

App Development & Modeling

BSc in Applied Computing

Produced
by

Eamonn de Leastar (edelestar@wit.ie)

Department of Computing, Maths & Physics
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRCE

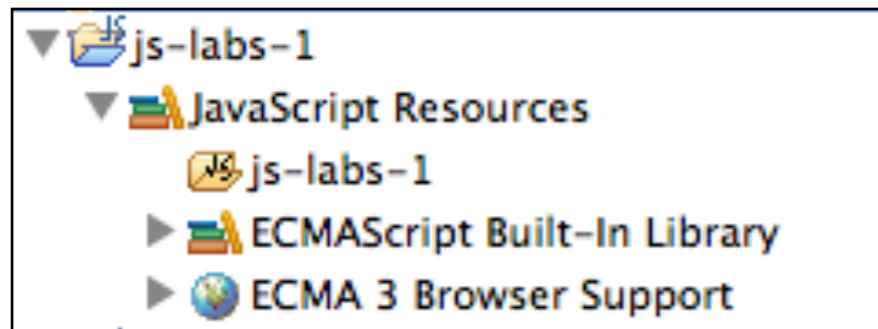


Javascript Part 1a

A Web Page with HTML, CSS & Javascript

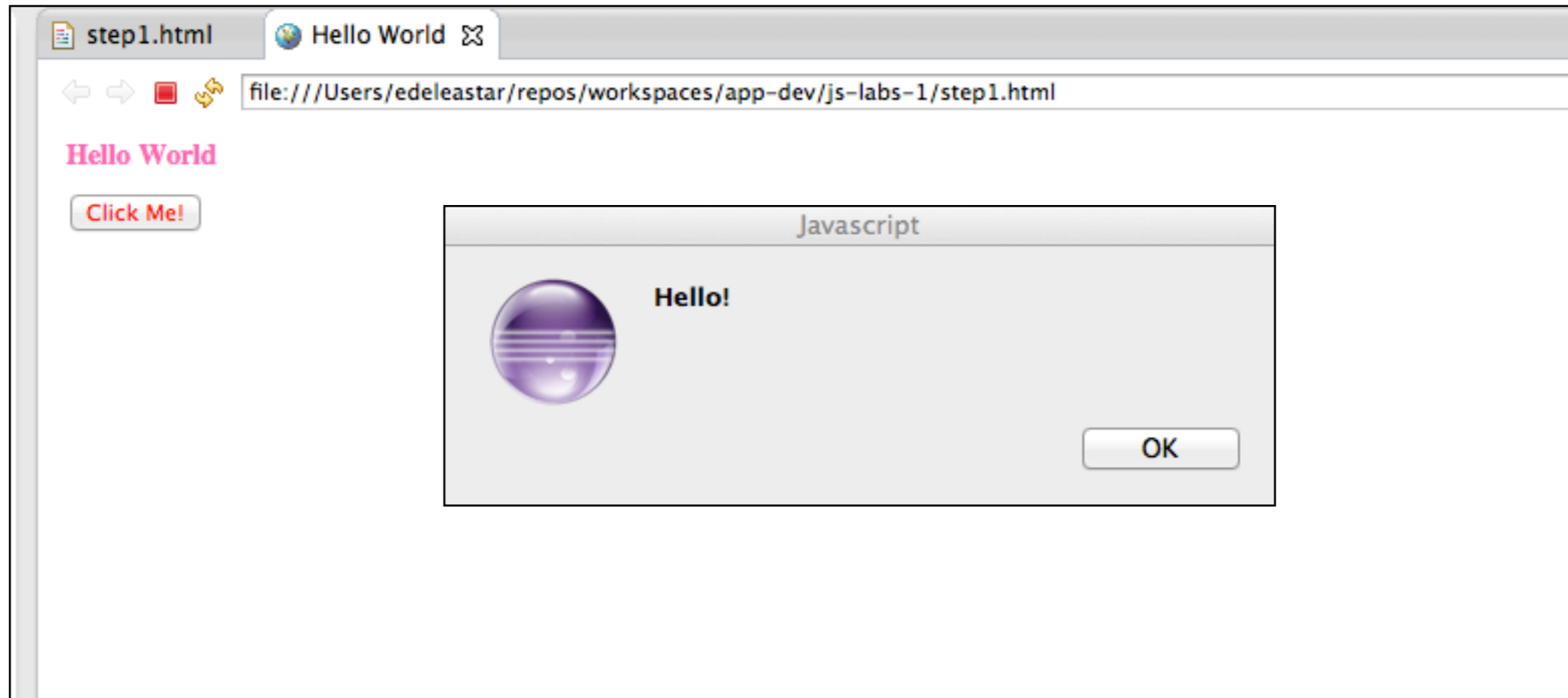
```
<html>
<head>
  <title>Hello World</title>
  <!-- CSS for presentation -->
  <style type="text/css">
    h1 { font-size: 14px; color: hotpink; }
    button { color: red; }
  </style>
  <!-- JavaScript for interactivity -->
  <script type="text/javascript">
    function buttonClick()
    {
      alert("Hello!");
    }
  </script>
</head>
<body>
  <h1>Hello World</h1>
  <button onClick="buttonClick();">Click Me!</button>
</body>
</html>
```

Eclipse Project

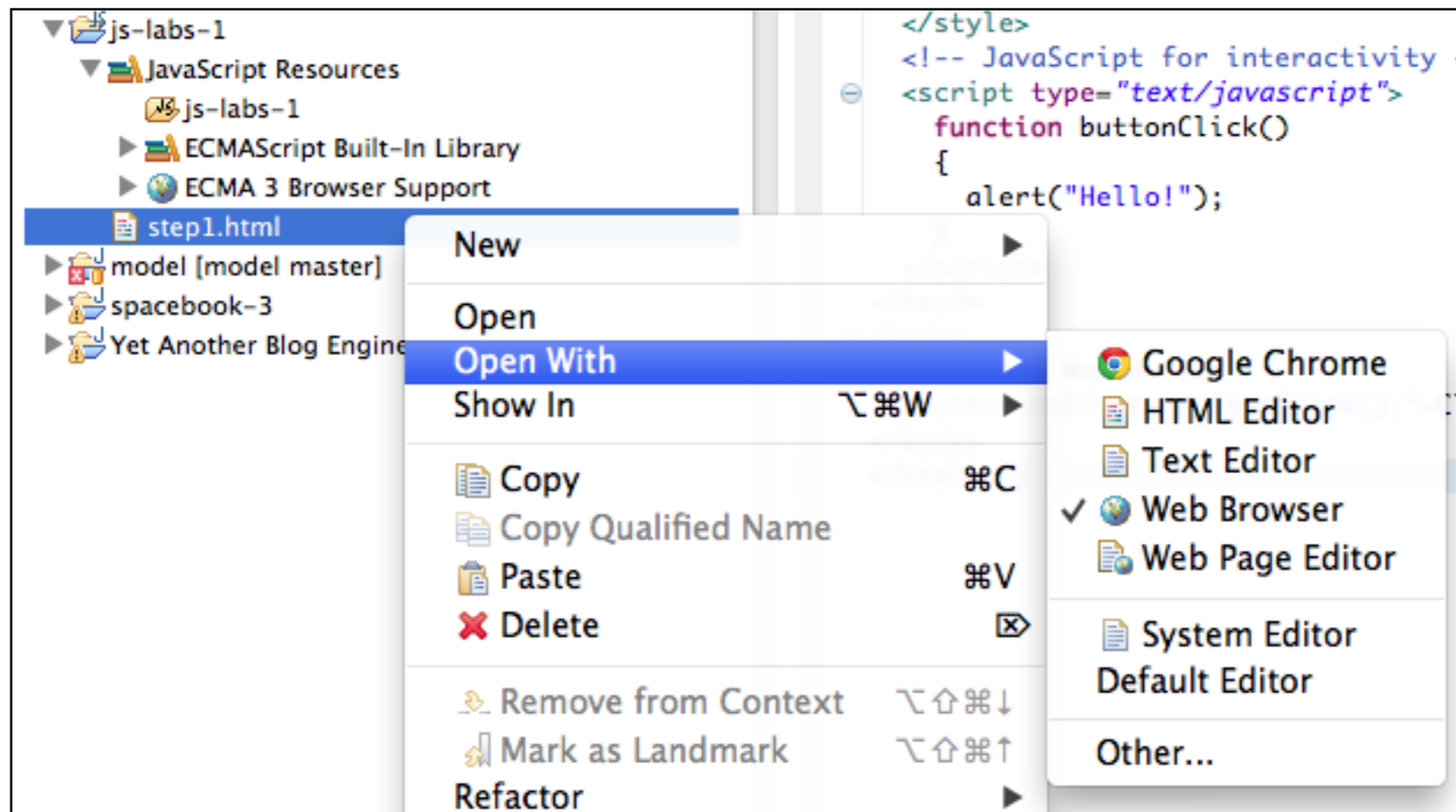


```
step1.html Hello World
<html>
<head>
  <title>Hello World</title>
  <!-- CSS for presentation -->
  <style type="text/css">
    h1 { font-size: 14px; color: hotpink; }
    button { color: red; }
  </style>
  <!-- JavaScript for interactivity -->
  <script type="text/javascript">
    function buttonClick()
    {
      alert("Hello!");
    }
  </script>
</head>
<body>
  <h1>Hello World</h1>
  <button onClick="buttonClick();">Click Me!</button>
</body>
</html>
```

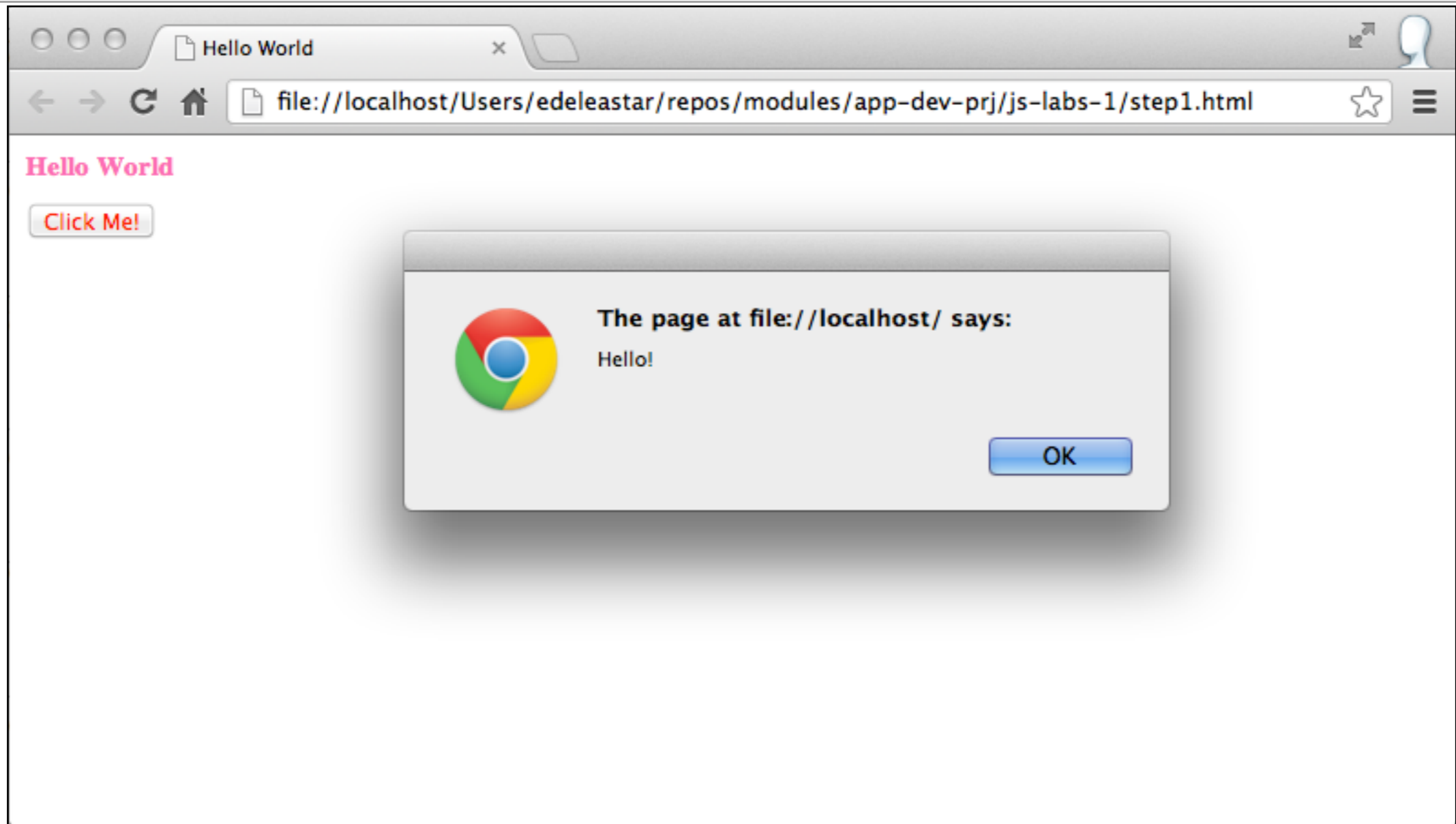
Eclipse Project - Browser



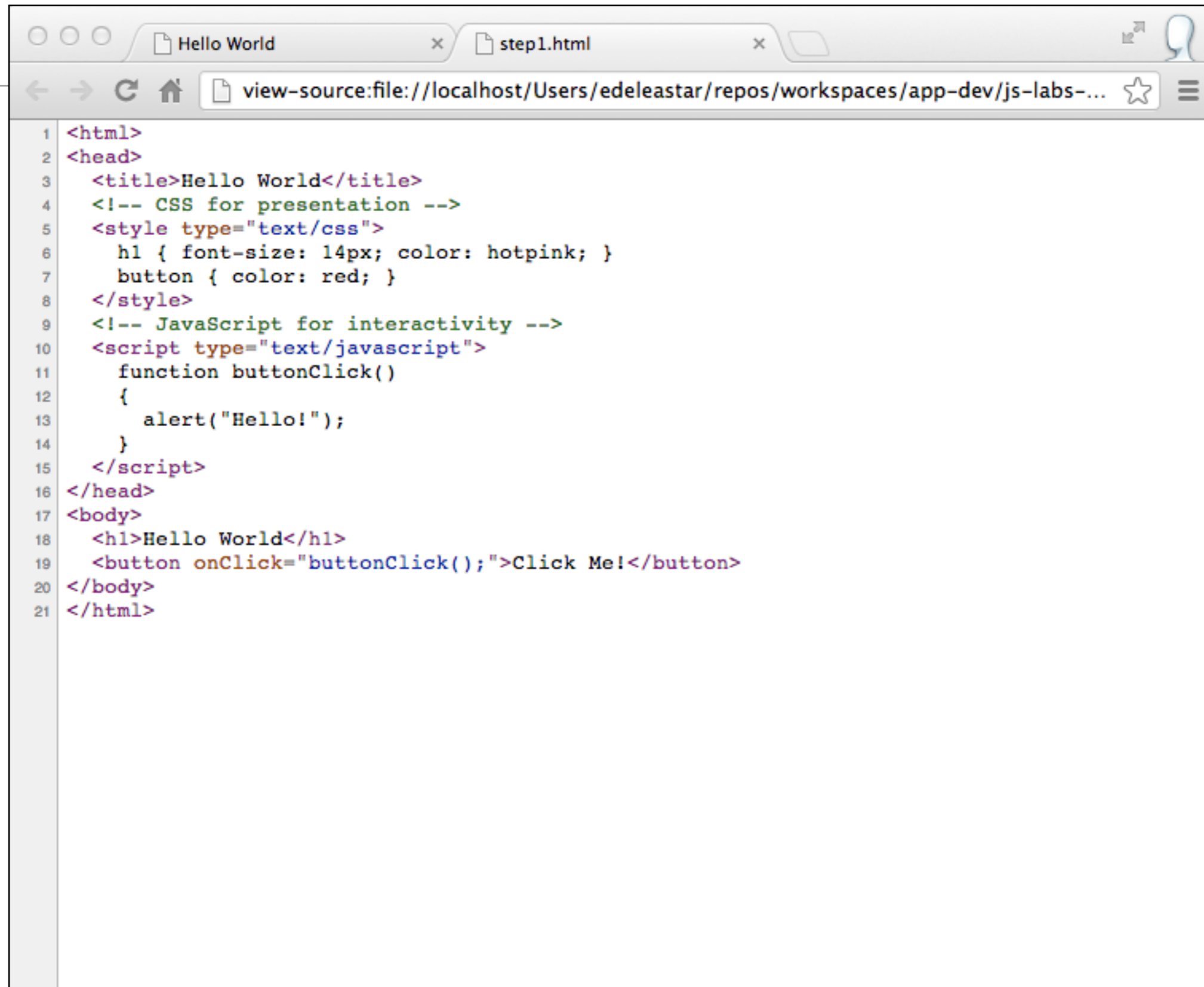
Associate html file with Google Chrome



- Change Eclipse Settings to load Chrome instead of build in editor...



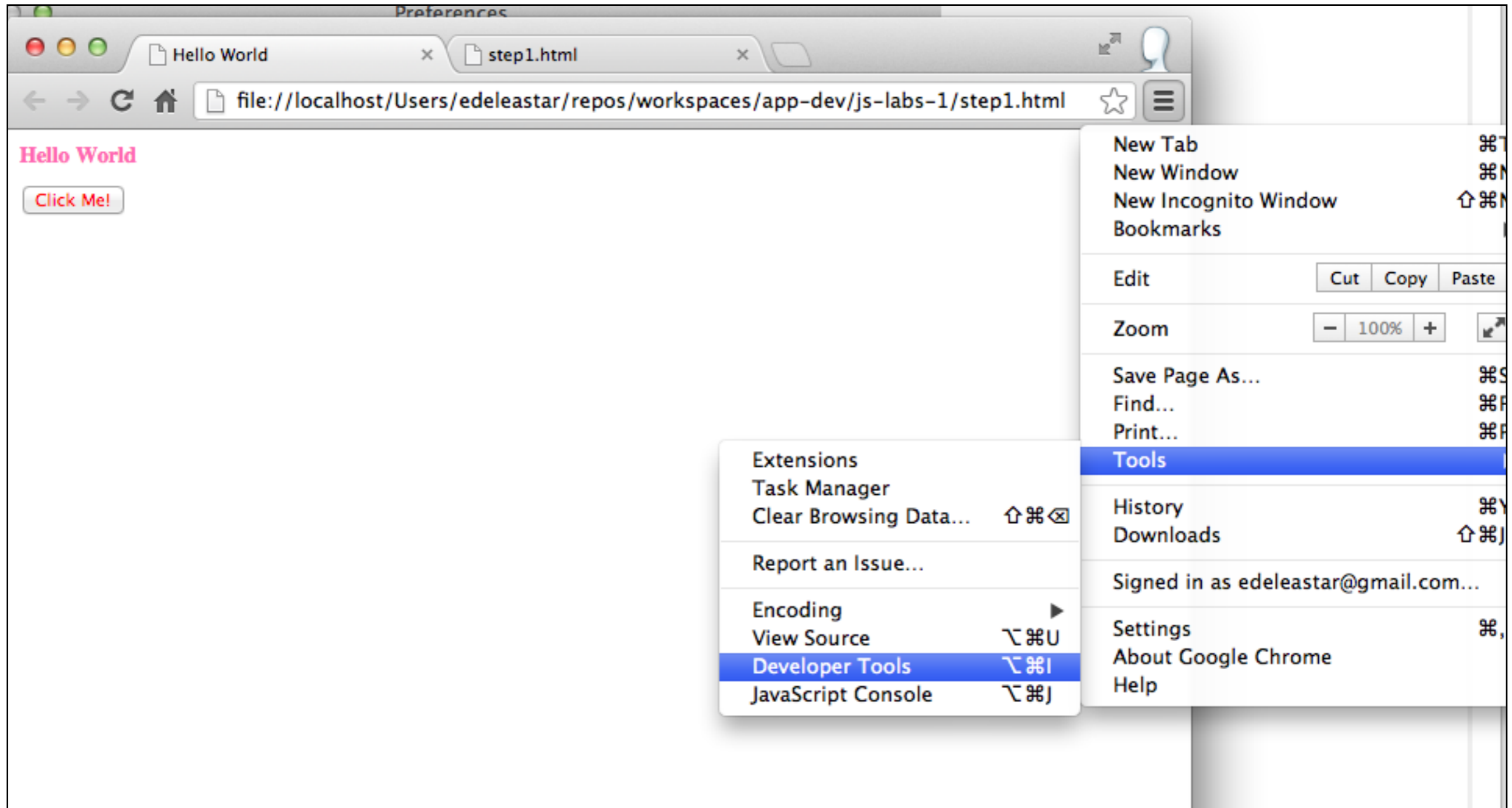
View->Source

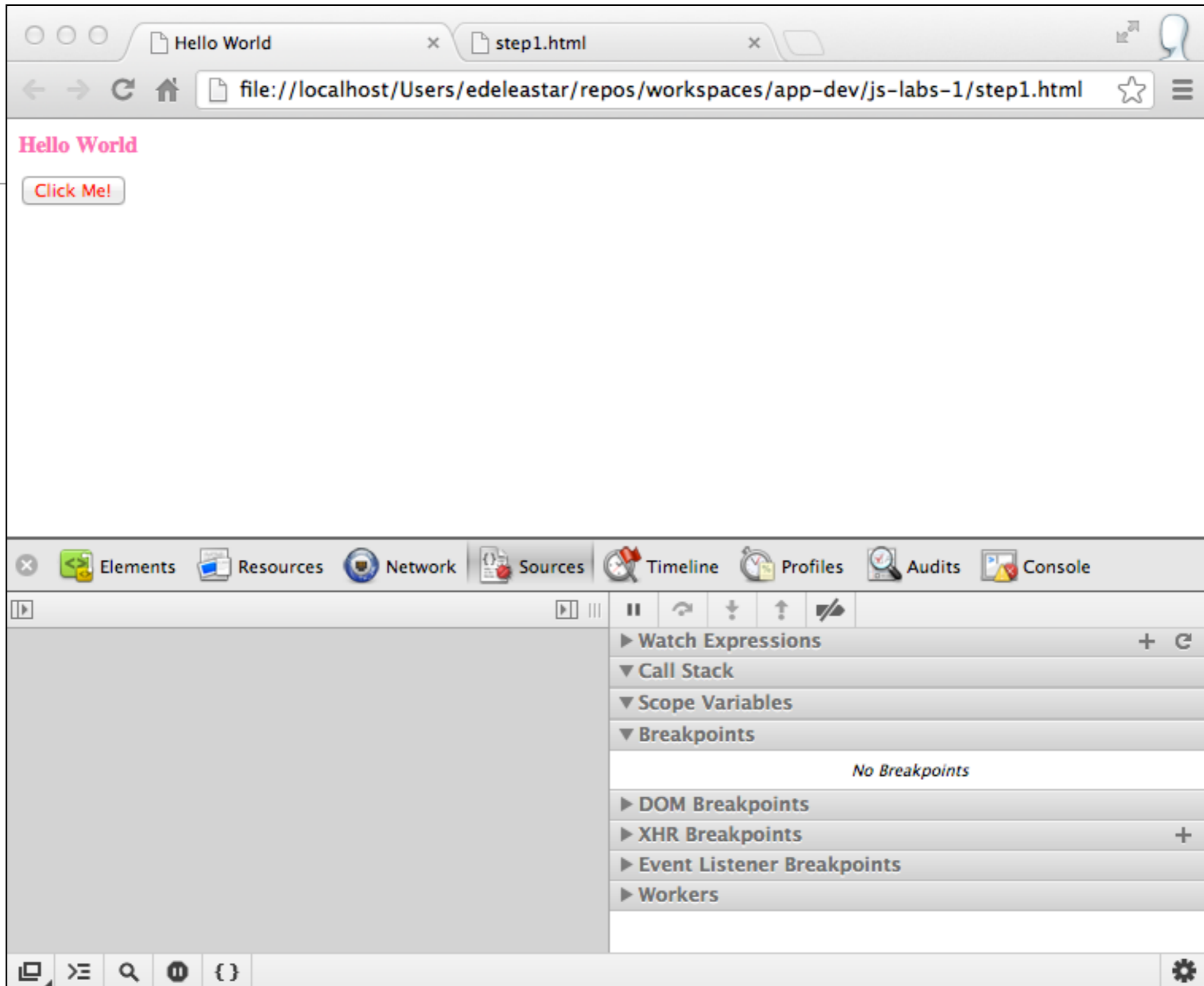


The image shows a web browser window with two tabs: 'Hello World' and 'step1.html'. The address bar shows the file path: 'view-source:file:///localhost/Users/edelestar/repos/workspaces/app-dev/js-labs-...'. The main content area displays the source code of the HTML document, which includes a title, CSS for styling, and JavaScript for interactivity. The code is as follows:

```
1 <html>
2 <head>
3   <title>Hello World</title>
4   <!-- CSS for presentation -->
5   <style type="text/css">
6     h1 { font-size: 14px; color: hotpink; }
7     button { color: red; }
8   </style>
9   <!-- JavaScript for interactivity -->
10  <script type="text/javascript">
11    function buttonClick()
12    {
13      alert("Hello!");
14    }
15  </script>
16 </head>
17 <body>
18   <h1>Hello World</h1>
19   <button onClick="buttonClick();">Click Me!</button>
20 </body>
21 </html>
```


View->Developer Tools

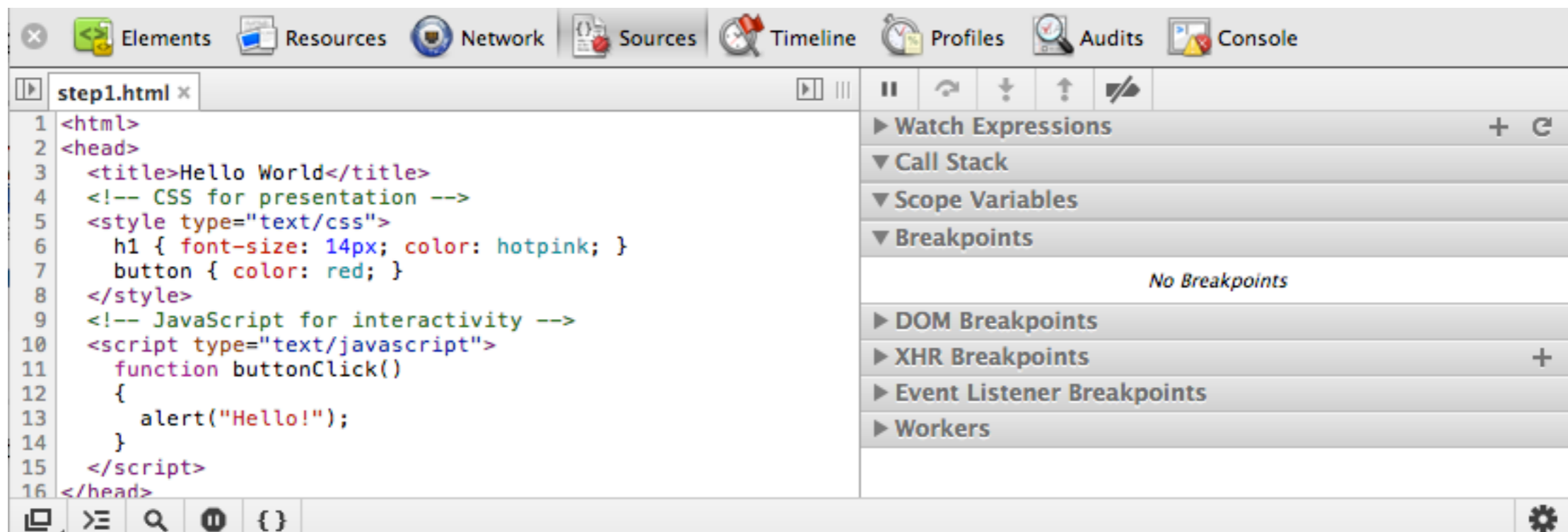




The image shows a web browser window with a single tab titled "Hello World". The address bar displays the file path: `file:///localhost/Users/edeleastar/repos/workspaces/app-dev/js-labs-1/step1.html`. The page content includes the text "Hello World" in pink and a button labeled "Click Me!".

Below the page content is the Chrome DevTools interface. The "Sources" panel is active, showing a tree view of the file system with "step1.html" selected. The right-hand pane of the Sources panel contains a list of debugging features: Watch Expressions, Call Stack, Scope Variables, Breakpoints (with "No Breakpoints" displayed), DOM Breakpoints, XHR Breakpoints, Event Listener Breakpoints, and Workers. A blue arrow points from the left side of the image to the "Sources" tab in the DevTools toolbar.

Source view in Developer Tools



Placement of Script - top

```
<!--Attempting to access an element too early will have  
unexpected results.-->  
<!doctype html>  
<html>  
<head>  
<script type="text/javascript">  
  
    var title = document.getElementById("hello-world");  
    console.log( title );  
  
</script>  
</head>  
<body>  
    <h1 id="hello-world">Hello World</h1>  
</body>  
</html>
```

- If the code will interact with the elements on the page, you have to make sure those elements exist at the time the script is executed.
- This common pitfall can be seen in the example above.
- The script for finding the element with the ID "hello-world" will be executed before the element is defined in the document.

The image shows a web browser window with three tabs: 'step2.html', 'step2a.html', and 'step2b.html'. The address bar shows a file path: 'file:///localhost/Users/edelestar/repos/modules/app-development-modeling-2013/p...'. The main content area displays 'Hello World' in a large, bold, black serif font. Below the content area is a developer tools toolbar with icons for Elements, Resources, Network, Sources, Timeline, Profiles, Audits, and Console. The Console panel is active, showing a single log entry: 'null' at 'step2a.html:8'. At the bottom of the browser window, there is a status bar with icons for zooming, a dropdown menu showing '<top frame>', and filters for 'All', 'Errors', 'Warnings', and 'Logs'. A gear icon for settings is on the far right.

Placement - Page end

```
<!--Moving the script to the bottom of the page will make
sure the element exists.-->
<!doctype html>
<html>
<head>
</head>
<body>
  <h1 id="hello-world">Hello World</h1>
  <script type="text/javascript">

      var title = document.getElementById("hello-world");
      console.log( title );

  </script>
</body>
</html>
```

- It is a common pattern to move scripts to the bottom of the page, prior to the closing HTML `<body>` tag. This will guarantee that elements are defined when the script is executed.

The image shows a web browser window with three tabs: 'step2.html', 'step2a.html', and 'step2b.html'. The address bar shows a file path: 'file:///localhost/Users/edelestar/repos/modules/app-development-modeling-2013/p...'. The main content area displays 'Hello World' in a large, bold, black serif font. Below the content area is the developer tools interface, which includes a toolbar with icons for Elements, Resources, Network, Sources, Timeline, Profiles, Audits, and Console. The Console panel is active, showing the source code: `<h1 id="hello-world">Hello World</h1>` with the file name 'step2b.html:12' on the right. At the bottom of the developer tools, there is a filter menu set to 'All' and buttons for 'Errors', 'Warnings', and 'Logs'.

Comments

```
// Single and multi line comments.  
// this is an example of a single line comment.  
  
/*  
 * this is an example  
 * of a  
 * multi line  
 * comment.  
 */
```

- Similar Rules to Java

Whitespace

- Whitespace is also ignored in JavaScript.
- There are many tools that will strip out all the whitespace in a program, reducing the overall file size and improving network latency.
- Given the availability of tools like these, whitespace should be leveraged to make the code as readable as possible.

```
// Whitespace is insignificant.  
var hello = "Hello";  
  
var world    =    "World!";
```

Reserved Words

- `break`
- `case`
- `catch`
- `continue`
- `debugger`
- `default`
- `delete`
- `do`
- `else`
- `finally`
- `for`
- `function`
- `if`
- `in`
- `instanceof`
- `new`
- `return`
- `switch`
- `this`
- `throw`
- `try`
- `typeof`
- `var`
- `void`
- `while`
- `with`

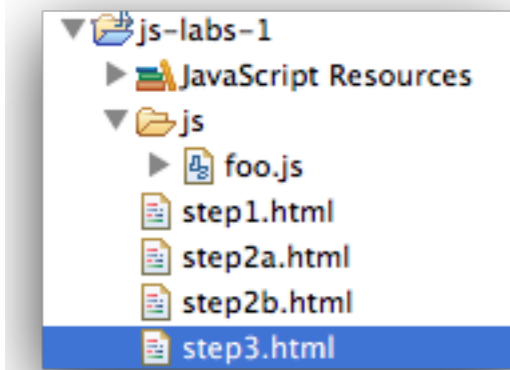
- Significant overlap with Java
- However, meaning often different in subtle ways

Identifiers

- Identifiers are used to give variables and functions a unique name so they can subsequently be referred to by that name.
- The name of an identifier must follow a few rules:
 - Cannot be a reserved word.
 - Can only be composed of letters, numbers, dollar signs, and underscores.
 - The first character cannot be a number.

```
// Valid identifier names.  
var myAwesomeVariable = "a";  
var myAwesomeVariable2 = "b";  
var my_awesome_variable = "c";  
var $my_AwesomeVariable = "d";  
var _my_awesome_variable_$ = "e";
```

Running a Program

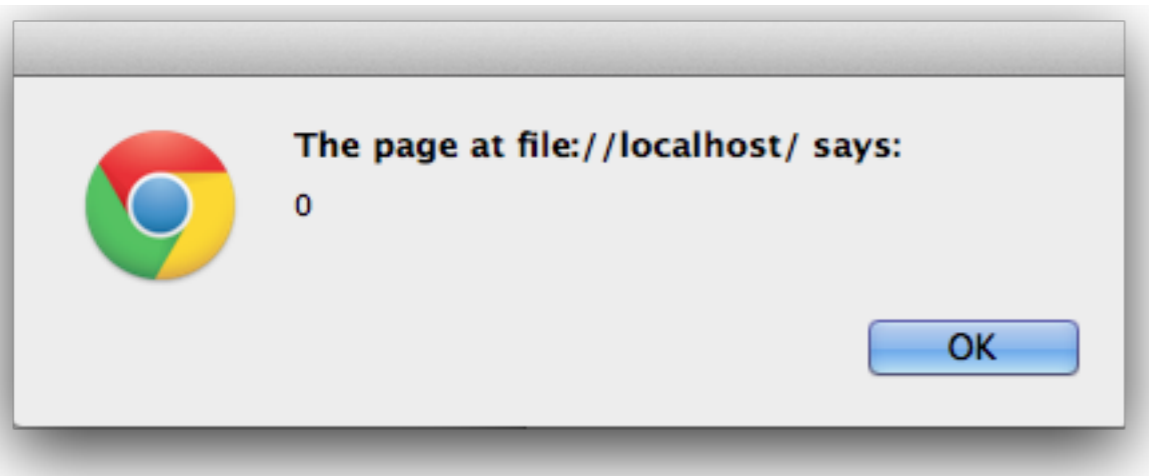


```
<!doctype html>
<html>
<head>
  <script src="js/foo.js"></script>
</head>
<body>

  <h1 id="hello-world">Hello World</h1>

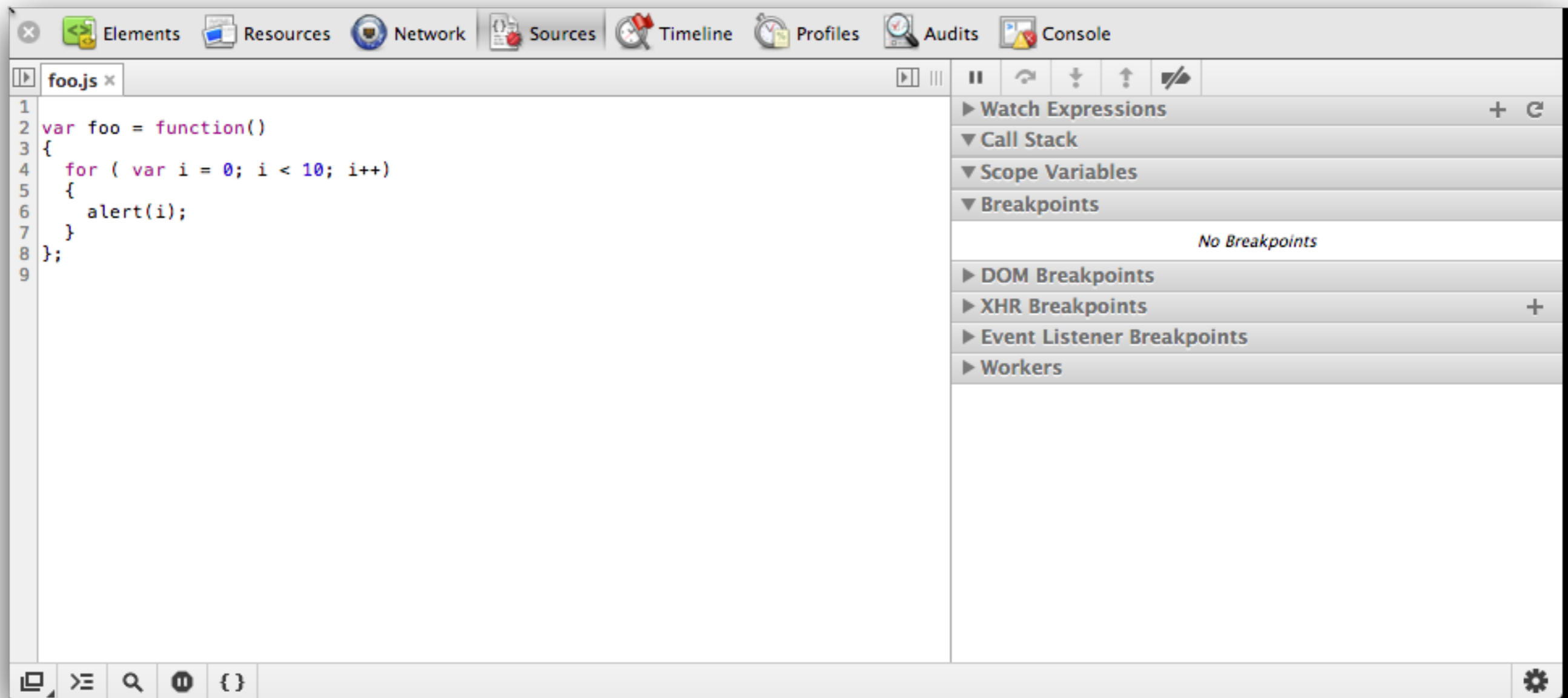
  <script type="text/javascript">
    foo();
  </script>

</body>
</html>
```



```
var foo = function()
{
  for ( var i = 0; i < 10; i++)
  {
    alert(i);
  }
};
```

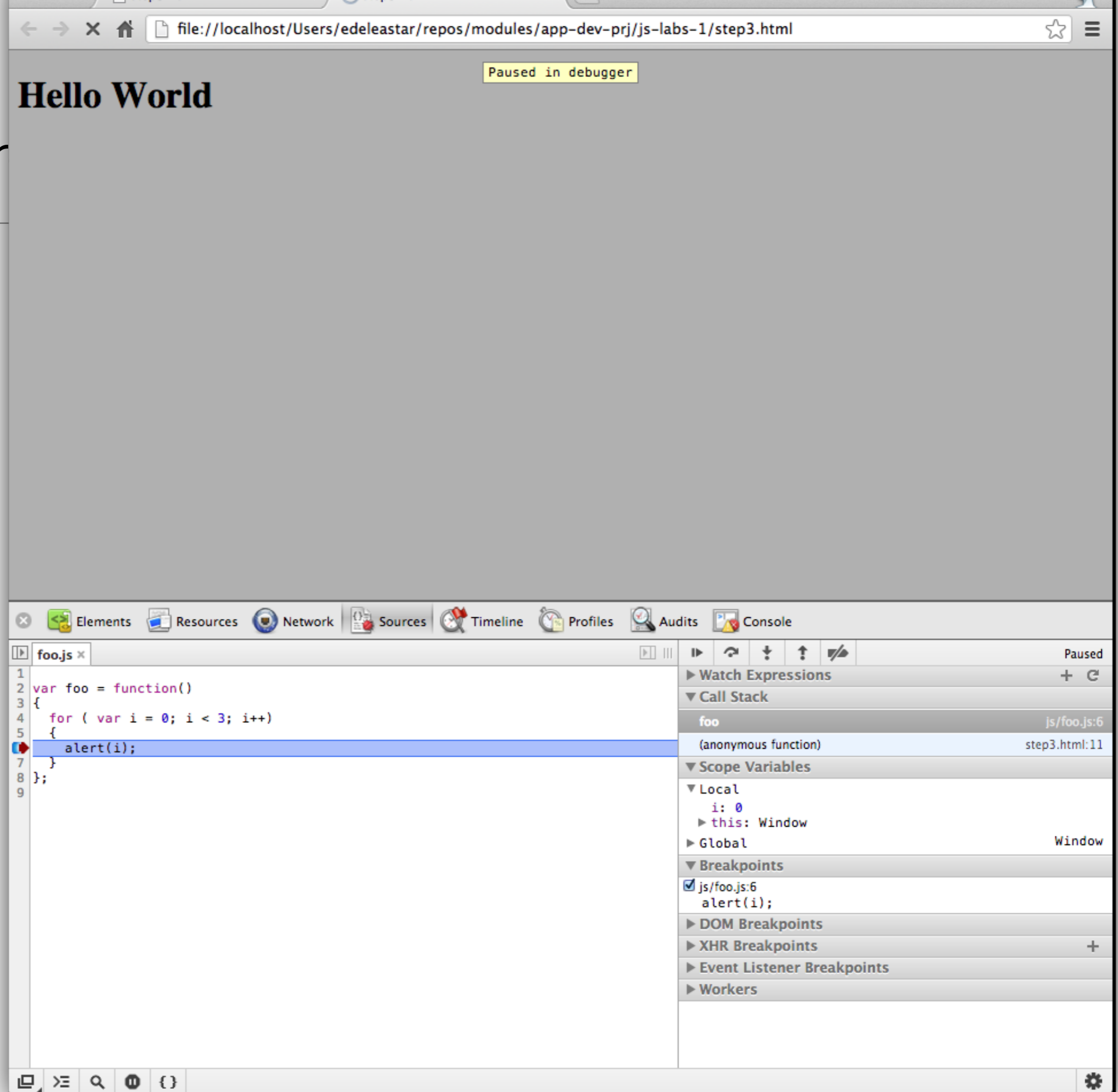
Chrome Developer Tools View



- In developer tools - and select "Sources". Press the "Navigator" button (small button on top left) and locate and display the foo.js file

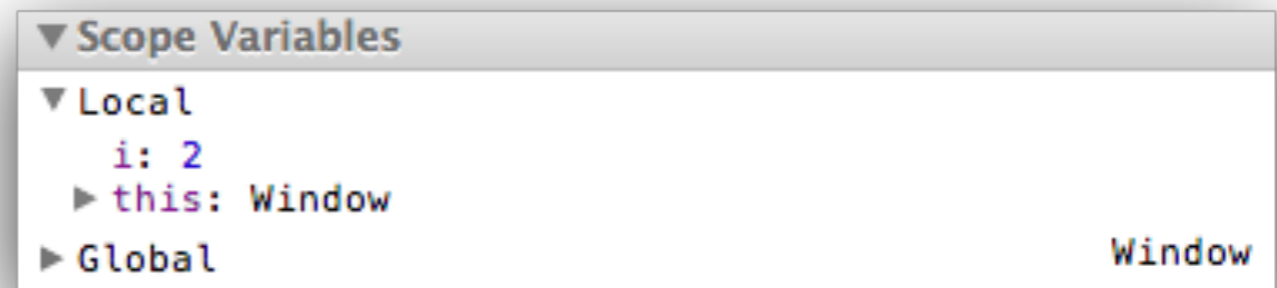
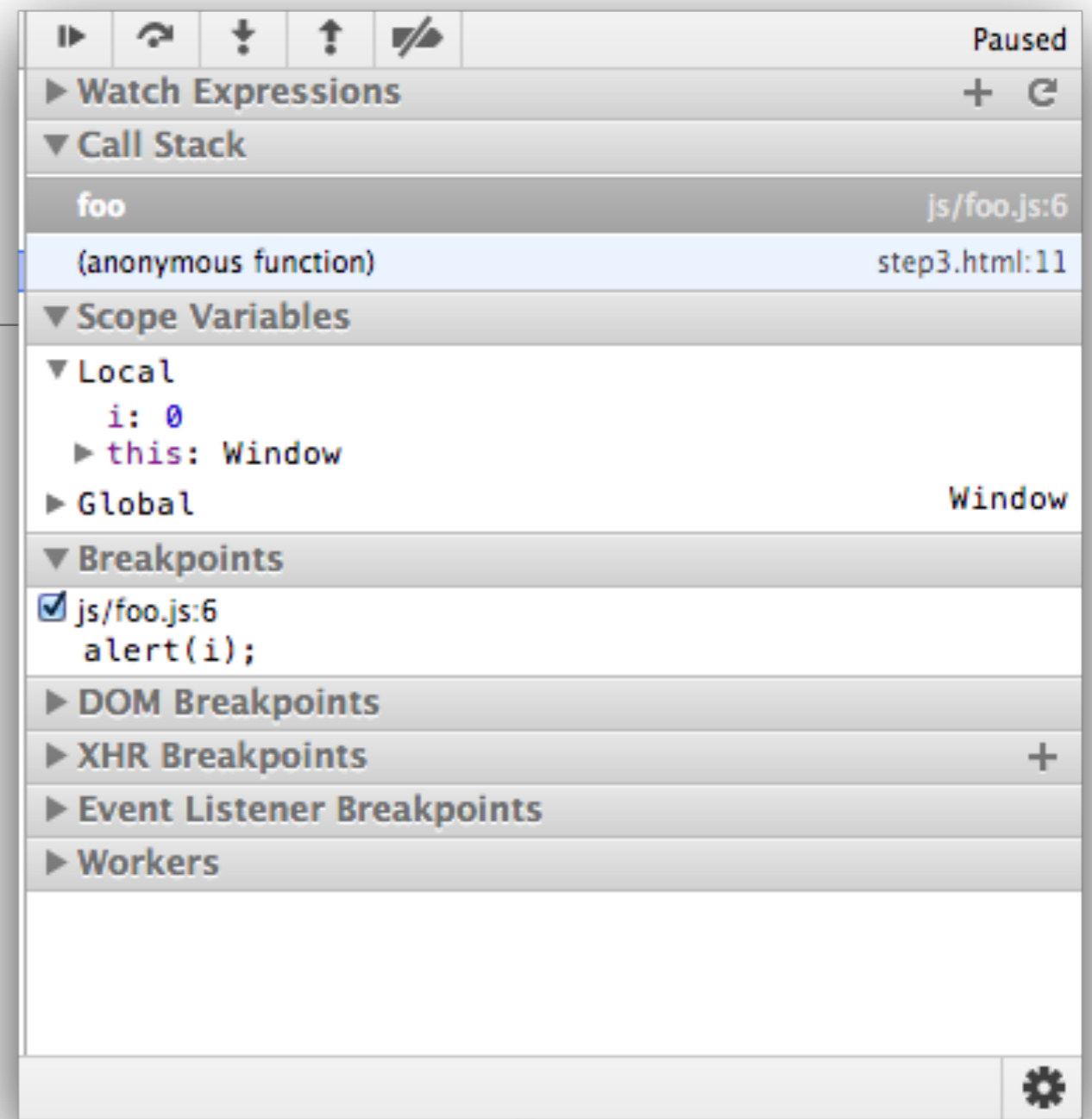
Setting Br

- In Chrome Sources view, click on margin alongside the alert line - this should place a marker as shown:



Viewing Variables

- With the javascript program paused - because the marker we set down above is a 'breakpoint'. This means the programs is waiting your command to resume.
- Hover on the 5 buttons along the top for a few seconds each - and read the tooltip.
- In particular, experiment with the 'step over..' and 'step into...' buttons. Monitor the "Scope Variables" panel while you are doing this:



Types

- Types in JavaScript fall into two categories: primitives or objects. Primitive types include:
 - String
 - Number
 - Boolean
 - Null
 - Undefined

Strings

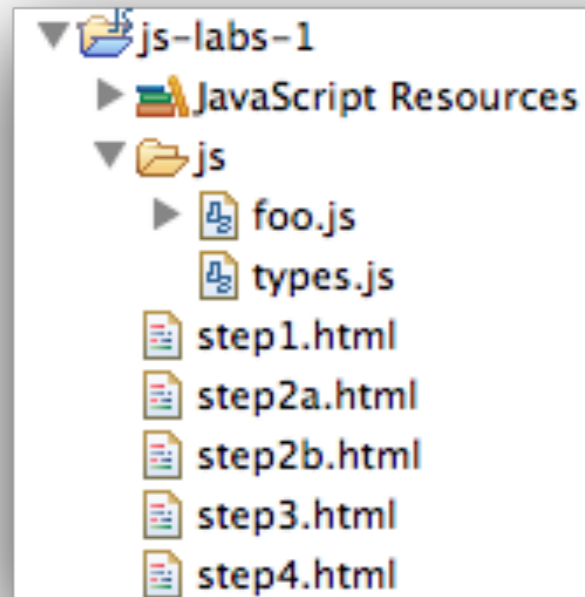
- Strings are text wrapped in single or double quotation marks.
- It is best practice to consistently use one or the other.
- There may be times when the string contains quotation marks that collide with the ones used to create the string.
- In this case, either escape the characters using a \ backslash or use different quotes around the string.

```
/ Strings can created with double or single quotes.
var a = "I am a string";
var b = 'So am I!';

alert( a );

alert( b );
// Sometimes a string may contain quotation marks.
var statement1 = 'He said "JavaScript is awesome!";
var statement2 = "He said \"JavaScript is awesome!\"";
```

Strings & Objects



```
<!doctype html>
<html>
  <head>
    <script src="js/types.js"></script>
  </head>
  <body>

    <h1 id="Hello Types">Hello World</h1>

  </body>
</html>
```

```
var a = "I am a string";
var b = 'So am I!';

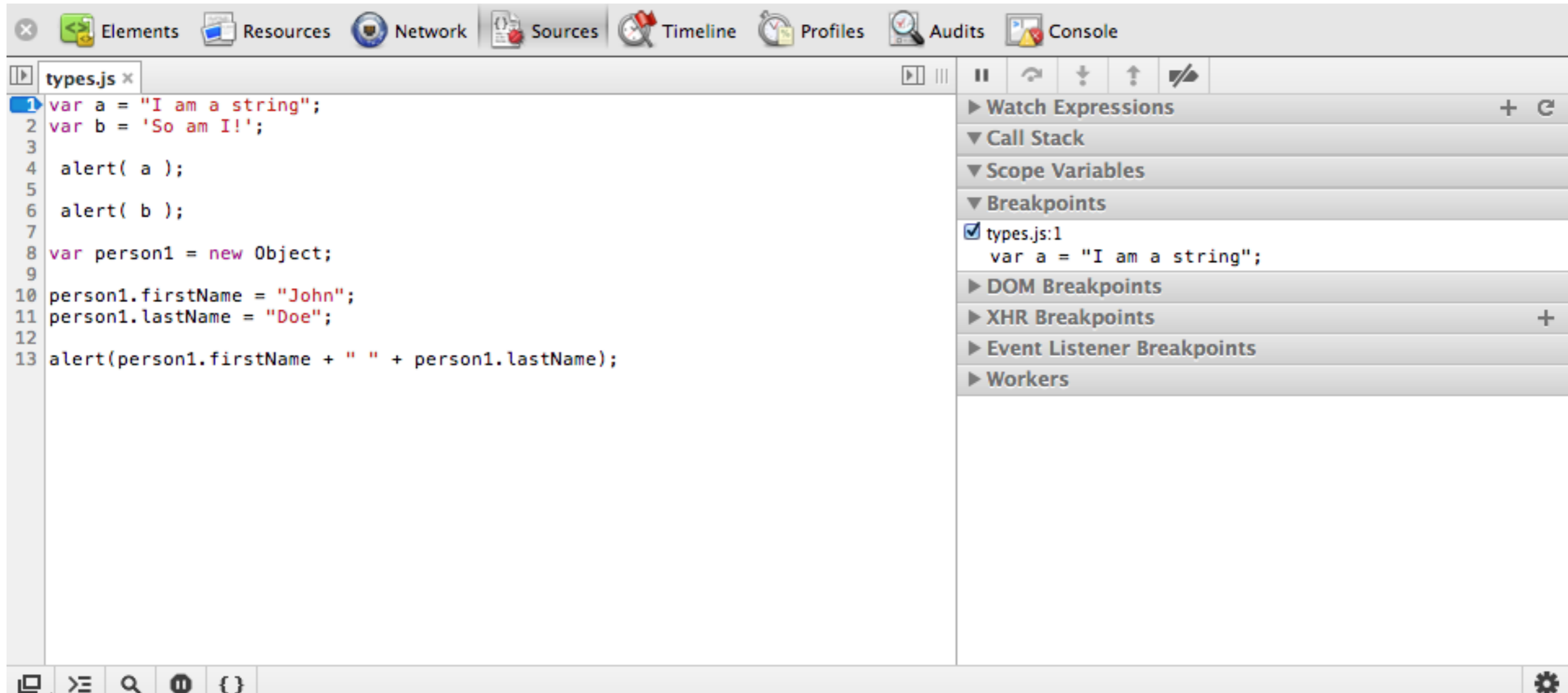
alert( a );
alert( b );

var person1 = new Object;

person1.firstName = "John";
person1.lastName = "Doe";

alert(person1.firstName + " " + person1.lastName);
```

- Opening the "Sources" tab and open the 'types.js' file and set a breakpoint (by clicking on the margin) on the second line:



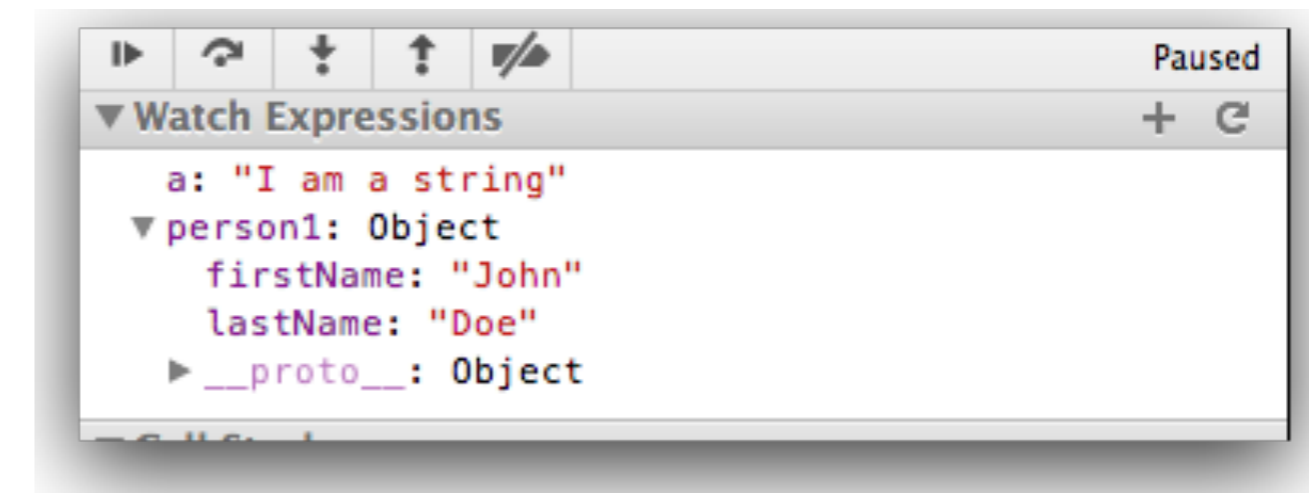
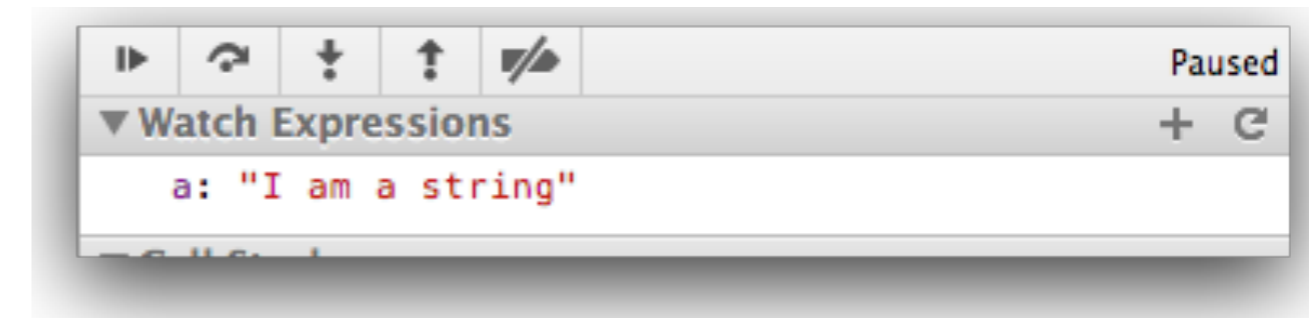
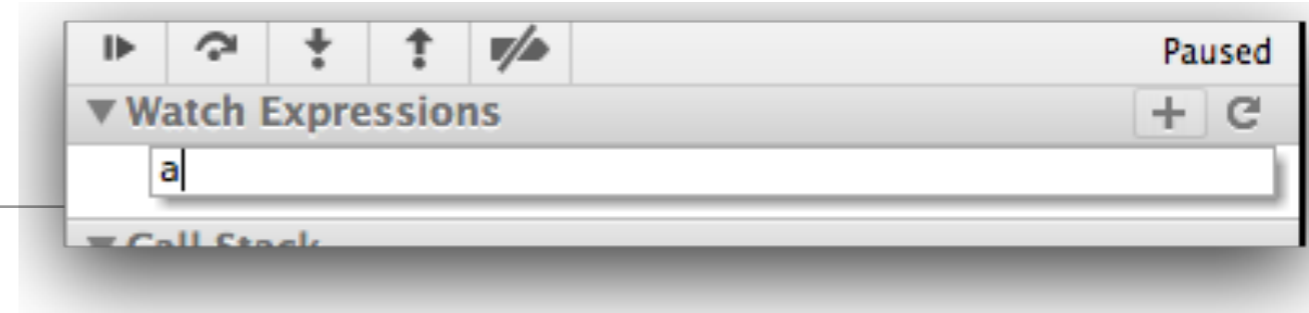
- Reload the page again note that you will be in 'debug' mode

The screenshot shows a web browser window with two tabs open, both titled 'step4.html'. The address bar shows the file path: `file:///localhost/Users/edelestar/repos/modules/app-dev-prj/js-labs-1/step4.html`. A yellow box in the center of the browser page says 'Paused in debugger'. Below the browser window is the Chrome DevTools interface. The 'Sources' panel is active, showing a file named 'types.js' with the following code:

```
1 var a = "I am a string";
2 var b = 'So am I!';
3
4 alert( a );
5 alert( b );
6
7 var person1 = new Object();
8
9 person1.firstName = "John";
10 person1.lastName = "Doe";
11
12 alert(person1.firstName + " " + person1.lastName);
```

The 'Scope Variables' panel on the right shows the 'Global' scope with various built-in objects and functions, including 'Infinity', 'Array', 'ArrayBuffer', 'Attr', 'Audio', 'AudioProcessingEvent', 'BeforeLoadEvent', 'Blob', 'Boolean', 'CDATASection', 'CSSCharsetRule', 'CSSFontFaceRule', 'CSSImportRule', 'CSSMediaRule', 'CSSPageRule', 'CSSPrimitiveValue', 'CSSRule', 'CSSRuleList', and 'CSSStyleDeclaration'.

- Single step through the lines and observe.
- The "Scope Variables" view is not much use here. Instead select locate the 'Watch Expressions' and press the "+" button:
- enter the name of a variable - 'a' in this instance - and press return:
- Experiment with the debug buttons - particularly the 'Step over' and 'Step into' buttons.
- See if you can monitor the 'person1' object - you should be able to view it's contents something like this:
- You can restart the 'program' at any stage by reloading the page in Chrome.



Numbers

- Number types are any positive or negative numeric value. There is no distinction between integer and floating point val

```
// Numbers are any whole or floating point integer.  
var num1 = 100;  
var num2 = 100.10;  
var num3 = 0.10;
```

Boolean

- Boolean types are either true or false

```
// Boolean values.  
var okay = true;  
var fail = false;
```


Null and Undefined

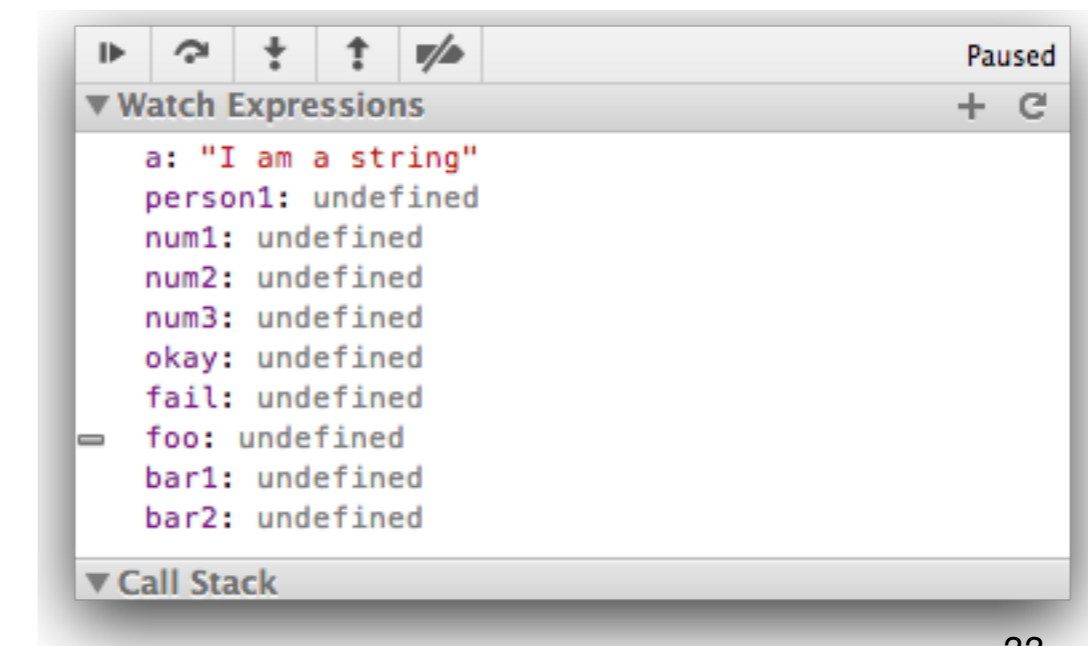
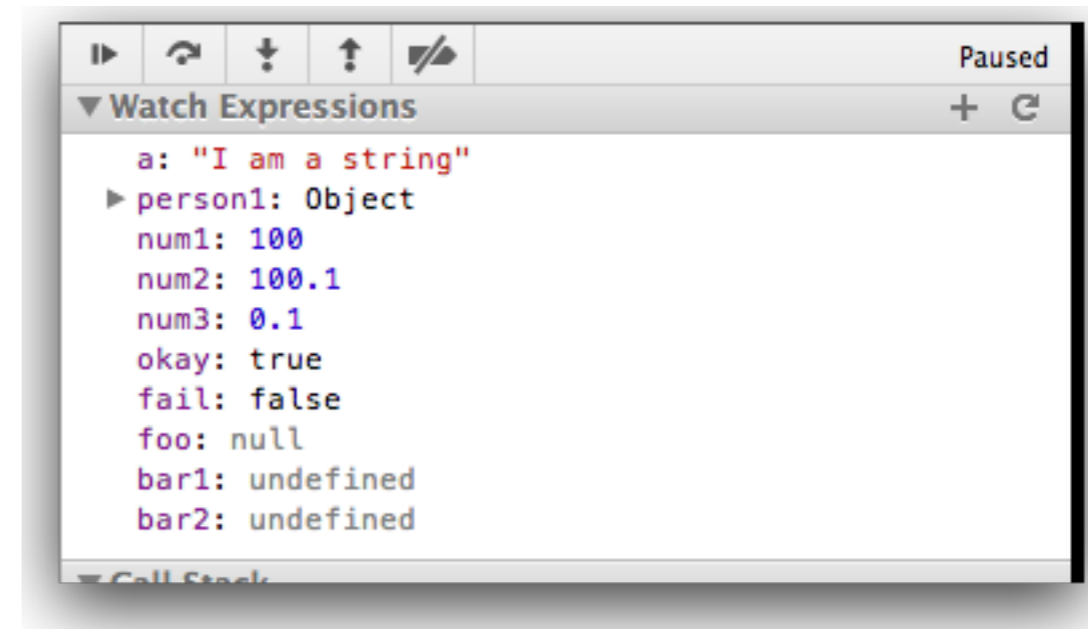
- Null and undefined are special types in JavaScript.
- Null types are a value that represent the absence of a value.
- Undefined types represent a state in which no value has been assigned at all.
- This type is created in two ways:
 - by using the undefined keyword
 - or by not defining a value at all.

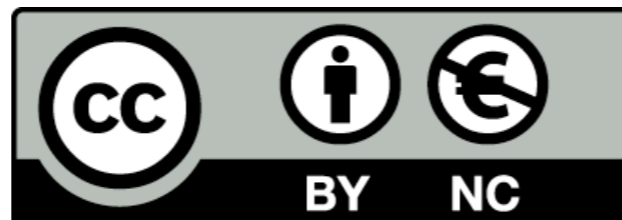
```
// Two ways to achieve an undefined value.
```

```
var foo = null;
```

```
var bar1 = undefined;
```

```
var bar2;
```





Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

