**Console Methods**

The console object provides access to the browser's debugging console (e.g. the Web console in Firefox). The specifics of how it works varies from browser to browser, but there is a de facto set of features that are typically provided.

The console object can be accessed from any global object. Window on browsing scopes and WorkerGlobalScope as specific variants in workers via the property console. It's exposed as Window.console, and can be referenced as console. For example:

JS

console.log("Failed to open the specified link");

[**Instance methods**](https://developer.mozilla.org/en-US/docs/Web/API/console#instance_methods)

1. [console.assert()](https://developer.mozilla.org/en-US/docs/Web/API/console/assert)

Log a message and stack trace to console if the first argument is false.

1. [console.clear()](https://developer.mozilla.org/en-US/docs/Web/API/console/clear)

Clear the console.

1. [console.count()](https://developer.mozilla.org/en-US/docs/Web/API/console/count)

Log the number of times this line has been called with the given label.

1. [console.countReset()](https://developer.mozilla.org/en-US/docs/Web/API/console/countReset)

Resets the value of the counter with the given label.

1. [console.debug()](https://developer.mozilla.org/en-US/docs/Web/API/console/debug)

Outputs a message to the console with the log level debug.

1. [console.dir()](https://developer.mozilla.org/en-US/docs/Web/API/console/dir)

Displays an interactive listing of the properties of a specified JavaScript object. This listing lets you use disclosure triangles to examine the contents of child objects.

1. [console.dirxml()](https://developer.mozilla.org/en-US/docs/Web/API/console/dirxml)

Displays an XML/HTML Element representation of the specified object if possible or the JavaScript Object view if it is not possible.

1. [console.error()](https://developer.mozilla.org/en-US/docs/Web/API/console/error)

Outputs an error message. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#using_string_substitutions) and additional arguments with this method.

1. console.exception() Non-standard Deprecated

An alias for error().

1. [console.group()](https://developer.mozilla.org/en-US/docs/Web/API/console/group)

Creates a new inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#using_groups_in_the_console), indenting all following output by another level. To move back out a level, call groupEnd().

1. [console.groupCollapsed()](https://developer.mozilla.org/en-US/docs/Web/API/console/groupCollapsed)

Creates a new inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#using_groups_in_the_console), indenting all following output by another level. However, unlike group() this starts with the inline group collapsed requiring the use of a disclosure button to expand it. To move back out a level, call groupEnd().

1. [console.groupEnd()](https://developer.mozilla.org/en-US/docs/Web/API/console/groupEnd)

Exits the current inline [group](https://developer.mozilla.org/en-US/docs/Web/API/console#using_groups_in_the_console).

1. [console.info()](https://developer.mozilla.org/en-US/docs/Web/API/console/info)

Informative logging of information. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#using_string_substitutions) and additional arguments with this method.

1. [console.log()](https://developer.mozilla.org/en-US/docs/Web/API/console/log)

For general output of logging information. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#using_string_substitutions) and additional arguments with this method.

1. [console.profile()](https://developer.mozilla.org/en-US/docs/Web/API/console/profile) Non-standard

Starts the browser's built-in profiler (for example, the [Firefox performance tool](https://firefox-source-docs.mozilla.org/devtools-user/performance/index.html)). You can specify an optional name for the profile.

1. [console.profileEnd()](https://developer.mozilla.org/en-US/docs/Web/API/console/profileEnd) Non-standard

Stops the profiler. You can see the resulting profile in the browser's performance tool (for example, the [Firefox performance tool](https://firefox-source-docs.mozilla.org/devtools-user/performance/index.html)).

1. [console.table()](https://developer.mozilla.org/en-US/docs/Web/API/console/table)

Displays tabular data as a table.

1. [console.time()](https://developer.mozilla.org/en-US/docs/Web/API/console/time)

Starts a [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#timers) with a name specified as an input parameter. Up to 10,000 simultaneous timers can run on a given page.

1. [console.timeEnd()](https://developer.mozilla.org/en-US/docs/Web/API/console/timeEnd)

Stops the specified [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#timers) and logs the elapsed time in milliseconds since it started.

1. [console.timeLog()](https://developer.mozilla.org/en-US/docs/Web/API/console/timeLog)

Logs the value of the specified [timer](https://developer.mozilla.org/en-US/docs/Web/API/console#timers) to the console.

1. [console.timeStamp()](https://developer.mozilla.org/en-US/docs/Web/API/console/timeStamp) Non-standard

Adds a marker to the browser performance tool's timeline ([Chrome](https://developer.chrome.com/docs/devtools/evaluate-performance/reference/) or [Firefox](https://profiler.firefox.com/docs/#/./guide-ui-tour-timeline)).

1. [console.trace()](https://developer.mozilla.org/en-US/docs/Web/API/console/trace)

Outputs a [stack trace](https://developer.mozilla.org/en-US/docs/Web/API/console#stack_traces).

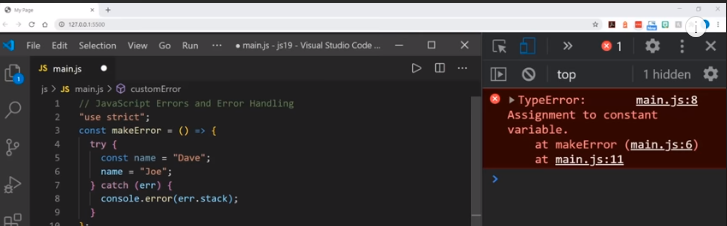
1. [console.warn()](https://developer.mozilla.org/en-US/docs/Web/API/console/warn)

Outputs a warning message. You may use [string substitution](https://developer.mozilla.org/en-US/docs/Web/API/console#using_string_substitutions) and additional arguments with this method.

**Important**

The non-standard stack property of an Error instance offers a trace of which functions were called, in what order, from which line and file, and with what arguments. The stack string proceeds from the most recent calls to earlier ones, leading back to the original

global scope call.



**Examples**

Outputting text to the console

The most frequently-used feature of the console is logging of text and other data. There are several categories of output you can generate, using the console.log(), console.info(), console.warn(), console.error(), or console.debug() methods. Each of these results in output styled differently in the log, and you can use the filtering controls provided by your browser to only view the kinds of output that interest you.

There are two ways to use each of the output methods; you can pass in a list of objects whose string representations get concatenated into one string, then output to the console, or you can pass in a string containing zero or more substitution strings followed by a list of objects to replace them.

Outputting a single object

The simplest way to use the logging methods is to output a single object:

JS

const someObject = { str: "Some text", id: 5 };

console.log(someObject);

The output looks something like this:

BASH

{str:"Some text", id:5}

Outputting multiple objects

You can also output multiple objects by listing them when calling the logging method, like this:

JS

const car = "Dodge Charger";

const someObject = { str: "Some text", id: 5 };

console.info("My first car was a", car, ". The object is:", someObject);

#### **Using string substitutions**

When passing a string to one of the console object's methods that accepts a string (such as log()), you may use these substitution strings:

%o or %O

Outputs a JavaScript object. Clicking the object name opens more information about it in the inspector.

%d or %i

Outputs an integer. Number formatting is supported, for example console.log("Foo %.2d", 1.1) will output the number as two significant figures with a leading 0: Foo 01.

%s

Outputs a string.

%f

Outputs a floating-point value. Formatting is supported, for example console.log("Foo %.2f", 1.1) will output the number to 2 decimal places: Foo 1.10.

**Note:** Precision formatting doesn't work in Chrome.

Each of these pulls the next argument after the format string off the parameter list. For example:

JSCopy to Clipboard

for (let i = 0; i < 5; i++) {

console.log("Hello, %s. You've called me %d times.", "Bob", i + 1);

}

The output looks like this:

Hello, Bob. You've called me 1 times.

Hello, Bob. You've called me 2 times.

Hello, Bob. You've called me 3 times.

Hello, Bob. You've called me 4 times.

Hello, Bob. You've called me 5 times.

#### Styling console output

You can use the %c directive to apply a CSS style to console output:

JSCopy to Clipboard

console.log(

"This is %cMy stylish message",

"color: yellow; font-style: italic; background-color: blue;padding: 2px",

);

The text before the directive will not be affected, but the text after the directive will be styled using the CSS declarations in the parameter.

Styled Text in Firefox console

You may use %c multiple times:

JSCopy to Clipboard

console.log(

"Multiple styles: %cred %corange",

"color: red",

"color: orange",

"Additional unformatted message",

);

The properties usable along with the %c syntax are as follows (at least, in Firefox — they may differ in other browsers):

* [background](https://developer.mozilla.org/en-US/docs/Web/CSS/background) and its longhand equivalents
* [border](https://developer.mozilla.org/en-US/docs/Web/CSS/border) and its longhand equivalents
* [border-radius](https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius)
* [box-decoration-break](https://developer.mozilla.org/en-US/docs/Web/CSS/box-decoration-break)
* [box-shadow](https://developer.mozilla.org/en-US/docs/Web/CSS/box-shadow)
* [clear](https://developer.mozilla.org/en-US/docs/Web/CSS/clear) and [float](https://developer.mozilla.org/en-US/docs/Web/CSS/float)
* [color](https://developer.mozilla.org/en-US/docs/Web/CSS/color)
* [cursor](https://developer.mozilla.org/en-US/docs/Web/CSS/cursor)
* [display](https://developer.mozilla.org/en-US/docs/Web/CSS/display)
* [font](https://developer.mozilla.org/en-US/docs/Web/CSS/font) and its longhand equivalents
* [line-height](https://developer.mozilla.org/en-US/docs/Web/CSS/line-height)
* [margin](https://developer.mozilla.org/en-US/docs/Web/CSS/margin)
* [outline](https://developer.mozilla.org/en-US/docs/Web/CSS/outline) and its longhand equivalents
* [padding](https://developer.mozilla.org/en-US/docs/Web/CSS/padding)
* text-\* properties such as [text-transform](https://developer.mozilla.org/en-US/docs/Web/CSS/text-transform)
* [white-space](https://developer.mozilla.org/en-US/docs/Web/CSS/white-space)
* [word-spacing](https://developer.mozilla.org/en-US/docs/Web/CSS/word-spacing) and [word-break](https://developer.mozilla.org/en-US/docs/Web/CSS/word-break)
* [writing-mode](https://developer.mozilla.org/en-US/docs/Web/CSS/writing-mode)

**Note:** The console message behaves like an inline element by default. To see the effects of padding, margin, etc. you should set it to for example display: inline-block.

### [Using groups in the console](https://developer.mozilla.org/en-US/docs/Web/API/console#using_groups_in_the_console)

You can use nested groups to help organize your output by visually combining related material. To create a new nested block, call console.group(). The console.groupCollapsed() method is similar but creates the new block collapsed, requiring the use of a disclosure button to open it for reading.

To exit the current group, call console.groupEnd(). For example, given this code:

JSCopy to Clipboard

console.log("This is the outer level");

console.group("First group");

console.log("In the first group");

console.group("Second group");

console.log("In the second group");

console.warn("Still in the second group");

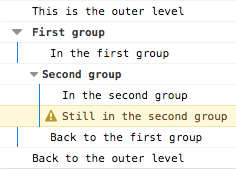
console.groupEnd();

console.log("Back to the first group");

console.groupEnd();

console.debug("Back to the outer level");

The output looks like this:



### [Timers](https://developer.mozilla.org/en-US/docs/Web/API/console#timers)

You can start a timer to calculate the duration of a specific operation. To start one, call the console.time() method, giving it a name as the only parameter. To stop the timer, and to get the elapsed time in milliseconds, just call the console.timeEnd() method, again passing the timer's name as the parameter. Up to 10,000 timers can run simultaneously on a given page.

For example, given this code:

JSCopy to Clipboard

console.time("answer time");

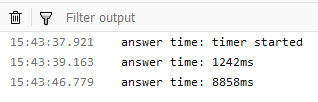
alert("Click to continue");

console.timeLog("answer time");

alert("Do a bunch of other stuff…");

console.timeEnd("answer time");

Will log the time needed by the user to dismiss the alert box, log the time to the console, wait for the user to dismiss the second alert, and then log the ending time to the console:



Notice that the timer's name is displayed both when the timer is started and when it's stopped.