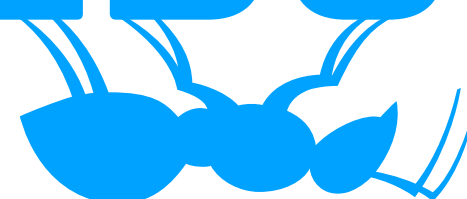
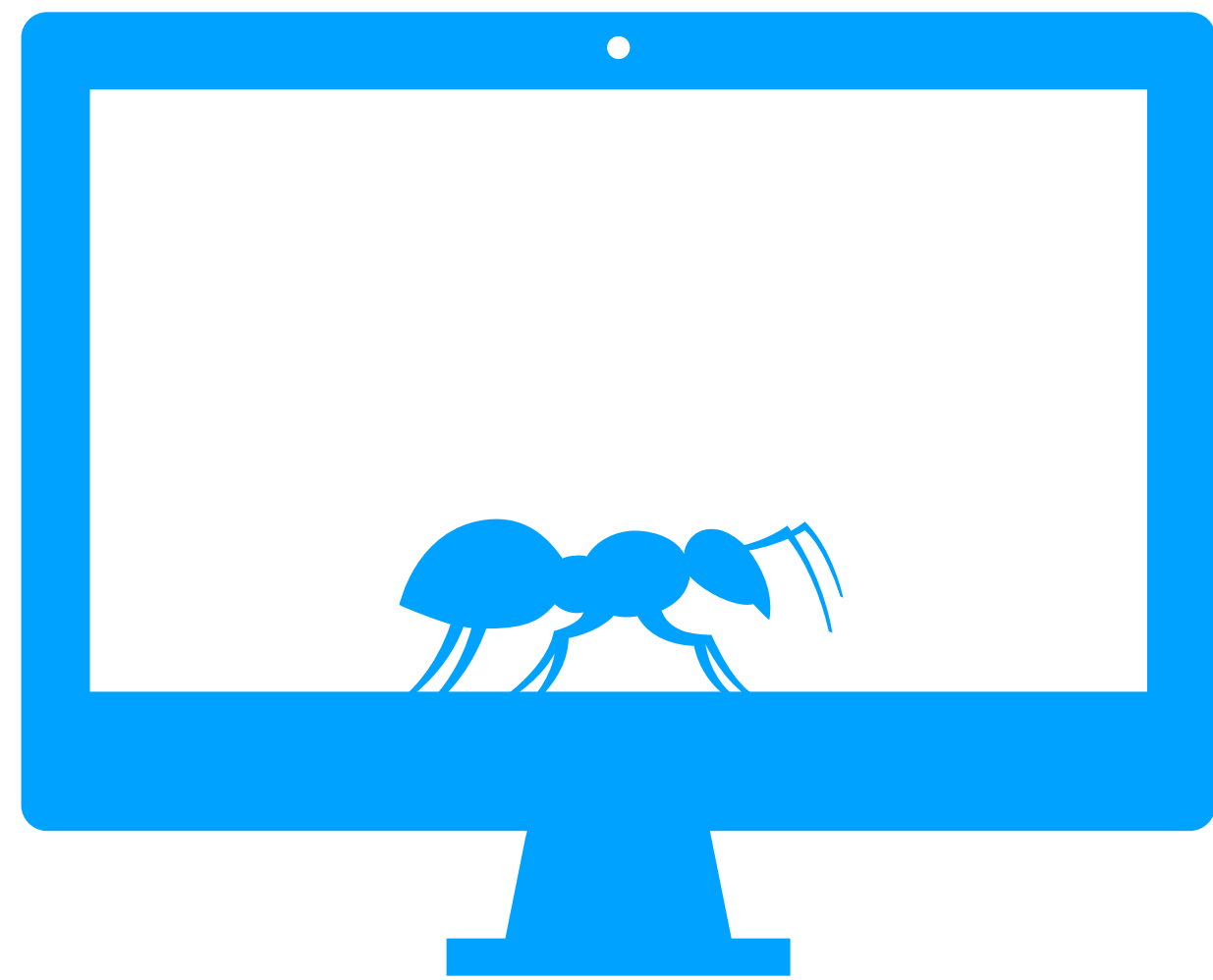
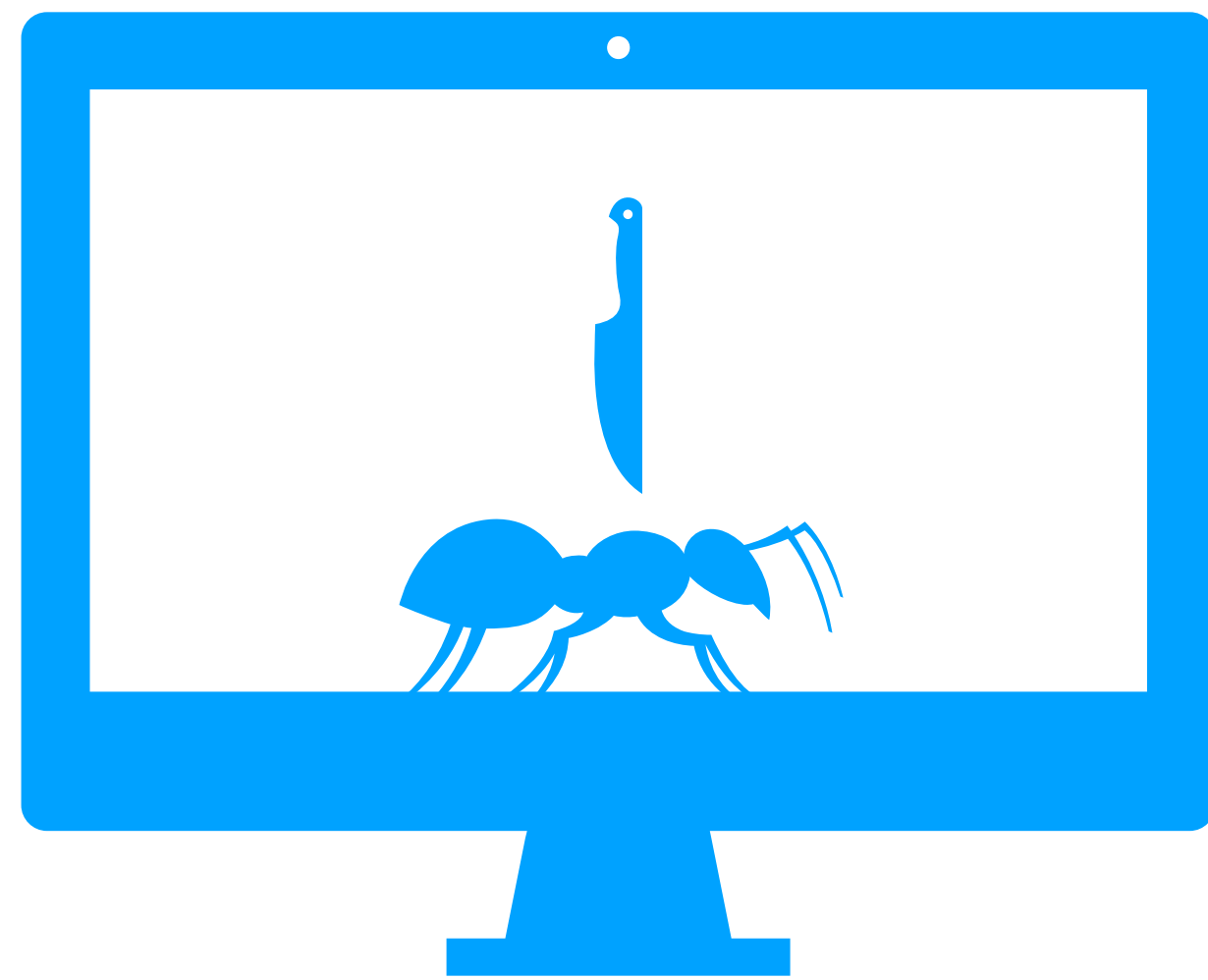


DEBUGGING





Every programmer
maeks **mistakes**
when they code.



Part of being a better programmer is learning how to **find** and **fix** your mistakes.

“Flaws in programs are usually called bugs. Bugs can be programmer errors or problems in other systems that the program interacts with. Some bugs are immediately apparent, while others are subtle and might remain [hidden](#) in a system for years.”

Eloquent JavaScript (2E), p.139

[Wikipedia entry on the [Y2K bug](#)]



“The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and it is then that ‘**Bugs**’—as such **little faults and difficulties** are called—show themselves and months of intense watching, study, and labor are requisite before commercial success or failure is certainly reached.”

— *Thomas Edison, game programmer, in 1878*

**DEBUGGING IS SEARCHING FOR
(AND FIXING) ERRORS IN CODE.**



Bugs happen because humans make mistakes, software is complex, and **computers do exactly what you tell them to do**—even when what you tell them is to do is incorrect.

(But bugs can also be fun, interesting, and useful.)

Q: HOW DO YOU DEBUG?


**HERE ARE SOME DEBUGGING
TIPS AND BEST PRACTICES I'VE
LEARNED OVER THE YEARS.**


DEBUGGING TIP #1


UNDERSTAND YOUR TOOLS. THEN USE THEM.





Q:
**HOW DO YOU DETECT
AND DIAGNOSE A CAR
PROBLEM *BEFORE* IT
LEAVES YOU STRANDED?**



mazda


 Read this first


 Quick Guide


 Visual Search


 Search by Theme


 Index


 Search by Switch/Button


 Warning/Indicator Light List

 Mazda Connect

 FAQ
(Frequently Asked Questions)

 Bookmarks

 Inquiries



 View PDF format



[Return](#)


Mazda CX-5


Web Owner's Manual


Search














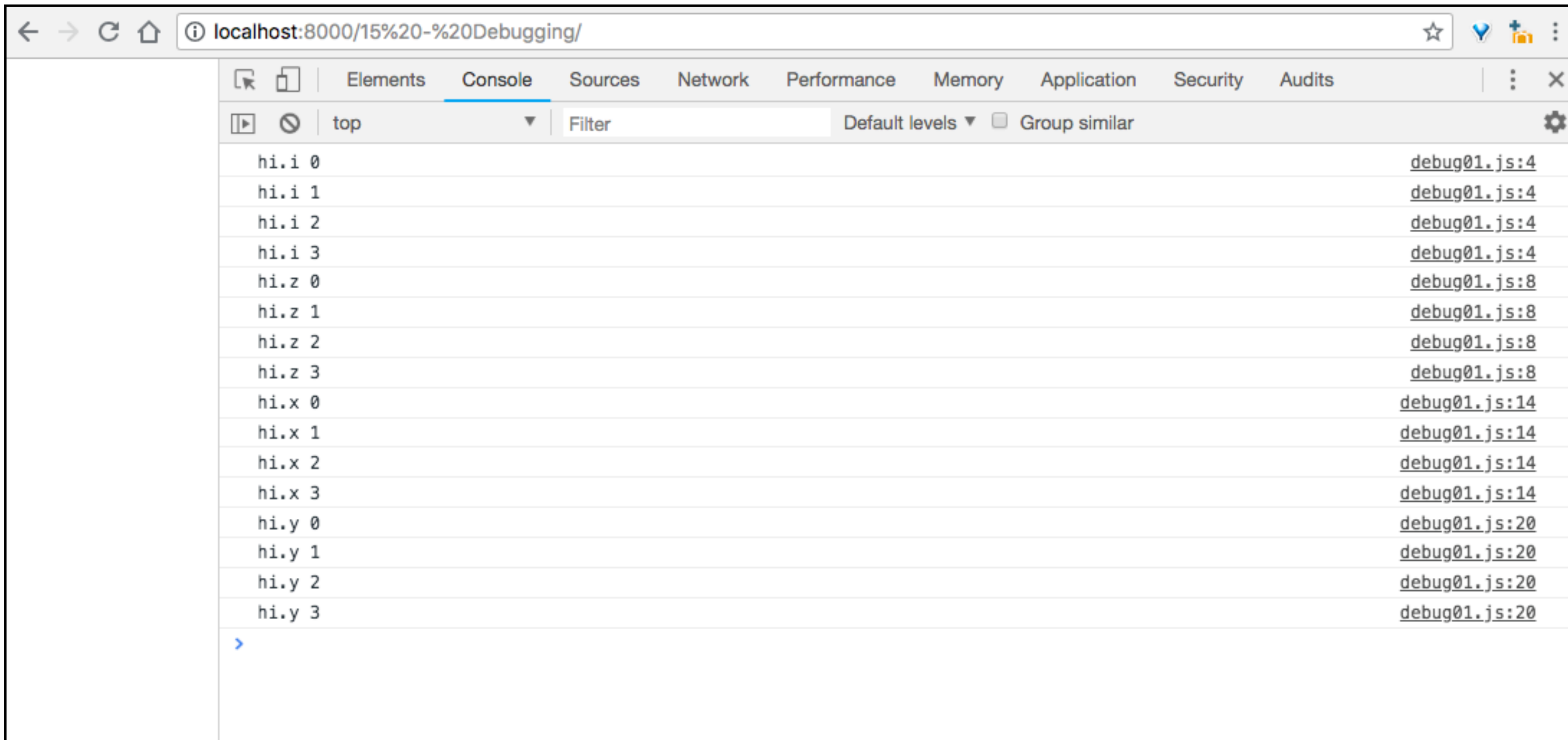
Your car's dashboard is designed to be an early warning system.

 <p>Engine Oil Warning Light</p>	<p>This warning light indicates low engine oil pressure.</p>
	<div>  CAUTION </div> <p><i>Do not run the engine if the oil pressure is low. Otherwise, it could result in extensive engine damage.</i></p> <p>Close Caution ^</p>
	<p>If the light illuminates or the warning indication is displayed while driving:</p> <ol style="list-style-type: none"> 1. Drive to the side of the road and park off the right-of-way on level ground. 2. Turn off the engine and wait 5 minutes for the oil to drain back into the sump. 3. Inspect the engine oil level. (Search) If it's low, add the appropriate amount of engine oil while being careful not to overfill.
	<div>  CAUTION </div> <p><i>Do not run the engine if the oil level is low. Otherwise, it could result in extensive engine damage.</i></p> <p>Close Caution ^</p> <p>4. Start the engine and check the warning light.</p> <p>If the light remains illuminated even though the oil level is normal or after adding oil, stop the engine immediately and have your vehicle towed to an expert repairer, we recommend an Authorised Mazda Repairer.</p>

The car's diagnostic tools (e.g., warning lights) provides [symbolic messages](#) that the operator uses to reference information in the owner's manual. This information can be specific instructions, precautionary measures, or warnings.

**MOST MATURE PROGRAMMING
ENVIRONMENTS HAVE SOME
MANNER OF DIAGNOSTIC
TOOLS TO HELP YOU DEBUG.**





JavaScript is native to the web, so we can use the (really excellent) **tools** that are available in our web browser of choice.

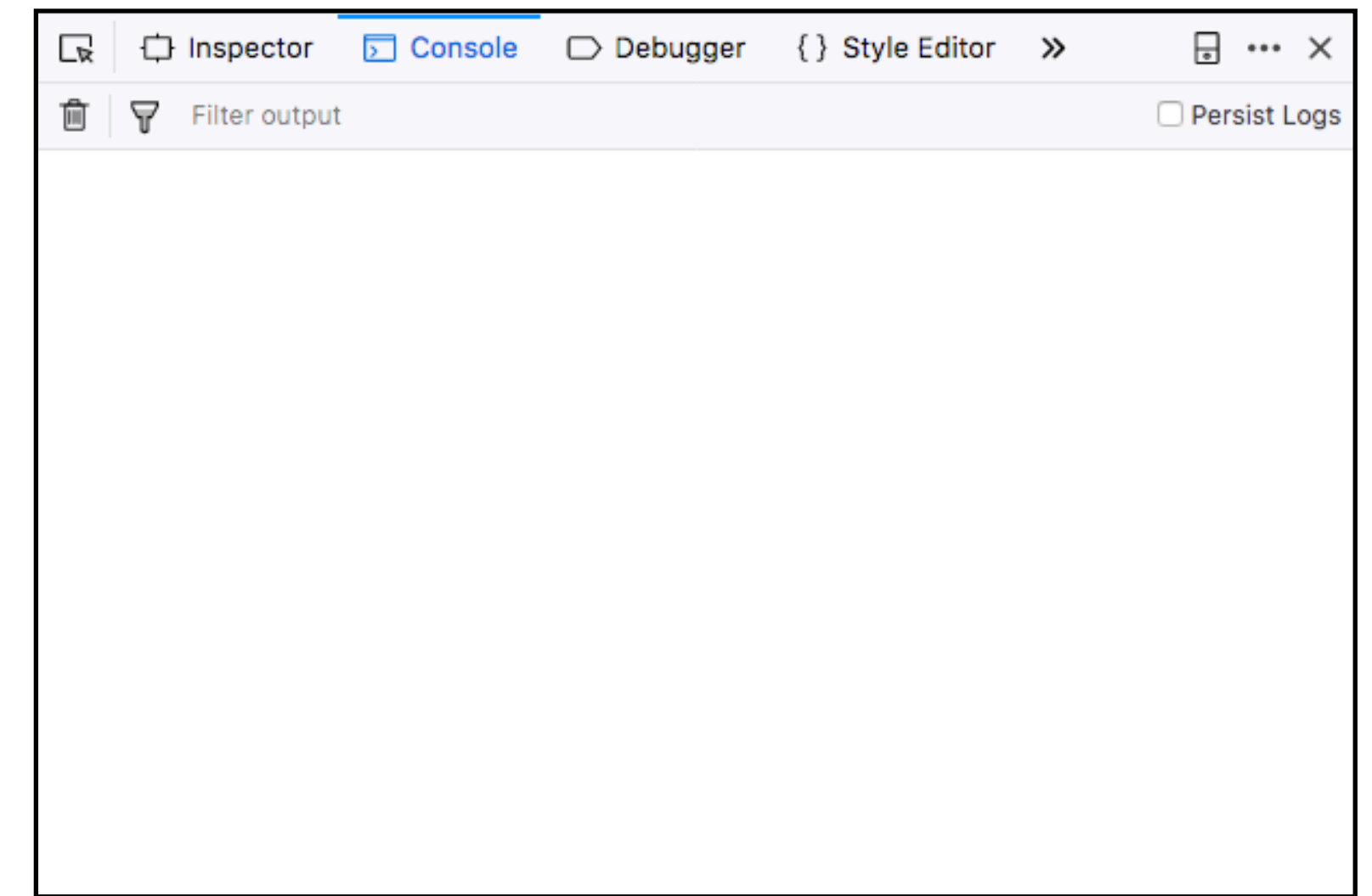
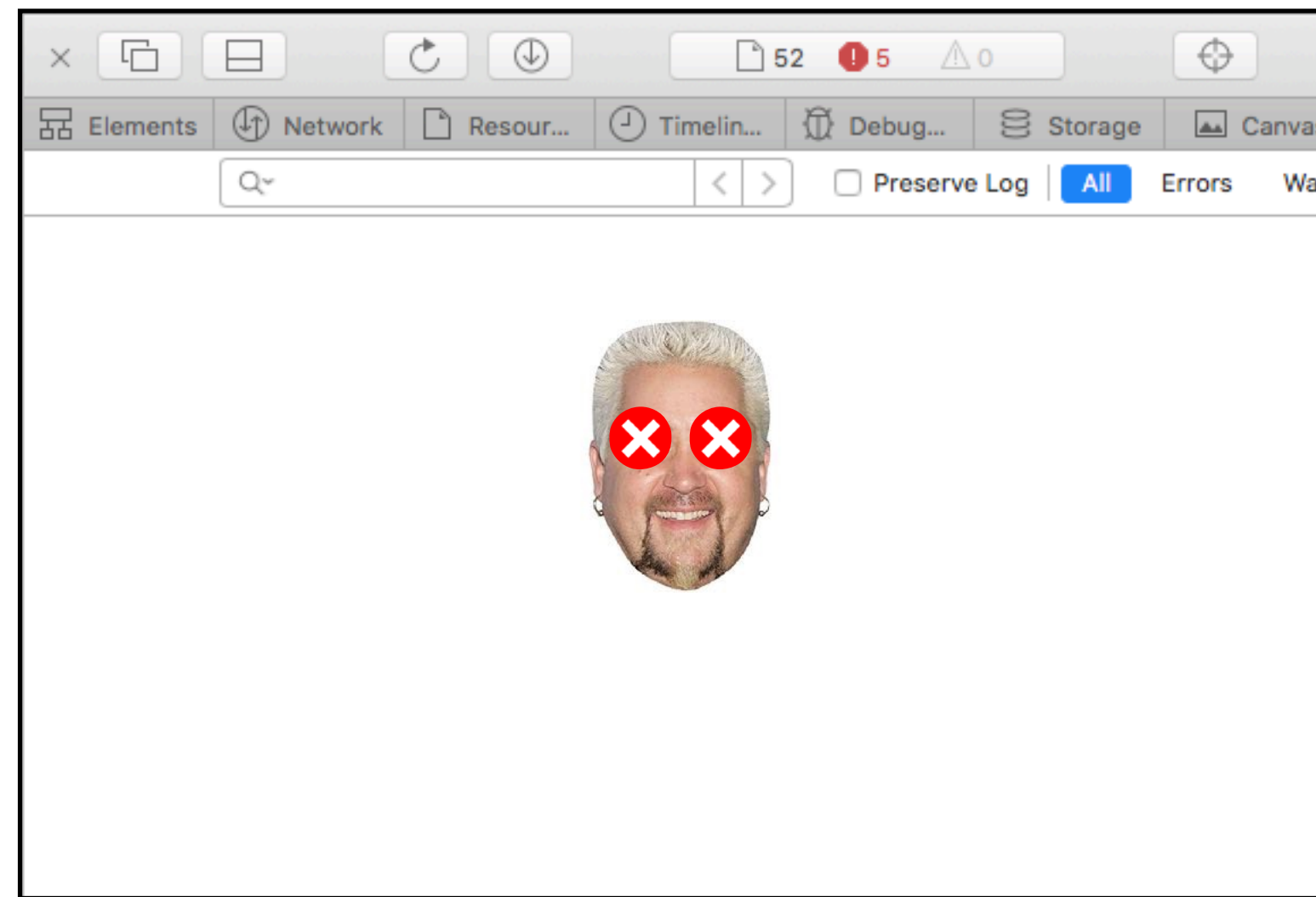
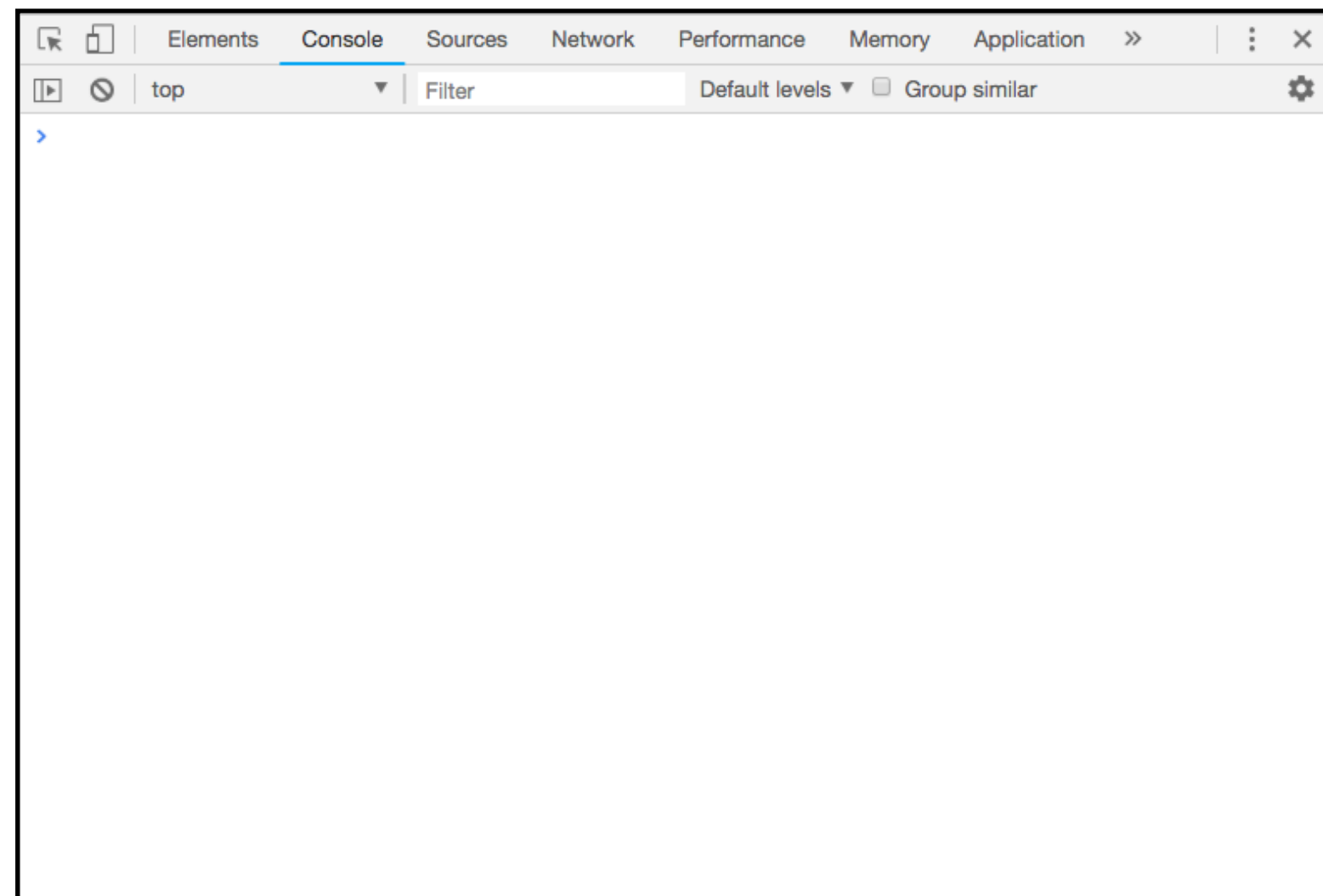
[This is debug01.js, if you need to see it.]

E'RY MAJOR BROWSER GOT A JS **CONSOLE**

CHROM

SAFIERI

FIRFAX

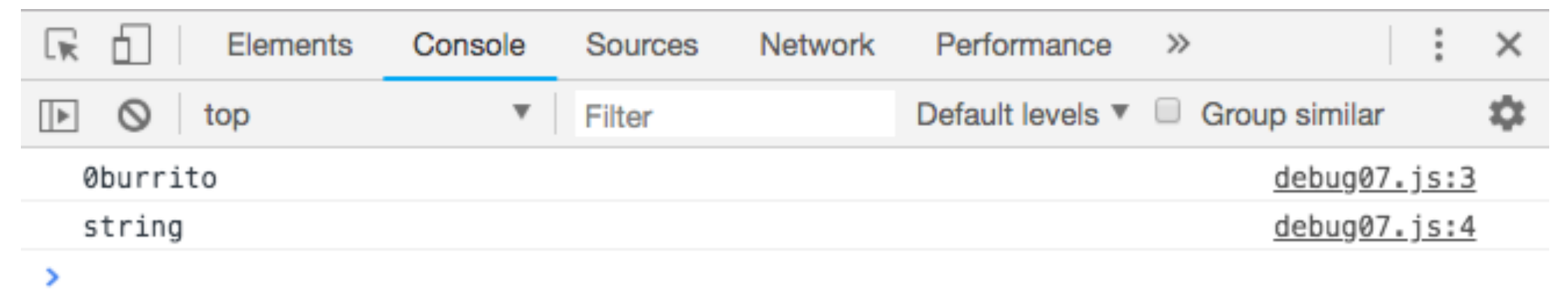


Access them with Command+Option+I (Mac) or Control+Shift+I (Windows, Linux)


```
trashcan = false * [] + "burrito"
```

Bad news: JavaScript has a unique set of “challenges” when it comes to debugging

```
trashcan = false * [] + "burrito"  
console.log(trashcan)  
console.log(typeof(trashcan))
```



JavaScript will happily construct and output a delicious “0burrito.”

```
"use strict";
```

```
trashcan = false * [] + "burrito"  
console.log(trashcan)  
console.log(typeof(trashcan))
```

```
✖ ▼ Uncaught ReferenceError: trashcan is not defined  
  at debug07.js:3  
  (anonymous) @ debug07.js:3
```

“JavaScript can be made a *little* more strict by enabling strict mode. This is done by putting the string “use strict” at the top of a file or a function body...

Putting a “use strict” at the top of your program rarely hurts and might help you spot a problem.”

Eloquent JavaScript (2E), p.140–1

[More info on [strict mode](#) at MDN]

DEBUGGING TIP #2

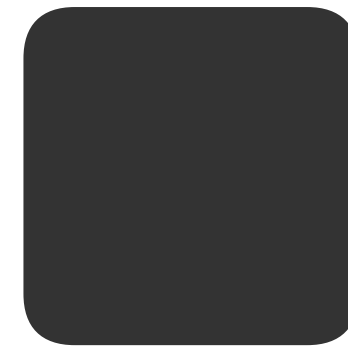
IF YOU HAVE AN ERROR—**STOP**. FIX IT BEFORE MOVING ON.



DEBUGGING TIP #3

YOUR ERROR IS OFTEN THE SIMPLEST POSSIBLE THING.

Hi, I'm a semicolon. You know,
that weird punctuation that
you're not really sure how to use
in everyday writing.



Turns out I'm pretty important in
(strict) JavaScript. I should end
every program statement.

**WITH OUR NEW
SEMICOLON FRIEND
IN MIND, HOW MANY
ERRORS WILL THIS
CODE GENERATE?**

[see debug02.js]

```
1  var numPlayers = 2
2
3  var player = {
4      class: 'Pirate Wizard',
5      health: 90,
6      spells: ['Fireball', 'Lightning Bolt', 'Summon Plank'],
7      ship: 'Magic Galleon',
8      momsName: 'Catherine'
9  }
10
11 console.log(numPlayers)
12 console.log(player)
13
14 if(player.health > 50) {
15     console.log(player.class)
16 }
17
18 ;;;
```

SURPRISE! AUTOMATIC SEMICOLON INSERTION IS A THING.

[see [JavaScript Semicolon Insertion](#) for *detailed* technical info]

JavaScript Semicolon Insertion Everything you need to know

Friday, May 28, 2010

Automatic semicolon insertion is one of JavaScript's most controversial syntactic features. There are also many misconceptions surrounding it.

Some JavaScript programmers use semicolons at the end of every statement, and some use them only where strictly required. Most do something in between, and a few even intentionally add extra semicolons as a matter of style.

Even if you use semicolons at the end of every statement, some constructs parse in non-obvious ways. Regardless of your preferences in semicolon usage, you must know the rules to write JavaScript professionally. If you remember a few simple rules, all of which are explained here, you will be able to understand how any program you might encounter will be parsed, and will be an expert on JavaScript automatic semicolon insertion, or ASI.

REMEMBER TO CHECK FOR

SMALL ERRORS FIRST:

SPELLING, CAPITALIZATION, PATH

NAMES, PUNCTUATION, ETC.

DEBUGGING TIP #4

LEARN YOUR DEBUGGER'S DIALECT.

✖ Uncaught ReferenceError: Invalid left-hand side in assignment debug03.js:22

Let's start with a common example: What does this error message mean?

JS ERROR OBJECTS

The Error Object

JavaScript has a built in error object that provides error information when an error occurs.

The error object provides two useful properties: name and message.

Error Object Properties

Property	Description
name	Sets or returns an error name
message	Sets or returns an error message (a string)

Error Name Values

Six different values can be returned by the error name property:

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURIComponent() has occurred

Source: [w3schools](https://www.w3schools.com/js/js_error_objects.asp)

Error Object name

✖ Uncaught ReferenceError

Error Object name

✖ Uncaught ReferenceError

This error was not caught in a
`catch` statement.



WHAT IS A CATCH STATEMENT?

WHEN THE BROWSER
THROWS AN ERROR, YOU
WANT TO **TRY** AND **CATCH** IT.

```
// test input field value for errors, and throw custom
messages
try {
    if(x == "")    throw "is empty";
    if(isNaN(x))   throw "is not a number";
    x = Number(x);
    if(x > 10)     throw "is too high";
    if(x < 5)      throw "is too low";
}
// execute if an error is caught
catch(err) {
    message.innerHTML = "Input " + x + " " + err;
}
// clear input field value after try/catch execute
finally {
    document.getElementById("demo").value = "NOM NOM";
}
```

[Let's load up debug03.html]

🔗 Syntax

```
try {  
    try_statements  
}  
[catch (exception_var_1 if condition_1) { // non-standard  
    catch_statements_1  
}]  
...  
[catch (exception_var_2) {  
    catch_statements_2  
}]  
[finally {  
    finally_statements  
}]
```

try_statements

The statements to be executed.

catch_statements_1, catch_statements_2

Statements that are executed if an exception is thrown in the `try` block.

exception_var_1, exception_var_2

An identifier to hold an exception object for the associated `catch` clause.

condition_1

A conditional expression.

finally_statements

Statements that are executed after the `try` statement completes. These statements execute regardless of whether an exception was thrown or caught.

As usual, Mozilla Developer Network has an excellent breakdown of the [try...catch syntax](#) with descriptions and examples.

Error Object name

✖ Uncaught ReferenceError

This error was not caught in a `catch` statement.

Your code has `referenced` (i.e., tried to use) a variable that doesn't exist. You either need to declare the variable, or make sure it's in the proper scope.

Error Object message

Invalid left-hand side in assignment

Error Object message

Invalid left-hand side in assignment

You did a bad.

Where you did a bad.

What **kind** of bad you did.

Console error bonus info

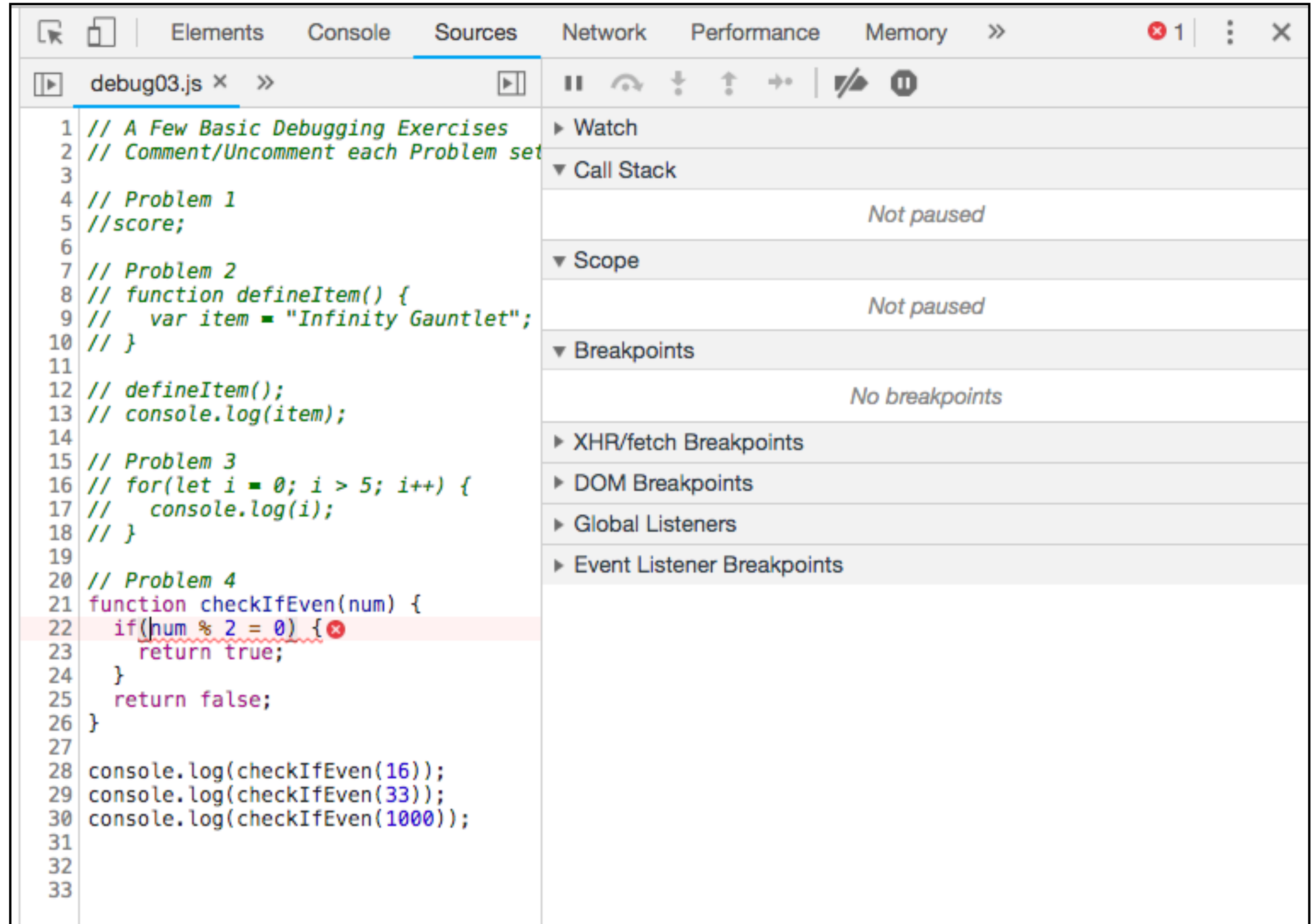
debug03.js:22

File

Line

Note the underline—clicking this will magically transport you to your error.

BEHOLD THE
ERROR THOU
HAST SOUGHT.



✖ Uncaught ReferenceError: Invalid left-hand side in assignment debug03.js:22

Summary: there's a lot of info packed into this cryptic string of words and numbers.

DEBUGGING TIP #5

LEAVE BREADCRUMBS TO FOLLOW.


```
console.log('error message');
```

**WE USE THIS CONSOLE
LOG METHOD A LOT, BUT
HOW DO WE USE IT?**


```
1 // Debugging example from Eloquent JS (2E) p. 143
2 // As written, function contains a bug
3 // Scroll below to see solution
4
5 function numberToString(n, base) {
6     var result = "", sign = "";
7     if (n < 0) {
8         sign = "-";
9         n = -n;
10    }
11    do {
12        result = String(n % base) + result;
13        n /= base;
14    } while (n > 0);
15
16    return sign + result;
17 }
18
19 console.log(numberToString(13, 10));
```

This example program tries to **convert a whole number to a string** in any base (e.g., decimal, binary, etc.) by repeatedly putting the number's last digit in a string then dividing the number by the base to remove its last digit.

Let's look at this step-by-step then check the output of the program (as written).



```
5 function numberToString(n, base) {
6     var result = "", sign = "";
7     if (n < 0) {
8         sign = "-";
9         n = -n;
10    }
11    do {
12        result = String(n % base) + result;
13        n /= base;
14    } while (n > 0);
15
16    return sign + result;
17 }
18
19 console.log(numberToString(13, 10));
```

Most of the program is a single function that accepts two parameters: the whole number we want to convert to a string and the desired base of that number.

For example, `numberToString(246, 2)` should return the binary value of 246 as a string, i.e., “11110110”

So our sample input (13, 10) should simply output: “13”



```
5 function numberToString(n, base) {  
6     var result = "", sign = "";  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

First, we're initializing two empty string variables to store and return the function's final output.



```
5 function numberToString(n, base) {  
6     var result = "", sign = "";  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

The if statement checks to see if our whole number is negative. If so, it stores the string “-” in the sign variable, then converts our number to a positive by flipping the sign.


```
5 function numberToString(n, base) {  
6     var result = "", sign = "";  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

The do-while loop is where most of the function's work is done. It will perform the two inner statements until our original number (stored in `n`) is 0 or less.

```
5 function numberToString(n, base) {
6     var result = "", sign = "";
7     if (n < 0) {
8         sign = "-";
9         n = -n;
10    }
11    do {
12        result = String(n % base) + result;
13        n /= base;
14    } while (n > 0);
15
16    return sign + result;
17 }
18
19 console.log(numberToString(13, 10));
```

This statement takes the remainder of n divided by base, converts it to type String, then concatenates it to the contents of results (which, the first time through the loop, will be an empty string "").

What should the remainder of 13 / 10 be?

```
5 function numberToString(n, base) {  
6     var result = "", sign = "";  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

We then assign n to the result of n divided by the base parameter.

After the first do, will the while loop loop again? And what's the current value of n?

```
5 function numberToString(n, base) {  
6     var result = "", sign = "";  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

Once our loop is finished, we want to return the requested string, which is a concatenation of sign and result.


```
5 function numberToString(n, base) {  
6     var result = "", sign = "";  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

What's the final output to the console?

ElementsConsoleSourcesNetworkPerformanceMemoryApplicationSecurity>>⋮✕

▶⏏top▼FilterDefault levels▼Group similar⚙

1.5e-3231.3e-3221.3e-3211.3e-3201.3e-3191.3e-3181.3e-3171.3e-3161.3e-3151.3e-3141.3e-3131.3e-3121.3e-3111.3e-3101.3e-3091.2999999999999999e-3081.2999999999999999e-3071.2999999999999999e-3061.2999999999999999e-3051.2999999999999999e-3041.2999999999999999e-3031.2999999999999999e-3021.2999999999999999e-3011.2999999999999999e-3001.2999999999999999e-2991.2999999999999999e-2981.2999999999999999e-2971.2999999999999999e-2961.2999999999999999e-2951.2999999999999999e-2941.2999999999999999e-2931.2999999999999999e-2921.2999999999999999e-2911.2999999999999999e-2901.2999999999999999e-2891.2999999999999999e-2881.2999999999999999e-2871.2999999999999999e-2861.2999999999999999e-2851.2999999999999999e-2841.2999999999999999e-2831.2999999999999999e-2821.2999999999999999e-2811.2999999999999999e-2801.2999999999999999e-2791.2999999999999999e-2781.2999999999999999e-2771.2999999999999999e-2761.2999999999999999e-2751.2999999999999999e-2741.3e-2731.3e-2721.3e-2711.3e-2701.3e-2691.3e-2681.3e-2671.3e-2661.3e-2651.3e-2641.3000000000000002e-2631.3000000000000002e-2621.3000000000000003e-2611.3000000000000003e-2601.3000000000000003e-2591.3000000000000003e-2581.3000000000000004e-2571.3000000000000003e-2561.3000000000000003e-2551.3000000000000003e-2541.3000000000000004e-2531.3000000000000004e-2521.3000000000000003e-2511.3000000000000003e-2501.3000000000000004e-2491.3000000000000004e-2481.3000000000000005e-2471.3000000000000004e-2461.3000000000000004e-2451.3000000000000004e-2441.3000000000000004e-2431.3000000000000005e-2421.3000000000000005e-2411.3000000000000005e-2401.3000000000000004e-2391.3000000000000004e-2381.3000000000000004e-2371.3000000000000005e-2361.3000000000000005e-2351.3000000000000005e-2341.3000000000000006e-2331.3000000000000005e-2321.3000000000000006e-2311.3000000000000005e-2301.3000000000000006e-2291.3000000000000006e-2281.3000000000000005e-2271.3000000000000005e-2261.3000000000000005e-2251.3000000000000006e-2241.3000000000000006e-2231.3000000000000005e-2221.3000000000000006e-2211.3000000000000006e-2201.3000000000000006e-2191.3000000000000007e-2181.3000000000000008e-2171.3000000000000007e-2161.3000000000000008e-2151.3000000000000008e-2141.3000000000000007e-2131.3000000000000008e-2121.3000000000000009e-2111.3000000000000008e-2101.3000000000000007e-2091.3000000000000007e-2081.3000000000000008e-2071.3000000000000009e-2061.3000000000000008e-2051.3000000000000008e-2041.3000000000000007e-2031.3000000000000007e-2021.3000000000000006e-2011.3000000000000007e-2001.3000000000000007e-1991.3000000000000008e-1981.3000000000000008e-1971.3000000000000008e-1961.3000000000000008e-1951.3000000000000008e-1941.3000000000000008e-1931.3000000000000007e-1921.3000000000000007e-1911.3000000000000006e-1901.3000000000000007e-1891.3000000000000007e-1881.3000000000000007e-1871.3000000000000008e-1861.3000000000000007e-1851.3000000000000007e-1841.3000000000000006e-1831.3000000000000005e-1821.3000000000000005e-1811.3000000000000006e-1801.3000000000000005e-1791.3000000000000005e-1781.3000000000000005e-1771.3000000000000005e-1761.3000000000000006e-1751.3000000000000006e-1741.3000000000000006e-1731.3000000000000005e-1721.3000000000000005e-1711.3000000000000004e-1701.3000000000000005e-1691.3000000000000005e-1681.3000000000000005e-1671.3000000000000005e-1661.3000000000000005e-1651.3000000000000005e-1641.3000000000000005e-1631.3000000000000005e-1621.3000000000000006e-1611.3000000000000006e-1601.3000000000000007e-1591.3000000000000006e-1581.3000000000000006e-1571.3000000000000006e-1561.3000000000000007e-1551.3000000000000007e-1541.3000000000000006e-1531.3000000000000007e-1521.3000000000000007e-1511.3000000000000007e-1501.3000000000000006e-1491.3000000000000006e-1481.3000000000000005e-1471.3000000000000004e-1461.3000000000000003e-1451.3000000000000003e-1441.3000000000000004e-1431.3000000000000003e-1421.3000000000000003e-1411.3000000000000004e-1401.3000000000000005e-1391.3000000000000004e-1381.3000000000000005e-1371.3000000000000006e-1361.3000000000000006e-1351.3000000000000006e-1341.3000000000000006e-1331.3000000000000005e-1321.3000000000000005e-1311.3000000000000004e-1301.3000000000000004e-1291.3000000000000004e-1281.3000000000000004e-1271.3000000000000003e-1261.3000000000000003e-1251.3000000000000004e-1241.3000000000000005e-1231.3000000000000004e-1221.3000000000000004e-1211.3000000000000004e-1201.3000000000000003e-1191.3000000000000002e-1181.3000000000000003e-1171.3000000000000003e-1161.3000000000000002e-1151.3000000000000002e-1141.3000000000000002e-1131.3000000000000002e-1121.3000000000000001e-1111.3000000000000001e-1101.3000000000000002e-1091.3000000000000001e-1081.3e-1071.3e-1061.3e-1051.3e-1041.3e-1031.3e-1021.2999999999999999e-1011.3e-1001.3e-991.3e-971.3e-961.3e-951.3e-941.3e-931.3e-921.3e-911.3e-901.3000000000000001e-891.3e-881.3e-871.3e-861.3e-851.3e-841.3e-831.3e-821.3e-811.3e-801.3000000000000001e-791.3e-781.3e-771.3e-761.3e-751.3e-741.3e-731.3e-721.3e-711.2999999999999998e-701.2999999999999998e-691.3e-681.3e-671.3e-661.3e-651.3e-641.3e-631.3e-621.2999999999999999e-611.3e-601.2999999999999998e-591.2999999999999999e-581.3e-571.3e-561.3e-551.3e-541.3e-531.3e-521.3e-511.3e-501.3000000000000002e-491.3000000000000001e-481.3e-471.3e-461.3000000000000001e-451.3000000000000002e-441.3000000000000002e-431.3000000000000002e-421.3000000000000002e-411.3000000000000003e-401.3000000000000003e-391.3000000000000003e-381.3000000000000003e-371.3000000000000004e-361.3000000000000003e-351.3000000000000002e-341.3000000000000001e-331.3000000000000001e-321.3000000000000002e-311.3000000000000001e-301.3e-291.3e-281.3e-271.3e-261.3e-251.3e-241.3e-231.3e-221.2999999999999999e-211.2999999999999998e-201.2999999999999998e-191.2999999999999998e-181.2999999999999999e-171.3e-161.3e-151.3e-141.3e-131.3000000000000001e-121.3000000000000002e-111.3000000000000002e-101.3e-91.3e-81.3e-70.00000130.0000130000000000000010.00013000000000000020.0013000000000000020.01300000000000010.131.33

Well then.

Despite showing no errors, we clearly have an unintended bug in our function.


```
5 function numberToString(n, base) {  
6     var result = "", sign = "";  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

This is the point where you stop and think about where you can strategically add [breadcrumbs](#) (i.e., console.log statements) to help you track down your bug.

Where would you add them?

Let's look at debug04.js

```

59  /*****
60  FIXED PROGRAM
61  *****/
62  function numberToString(n, base) {
63      var result = "", sign = "";
64      if (n < 0) {
65          sign = "-";
66          n = -n;
67      }
68      console.log('n before while: ' + n);
69      do {
70          result = String(n % base) + result;
71          n = Math.floor(n / base);
72          console.log('n in while: ' + n);
73      } while (n > 0);
74
75      return sign + result;
76  }
77
78  console.log(numberToString(13, 10));
79  console.log(numberToString(246, 2));

```

The console revealed that our problem was the `n /= base;` statement, because it was outputting floats instead of whole numbers.

`Math.floor()` was the solution. 👍

(`Math.floor` returns the largest integer less than or equal to a given number. Via [MDN](#).)

Also remember that we've been using this [breadcrumb](#) technique throughout the quarter. I like to drop `console.log()` messages in most state methods (but not update), so I always know where I am in my game.

```
// Simple 3-State Game Architecture
```

```
// define game
```

```
var game = new Phaser.Game(800, 600, Phaser.AUTO, 'phaser');
```

```
// define MainMenu state and methods
```

```
var MainMenu = function(game) {};
```

```
MainMenu.prototype = {
```

```
  preload: function() {
```

```
    console.log('MainMenu: preload');
```

```
  },
```

```
  create: function() {
```

```
    console.log('MainMenu: create');
```

```
  },
```

```
  update: function() {
```

```
    // main menu logic
```

```
  }
```

```
}
```

```
// define Gameplay state and methods
```

```
var Gameplay = function(game) {};
```

```
Gameplay.prototype = {
```

```
  preload: function() {
```

```
    console.log('Gameplay: preload');
```

```
  },
```

```
  create: function() {
```

```
    console.log('Gameplay: create');
```

```
  },
```

```
  update: function() {
```

```
    // Gameplay logic
```

```
  }
```

```
}
```

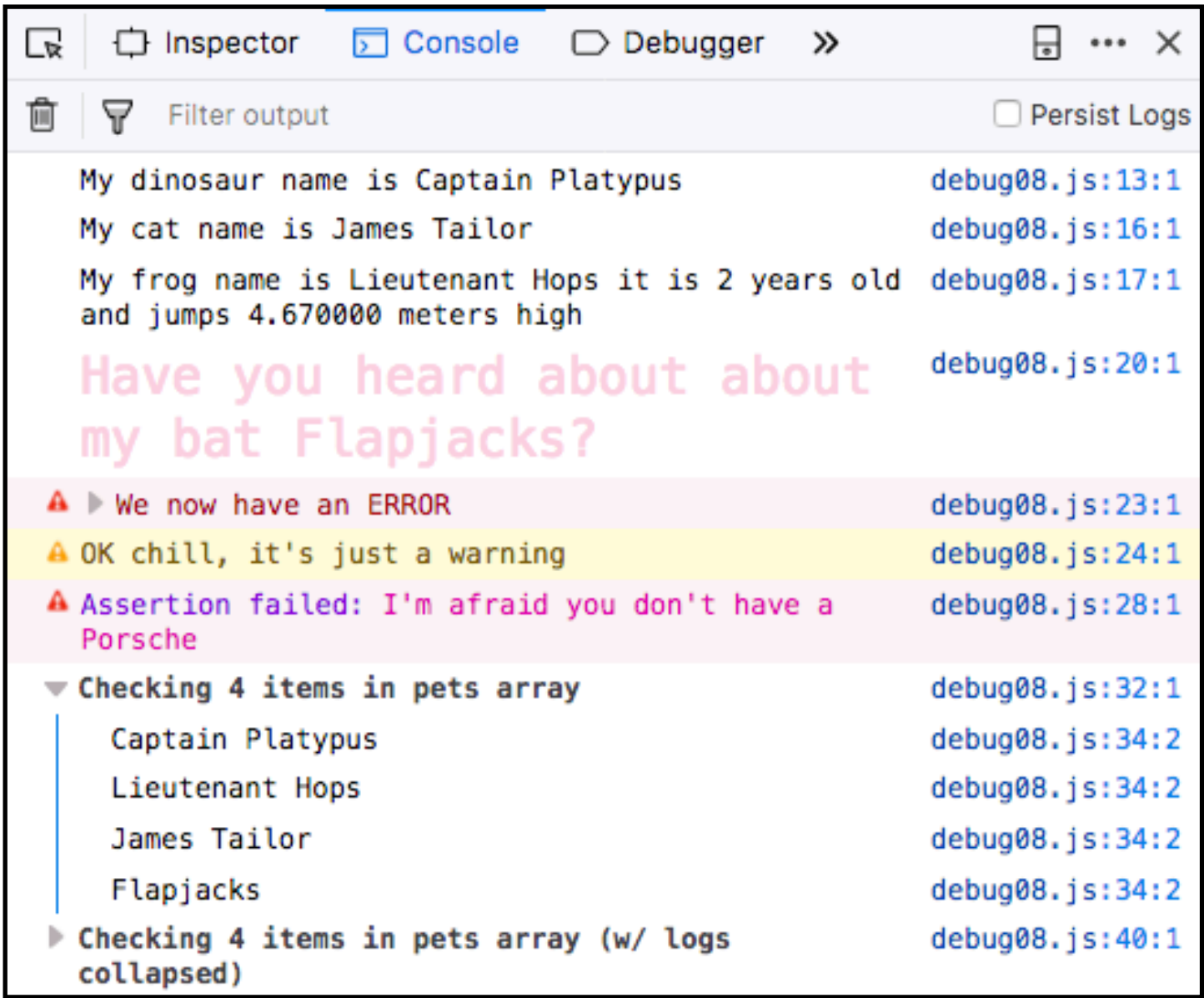

GUESS WHAT? THERE'S MORE TO CONSOLE THAN JUST LOGS()

(Reference: [“Diagnose and Log to Console”](#))

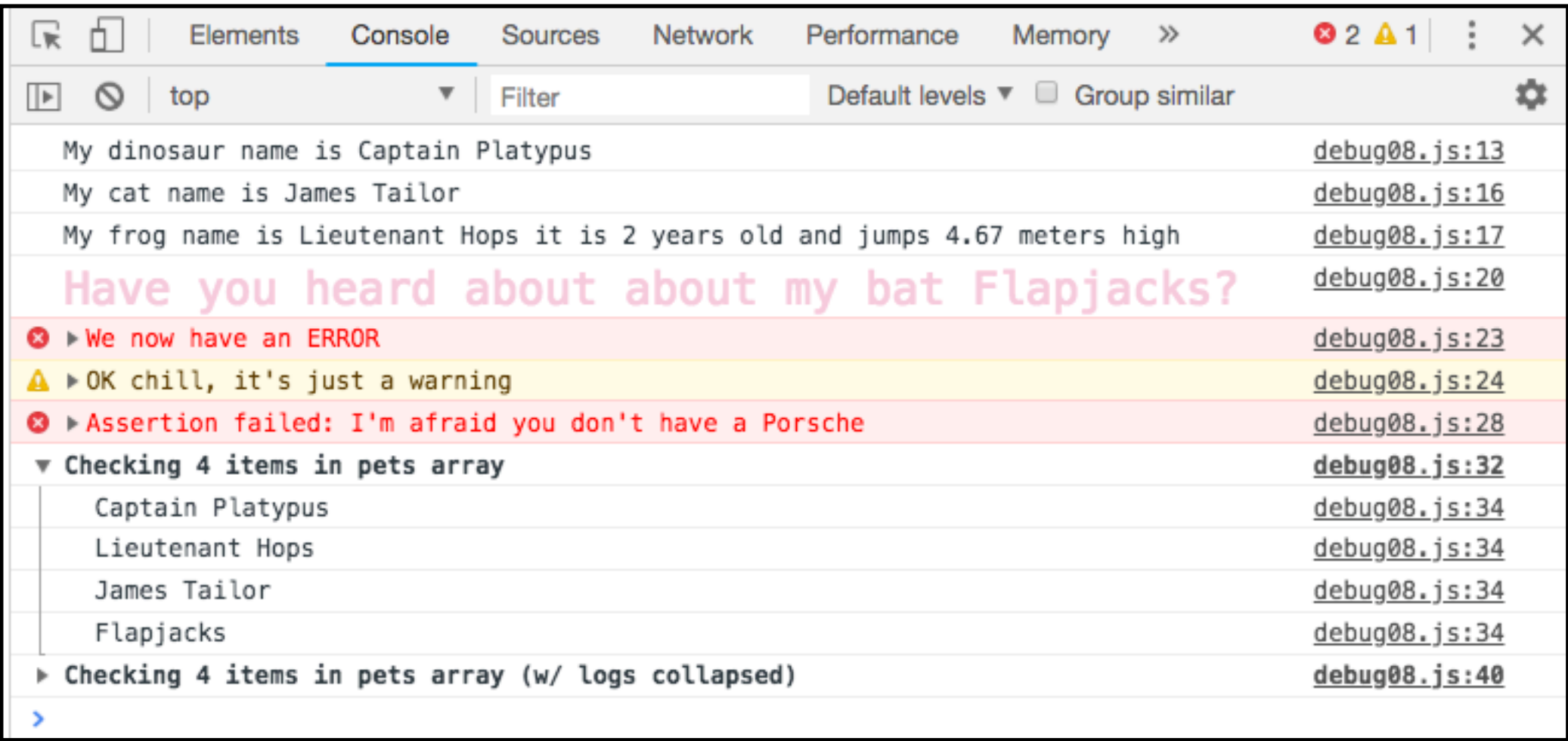
```
1 // different ways to output to console
2 "use strict";
3
4 var dinosaurName = "Captain Platypus";
5 var frogName = "Lieutenant Hops";
6 var frogJump = 4.67;
7 var frogAge = 2;
8 var catName = "James Taylor";
9 var batName = "Flapjacks";
10 var hasMazda = true, hasPorsche = false;
11
12 // concatenation
13 console.log('My dinosaur name is ' + dinosaurName);
14
15 // string substitution
16 console.log('My cat name is', catName);
17 console.log('My frog name is %s it is %i years old and jumps %f
    meters high', frogName, frogAge, frogJump);
18
19 // output with CSS formatting (%c)
20 console.log('%cHave you heard about my bat %s?', '
    color:#FACADE; font-size:20px', batName);
21
22 // error() and warn() methods to draw attention to the errors
23 console.error('We now have an ERROR');
24 console.warn('OK chill, it\'s just a warning');
```

[Let's step inside the mind of debug05.js]

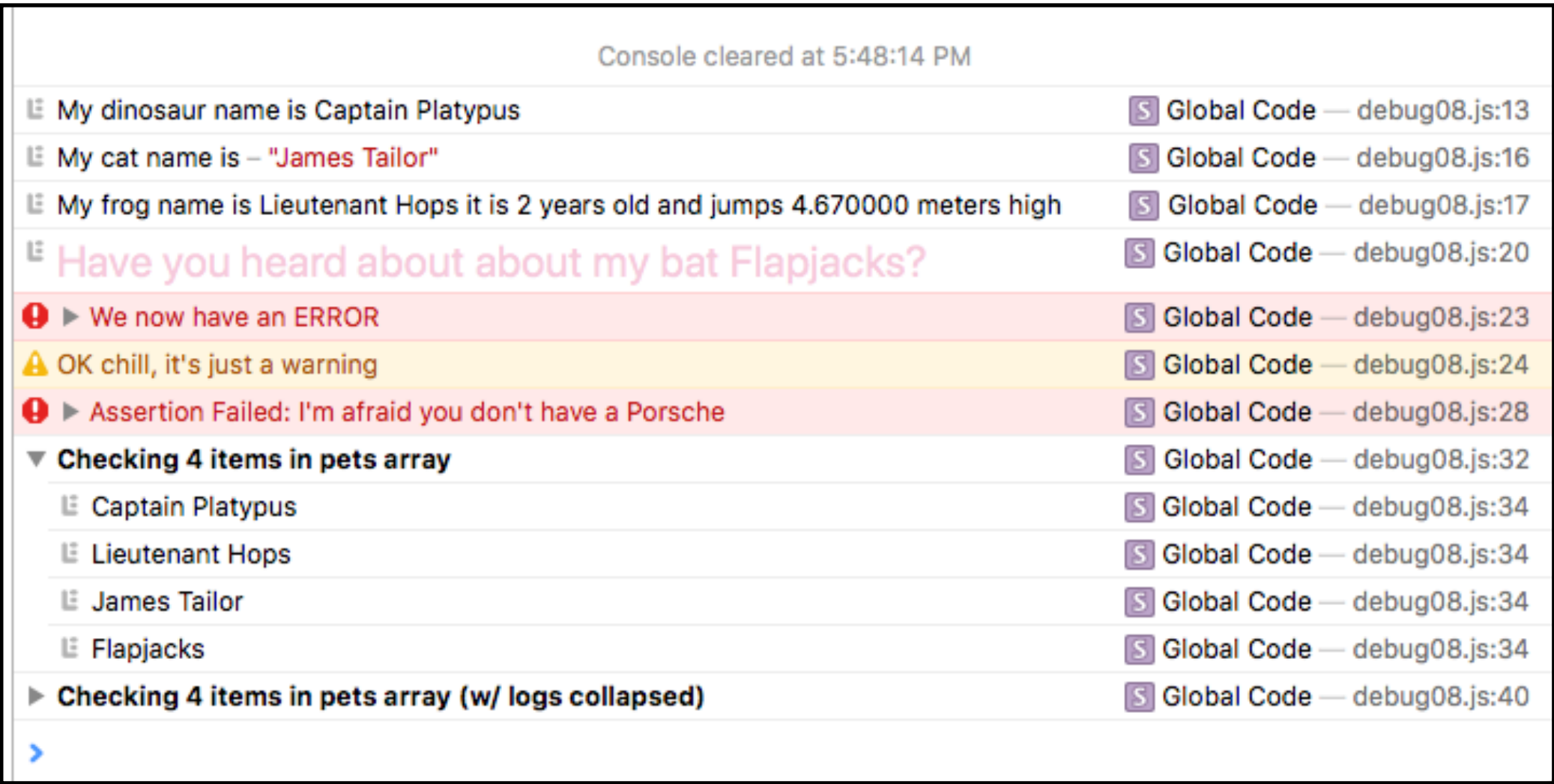
Also note that there are **slight variances** in console styles across browsers. Safari, for instance, uses a sans serif font versus monospace variants in Firefox and Chrome.



Firefox



Chrome



Safari

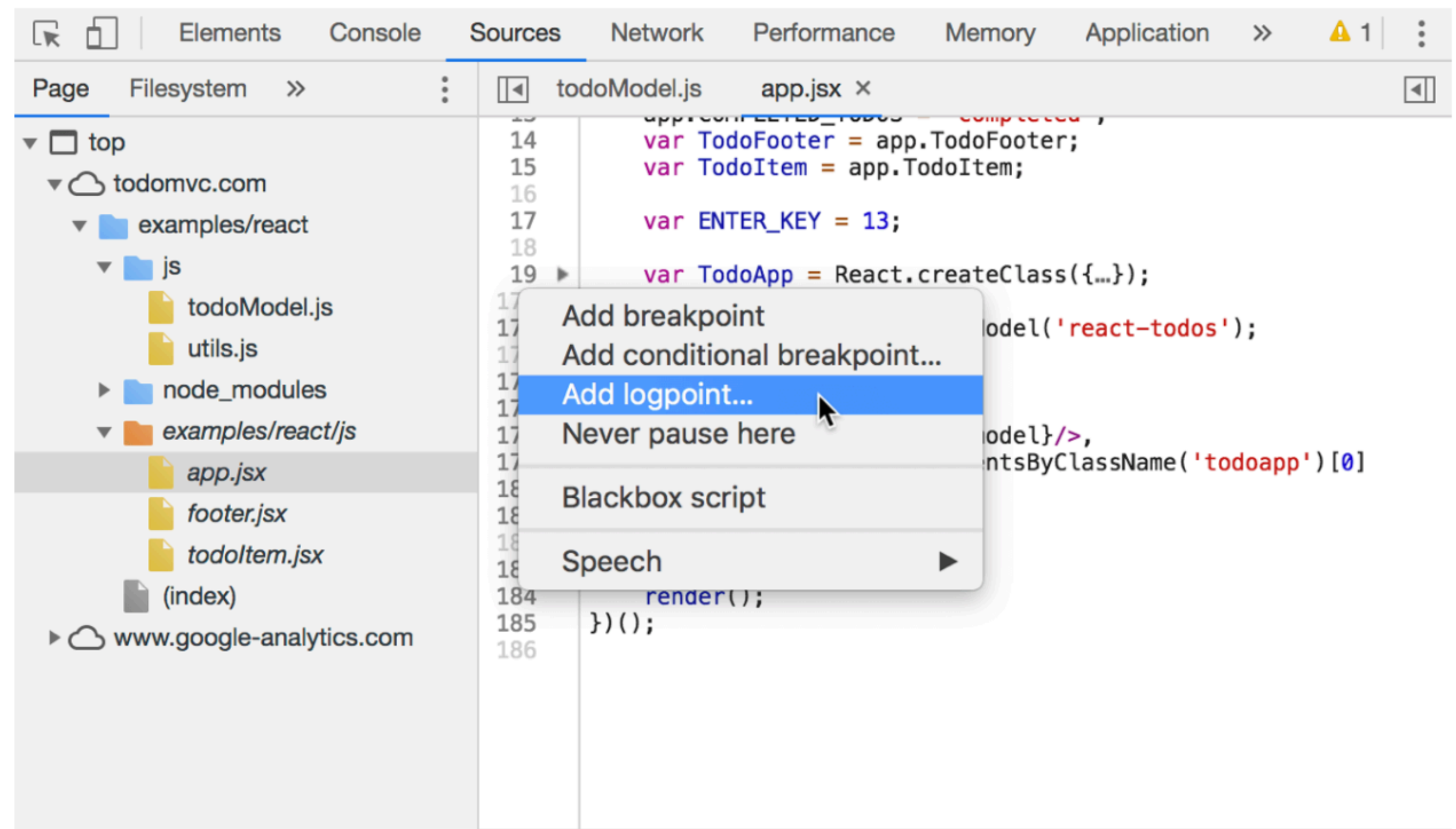
Logpoints are a brand new feature in Chrome DevTools!

Logpoints

Use Logpoints to log messages to the Console without cluttering up your code with `console.log()` calls.

To add a logpoint:

1. Right-click the line number where you want to add the Logpoint.



DEBUGGING TIP #6

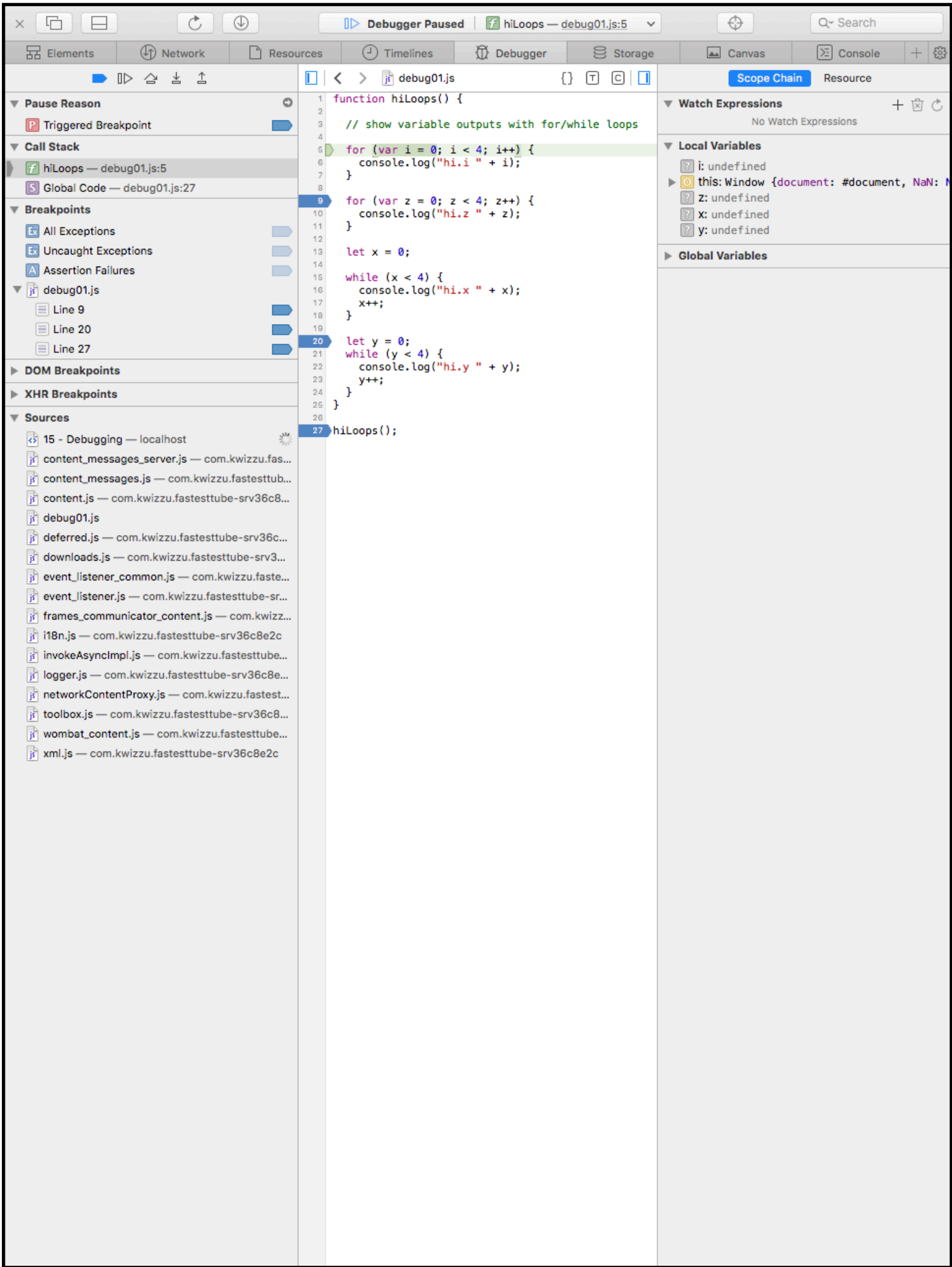
UNDERSTAND BREAKPOINTS AND STEPPING.



POINT BREAK

Twenty-seven banks in three years. Anything to catch the perfect wave.

VS.

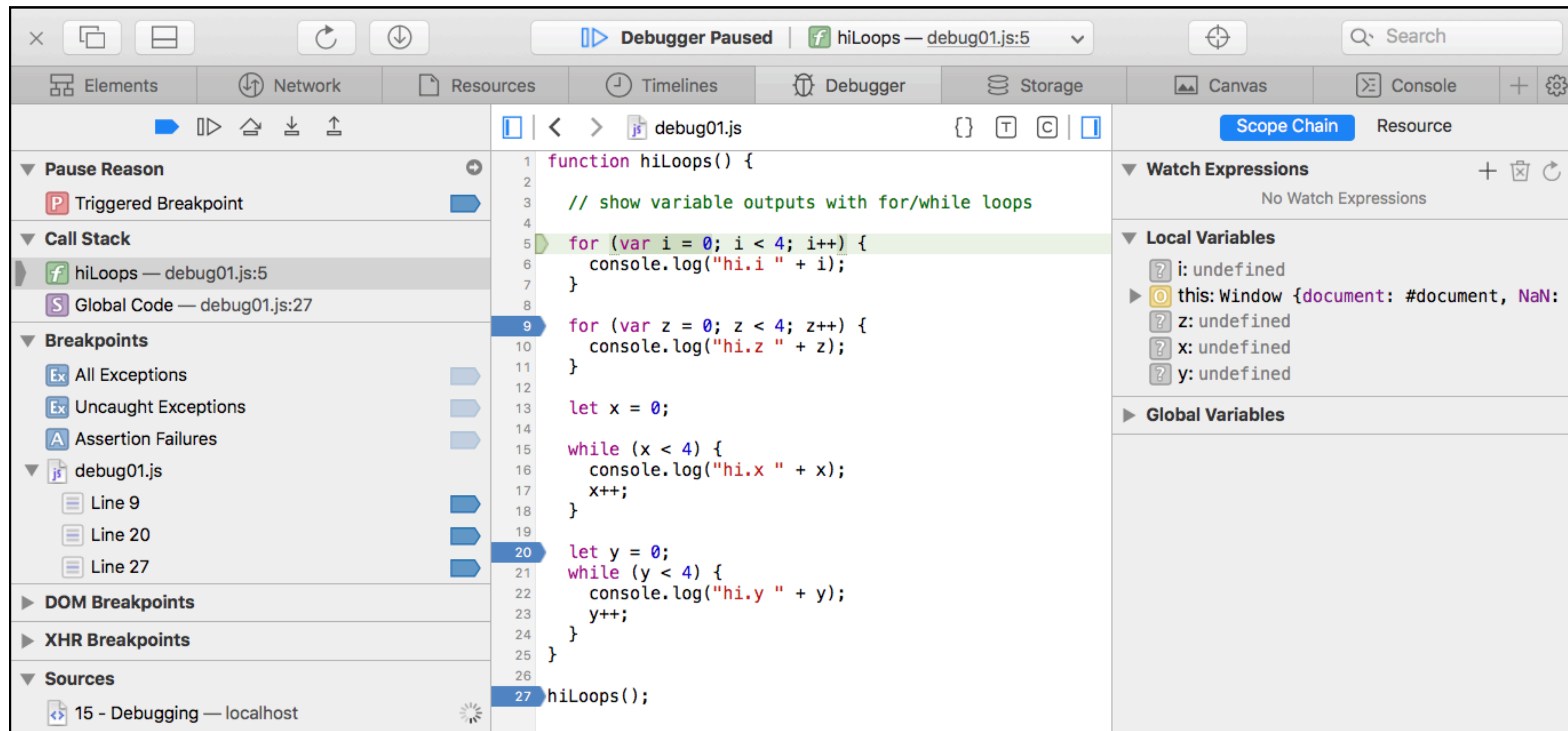


BREAKPOINT

Tells the debugger to stop executing, so you can examine variables, values, etc.

In this example, we have [three breakpoints](#) set: line 9, line 20, and line 27. When the browser runs this JavaScript file, it will stop execution whenever it reaches those lines. You set breakpoints by clicking the line number in the column to the left of code. A (Safari) breakpoint will appear as a [blue flag](#).

Q: On which line will execution stop first?

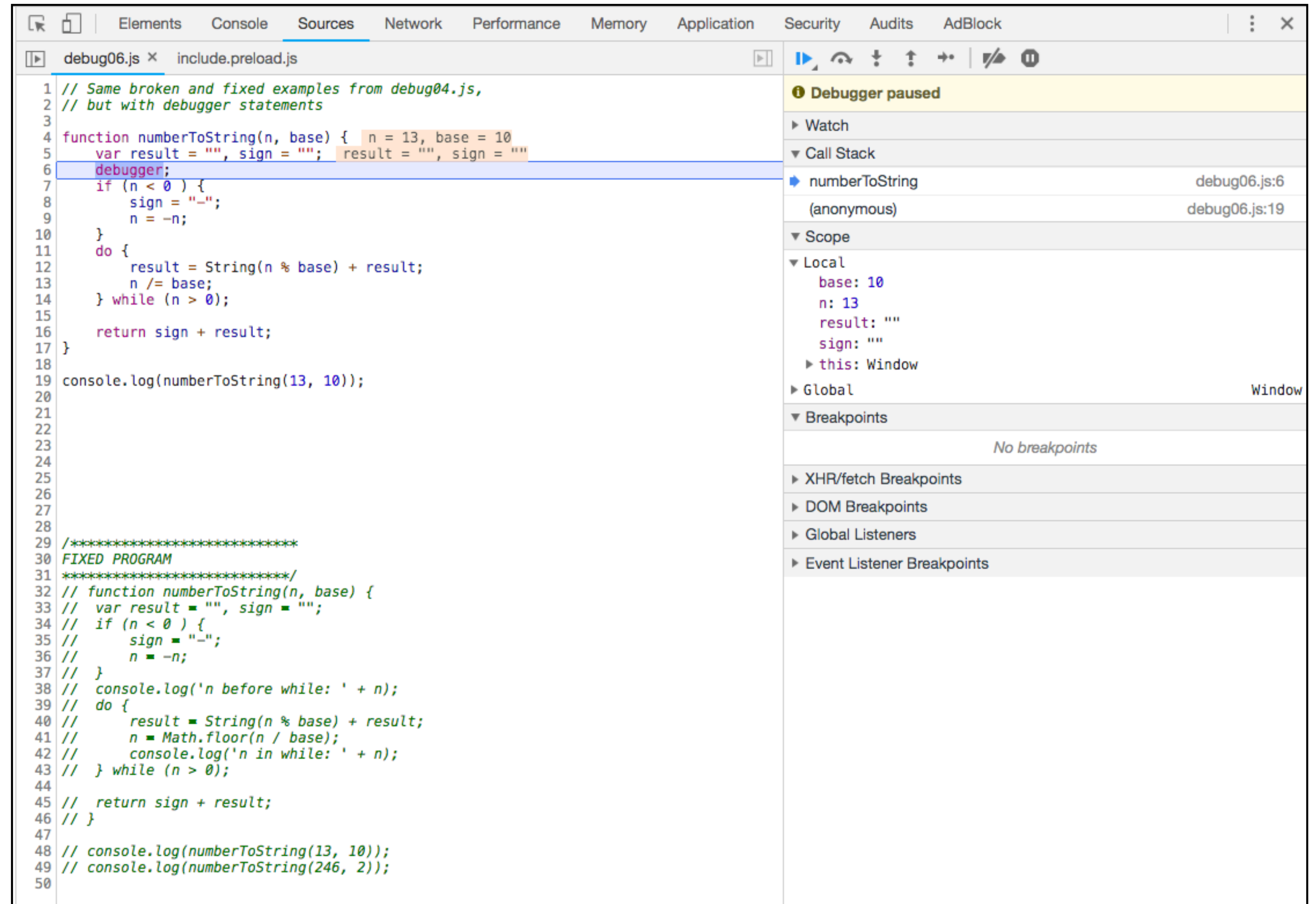


```
1 // Same broken and fixed examples from debug04.js,  
2 // but with debugger statements  
3  
4 function numberToString(n, base) {  
5     var result = "", sign = "";  
6     debugger;  
7     if (n < 0) {  
8         sign = "-";  
9         n = -n;  
10    }  
11    do {  
12        result = String(n % base) + result;  
13        n /= base;  
14    } while (n > 0);  
15  
16    return sign + result;  
17 }  
18  
19 console.log(numberToString(13, 10));
```

In addition to setting manual breakpoints in the browser's console debugger, we can also insert the `debugger;` statement in-line with our code. This halts execution and allows you to [step in](#) to the program at the designated line.

The Debugger is complex, but it gives us access to some powerful tools, like the ability to watch our code execute step-by-step.

Let's look at a few important commands first.

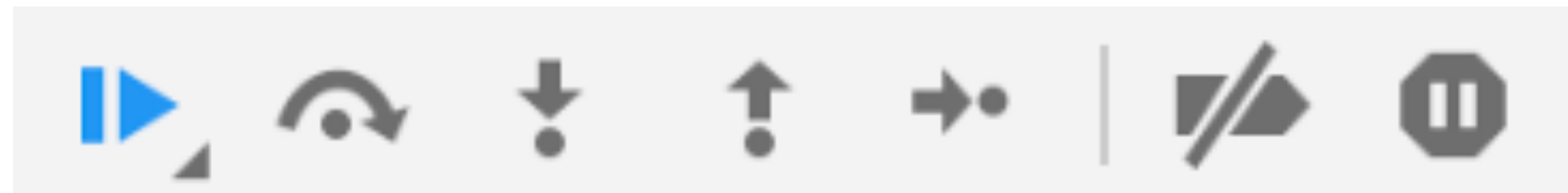
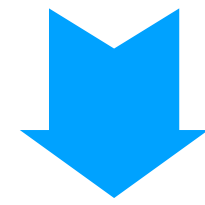


This is the toolbar in the JavaScript Debugging pane.*

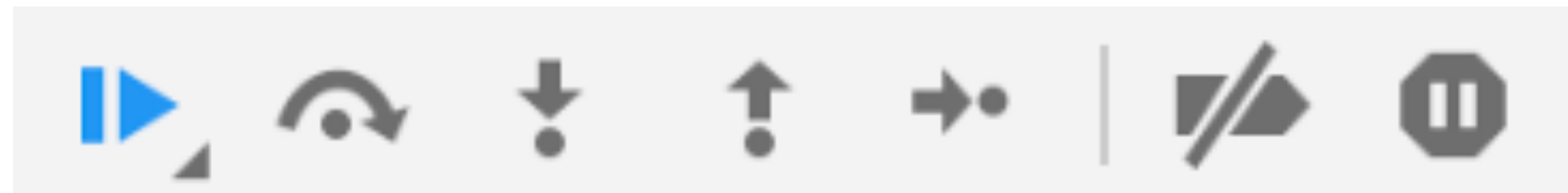
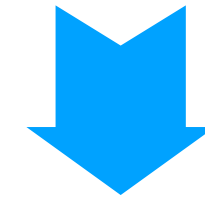


* These examples are from Chrome. The Debugger in other browsers functions similarly, though the icons and layout are slightly different.

When your code hits a breakpoint, it stops execution.
This icon allows your to [Resume Execution](#) (or vice-versa).



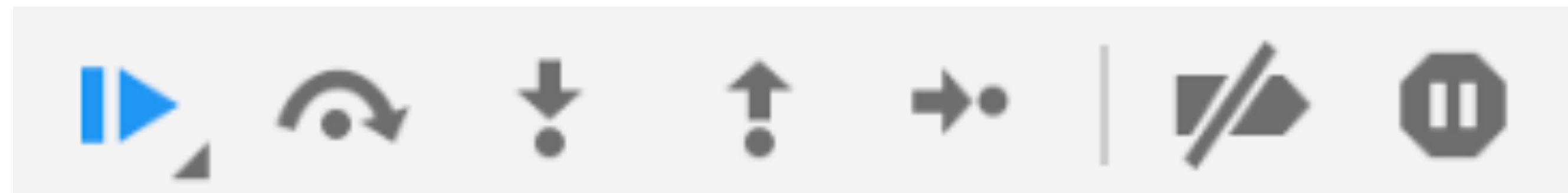
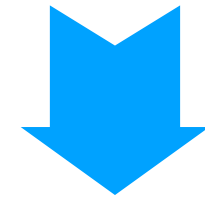
The [Step](#) icon allows you to execute the next code statement (i.e., “step”) in your program.



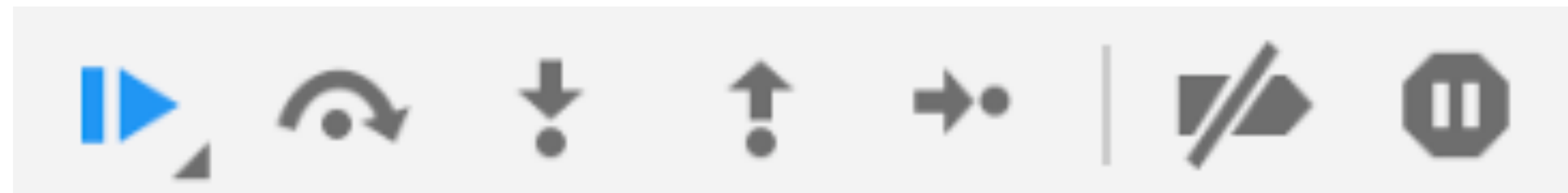
The [Step Over](#) icon allows you to execute the next function call *without* stepping into it. This is useful if the function you're currently paused in isn't relevant to the bug you're trying to fix.



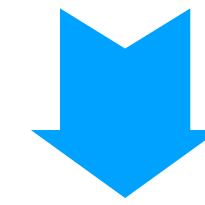
If you have paused on a function call that *is* relevant to the code you're debugging, the [Step Into](#) icon allows you to investigate that function.



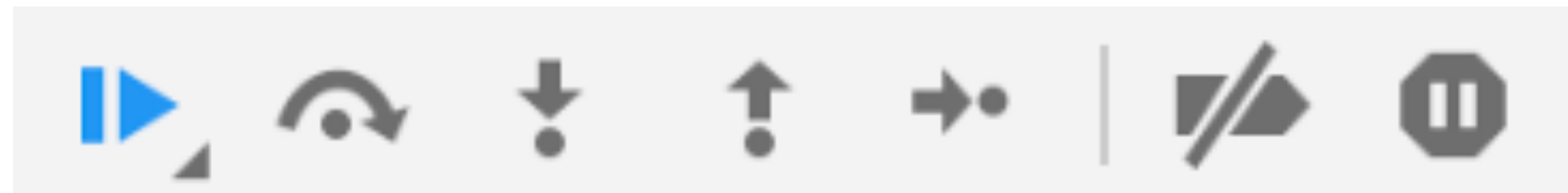
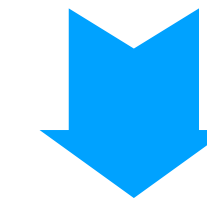
If you have paused **inside** a function call that is **not** relevant to the code you're debugging, the [Step Out](#) icon allows you to execute the remainder of that function's code.



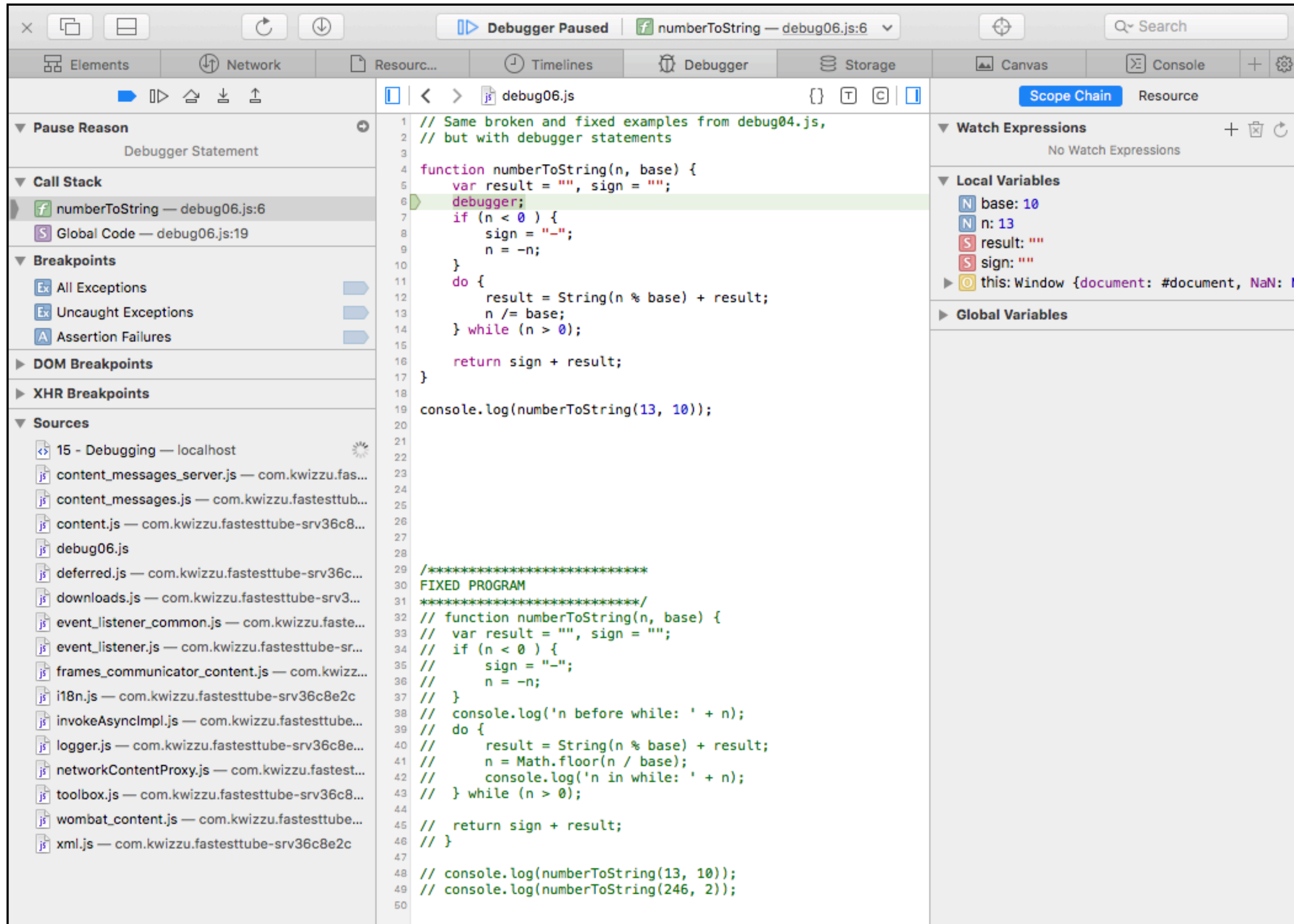
This [Disable Breakpoints](#) icon allows you to disable any breakpoint flags you've set in the Debugger pane.



And lastly, this icon allows you to [Pause on Exceptions](#).



[For a nice Chrome-specific tutorial, check out [“Get Started with Debugging JavaScript in Chrome DevTools.”](#)]



Let's take a deep breath and step into debug06.js

DEBUGGING TIP #7

USE VERSION CONTROL.

Undoing Merges

Now that you know how to create a merge commit, you'll probably make some by mistake. One of the great things about working with Git is that it's okay to make mistakes, because it's possible (and in many cases easy) to fix them.

Merge commits are no different. Let's say you started work on a topic branch, accidentally merged it into `master`, and now your commit history looks like this:

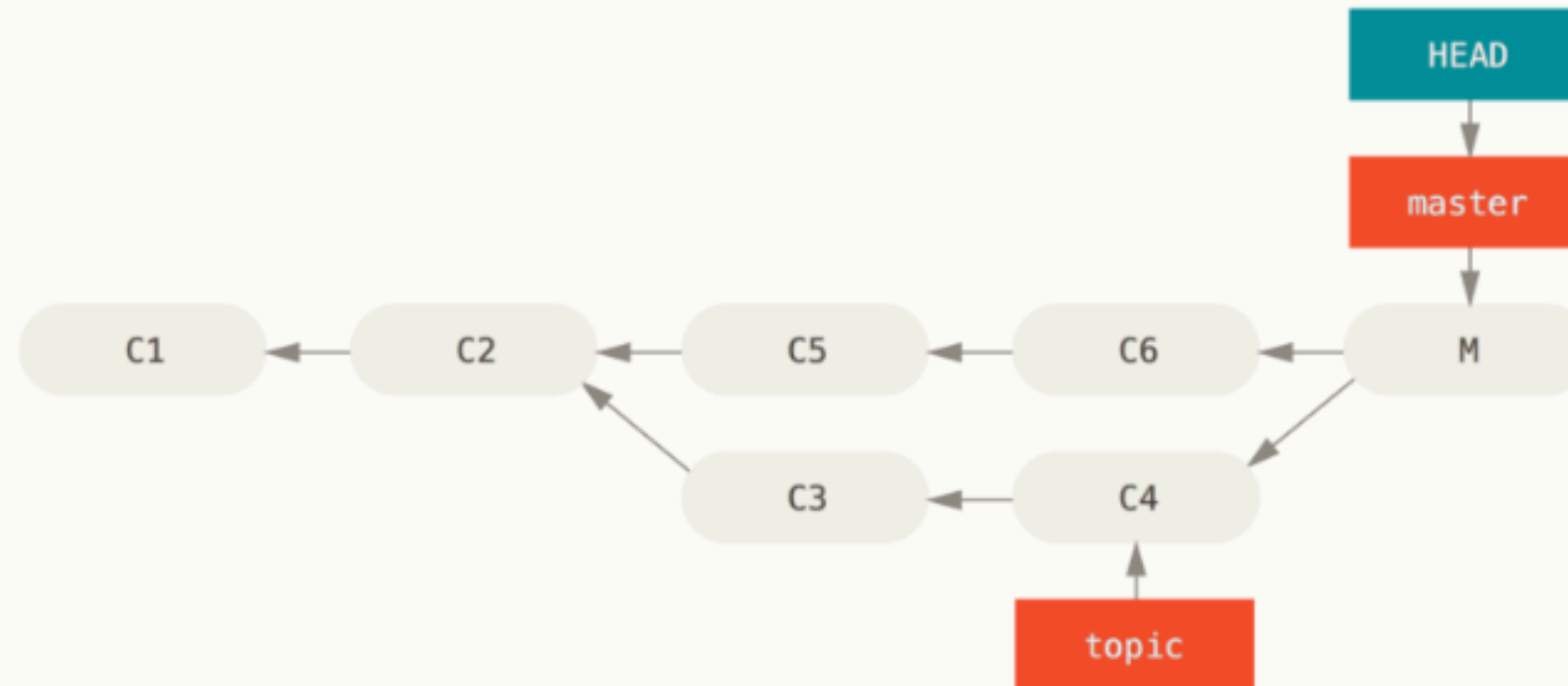


Figure 138. Accidental merge commit

There are two ways to approach this problem, depending on what your desired outcome is.

Here's one of thousands of resources on how to undo accidental git merges.

DEBUGGING TIP #8

SEARCH THE WEB FOR YOUR ERROR.

Here's a warning I saw in multiple assignments:

```
⚠ ▶ Phaser.Sound: Audio source already exists phaser.min.js:3
```

What's this error, and how do we fix it?

This took around 2 minutes to fix.



AllShoppingVideosImagesNewsMoreSettingsTools

About 141,000 results (0.41 seconds)

javascript - Phaser warning "Audio source already exists" when mp3 ...

<https://stackoverflow.com/.../phaser-warning-audio-source-already-exists-when-mp3-s...>

1 answer

Jun 14, 2018 - Well, as far as I can see you code seems to be doing things right. So I will try to answer your questions with the knowledge I have: 1. What does ...

javascript - How to end a state in **Phaser** as well as navigate back ...

Jun 21, 2017

javascript - Upgrading from **Phaser** v2.0.1 to v2.7.7 breaks **sound** ...

Apr 23, 2017

typescript - Playing **audio** with **Phaser** - Stack Overflow

Sep 9, 2016

cordova - Phonegap + **Phaser** game shuts down background **audio** ...

Dec 15, 2015

More results from stackoverflow.com



stackoverflow

Search...

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

Teams

Private Q&A

Create Team

1 Answer

activeoldestvotes

1

Well, as far as I can see you code seems to be doing things right. So I will try to answer your questions with the knowledge I have:

1. What does this message "Audio source already exists" mean ? Or should I ignore it?

The message means that there is already an instance of that sound playing as you can see in the place where it is raised:

```
if (this._sound && ***!this.allowMultiple***)
{
    console.warn('Phaser.Sound: Audio source already exists');

    // this._disconnectSource();
}
```

It will throw this error if the sound you are trying to play is already being played by Phaser.Sound and if is not allowMultiple... There it is the quid of the issue. AllowMultiple from [source code](#):

```
/**
 * @property {boolean} allowMultiple - This will allow you to have multiple instances of this
 * @default
 */
this.allowMultiple = false;
```

So basically is complaining that you are trying to spawn several instances of a sound that is not being allow multiple times. You shouldnt ignore it, but instead use the right flags.

Questions 2 and 3:

You shouldnt have re-add the resource, since thats why you load in the engine the source of audio, to can be reused through all the levels. Nor you have to do it for all the states.

In order to reuse an Sound in multiple states, you should be able to add the audio or any game object in the global scope and access it ([Here I found someone trying to do what you ask in the question](#)) Other ways will be to add this resources as an attribute to the game object, so you dont contaminate the global scope but only the Game object context. But I believe that is better strategy to **add this audios in different states** and manage their deletion/creation in the states. Mainly because JS is evil* and **mutability may play a bad card on you**

*Not that evil

To resolve this warning: Simply use the flag [allowMultiple](#) ([created in here](#)), eg:

```
this.sound1 = this.game.add.audio("button") // allowMultiple is false by default
this.sound2 = this.game.add.audio("punch");
// Allow multiple instances running at the same time for sound2
this.sound2.allowMultiple = true;
this.sound3 = this.game.add.audio("coin");
// Allow multiple instances running at the same time for sound3
this.sound3.allowMultiple = true;
```

shareimprove this answer

edited Jun 21 at 13:50

answered Jun 21 at 13:44

SirPeople

1,924 • 5 • 19

DEBUGGING TIP #9

SLEEP ON IT. LET THE BRAIN DO ITS WORK.



WHERE MANY ACTIONSCRIPT BUGS WERE SOLVED

DEBUGGING TIP #10

GRAB ANOTHER SET OF EYES AND HANDS.

[Let's test our newfound debugging powers on debug07.js]

MORE RESOURCES...

[The [Rithm School](#) section on Debugging JavaScript is short and succinct.]



[Learn Full Time](#) ▾

[Coding Workshops](#)

[Free Online Courses](#)

[Partnerships](#) ▾

[About](#) ▾

[Apply Now](#)

{ Rithm School Intermediate JavaScript Part I. }

You've made it to the first part of Rithm's free Intermediate JavaScript course! You're ready to learn how to debug like a professional, access and create values in more complex data structures, dive deep into callbacks and closures, and learn all about event-driven programming with the DOM. You'll also learn about some more advanced built-in methods on arrays. By the end of the course, you'll be able to build interactive browser based games and applications with JavaScript! If you want to go back and review some fundamentals, head over to our [JavaScript Fundamentals course](#).

The goal of this material is to get you familiar with intermediate JavaScript to prepare you for our full time program. When you're ready, get started with [Debugging JavaScript](#). Please be sure to let us know if you have any questions as you go along. Good luck!

1 Debugging JavaScript

⌚ 3 sections, 1 - 2 hours



JavaScript Errors



Debugging with the Sources Tab



Debugging Exercises



POPULAR: [JavaScript Promises](#) [fetch API](#) [React.js](#) [Cache API](#) [ES6 Features](#) [Node.js](#) [JavaScript](#) [jQuery](#)

JavaScript Errors and How to Fix Them

By [Jani Hartikainen](#) on January 22, 2015



JavaScript can be a nightmare to debug: Some errors it gives can be very difficult to understand at first, and the line numbers given aren't always helpful either. Wouldn't it be useful to have a list where you could look to find out what they mean and how to fix them? Here you go!



Below is a list of the strange errors in JavaScript. Different browsers can give you different messages for the same error, so there are several different examples where applicable.

How to read errors?

Before the list, let's quickly look at the structure of an error message. Understanding the structure helps understand the errors, and you'll have less trouble if you run into any errors not listed here.

A typical error from Chrome looks like this:

Popular Topics



[.htaccess](#) [AJAX](#) [Canvas & SVG](#) [CSS](#) [Dojo](#) [Firefox OS](#)
[HTML5](#) [JavaScript](#) [jQuery](#) [Media](#) [Mobile](#) [MooTools](#)
[Node.js](#) [Performance](#) [PHP](#) [SEO](#) [Shell](#) [WordPress](#)

Popular Features



[How to Create a RetroPie on Raspberry Pi - Graphical Guide](#)

[Conquering Impostor Syndrome](#)

[Being a Dev Dad](#)

[JavaScript Promise API](#)

[7 Essential JavaScript Functions](#)

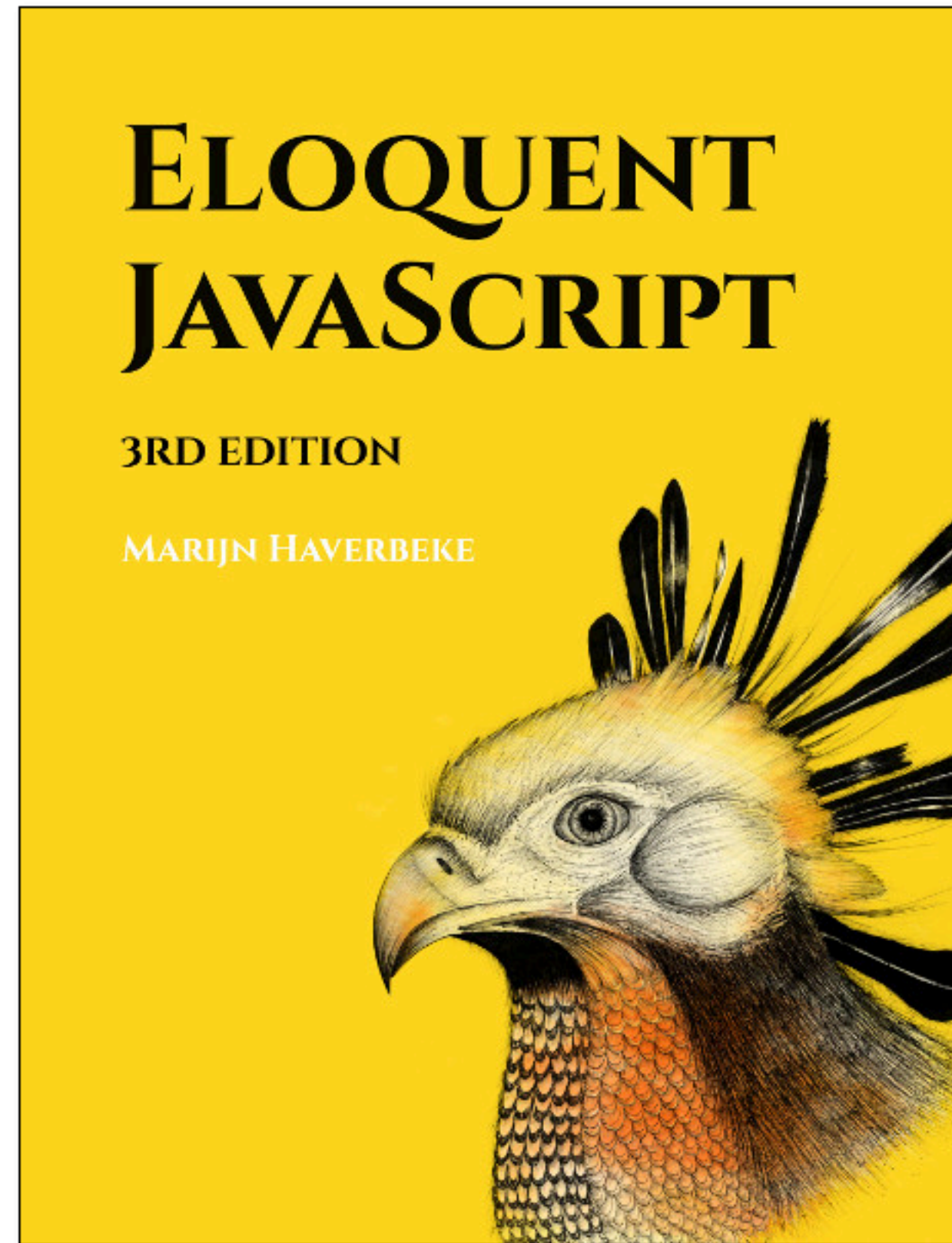
[I'm an Impostor](#)

[fetch API](#)

ELOQUENT JAVASCRIPT

3RD EDITION

This is a book about JavaScript, programming, and the wonders of the digital. You can read it online here, or get your own [paperback copy](#) of the *second* edition. A paper third edition is expected to be available this October.



Written by Marijn Haverbeke.

CONTENTS

Introduction

1. Values, Types, and Operators (Part 1: Language)
2. Program Structure
3. Functions
4. Data Structures: Objects and Arrays
5. Higher-order Functions
6. The Secret Life of Objects
7. Project: A Robot
8. Bugs and Errors
9. Regular Expressions
10. Modules
11. Asynchronous Programming
12. Project: A Programming Language
13. JavaScript and the Browser (Part 2: Browser)
14. The Document Object Model
15. Handling Events
16. Project: A Platform Game
17. Drawing on Canvas
18. HTTP and Forms
19. Project: A Pixel Art Editor
20. Node.js (Part 3: Node)
21. Project: Skill-Sharing Website

And let's not forget our excellent online textbook:
[Chapter 8: Bugs and Errors](#)