Debugging JavaScript

Introduction

In this presentation we will look at techniques for debugging JavaScript code. We'll look at how to use the **`alert`** and **`document.write`** methods to display debugging information. We'll also take a look at the error console. Finally, we'll see an introduction to the Firebug debugging extension that is available for the Firefox web browser.

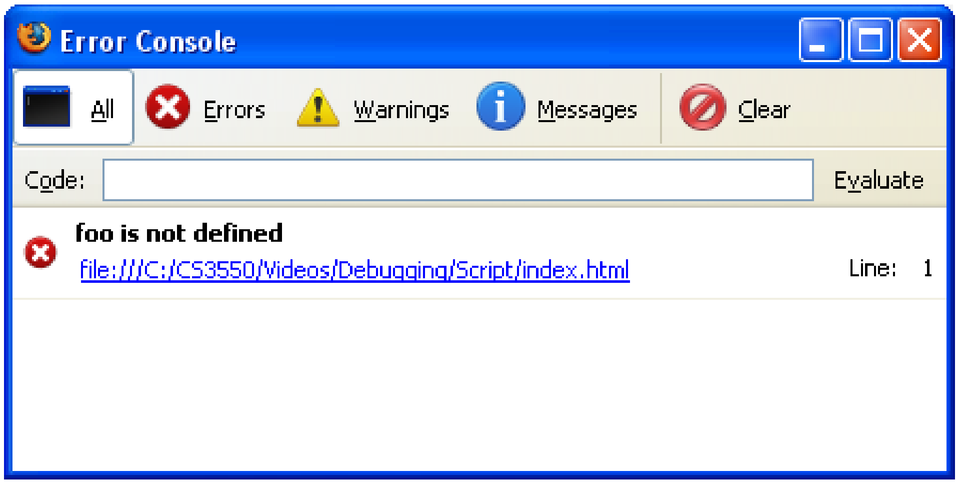
Chrome and Safari have built-in debugging tools that offer features that are similar to what's available in Firebug. To use the debugging tools in Chrome, go to the **View** menu, choose **Developer**, and then choose **Developer Tools** from the submenu. To use the debugging tools in Safari, you will need to change the preferences in Safari to display the **Develop** menu:
Go to the **Safari** menu and choose **Preferences**.
In the Preferences window, click the **Advanced** tab.
Click to put a check in the checkbox next to **Show Develop menu**.

The error console

The simplest debugging tool to use is the error console that some web browsers provide. If your JavaScript code doesn't do what you expect it to do, open the error console and see if there are error messages. In Firefox, look for the error console in the Tools menu.

Microsoft Internet Explorer doesn't have the same kind of console as Firefox, but you can look for script errors in the status bar.

Here is an example of an error message which resulted when I changed **`document.write`** to **`document.foo`**:



foo is not defined
file:///C:/CS3550/Videos/Debugging/Script/index.html Line: 1

The **error** button below generates a JavaScript error (calling the function **`foo`**, which is undefined), so you can see how your browser shows error information.
error

The `alert` method

The simplest debugging technique is to use the alert method, which will show a dialog box with debugging information. For example, if you want to see what the value of the variable **x** is at a certain point in your program, you can use a JavaScript statement like this:

```
alert("x is: " + x);
```

Click this button to see an alert: alert

Although alerts are easy to write and use, they don't work well when you need to see more than two or three values. For example, if you have a loop in your program that executes 100 times, you wouldn't want to click on an OK button 100 times, which is what you'd have to do if you use an alert in the body of the loop. Also, once you click on the OK button there's no way to go back and look at the information again.

The `document.write()` method

An alternative to using an alert dialog box is to use document.write to write debugging information to the body of the HTML document. To see what the value of the variable x is you could use a statement like this:

```
document.write("x is: " + x);
```

x is: 5

With document.write, you don't have to click on the OK button, and your debugging information stays on the screen. However, document.write is only called when the document is loaded, so if you are using JavaScript to modify the page after it has been loaded document.write will be useful. That is a major disadvantage of using document.write for debugging, especially for the kinds of web applications we will making in this class. A minor disadvantage of document.write is that your debugging information might be interspersed with the normal content of the document and so both the normal content and the debugging information might be difficult to read.

The `innerHTML` property

The innerHTML property can be used to modify the contents of an HTML page after the page has been loaded. You can create a div (or other suitable HTML element) on your page that is specifically for debugging and write code that adds debugging information to the div where you can easily read it. When you are finished debugging the code you can remove the div.

Although the innerHTML property is not standard, most browsers support it, so this debugging technique is very useful.

Click on the **test** button below to see how the contents of the div (with the yellow background) change.
test

Debug information will replace this text.

We haven't yet learned enough about client-side JavaScript to look at the code used for this kind of debugging, but in a few weeks we will see how to use the innerHTML property for debugging and for web applications in general.

The Firebug script debugger

The next debugging technique that we will look at is a script debugger called Firebug. Firebug can be used with the Firefox browser. Another version of Firebug, called Firebug Lite, can be used with other browsers.

You can install Firebug by going to the Firebug web site at http://getfirebug.com and clicking on the "INSTALL FIREBUG" button. Next, click on the green "Add to Firefox" button. Firefox will then ask you to verify that you want to install the add-on.

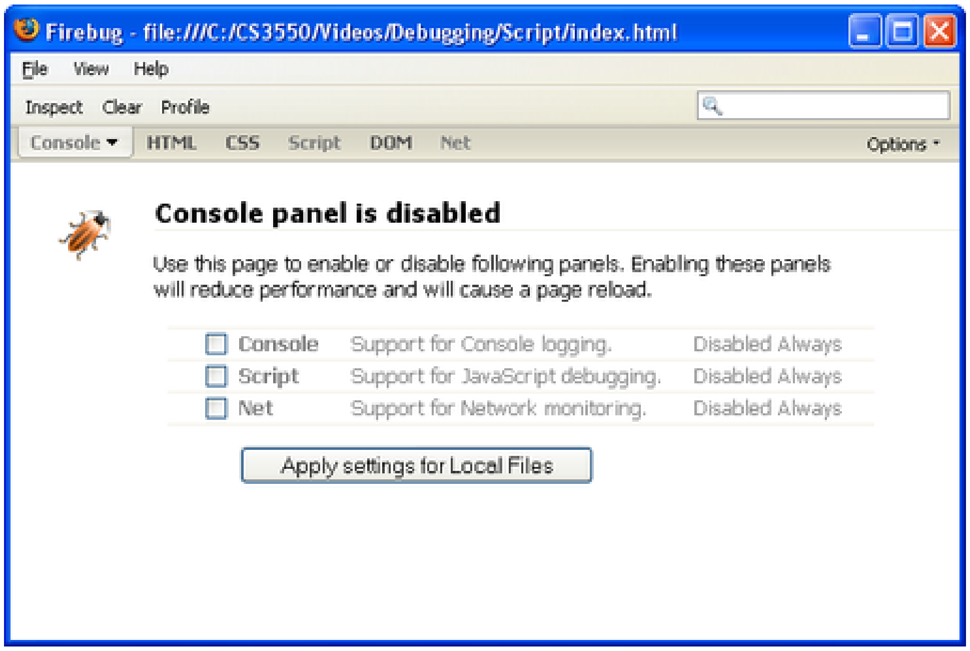
Firebug Lite is a little more complicated to work with since it is JavaScript code that must be added to the web page that has the scripts you want to debug. To use Firebug Lite, go to http://getfirebug.com/lite.html. There you will see a script tag that you need to insert in a page that you want to contain Firebug Lite. Another option is the Firebug Lite bookmarklet which is available on the same web page. Bookmarklets are an easy way of attaching JavaScript code to any web page.

Using Firebug

A video presentation of this section is available here.

To use Firebug, open the **Tools** menu in Firefox, select **Firebug**, and then choose **Open Firebug** or **Open Firebug in New Window**

When you first open Firebug, the Console, Script, and Net panels will be disabled and you will see a window like this:


Firebug - file:///C:/CS3550/Videos/Debugging/Script/index.html
File View Help
Inspect Clear Profile
Console HTML CSS Script DOM Net Options

Console panel is disabled

Use this page to enable or disable following panels. Enabling these panels will reduce performance and will cause a page reload.

☐ Console	Support for Console logging.	Disabled Always
☐ Script	Support for JavaScript debugging.	Disabled Always
☐ Net	Support for Network monitoring.	Disabled Always

Apply settings for Local Files

For this demonstration I will use the Console panel and the Script panel, so I will enable them by clicking on the check boxes and then clicking on the button that says "Apply settings for Local Files". When I open a Firebug window for an HTML document from a different site (other than local files) I will again need to enable the panels that I want to use.

For sample code to demonstrate Firebug, I will use a simple script that uses **`document.write`** to draw boxes like this in an HTML document:

```
***********
***********
***********
***********
```

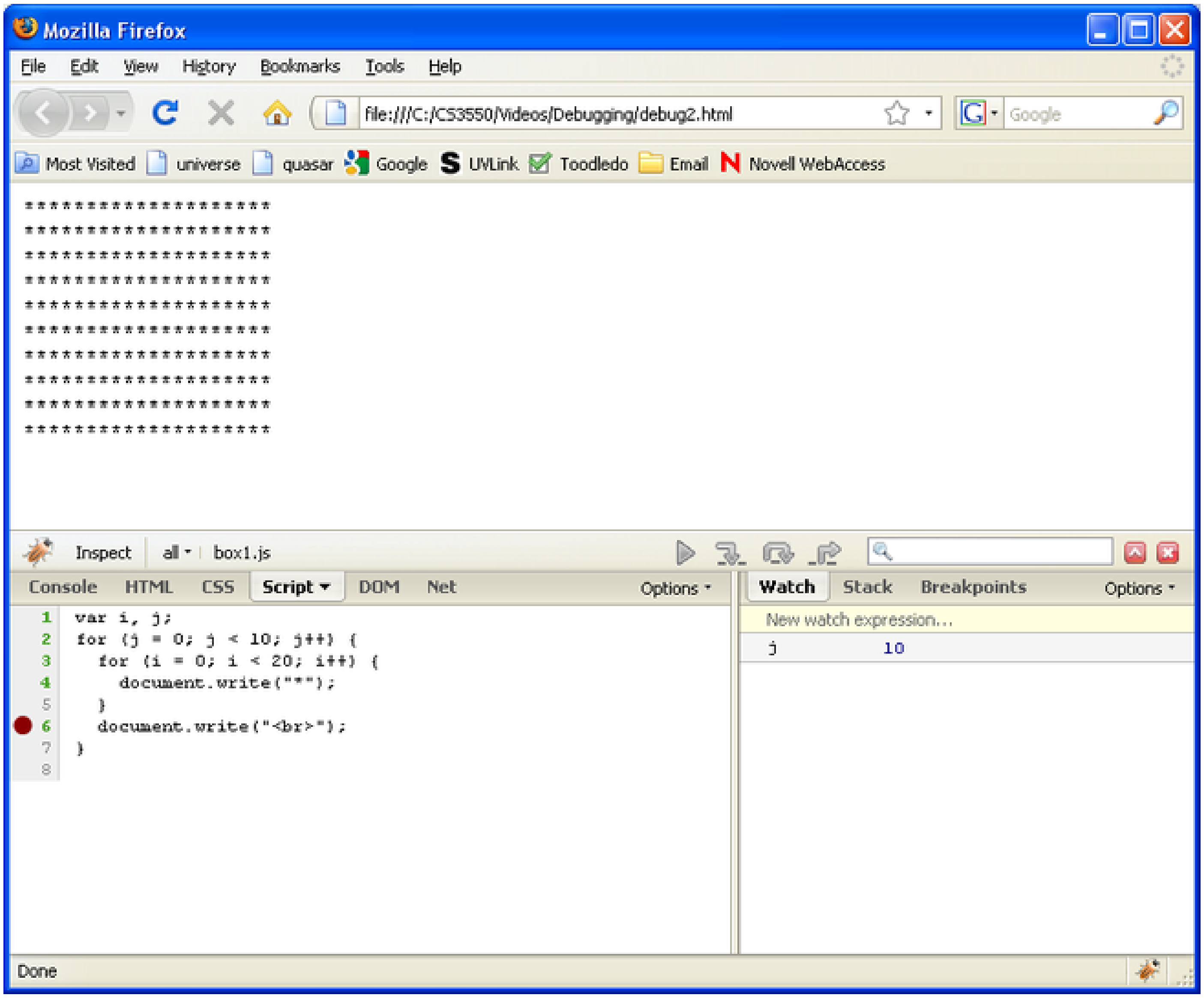
Let's look at a script that draws a rectangle 20 stars across and 10 stars across. Here's the code:

```
var i, j;
for (j = 0; j < 10; j++) {
   for (i = 0; i < 20; i++) {
      document.write("*");
   }
   document.write("<br>");
}
```

In Firebug I will set a breakpoint on the second call to **`document.write`**. I will also put a watch on the variable **`j`**. Then I will step through the code and see the value of **`j`** change each time through the outer loop.

To set a breakpoint, I first need to select the file with the JavaScript code I am debugging. At the top of the Firebug panel (or window) is a drop-down menu which can be used to select the file. In this case, the file is **`box1.js`**, so I select that file from the menu and my JavaScript code is displayed.

Now I can click in the area to the left of the line numbers to set a breakpoint. Breakpoints are shown by red circles. To set a watch on a variable, I can click on the **Watch** tab in the right-hand pane. Then I click on **New watch expression...** and enter **j** to create a watch on the variable **`j`**.

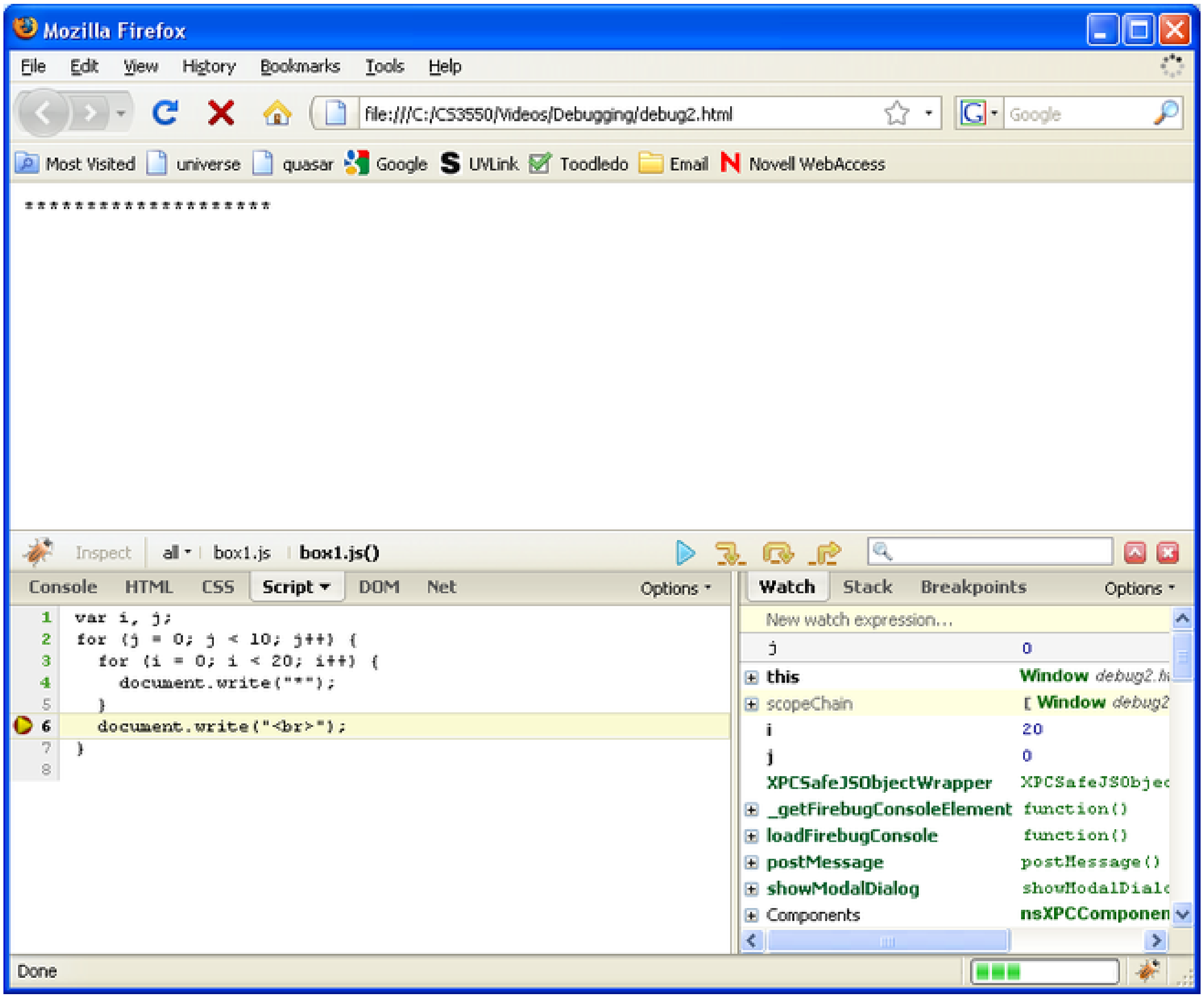

Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/CS3550/Videos/Debugging/debug2.html
Most Visited universe quasar Google UVLink Toodledo Email Novell WebAccess
Inspect all box1.js
Console HTML CSS Script DOM Net Options
Watch Stack Breakpoints Options
New watch expression...
Done

```
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
```

```
var i, j;
for (j = 0; j < 10; j++) {
  for (i = 0; i < 20; i++) {
    document.write("*");
  }
  document.write("<br>");
}
```

```
j          10
```

I'm ready to debug the program, so I use the Reload button to execute the JavaScript code. The first line of stars is added to the document window, and then execution stops at the line with a breakpoint. Firebug draws a yellow triangle over the breakpoint circle to show that that is where execution has stopped. In the pane at the right, I can see the current value of **`j`**, which is zero.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/CS3550/Videos/Debugging/debug2.html
Most Visited universe quasar Google UVLink Toodledo Email Novell WebAccess
Inspect all box1.js box1.js()
Console HTML CSS Script DOM Net Options
Watch Stack Breakpoints Options
Done

```
********************
```

```
var i, j;
for (j = 0; j < 10; j++) {
  for (i = 0; i < 20; i++) {
    document.write("*");
  }
  document.write("<br>");
}
```

```
New watch expression...
j                              0
this                           Window debug2.h
scopeChain                     [ Window debug2
i                              20
j                              0
XPCSafeJSObjectWrapper         XPCSafeJSObjec
_getFirebugConsoleElement      function()
loadFirebugConsole             function()
postMessage                    postMessage()
showModalDialog                showModalDial
Components                     nsXPCComponen
```

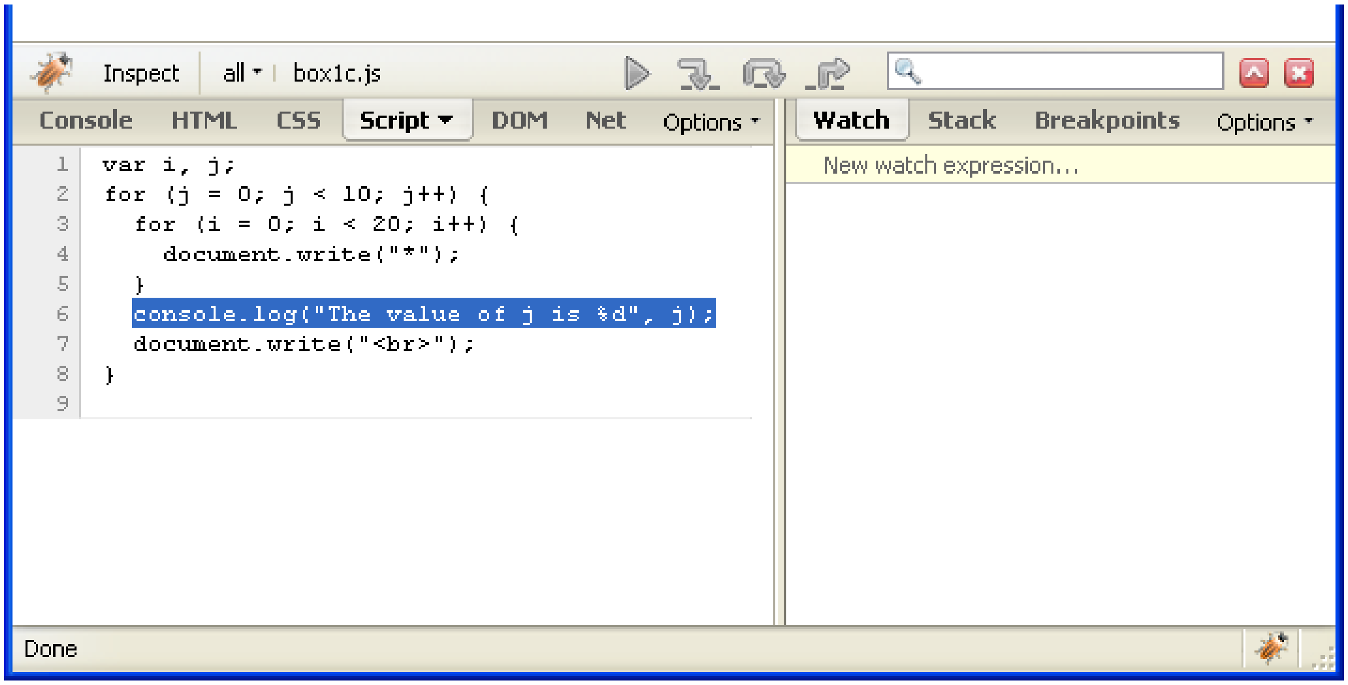
When I want to resume execution I can use the bottons at the top of the Firebug panel to continue execution (the blue triangle button), step into a function call, step over a function call, or step out of a function call.

The Firebug Console

Now let's look at the Firebug console. I can add a statement like this to print out the values of **j**:

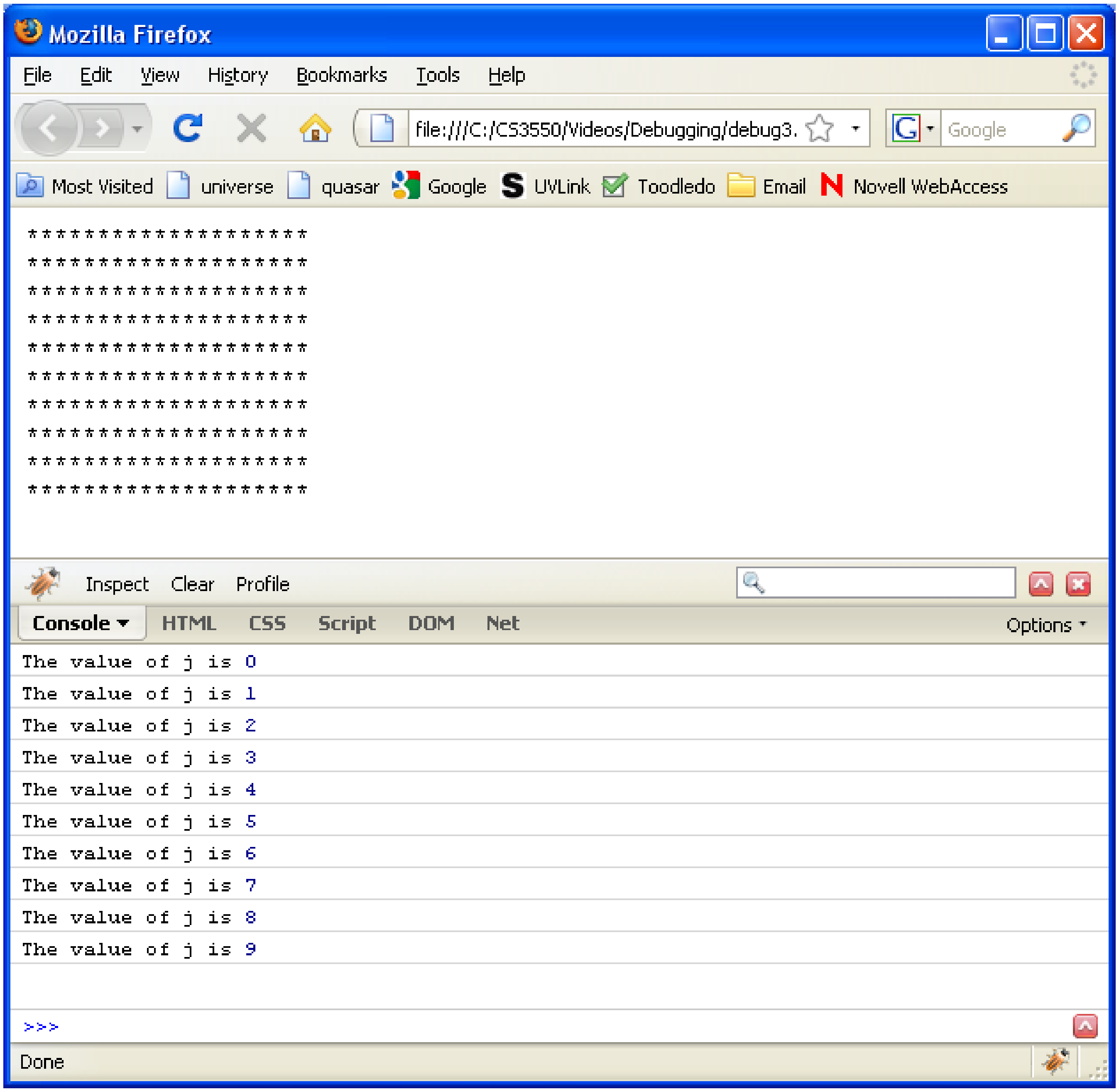
```
console.log("The value of j is %d", j);
```

Here's what it looks like in Firebug:



```
var i, j;
for (j = 0; j < 10; j++) {
  for (i = 0; i < 20; i++) {
    document.write("*");
  }
  console.log("The value of j is %d", j);
  document.write("<br>");
}
```

Here's the output:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/CS3550/Videos/Debugging/debug3.
Google
Most Visited universe quasar Google UVLink Toodledo Email Novell WebAccess
Inspect Clear Profile
Console HTML CSS Script DOM Net Options
Done

```
********************
********************
********************
********************
********************
********************
********************
********************
********************
********************
```

```
The value of j is 0
The value of j is 1
The value of j is 2
The value of j is 3
The value of j is 4
The value of j is 5
The value of j is 6
The value of j is 7
The value of j is 8
The value of j is 9

>>>
```

Summary

These are the JavaScript debugging techniques that we've seen:

- Using the web browser's error consle
- Using the **`alert`** method
- Writing debugging information to the web page using the **`document.write()`** method
- Writing debugging information to a div (or other HTML element) using the **`innerHTML`** property
- Using the Firebug script debugger, including breakpoints, watches, and printing to the console