Run JavaScript in the Console

Kayce Basques

This interactive tutorial shows you how to run JavaScript in the Chrome DevTools Console. See Get Started With Logging Messages to learn how to log messages to the Console. See Get Started With Debugging JavaScript to learn how to pause JavaScript code and step through it one line at a time.

Figure 1. The **Console**.

Overview

The **Console** is a REPL, which stands for Read, Evaluate, Print, and Loop. It reads the JavaScript that you type into it, evaluates your code, prints out the result of your expression, and then loops back to the first step.

Set up DevTools

This tutorial is designed so that you can open up the demo and try all the workflows yourself. When you physically follow along, you're more likely to remember the workflows later.

1. Press Command+Option+J (Mac) or Control+Shift+J (Windows, Linux, ChromeOS) to open the **Console**, right here on this very page.

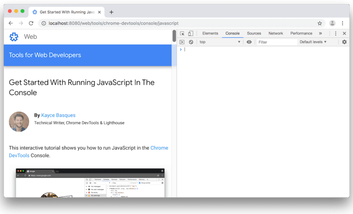

Figure 2. This tutorial on the left, and DevTools on the right.

View and change the page's JavaScript or DOM

When building or debugging a page, it's often useful to run statements in the **Console** in order to change how the page looks or runs.

1. Notice the text in the button below.

   Hello, World!

2. Type `document.getElementById('hello').textContent = 'Hello, Console!'` in the **Console** and then press Enter to evaluate the expression. Notice how the text inside the button changes.

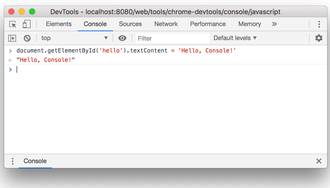

Figure 3. How the Console looks after evaluating the expression above.

Below the code that you evaluated you see `"Hello, Console!"`. Recall the 4 steps of REPL: read, evaluate, print, loop. After evaluating your code, a REPL prints the result of the expression. So `"Hello, Console!"` must be the result of evaluating `document.getElementById('hello').textContent = 'Hello, Console!'`.

Run arbitrary JavaScript that's not related to the page

Sometimes, you just want a code playground where you can test some code, or try out new JavaScript features you're not familiar with. The Console is a perfect place for these kinds of experiments.

1. Type `5 + 15` in the Console. The result `20` will appear below your expression (unless your expression takes too much time to evaluate).
2. Press `Enter` to evaluate the expression. The Console prints the result of the expression below your code. **Figure 4** below shows how your Console should look after evaluating this expression.
3. Type the following code into the **Console**. Try typing it out, character-by-character, rather than copy-pasting it.

```
function add(a, b=20) {
  return a + b;
}
```

   See define default values for function arguments if you're unfamiliar with the `b=20` syntax.

4. Now, call the function that you just defined.

```
add(25);
```

Figure 4. How the Console looks after evaluating the expressions above.

`add(25)` evaluates to `45` because when the `add` function is called without a second argument, `b` defaults to `20`.

You will not be able to run any code in this console session until your function has returned. If that takes too long, you can use the **Task Manager** to cancel the time-intensive computation; however, it will also cause the current page to fail and all data you have entered will be lost.

Next steps

See Run JavaScript to explore more features related to running JavaScript in the Console.

DevTools lets you pause a script in the middle of its execution. While you're paused, you can use the **Console** to view and change the page's `window` or `DOM` at that moment in time. This makes for a powerful debugging workflow. See Get Started With Debugging JavaScript for an interactive tutorial.

The **Console** also supports a set of format specifiers. See Format and style messages in the Console to explore all the method to format and style console messages.

Apart from that, the **Console** also has a set of convenience functions that make it easier to interact with a page. For example:

- Rather than typing `document.querySelector()` to select an element, you can type `$()`. This syntax is inspired by jQuery, but it's not actually jQuery. It's just an alias for `document.querySelector()`.
- `debug(function)` effectively sets a breakpoint on the first line of that function.
- `keys(object)` returns an array containing the keys of the specified object.

See Console Utilities API Reference to explore all the convenience functions.

Last updated 2018-04-18 UTC.