CIS 4004: Web Based Information Technology Spring 2013

Introduction To AJAX – Part 1

Instructor :   Dr. Mark Llewellyn
                      markl@cs.ucf.edu
                      HEC 236, 407-823-2790
          http://www.cs.ucf.edu/courses/cis4004/spr2013

Department of Electrical Engineering and Computer Science
University of Central Florida

AJAX

- Ajax is a catchy name that is given to a JavaScript programming technique that enables data to be moved between the browser and the server without the usual "round-trip to the server and a page refresh" – the only model by which the Web previously worked.

- In the previous model, even updating a single word in the page required that the server send an entirely new page to the browser. A new page was the only context in which new data could arrive in the browser.

- Ajax provides a new model. Using Ajax you can request data from the server and then use the DOM scripting techniques, with which you are now familiar, to add that new data into the page. This occurs without the user having to wait for an entire new page to load – in fact, without reloading the page at all.

AJAX

- This capability lets you deliver a more application-like experience to the user.

- If the user has a reasonably fast Internet connection, there is often little or no perceivable delay between clicking a link and seeing new data appear in the page. With Ajax, the response of your site is less of the old "click and wait" experience and much more like a regular application running on the user's local system. Things happen as soon as you click or select from a menu without the rest of the screen changing.

- A more subtle but powerful change that Ajax brings is a sense of place; instead of perceiving the site as a series of discrete pages, the experience is now a workspace that changes and updates as the user works.

AJAX – By The Letters

- Ajax is fairly simple to understand and not too difficult to implement.

- The first A of Ajax stands for Asynchronous. This means that there is no timing requirement for a communication transmission. In other words, the Ajax request that is made by the browser does not affect the browser's other activities.

- In the regular round-trip model, the user can do nothing after, say, submitting a form, except wait until a new page is served back to the browser. The process is entirely synchronous – one event must complete before the next event can start.

AJAX – By The Letters

- Ajax is asynchronous, once the request is sent off to the server, control is immediately restored to the user, who can continue working while the request is being fulfilled.

- When the requested data is delivered to the browser from the server, a pre-assigned callback function is automatically called and the data is then processed and displayed by JavaScript.

- The J in Ajax is for JavaScript. JavaScript handles the entire Ajax transaction. We'll be focusing on the JavaScript programming techniques that allow you to implement Ajax functionality in your web site.

- The second A in Ajax is simply And.

AJAX – By The Letters

- The X in Ajax stands for XML. This is somewhat misleading. In the original proposal, XML was the data format that would be returned from the server.

- However, data can be returned from the server in many formats, of which XML is one. For example, HTML, plain text, and JavaScript Object Notation (JSON) are all commonly requested data formats used in Ajax transactions. We'll see how to work with all of these in subsequent parts of the notes.

    – JSON is a data format based on the object literal construct and is a very compact data format that can be evaluated by JavaScript as code. As a result, JSON is frequently replacing XML as the data format for modern applications. However, Ajax doesn't sound as good as Ajax!

Setting Up A Server

- In order to utilize the benefits of Ajax, your web page will be interacting with the web server while the user manipulates your web page. Thus, we need a server to host the web page and to be able to respond to the user requests.

- The Apache HTTP server is a very popular server for this sort of application. Go to the Apache web site at http://apache.org and download and install any of the more recent versions of Apache compatible with your system. The most recent version is 2.4.4, but any of the versions in the 2.2.X family would be fine.

- During the installation process, set the port for Apache to listen on to 8080, or modify the `httpd.conf` file once it is installed.

Setting Up A Server

- When Apache is installed correctly and the server is running, enter `localhost:8080` (or use whichever port you have Apache listening on) and you should see the browser display as shown:

Note: I am running this Apache server on port 8081.
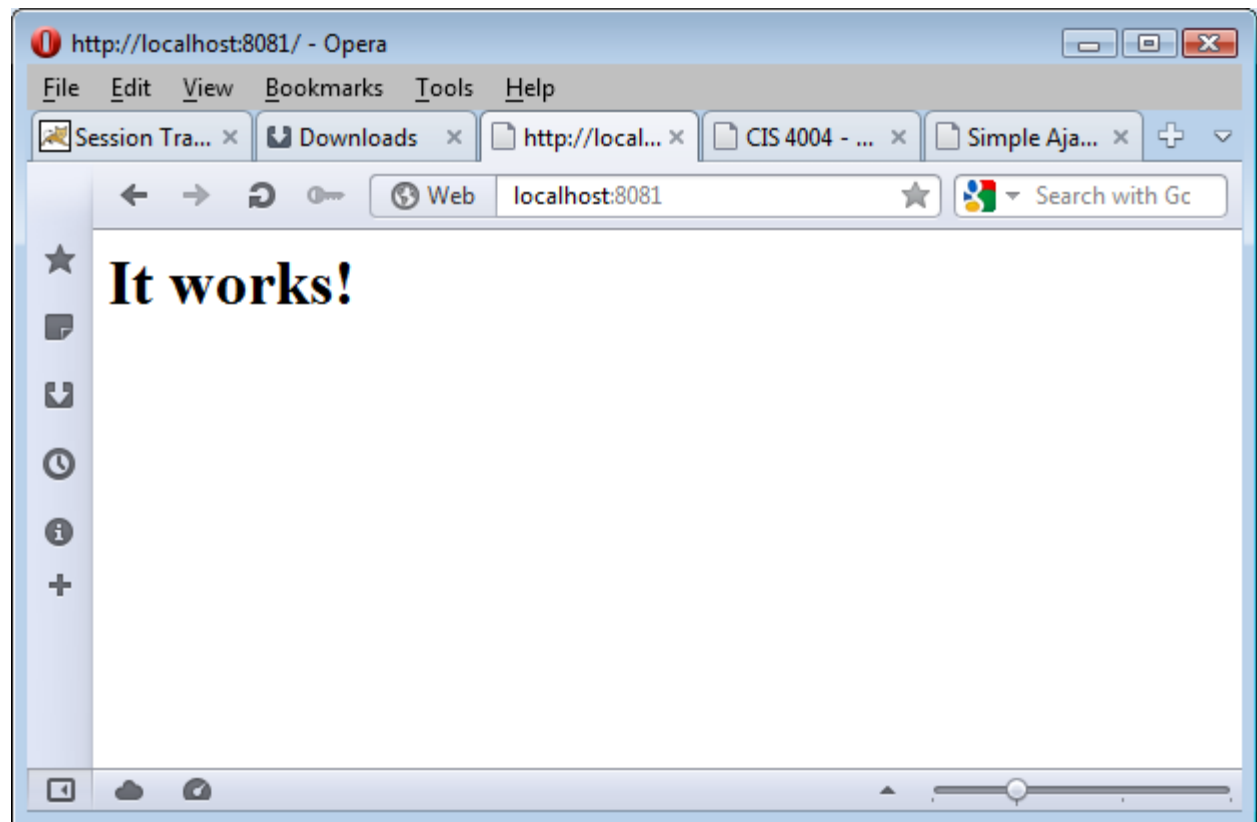
Communicating With The Server

- The Hypertext Transfer Protocol (HTTP) defines how transactions between the browser and server are handled. While the details of the protocol can get somewhat complicated, you really only need to understand the GET and POST methods of the protocol.

- In the scope of an HTTP request method, we mean *method* as in "a way of doing things" and not as in "a function in an object."

- A server request begins with an action that causes the browser to make either a GET or POST request to the server.

The GET Method

- A GET request, typically made when a link is clicked, is a URL with a query string, such as:

  `display_product_info.html?productID=34553&color=red`

- The query string (highlighted) is separated from the rest of the URL with a question mark. It consists of a number of name/value pairs separated by ampersands that are passed to the requested page. These name/value pairs serve the same purpose as the arguments passed to a function; they contain data that the requested page will use when processing the request.

The GET Method

- The entire GET request string is visible in the browser's address bar, and it's tempting for users to modify the query string to see what they might get (a different user's account , perhaps?)

- Thus, it's important to only use GET when the user is requesting non-sensitive information – a particular news story perhaps.

The POST Method

- A POST method is typically made when a form is submitted. If you look at the markup for a form, you'll notice that the `method` attribute is almost always POST.

- For each field of the form, the `name` attribute value and the data entered in that field are passed as the name/value pair.

- However, while the URL is visible in the browser's address bar, the name/value pairs are sent behind the scenes to the browser and are not visible in the address bar.

- As a rule of thumb, if you are sending data that will be stored on the server, use POST. It you are simply requesting a page, use GET.

The Traditional Model

- Under what we'll refer to as the traditional model – the only model prior to Ajax – the data is passed to the server via a POST or GET request and is then processed by the middleware (such as PHP, .NET, or Java). A new web page is then generated and served back to the browser in response to the request.

- The downside of the traditional model is that no user activity can take place between the request being submitted and the new page being entirely rendered in the browser. Once that link is clicked or the form submitted, the user must wait for that new page to display.

- Although a slow response can lead to frustration and random clicking on the part of the user, the synchronous nature of the traditional "click and wait" mode is very familiar and comfortable to the user.
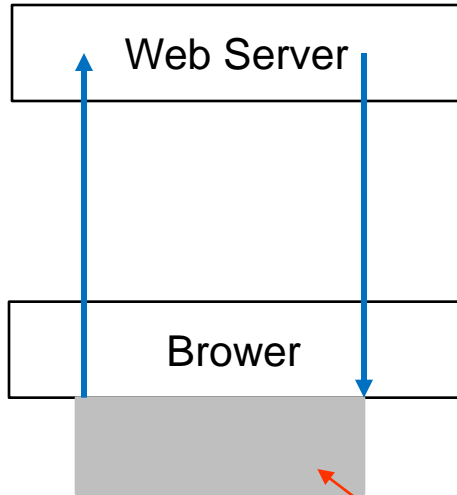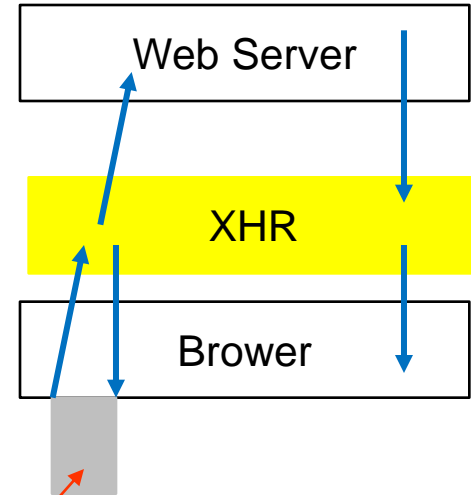
The Ajax Model

- Under the Ajax model, when a request is made to the server, again using a POST or GET request, it is made via the browser's XHR (**X**ML**H**ttp**R**equest) object.

- We'll look in more detail at the XHR object later, but it is the key to Ajax because it acts as an intermediary between the browser and the server.

- Actually initiating an Ajax request by calling the XHR object takes a matter of milliseconds, and then control is returned to the user while the XHR object fulfills the request.

- Because the request is now happening "in the background", the user can continue working as soon as the XHR object receives the request.

The Ajax Model

Traditional Round-trip Request

Ajax-enabled Request

Web Server

Brower

Web Server

XHR

Brower

Time elapsed until user regains control of the page

The Ajax Model

- The most significant advantage of using Ajax is that you can request specific data, not just an entire page, as a response.

- For example, the user could add an item to a shopping cart and, instead of waiting while the server is updating the cart status, the user could do more shopping and add a second item.

- Once the server update is complete, the server could return just the text needed to update the onscreen cart display, and this text would then be added into the page using DOM scripting without a refresh of the page.

- With data able to flow into the browser in this fashion and the necessity for constant page refresh removed, a web site can be transformed into a responsive online application. This is why Ajax has generated so much excitement among web developers.

The `XMLHttpRequest` Object

- Ajax is made possible because of the browser's `XMLHttpRequest` (XHR) object.

- The XHR object can make requests to the server and receive data in response. It was developed as part of Microsoft's Active X strategy to enable its online mail products to communicate with a mail server.

- It was only recently made part of the official W3C standard, even though it has been well supported in all the major browsers for several years.

- Microsoft's implementation of the XHR object is different from the implementation in W3C browsers, so like the event object that we've already dealt with, some cross-browser compatibility issues exist.

How to Use The XMLHttpRequest Object

- To use the XMLHttpRequest object, you must be able to communicate with it and then monitor its activity so you know when it has successfully acquired the requested data.

- The way you do this is to write a function (JavaScript) that can communicate with the browser's XHR object and can handle all your application's Ajax requests.

- This kind of a function is known as a wrapper. A wrapper servers as an interface to an object of usually complex functionality - in this case the XHR object – and manages all communications with it.

- Your code will talk to the wrapper function, and the wrapper function will be written to handle the complexities of managing the XHR object.

How to Use The `XMLHttpRequest` Object

- When you call the wrapper function, you will pass it two arguments:

  - The name of the requested resource – a filename that will provide the data you want from the server.

  - The name of your callback function – a function that will be called when the request completes. This function will receive the returned data and process it in some way according to your applications needs.

- An `XMLHttpRequest` wrapper function greatly simplifies your life because it abstracts away from the rest of your code all the complexities of managing an Ajax transaction and the associated cross-browser differences.

How to Use The `XMLHttpRequest` Object

- At any point where your application requires data from the server, you can just call the wrapper function, passing it the two required arguments.

- When the transaction is complete and the server returns the requested data, the wrapper function will pass the data to the specified callback function – the function you write to process the returned data.

- Once the callback function is called, the process is complete.

- Next, let's look at creating this wrapper function.

How to Use The `XMLHttpRequest` Object

- The wrapper function needs to accomplish five basic operations:

  - Create a new instance of the XHR object.

  - Define a function to monitor the request's progress.

  - Send the request via the XHR instance.

  - Check that the request was successful when the server responds.

  - Pass the returned data to the assigned callback function to be used by the application.

- We'll look at each of these steps individually, then put them all together into our wrapper function.

Creating The Wrapper Function: Step 1

- The first step in using the XHR object is to instantiate a new instance of it. In this example, we'll store it in a variable called `ajaxObj`.

- For a W3C browser, you would write:

```
ajaxObj = new XMLHttpRequest();
```

- For a Microsoft browser, you would write:

```
ajaxObj = new ActiveXObject("Microsoft.XMLHTTP");
```

- Written as a single cross-browser compatible statement we'd have:

```
var ajaxObj = (window.ActiveXObject)

  ? new XMLHttpRequest()

  : new ActiveXObject("Microsoft.XMLHTTP");
```

Creating The Wrapper Function: Step 2

- After you've instantiated the XHR object, you can then use its properties and methods to request and manage the movement of data between the browser and the server.

- A crucial component of the XHR process is the server's communication with the browser. Without this feedback from the server, you would never know when the request has completed.

- At key points in the process, the server updates the XHR object's `readystate` property with a numerical value that defines the current state of the request.

- There are five possible values for the current state. These are shown on the next page.

Creating The Wrapper Function: Step 2

Value	State
0	Uninitialized – the object exists but the `open` method has not been called.
1	Loading – the `open` method has been called but the `send` method has not.
2	Loaded – the `send` method has been called and the request is in process.
3	Interactive – the server is sending a response.
4	Complete – the response has been sent.

- The wrapper function can be notified each time this state changes by monitoring the `onreadystatechange` event handler, which as its name suggests is called each time the XHR object's `readystate` property is updated by the server.

Creating The Wrapper Function: Step 2

- Because the order of these responses is different between browsers, and because all you really need to know is when the request is completed, you simply want to monitor for a `readystate` property value of 4 each time the `onreadystatechange` event handler is triggered.

- You do this by assigning a function to the `onreadystatechange` event handler in which you can track the state of the request.

- You need to specify that function before making the request, because, as you can see from the descriptions of the values of the `readystate` property, the server starts sending back request state information even before the request is fully submitted.

Creating The Wrapper Function: Step 2

- As a result of all of this, after instantiating the XHR object, but before you do anything with it, you want to define the function that will process the `onreadystatechange` information.

- The XHR object also tracks the three-digit HTTP status of the request. You are probably all too familiar with HTTP-status 404 – Page Not Found, which you get when you request a URL that doesn't point to a valid resource.

- What you want your code to do is to check for HTTP-status 200 – success.

- Upon determining a status of 200, your object can safely assume the request has arrived and pass whatever is in the response to the callback function.

Creating The Wrapper Function: Step 2

- In the sequence of your code, once the `readystate` is 4, you check for the request's HTTP status (which is stored in the XHR object's `status` property).

- If the status is 200, the `responseText` property's value – the requested data – is passed to the callback function.

- This is normally done by assigning an anonymous function (a function without a name) that makes these checks to the `onreadystatechange` event handler each time the `readystate` changes.

  – Note: This is a very similar concept to the `onload` event handler, because both are called by events that are not initiated by the user.

Creating The Wrapper Function: Step 2

- The code for this would look like:

```
maAjaxObj.onreadystatechange = function {

    if (ajaxObj.readystate == 4 &&

        ajaxObj.status == 200)

    {      cbFunc(ajaxObj.responseText);

    }//endif

}//end monitor readystate
```

- With the monitoring now in place for the request, we're all set to make the actual request for the data that we want from the server.

Creating The Wrapper Function: Step 3

- The first decision you must make when requesting data from the server is the method to use to make the request – GET or POST.

- Basically, if you just want to get data from the server, use the request method GET. If you want to update data on the server, use the request method POST.

- We'll look at a GET method request first:

- A GET request requires the use of two methods of the XHR object: `open` and `send`. The `open` method allows you to specify the kind of request method (GET or POST) you want to make, the name of the file you are requesting, and a Boolean value that defines whether you are making an asynchronous or synchronous request.

Creating The Wrapper Function: Step 3

- The format of this statement is:

  ```
  objName.open(requestMethod, URL, asynchronous?);
  ```

- If you set the third value to false, the application will stop running until the request is fulfilled (synchronous behavior), which defeats one of the key benefits of Ajax.

- Almost always, you'll set the third parameter to true (asynchronous mode) so that the user can continue working while the request is being fulfilled.

- The name of the file is simply a URL and since we're using a GET, can be extended with a query string of name/value pairs, such as:

  ```
  myAjaxObj.open=('GET', 'lookUpInfo.php?username=Mark, true);
  ```

Creating The Wrapper Function: Step 3

- Now that the request is defined, you can send the request to the server suing the `send` method.

- The format of this statement is:

  `myAjaxObj.send(null);`

- When using a GET method, the data argument of the send method is always set to null; you don't need to include any data with the send.

- If you want to send information to further define your request, you can append a query string to the URL, as in the previous example.

Creating The Wrapper Function: Step 3

- When sending a request with the POST method, two additional steps are required.

- First you set the request method to POST.

- Second you don't append a query string to the URL as you do with a GET; any data that is part of the request is sent separately from the URL as the argument of the `send` method.

- To do this, the content type of the HTTP header must be set correctly using the XHR object's `setRequestHeader` method, so that this data is handled correctly when it arrives at the server.

- The `open` and `send` steps for a POST request are shown on the next page.

Creating The Wrapper Function: Step 3

```
myAjaxObj.open = ('POST', 'updateInfo.php', true);

myAjaxObj.setRequestHeader('Content-Type','application/x-www-form-urlencoded');

myAjaxObj.send('oldemail=mark@cs.edu&newemail=markl@cs.ucf.edu');
```

Creating The Wrapper Function: Step 4

- Regardless of whether you use a GET or POST to make your request, your Ajax function now needs to keep track of the request by monitoring the `onreadystatechange` event handler using the anonymous function assigned to it in step 2.

- Once you get a `readystate` of 4 and a `status` of 200, the request data has arrived.

Creating The Wrapper Function: Step 5

- When the callback function is called, it is passed the `responseText` property value as its argument, which is the requested data, and the Ajax wrapper function's work is complete.

- The complete wrapper function that we've created in these five steps is shown in its entirety on the next page.

> Note: All of the server-side files that you need for an Ajax application that is running on an Apache server will reside in folders inside the `htdocs` folder of Apache. As you can see on the next page, I've created a folder named `ajax_request` inside of the Apache `htdocs` folder.
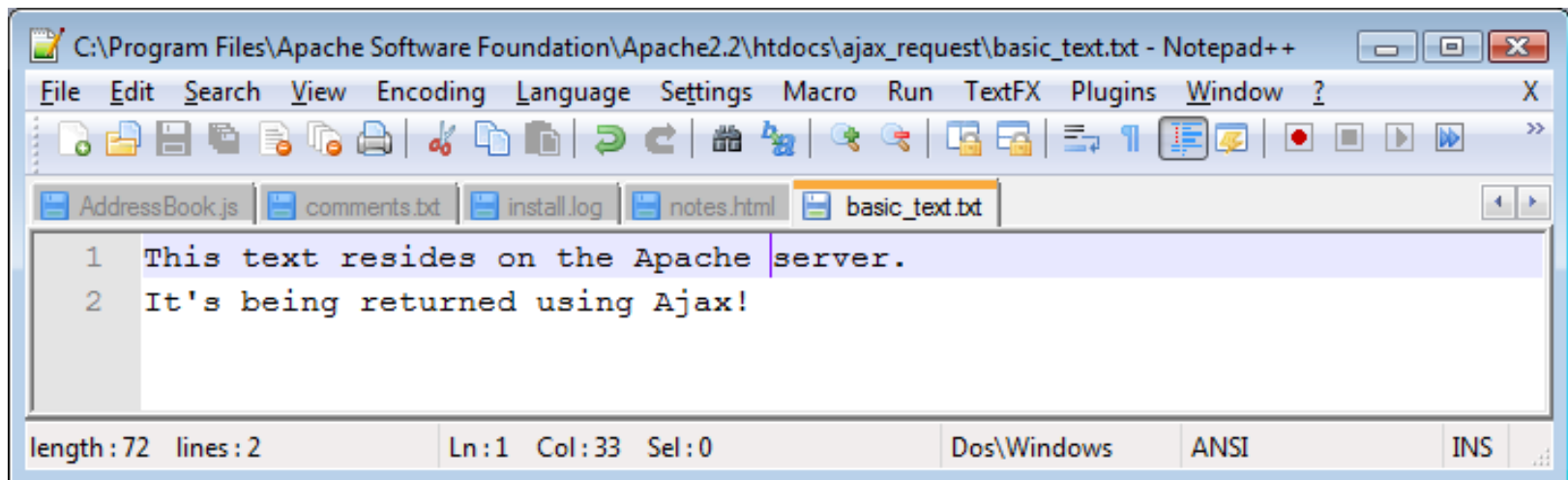
File   Edit   Search   View   Encoding   Language   Settings   Macro   Run   Plugins   Window   ?   X

PS-Part5-p15-signedV2.ps1 | config.php | ajax_request.html | basic_text.txt | ajax_wrapper.js

```javascript
1     //A simple Ajax request function
2   function ajaxRequest(url, cbFunc) {   //parameters: the url of the data, the callback function
3        if (document.getElementById) { // does browser support this property?
4        // if so, determine which browser and create a new ajaxObj obj
5            var ajaxObj = (window.ActiveXObject) ? new ActiveXObject("Microsoft.XMLHTTP")
6                                                 : new XMLHttpRequest();
7        }
8        if (ajaxObj) { // if the ajaxObj object was successfully created?
9            ajaxObj.onreadystatechange = function() { // set up function to run whenever readyState changes, before making
10               if (ajaxObj.readyState == 4 && ajaxObj.status == 200) { // if true, then the request data has arrived
11                   cbFunc(ajaxObj.responseText); // pass the data to the callback function
12               }
13           }
14       ajaxObj.open("GET", url, true); // open server connection - GET request, requested URL, Asychronous set to true
15       ajaxObj.send(null); // send the request
16       }
```

JavaScript file   length : 967   lines : 17   Ln : 8   Col : 70   Sel : 0   Dos\Windows   ANSI   INS

Using The Ajax Wrapper Function

- Next, we'll use the Ajax function to request some text from a file called `basic_text.txt` on the server.

- The Ajax function will return the file's text as a string that we'll display in our browser page.

- A simple file request such as this requires no server-side middleware (no PHP for example) – the file simply has to be at the specified location on the server.

Using The Ajax Wrapper Function

- Shown on the next page is part of the markup that includes the link which calls a function to request the data and an empty element into which We'll add the requested text when it is received from the server. (The complete files are on the course website for you to try.)

- In order to clearly establish the logical connection between the calling function and its associated callback function, I use the same basic name for both functions except for that the callback function name is prefixed with "`cb`" (for callback).

- The `readFile` function called by the link passes the Ajax function the two required arguments – the URL of the file and the name of the callback function that will display the data when it is returned.

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

notes.html   basic_text.txt   ajax_request_scope_problem.html   ajax_request.html

```
23
24    // use the Ajax request function
25    // call to Ajax function
26  □ function readFile() { // makes the call to the ajaxObj
27        ajaxRequest('basic_text.txt',cbReadFile);
28        }
29    // callback function
30  □ function cbReadFile(theData) { // process response from Ajax function
31        var theDisplay = document.getElementById('display');
32        theDisplay.innerHTML = theData;
33        }
34
35    </script>
36    </head>
37
38  □<body>
39  □<div>
40        <h3>My First Ajax Demo</h3>
41        <a href="basic_text.txt" onclick="readFile(); return false;">Get text</a>
42        <p id="display"></p>
43    </div>
44    </body>
```
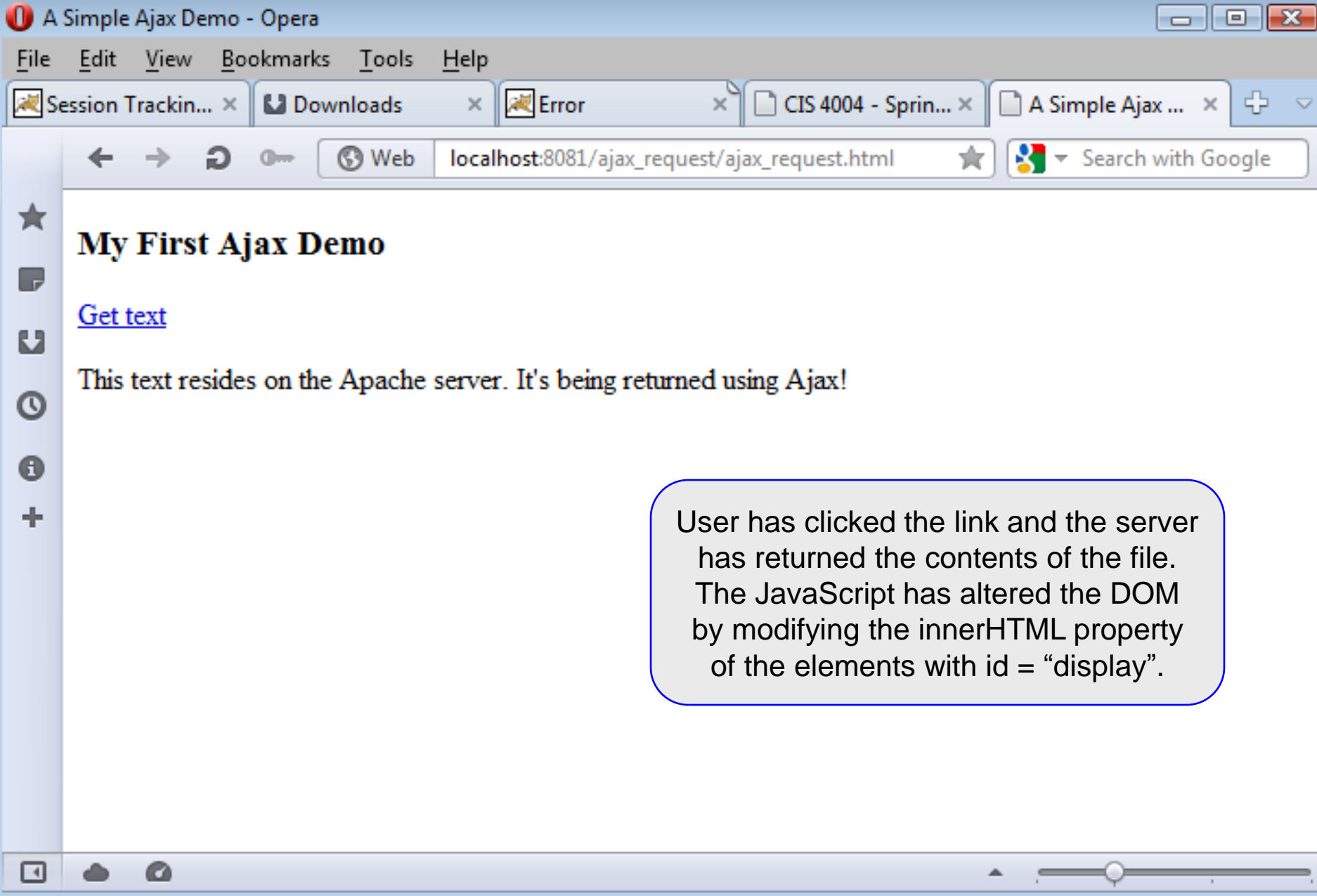
Hyper Text Markup Language file   length : 1564   lines : 46   Ln : 1   Col : 1   Sel : 0   UNIX   ANSI   INS

A Simple Ajax Demo - Opera

File Edit View Bookmarks Tools Help

Session Trackin... ✕ | Downloads ✕ | Error ✕ | CIS 4004 - Sprin... ✕ | A Simple Ajax ... ✕

localhost:8081/ajax_request/ajax_request.html
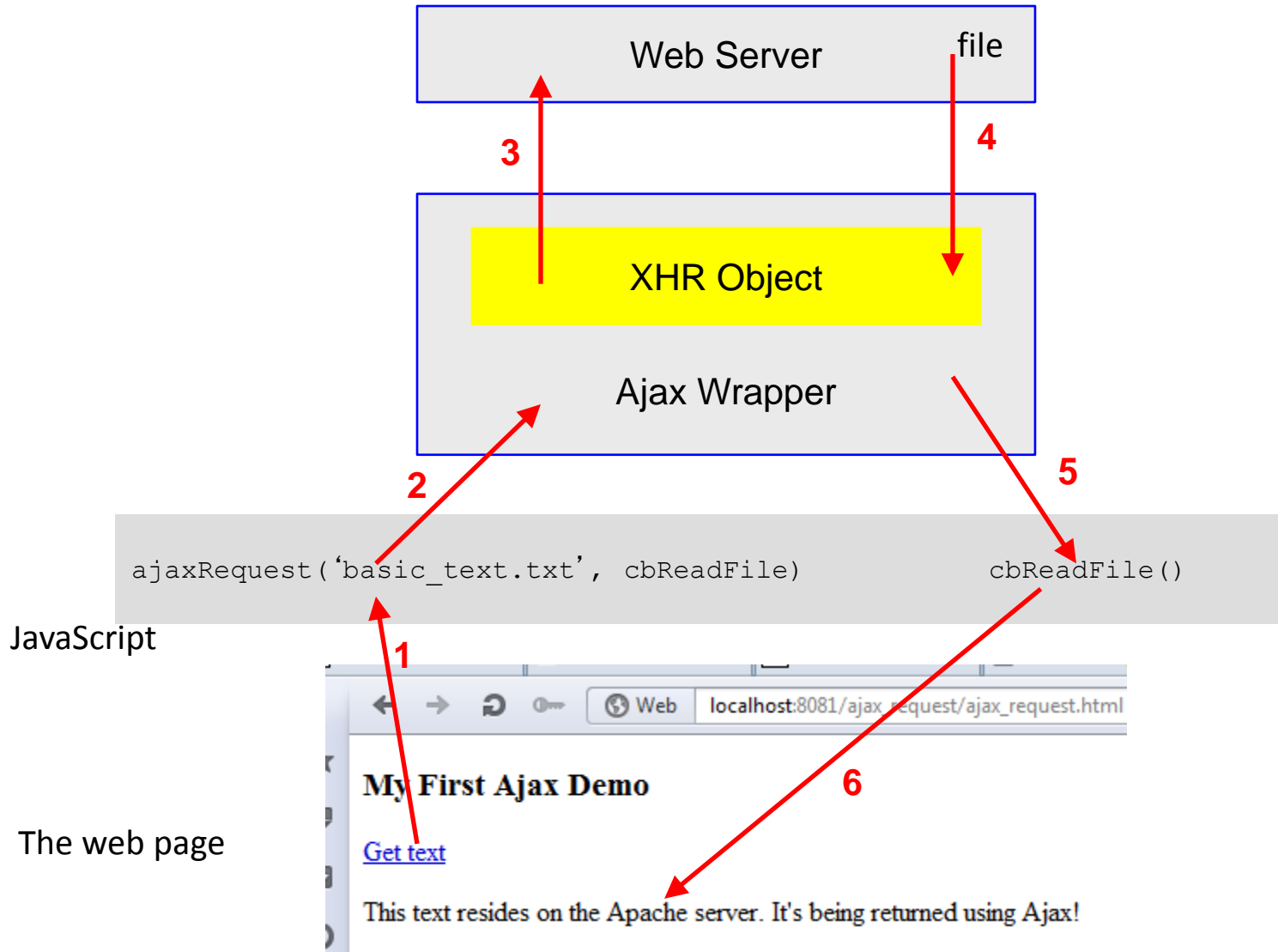
Search with Google

My First Ajax Demo

Get text

This text resides on the Apache server. It's being returned using Ajax!

User has clicked the link and the server has returned the contents of the file. The JavaScript has altered the DOM by modifying the innerHTML property of the elements with id = "display".

How The Request Is Processed

Web Server   file

3   **4**

XHR Object

Ajax Wrapper

2   **5**

```
ajaxRequest('basic_text.txt', cbReadFile)                cbReadFile()
```

JavaScript

1

The web page

localhost:8081/ajax_request/ajax_request.html

My First Ajax Demo

Get text

6

This text resides on the Apache server. It's being returned using Ajax!
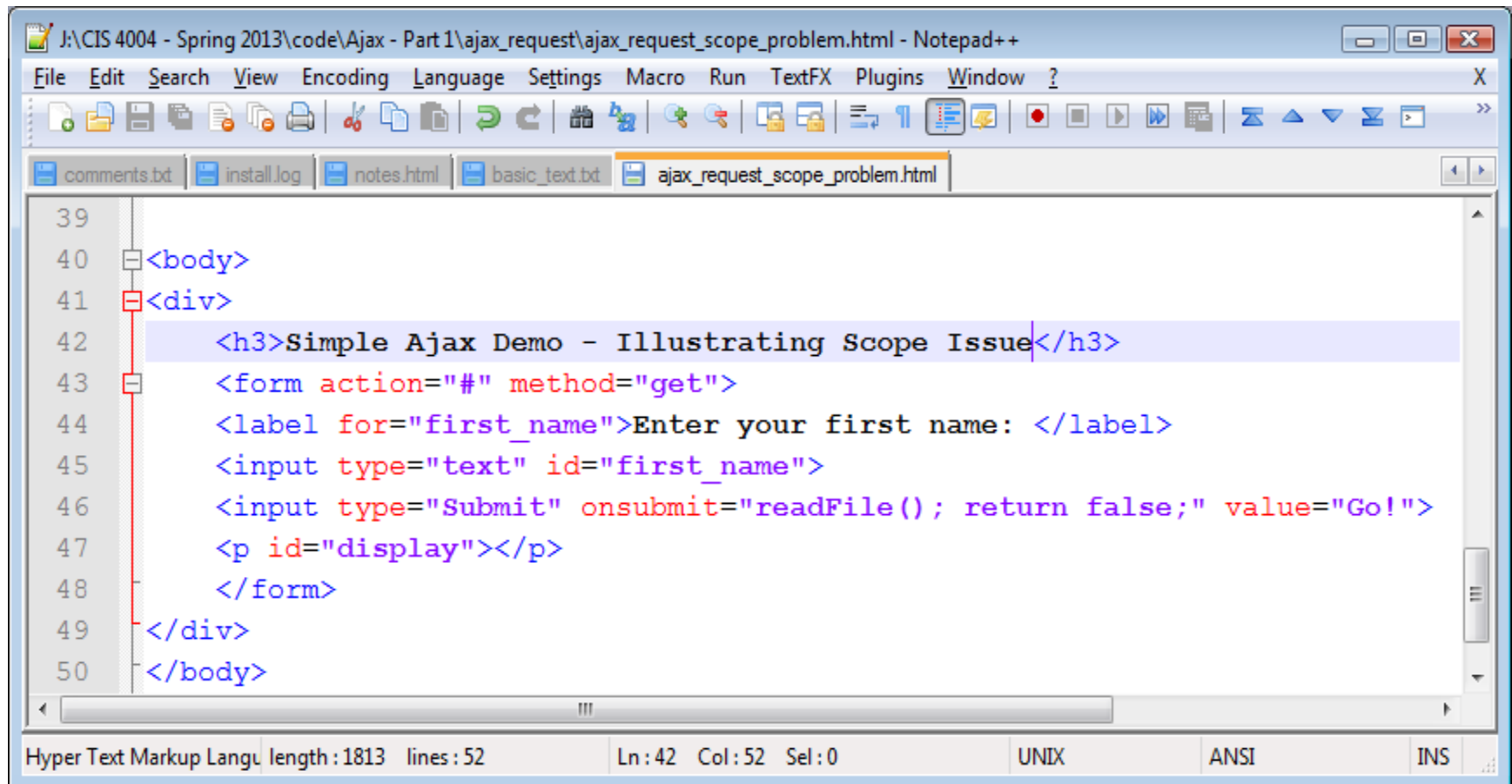
Using An Object Literal To Maintain State

- There is a problem with using two functions for an Ajax call – one to make the call and the other to handle the callback.

- The problem is that you are passing control from one function to the other. However, because that control goes via the Ajax function, you can't at that time pass data from the calling function to the callback function, because one does not directly call the other.

- The variables that existed in the calling function are not accessible to the callback function.

- This is a problem of maintaining state in your application – where important values must persist for later reference.

- The following example will illustrate this concept.

Using An Object Literal To Maintain State

- In this example, we'll get a user's name from a form field so that we can display a personalized welcome message. The markup is shown below:

```
39
40    <body>
41    <div>
42        <h3>Simple Ajax Demo - Illustrating Scope Issue</h3>
43        <form action="#" method="get">
44        <label for="first_name">Enter your first name: </label>
45        <input type="text" id="first_name">
46        <input type="Submit" onsubmit="readFile(); return false;" value="Go!">
47        <p id="display"></p>
48        </form>
49    </div>
50    </body>
```

Using An Object Literal To Maintain State

- Notice in the markup that I'm using the `onsubmit` as the event for the Submit button. Unlike `onclick`, `onsubmit` can also be triggered from the keyboard if the button has focus.

- Also, I've left the form action value as # - an anchor link, and not supplied a URL. Normally, and more correctly, you would put a URL here to the server side script that would process the data in the form if JavaScript were not available. I omitted this here so you won't confuse the point of this example.

- Once, the user's name has been obtained, we'll fetch the welcome message from the server and add the user's name to it.

- I've modified the calling function and the callback functions as shown on the next page.

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

comments.txt | install.log | notes.html | basic_text.txt | **ajax_request_scope_problem.html** | ajax_request_scope_problem.html
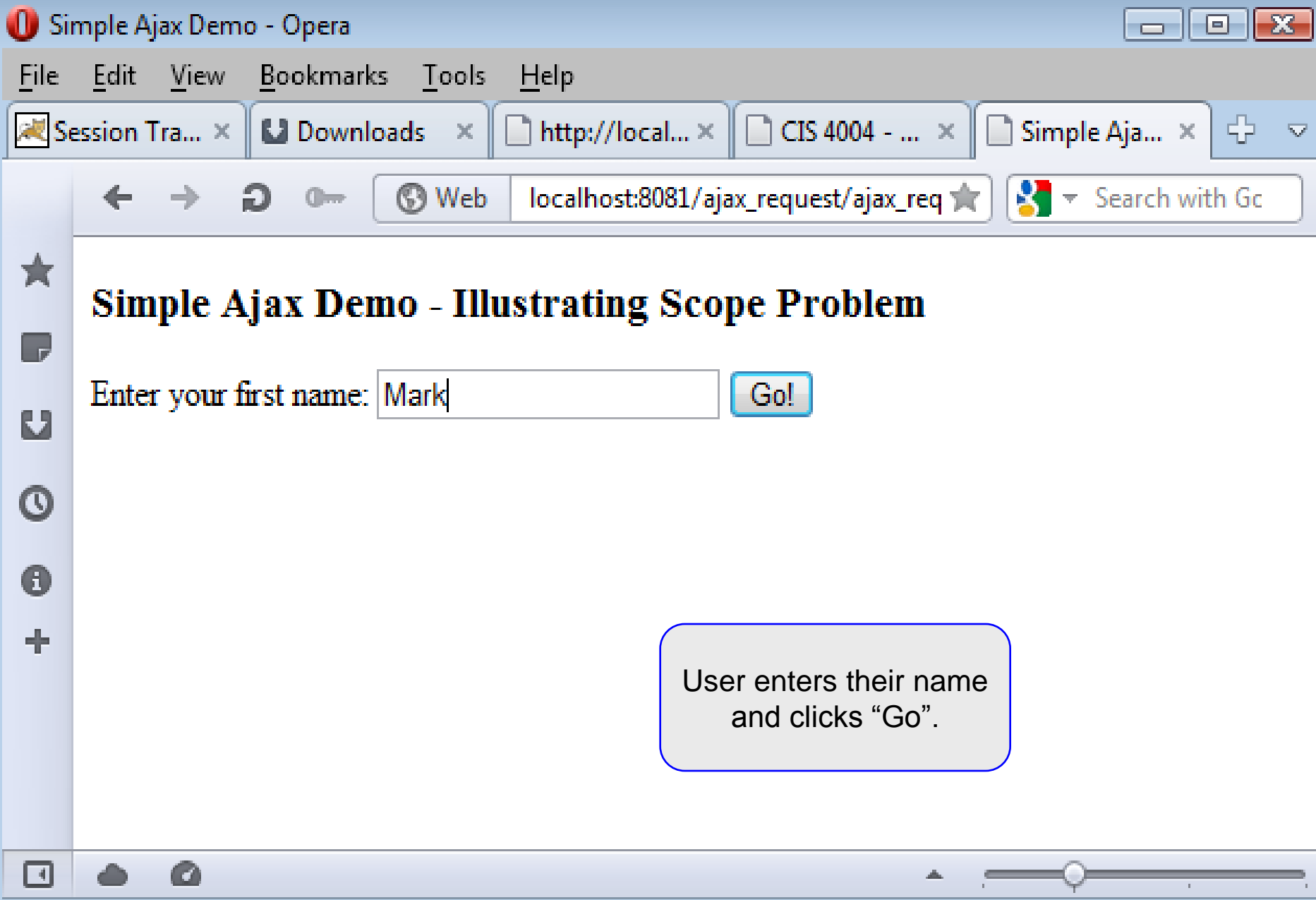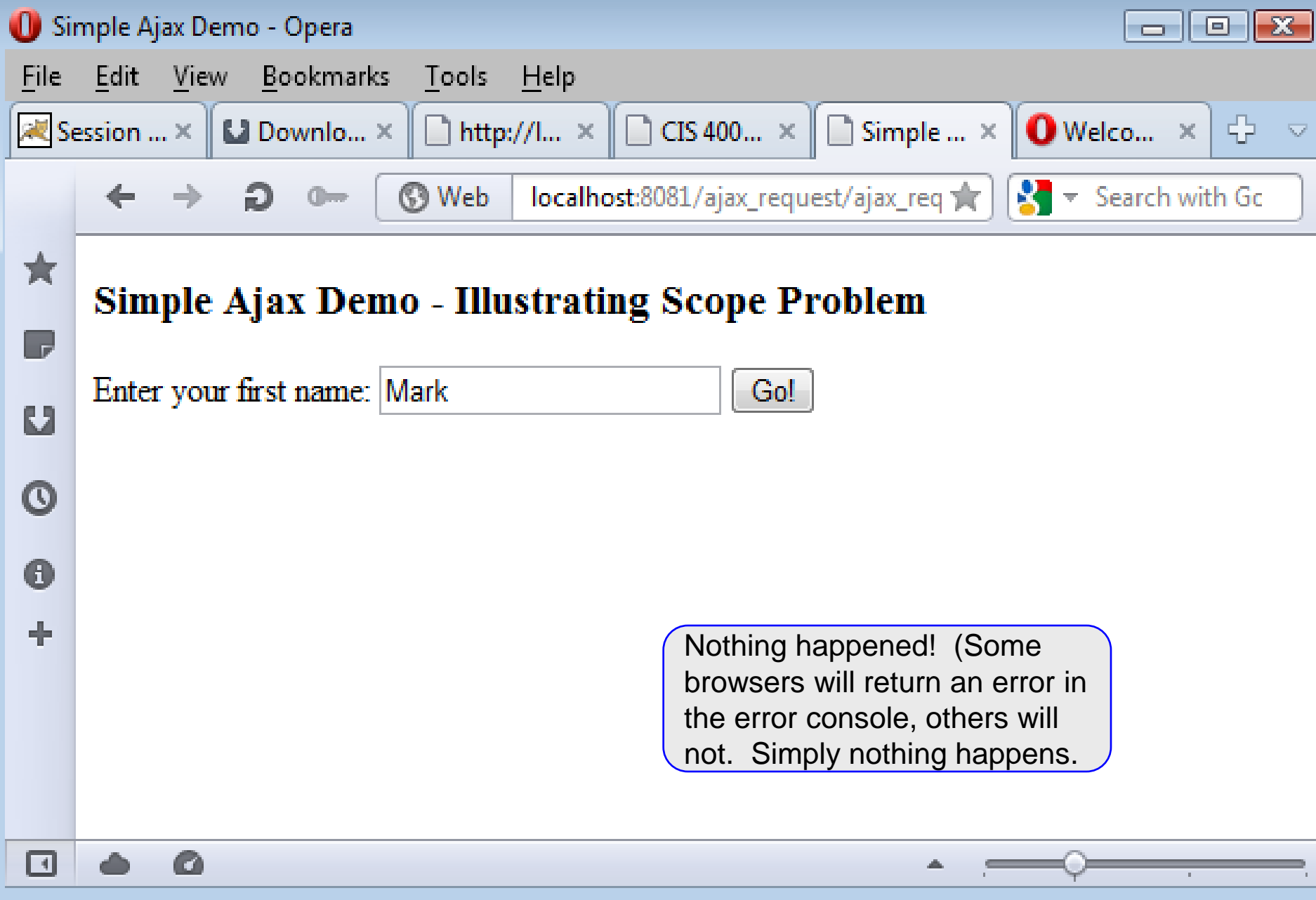
```
26      // call to Ajax function
27    function readFile() {  // makes the call to the ajaxObj
28         var firstName = (document.getElementById("first_name").value)
29         ajaxRequest('basic_text.txt',cbReadFile);
30         }
31    // callback function
32    function cbReadFile(theData) {  // process response from Ajax function
33         var theDisplay = document.getElementById('display');
34         theDisplay.innerHTML = "Hi, " + firstName + ". " + theData;
35         }
36
```

Hyper Text Markup Langu length : 1813   lines : 52   Ln : 42   Col : 52   Sel : 0   UNIX   ANSI   INS

Simple Ajax Demo - Illustrating Scope Problem

Enter your first name: Mark   [Go!]

User enters their name and clicks "Go".

Using An Object Literal To Maintain State

- The solution to this scoping problem is to refactor the code involving the two functions as an object literal where both the call and callback functions share the same scope (the object).

- The refactored code is shown on the next page:

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

notes.html | basic_text.txt | ajax_request_scope_problem.html | ajax_request.html | ajax_request_obj_literal.html

```
25
26      // use the Ajax request function
27      // call to Ajax function
28    ajaxFile = {
29          readFile: function () {  // makes the call to the ajaxObj
30          ajaxFile.firstName = (document.getElementById("first_name").value) || 'Visitor';
31          ajaxRequest('basic_text.txt',ajaxFile.cbReadFile);
32          },
33      // callback function
34          cbReadFile: function (theData) {  // process response from Ajax function
35          var theDisplay = document.getElementById('display');
36          theDisplay.innerHTML = "Hi, " + ajaxFile.firstName + ". " + theData;    //
37          }
38    }
39    </script>
40    </head>
41
```

Hyper Text Markup Language file   length : 2010   lines : 54   Ln : 1   Col : 1   Sel : 0   UNIX   ANSI   INS
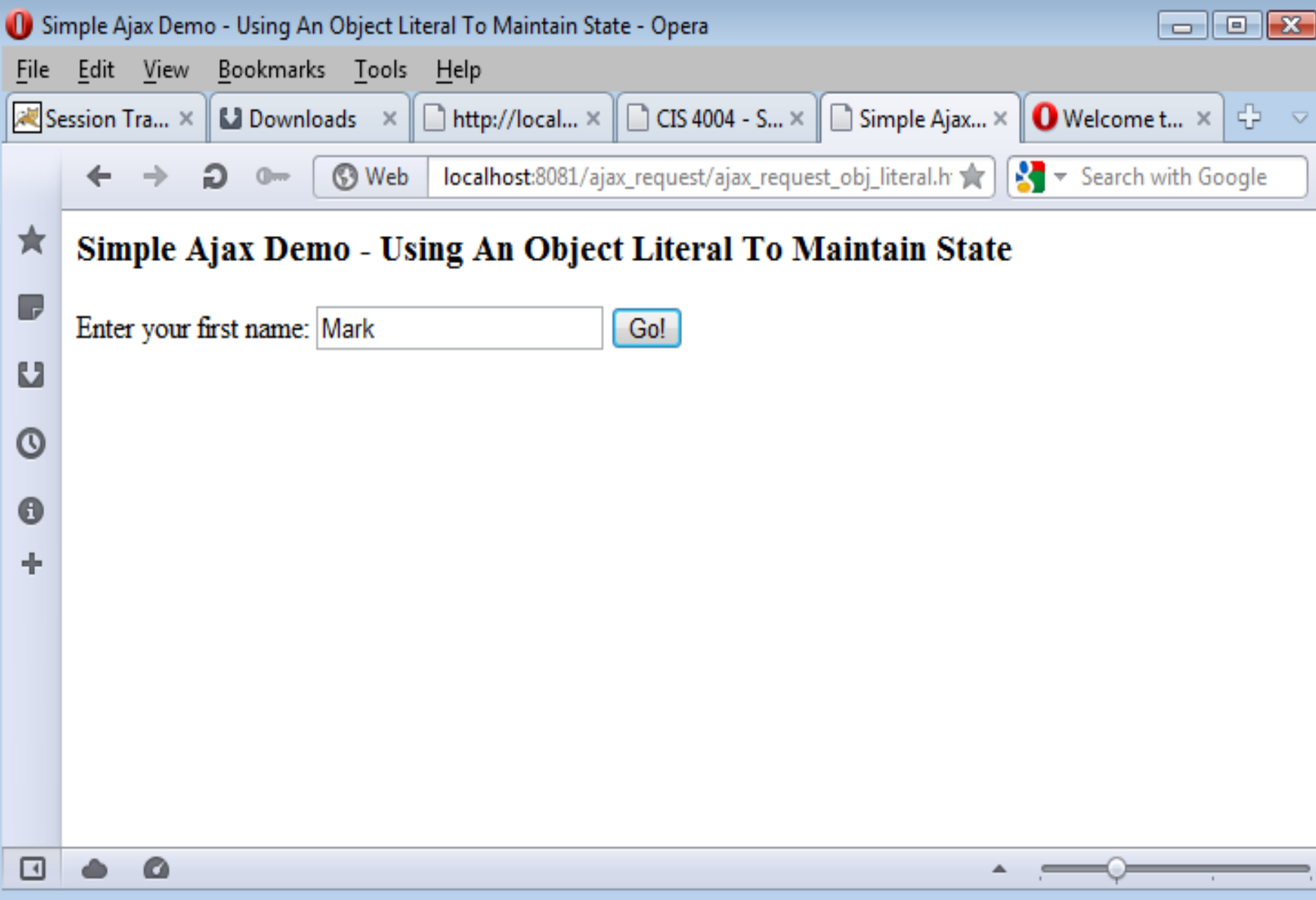
Simple Ajax Demo - Using An Object Literal To Maintain State - Opera

File   Edit   View   Bookmarks   Tools   Help

Session Tra... × | Downloads × | http://local... × | CIS 4004 - S... × | Simple Ajax... × | Welcome t... ×

localhost:8081/ajax_request/ajax_request_obj_literal.h   Search with Google

Simple Ajax Demo - Using An Object Literal To Maintain State

Enter your first name: Mark   Go!

Hi, Mark. This text resides on the Apache server. It's being returned using Ajax!

File   Edit   View   Bookmarks   Tools   Help

Session Tra... ✕ | Downloads ✕ | http://local... ✕ | CIS 4004 - S... ✕ | Simple Ajax... ✕ | Welcome t... ✕ | ➕

localhost:8081/ajax_request/ajax_request_obj_literal.h ⭐ | Search with Google

Simple Ajax Demo - Using An Object Literal To Maintain State

Enter your first name: [   ] [Go!]

Hi, Visitor. This text resides on the Apache server. It's being returned using Ajax!

> User clicked the "go" button without entering their name. The code uses the optional/default "Visitor" value in this case.