



# Troubleshooting, Debugging, and Cross-Browser Issues

Bok, Jong Soon  
[jongsoon.bok@gmail.com](mailto:jongsoon.bok@gmail.com)  
<https://github.com/swacademy>

# Simple Ways to Debug

- Is to use some form of output functionality to check the values of variables after processing.  
i.e. `alert(someVariable) ;`  
`document.write(someVariable) ;`
- If interested in a quick check, don't want to open a debugger, add the `alert` box, check the values you need to check, and then immediately remove the alert box.

# Simple Ways to Debug (Cont.)

```
<script type="text/javascript">
function sumNumbers(numArray) {
    var result = 0;
    if (numArray.length > 0) {
        for (var i = 0; i < numArray.length; i++) {
            if (typeof numArray[i] == "number") {
                result += numArray[i];
            } else {
                result = NaN;
                break;
            }
        }
    } else {
        result = NaN;
    }
    return result;
}
var ary = new Array(1,15,"three",5,5);
var res = sumNumbers(ary); // res is NaN
if (isNaN(res)) alert("Encountered a bad array or array element");
</script>
```

# Development and Debugging Tools by Browser

- Google Chrome Developer Tools
  - <https://developers.google.com/chrome-developer-tools/>
- Firefox Firebug
  - <http://getfirebug.com/faq/>
- IE Built-in Debugger
  - [http://msdn.microsoft.com/en-us/library/ie/gg699336\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/ie/gg699336(v=vs.85).aspx)
- Opera Dragonfly
  - <http://www.opera.com/dragonfly/>
- Safari Web Development Tools
  - <https://developer.apple.com/technologies/safari/developer-tools.html>

# Sample Code to Debugging

```
2 <html>
3   <head>
4     <title> JavaScript Debugging Example </title>
5     <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
6     <script type="text/javascript">
7       var display;
8       function init() {
9         display = document.getElementById("results");
10      }
11      function firstParam() {
12        //set breakpoint here
13        var a = 5;
14        secondParam(a);
15      }
16      function secondParam(a) {
17        var b = 10;
18        thirdParam(a, b);
19      }
20      function thirdParam(a, b) {
21        var c = 15;
22        var d = a + b + c;
23        if (window.console && window.console.log) {
24          window.console.log(a + " + " + b + " + " + c + " = " + d);
25        } else {
26          display.innerHTML = a + " + " + b + " + " + c + " = " + d;
27        }
28      }
29    </script>
30  </head>
31  <body onload="init()">
32    <p><button onclick="firstParam();">Run</button></p>
33    <div id="results"></div>
34  </body>
35 </html>
```

# Sample Code to Debugging (Cont.)

```
2 <html>
3   <head>
4     <title> New Document </title>
5     <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
6     <script type="text/javascript">
7       function hello(){
8         var msg = "Hello, World!";
9         console.log("%s", msg);
10      }
11    </script>
12  </head>
13  <body onload="hello()">
14    <p>Hi</p>
15  </body>
16 </html>
```

# Chrome Developer Tools

- Select the Wrench menu  or  at the top-right of your browser window, then select Tools -> Developer tools.
- Right-click on any page element and select Inspect element.



# Chrome Developer Tools (Cont.)

## Resources

- <http://www.youtube.com/watch?v=nOEw9iiopwl>
- <http://www.youtube.com/watch?v=htZAU7FM7GI>

# Chrome Developer Tools (Cont.)

- On Windows and Linux, press
  - **Control - Shift - I** keys to open Developer Tools
  - **Control - Shift - J** to open Developer Tools and bring focus to the Console.
  - **Control - Shift - C** to toggle Inspect Element mode.
- On Mac, press
  - **⌘I (Command - Option - I)** keys to open Developer Tools
  - **⌘J (Command - Option - J)** to open Developer Tools and bring focus to the Console.
  - **⌃C (Control - Option - C)** to toggle Inspect Element mode.

# Chrome Developer Tools: Keyboard Shortcuts

- <https://developers.google.com/chrome-developer-tools/docs/shortcuts>

The screenshot shows a web browser window with the Google Developers homepage. The navigation bar includes links for Home, Products (which is the active tab), Events, Showcase, Live, and Groups. The main content area is titled "Chrome Developer Tools" and features a section titled "Chrome Developer Tools: Keyboard Shortcuts". A sidebar on the left contains a navigation menu with items like Overview, Documentation (with sub-options for Elements Panel, Resources Panel, Network Panel, Scripts Panel, Timeline Panel, Profiles Panel, Console, and Keyboard Shortcuts), Remote Debugging, Resources, and Contributing.

## Chrome Developer Tools: Keyboard Shortcuts

To access the developer tools, open a web page or web app in Google Chrome. Then take one of the following actions:

- Select the Wrench menu at the top-right of your browser window, then select Tools -> Developer tools.
- Right-click on any page element and select Inspect element.
- On Windows and Linux, press
  - Control - Shift - I keys to open Developer Tools
  - Control - Shift - J to open Developer Tools and bring focus to the Console.
  - Control - Shift - C to toggle Inspect Element mode.
- On Mac, press
  - ⌘I (Command - Option - I) keys to open Developer Tools
  - ⌘J (Command - Option - J) to open Developer Tools and bring focus to the Console.
  - ⌘C (Command - Shift - C) to toggle Inspect Element mode.

Once in the Developer Tools window, hit ? to see the list of supported shortcuts:

[Global](#)

# Chrome Developer Tools (Cont.)

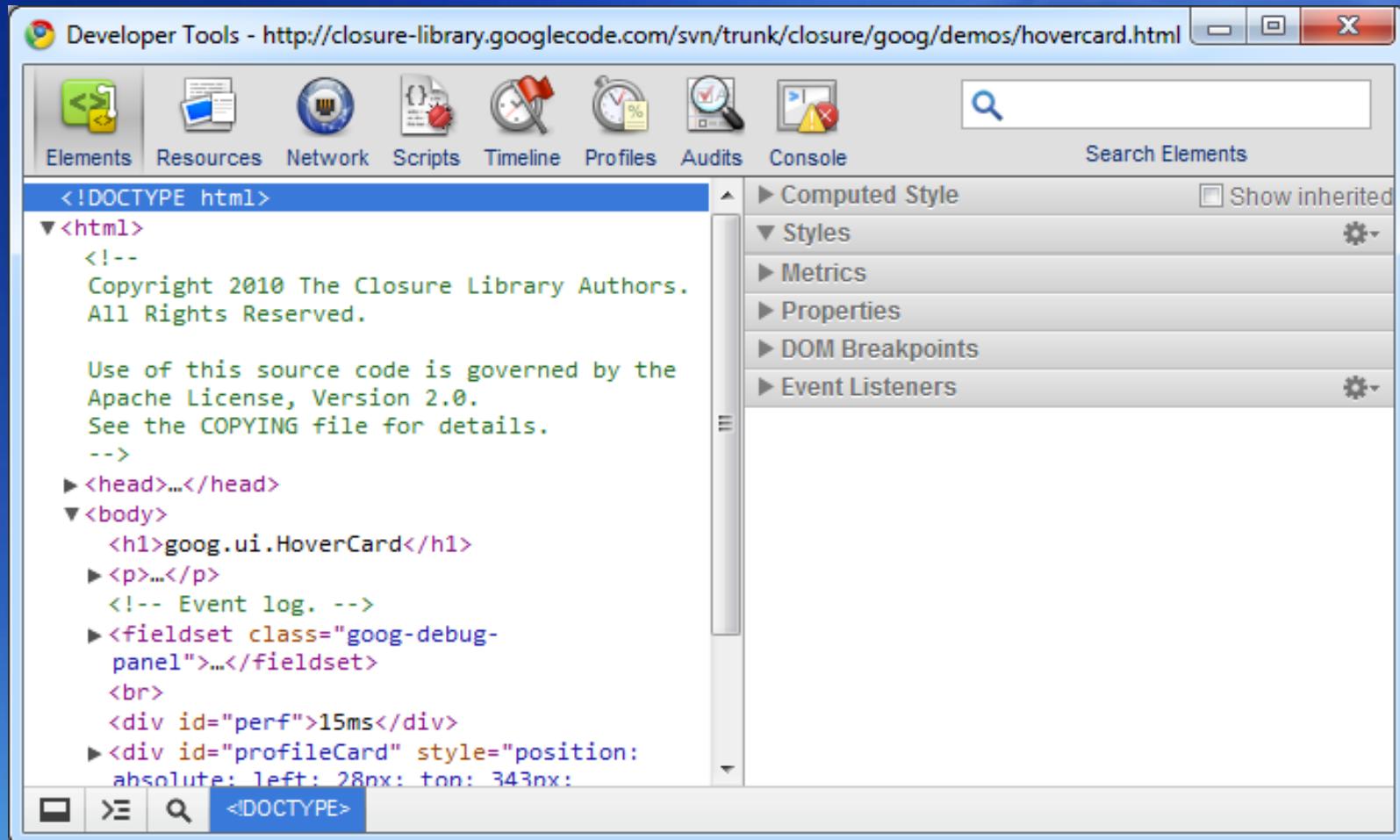
- Are organized into task-oriented groups.
- Are represented by icons in the toolbar at the top of the window.



- Lets you work with a specific type of page or app information, including DOM elements, resources, and scripts.
- Provides a search field that enables you to search the current panel.

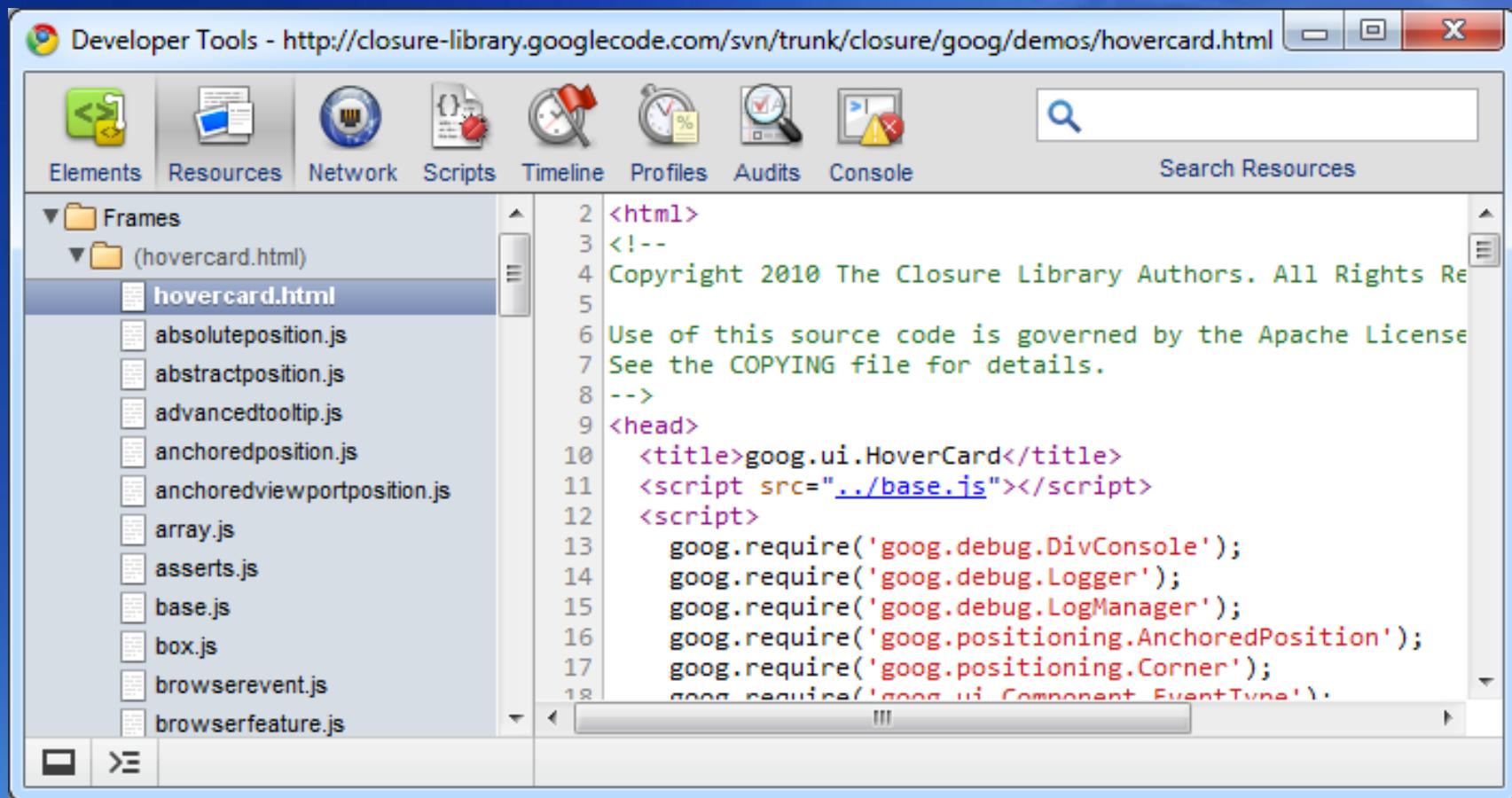
# Chrome Developer Tools (Cont.)

## Elements Panel



# Chrome Developer Tools (Cont.)

## Resources Panel



# Chrome Developer Tools (Cont.)

## Network Panel

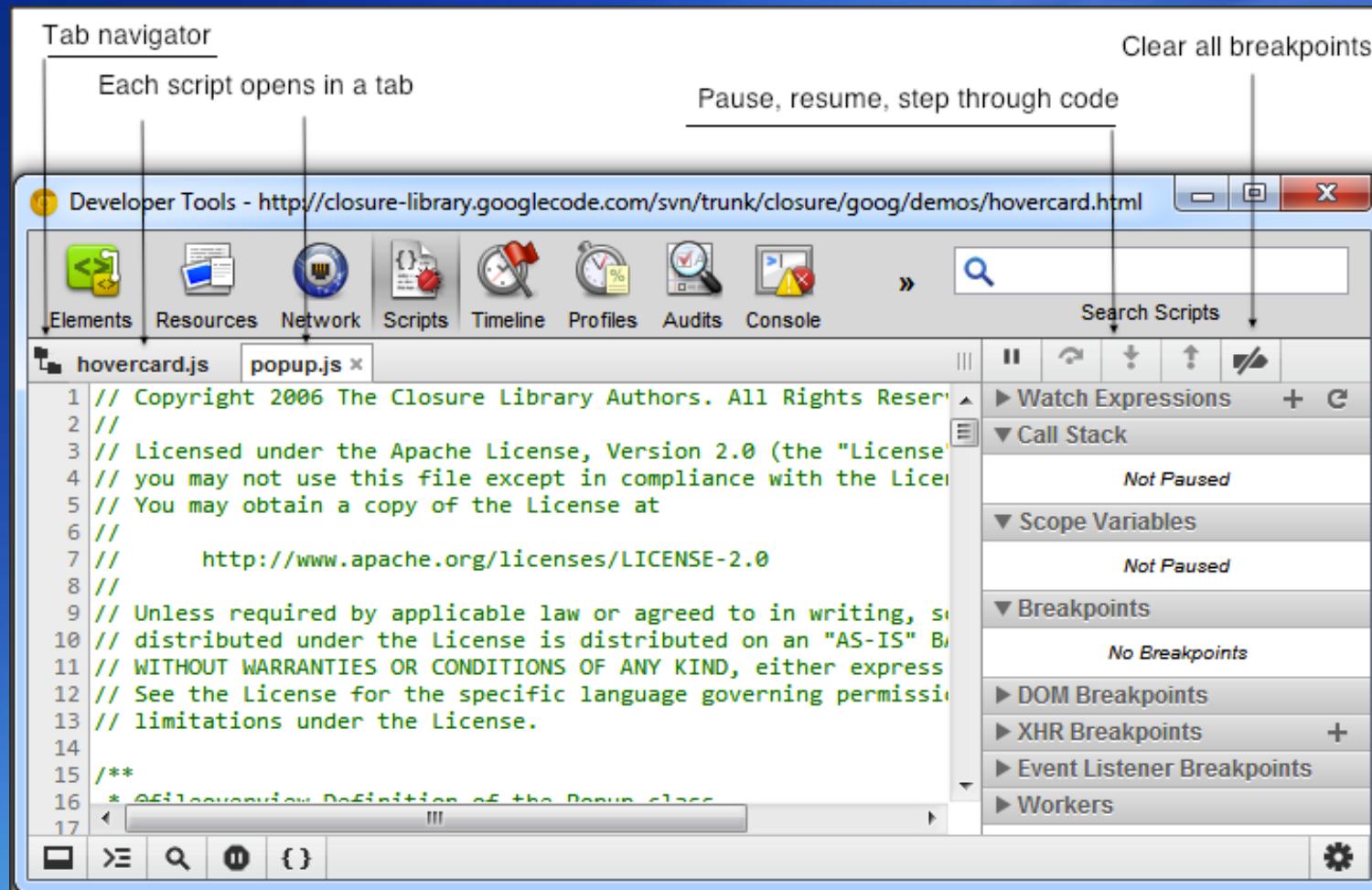
The screenshot shows the Network panel of the Chrome Developer Tools. The title bar indicates the URL: `Developer Tools - http://closure-library.googlecode.com/svn/trunk/closure/goog/demos/hovercard.html`. The Network tab is selected, showing a list of network requests:

Name	Path	Method	Status	Type	Initiator	Size	Time	Timeline	732ms	1.10s	1.46s
			Text			Content	Laten				
hovercard.html	/svn/trunk/closure/c	GET	304	text...	Other	212B	161ms	blue toggle			
hovercard.css	/svn/trunk/closure/c	GET	304	text...	hovercard.h Parser	209B 963B	131ms 124ms	green toggle			
demo.css	/svn/trunk/closure/c	GET	304	text...	hovercard.h Parser	210B 1.17Kb	122ms 114ms	green toggle			
base.js	/svn/trunk/closure/c	GET	304	text...	hovercard.h Parser	174B 47.15Kb	29ms 22ms	orange toggle			
deps.js	/svn/trunk/closure/c	GET	304	text...	base.js:541 Script	175B 204.1E	30ms 22ms	orange toggle			
relativetimeprovi	/svn/trunk/closure/c	GET	304	text	base.js:541	221B	136ms	orange toggle			

At the bottom, there are tabs for `All`, `Documents`, `Stylesheets`, `Images`, `Scripts`, `XHR`, `Fonts`, and `WebSockets`.

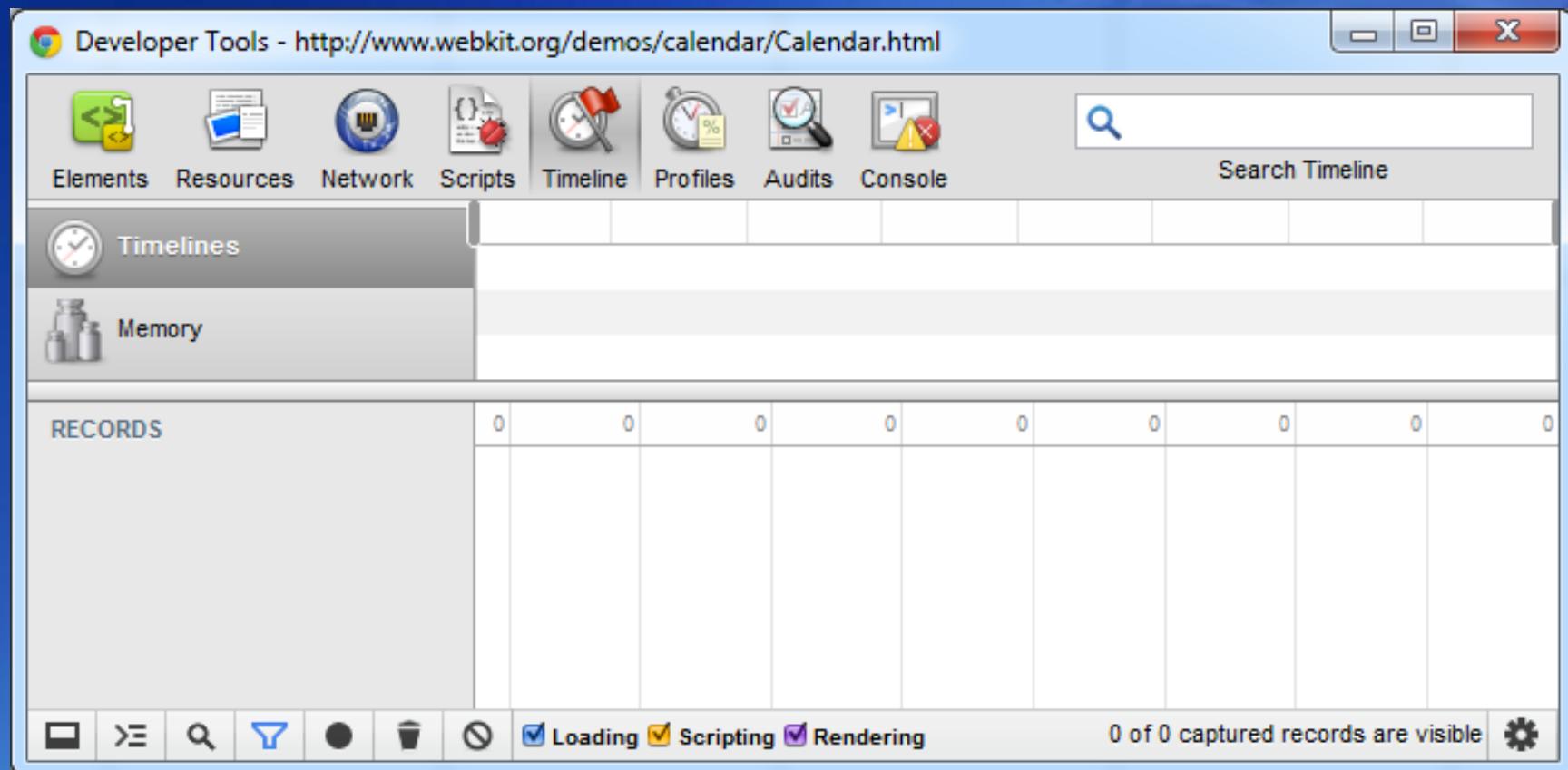
# Chrome Developer Tools (Cont.)

## Scripts Panel



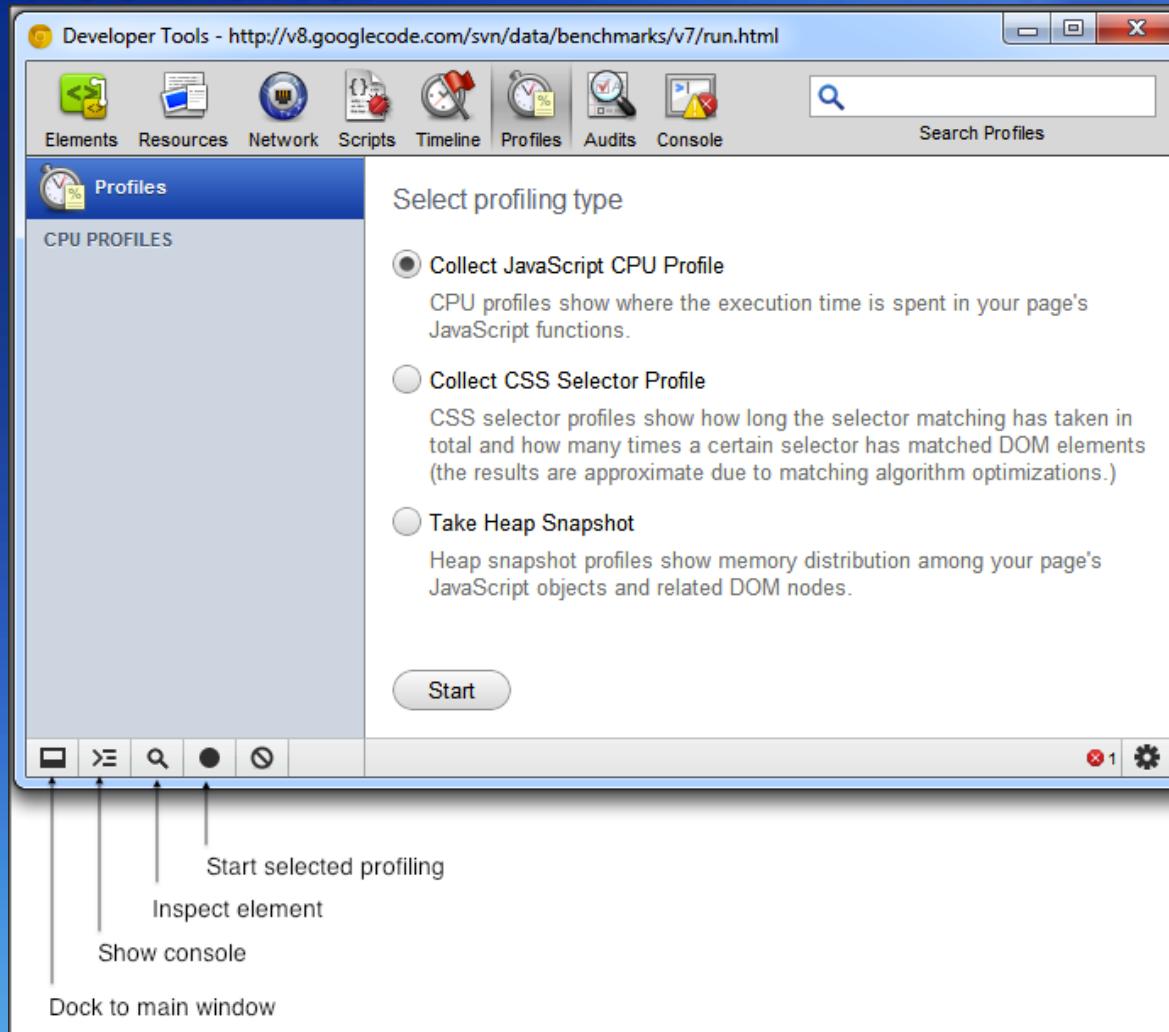
# Chrome Developer Tools (Cont.)

## Timeline Panel



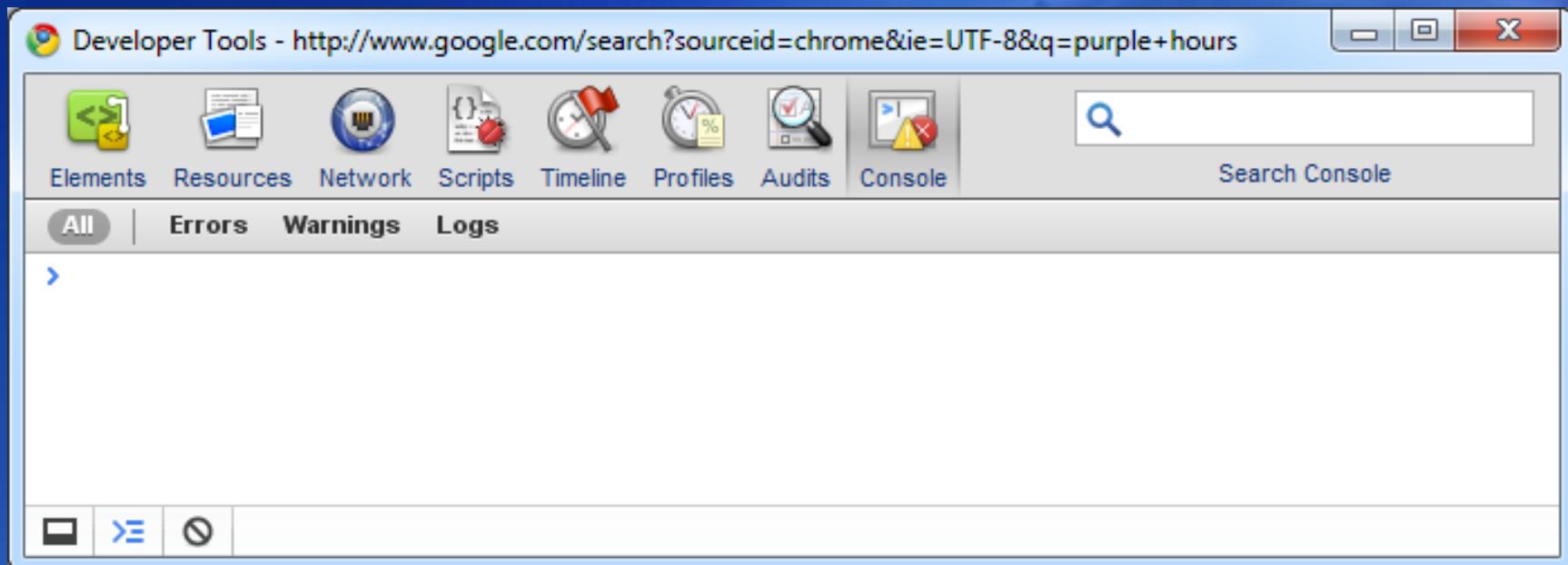
# Chrome Developer Tools (Cont.)

## • Profiles Panel

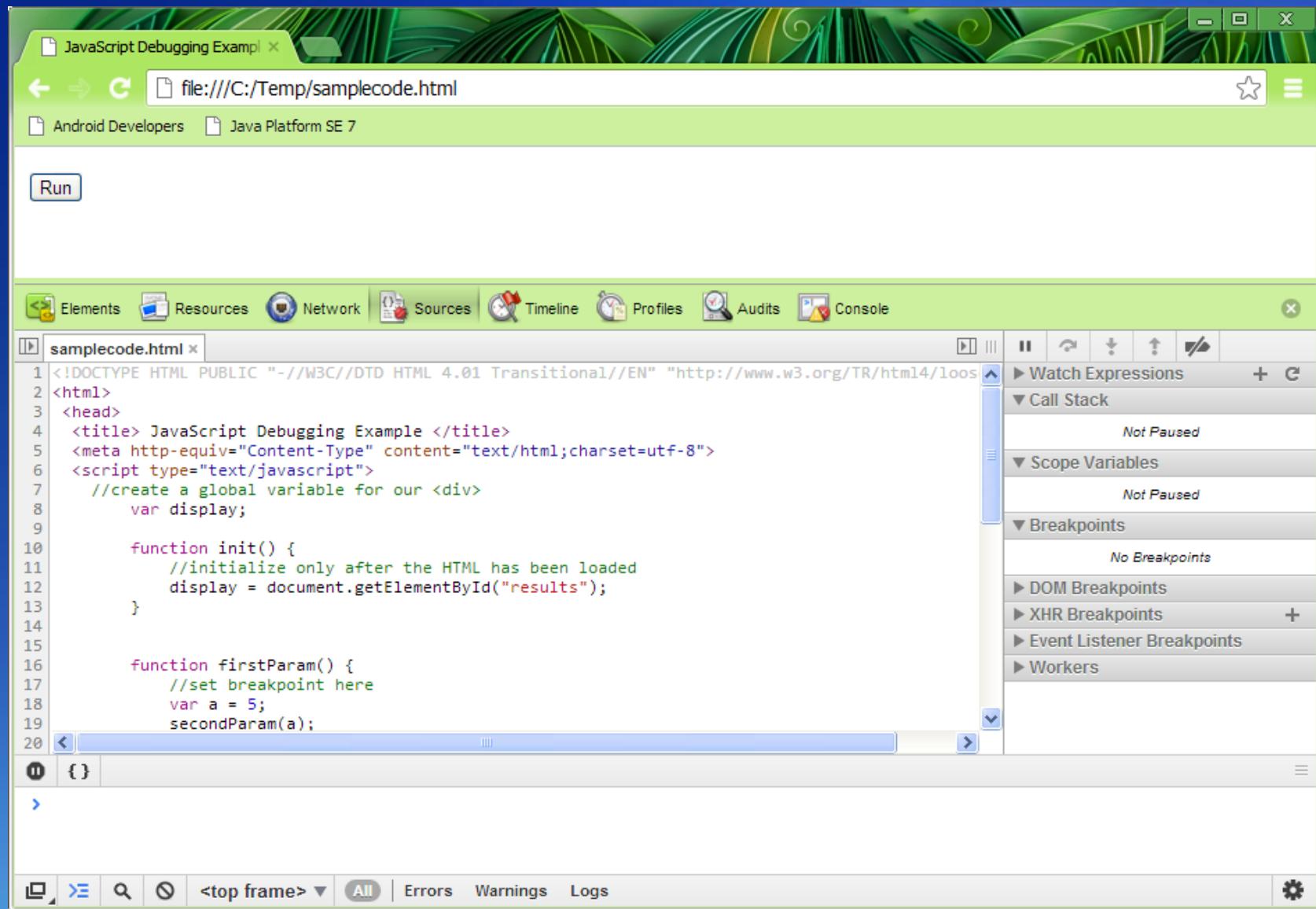


# Chrome Developer Tools (Cont.)

## Console Panel



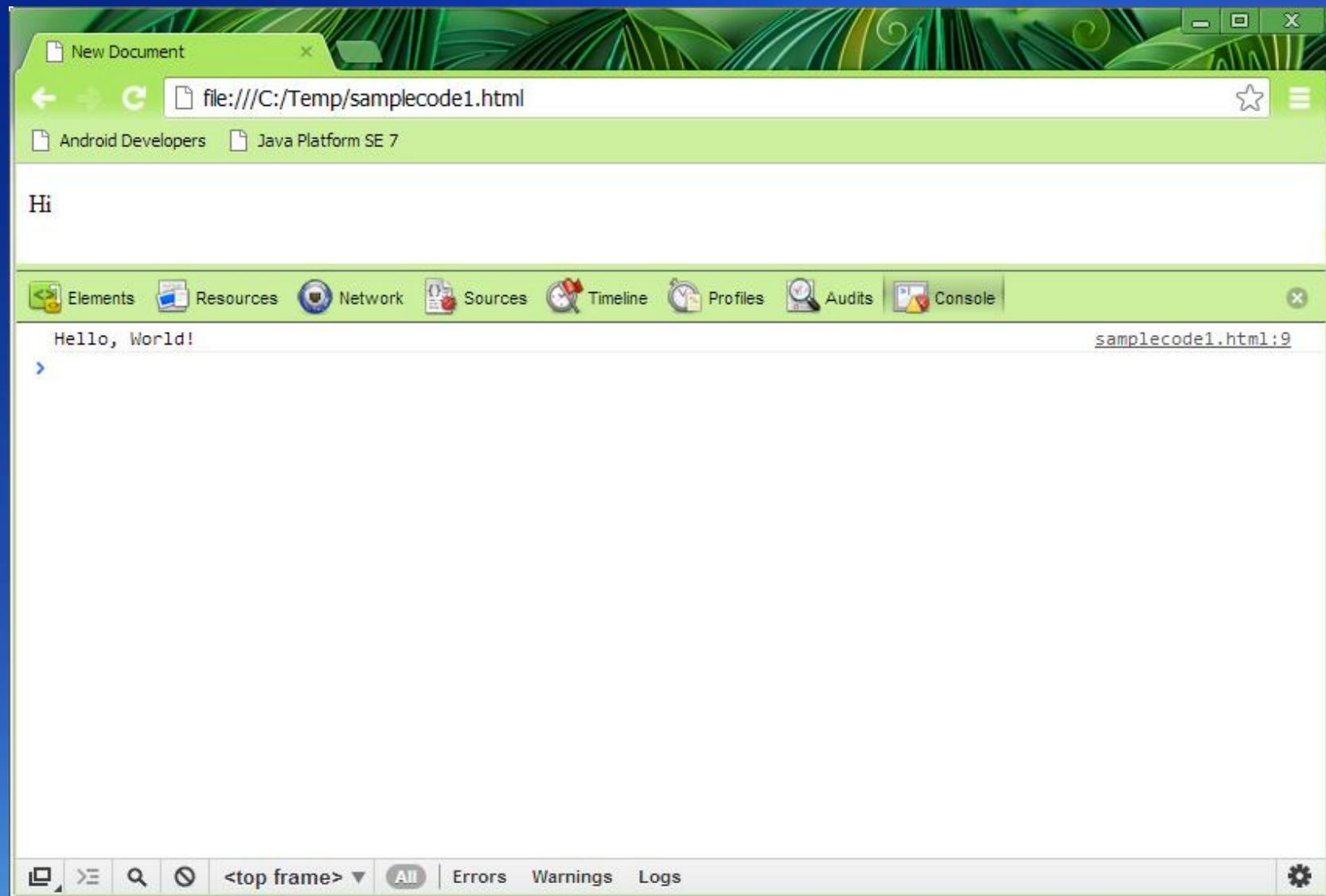
# Debugging Test for Chrome Developer Tools



# Debugging Test for Chrome Developer Tools (Cont.)

1. In Chrome Browser, load the example.
2. Press **Ctrl + Shift + I** to open the **Chrome Developer** tools, and click the **Sources** tab.
3. In the bottom pane, scroll to the first function, click the line that says "**var a = 5;**", and click **Breakpoints** tab in the right pane.
4. Click **F8** to process **Pause script execution**, and then click **Refresh** in the browser toolbar, and then click **F11** to process **Step into next function call**.
5. Click **Run** button in the web page, click the **Scope Variables** tab on the right panel.

# Using console.log in Chrome



# Firefox and Firebug

• <http://getfirebug.com>

The screenshot shows the Firebug website loaded in a Mozilla Firefox browser window. The title bar reads "Firebug - Mozilla Firefox". The address bar shows "getfirebug.com". The main content area features a large orange firebug logo and the text "Firebug Web Development Evolved.". A sidebar on the left has a large firebug illustration. The main content area includes sections for "What is Firebug?", "Documentation", "Community", and "Get Involved". A prominent red button says "Install Firebug". Below it are links for "Other Versions", "Firebug Lite", and "Extensions". A central box highlights "The most popular and powerful web development tool" with a list of features and a "More Features" link. Below this are sections for "Inspect", "Log", "Profile", "Debug", "Analyze", and "Layout". Each section has an icon and a brief description.

What is Firebug?  
Introduction and Features

Documentation  
FAQ and Wiki

Community  
Discussion forums and lists

Get Involved  
Hack the code, create extensions

Install Firebug  
for Firefox, 100% free and open source

Other Versions | Firebug Lite | Extensions

The most popular and powerful web development tool

- ✓ Inspect HTML and modify style and layout in real-time
- ✓ Use the most advanced JavaScript debugger available for any browser
- ✓ Accurately analyze network usage and performance
- ✓ Extend Firebug and add features to make Firebug even more powerful
- ✓ Get the information you need to get it done with Firebug.

More Features »

Introduction to Firebug  
Firebug pyroentomologist Rob Campbell gives a quick introduction to Firebug.  
Watch now »

More Screencasts »

Inspect  
Pinpoint an element in a webpage with ease and precision.

Log  
Send messages to the console direct from your webpage through Javascript.

Profile  
Measure your Javascript performance in the Console's Profiler.

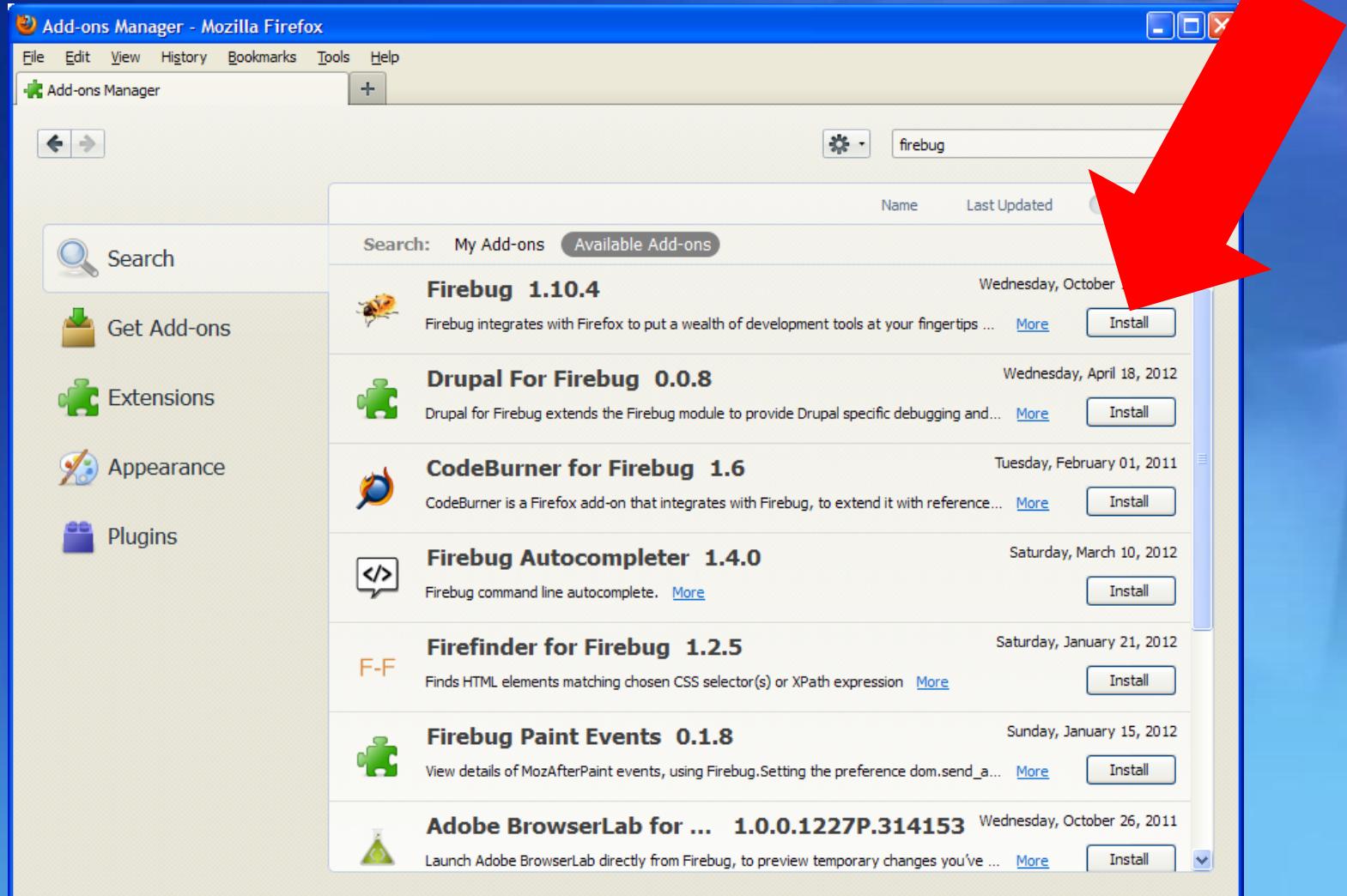
Debug  
Step-by-step interactive debugging in a visual environment.

Analyze  
Look at detailed measurements of your site's network activity.

Layout  
Tweak and position HTML elements with CSS and the Layout panel.

# Firefox and Firebug (Cont.)

## Firebug Installation on Firefox.



# Debugging Test for Firefox Firebug

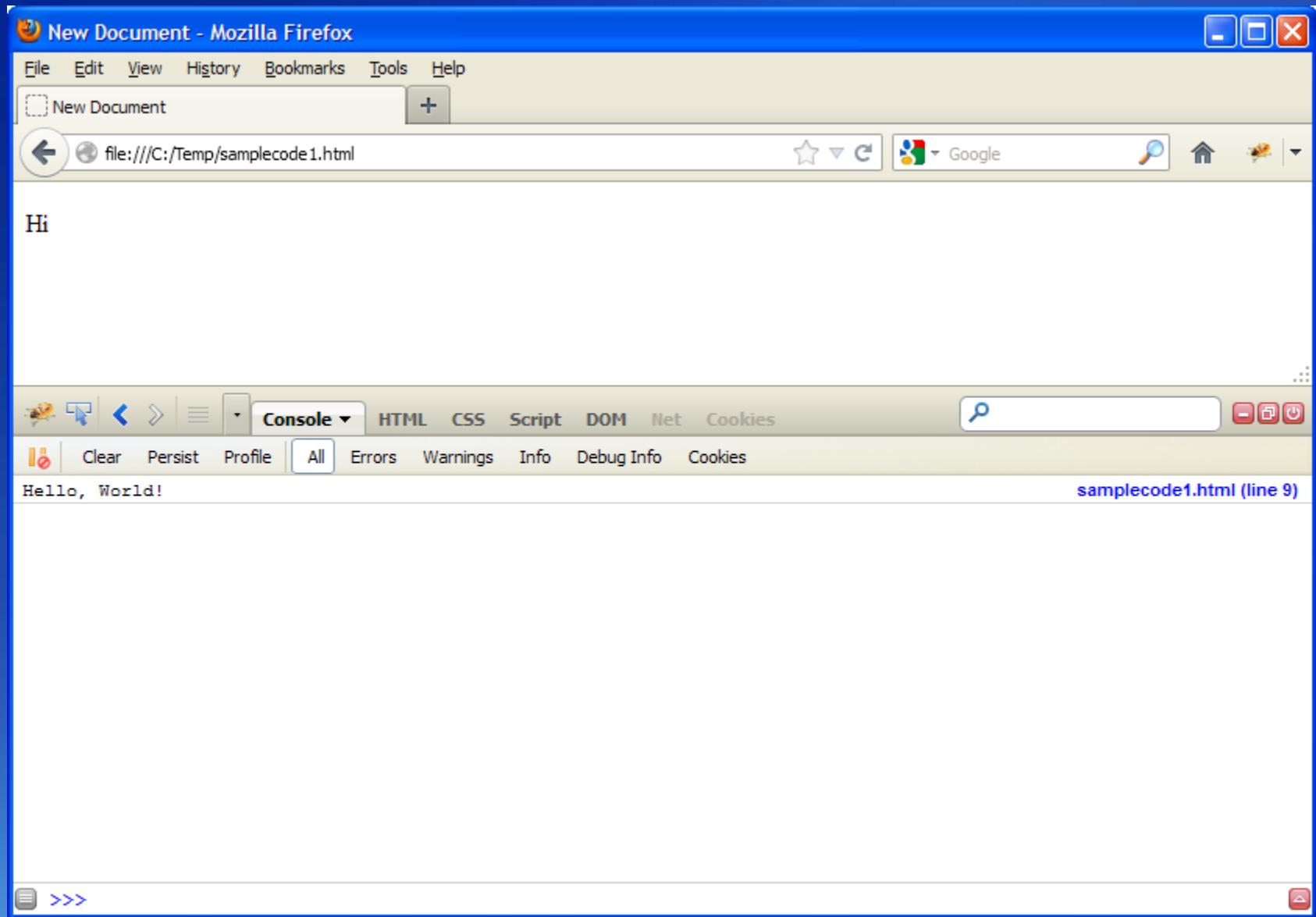
The screenshot shows a Mozilla Firefox browser window titled "JavaScript Debugging Example - Mozilla Firefox". The address bar displays "file:///C:/Temp/samplecode.html". The Firebug toolbar at the top includes tabs for Console, HTML, CSS, Script (selected), DOM, Net, and Cookies. The Script tab shows the source code of "samplecode.html" with breakpoints set at line 18 and line 25. A yellow highlight is on line 25, which contains the call to "thirdParam(a, b);". The Watch panel on the right shows variables: "this" (Window), "a" (5), "arguments" ([5]), "b" (10), and "toString" (function). The Stack and Breakpoints panels are also visible.

```
15
16      function firstParam() {
17          //set breakpoint here
18          var a = 5;
19          secondParam(a);
20      }
21
22
23      function secondParam(a) {
24          var b = 10;
25          thirdParam(a, b);
26      }
27
28      function thirdParam(a, b) {
29          var c = 15;
30          var d = a + b + c;
31
32          if (window.console && window.console.log) {
33              window.console.log(a + " + " + b + " + " + c + " = " + c
```

# Debugging Test for Firefox Firebug (Cont.)

1. In Firefox Browser, load the example.
2. Press **F12** to open the **Open Firebug** tools, and click the **Script** tab.
3. In the bottom pane, scroll to the first function, click the line that says "**var a = 5;**".
4. Click **Run** button in the web page, click **F11** to process **Step Into**.
5. Click the **Watch** tab on the right panel.

# Using console.log in Firefox



# Opera and Dragonfly

- <http://www.opera.com/dragonfly/>

The screenshot shows the Opera browser window displaying the Opera Dragonfly developer tools. The title bar reads "Opera Dragonfly - Oper...". The address bar shows "www.opera.com/dragonfly/". The main content area features a large black dragonfly silhouette against a hexagonal background. At the top right is the "OPERA software" logo. Below the dragonfly, the text "Opera Dragonfly" is displayed in large white letters. A descriptive paragraph follows: "Fast, lean and powerful. Meet Opera Dragonfly — our fully-featured suite of developer tools, designed to make your job easier. It's just a right-click away. No install required." At the bottom, there are four sections: "Built-in" (with a video player thumbnail), "Fully loaded" (with text about developer tools), "Cross-device" (with text about remote debugging), and "Watch video".

**Opera Dragonfly**

Fast, lean and powerful. Meet Opera Dragonfly — our fully-featured suite of developer tools, designed to make your job easier. It's just a right-click away. No install required.

**Built-in**

Opera Dragonfly is built right into the browser and updates automatically behind the scenes. Launch Opera Dragonfly with **Ctrl + Shift + I** on Windows and Linux, or **⌘ + ⌘ + I** on Mac. Alternatively, you can target a

**Fully loaded**

A full suite of tools puts you in control. Step through your code, manipulate the DOM, monitor network traffic, search for what you need, filter away what you don't, and a whole lot more. Whether you are a developer or a designer,

**Cross-device**

Develop for mobile, tablet or TV? Opera Dragonfly has that covered too. Remote debugging allows you to connect to the Opera browser on your device and to start debugging from your PC or Mac. All the features are available, so you

# Debugging Test for Opera Dragonfly

The screenshot shows the Opera Dragonfly JavaScript debugger interface. At the top, there's a toolbar with icons for file operations, a search bar, and a Google search bar. Below the toolbar is a main window divided into several panes.

- Code Editor:** On the left, a code editor displays the following JavaScript code:

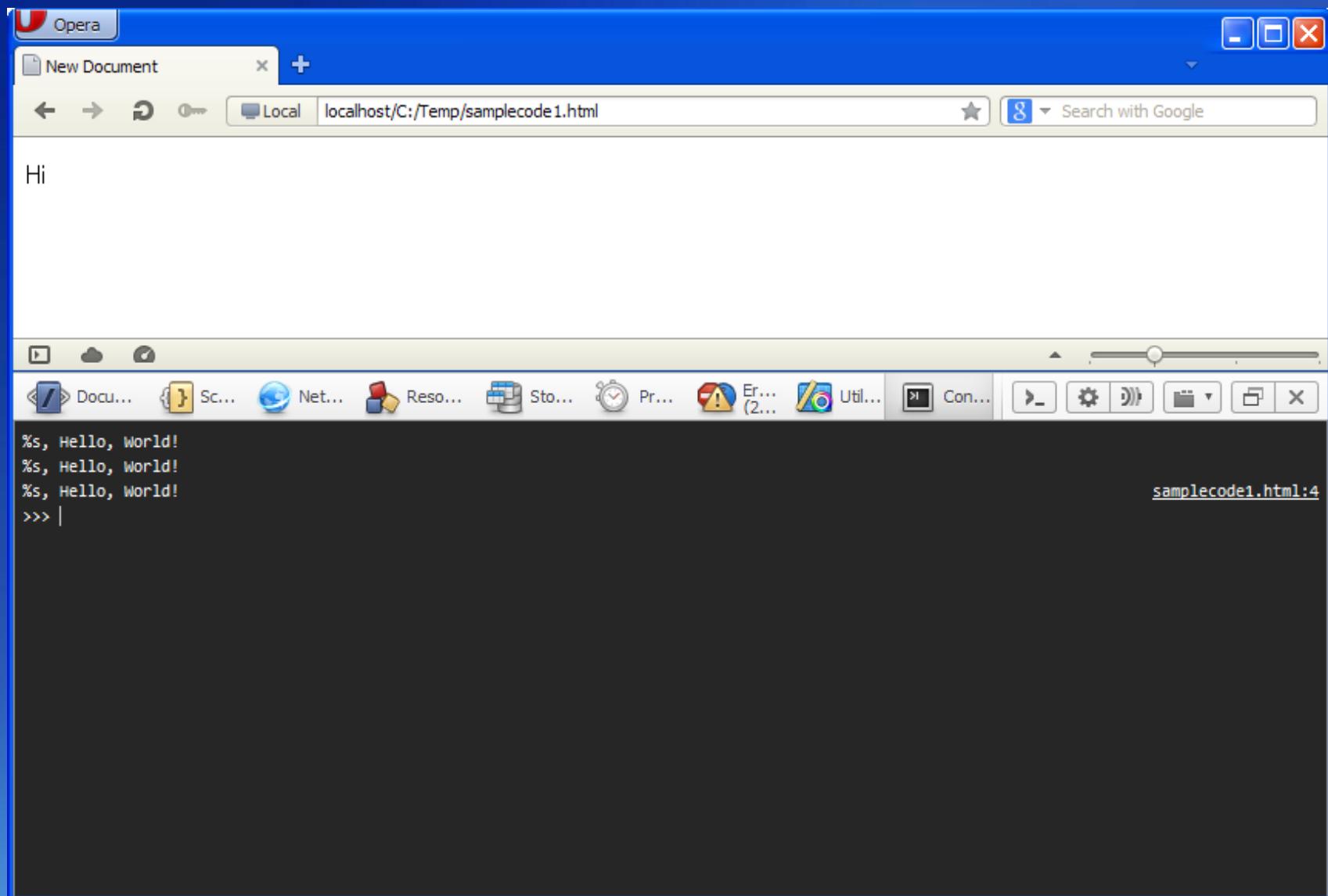
```
1      var display;
2      function init() {
3          display = document.getElementById("results");
4      }
5      function firstParam() {
6          //set breakpoint here
7          var a = 5;
8          secondParam(a);
9      }
10     function secondParam(a) {
11         var b = 10;
12         thirdParam(a, b);
13     }
14     function thirdParam(a, b) {
15         var c = 15;
16         var d = a + b + c;
17         if (window.console && window.console.log) {
18             window.console.log(a + " + " + b + " + " + c + " = " + d);
19         } else {
20             display.innerHTML = a + " + " + b + " + " + c + " = " + d;
21         }
22     }
23 }
24 }
```

A red dot at line 9 indicates a breakpoint has been set.
- Toolbars:** A horizontal toolbar below the code editor contains icons for document, script, network, resources, storage, profiling, errors, utilities, console, and other developer tools.
- Inspector Panels:** On the right side, there are several panels:
  - State:** Shows the current state of variables and objects.
  - Breakpoints:** Shows a list of breakpoints, with one at line 9 highlighted.
  - Search:** A search bar for finding specific values.
  - Watches:** A list of watched variables.
  - Return Values:** A section indicating "No return values".
  - Call Stack:** A stack trace showing the execution path: `thirdParam samplecode.html:16`, `secondParam samplecode.html:13`, `firstParam samplecode.html:9`, `(anonymous) samplecode.html:1`, and `(global)`.
  - Inspection:** A panel for inspecting specific objects, currently showing "this Window" with properties `a` (value 5) and `arguments Arguments` with properties `b` (value 10), `c` (value undefined), and `d` (value undefined).
  - Scope Chain:** A panel showing the scope chain.

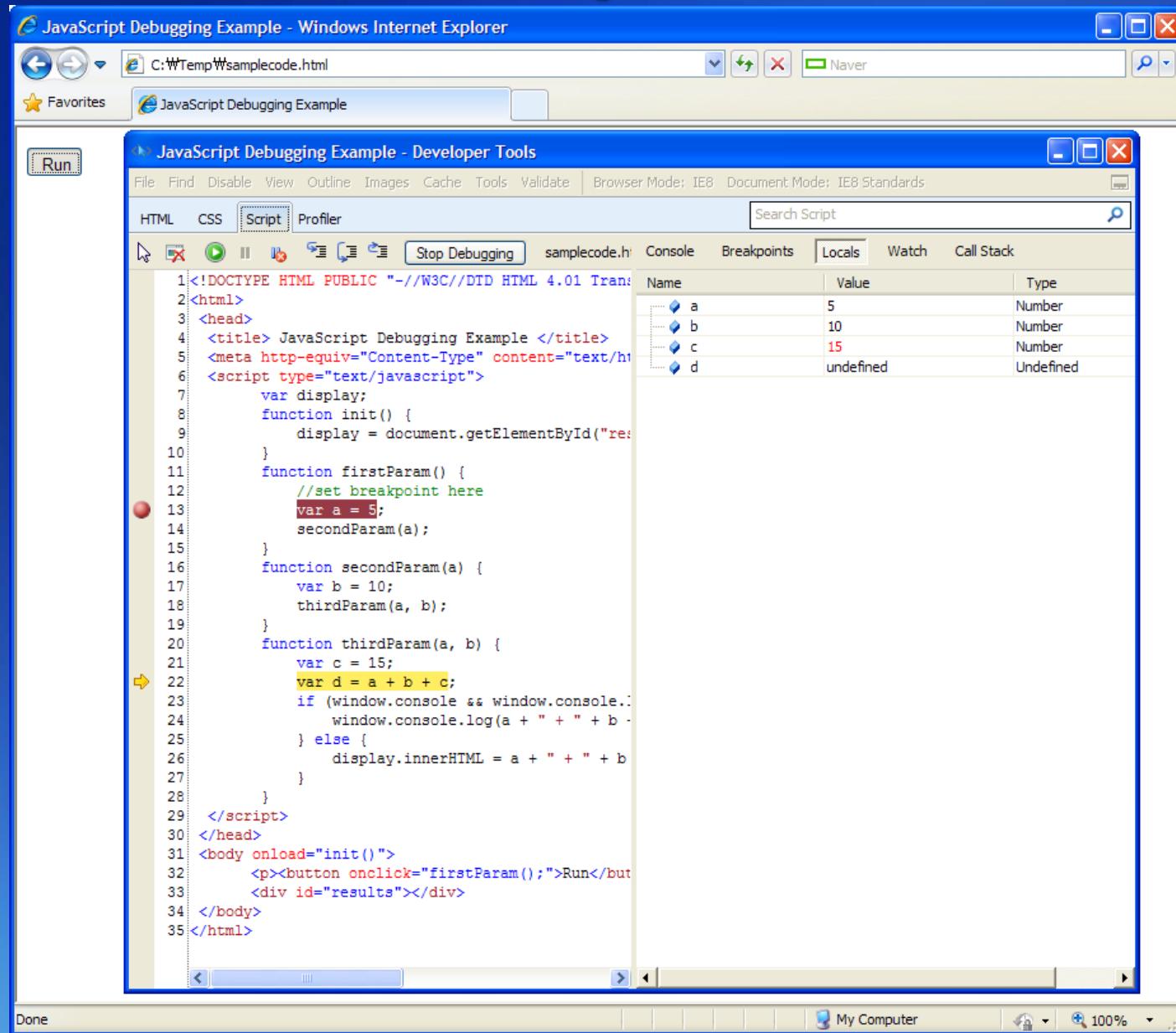
# Debugging Test for Opera Dragonfly (Cont.)

1. In Opera Browser, load the example.
2. Press **Ctrl + Shift + I** to open the **Opera Dragonfly** tools, and click the **Scripts** tab.
3. In the bottom pane, scroll to the first function, click the line that says "**var a = 5;**".
4. Click **Run** button in the web page, click **F11** to process **Step Into**.
5. Click the **Inspecition** tab on the right panel.

# Using console.log in Opera



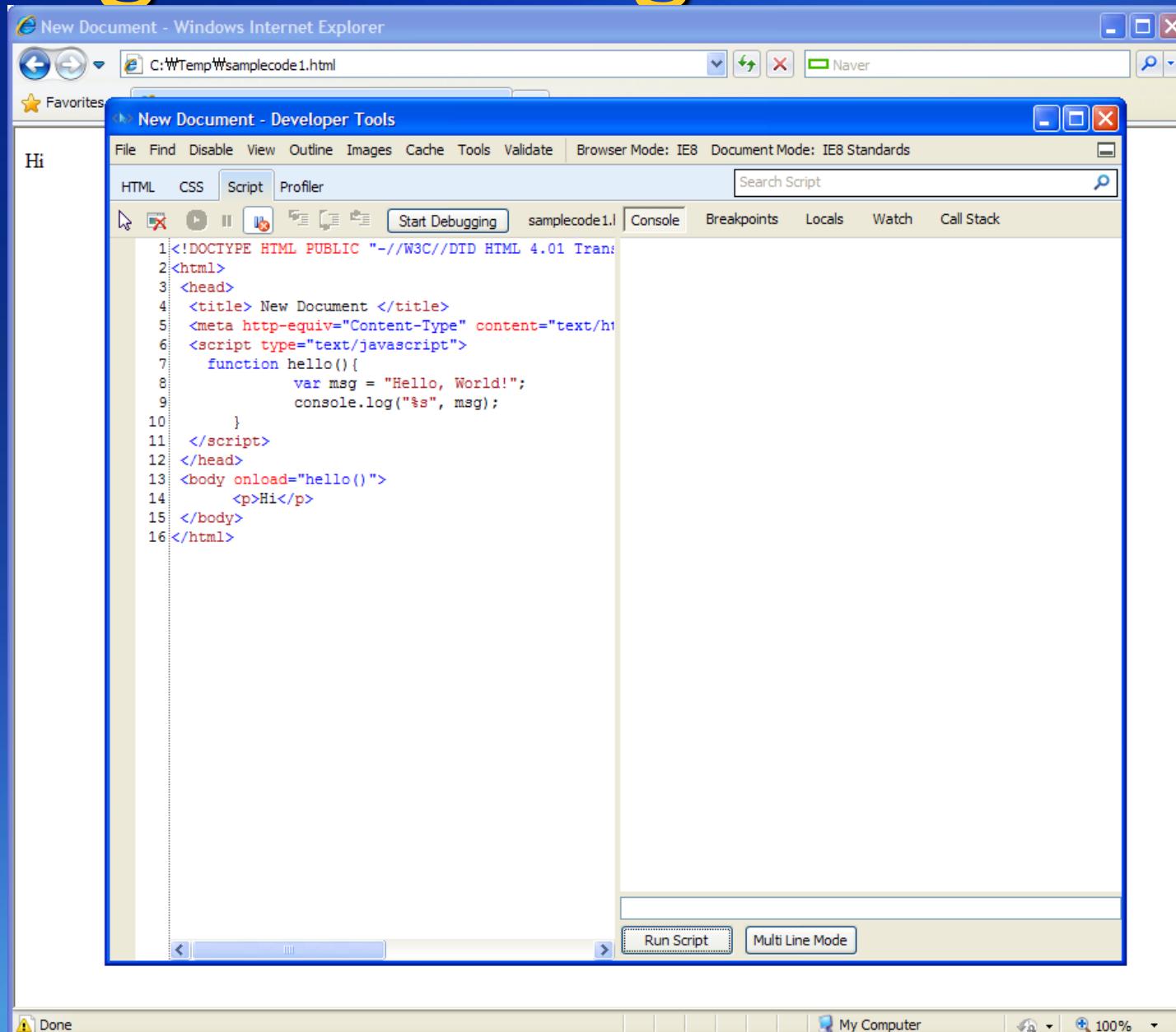
# IE's F12 Developer Tools



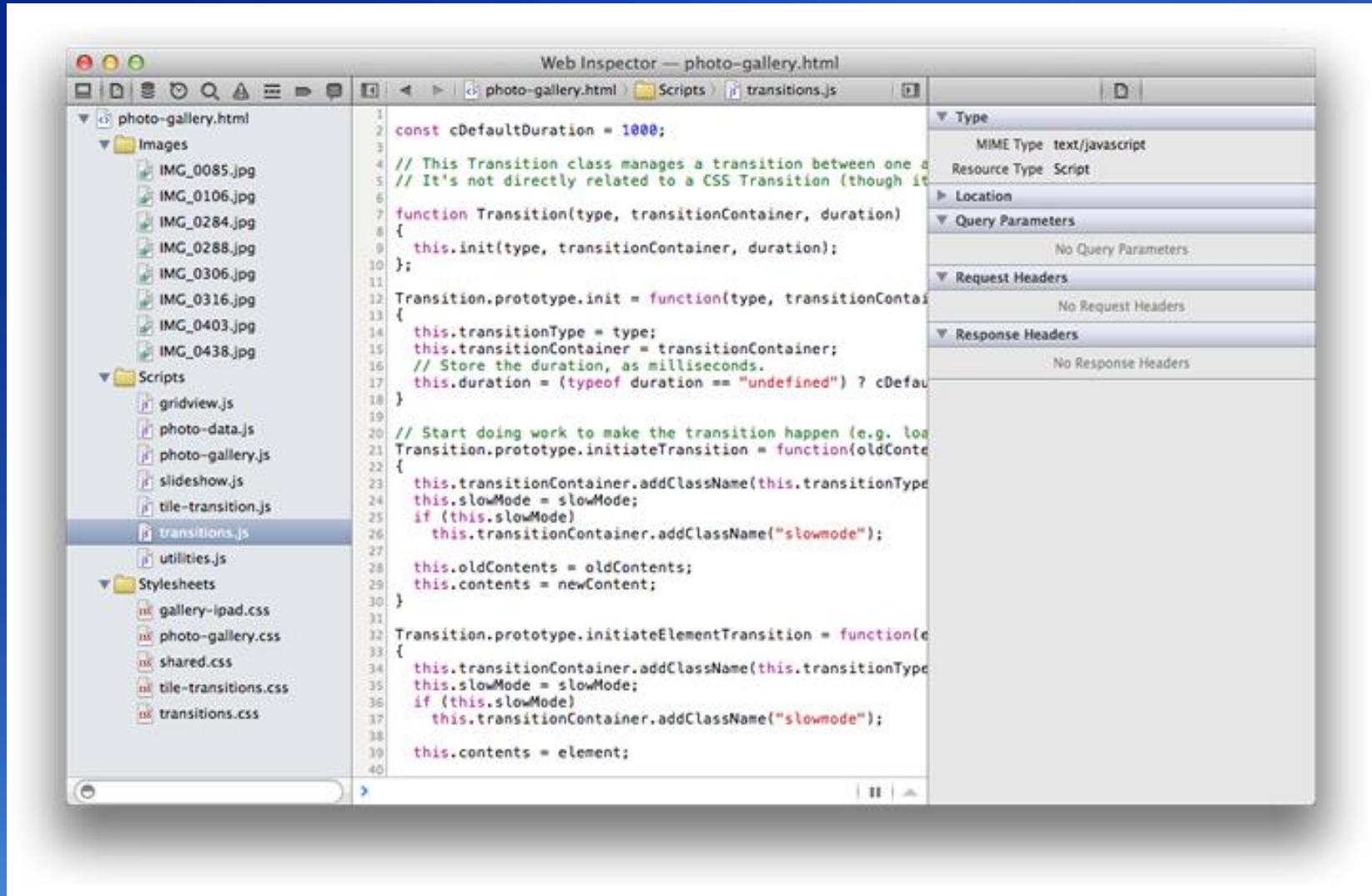
# Debugging Test for IE (Cont.)

1. In Internet Explorer 9, load the example.
2. Press **F12** to open the **F12** tools, and click the **Script** tab.
3. In the left pane, scroll to the first function, right-click the line that says "var a = 5;", and click **Insert breakpoint**.
4. Click **Start debugging**, and then on the webpage in the browser, click the **Run** button.
5. In **F12** tools, click the **Watch** tab on the right side, and add the variables "a, b, c, and d.".
6. Step through your code by pressing **F11**, or by clicking the **Step into** button, and watch the variables on the **Watch** tab.

# Using console.log in IE

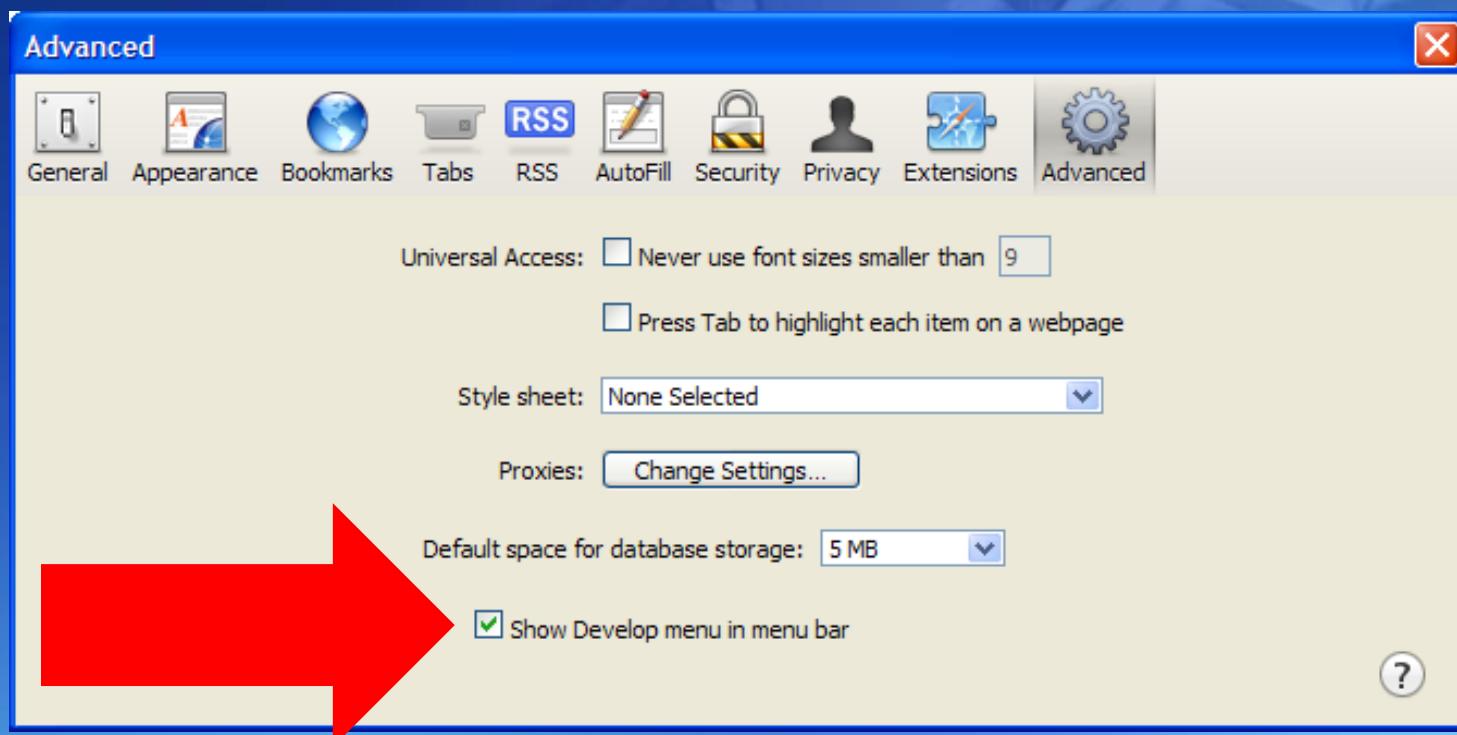


# Safari Web Development Tools

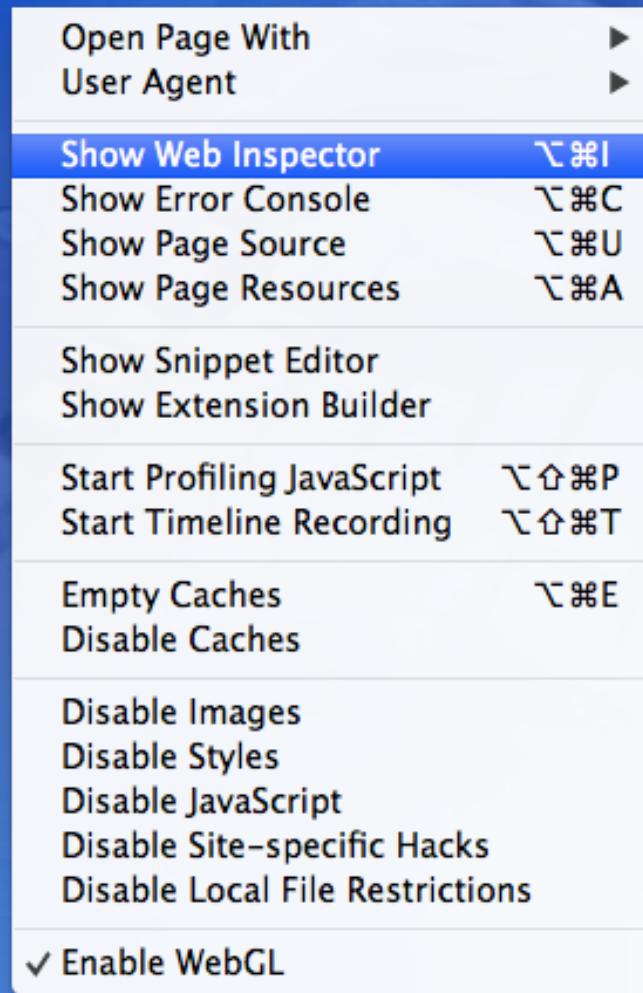
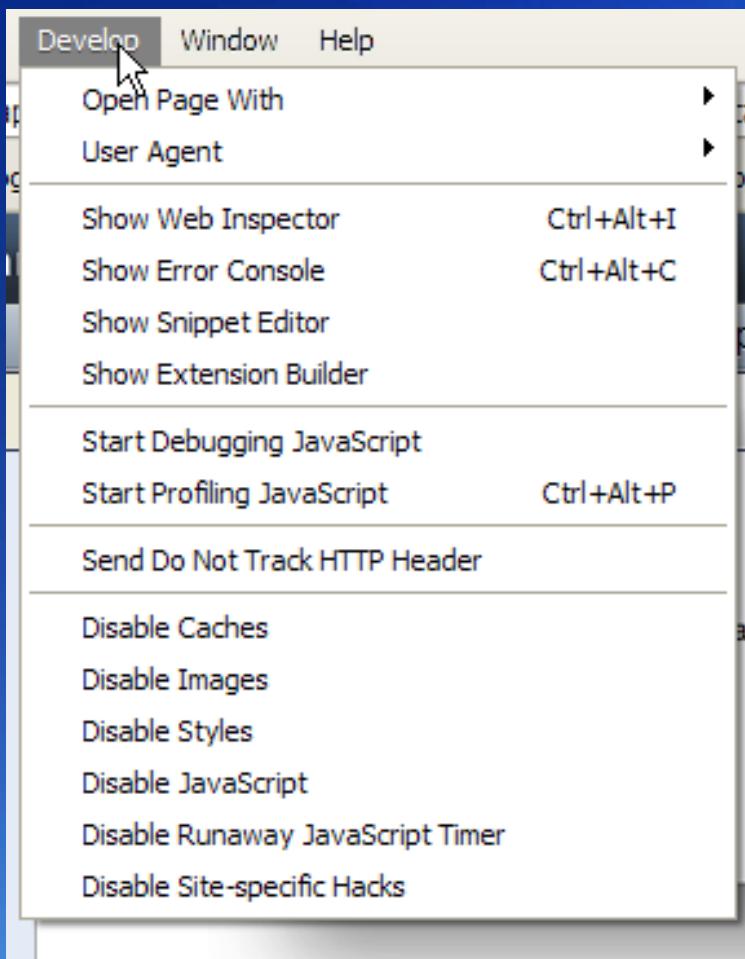


# Safari Web Development Tools (Cont.)

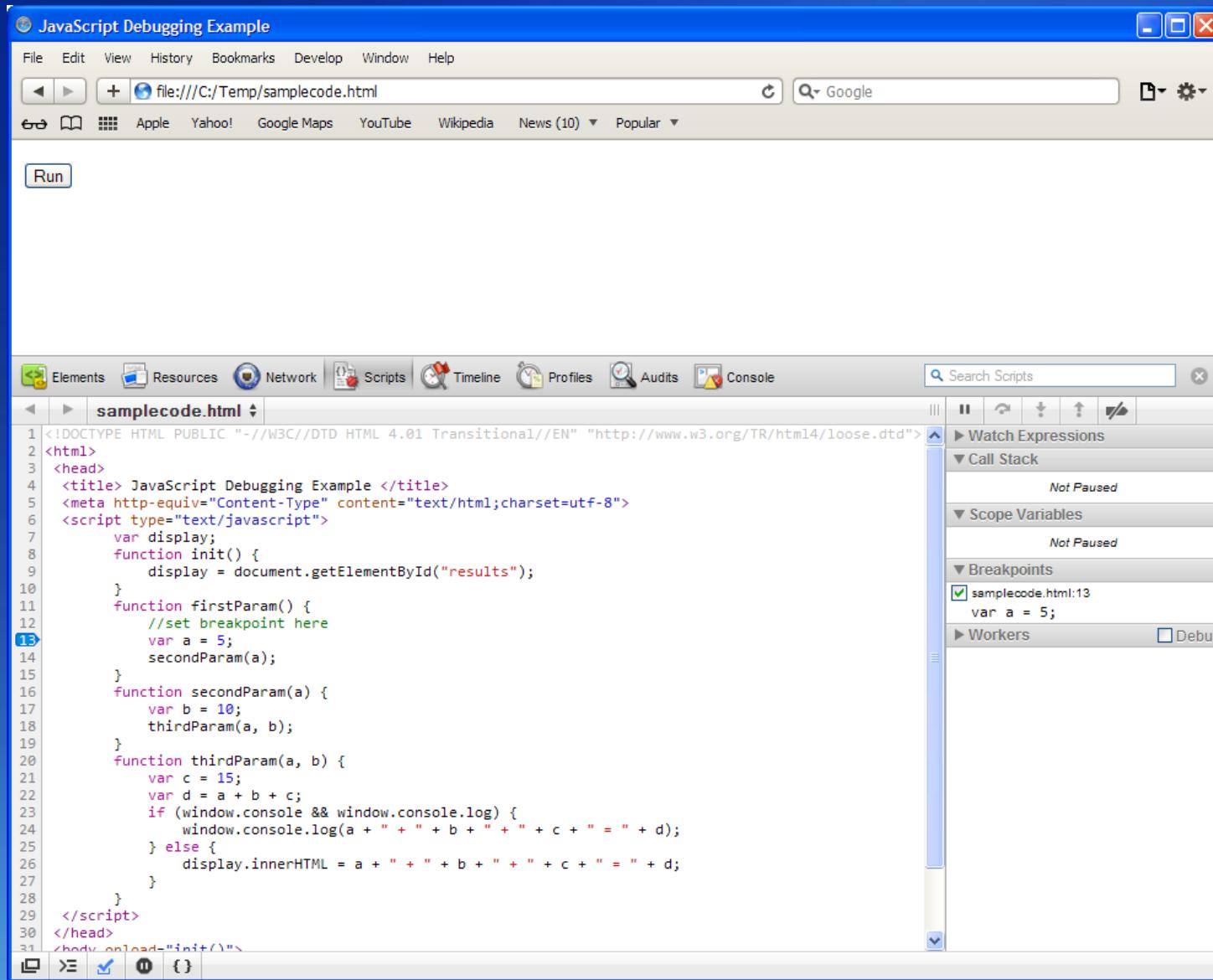
1. In Safari **preferences**, click **Advanced**, then select "Show Develop menu in menu bar"



# Safari Web Development Tools (Cont.)



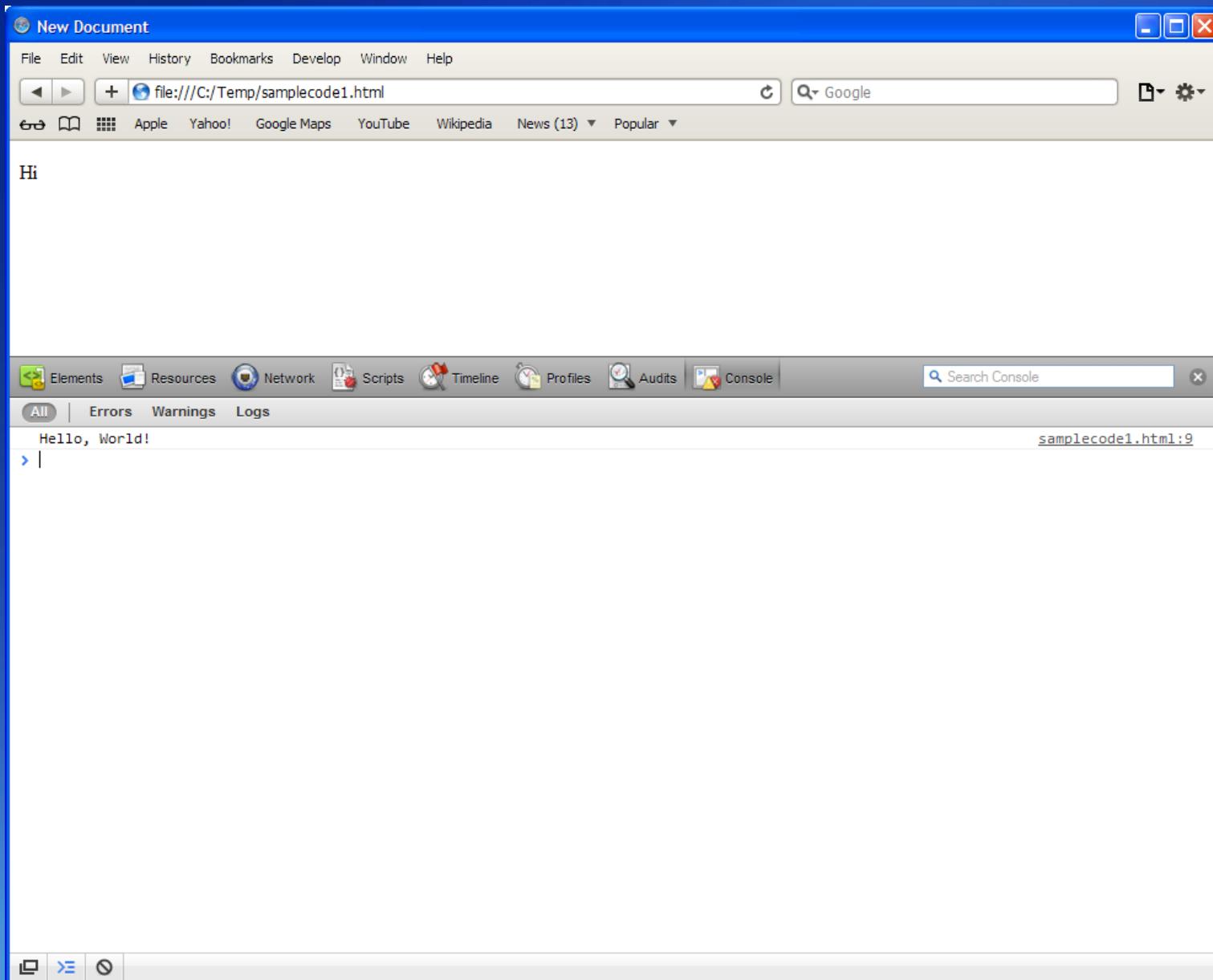
# Debugging Test for Safari



# Debugging Test for Safari (Cont.)

1. In Safari Browser, load the example.
2. Press **Show Web Inspector** in **Develop** Menu to open the **Safari Development Tool** tools, and click the **Scripts** tab.
3. In the bottom pane, scroll to the first function, click the line that says "**var a = 5;**", and click **Breakpoints** tab in the right pane.
4. Click **Pause script execution** button, and then click **Refresh** in the browser toolbar, and then click **Step into next function call** button on the right panel.
5. Click **Run** button in the web page, click the **Scope Variables** tab on the right panel.

# Using console.log in Safari



# Dealing with Cross-Browser Differences

- 15 years ago, dealing with cross-browser differences between the two dominant browsers at the time—Internet Explorer and Netscape’s Navigator—was a heroic and painful process.
- Differences still exist, and all of the differences aren’t just between Firefox-Opera-Safari-Chrome and Internet Explorer.

# Object Detection

```
<script type="text/javascript">
// *** BROWSER VERSION ***
this.major = parseInt(navigator.appVersion)
this.minor = parseFloat(navigator.appVersion)
this.nav = ((agt.indexOf('mozilla') != -1) && ((agt.indexOf('spoofed') == -1)
..... && (agt.indexOf('compatible') == -1)))
this.nav2 = (this.nav && (this.major == 2))
this.nav3 = (this.nav && (this.major == 3))
this.nav4 = (this.nav && (this.major == 4))
this.nav4up = this.nav && (this.major >= 4)
this.navonly = (this.nav && (agt.indexOf(";nav") != -1))
this.ie = (agt.indexOf("msie") != -1)
this.ie3 = (this.ie && (this.major == 2))
this.ie4 = (this.ie && (this.major == 4))
this.ie4up = this.ie && (this.major >= 4)
this.opera = (agt.indexOf("opera") != -1)
</script>
```

# Object Detection (Cont.)

- With object detection, the JavaScript application accesses the object being detected in a conditional statement.
- If the object doesn't exist, the condition evaluates to **false**.
- As an example, the following checks to ensure that the most basic level of object model support is provided in the browser or other user agent.

```
if (document.getElementById) . . .
```

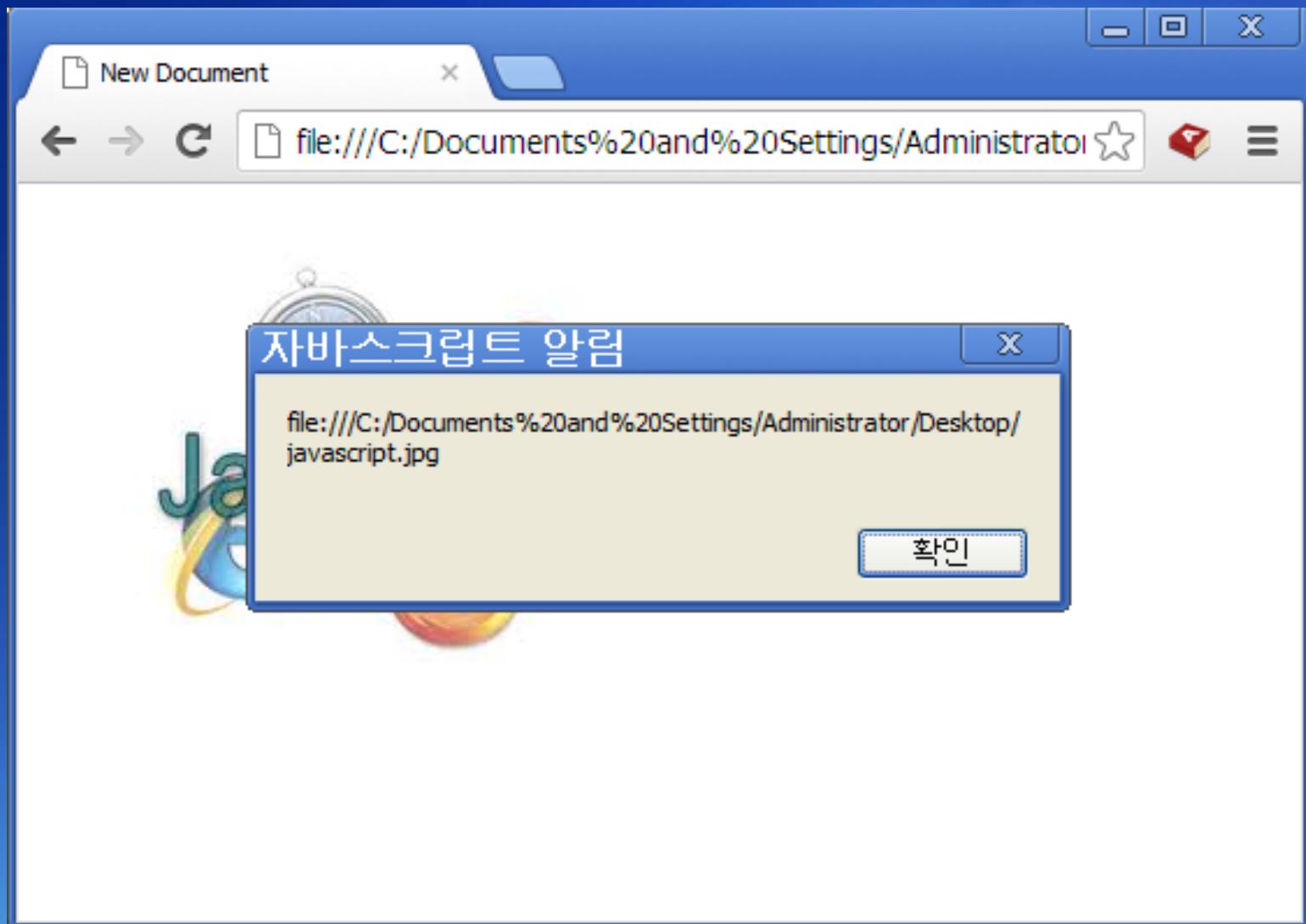
# Object Detection (Cont.)

```
<script type="text/javascript">
function showBlock(evnt) {
    var theEvent = evnt ? evnt : window.event;
    var theSrc = theEvent.target ? theEvent.target : theEvent.srcElement;
    var itemId = "elements" + theSrc.id.substr(5,1);
    var item = document.getElementById(itemId);
    if (item.style.display=='none') {
        item.style.display='block';
    } else {
        item.style.display='none';
    }
}
</script>
```

# Object Detection (Cont.)

```
<style type="text/css">
    div {
        position: absolute;
        top: 30px;
        left: 50px;
    }
    #div1 img {
        filter:
        progid:DXImageTransform.Microsoft.AlphaImageLoader(src=javascript.jpg,
        sizingMethod='scale');
    }
</style>
<script type="text/javascript">
    window.onload=function() {
        document.getElementById("div1").onclick=getSrc;
    }
    function getSrc(evnt) {
        var theEvent = evnt ? evnt : window.event;
        var theSrc = theEvent.target ? theEvent.target : theEvent.srcElement;
        alert(theSrc.src);
    }
</script>
<body>
    <div id="div1">
        
    </div>
</body>
```

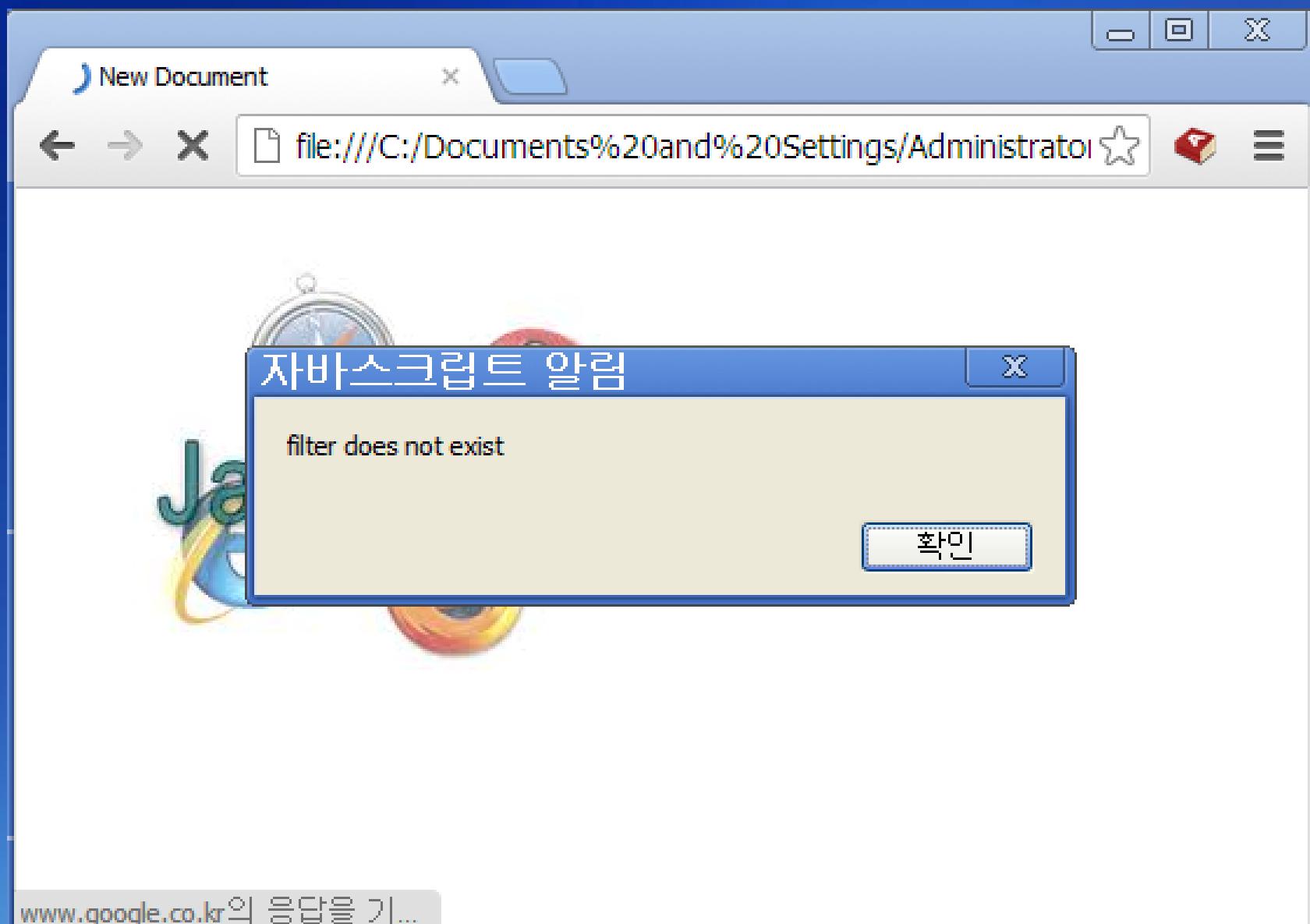
# Object Detection (Cont.)



# Object Detection (Cont.)

```
<script type="text/javascript">
window.onload=function() {
    var divElement = document.getElementById("div1");
    var tst1 = (divElement.style.filter) ? "filter exists" : "filter does not exist";
    alert(tst1);
    var tst2 = (typeof(divElement.style.filter) !== "undefined") ? "filter exists" : "filter does not exist";
    alert(tst2);
    var tst3 = (divElement.style.filter !== undefined) ? "filter exists" : "filter does not exist";
    alert(tst3);
    var tst4 = (divElement.style.opacity) ? "opacity exists" : "opacity does not exist";
    alert(tst4);
    var tst5 = (typeof(divElement.style.opacity) !== "undefined") ? "opacity exists" : "opacity does not exist";
    alert(tst5);
}
</script>
```

# Object Detection (Cont.)



# Test Your Knowledge : Quiz

1. Write the code to check to see what the value of a variable is, without recourse to browser-specific debugger.

# Test Your Knowledge : Quiz

1. Write the code to check to see what the value of a variable is, without recourse to browser-specific debugger.

```
// test some variable
alert(firstName);
// value will either be set, null, or undefined
```

# Test Your Knowledge : Quiz (Cont.)

2. The CSS attribute `text-shadow` has an interesting history. It was added to CSS2, but no browser implemented it, and it was removed in CSS 2.1. However, then browsers implemented it, and now it's being added into CSS3. At the time of this writing, this attribute was implemented in two of our four target browsers. Can you use object detection to successfully test for this style attribute? Write cross browser code that sets this CSS attribute for an existing header element.

# Test Your Knowledge : Quiz (Cont.)

2. The CSS attribute `text-shadow` has an interesting history. It was added to CSS2, but no browser implemented it, and it was removed in CSS 2.1. However, then browsers implemented it, and now it's being added into CSS3. At the time of this writing, this attribute was implemented in two of our four target browsers. Can you use object detection to successfully test for this style attribute? Write cross browser code that sets this CSS attribute for an existing header element.

```
var headerElement = document.getElementById("pageHeader");  
headerElement.style.textShadow="#ff0000 2px 2px 3px";
```