Learn How To Debug JavaScript with Chrome DevTools

Ditch console.log debugging once and for all! Learn how to use breakpoints to debug code within the Chrome Developer Tools

Brandon Morelli Sep 7

Creator of @codeburstio — Frequently posting web development tutorials & articles. Follow me on Twitter too: @BrandonMorelli

As a new developer, finding and fixing bugs can be difficult. You may be tempted to randomly use `console.log()` in an attempt to get your code working correctly. **Not Anymore!**

This article is all about debugging the correct way! You'll learn how to use the Chrome Developer Tools to set up breakpoints and step through your code. This workflow is often a much more efficient way to find and fix bugs in your code.

This tutorial shows you how to debug one specific issue, but the general workflow you learn is helpful for debugging all types of JavaScript bugs.

Step 1: Reproduce the bug

Reproducing the bug is always the first step to debugging. To "reproduce the bug" means to find a series of actions that consistently causes the bug to appear. You may need to reproduce the bug many times, so try to eliminate any unnecessary steps.

Follow along with the instructions below to reproduce the bug that you're going to fix in this tutorial.

Here is the webpage we'll be working with in this tutorial. Make sure to

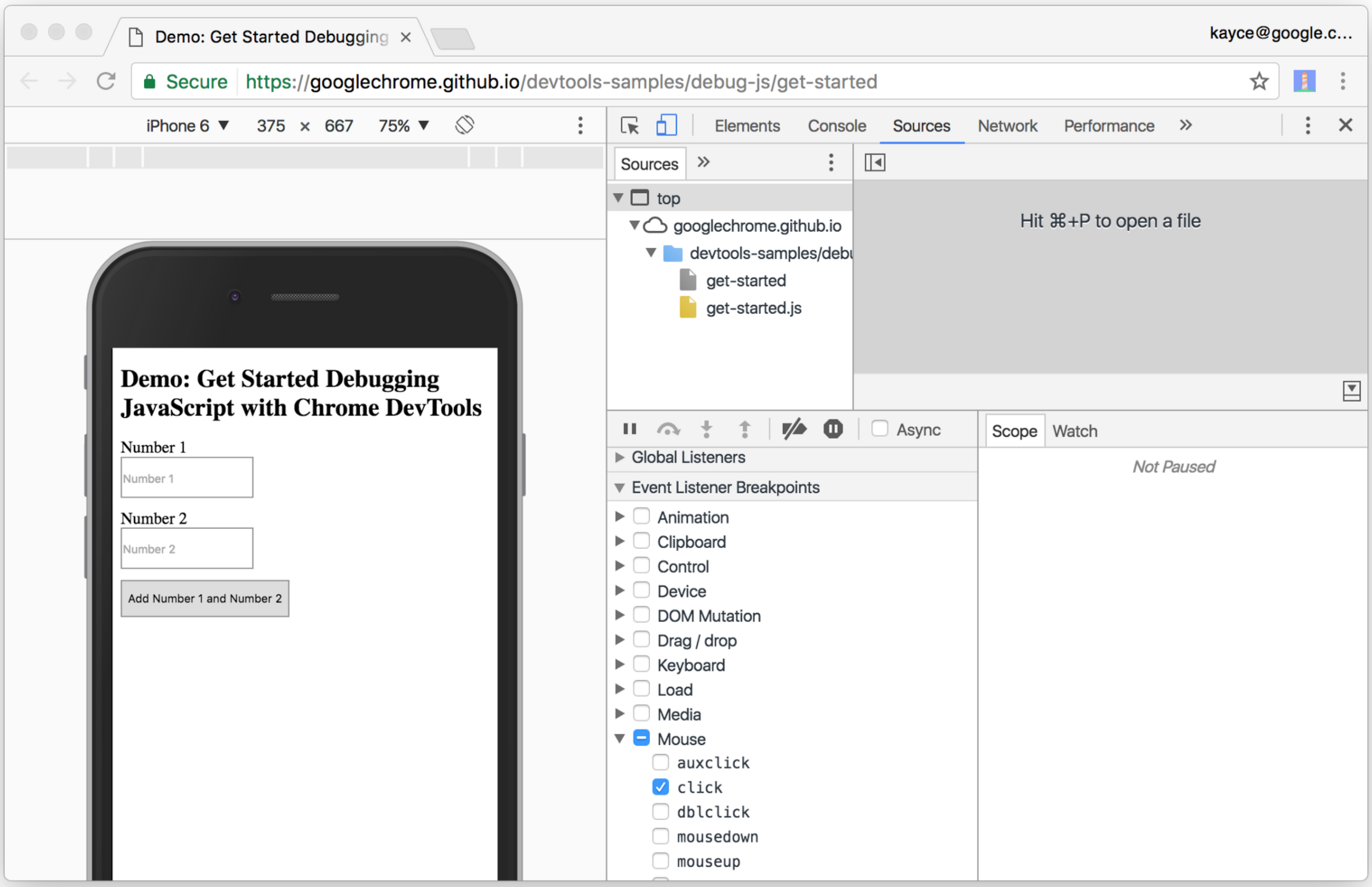
open this page in a new tab: OPEN DEMO.
In the demo, enter `5` for **Number 1**.
Enter `1` for **Number 2**.
Click **Add Number 1 and Number 2**.
Look at the label below the inputs and button. It says `5 + 1 = 51`.

Whoops. That result is wrong. The result should be `6`. This is the bug that you're going to fix.

Step 2: Pause the code with a breakpoint

DevTools lets you pause your code in the middle of its execution, and examine the values of *all* variables at that moment in time. The tool for pausing your code is called a **breakpoint**. Try it now:

Go back to the demo and open DevTools by pressing Command+Option+I (Mac) or Control+Shift+I (Windows, Linux).
Click the **Sources** tab.
Click **Event Listener Breakpoints** to expand the section. DevTools reveals a list of expandable event categories, such as **Animation** and **Clipboard**.
Next to the **Mouse** event category, click **Expand**
Check the **click** checkbox.



Back on the demo, click **Add Number 1 and Number 2** again. DevTools pauses the demo and highlights a line of code in the **Sources** panel. DevTools highlights this line of code:

```
function onClick() {
```

Why?

When you select ‘**click**’, you set up an event-based breakpoint on all `click` events. When *any* node is clicked, and that node has a `click` handler, DevTools automatically pauses on the first line of that node's`click` handler.

Step 3: Step through the code

One common cause of bugs is when a script executes in the wrong order. Stepping through your code enables you to walk through your code’s execution, one line at a time, and figure out exactly where it’s executing in a different order than you expected. Try it now:

On the **Sources** panel of DevTools, click **Step into next function call**

Step into next function call button

This button allows you to step through the execution of the `onClick()` function, one line at a time. Stop when DevTools highlights the following line of code:

```
if (inputsAreEmpty()) {
```

Now click the **Step over next function call** button**:**

Step over next function call button

This tells DevTools to execute `inputsAreEmpty()` without stepping into it. Notice how DevTools skips a few lines of code. This is because `inputsAreEmpty()` evaluated to false, so the `if` statement's block of code didn't execute.

That's the basic idea of stepping through code. If you look at the code in `get-started.js`, you can see that the bug is probably somewhere in the `updateLabel()` function. Rather than stepping through every line of code, you can use another type of breakpoint to pause the code closer to the location of the bug.

Step 4: Set another breakpoint

Line-of-code breakpoints are the most common type of breakpoint. When you've got a specific line of code that you want to pause on, use a line-of-code breakpoint. Try it now:

Look at the last line of code in `updateLabel()`, which looks like this:

```
label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;
```

To the left of this code, you can see the line number of this particular line of code: **32**. Click on **32**. DevTools puts a blue icon on top of **32**. This means that there is a line-of-code breakpoint on this line. DevTools now always pauses before this line of code is executed.

Click the **Resume script execution** button:

▶

Resume script execution button

The script continues executing until it reaches the line of code you placed the breakpoint on.

Look at the lines of code in `updateLabel()` that have already executed. DevTools prints out the values of `addend1`, `addend2`, and `sum`.

The value of `sum` looks suspicious. It looks like it's being evaluated as a string, when it should be a number. This may be the cause of the bug.

Step 5: Check variable values

Another common cause of bugs is when a variable or function produces a different value than expected. Many developers use `console.log()` to see how values change over time, but `console.log()` can be tedious and ineffective for two reasons. One, you may need to manually edit your code with lots of calls to `console.log()`. Two, you may not know exactly which variable is related to the bug, so you may need to log out many variables.

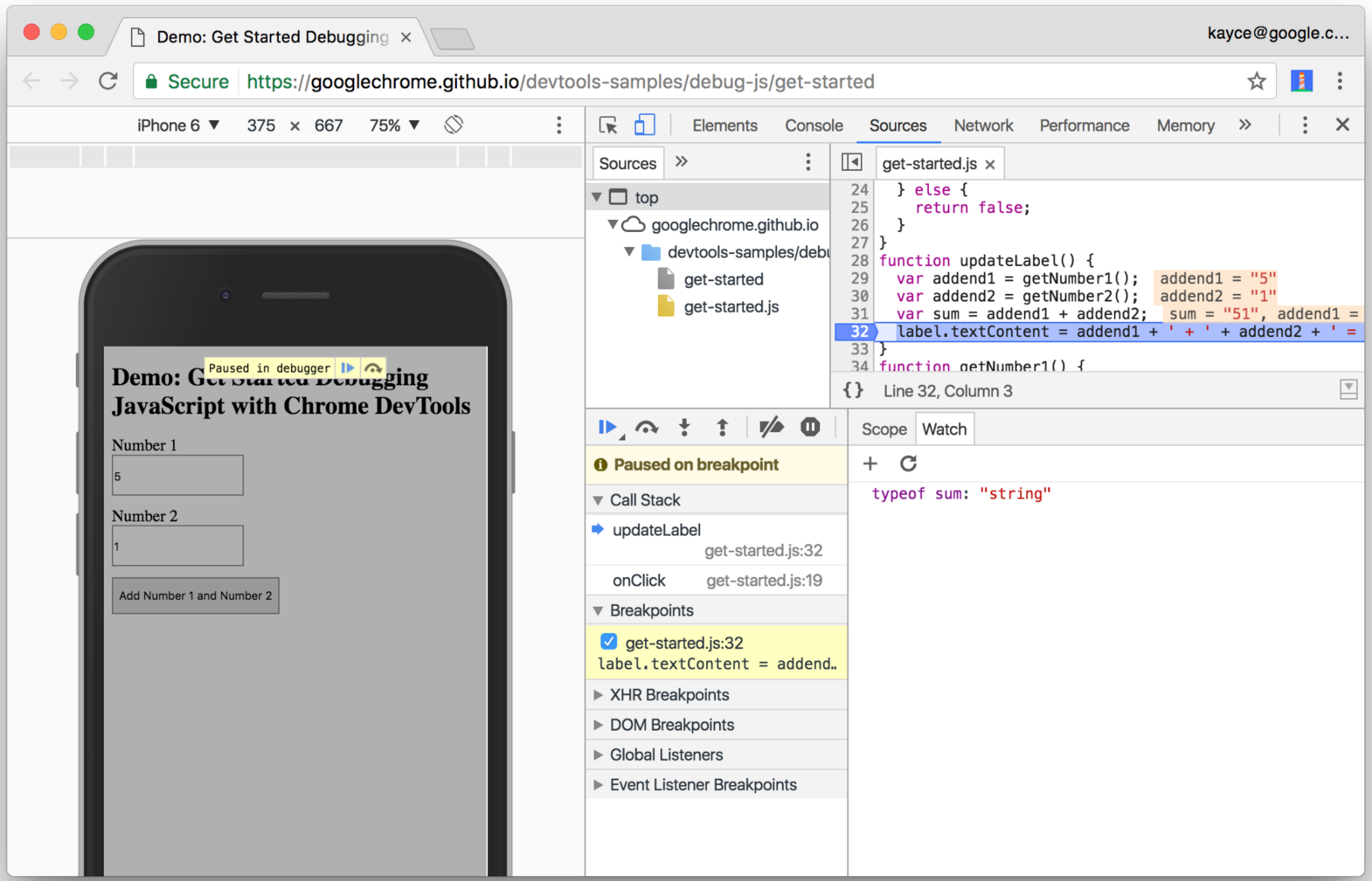
One DevTools alternative to `console.log()` is Watch Expressions. Use Watch Expressions to monitor the value of variables over time. As the name implies, Watch Expressions aren't just limited to variables. You can store any valid JavaScript expression in a Watch Expression. Try it now:

On the **Sources** panel of DevTools, click **Watch**. The section expands. Click **Add Expression**

+

Add Expression button

Type `typeof sum`.
Press Enter. DevTools shows `typeof sum: "string"`. The value to the right of the colon is the result of your Watch Expression.


Demo: Get Started Debugging
https://googlechrome.github.io/devtools-samples/debug-js/get-started

```
  } else {
    return false;
  }
}
function updateLabel() {
  var addend1 = getNumber1();  addend1 = "5"
  var addend2 = getNumber2();  addend2 = "1"
  var sum = addend1 + addend2;  sum = "51", addend1 =
  label.textContent = addend1 + ' + ' + addend2 + ' =
}
function getNumber1() {
```

Line 32, Column 3

Paused on breakpoint

Call Stack: updateLabel get-started.js:32; onClick get-started.js:19

Breakpoints: get-started.js:32 `label.textContent = addend…`

```
typeof sum: "string"
```

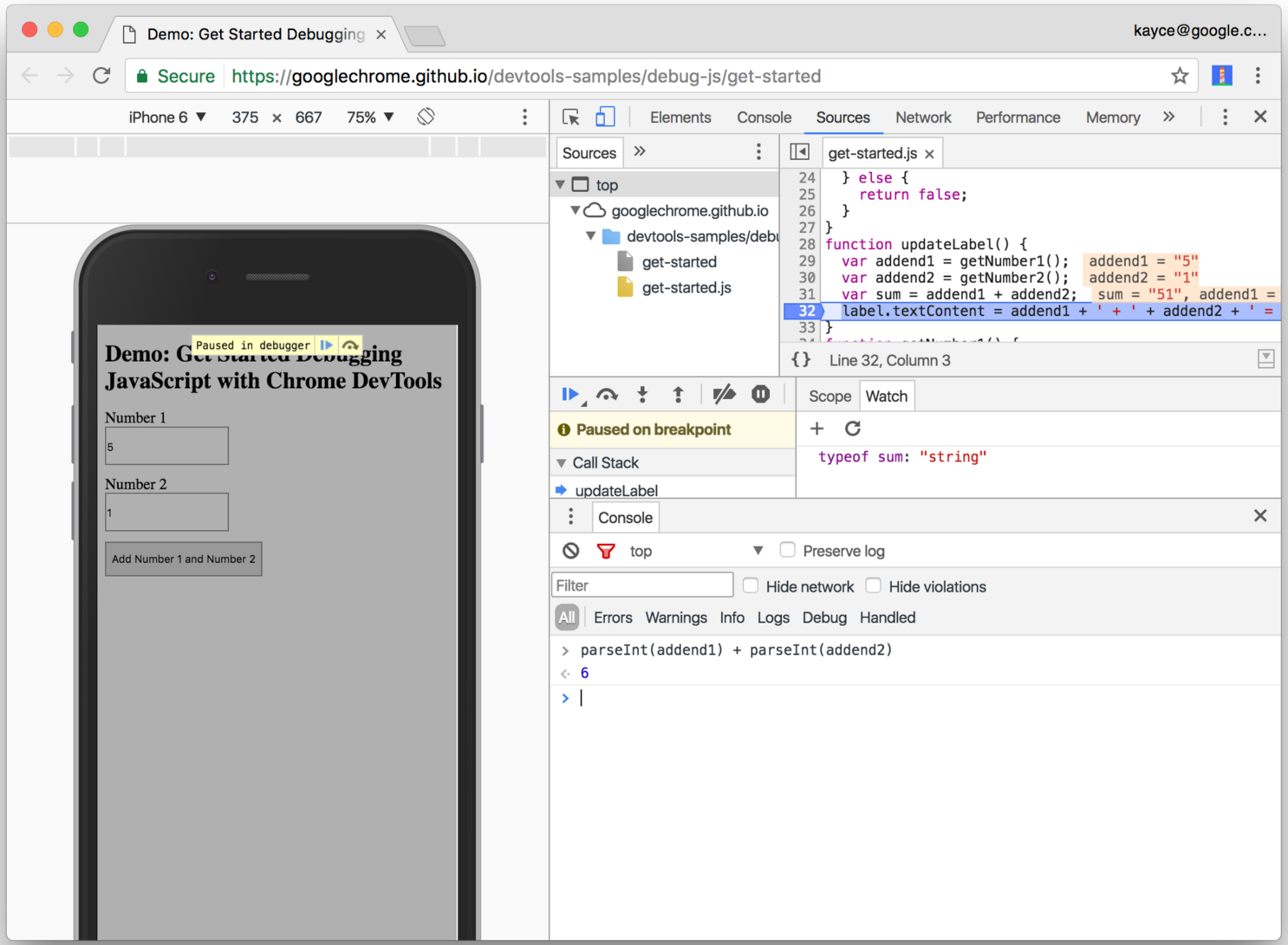
Paused in debugger
Demo: Get Started Debugging JavaScript with Chrome DevTools
Number 1
5
Number 2
1
Add Number 1 and Number 2

As suspected, `sum` is being evaluated as a string, when it should be a number. This is the cause of the demo's bug.

A second DevTools alternative to `console.log()` is the Console. Use the Console to evaluate arbitrary JavaScript statements. Developers commonly use the Console to overwrite the variable values when debugging. In your case, the Console can help you test out potential fixes for the bug you just discovered. Try it now:

If you don't have the Console drawer open, press Escape to open it. It opens at the bottom of your DevTools window.
In the Console, type `parseInt(addend1) + parseInt(addend2)`.
Press Enter. DevTools evaluates the statement and prints out `6`, which is

the result you expect the demo to produce.


Demo: Get Started Debugging
kayce@google.c...
Secure https://googlechrome.github.io/devtools-samples/debug-js/get-started
iPhone 6 375 × 667 75%
Elements Console Sources Network Performance Memory
Sources
top
googlechrome.github.io
devtools-samples/debu
get-started
get-started.js
Paused in debugger
Demo: Get Started Debugging JavaScript with Chrome DevTools
Number 1
Number 2
Add Number 1 and Number 2
Paused on breakpoint
Call Stack
updateLabel
Scope Watch
Console
top
Preserve log
Filter
Hide network
Hide violations
All Errors Warnings Info Logs Debug Handled

```
  } else {
    return false;
  }
}
function updateLabel() {
  var addend1 = getNumber1();   addend1 = "5"
  var addend2 = getNumber2();   addend2 = "1"
  var sum = addend1 + addend2;   sum = "51", addend1 =
  label.textContent = addend1 + ' + ' + addend2 + ' =
}
```

Line 32, Column 3

```
typeof sum: "string"
```

```
> parseInt(addend1) + parseInt(addend2)
< 6
> |
```

Step 6: Apply a fix

You’ve identified a potential fix for the bug. All that’s left is to try out your fix by editing the code and re-running the demo. You don’t need to leave DevTools to apply the fix. You can edit JavaScript code directly within the DevTools UI. Try it now:

In the code editor on the **Sources** panel of DevTools, replace `var sum = addend1 + addend2` with `var sum = parseInt(addend1) + parseInt(addend2);`. This is one line above where you are currently paused.
Press Command+S (Mac) or Control+S (Windows, Linux) to save your change. The background of the code changes to red to indicate that the script has been changed within DevTools.
Click **Deactivate breakpoints**

Deactivate breakpoints button

It changes blue to indicate that it is active. While this is set, DevTools ignores any breakpoints you've set.

Click **Resume script execution**

Resume script execution button

Try out the demo with different values. The demo should now be calculating the sums correctly.

> *Portions of this page are modifications based on work created and shared by Google and used according to terms described in the Creative Commons 3.0 Attribution License. This article was adapted from: Get Started with Debugging JavaScript in Chrome DevTools by Kayce Basques.*

Want more JavaScript? Check out my recent tutorials/articles:

August JavaScript Recap: 17 Articles and Tutorials I wrote this month!
JavaScript — WTF is ES6, ES8, ES 2017, ECMAScript… ?
Learn JavaScript ES6 — Array.find & Array.findIndex

If this post was helpful, please click the clap 👏button below a few times to show your support! ⬇⬇

Show your support

Clapping shows how much you appreciated Brandon Morelli's story.