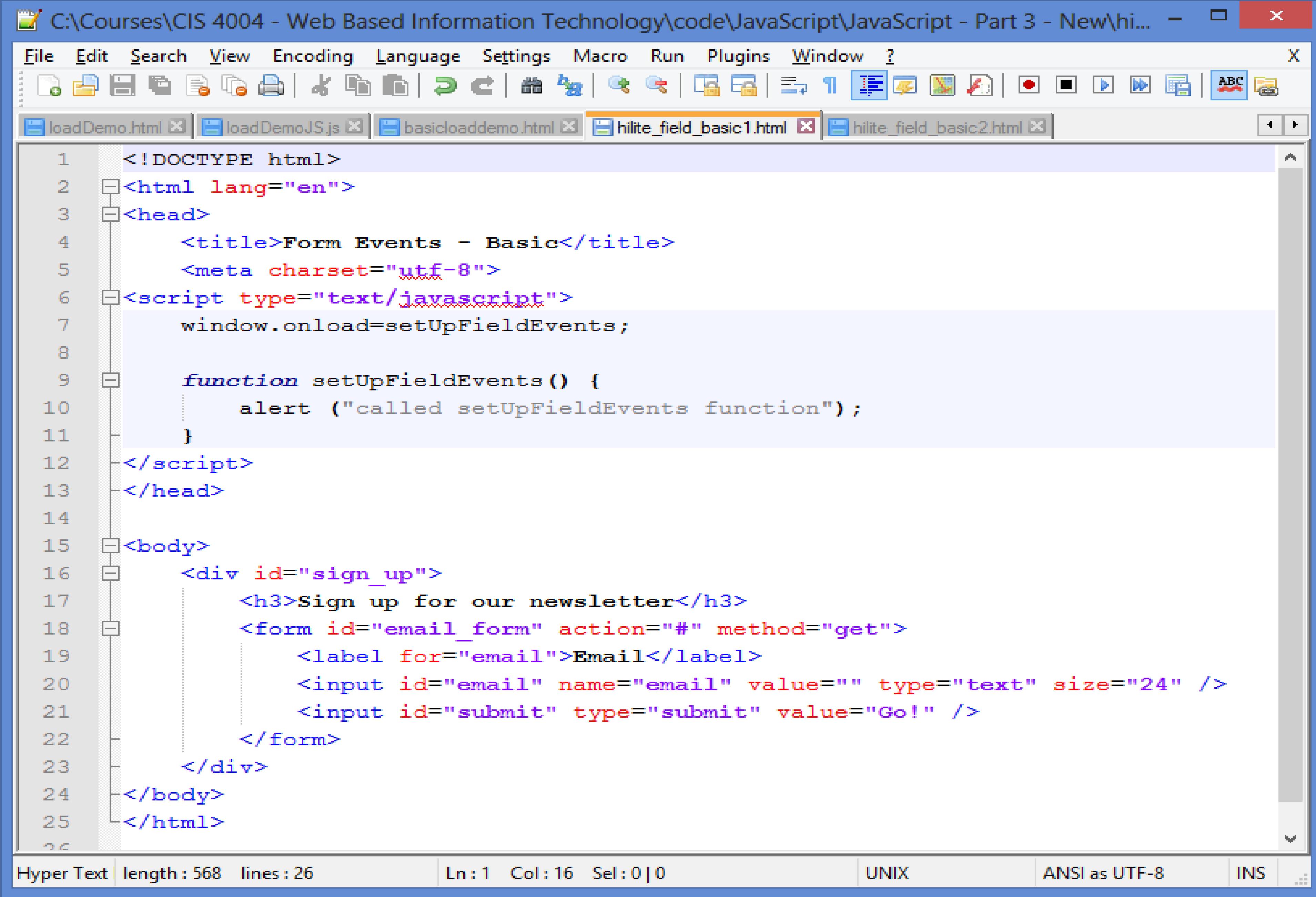
Adding Event Listeners on Page Load

- Step 1 in the development process is to ensure that the load event is triggering the function that will set up the event listeners.

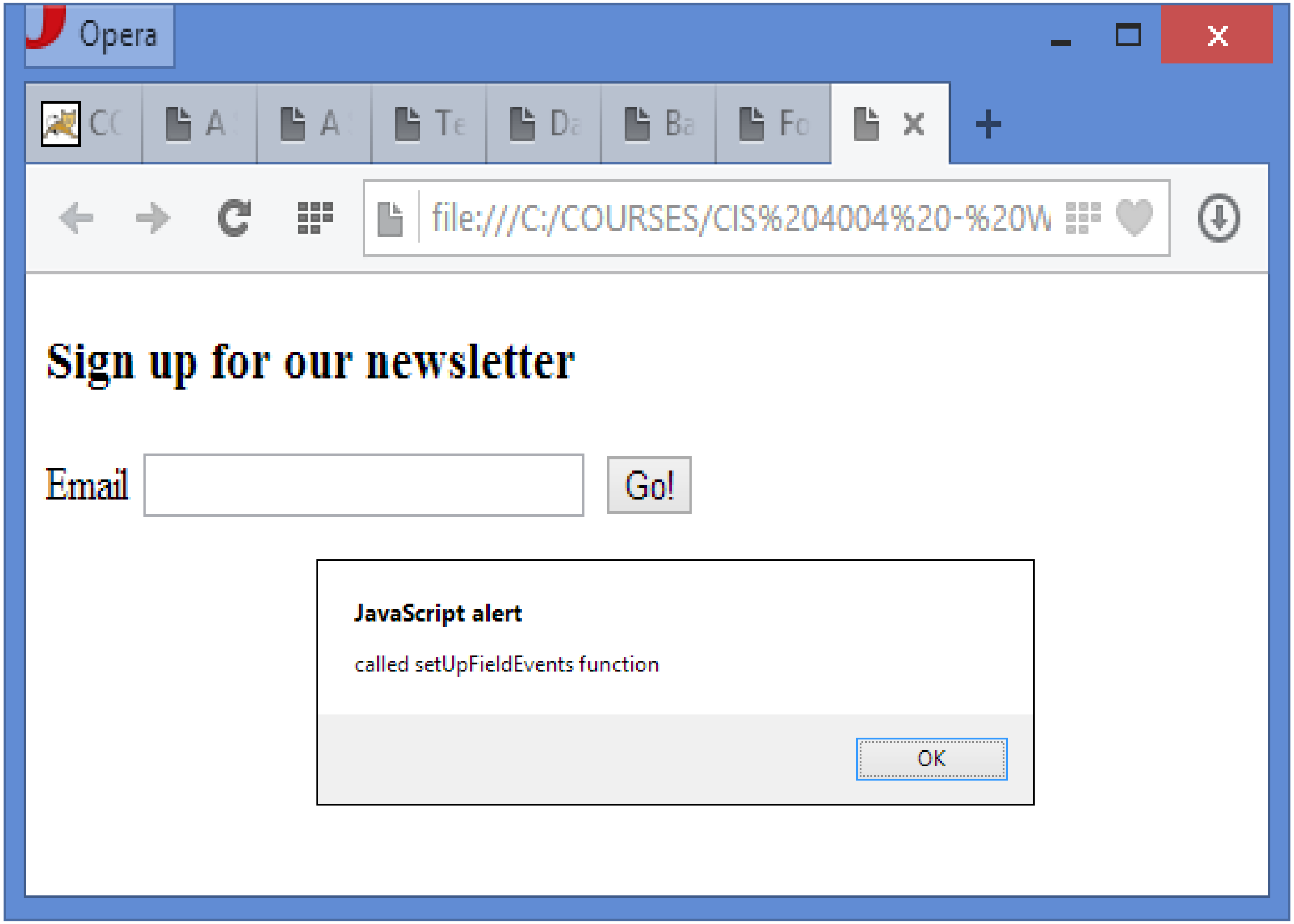
- The markup for this example is shown on the next page, but the only significant element is the form input field.

- Notice that all I did was set up the `onload` event to trigger the function `setUpFieldEvents`. In order to ensure that the function is being called properly, I just used a JavaScript alert box to display. So I now know that the function is being triggered properly by the `onload` event.

- As with some of the other examples, I'm including the JavaScript in the markup file for ease of viewing here…normally it would be external to the markup.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\hi...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
loadDemo.html
loadDemoJS.js
basicloaddemo.html
hilite_field_basic1.html
hilite_field_basic2.html
Hyper Text length : 568 lines : 26
Ln : 1 Col : 16 Sel : 0 | 0
UNIX
ANSI as UTF-8
INS

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Form Events - Basic</title>
    <meta charset="utf-8">
<script type="text/javascript">
    window.onload=setUpFieldEvents;

    function setUpFieldEvents() {
        alert ("called setUpFieldEvents function");
    }
</script>
</head>

<body>
    <div id="sign_up">
        <h3>Sign up for our newsletter</h3>
        <form id="email_form" action="#" method="get">
            <label for="email">Email</label>
            <input id="email" name="email" value="" type="text" size="24" />
            <input id="submit" type="submit" value="Go!" />
        </form>
    </div>
</body>
</html>
```



Sign up for our newsletter

Email Go!

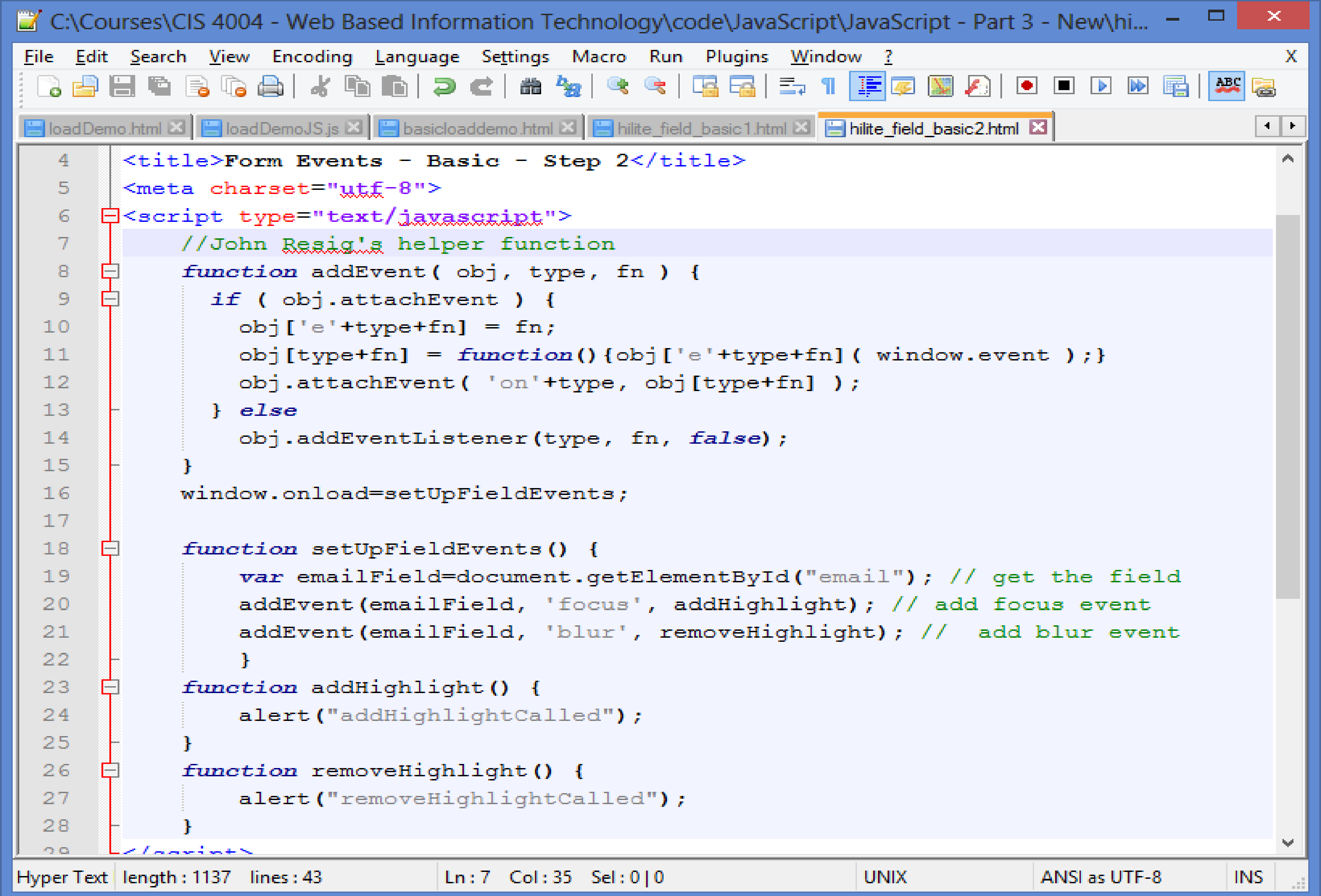
JavaScript alert

called setUpFieldEvents function

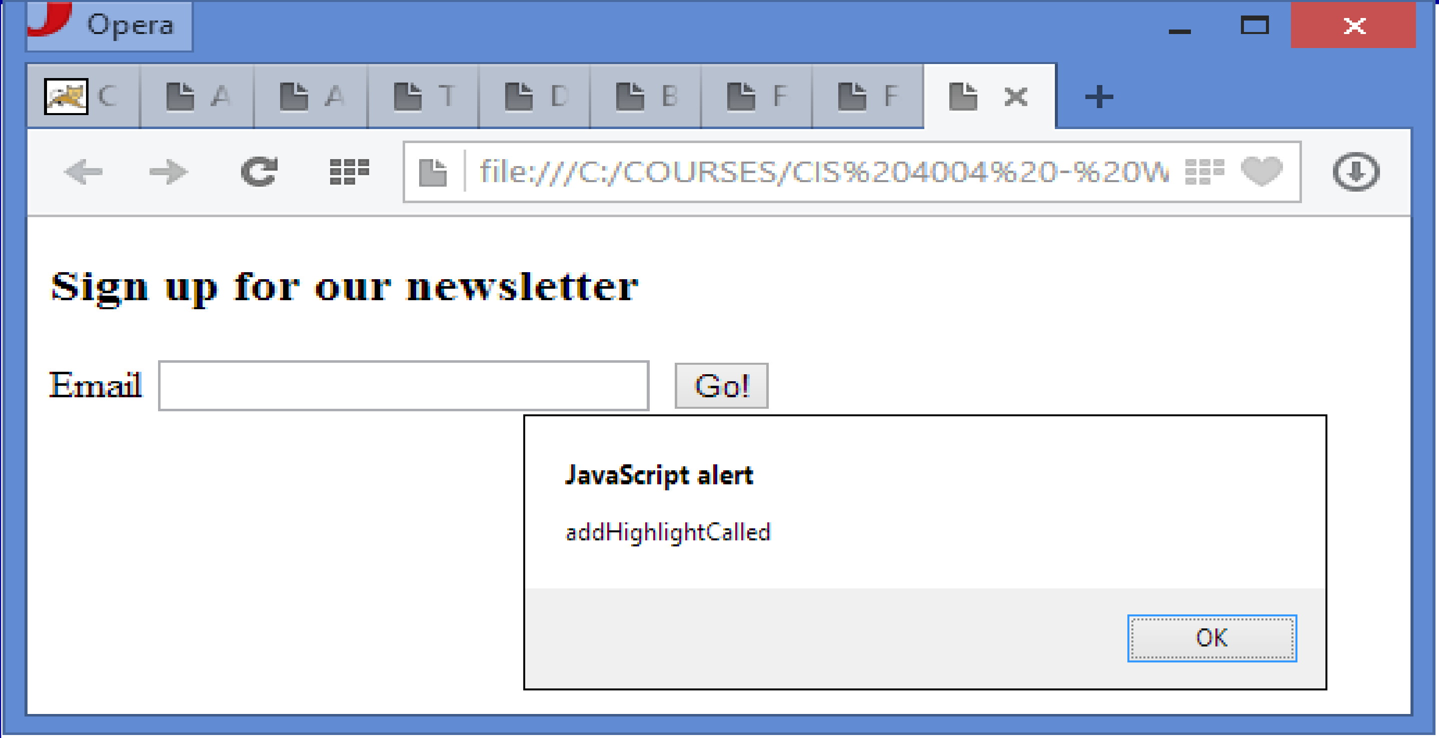
OK

Adding Event Listeners on Page Load

- In step 2 we'll actually add the code to the `setUpFieldEvents` function that will add the event listeners. We'll use Resig's `addEvent` helper function to ensure that our page will render properly in any browser.
- The markup is shown on the next page.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\hi...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
loadDemo.html
loadDemoJS.js
basicloaddemo.html
hilite_field_basic1.html
hilite_field_basic2.html
Hyper Text
length : 1137 lines : 43
Ln : 7 Col : 35 Sel : 0 | 0
UNIX
ANSI as UTF-8
INS

```
<title>Form Events - Basic - Step 2</title>
<meta charset="utf-8">
<script type="text/javascript">
    //John Resig's helper function
    function addEvent( obj, type, fn ) {
      if ( obj.attachEvent ) {
        obj['e'+type+fn] = fn;
        obj[type+fn] = function(){obj['e'+type+fn]( window.event );}
        obj.attachEvent( 'on'+type, obj[type+fn] );
      } else
        obj.addEventListener(type, fn, false);
    }
    window.onload=setUpFieldEvents;

    function setUpFieldEvents() {
        var emailField=document.getElementById("email"); // get the field
        addEvent(emailField, 'focus', addHighlight); // add focus event
        addEvent(emailField, 'blur', removeHighlight); //  add blur event
        }
    function addHighlight() {
        alert("addHighlightCalled");
    }
    function removeHighlight() {
        alert("removeHighlightCalled");
    }
</script>
```



Sign up for our newsletter

Email

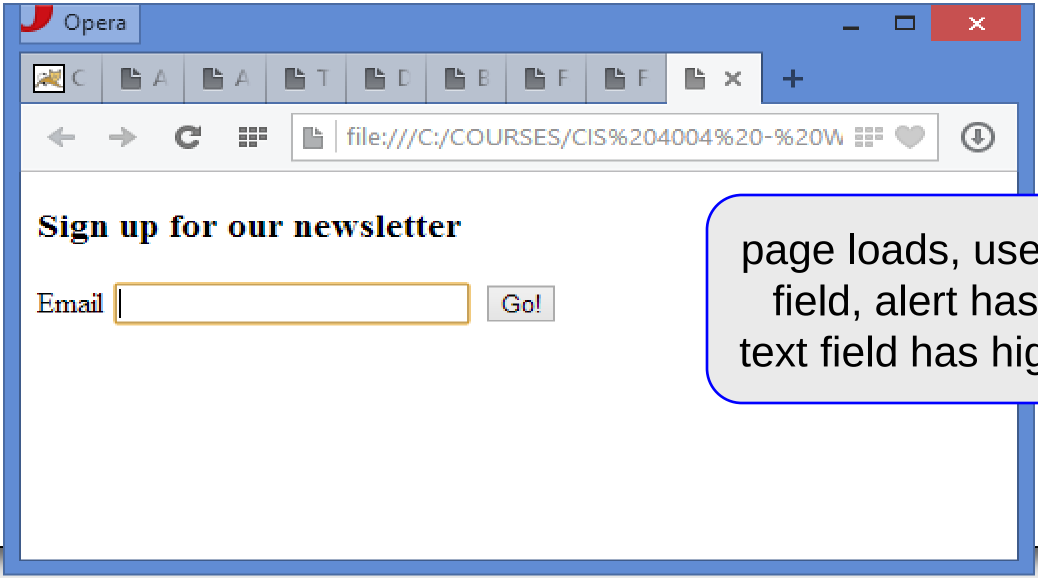
Go!

JavaScript alert

addHighlightCalled

OK

page loads, user clicks in text field

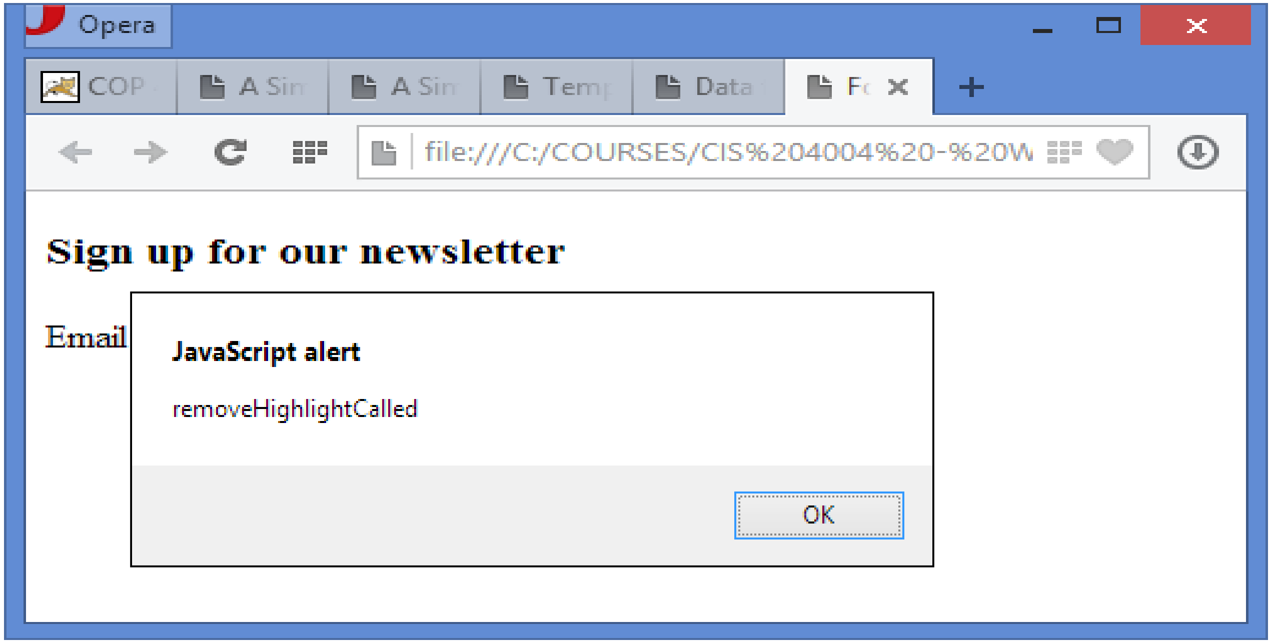


Sign up for our newsletter

Email

Go!

page loads, user clicks in text field, alert has displayed – text field has highlight applied



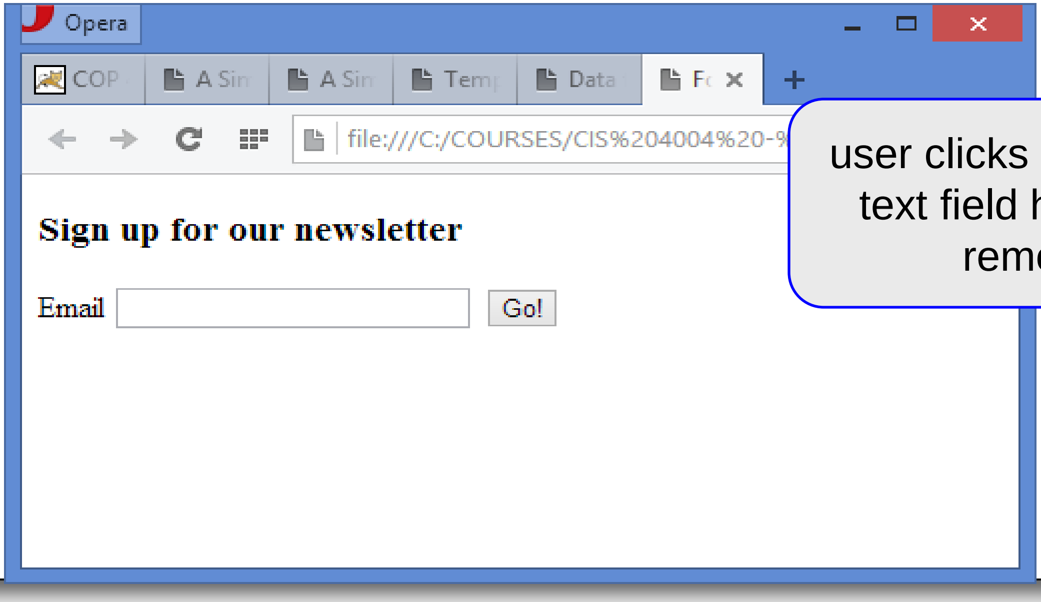
Sign up for our newsletter

Email

JavaScript alert

removeHighlightCalled

user clicks outside of text field, alert displays indicating loss of focus.



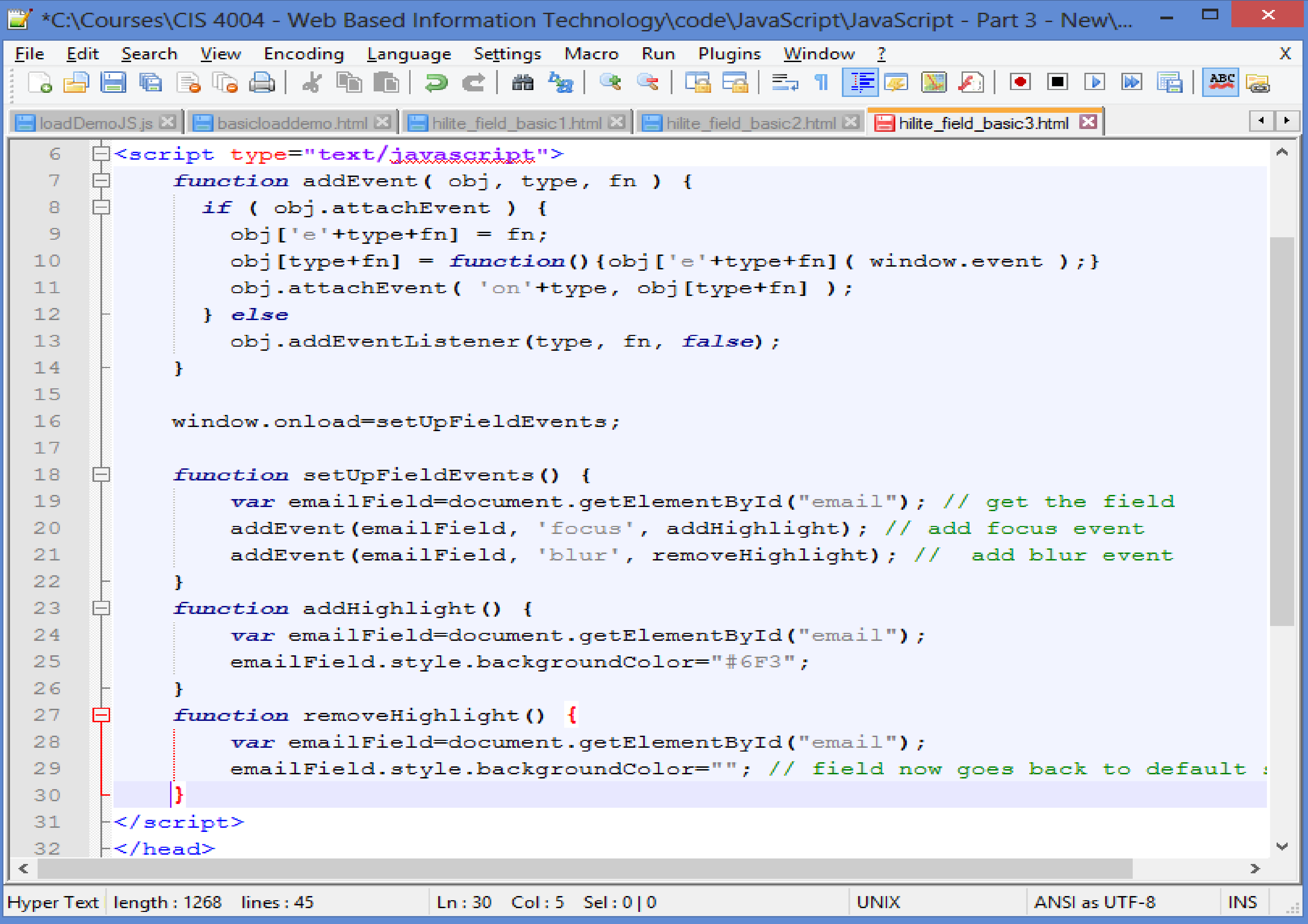
Sign up for our newsletter

Email

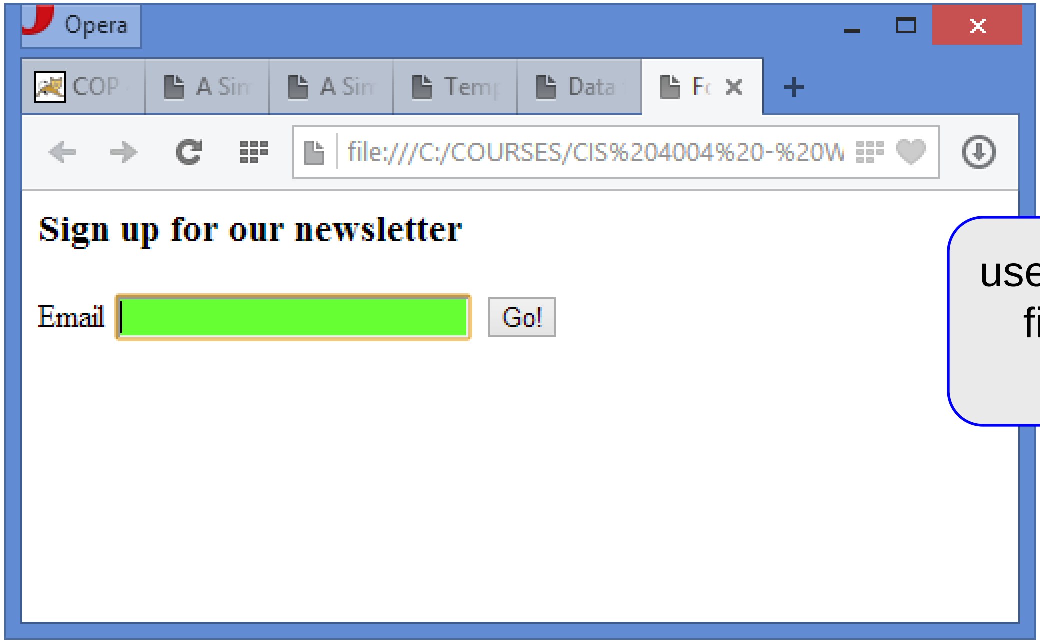
user clicks ok in alert – text field highlight is removed

Adding Event Listeners on Page Load

- In step 3 we’ll replace the alert code in the functions with the actual highlighting that we originally intended.
- The markup is shown on the next page.


*C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
loadDemoJS.js
basicloaddemo.html
hilite_field_basic1.html
hilite_field_basic2.html
hilite_field_basic3.html
Hyper Text
length : 1268 lines : 45
Ln : 30 Col : 5 Sel : 0 | 0
UNIX
ANSI as UTF-8
INS

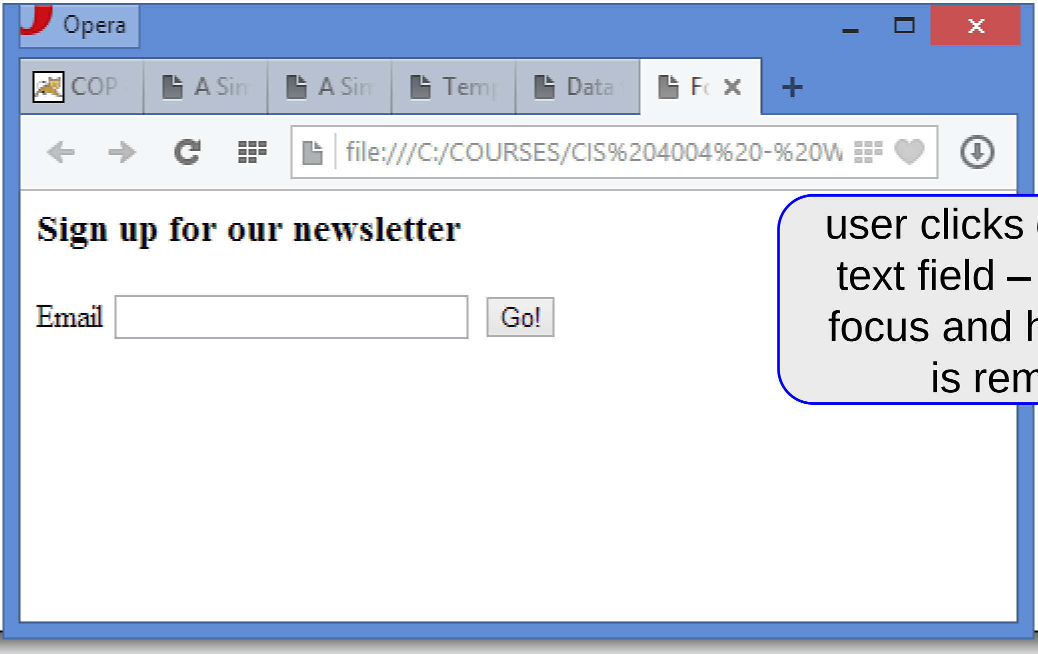
```
<script type="text/javascript">
    function addEvent( obj, type, fn ) {
      if ( obj.attachEvent ) {
        obj['e'+type+fn] = fn;
        obj[type+fn] = function(){obj['e'+type+fn]( window.event );}
        obj.attachEvent( 'on'+type, obj[type+fn] );
      } else
        obj.addEventListener(type, fn, false);
    }

    window.onload=setUpFieldEvents;

    function setUpFieldEvents() {
        var emailField=document.getElementById("email"); // get the field
        addEvent(emailField, 'focus', addHighlight); // add focus event
        addEvent(emailField, 'blur', removeHighlight); //  add blur event
    }
    function addHighlight() {
        var emailField=document.getElementById("email");
        emailField.style.backgroundColor="#6F3";
    }
    function removeHighlight() {
        var emailField=document.getElementById("email");
        emailField.style.backgroundColor=""; // field now goes back to default
    }
</script>
</head>
```



Sign up for our newsletter

Email

user clicks in text field – field is in focus and thus highlighted



Sign up for our newsletter

Email

user clicks outside the text field – field loses focus and highlighting is removed

Accessing the Document via the DOM

- The DOM (Document Object Model) models a web document as a set of nodes, including element nodes, text nodes, and attribute nodes. Both elements and their text content are separate nodes. Attribute nodes are the attributes of the elements.

- A web document (HTML document) is accessible via the DOM.

- We actually already did this in some previous examples when our JavaScript contained the `document.write()` statements.

- The `document` is the object that you want to access/alter, and using the `write()` method is one way to do that.

Accessing the Document via the DOM

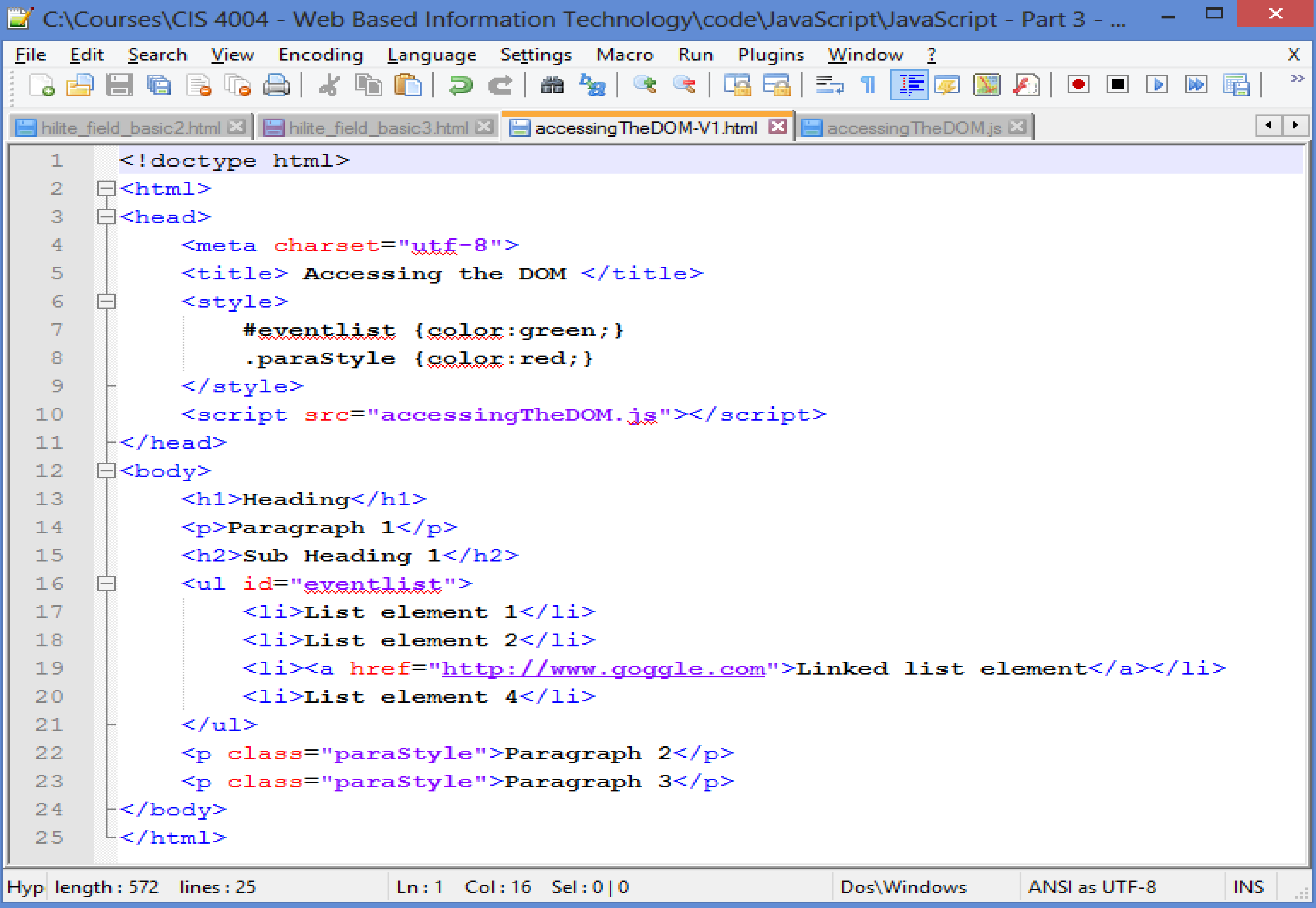
- The `document.write()` statement adds a text string to the document and not a set of nodes and attributes, and you cannot separate the JavaScript out into a separate file – `document.write()` works only where you put it in the HTML.

- What you'd really like is a way to reach where you want to change or add content, and this is exactly what the DOM and its methods provide you.

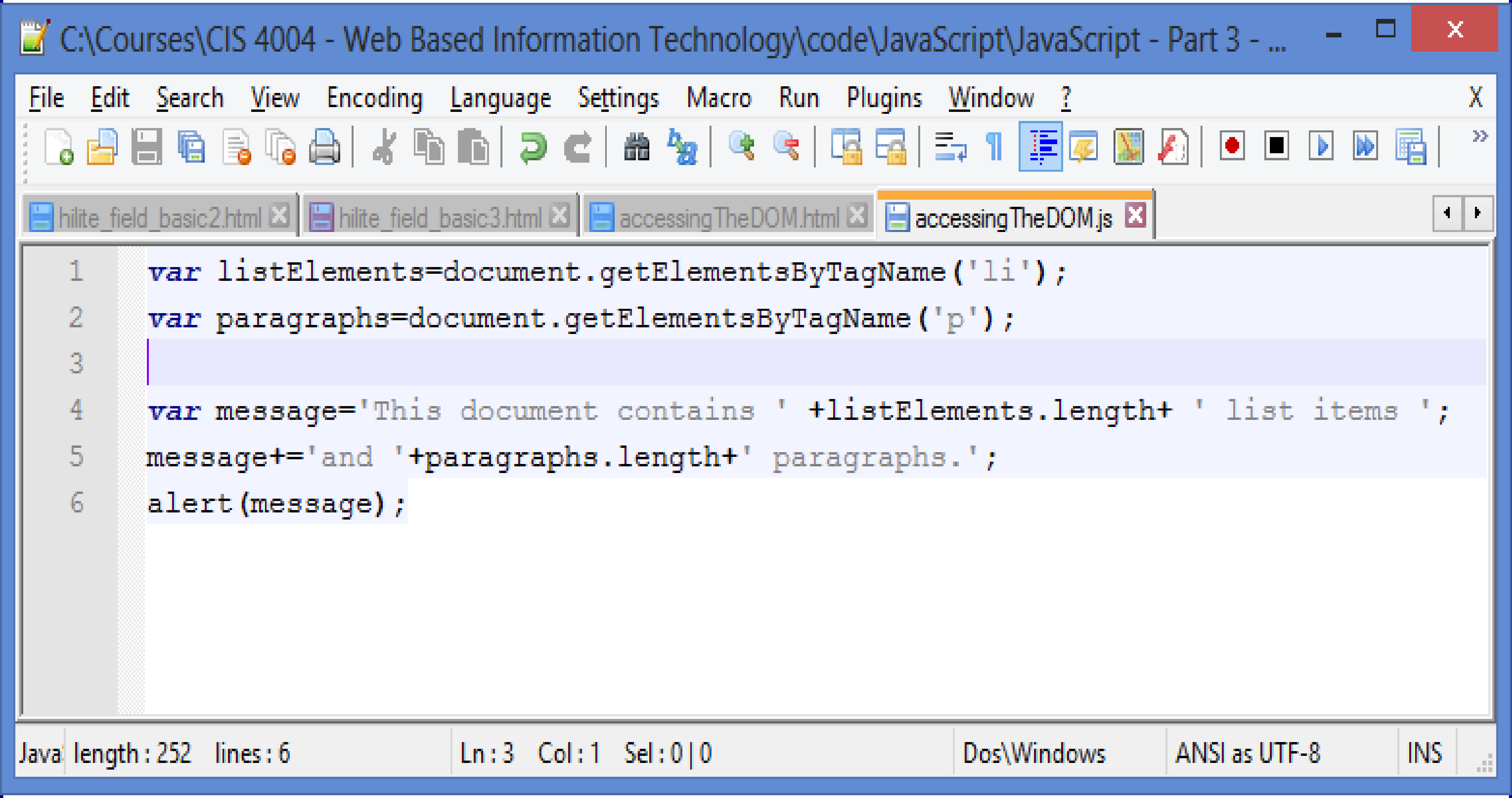
- You can reach elements of the document with three methods:
  - `document.getElementByTagName('p');`
  - `document.getElementByID('id');`
  - `document.getElementsByClassName('cssClass');`

Accessing the Document via the DOM

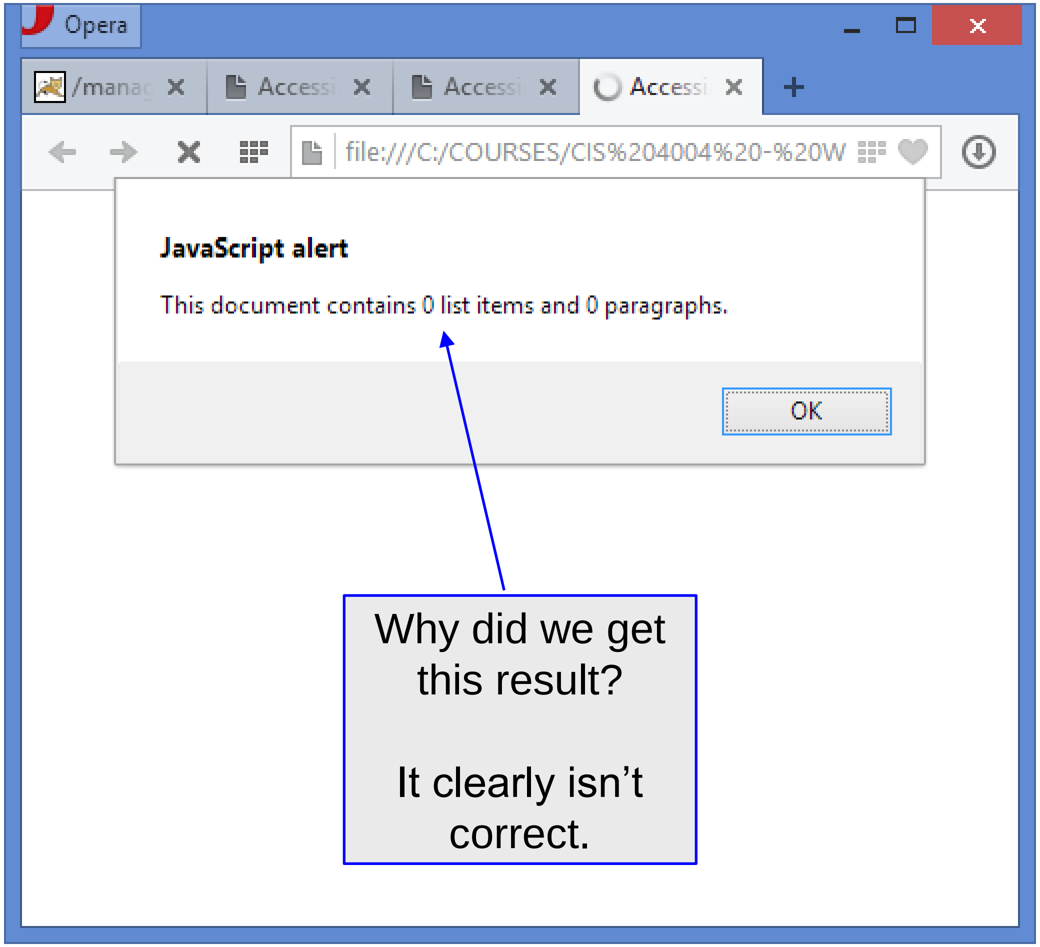
- Let's write a small JavaScript example that utilizes these methods.
- We'll use a small, almost generic, HTML document to illustrate the effect these methods have in accessing the DOM.
- The JavaScript that we'll create will simply count the number of list items and paragraphs in our document.
- The HTML document is shown on page 43 and the JavaScript is shown on page 44.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - ...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hilite_field_basic2.html
hilite_field_basic3.html
accessingTheDOM-V1.html
accessingTheDOM.js
Hyp length : 572 lines : 25
Ln : 1 Col : 16 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title> Accessing the DOM </title>
    <style>
        #eventlist {color:green;}
        .paraStyle {color:red;}
    </style>
    <script src="accessingTheDOM.js"></script>
</head>
<body>
    <h1>Heading</h1>
    <p>Paragraph 1</p>
    <h2>Sub Heading 1</h2>
    <ul id="eventlist">
        <li>List element 1</li>
        <li>List element 2</li>
        <li><a href="http://www.goggle.com">Linked list element</a></li>
        <li>List element 4</li>
    </ul>
    <p class="paraStyle">Paragraph 2</p>
    <p class="paraStyle">Paragraph 3</p>
</body>
</html>
```


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - ...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hilite_field_basic2.html
hilite_field_basic3.html
accessingTheDOM.html
accessingTheDOM.js
Java length : 252 lines : 6
Ln : 3 Col : 1 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```
var listElements=document.getElementsByTagName('li');
var paragraphs=document.getElementsByTagName('p');

var message='This document contains ' +listElements.length+ ' list items ';
message+='and '+paragraphs.length+' paragraphs.';
alert(message);
```



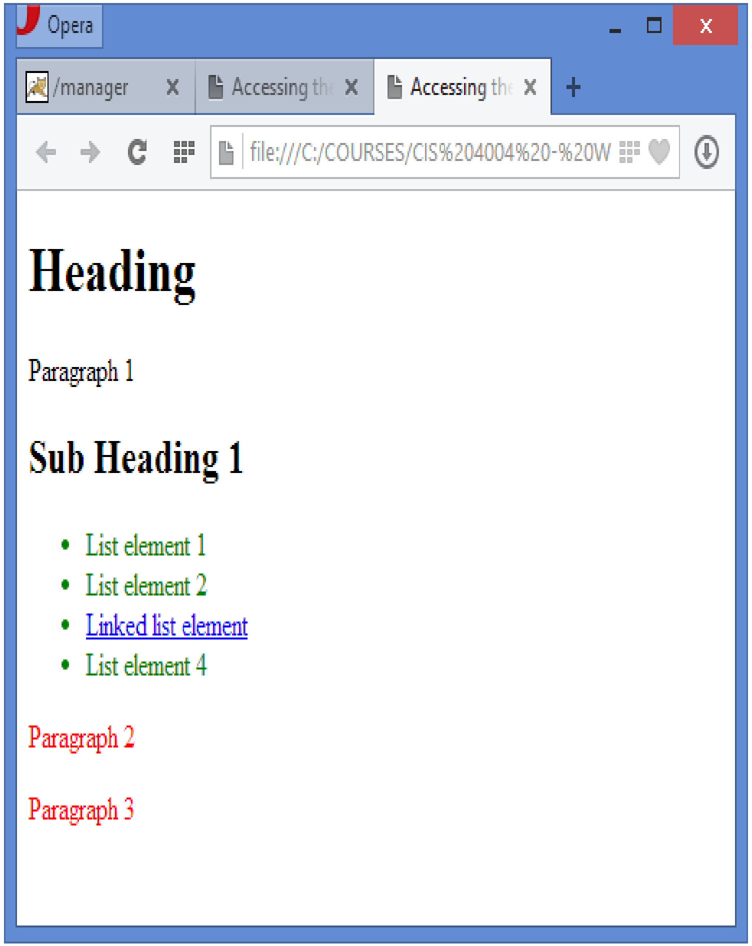
JavaScript alert

This document contains 0 list items and 0 paragraphs.

Why did we get this result?

It clearly isn't correct.

Answer: Because the document wasn't loaded when the JavaScript was executed so there were no list items and no paragraphs…yet.



Heading

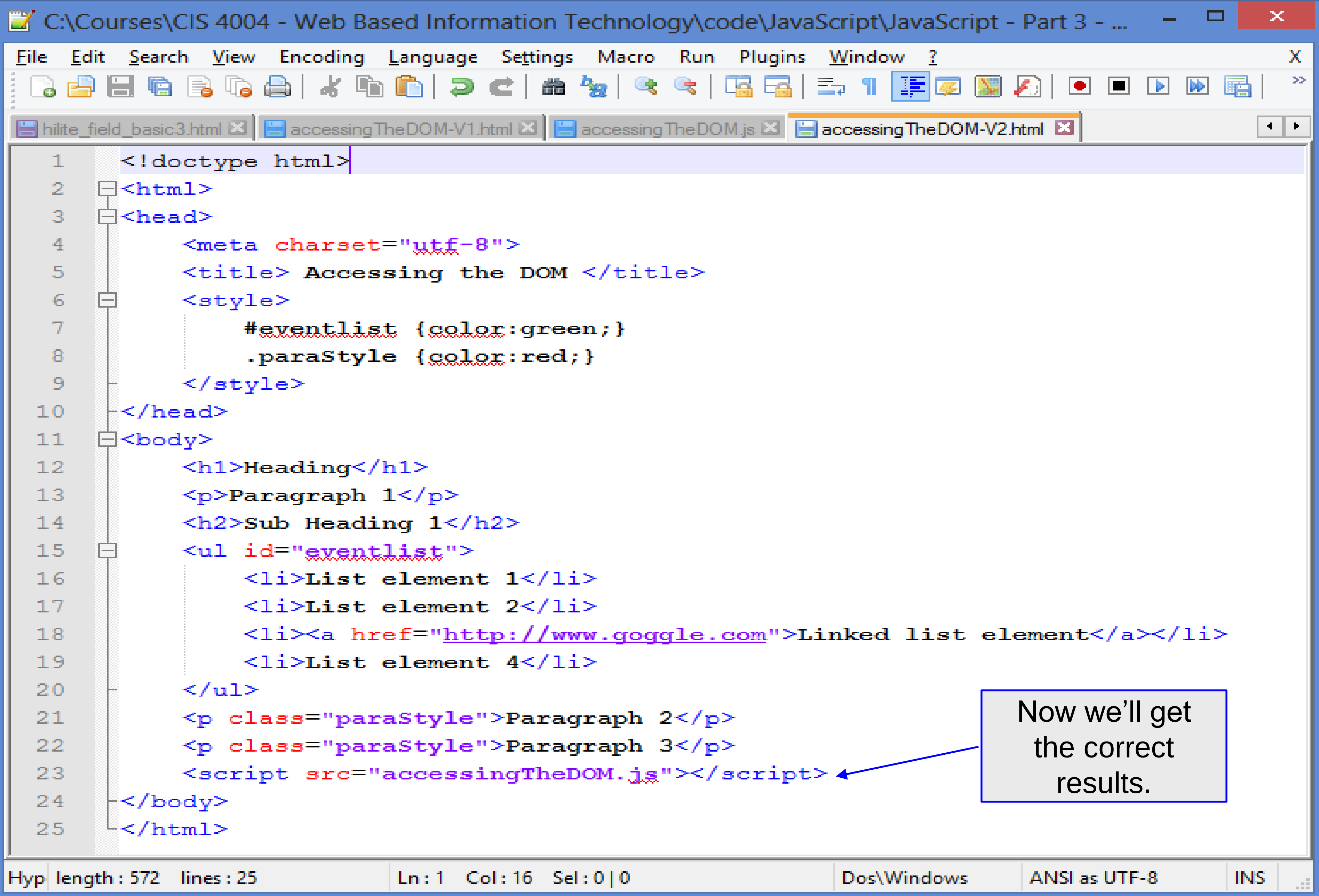
Paragraph 1

Sub Heading 1

- List element 1
- List element 2
- Linked list element
- List element 4

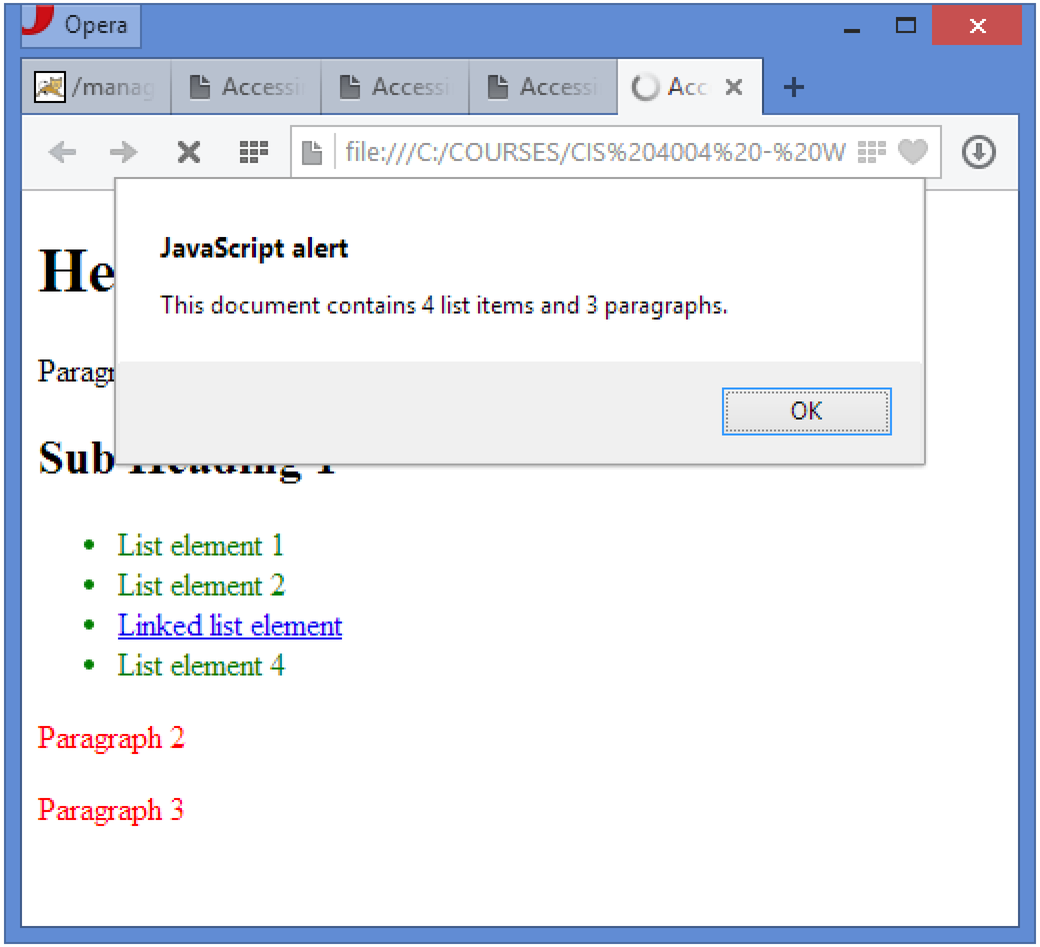
Paragraph 2

Paragraph 3


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - ...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hilite_field_basic3.html
accessingTheDOM-V1.html
accessingTheDOM.js
accessingTheDOM-V2.html
Hyp length : 572 lines : 25
Ln : 1 Col : 16 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title> Accessing the DOM </title>
    <style>
        #eventlist {color:green;}
        .paraStyle {color:red;}
    </style>
</head>
<body>
    <h1>Heading</h1>
    <p>Paragraph 1</p>
    <h2>Sub Heading 1</h2>
    <ul id="eventlist">
        <li>List element 1</li>
        <li>List element 2</li>
        <li><a href="http://www.goggle.com">Linked list element</a></li>
        <li>List element 4</li>
    </ul>
    <p class="paraStyle">Paragraph 2</p>
    <p class="paraStyle">Paragraph 3</p>
    <script src="accessingTheDOM.js"></script>
</body>
</html>
```

Now we'll get the correct results.



JavaScript alert

This document contains 4 list items and 3 paragraphs.

OK

He

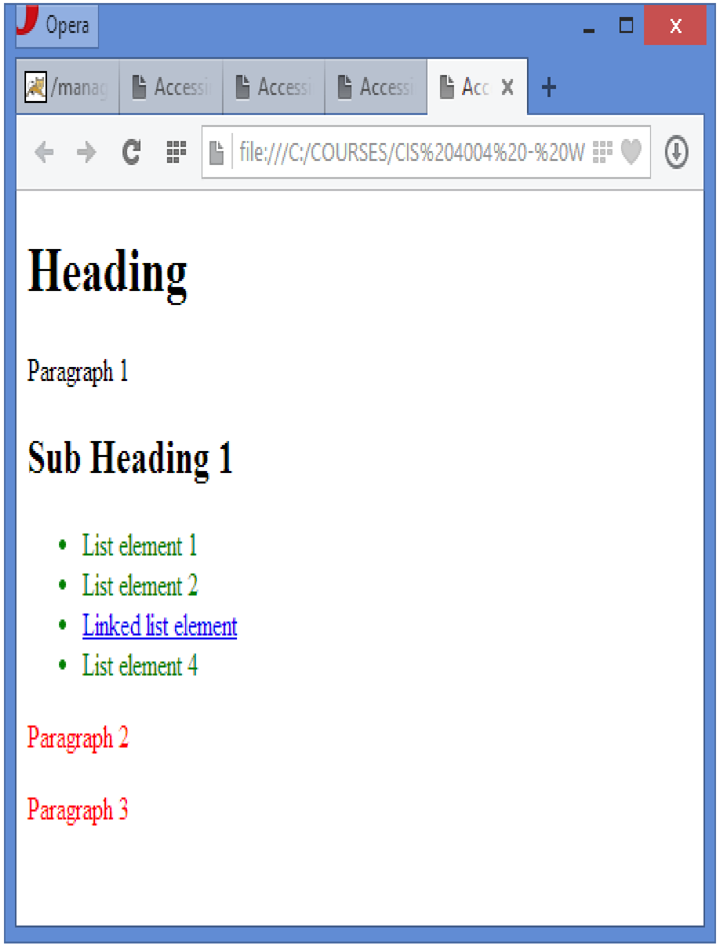
Paragr

Sub Heading 1

- List element 1
- List element 2
- Linked list element
- List element 4

Paragraph 2

Paragraph 3



Heading

Paragraph 1

Sub Heading 1

- List element 1
- List element 2
- Linked list element
- List element 4

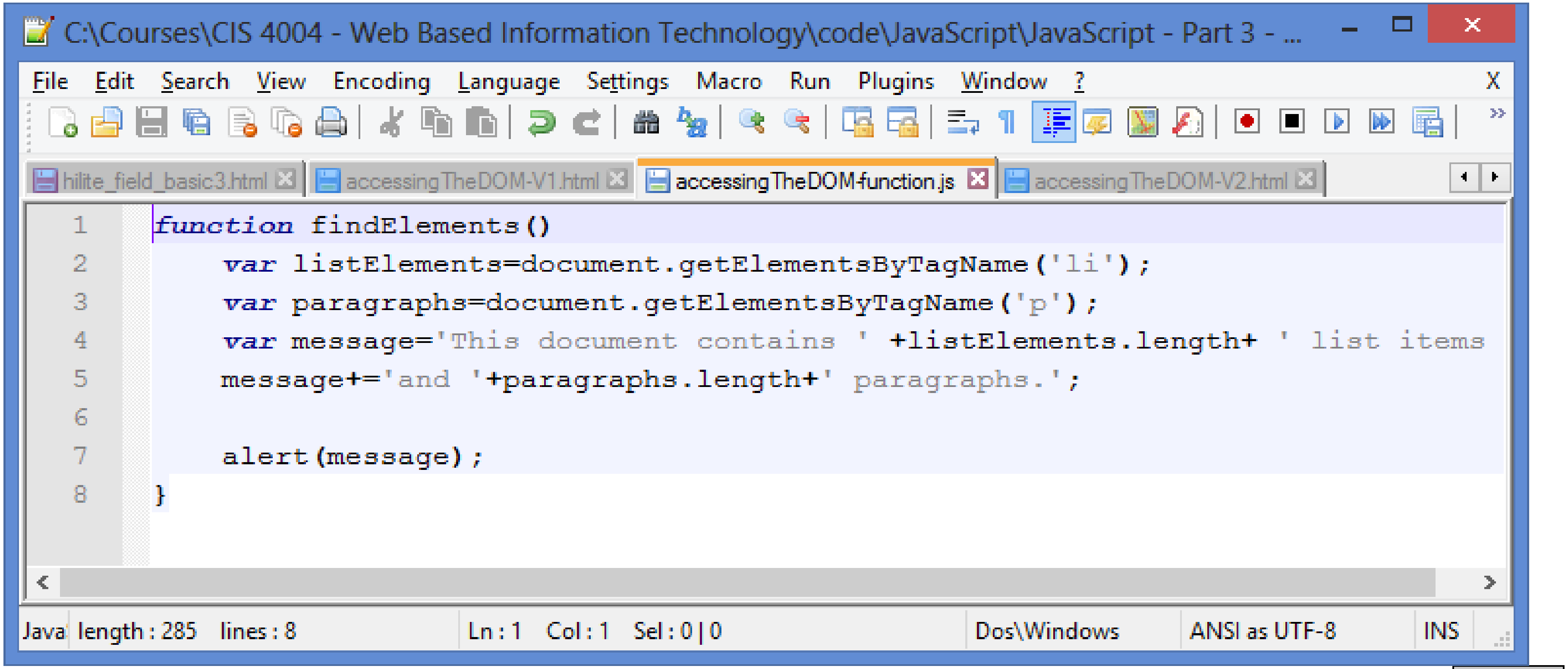
Paragraph 2

Paragraph 3

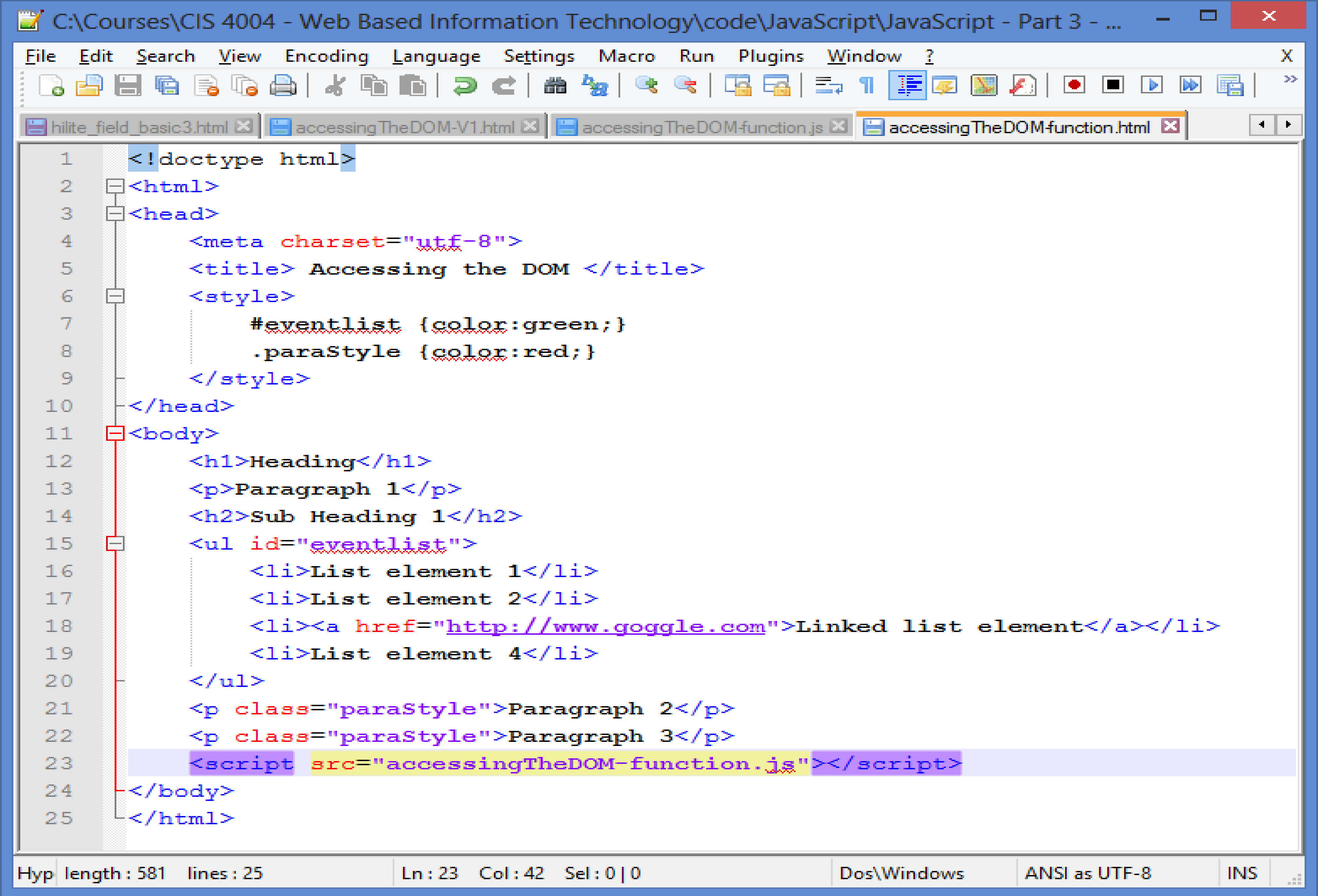
Accessing the Document via the DOM

- Now let’s modify this code a bit.

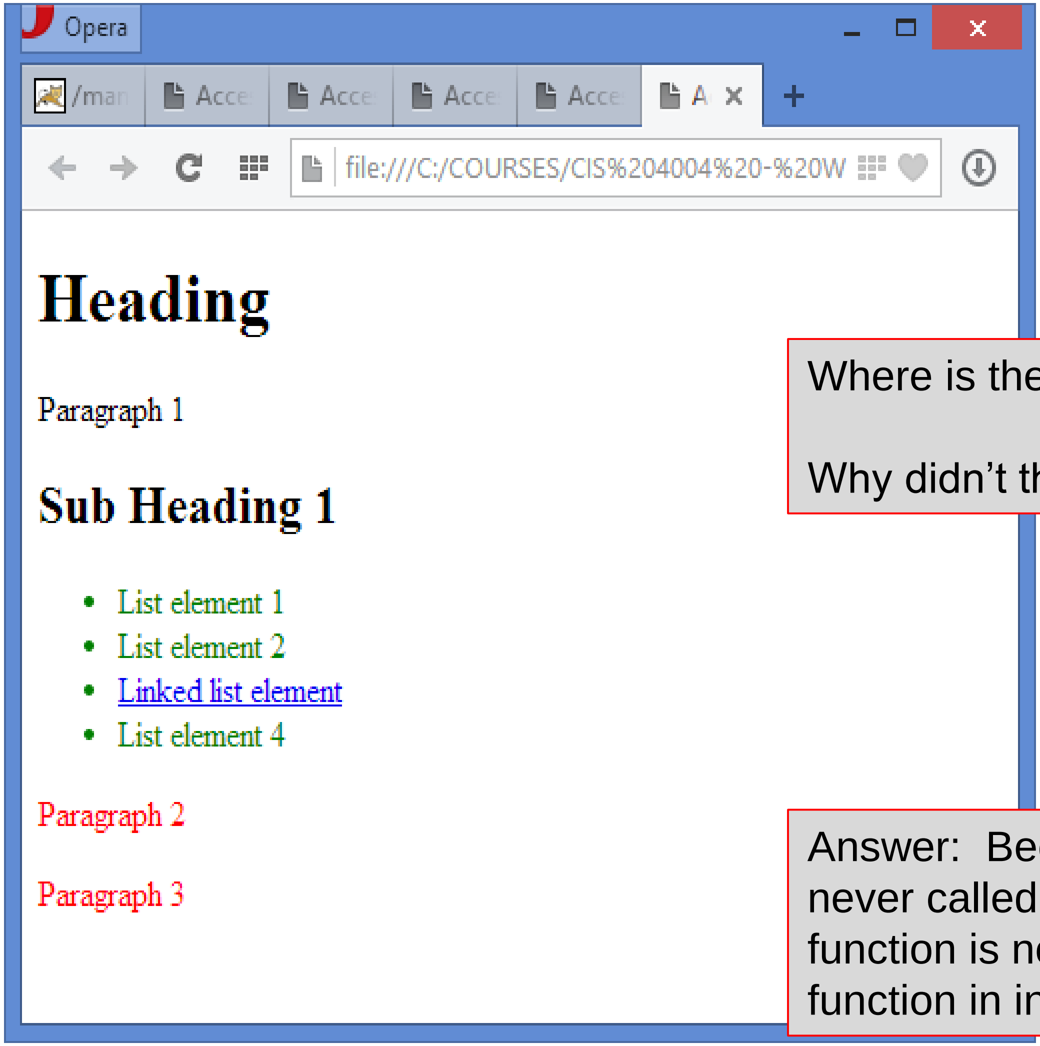
- Let’s re-write the JavaScript as a function rather than just straight-line code.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - ...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hilite_field_basic3.html
accessingTheDOM-V1.html
accessingTheDOM-function.js
accessingTheDOM-V2.html
Java length : 285 lines : 8
Ln : 1 Col : 1 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```
function findElements()
    var listElements=document.getElementsByTagName('li');
    var paragraphs=document.getElementsByTagName('p');
    var message='This document contains ' +listElements.length+ ' list items
    message+='and '+paragraphs.length+' paragraphs.';

    alert(message);
}
```


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - ...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
hilite_field_basic3.html
accessingTheDOM-V1.html
accessingTheDOM-function.js
accessingTheDOM-function.html
Hyp length : 581 lines : 25
Ln : 23 Col : 42 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title> Accessing the DOM </title>
    <style>
        #eventlist {color:green;}
        .paraStyle {color:red;}
    </style>
</head>
<body>
    <h1>Heading</h1>
    <p>Paragraph 1</p>
    <h2>Sub Heading 1</h2>
    <ul id="eventlist">
        <li>List element 1</li>
        <li>List element 2</li>
        <li><a href="http://www.goggle.com">Linked list element</a></li>
        <li>List element 4</li>
    </ul>
    <p class="paraStyle">Paragraph 2</p>
    <p class="paraStyle">Paragraph 3</p>
    <script src="accessingTheDOM-function.js"></script>
</body>
</html>
```



Heading

Paragraph 1

Sub Heading 1

- List element 1
- List element 2
- Linked list element
- List element 4

Paragraph 2

Paragraph 3

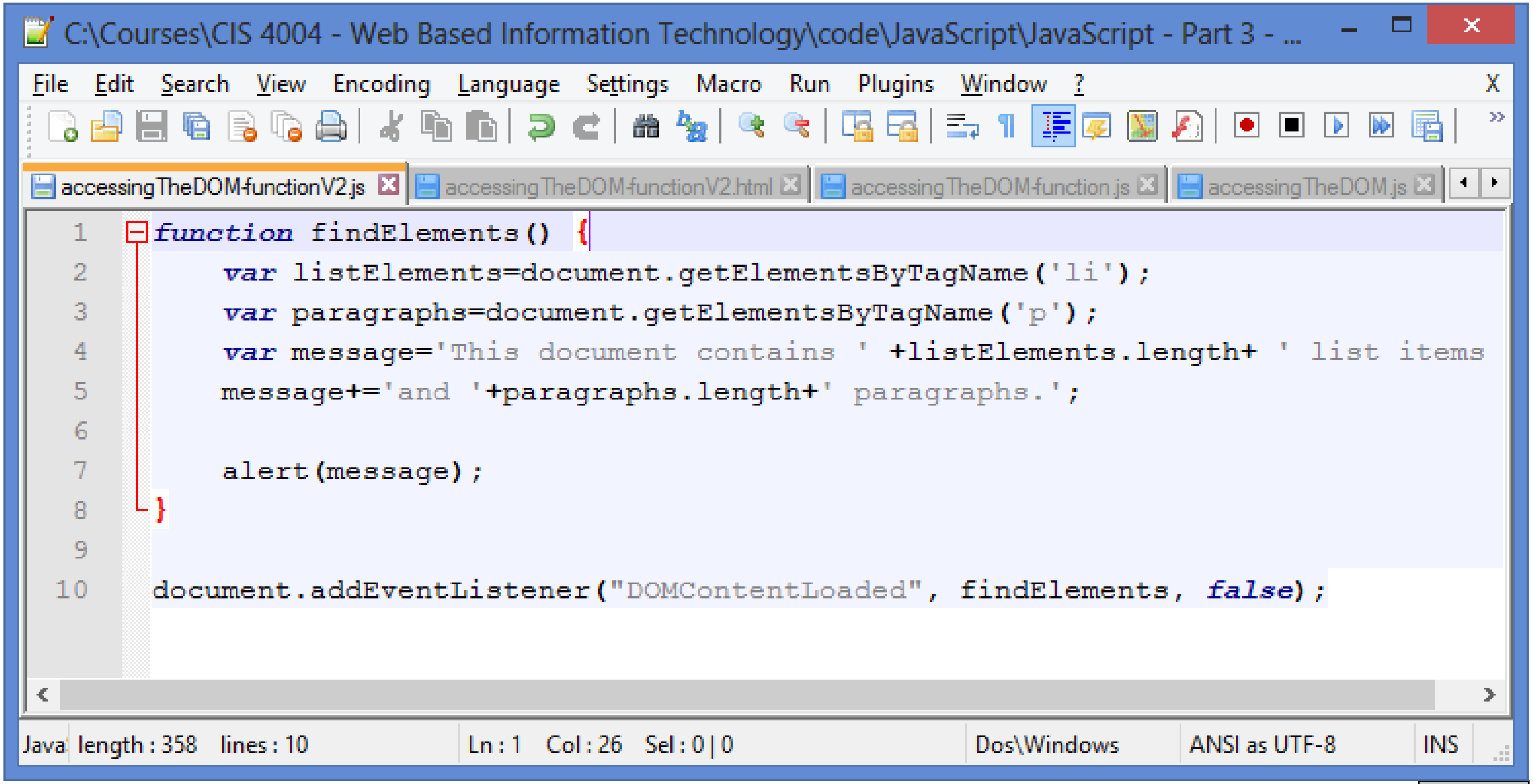
Where is the alert?

Why didn’t the function execute?

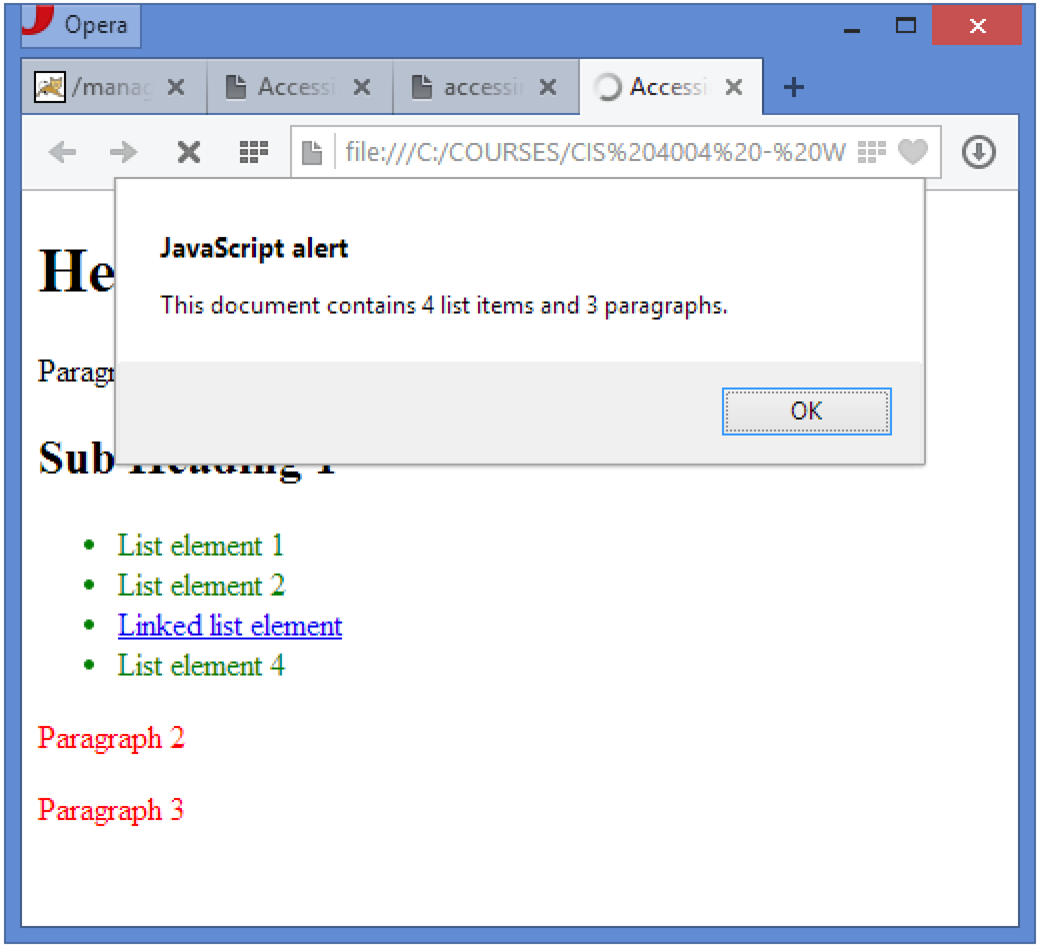
Answer: Because the function was never called. JavaScript inside a function is not executed unless the function in invoked (called).

Accessing the Document via the DOM

- We need to modify the JavaScript so that we call our function. This is shown below:


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - ...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
accessingTheDOMfunctionV2.js
accessingTheDOMfunctionV2.html
accessingTheDOMfunction.js
accessingTheDOM.js
Java length : 358 lines : 10
Ln : 1 Col : 26 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```
function findElements() {
    var listElements=document.getElementsByTagName('li');
    var paragraphs=document.getElementsByTagName('p');
    var message='This document contains ' +listElements.length+ ' list items
    message+='and '+paragraphs.length+' paragraphs.';

    alert(message);
}

document.addEventListener("DOMContentLoaded", findElements, false);
```



JavaScript alert

This document contains 4 list items and 3 paragraphs.

He

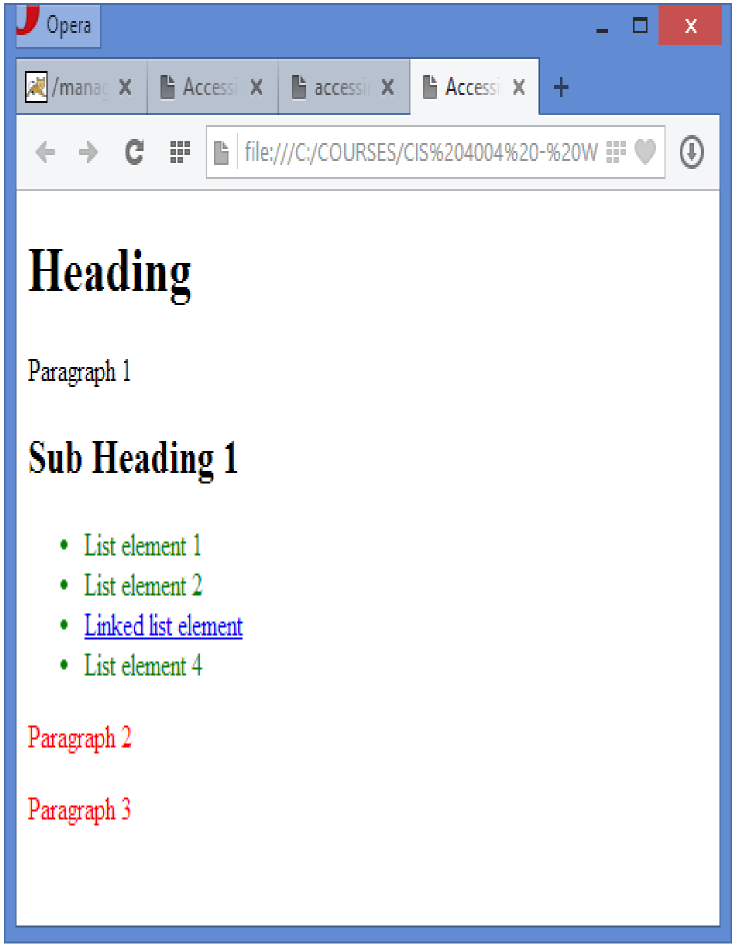
Parag

Sub

- List element 1
- List element 2
- Linked list element
- List element 4

Paragraph 2

Paragraph 3



Heading

Paragraph 1

Sub Heading 1

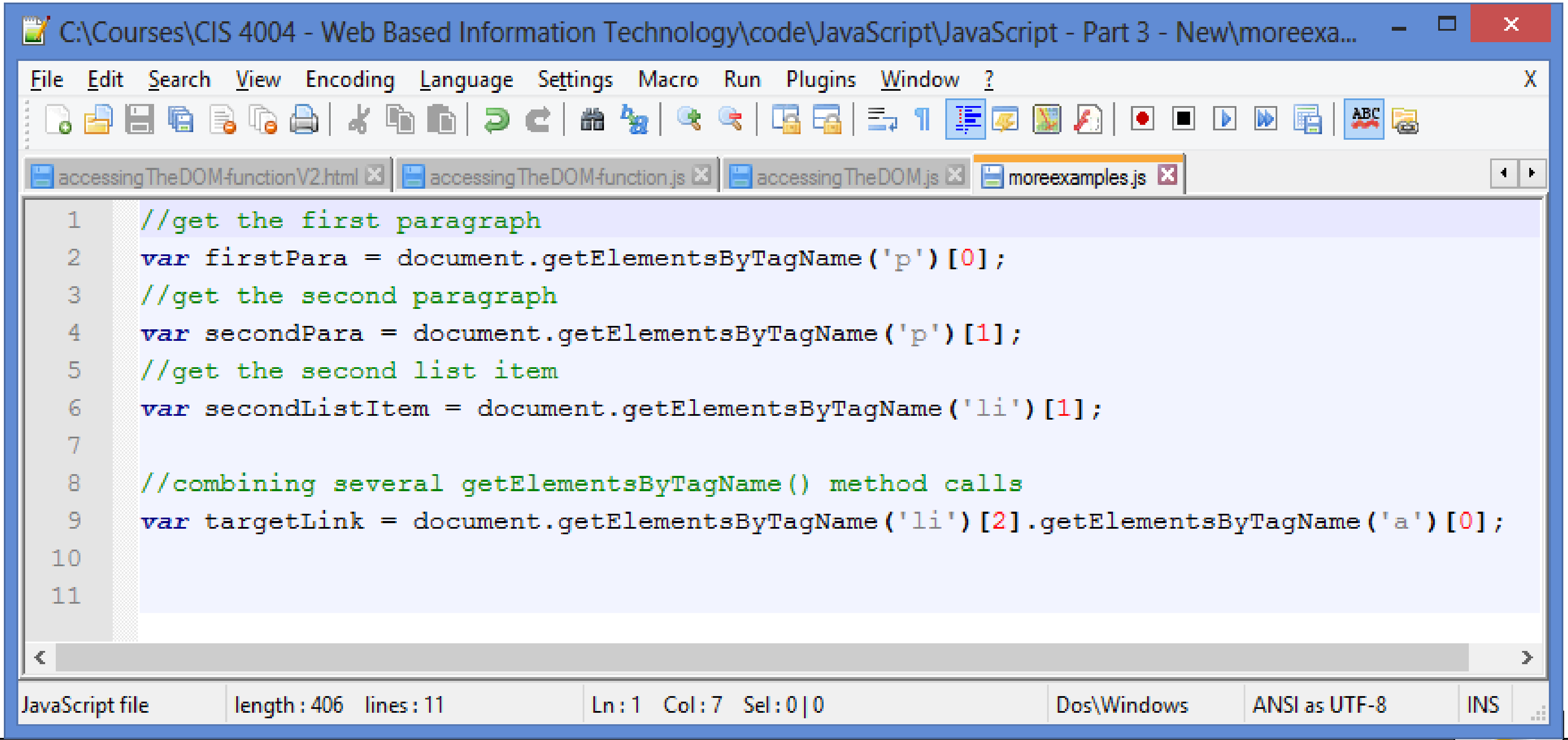
- List element 1
- List element 2
- Linked list element
- List element 4

Paragraph 2

Paragraph 3

Accessing the Document via the DOM

- You can also access each of the elements of a certain name just like you would access an array. Keep in mind though that JavaScript array counters begin a 0 and not at 1.

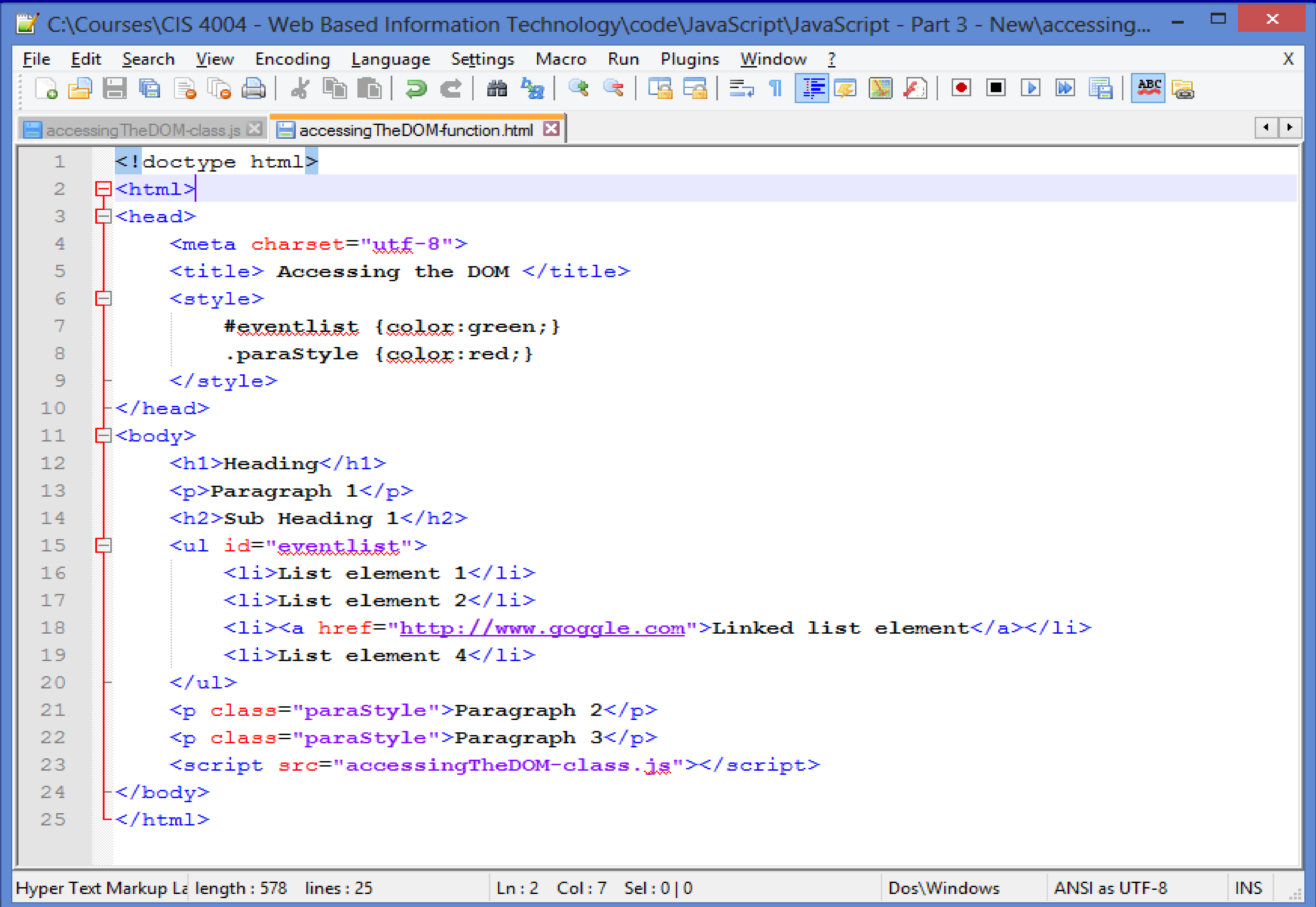


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\moreexa...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
accessingTheDOM-functionV2.html
accessingTheDOM-function.js
accessingTheDOM.js
moreexamples.js

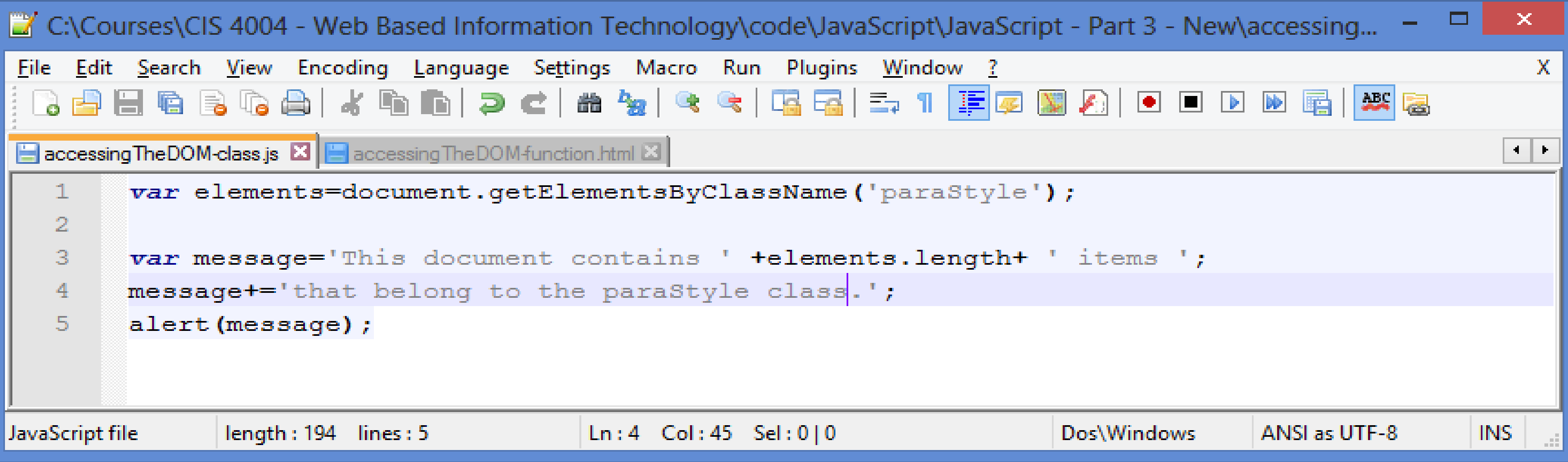
```
//get the first paragraph
var firstPara = document.getElementsByTagName('p')[0];
//get the second paragraph
var secondPara = document.getElementsByTagName('p')[1];
//get the second list item
var secondListItem = document.getElementsByTagName('li')[1];

//combining several getElementsByTagName() method calls
var targetLink = document.getElementsByTagName('li')[2].getElementsByTagName('a')[0];
```

Accessing the Document via the DOM

- The `getElementsByClassName()` and `getElementById()`, work in much the same way as we've just illustrated with `getElementsByTagName()`.
- The following example, illustrates the `getElementsByClassName()` method.

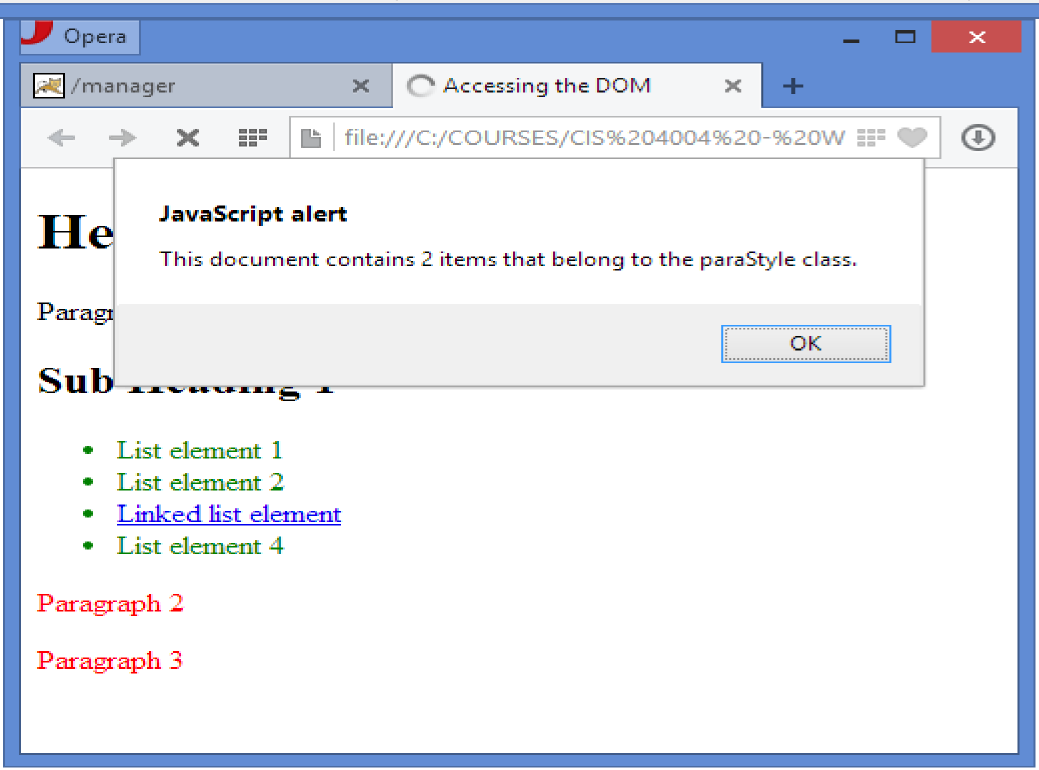

C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\accessing...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
accessingTheDOM-class.js
accessingTheDOM-function.html
Hyper Text Markup La length : 578 lines : 25
Ln : 2 Col : 7 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title> Accessing the DOM </title>
    <style>
        #eventlist {color:green;}
        .paraStyle {color:red;}
    </style>
</head>
<body>
    <h1>Heading</h1>
    <p>Paragraph 1</p>
    <h2>Sub Heading 1</h2>
    <ul id="eventlist">
        <li>List element 1</li>
        <li>List element 2</li>
        <li><a href="http://www.goggle.com">Linked list element</a></li>
        <li>List element 4</li>
    </ul>
    <p class="paraStyle">Paragraph 2</p>
    <p class="paraStyle">Paragraph 3</p>
    <script src="accessingTheDOM-class.js"></script>
</body>
</html>
```


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\accessing...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
accessingTheDOM-class.js
accessingTheDOM-function.html

```
var elements=document.getElementsByClassName('paraStyle');

var message='This document contains ' +elements.length+ ' items ';
message+='that belong to the paraStyle class.';
alert(message);
```

JavaScript file
length : 194 lines : 5
Ln : 4 Col : 45 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS



He

Paragr

Sub

JavaScript alert

This document contains 2 items that belong to the paraStyle class.

- List element 1
- List element 2
- Linked list element
- List element 4

Paragraph 2

Paragraph 3

Accessing the Document via the DOM

- Examine the HTML document that we’ve been using for our recent examples.

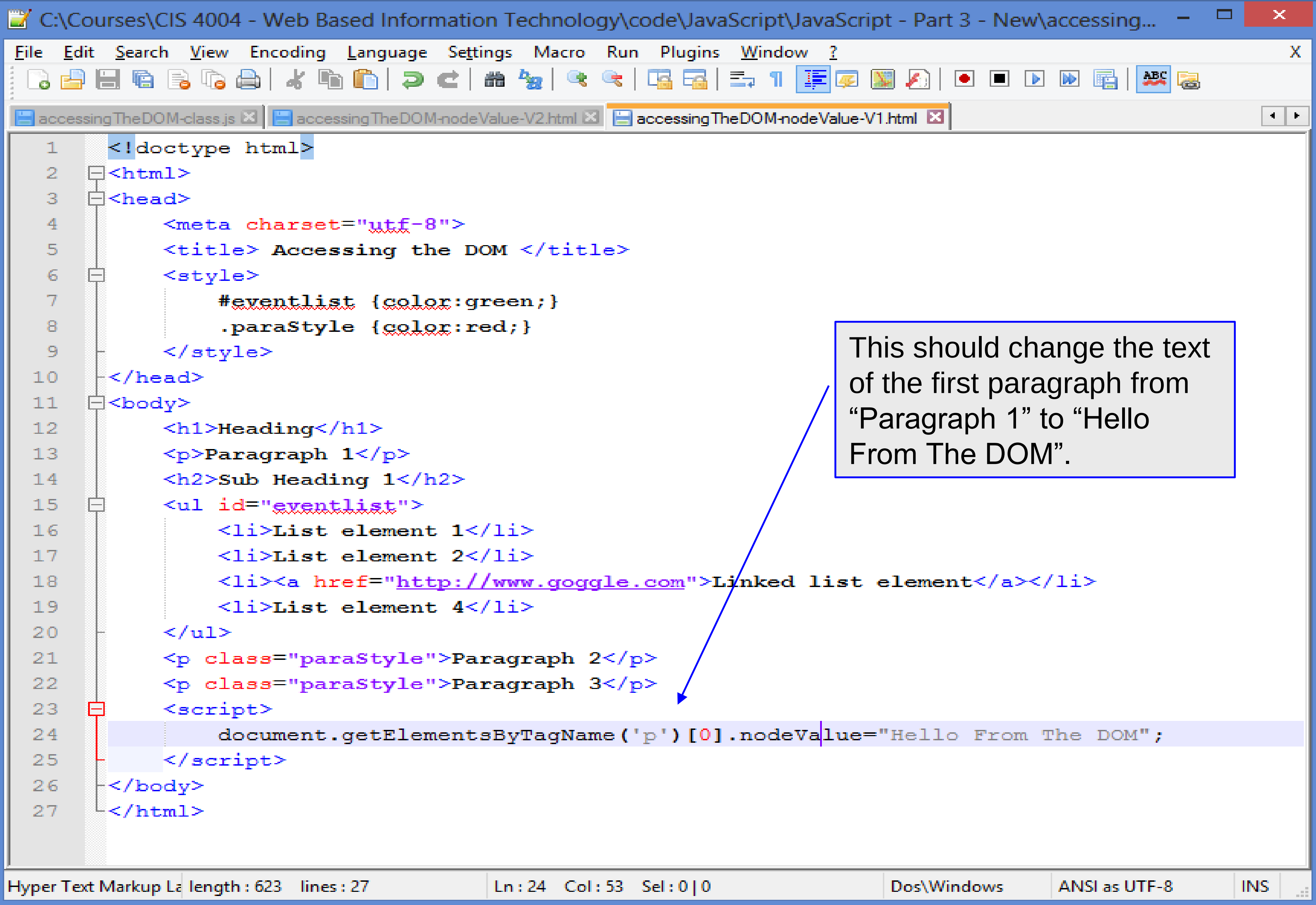
- All of the `<p>`, `<h1>`, `<h2>`, `and` `<ul>` elements are children of the `<body>` element and they are all siblings.

- The `<li>` elements are all children of the `<ul>` element and are siblings to one another. The `<a>` element is a child of the third `<li>` element.

- However, there are even more children. The text inside the `<p>`, `<h1>`, `<h2>`, `<li>`, `and` `<a>` elements also consist of nodes in the DOM, and while they are not elements, they still follow the same relationship rules.

Accessing the Document via the DOM

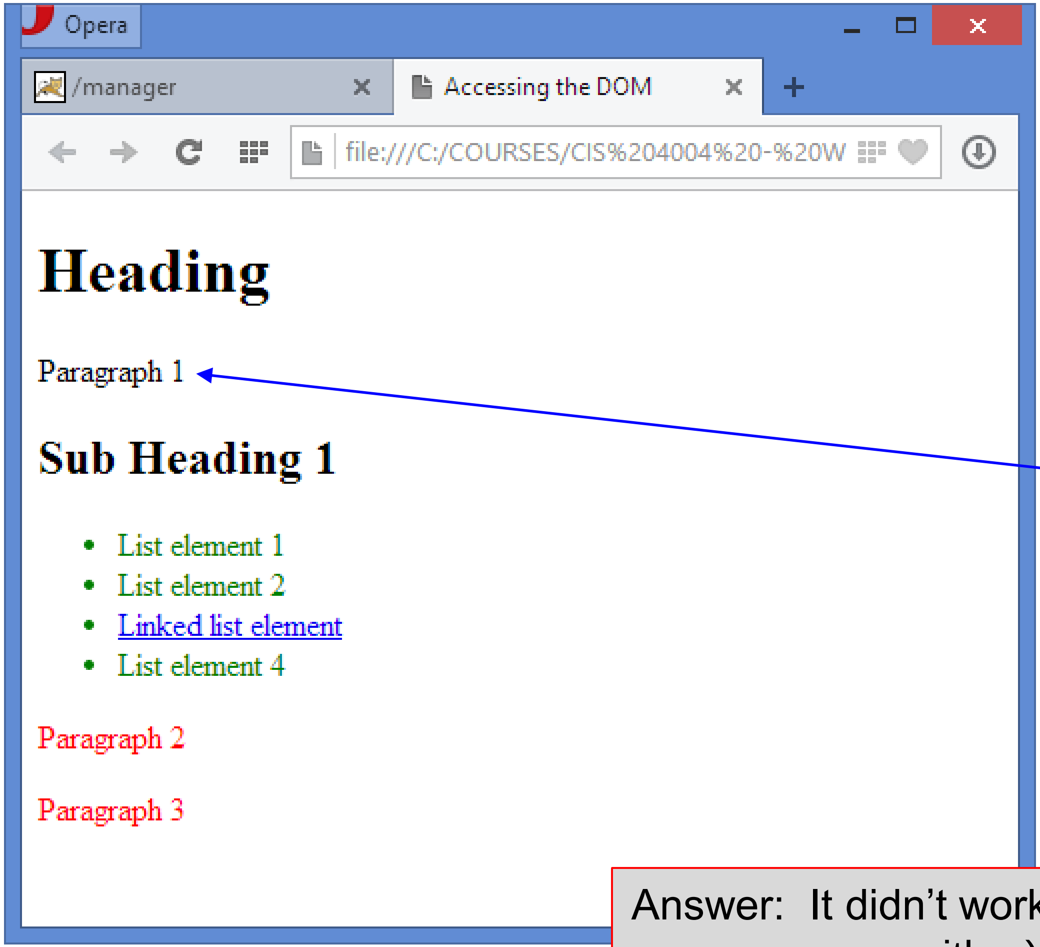
- Every node in the document has several valuable properties:
  - The most important property is `nodeType`, which describes what the node is – an `element`, an `attribute`, a `comment`, `text`, or one of several more types (there are 12 in all – we'll see them all later). Mostly, though the only valuable types are `nodeType` 1 and `nodeType` 3, where 1 is a element node and 3 is a text node.
  - Another important property is `nodeName`, which is the name of the element or #text if it is a text node. Depending on the type of document and the user agent (browser), `nodeName` can be either uppercase or lowercase, which is why it is a good idea to convert it to lowercase before testing for a certain name. Use the `toLowerCase()` method of the `string` object for that, such as: `if(obj.nodeName.toLowerCase()=='li'){};`. For element nodes, you can use the `tagName` property.
  - `nodeValue` is the value of the node; `null` if it is an element, and the text content if it is a text node. In the case of text nodes, `nodeValue` can be read and set, which allows you to alter the text content of the element.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\accessing...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
accessingTheDOM-class.js
accessingTheDOM-nodeValue-V2.html
accessingTheDOM-nodeValue-V1.html

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title> Accessing the DOM </title>
    <style>
        #eventlist {color:green;}
        .paraStyle {color:red;}
    </style>
</head>
<body>
    <h1>Heading</h1>
    <p>Paragraph 1</p>
    <h2>Sub Heading 1</h2>
    <ul id="eventlist">
        <li>List element 1</li>
        <li>List element 2</li>
        <li><a href="http://www.goggle.com">Linked list element</a></li>
        <li>List element 4</li>
    </ul>
    <p class="paraStyle">Paragraph 2</p>
    <p class="paraStyle">Paragraph 3</p>
    <script>
        document.getElementsByTagName('p')[0].nodeValue="Hello From The DOM";
    </script>
</body>
</html>
```

Hyper Text Markup La length : 623 lines : 27
Ln : 24 Col : 53 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

This should change the text of the first paragraph from “Paragraph 1” to “Hello From The DOM”.



Heading

Paragraph 1

Sub Heading 1

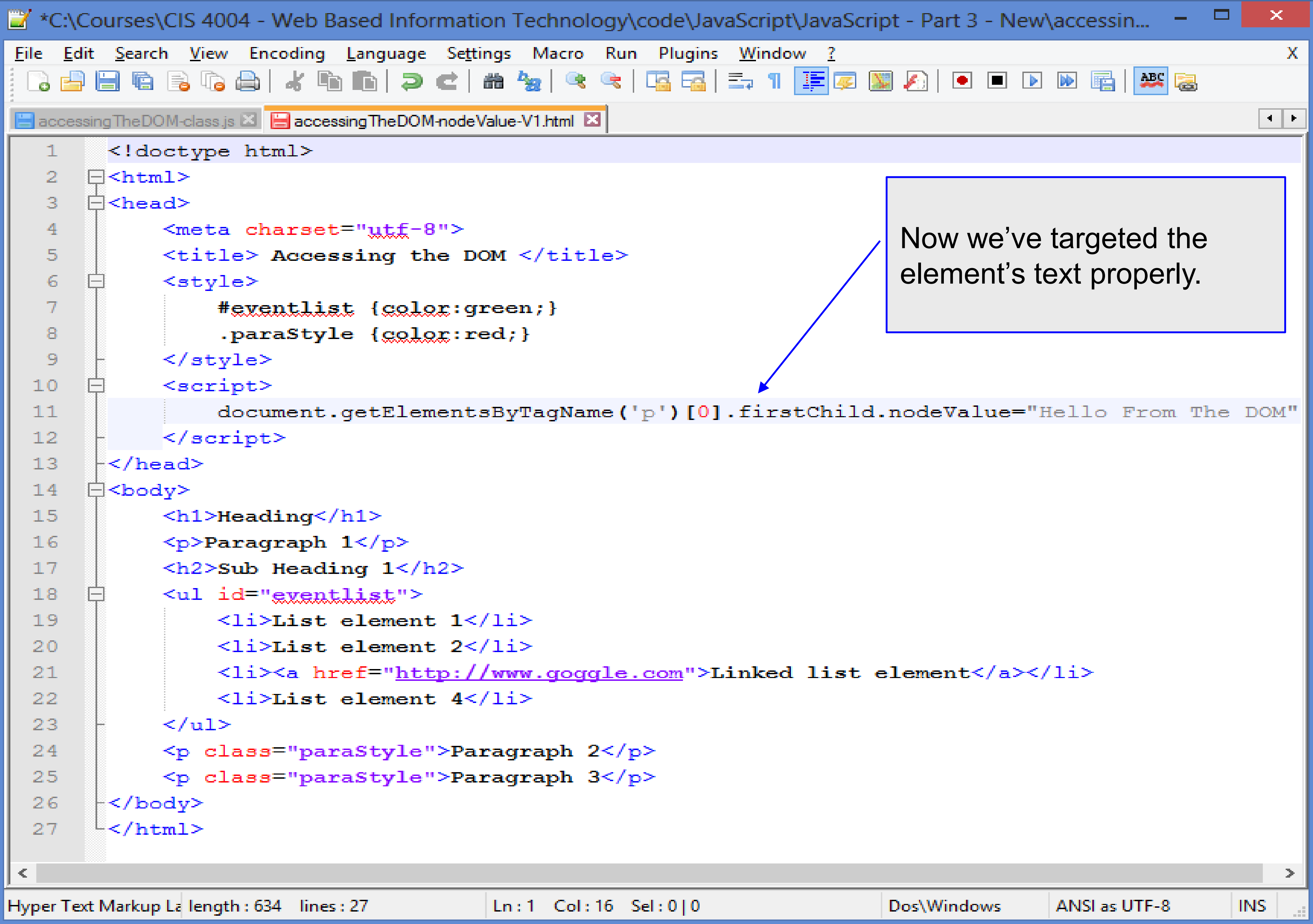
- List element 1
- List element 2
- Linked list element
- List element 4

Paragraph 2

Paragraph 3

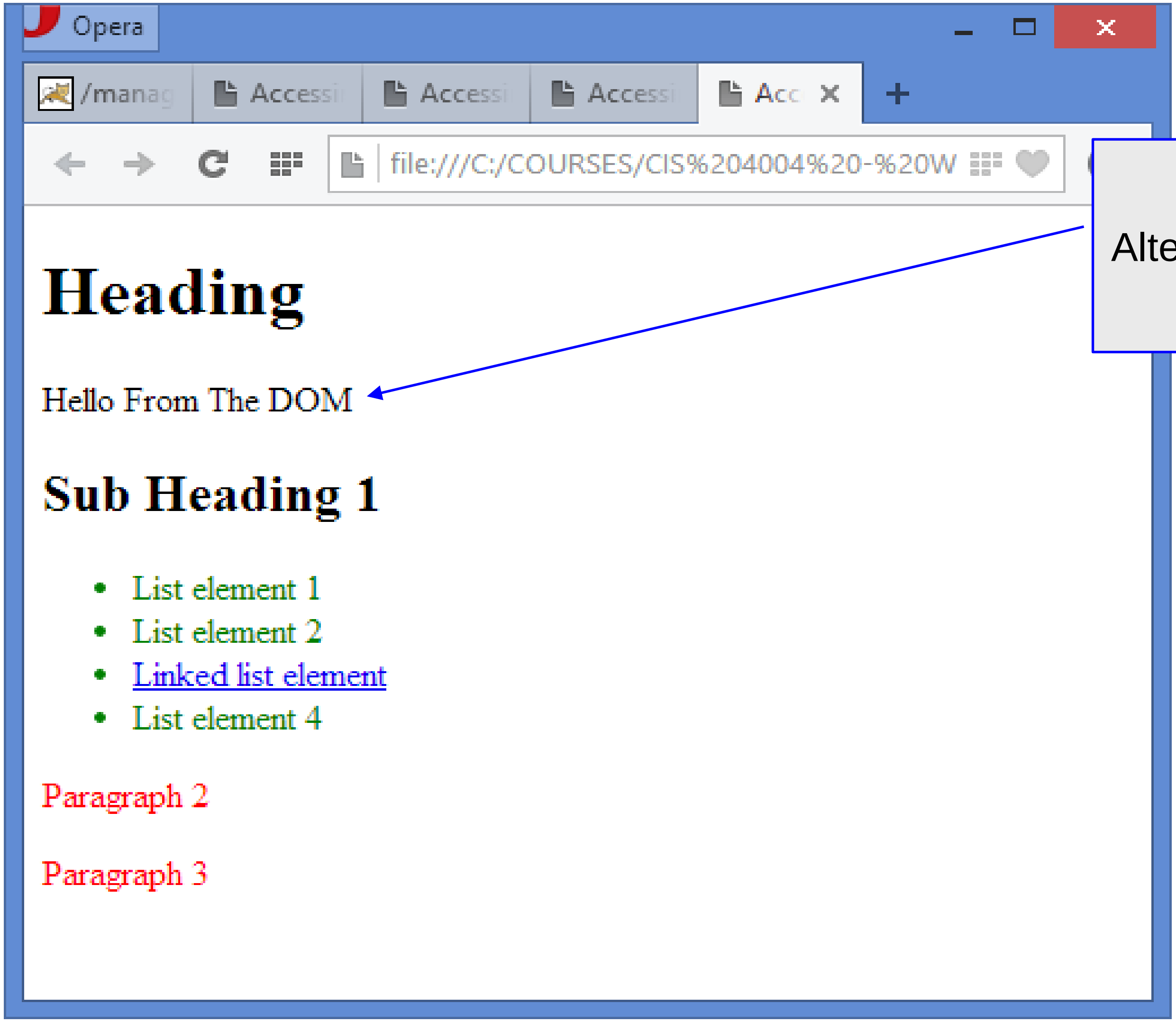
It didn’t work! Why not?

Answer: It didn’t work (and strangely enough – it didn’t cause an error either), because the first paragraph is an element node. If you want to change the text inside that paragraph, you need to access the text node inside it, or in other words, the first child node of the paragraph.
See next page.


*C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\accessin...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
accessingTheDOM-class.js
accessingTheDOM-nodeValue-V1.html
Hyper Text Markup La length : 634 lines : 27
Ln : 1 Col : 16 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title> Accessing the DOM </title>
    <style>
        #eventlist {color:green;}
        .paraStyle {color:red;}
    </style>
    <script>
        document.getElementsByTagName('p')[0].firstChild.nodeValue="Hello From The DOM"
    </script>
</head>
<body>
    <h1>Heading</h1>
    <p>Paragraph 1</p>
    <h2>Sub Heading 1</h2>
    <ul id="eventlist">
        <li>List element 1</li>
        <li>List element 2</li>
        <li><a href="http://www.goggle.com">Linked list element</a></li>
        <li>List element 4</li>
    </ul>
    <p class="paraStyle">Paragraph 2</p>
    <p class="paraStyle">Paragraph 3</p>
</body>
</html>
```

Now we’ve targeted the element’s text properly.



Heading

Hello From The DOM

Sub Heading 1

- List element 1
- List element 2
- Linked list element
- List element 4

Paragraph 2

Paragraph 3

Altered text.

Accessing the Document via the DOM

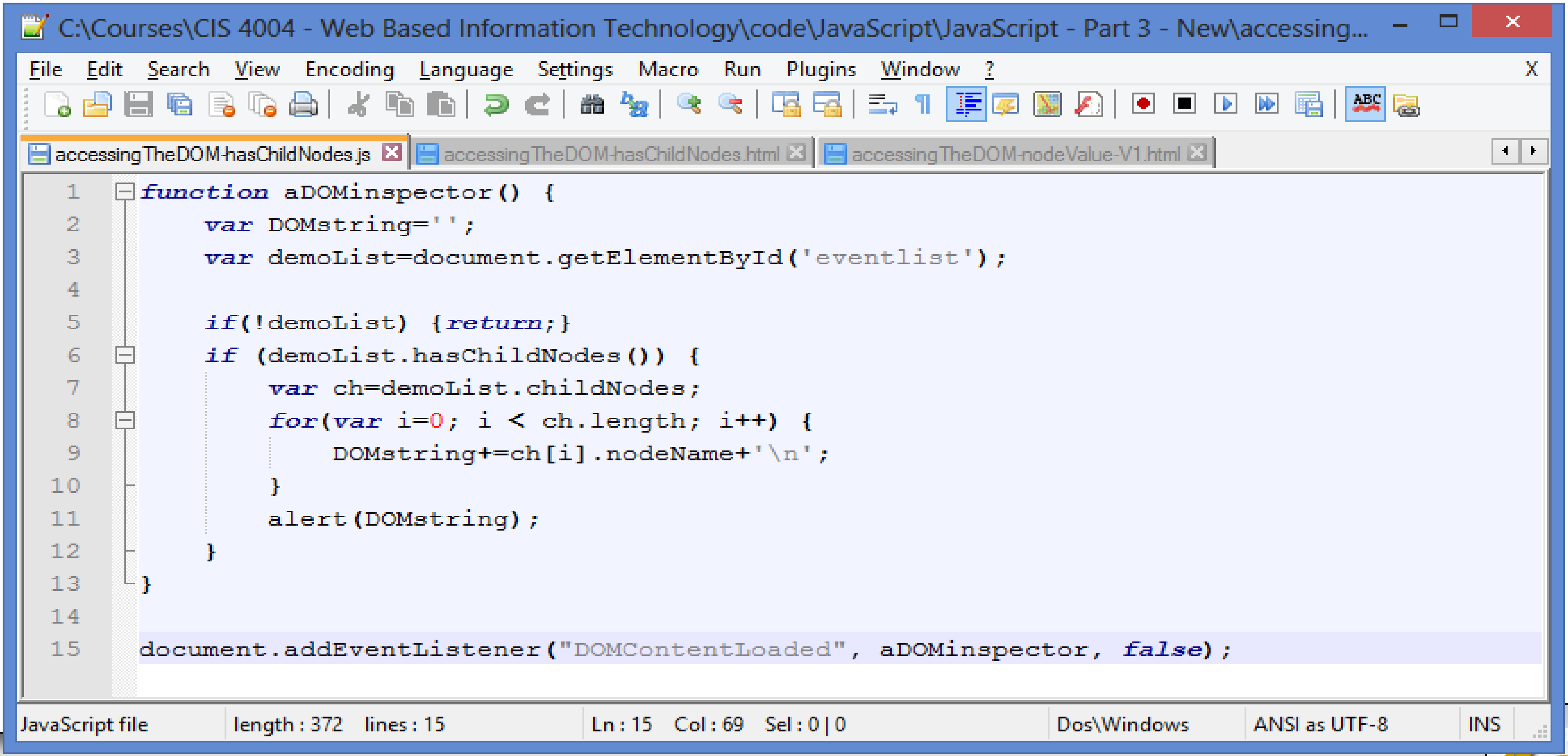
- The `firstChild` property that we utilized in the previous example is a shortcut. Every element can have any number of children, listed in a property called `childNodes`.

- Here are a few important aspects about `childNodes`:
  - `childNodes` is a list of all the first-level children of the element. It does not cascade down into deeper levels.
  - You can access a child element of the current element via the array counter or the `item()` method.
  - The shortcut properties `firstChild` and `lastChild` are easier to use versions of `element.childNodes[0]` and `element.childNodes[element.childNodes.length-1]`.
  - You can check if an element has any children by calling the method `hasChildNodes()`, which returns a Boolean value.

Accessing the Document via the DOM

- Let’s modify our running example to access the `<ul>` element and obtain information about its children.

- The JavaScript shown below, utilizes the `childNodes` property and the `hasChildNodes()` method.



C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\accessing...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
accessingTheDOM-hasChildNodes.js
accessingTheDOM-hasChildNodes.html
accessingTheDOM-nodeValue-V1.html

```
function aDOMinspector() {
    var DOMstring='';
    var demoList=document.getElementById('eventlist');

    if(!demoList) {return;}
    if (demoList.hasChildNodes()) {
        var ch=demoList.childNodes;
        for(var i=0; i < ch.length; i++) {
            DOMstring+=ch[i].nodeName+'\n';
        }
        alert(DOMstring);
    }
}

document.addEventListener("DOMContentLoaded", aDOMinspector, false);
```

JavaScript file
length : 372 lines : 15
Ln : 15 Col : 69 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

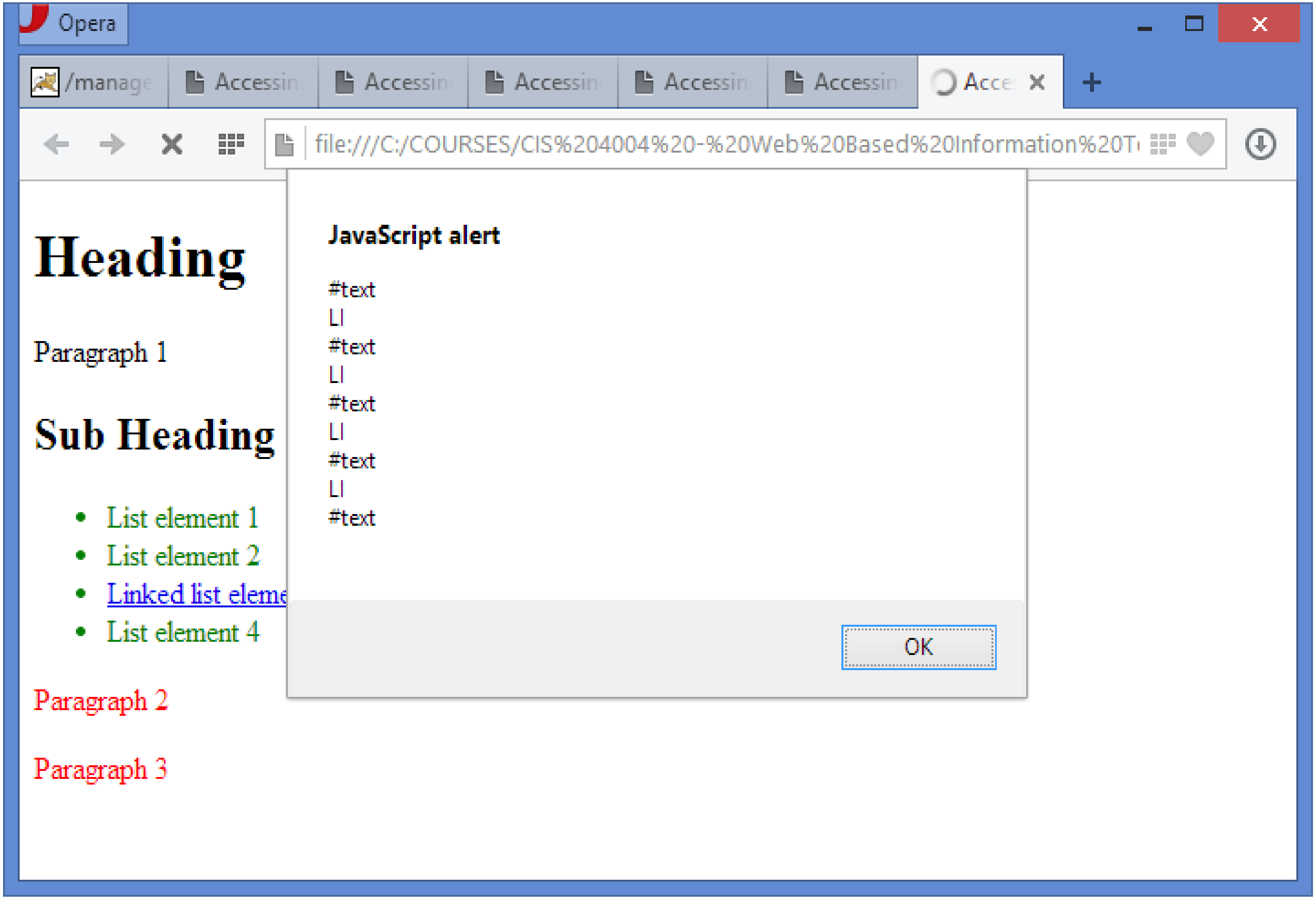
Accessing the Document via the DOM

- We created an empty string called DOM string and then check for DOM support and whether the UL element with the right id attribute is defined.

- Then we test whether the element has child notes and, if it does, store them in a variable named ch.

- Next we loop through the variable, which automatically becomes an array, and add the nodeName of each child to the DOMstring, followed by a line break. Let's modify our running example to access the `<ul>` element and obtain information about its children.

- The JavaScript shown below, utilizes the `childNodes` property and the `hasChildNodes()` method.


Opera
/manage
Accessin
Accessin
Accessin
Accessin
Accessin
Acce
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20T

Heading

Paragraph 1

Sub Heading

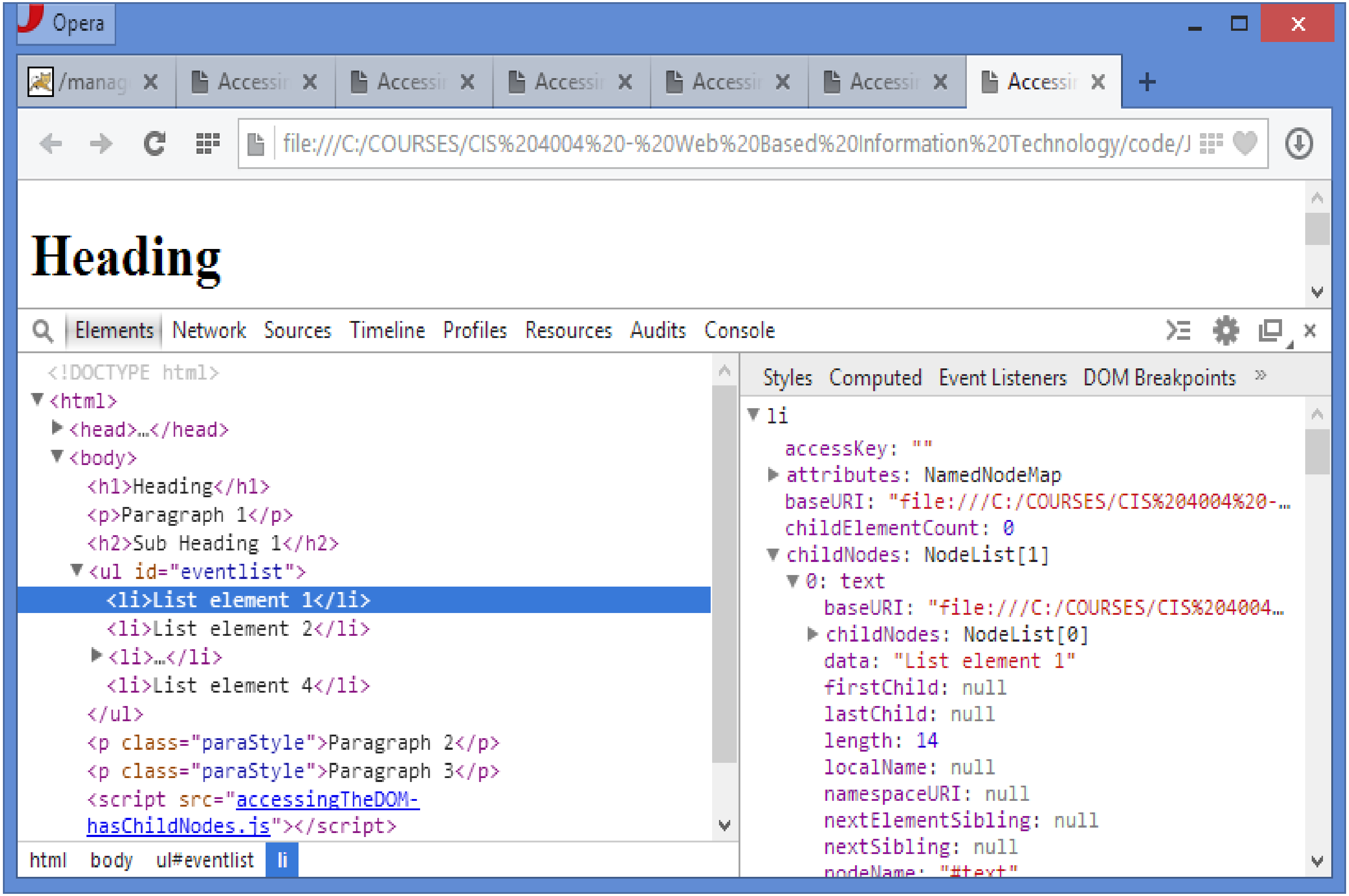
- List element 1
- List element 2
- Linked list eleme
- List element 4

Paragraph 2

Paragraph 3

JavaScript alert

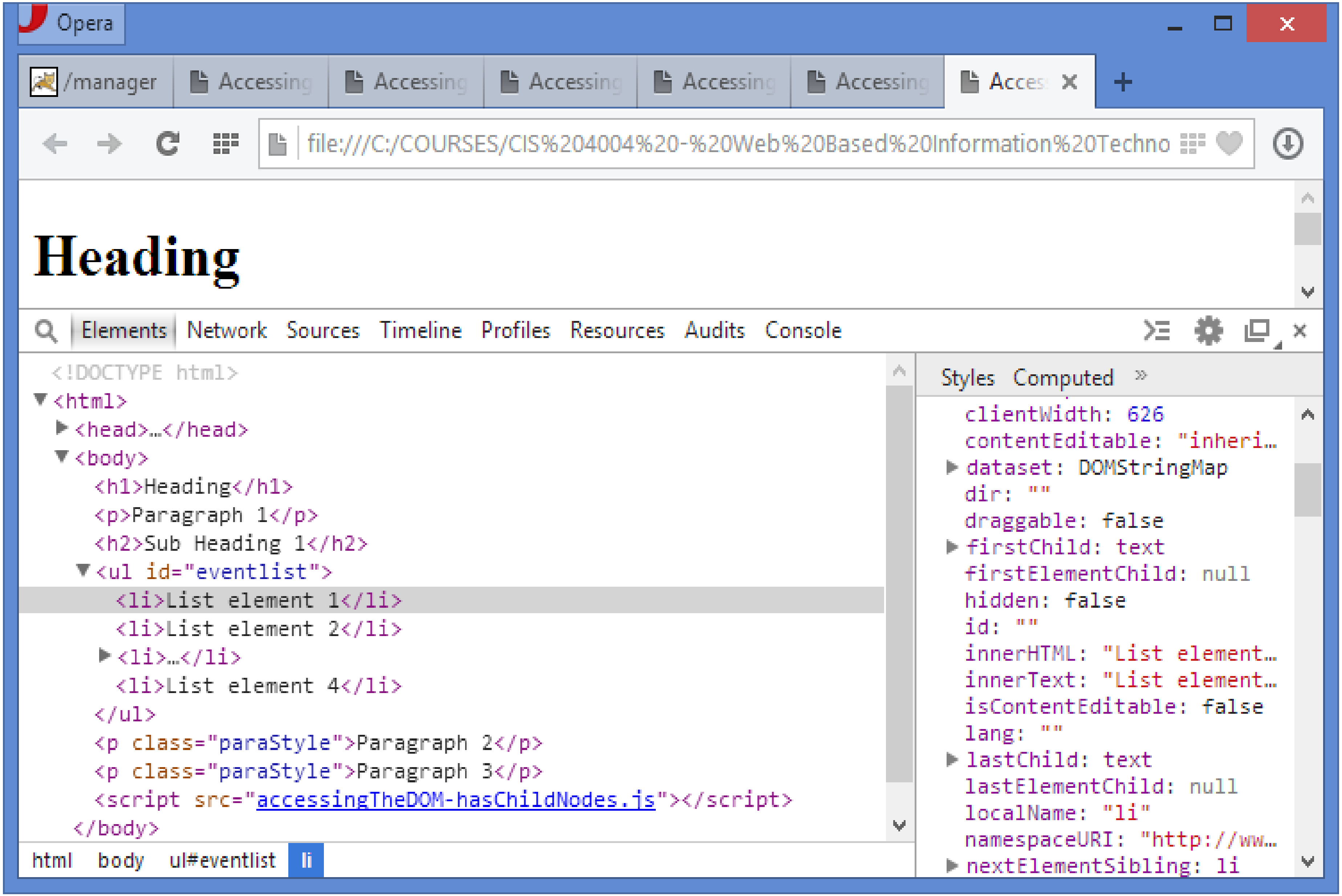
```
#text
LI
#text
LI
#text
LI
#text
LI
#text
```



Heading

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
     <h1>Heading</h1>
     <p>Paragraph 1</p>
     <h2>Sub Heading 1</h2>
    ▼<ul id="eventlist">
       <li>List element 1</li>
       <li>List element 2</li>
      ▶<li>…</li>
       <li>List element 4</li>
     </ul>
     <p class="paraStyle">Paragraph 2</p>
     <p class="paraStyle">Paragraph 3</p>
     <script src="accessingTheDOM-
     hasChildNodes.js"></script>
```

```
▼li
   accessKey: ""
  ▶attributes: NamedNodeMap
   baseURI: "file:///C:/COURSES/CIS%204004%20-…
   childElementCount: 0
  ▼childNodes: NodeList[1]
    ▼0: text
       baseURI: "file:///C:/COURSES/CIS%204004…
      ▶childNodes: NodeList[0]
       data: "List element 1"
       firstChild: null
       lastChild: null
       length: 14
       localName: null
       namespaceURI: null
       nextElementSibling: null
       nextSibling: null
       nodeName: "#text"
```



Heading

```
<!DOCTYPE html>
▼<html>
  ▶<head>…</head>
  ▼<body>
     <h1>Heading</h1>
     <p>Paragraph 1</p>
     <h2>Sub Heading 1</h2>
    ▼<ul id="eventlist">
       <li>List element 1</li>
       <li>List element 2</li>
      ▶<li>…</li>
       <li>List element 4</li>
     </ul>
     <p class="paraStyle">Paragraph 2</p>
     <p class="paraStyle">Paragraph 3</p>
     <script src="accessingTheDOM-hasChildNodes.js"></script>
   </body>
```

```
  clientWidth: 626
  contentEditable: "inheri…
▶ dataset: DOMStringMap
  dir: ""
  draggable: false
▶ firstChild: text
  firstElementChild: null
  hidden: false
  id: ""
  innerHTML: "List element…
  innerText: "List element…
  isContentEditable: false
  lang: ""
▶ lastChild: text
  lastElementChild: null
  localName: "li"
  namespaceURI: "http://ww…
▶ nextElementSibling: li
```

Traversing And Modifying A DOM Tree

- The DOM enables you to programmatically access a document's elements, allowing you to modify its contents dynamically using JavaScript.

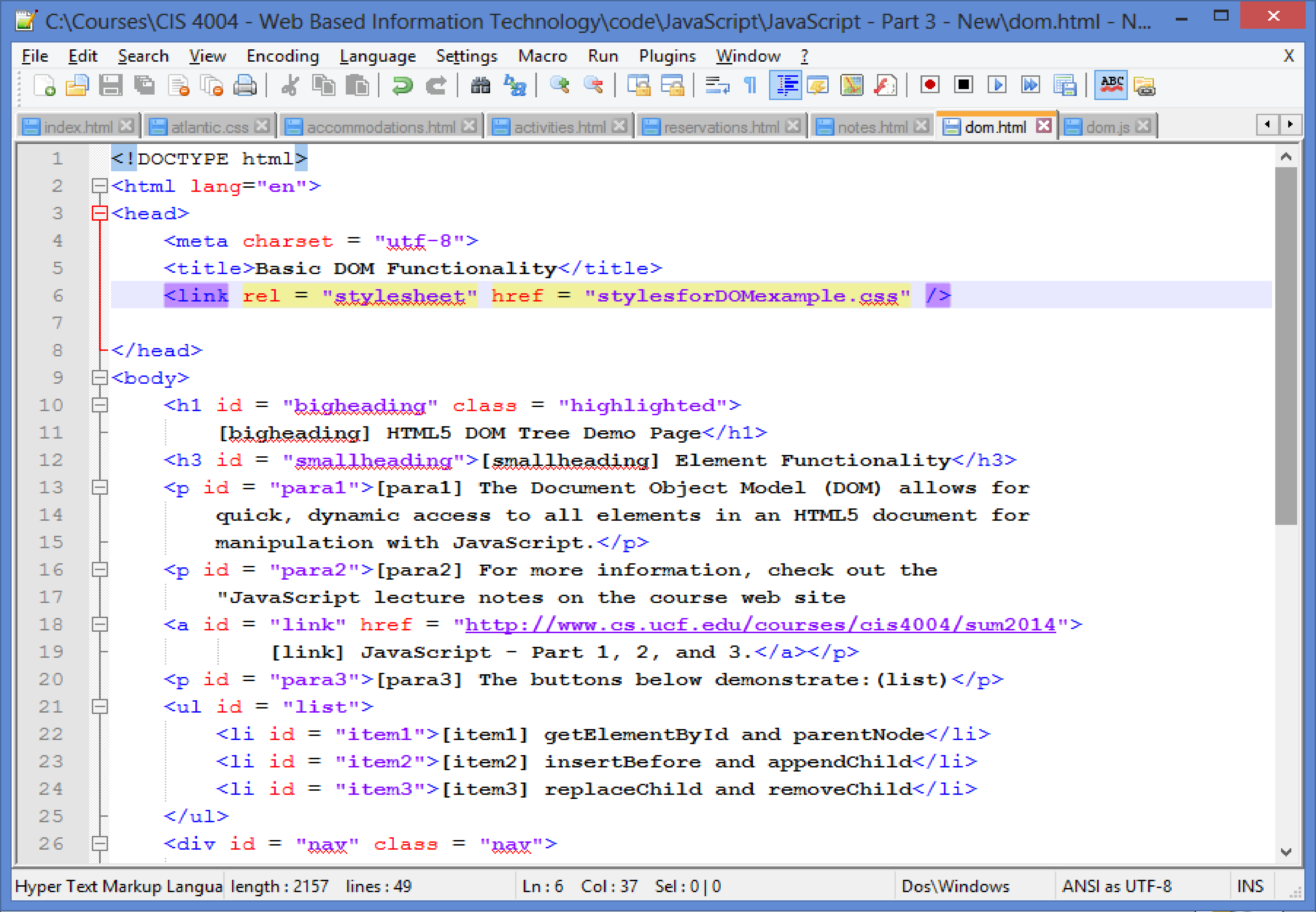
- The HTML5/CSS/JavaScript example we'll use is available on the course website, I did not include all of the markup in these notes. The example will allow you to traverse the DOM tree, modify nodes and create or delete content dynamically.

- The CSS class `highlighted` is applied dynamically to elements in the document as they are selected, added, or deleted using the form at the bottom of the document.

- As you play around with this example, be sure to do it in the developer tool so that you can see the DOM tree as well.

Traversing And Modifying A DOM Tree

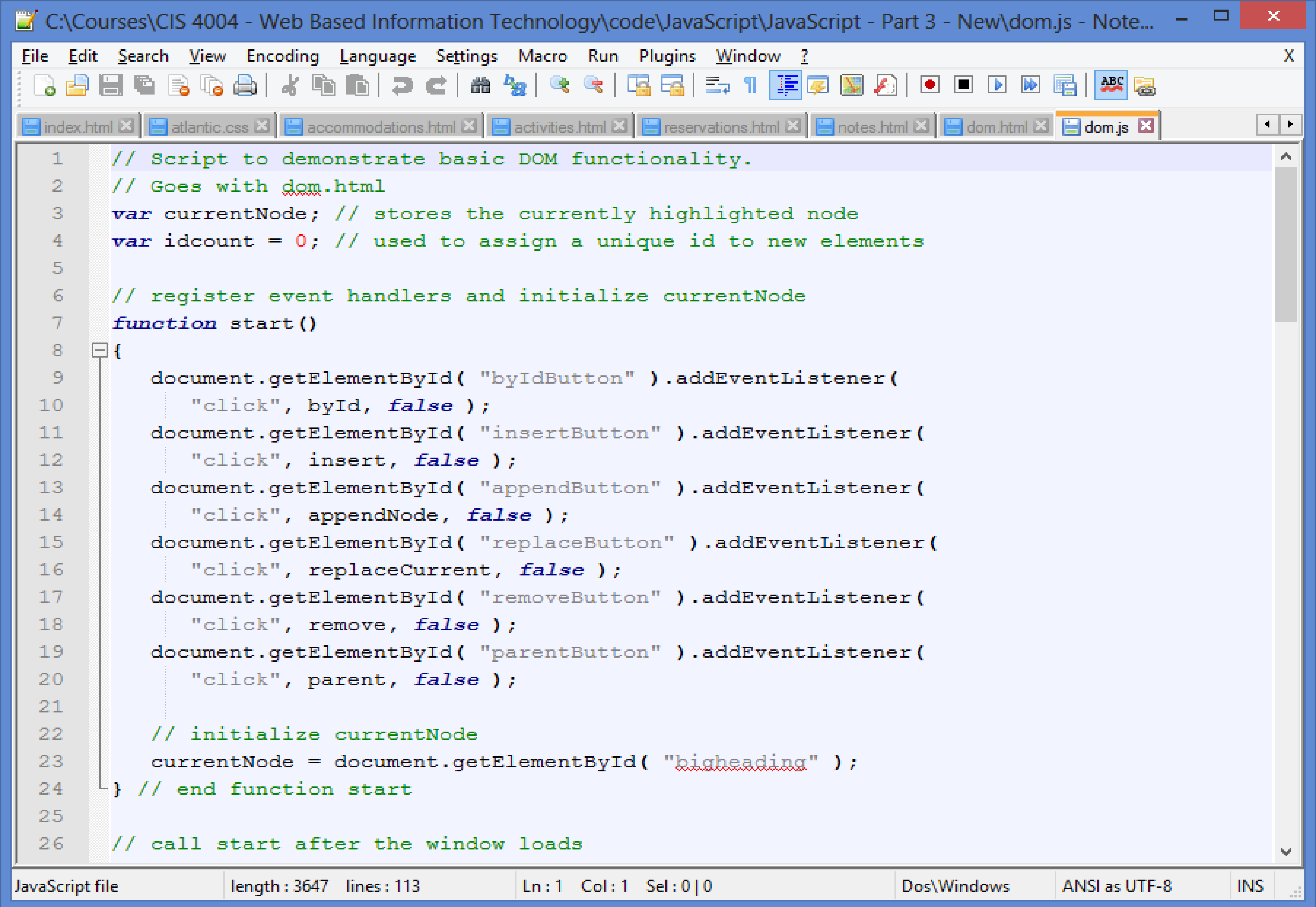
- The HTML5 document is manipulated dynamically by modifying its DOM tree.
- Each element has an `id` attribute, which is also displayed in square brackets at the beginning of the element (so you can see which element is which). (See next page for snippet of markup.)
- The `click` event listeners are registered in the JavaScript (available on the course website) for the six buttons that call corresponding functions to perform the actions described by the button's `values`.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\dom.html - N...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
index.html
atlantic.css
accommodations.html
activities.html
reservations.html
notes.html
dom.html
dom.js
Hyper Text Markup Langua length : 2157 lines : 49
Ln : 6 Col : 37 Sel : 0 | 0
Dos\Windows
ANSI as UTF-8
INS

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset = "utf-8">
    <title>Basic DOM Functionality</title>
    <link rel = "stylesheet" href = "stylesforDOMexample.css" />

</head>
<body>
    <h1 id = "bigheading" class = "highlighted">
        [bigheading] HTML5 DOM Tree Demo Page</h1>
    <h3 id = "smallheading">[smallheading] Element Functionality</h3>
    <p id = "para1">[para1] The Document Object Model (DOM) allows for
        quick, dynamic access to all elements in an HTML5 document for
        manipulation with JavaScript.</p>
    <p id = "para2">[para2] For more information, check out the
        "JavaScript lecture notes on the course web site
    <a id = "link" href = "http://www.cs.ucf.edu/courses/cis4004/sum2014">
            [link] JavaScript - Part 1, 2, and 3.</a></p>
    <p id = "para3">[para3] The buttons below demonstrate:(list)</p>
    <ul id = "list">
        <li id = "item1">[item1] getElementById and parentNode</li>
        <li id = "item2">[item2] insertBefore and appendChild</li>
        <li id = "item3">[item3] replaceChild and removeChild</li>
    </ul>
    <div id = "nav" class = "nav">
```

Traversing And Modifying A DOM Tree

- The JavaScript begins by declaring two variables.
- Variable `currentNode` keeps track of the currently highlighted node (the initially highlighted node is the `[bigheading]`, the functionality of each button depends on which node in the document (DOM tree) is currently selected.
- The function `start` registers the event handlers for the buttons, then initializes the `currentNode` to the `<h1>` element, the element with `id = bigheading`.
- Note that the function `start` is called when the window's `load` event occurs.


C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\dom.js - Note...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
index.html atlantic.css accommodations.html activities.html reservations.html notes.html dom.html dom.js
JavaScript file length : 3647 lines : 113 Ln : 1 Col : 1 Sel : 0 | 0 Dos\Windows ANSI as UTF-8 INS

```
// Script to demonstrate basic DOM functionality.
// Goes with dom.html
var currentNode; // stores the currently highlighted node
var idcount = 0; // used to assign a unique id to new elements

// register event handlers and initialize currentNode
function start()
{
   document.getElementById( "byIdButton" ).addEventListener(
      "click", byId, false );
   document.getElementById( "insertButton" ).addEventListener(
      "click", insert, false );
   document.getElementById( "appendButton" ).addEventListener(
      "click", appendNode, false );
   document.getElementById( "replaceButton" ).addEventListener(
      "click", replaceCurrent, false );
   document.getElementById( "removeButton" ).addEventListener(
      "click", remove, false );
   document.getElementById( "parentButton" ).addEventListener(
      "click", parent, false );

   // initialize currentNode
   currentNode = document.getElementById( "bigheading" );
} // end function start

// call start after the window loads
```

Traversing And Modifying A DOM Tree

- The JavaScript variable `idcount` is used to assign a unique id to any new elements that are dynamically created by the user.
- The remainder of the JavaScript contains the event handling functions for the buttons and two helper functions (`switchTo` and `createNewNode`) that are called by the event handlers.
- Over the next few pages, I'll explain how each of the buttons and its corresponding event handler works. Before reading on, you should download the markup, the style sheet, and the JavaScript files and play around with the page a bit to get a feel for what's happening with the page as the user manipulates the page.

Finding and Highlighting an Element Using `getElementById`, `setAttribute` and `getAttribute`

- The first row of the form allows the user to enter the `id` of an element into the text field and click the `Get By id` button to find and highlight the element.
- The button’s click event calls function `byId()`.

```
// 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
```

Finding and Highlighting an Element Using `getElementById`, `setAttribute` and `getAttribute`

- First, the `byId()` function uses `getElementById` to assign the contents of the text field to the variable `id`.
- Next , the `byID()` function uses `getElementById` to find the element whose id attribute matches the value of variable `id` and assigns this to the variable `target`.
- If an element is found with the specified `id`, and object is returned; otherwise, `null` is returned.
- Next, the function checks to see whether `target` is an object (any object used as a boolean expression is true, while `null` is false). If `target` evaluates to true, the `switchTo()` helper function is called with `target` as its argument.

Finding and Highlighting an Element Using `getElementById`, `setAttribute` and `getAttribute`

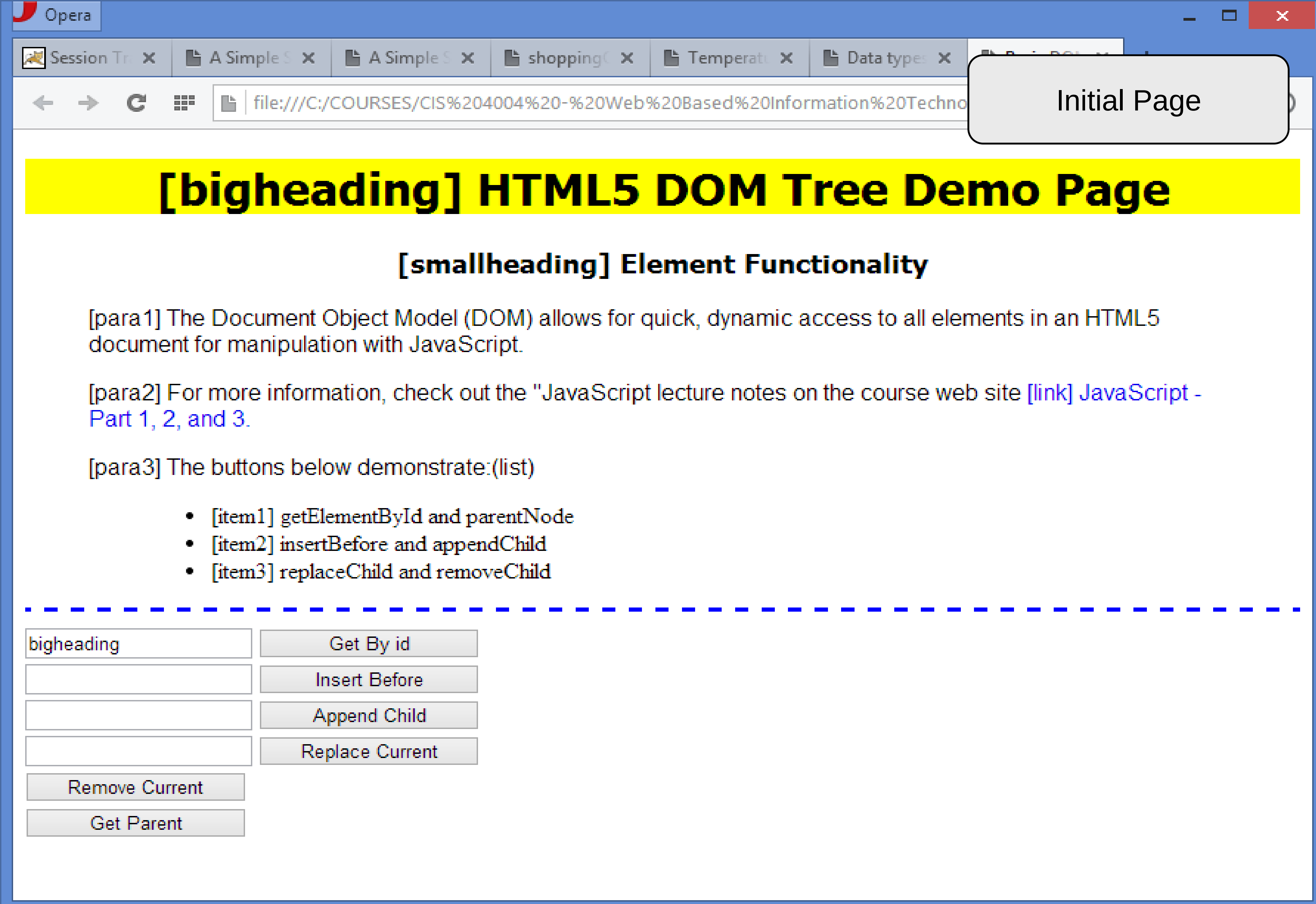
- The `switchTo()` helper function is used a lot in this JavaScript to highlight an element in the page. The current element is given a yellow background (via the CSS class `highlighted`).

- The DOM element methods `setAttribute` and `getAttribute` allow you to modify and get an attribute's value, respectively.

- The function `switchTo` function uses the `setAttribute` method to set the current node's class attribute to the empty string. This clears the `class` attribute to remove the highlighted class from the `currentNode` before the new node is highlighted.

Finding and Highlighting an Element Using `getElementById`, `setAttribute` and `getAttribute`

- The last thing the `byID` function does is uses the `getAttribute` method to get the `currentNode`'s `id` and assign it to the input field's `value` property.
- This isn't necessary when this helper function is called by `byID`, but as you'll see later, other functions call `switchTo` as well. In these cases, this line ensures that the text field's `value` contains the currently selected node's `id`.
- Notice that `setAttribute` was not used to change the `value` of the input field. Methods `setAttribute` and `getAttribute` do not work for user-modifiable content, such as the value displayed in an input field.



Initial Page

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

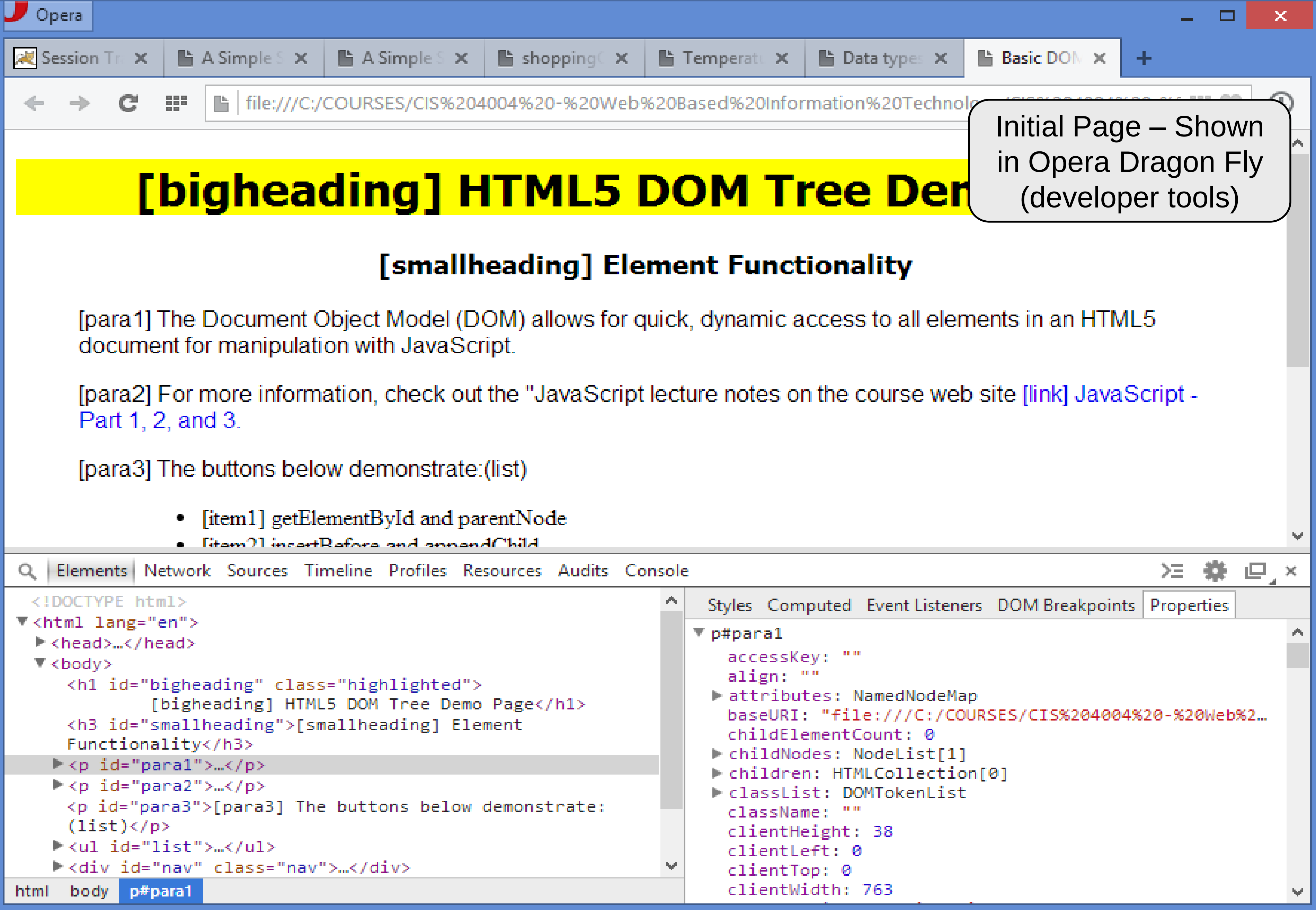
[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

bigheading



Initial Page – Shown in Opera Dragon Fly (developer tools)

[bigheading] HTML5 DOM Tree Dem

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

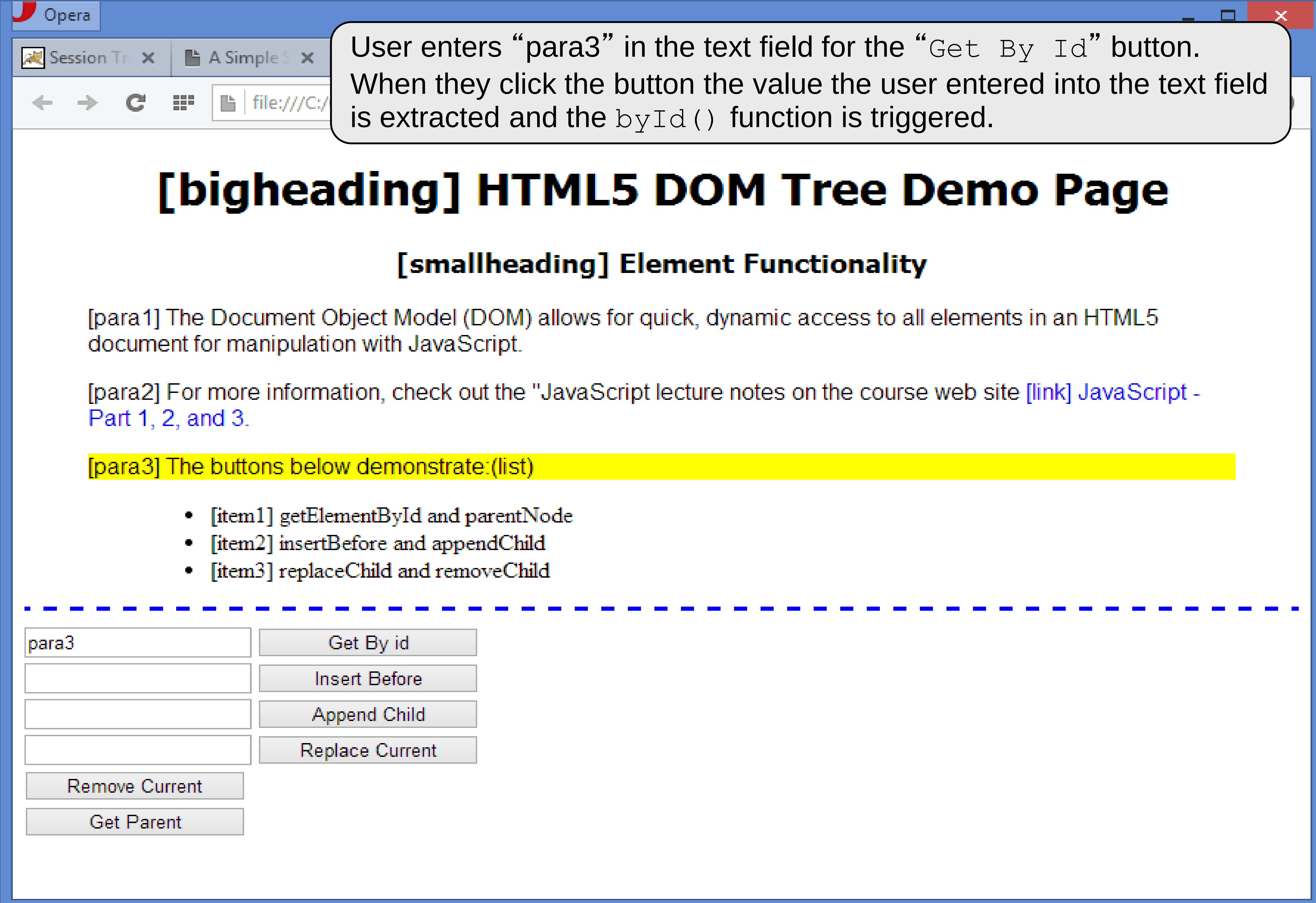
[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild

```
<!DOCTYPE html>
▼<html lang="en">
  ▶<head>…</head>
  ▼<body>
    <h1 id="bigheading" class="highlighted">
             [bigheading] HTML5 DOM Tree Demo Page</h1>
    <h3 id="smallheading">[smallheading] Element
    Functionality</h3>
   ▶<p id="para1">…</p>
   ▶<p id="para2">…</p>
    <p id="para3">[para3] The buttons below demonstrate:
    (list)</p>
   ▶<ul id="list">…</ul>
   ▶<div id="nav" class="nav">…</div>
```

```
▼ p#para1
    accessKey: ""
    align: ""
  ▶ attributes: NamedNodeMap
    baseURI: "file:///C:/COURSES/CIS%204004%20-%20Web%2…
    childElementCount: 0
  ▶ childNodes: NodeList[1]
  ▶ children: HTMLCollection[0]
  ▶ classList: DOMTokenList
    className: ""
    clientHeight: 38
    clientLeft: 0
    clientTop: 0
    clientWidth: 763
```

User enters “para3” in the text field for the “`Get By Id`” button. When they click the button the value the user entered into the text field is extracted and the `byId()` function is triggered.



[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

para3 | Get By id

| Insert Before

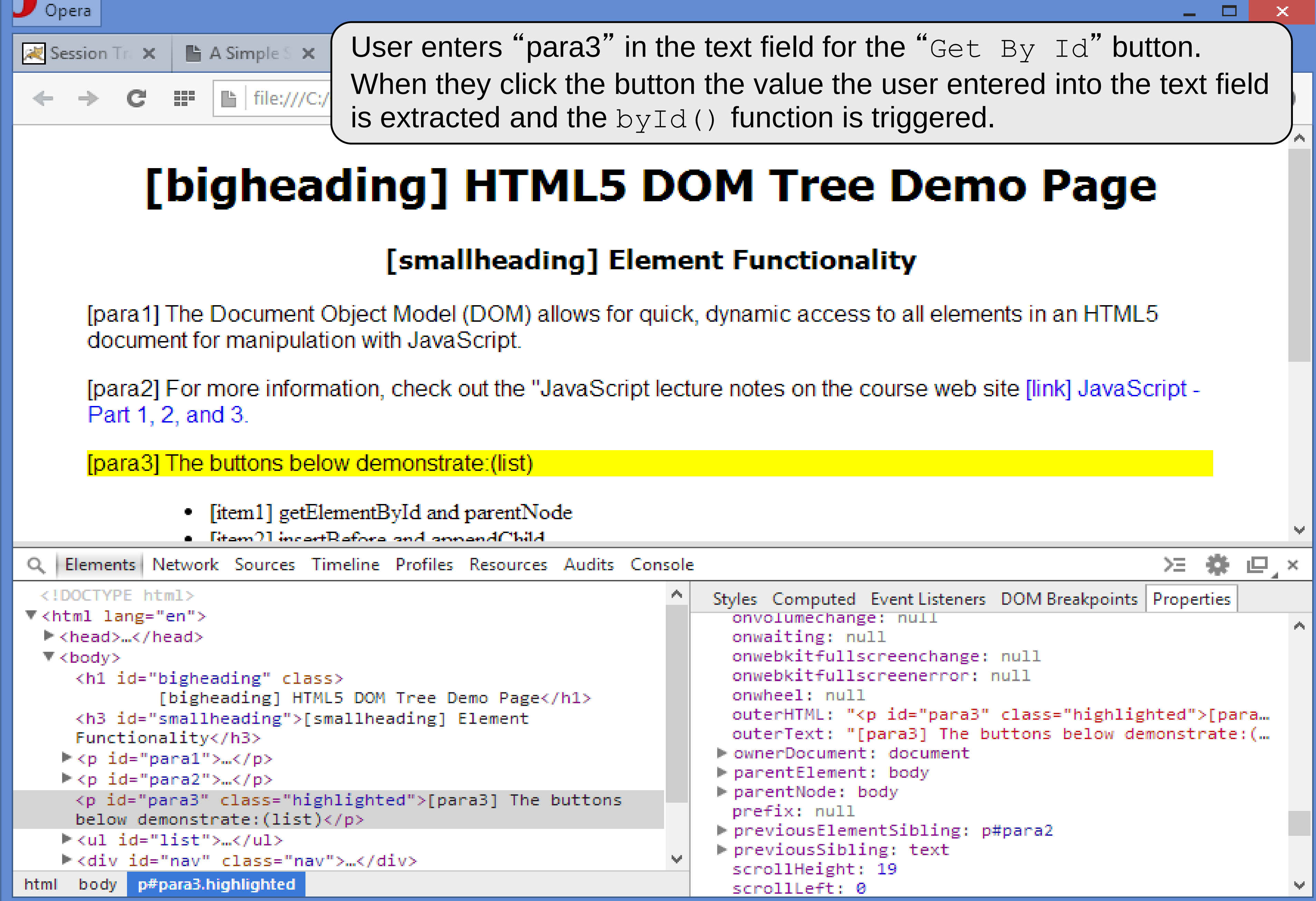
| Append Child

| Replace Current

Remove Current

Get Parent

User enters “para3” in the text field for the “`Get By Id`” button. When they click the button the value the user entered into the text field is extracted and the `byId()` function is triggered.



[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild

```
<!DOCTYPE html>
▼<html lang="en">
  ▶<head>…</head>
  ▼<body>
     <h1 id="bigheading" class>
              [bigheading] HTML5 DOM Tree Demo Page</h1>
     <h3 id="smallheading">[smallheading] Element
     Functionality</h3>
    ▶<p id="para1">…</p>
    ▶<p id="para2">…</p>
     <p id="para3" class="highlighted">[para3] The buttons
     below demonstrate:(list)</p>
    ▶<ul id="list">…</ul>
    ▶<div id="nav" class="nav">…</div>
```

html body p#para3.highlighted

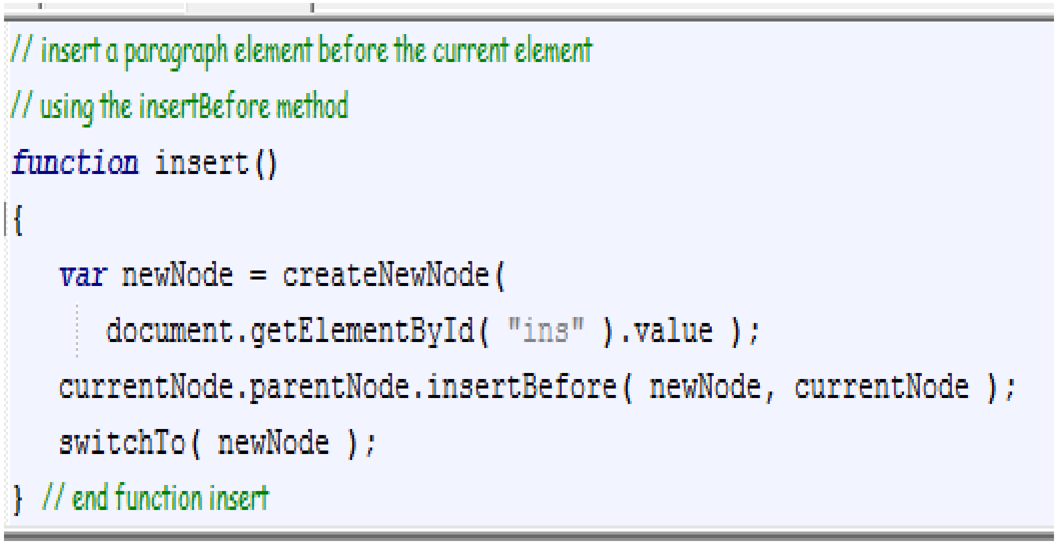
```
  onvolumechange: null
  onwaiting: null
  onwebkitfullscreenchange: null
  onwebkitfullscreenerror: null
  onwheel: null
  outerHTML: "<p id="para3" class="highlighted">[para…
  outerText: "[para3] The buttons below demonstrate:(…
▶ ownerDocument: document
▶ parentElement: body
▶ parentNode: body
  prefix: null
▶ previousElementSibling: p#para2
▶ previousSibling: text
  scrollHeight: 19
  scrollLeft: 0
```

Creating and Inserting New Elements Using `insertBefore` and `appendChild`

- The second and third rows of the form allow the user to create a new element and insert it before or as a child of the current node, respectively.
- If the user enters text in the second text field and clicks the `Insert Before` button, the text is placed in a new paragraph element, which is inserted into the document before the currently selected element.
- The `Insert Before` button's click event calls function `insert()`.

Creating and Inserting New Elements Using `insertBefore` and `appendChild`

- The `insert()` function calls the `createNewNode()` function , passing it the value of the “ins” input field as an argument.
- The helper function `createNewNode()` creates a paragraph node that contains the text passed to it.

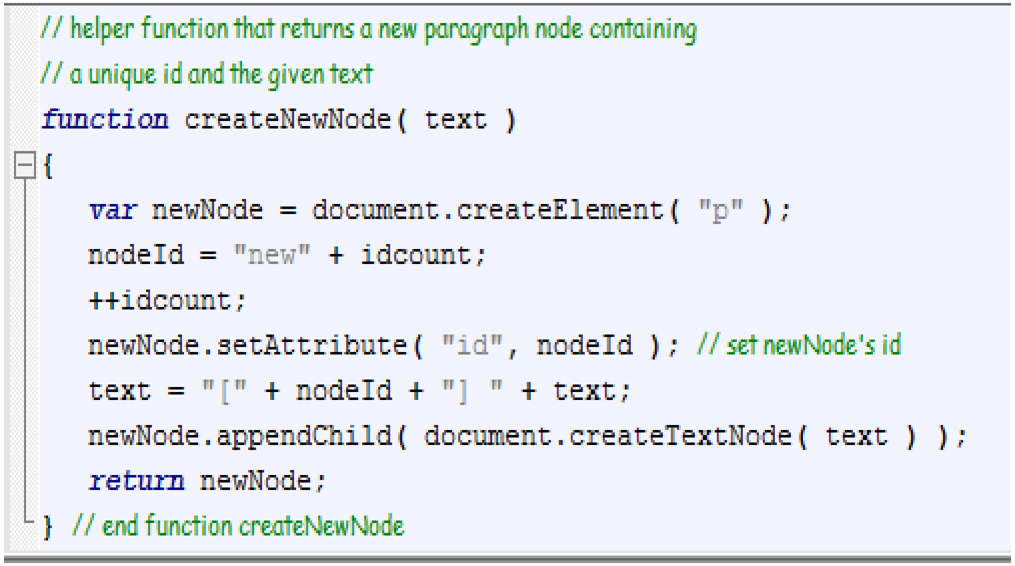


```
// insert a paragraph element before the current element
// using the insertBefore method
function insert()
{
   var newNode = createNewNode(
      document.getElementById( "ins" ).value );
   currentNode.parentNode.insertBefore( newNode, currentNode );
   switchTo( newNode );
} // end function insert
```

Creating and Inserting New Elements Using `insertBefore` and `appendChild`

- Function `createNewNode()` creates a `<p>` element using the document’s `createElement` method, which creates a new DOM node, taking the tag name as an argument.

- The `createElement` method creates an element…it does ***not*** insert the element on the page.



```
// helper function that returns a new paragraph node containing
// a unique id and the given text
function createNewNode( text )
{
   var newNode = document.createElement( "p" );
   nodeId = "new" + idcount;
   ++idcount;
   newNode.setAttribute( "id", nodeId ); // set newNode's id
   text = "[" + nodeId + "] " + text;
   newNode.appendChild( document.createTextNode( text ) );
   return newNode;
} // end function createNewNode
```

Creating and Inserting New Elements Using `insertBefore` and `appendChild`

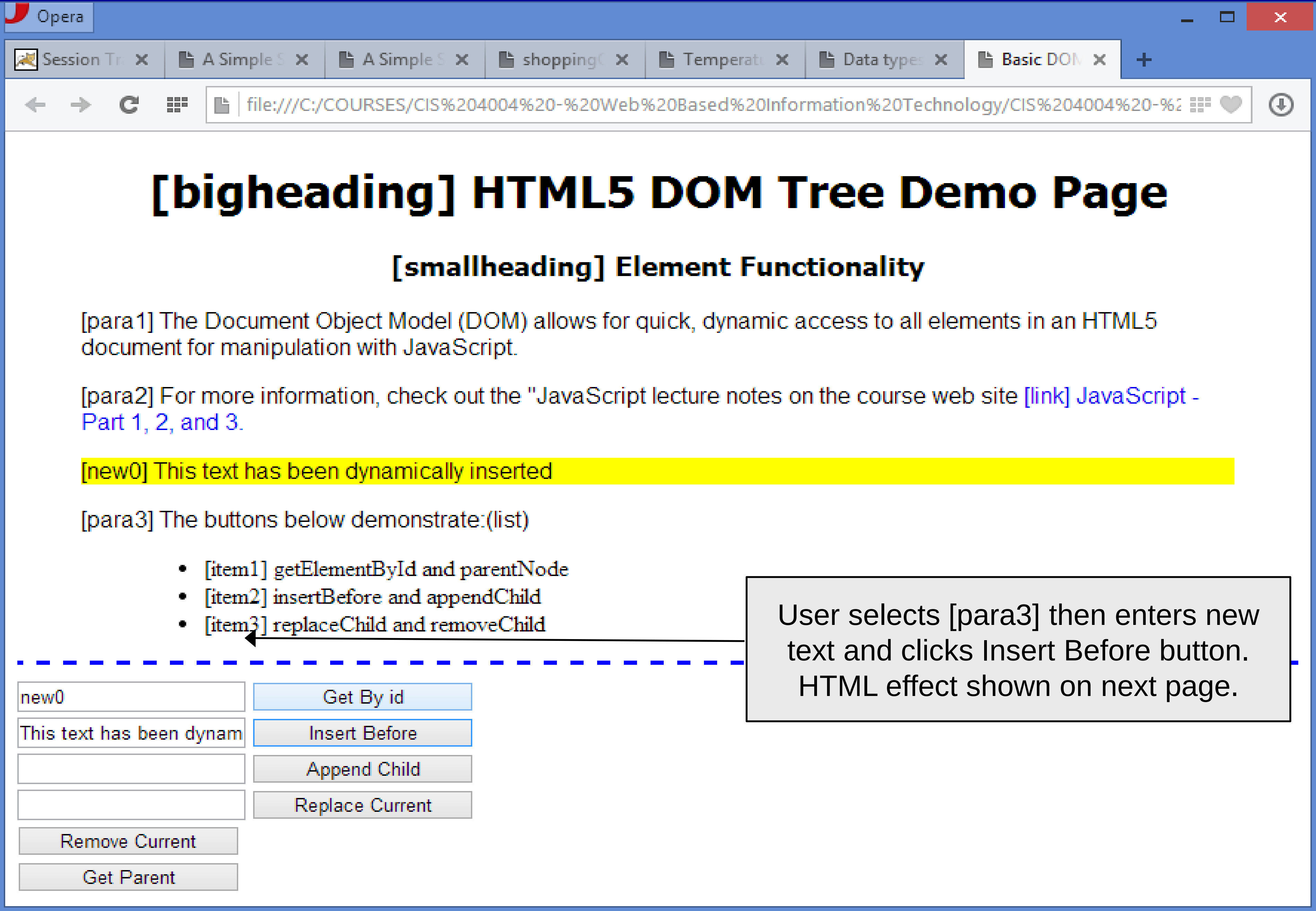
- To create the new element, a unique `id` for it is created by concatenating the string “`new`” with the current value of `idcount`.
- The `setAttribute` function is then called to set the `id` of the new element.
- The value of the text is concatenated with the square brackets used to identify the nodes to the user.
- Then the document’s `createTextNode` method is called to create a node that contains only text. This new node is then used as the argument to the `appendChild` method, which inserts a child node after any existing children of the node on which it is called.

Creating and Inserting New Elements Using `insertBefore` and `appendChild`

- After the `<p>` element is created by `createNewNode` that function returns the new node to the `insert` function, where it' s assigned to the variable `newNode`.
- The `newNode` is then inserted before the currently selected node.
- The `parentNode` property contains a node's parent. This property is used in the `insert` function to get the current node's parent. Then the `insertBefore` method is invoked on the parent node with `newNode` and `currentNode` as its arguments. This causes `newNode` to be inserted as a child of the parent directly before `currentNode`.

Creating and Inserting New Elements Using `insertBefore` and `appendChild`

- Finally, the `switchTo` helper function is called to set the highlighted class on the newly created element.
- The input field and button on the third line of the input form allows the user to append a new paragraph node as a child of the current element.
- This is done in a similar manner to the `Insert Before` button's `insert` function. However, in this case the function `appendNode` creates the new node and inserts it as a child of the current node. Examine the JavaScript more closely to see how this mirrors the insert function and also how it differs.


Opera
Session Tr
A Simple S
A Simple S
shopping(
Temperatu
Data types
Basic DOM
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20Technology/CIS%204004%20-%2

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[new0] This text has been dynamically inserted

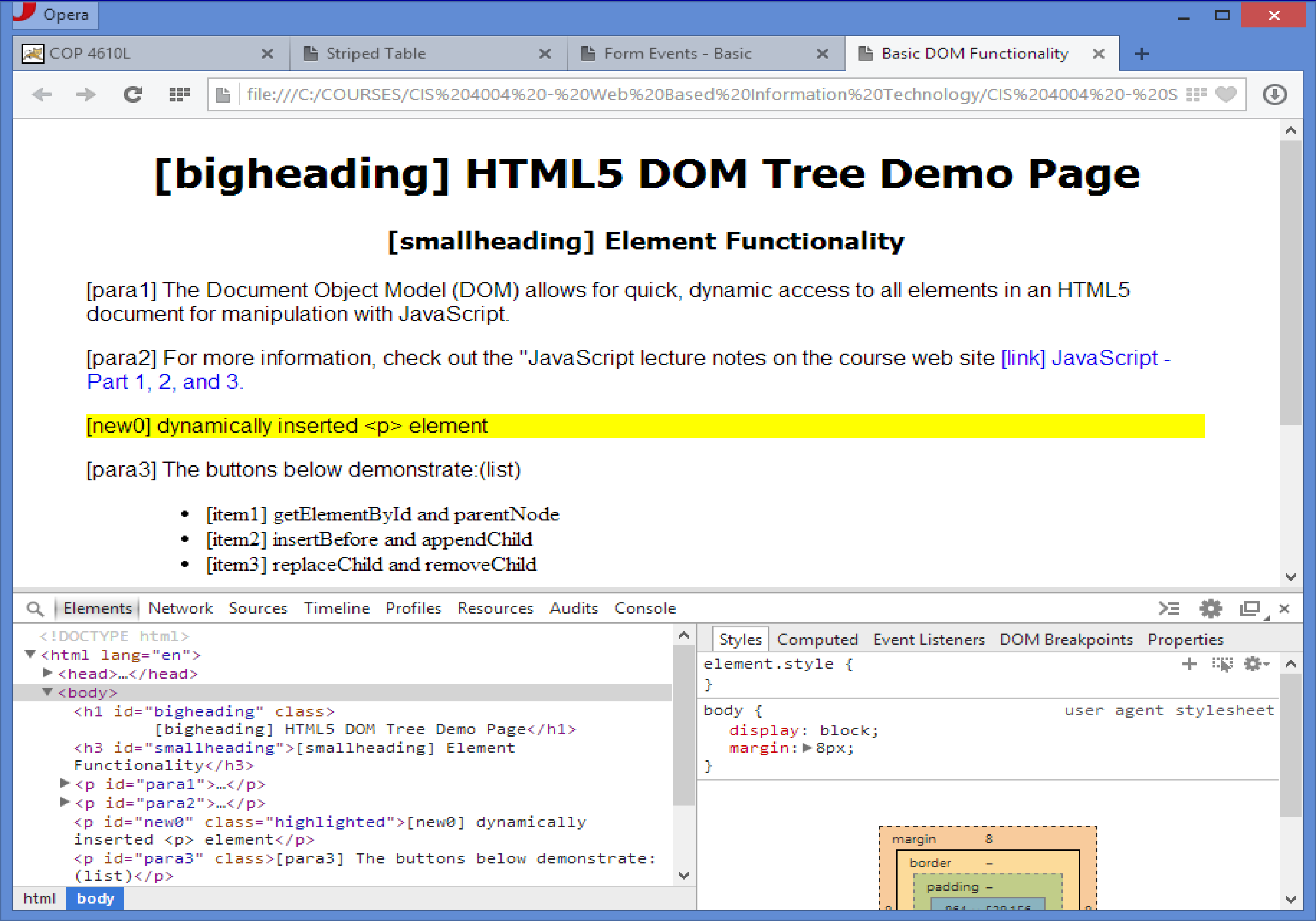
[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

new0

This text has been dynam

User selects [para3] then enters new text and clicks Insert Before button. HTML effect shown on next page.



[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[new0] dynamically inserted <p> element

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

```
<!DOCTYPE html>
▼<html lang="en">
  ▶<head>…</head>
  ▼<body>
      <h1 id="bigheading" class>
            [bigheading] HTML5 DOM Tree Demo Page</h1>
      <h3 id="smallheading">[smallheading] Element
      Functionality</h3>
    ▶<p id="para1">…</p>
    ▶<p id="para2">…</p>
      <p id="new0" class="highlighted">[new0] dynamically
      inserted <p> element</p>
      <p id="para3" class>[para3] The buttons below demonstrate:
      (list)</p>
```

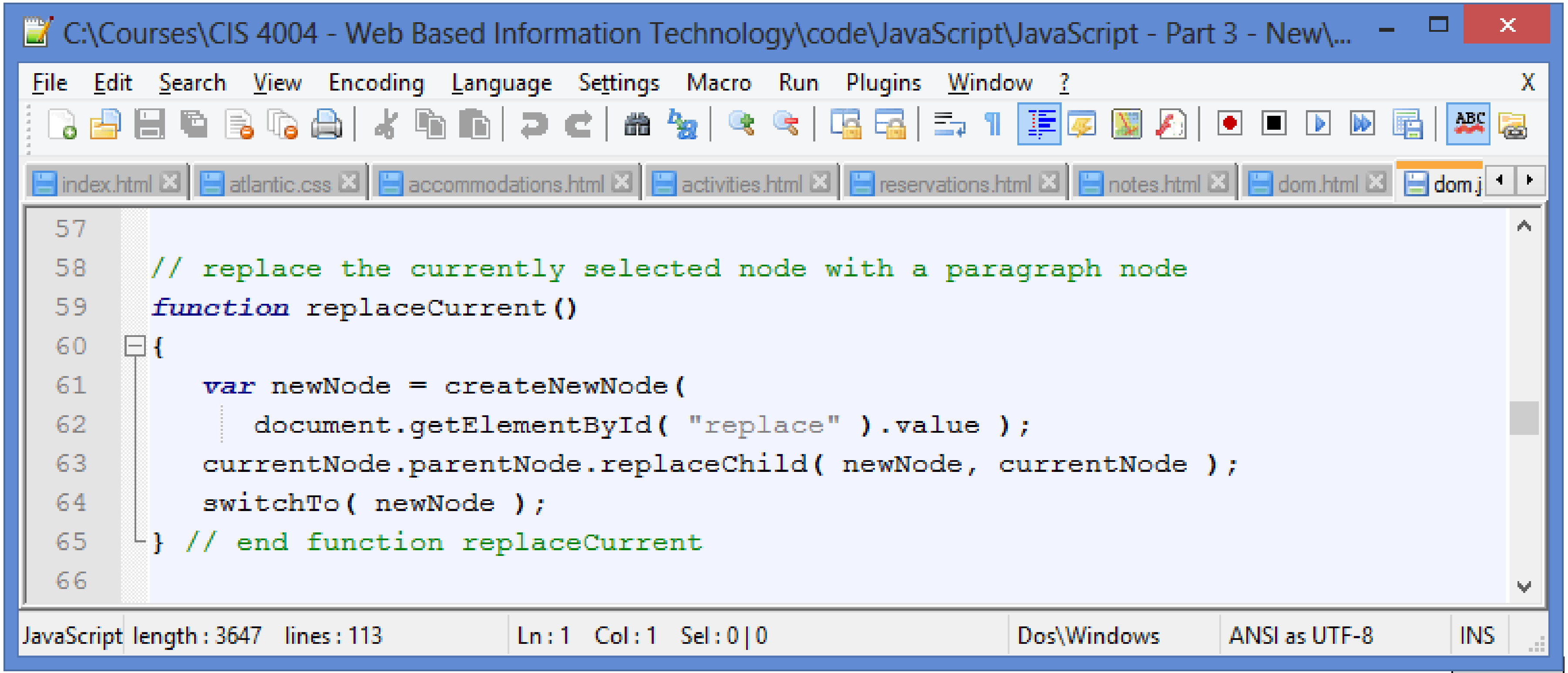
```
element.style {
}
body {                                       user agent stylesheet
  display: block;
  margin:▶8px;
}
```

Replacing and Removing Elements Using `replaceChild` and `removeChild`

- The next two buttons on the input form provide the user with the ability to replace the current element with a new `<p>` element or simply to remove the element entirely.
- When the user clicks the `Replace Current` button, the function `replaceCurrent` is called.
- In function `replaceCurrent`, the `createNewNode` helper function is called in much the same manner as it was when the `InsertBefore` or `AppendChild` buttons were clicked.
- The user’s text is retrieved from the input field in the form and the parent of the current node is determined, then the `replaceChild` method is invoked on the parent.

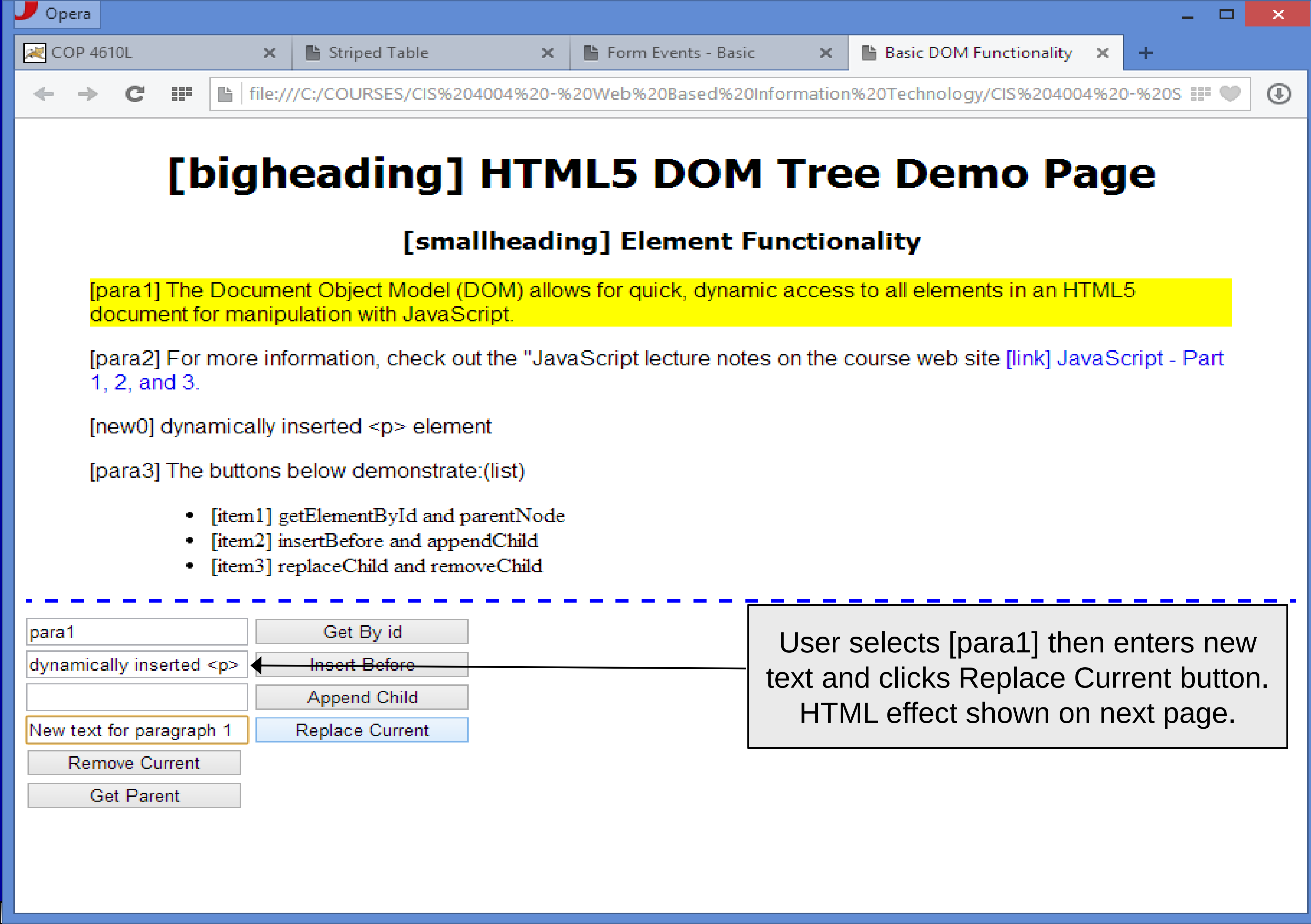
Replacing and Removing Elements Using `replaceChild` and `removeChild`

- The `replaceChild` method takes two arguments, the first of which is the new node to be inserted, and the second is the node to be replaced.



C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
index.html atlantic.css accommodations.html activities.html reservations.html notes.html dom.html dom.j
JavaScript length : 3647 lines : 113 Ln : 1 Col : 1 Sel : 0 | 0 Dos\Windows ANSI as UTF-8 INS

```

// 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

```


Opera
COP 4610L
Striped Table
Form Events - Basic
Basic DOM Functionality
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20Technology/CIS%204004%20-%20S

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

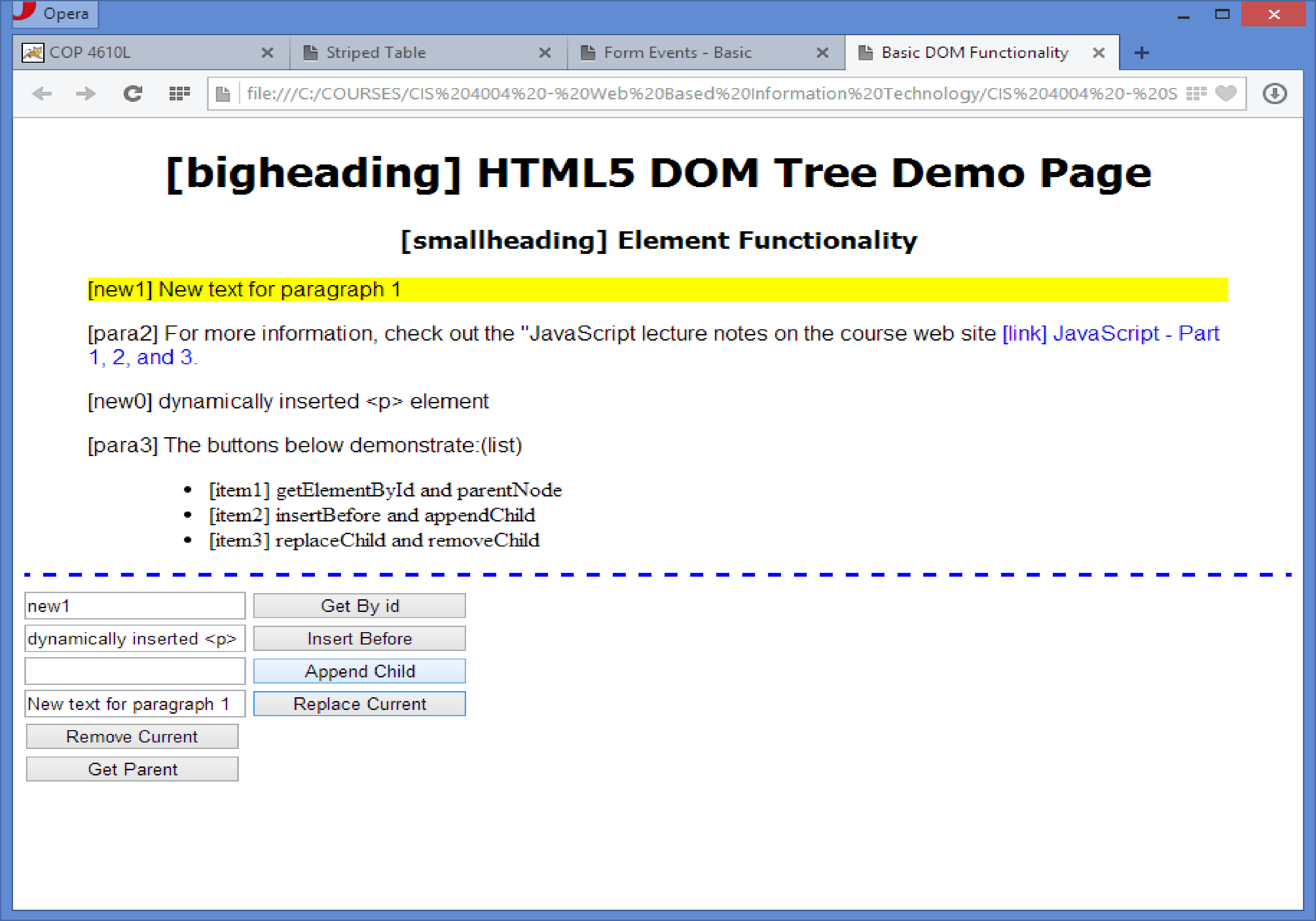
[new0] dynamically inserted <p> element

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

para1	Get By id
dynamically inserted <p>	Insert Before
	Append Child
New text for paragraph 1	Replace Current
Remove Current	
Get Parent	

User selects [para1] then enters new text and clicks Replace Current button. HTML effect shown on next page.


Opera
COP 4610L
Striped Table
Form Events - Basic
Basic DOM Functionality
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20Technology/CIS%204004%20-%20S

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[new1] New text for paragraph 1

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[new0] dynamically inserted <p> element

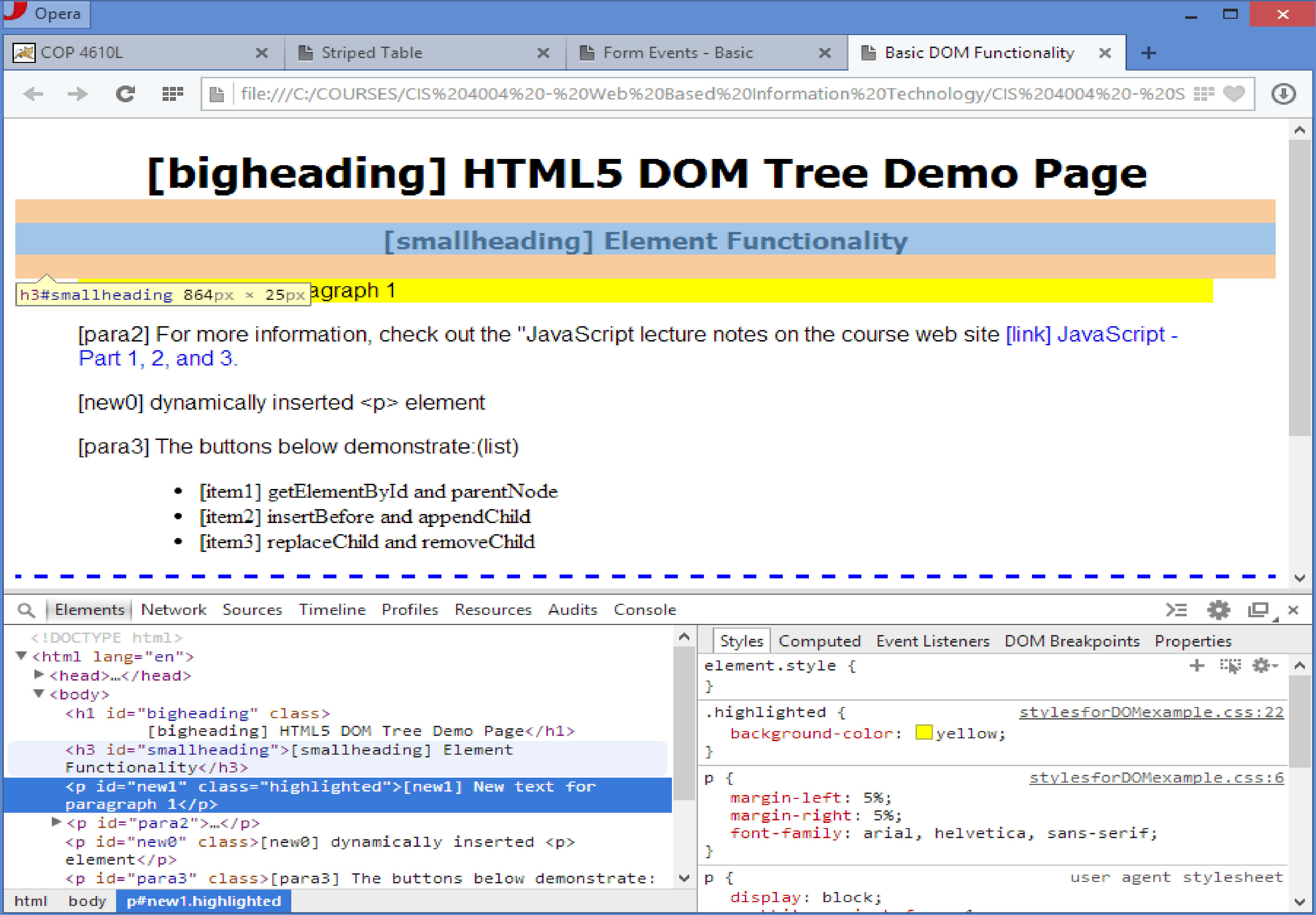
[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

new1

dynamically inserted <p>

New text for paragraph 1



[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

agraph 1

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[new0] dynamically inserted <p> element

[para3] The buttons below demonstrate:(list)

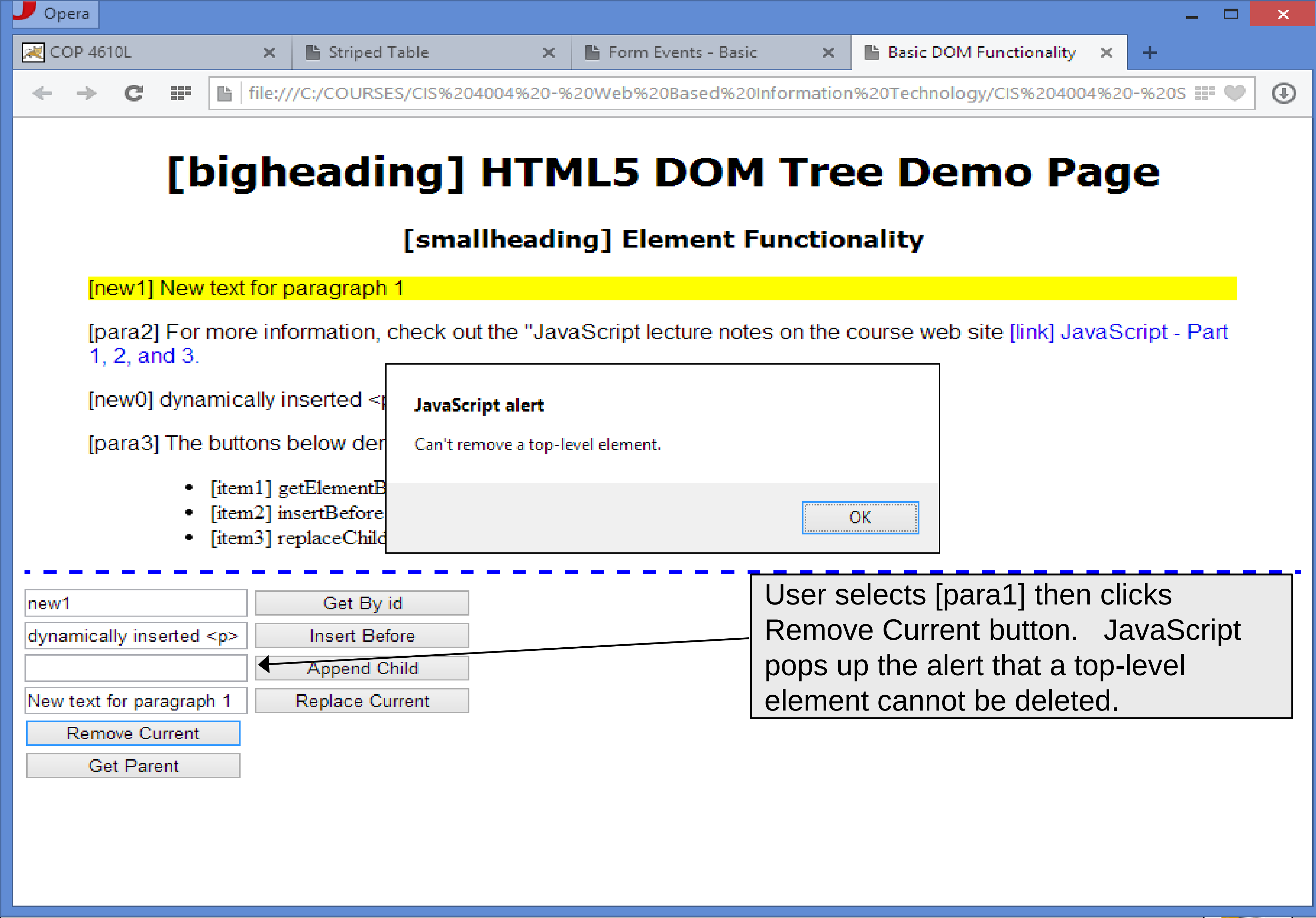
- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

```
<!DOCTYPE html>
▼<html lang="en">
  ▶<head>…</head>
  ▼<body>
     <h1 id="bigheading" class>
              [bigheading] HTML5 DOM Tree Demo Page</h1>
     <h3 id="smallheading">[smallheading] Element
     Functionality</h3>
     <p id="new1" class="highlighted">[new1] New text for
     paragraph 1</p>
    ▶<p id="para2">…</p>
     <p id="new0" class>[new0] dynamically inserted <p>
     element</p>
     <p id="para3" class>[para3] The buttons below demonstrate:
```

```
element.style {
}
.highlighted {                          stylesforDOMexample.css:22
  background-color: yellow;
}
p {                                      stylesforDOMexample.css:6
  margin-left: 5%;
  margin-right: 5%;
  font-family: arial, helvetica, sans-serif;
}
p {                                          user agent stylesheet
  display: block;
```

Replacing and Removing Elements Using `replaceChild` and `removeChild`

- Clicking the Remove Current button calls the remove function in the JavaScript which removes the currently selected element entirely and highlights the parent.
- If the node’s parent is the body element, an error message is displayed to indicate that a top level element cannot be deleted.
- The next page illustrates this error condition.



[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[new1] New text for paragraph 1

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[new0] dynamically inserted <p

[para3] The buttons below der

- [item1] getElementB
- [item2] insertBefore
- [item3] replaceChild

JavaScript alert

Can't remove a top-level element.

new1

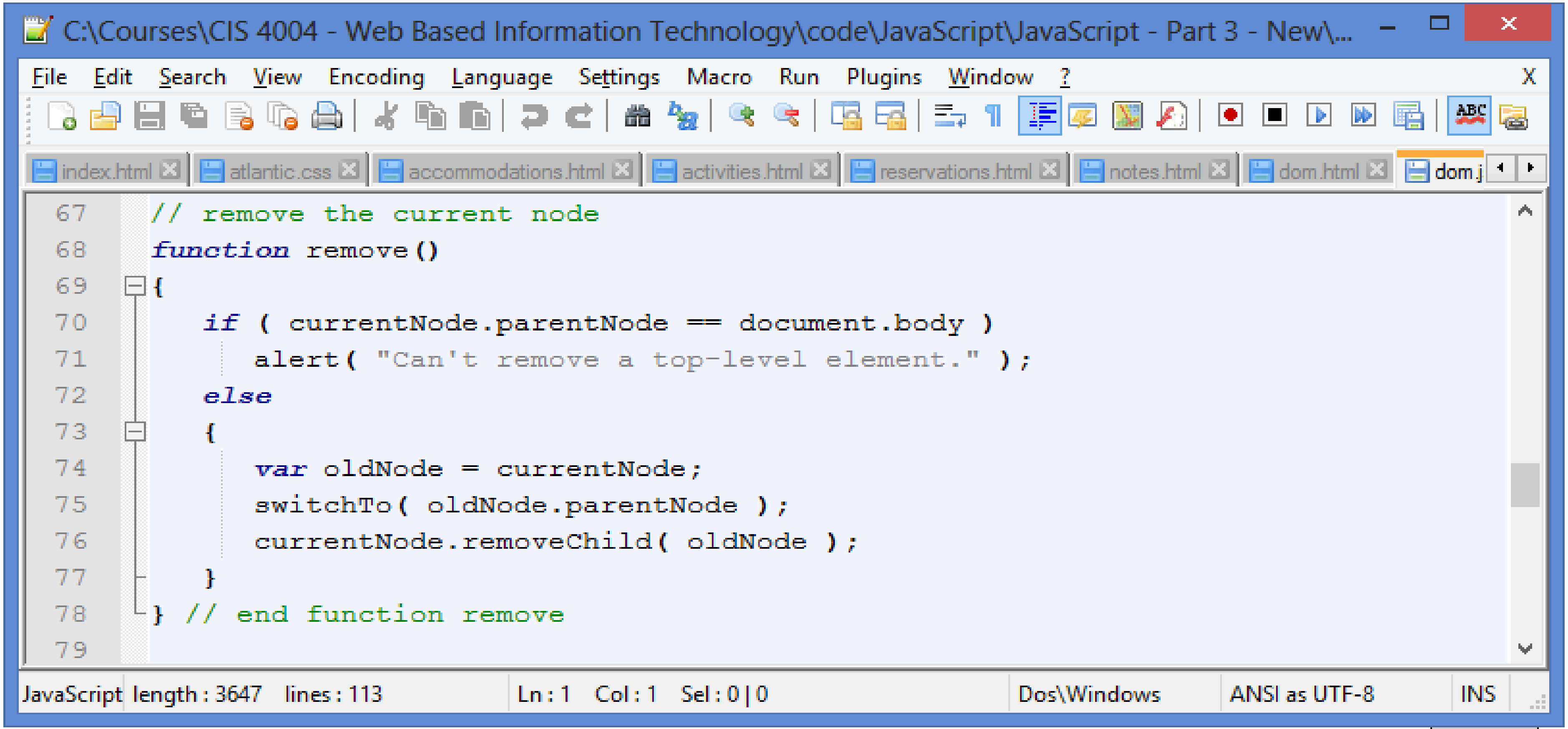
dynamically inserted <p>

New text for paragraph 1

User selects [para1] then clicks Remove Current button. JavaScript pops up the alert that a top-level element cannot be deleted.

Replacing and Removing Elements Using `replaceChild` and `removeChild`

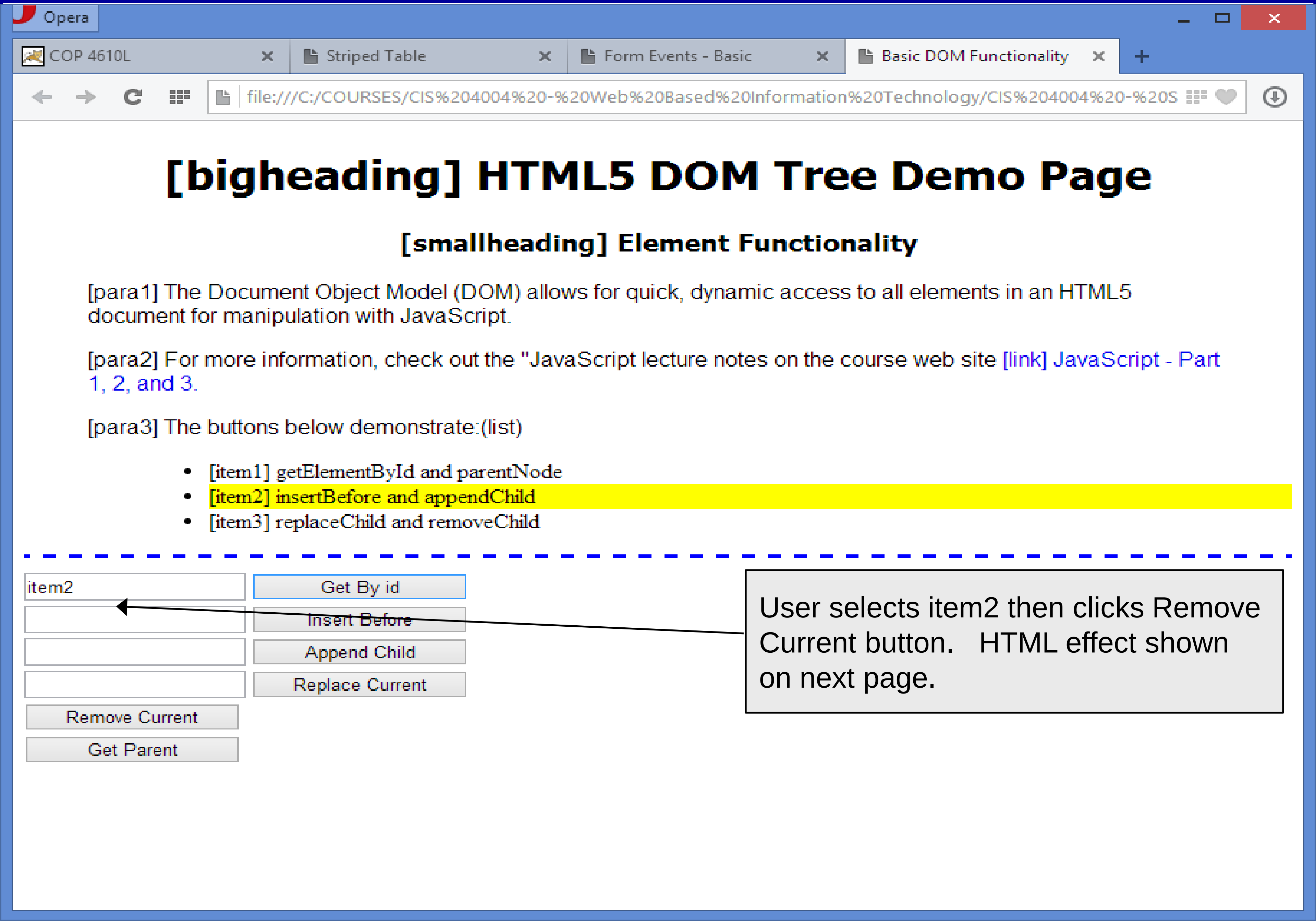
- In general, `parent.removeChild( child )` looks in a parent’s list of children for child and removes it.



C:\Courses\CIS 4004 - Web Based Information Technology\code\JavaScript\JavaScript - Part 3 - New\...
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
index.html atlantic.css accommodations.html activities.html reservations.html notes.html dom.html dom.j

```
// 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
```

JavaScript length : 3647 lines : 113 Ln : 1 Col : 1 Sel : 0 | 0 Dos\Windows ANSI as UTF-8 INS


Opera
COP 4610L
Striped Table
Form Events - Basic
Basic DOM Functionality
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20Technology/CIS%204004%20-%20S

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

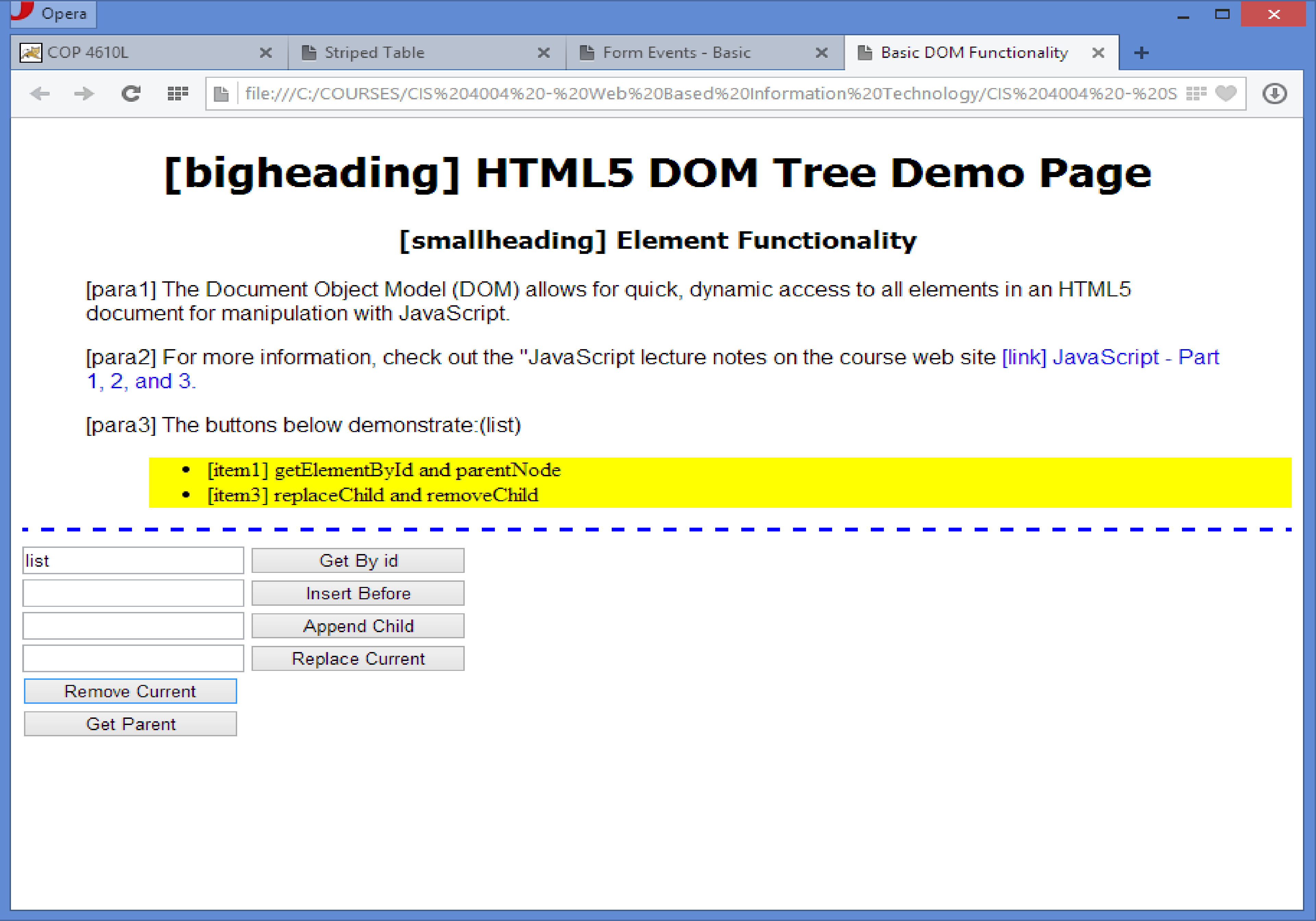
[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

item2

User selects item2 then clicks Remove Current button. HTML effect shown on next page.



Opera
COP 4610L
Striped Table
Form Events - Basic
Basic DOM Functionality
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20Technology/CIS%204004%20-%20S

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

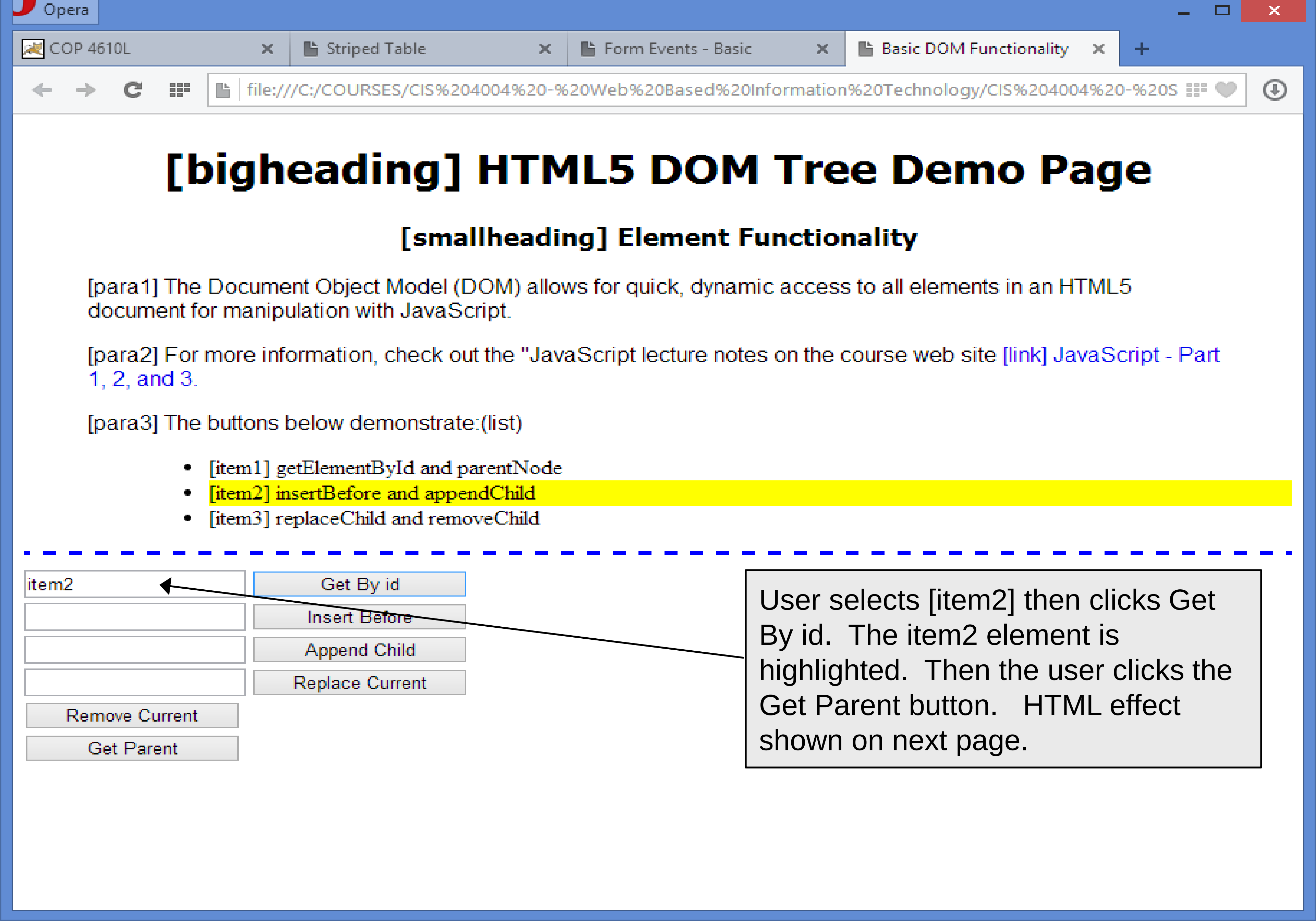
[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item3] replaceChild and removeChild

list

Determining the Parent Element

- The final piece of functionality in this DOM demo is the button that allows the user to identify the parent of the selected element.
- This is done by calling the `parent` function. This function simply gets the parent node, again making sure its not the body element since we will not allow selecting the entire body element.
- When the parent node is determined, the `switchTo` function is called to highlight the parent node.
- This sequence is illustrated by the next two slides.


Opera
COP 4610L
Striped Table
Form Events - Basic
Basic DOM Functionality
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20Technology/CIS%204004%20-%20S

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

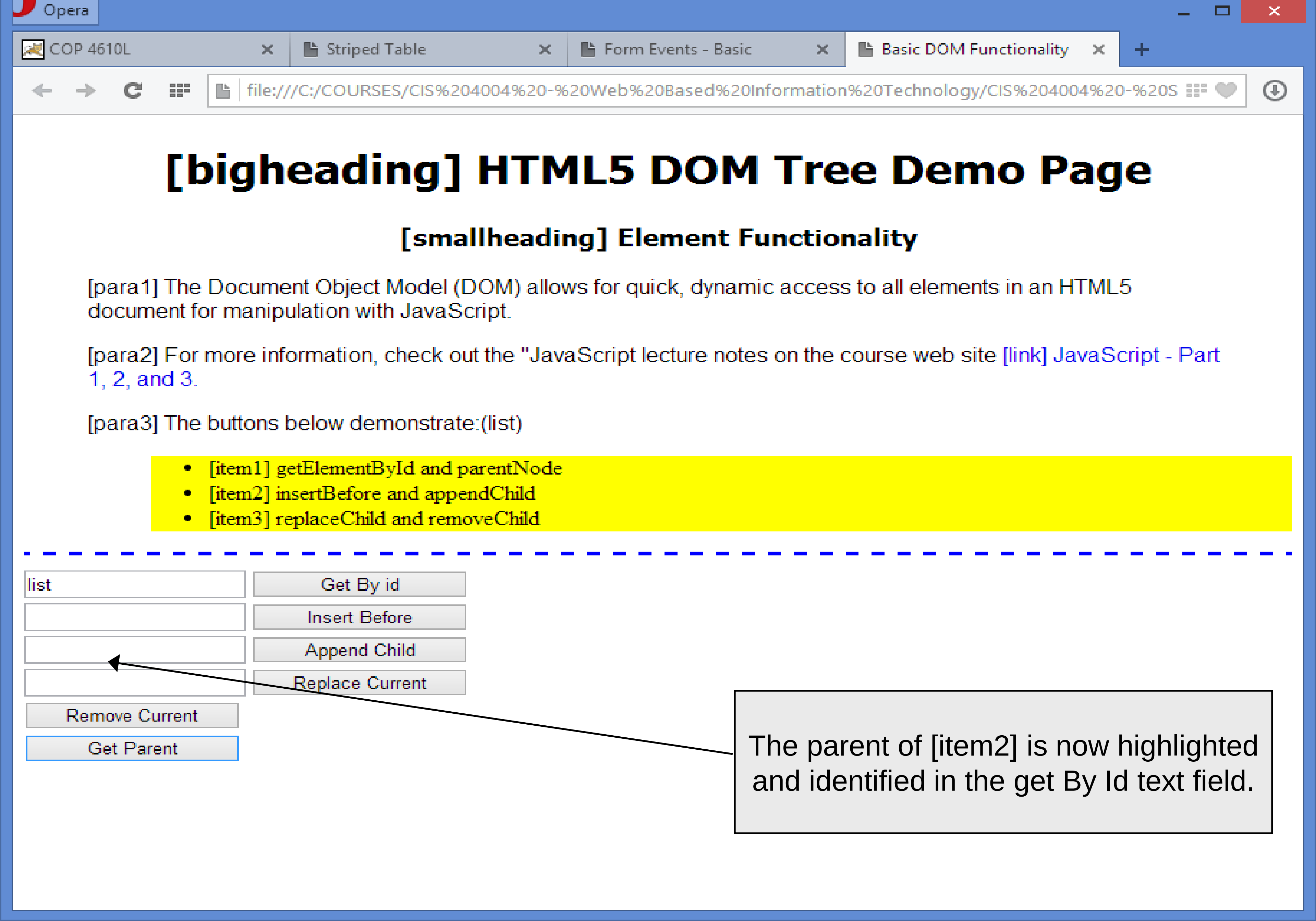
[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

item2

User selects [item2] then clicks Get By id. The item2 element is highlighted. Then the user clicks the Get Parent button. HTML effect shown on next page.


Opera
COP 4610L
Striped Table
Form Events - Basic
Basic DOM Functionality
file:///C:/COURSES/CIS%204004%20-%20Web%20Based%20Information%20Technology/CIS%204004%20-%20S

[bigheading] HTML5 DOM Tree Demo Page

[smallheading] Element Functionality

[para1] The Document Object Model (DOM) allows for quick, dynamic access to all elements in an HTML5 document for manipulation with JavaScript.

[para2] For more information, check out the "JavaScript lecture notes on the course web site [link] JavaScript - Part 1, 2, and 3.

[para3] The buttons below demonstrate:(list)

- [item1] getElementById and parentNode
- [item2] insertBefore and appendChild
- [item3] replaceChild and removeChild

list

The parent of [item2] is now highlighted and identified in the get By Id text field.