**Debugging**

**Let's talk about debugging JavaScript with Chrome DevTools.**

Debugging JavaScript can be a terrifying thing for new developers. If your code already isn't working, how are you going to find out why it isn't working?

So we'll talk about debugging in this Hot Tip. *Before we talk about debugging we have to talk about what happens when the check engine light turns on in your car (If you don't like analogies skip the next 3 paragraphs)*:

If you are like me, when your check engine light comes in on your car you feel helpless. You don't think that you can fix it on your own so you drive to a professional to use a diagnostic machine to tell you what is wrong with your car. And you hope they won't rip you off because they have this secret knowledge advantage over you.

The same thing can happen as a new developer. Your code is broken. It is very mysterious to you so you assume that you need to get someone highly skilled to look at your code and tell you what to do to fix it. As a new developer on a team you aren't worried about a teammate or a mentor ripping you off financially but you may worry that you are bugging them and pulling them away from their work, or family time or whatever they were doing before you came asking for help.

All that to say, you should always feel comfortable reaching out to others around you when you get stuck with your code. However, there are certain action steps you **can** **and should** take before reaching out to others asking for their time. It will allow your teammates to focus on their tasks and will help you become a better developer all at the same time!

**In this tutorial we will walk through how to solve a specific bug.** Hopefully, the workflow of solving this bug will help you solve bugs that you run into!

How can you know that this approach will work for you? Because I'm following the \*workflow that the Google Chrome team recommends (they are much smarter than I)!

**Step 1: Reproduce the bug**

To "reproduce a bug" means that you can reenact a series of actions to produce the same bug to appear. This is an important skill in the work place. Your teammates won't enjoy you saying that something "just doesn't work" and you don't take the effort to share the exact steps you are taking, the result you expected versus the result that actually took place.

So let's go ahead and track the series of events that leads to a bug that we are going to solve.

1. Go ahead and click [here](http://bit.ly/javascript-debugging) to open up our webpage that has the bug. Please be sure to open up this link in a new tab!

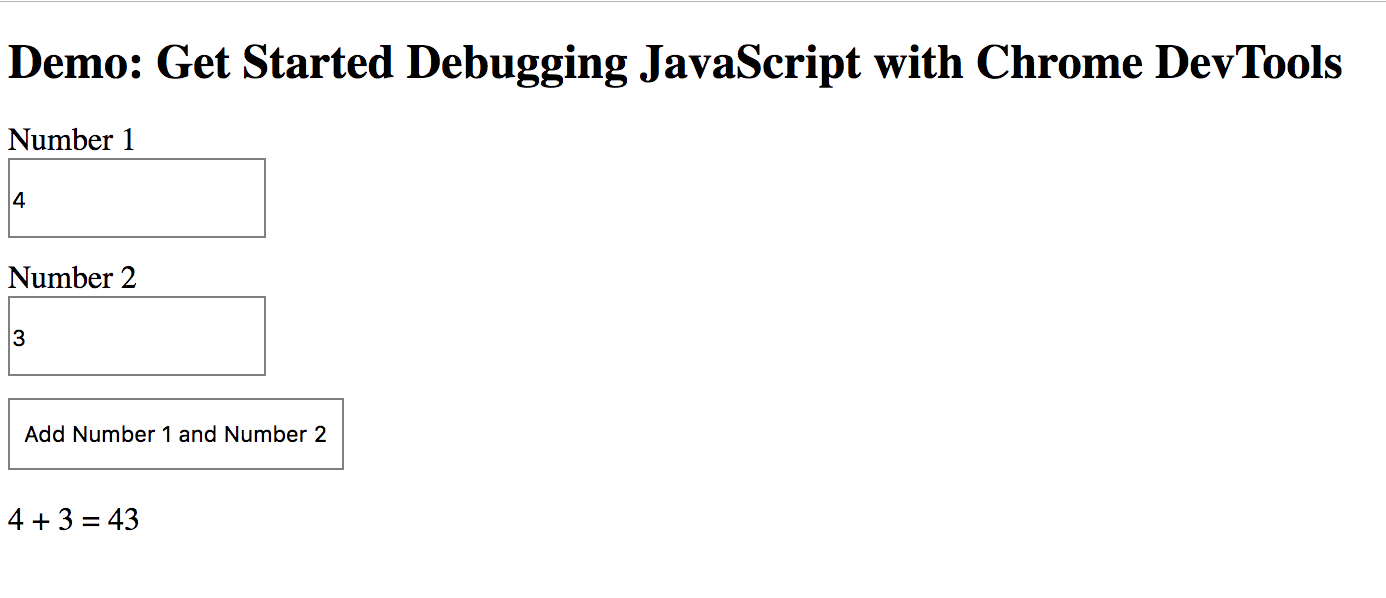
2. In the input field for **Number 1** provide the number 4.

3. In the input field for **Number 2** provide the number 3.

4. Click **Add Number 1 and Number 2 button**.

Result/Bug: 4 + 3 = 43

Expected Result 4 + 3 = 7



**Step 2: Introducing the Sources panel**

The sources panel is a great place to begin in debugging your JavaScript. You can read about the sources panel [here](https://developers.google.com/web/tools/chrome-devtools/sources) is you would like to take a deeper dive into it.

1. Open DevTools by pressing Command+Option+I (Mac) or Control+Shift+I (Windows, Linux). This shortcut opens the **Console** panel.

A screenshot of a computer

Description automatically generated

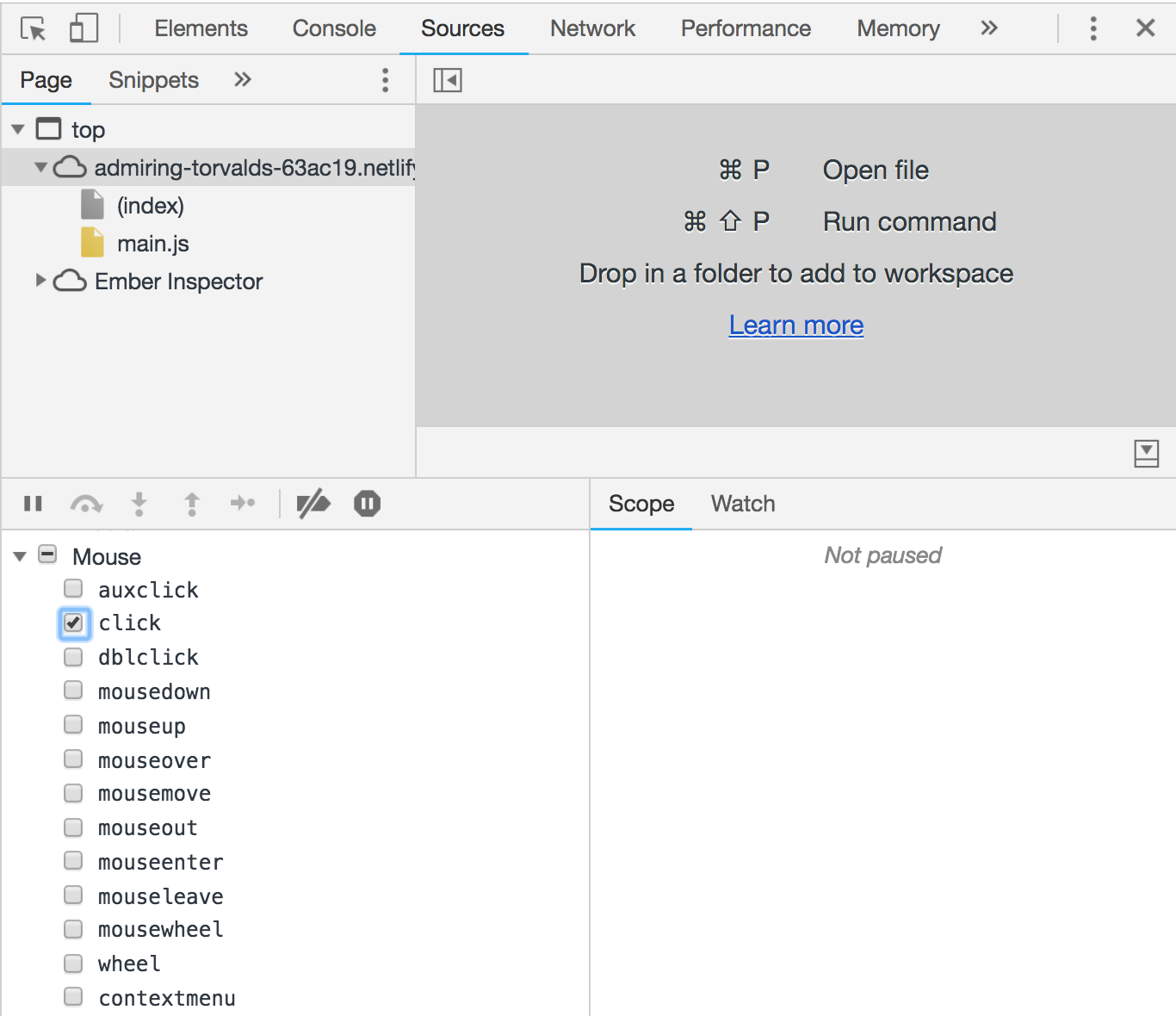
2. Now click on the **sources** tab.

A screenshot of a computer

Description automatically generated

3. Click on **Event Listener Breakpoints** to expand the section.

4. Then expand the **Mouse** event category and check the **click** box.



5. Go back to the demo and click the **ADD Number 1 and Number 2** button again. You should find that the dev tools highlights the line of code inside of the onClick function. You will also see in your window a message "Paused in debugger":

A screenshot of a computer

Description automatically generated

What Happened?

When we selected the '**click**' check box for mouse events we set up an event-based breakpoint on all click events. So any node that has a click handler will cause the DevTools to automatically pause on the first line of the click handler.

**Step 3: Step through the code**

Stop when DevTools highlights the following line of code. Setting the Event Listener Breakpoint listening for mouse clicks brought us into the first lie of the onClick() function:

1. if (inputsAreEmpty()) {

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

**Step over next function call button**

This tells DevTools to execute inputsAreEmpty(). Notice how DevTools skips a few lines of code. This is because inputsAreEmpty() evaluated to false, as we have provided values for both inputs, so the if statement's block of code didn't execute.

You should then see the following line of code highlighted as we stepped over to the next function call:

1. updateLabel()

You can then step into the next function calland that will take you into the first line of code in updateLabel().

So that is a brief introduction to how you can step through your code. If you check out the extra resources for this lesson I'll share in a video as I walk through this code step by step. If this is a new code base for you and you are new to development this code might feel overwhelming to you. I'll show you how to demystify how look at how your JavaScript, or someone else's JavaScript, is being ran.

However, if you have a bit more experience and look at the code in main.js, you can see that the bug is probably somewhere in the updateLabel() function. This is where we are getting the values for the two numbers and adding them together.

1. function updateLabel() {
2. var addend1 = getNumber1();
3. var addend2 = getNumber2();
4. var sum = addend1 + addend2;
5. label.textContent = addend1 + ' + ' + addend2 + ' = ' + sum;
6. }

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:

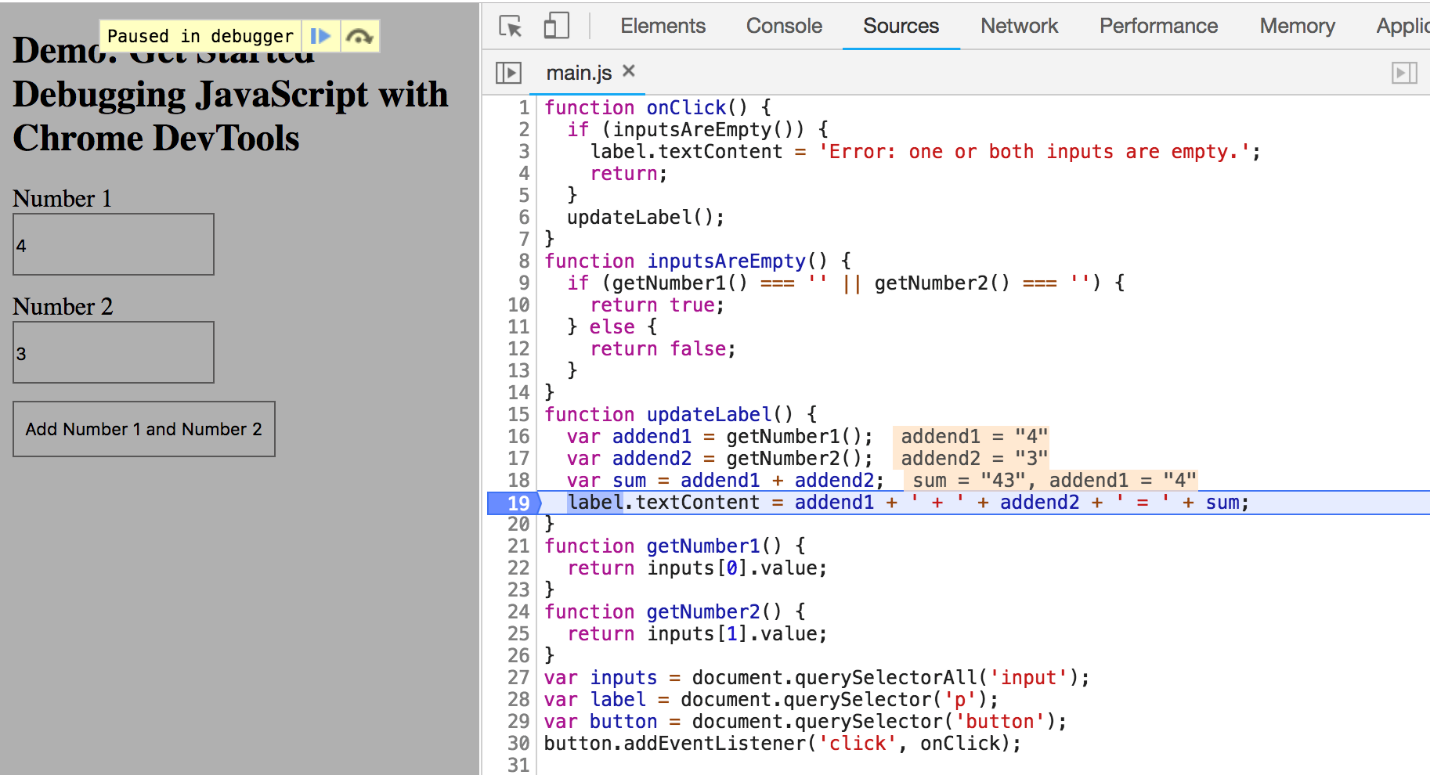
* Look at the last line of code in updateLabel(), which looks like this:

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

To the left of this code, you can see the line number of this particular line of code: **19**. Click on **19**. DevTools puts a blue icon on top of 19. 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. We are getting closer!

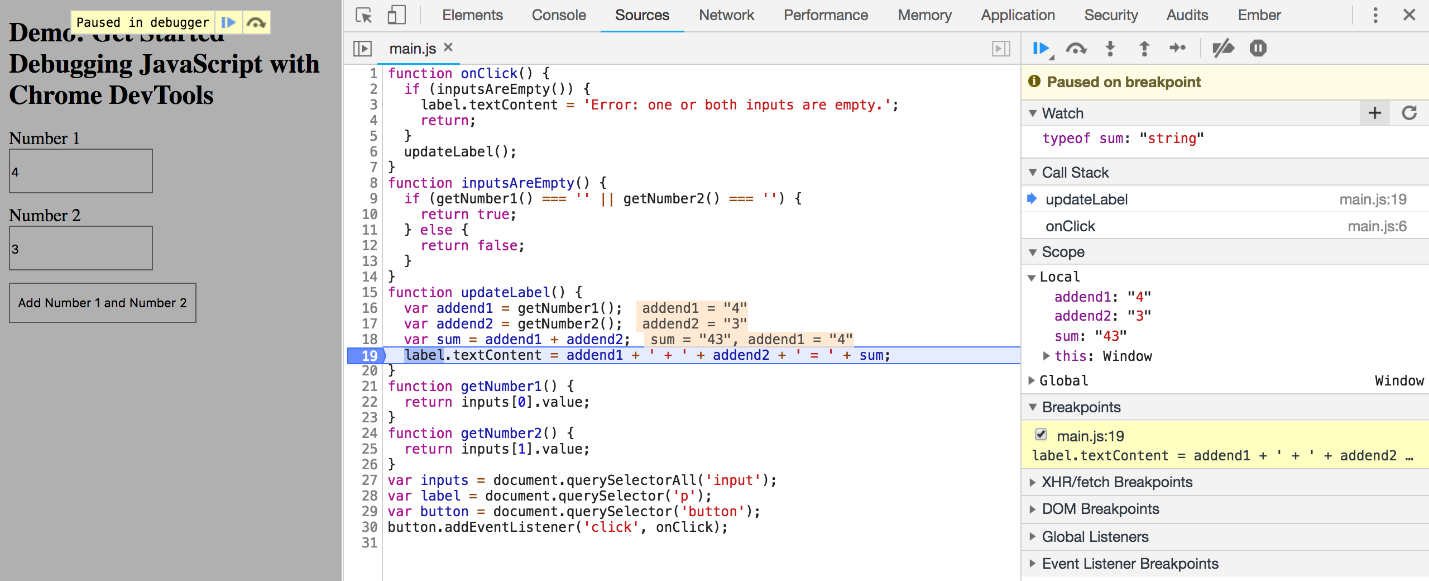
**Step 5: Check variable values**

This is where some developers start to use console.log's to debug their code. An alternative and more precise approach is to use 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. This may already be pre-populated for you. If it isn't:
* Press Enter. DevTools shows typeof sum: "string". The value to the right of the colon is the result of your Watch Expression.



As suspected, sum is being evaluated as a string, when it should be a number. This is certainly the cause of the demo's bug. Our program is simply doing string concatenation.

To see this in practice open up your console and type in following and notice:

1. "Hi, " + "neighbor."; // "Hi, neighbor."
2. // add numbers
3. 4 + 3; // 7
4. "4" + "3"; // "43"

When we add the string "Hi, " to the string "neighbor."    "Hi, neighbor."

When we add the number 4 to the number 3 we get the number 7.

When we add the string "4" to the string "3" we get the string "43".

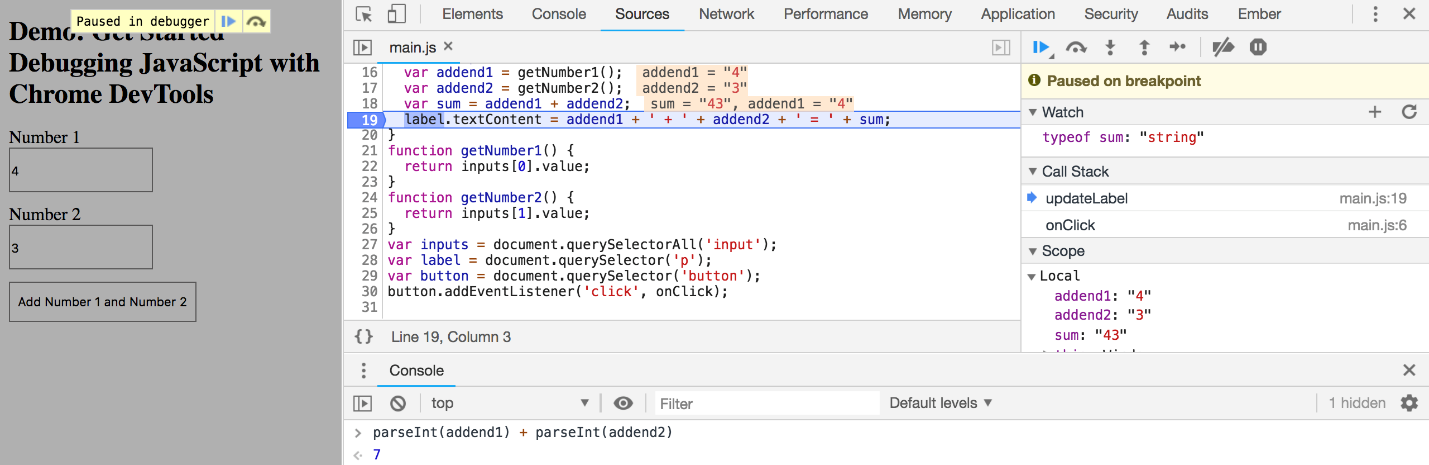
So this is where we start to pull from our JavaScript toolbox. We know what the problem is. Now, on to the solution!

What will turn a string into a number? If you didn't know you could Google that question and I'm certain one of the top search results would have been to use the parseInt() function! The parseInt function converts its first argument to a string, parses it, and returns an integer. That sounds like it could work for us!Try the following in your console:

1. parseInt("4"); // 4
2. parseInt("3"); // 3

Let's test this out in the console:

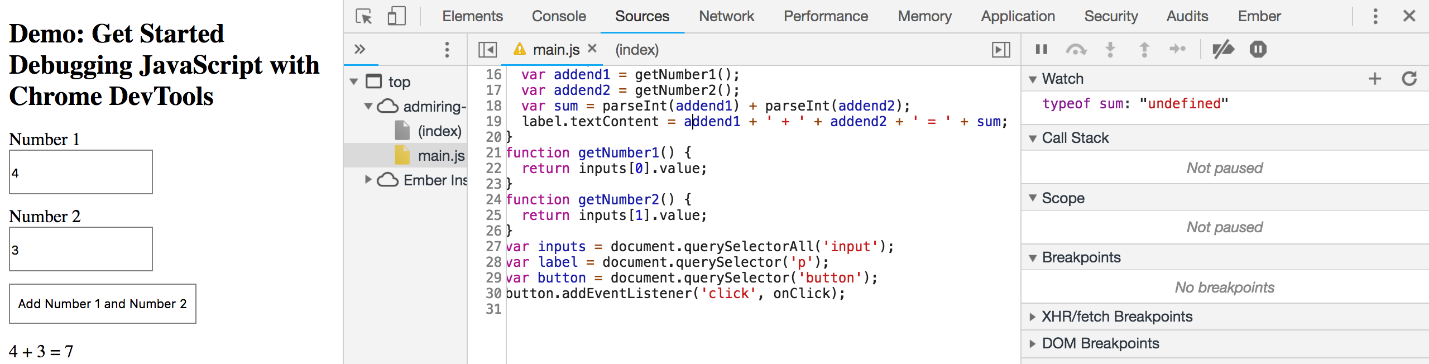
* If you don’t have the Console drawer open, press Escape to open it. It opens at the bottom of your DevTools window.
* In the Console, type parseInt(addend1) + parseInt(addend2).
* Press Enter. DevTools evaluates the statement and prints out 7, which is the result you expect the demo to produce.



**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:

1. 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.
   1. 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.
   2. Click **Deactivate breakpoints button**
2. It changes blue to indicate that it is active. While this is set, DevTools ignores any breakpoints you’ve set.
   1. Click **Resume script execution button**



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

When you are done be sure to remove your breakpoint from line 19.

Portions of this page are modifications based on work created and [shared by Google](https://developers.google.com/readme/policies/) and used according to terms described in the [Creative Commons 3.0 Attribution License](http://creativecommons.org/licenses/by/3.0/). This article was adapted from: [Get Started with Debugging JavaScript in Chrome DevTools](https://developers.google.com/web/tools/chrome-devtools/javascript/) by Kayce Basques.