Console API Reference

By Kayce Basques
Technical Writer for Chrome DevTools

By Meggin Kearney
Meggin is a Tech Writer

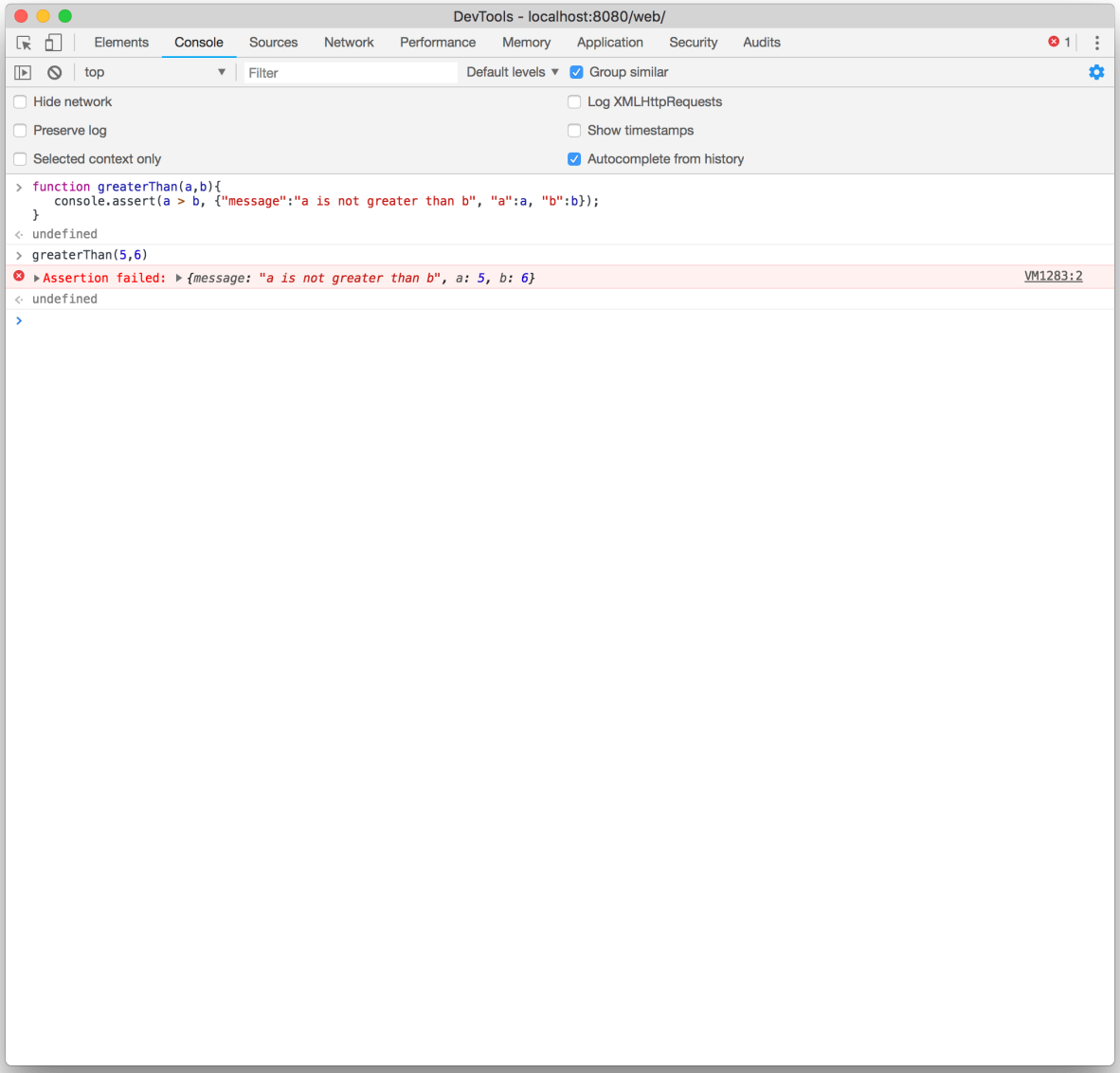
By Paul Bakaus
Open Web Developer Advocate at Google • Tools, Performance, Animation, UX

Use the Console API to write information to the console, create JavaScript profiles, and start a debugging session.

console.assert(expression, object)

Writes an error to the console when the evaluated expression is `false`.

```
function greaterThan(a,b) {
  console.assert(a > b, {"message":"a is not greater than b","a":a,"b":b});
}
greaterThan(5,6);
```



console.clear()

Clears the console.

```
console.clear();
```

If the **Preserve log** checkbox is enabled, `console.clear()` is disabled. However, pressing the **clear console** button (🚫) or typing the shortcut Ctrl+L while the Console is in focus still works.

See Clearing the console for more information.

console.count(label)

Writes the number of times that **`count()`** has been invoked at the same line and with the same label.

```
function login(name) {
  console.count(name + ' logged in');
}
```

```
> function login(name) {
    console.count(name + ' logged in');
  }
<· undefined
> login('john');
  john logged in: 1                    VM1339:2
<· undefined
> login('john');
  john logged in: 2                    VM1339:2
<· undefined
> login('mary');
  mary logged in: 1                    VM1339:2
<· undefined
> login('john');
  john logged in: 3                    VM1339:2
<· undefined
```

See Counting Statement Executions for more examples.

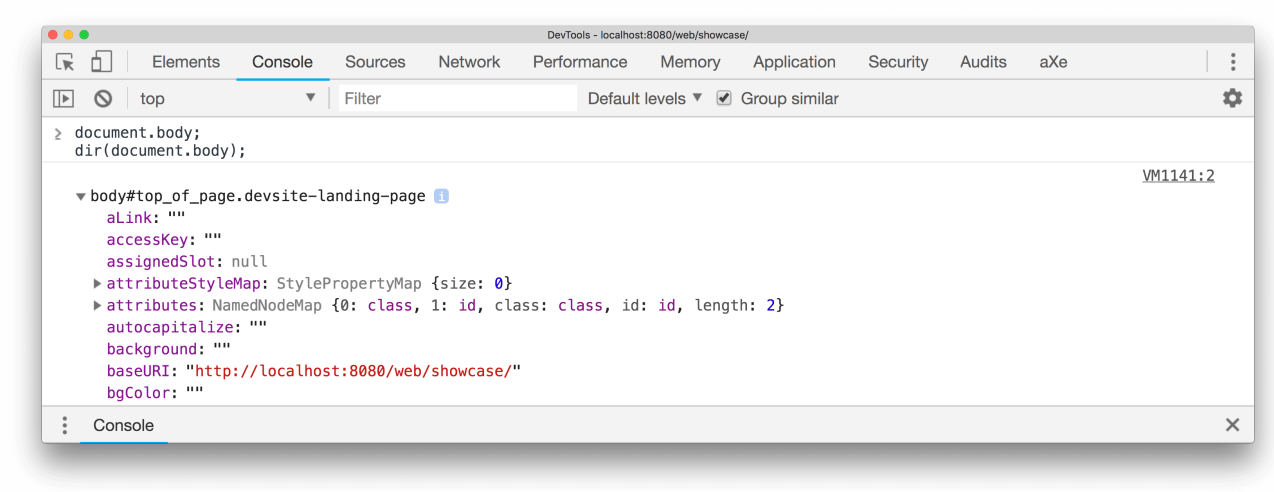
console.debug(object [, object, ...])

Identical to **`console.log()`**.

console.dir(object)

Prints a JavaScript representation of the specified object. If the object being logged is an HTML element, then the properties of its DOM representation are printed, as shown below:

```
console.dir(document.body);
```



Learn about the functionally equivalent object formatter (`%O`) and more in String substitution and formatting.

console.dirxml(object)

Prints an XML representation of the descendant elements of **`object`** if possible, or the JavaScript representation if not. Calling **`console.dirxml()`** on HTML and XML elements is equivalent to calling **`console.log()`**.

```
console.dirxml(document);
```

```
> var obj = {"name":"obj"};
<· undefined
> console.dirxml(obj);
  Object {name: "obj"}                                          VM521:1
<· undefined
> console.dirxml(document);
                                                                VM522:1
  ▼#document
     <!DOCTYPE html>
     <html i18n-values="dir:textdirection;
                        hascustombackground:hasCustomBackground;
                        bookmarkbarattached:bookmarkbarattached;
                        lang:language" dir="ltr" hascustombackground=
   "false" bookmarkbarattached="false" lang="en" i18n-processed>
      ▶<head>…</head>
      ▶<body>…</body>
     </html>
<· undefined
> // equivalent to console.dirxml above
  console.log(document);
                                                                VM683:2
  ▼#document
     <!DOCTYPE html>
     <html i18n-values="dir:textdirection;
                        hascustombackground:hasCustomBackground;
                        bookmarkbarattached:bookmarkbarattached;
                        lang:language" dir="ltr" hascustombackground=
   "false" bookmarkbarattached="false" lang="en" i18n-processed>
      ▶<head>…</head>
      ▶<body>…</body>
     </html>
<· undefined
```

console.error(object [, object, ...])

Prints a message similar to `console.log()`, styles the message like an error, and includes a stack trace from where the method was called.

```
console.error('error: name is undefined');
```

```
> function logName(obj) {
    if (obj.name === undefined) {
      console.error('name is undefined');
    }
  }
<· undefined
> logName({});
✕ ▼ name is undefined                                  VM2249:3
      logName              @ VM2249:3
      (anonymous function) @ VM2250:1
<· undefined
```

console.group(object[, object, ...])

Starts a new logging group with an optional title. All console output that occurs after **`console.group()`** and before **`console.groupEnd()`** is visually grouped together.

```
function name(obj) {
  console.group('name');
  console.log('first: ', obj.first);
  console.log('middle: ', obj.middle);
  console.log('last: ', obj.last);
  console.groupEnd();
}

name({"first":"Wile","middle":"E","last":"Coyote"});
```

```
▼ name                                  VM131:2
    first:  Wile                        VM131:3
    middle:  E                          VM131:4
    last:  Coyote                       VM131:5
```

You can also nest groups:

```
function name(obj) {
  console.group('name');
  console.log('first: ', obj.first);
  console.log('middle: ', obj.middle);
  console.log('last: ', obj.last);
  console.groupEnd();
```

```
}

function doStuff() {
  console.group('doStuff()');
  name({"first":"Wile","middle":"E","last":"coyote"});
  console.groupEnd();
}

doStuff();
```

```
▼ doStuff()                                   VM178:10
  ▼ name                                      VM178:2
      first:  Wile                            VM178:3
      middle:  E                              VM178:4
      last:  coyote                           VM178:5
```

console.groupCollapsed(object[, object, ...])

Creates a new logging group that is initially collapsed instead of open.

```
console.groupCollapsed('status');
console.log("peekaboo, you can't see me");
console.groupEnd();
```

console.groupEnd()

Closes a logging group. See `console.group` for an example.

console.info(object [, object, ...])

Prints a message like `console.log()` but also shows an icon (blue circle with white "i") next to the output.

console.log(object [, object, ...])

Displays a message in the console. Pass one or more objects to this method. Each object is evaluated and concatenated into a space-delimited string.

```
console.log('Hello, Logs!');
```

Format specifiers

The first object you pass can contain one or more **format specifiers**. A format specifier is composed of the percent sign (%) followed by a letter that indicates the formatting to apply.

Related Guides:

- Organizing Console Output

console.profile([label])

Starts a JavaScript CPU profile with an optional label. To complete the profile, call **`console.profileEnd()`**. Each profile is added to the **Profiles** panel.

```
function processPixels() {
  console.profile("processPixels()");
  // later, after processing pixels
  console.profileEnd();
}
```

console.profileEnd()

Stops the current JavaScript CPU profiling session if one is in progress and prints the report to the **Profiles** panel.

See **`console.profile()`** for an example.

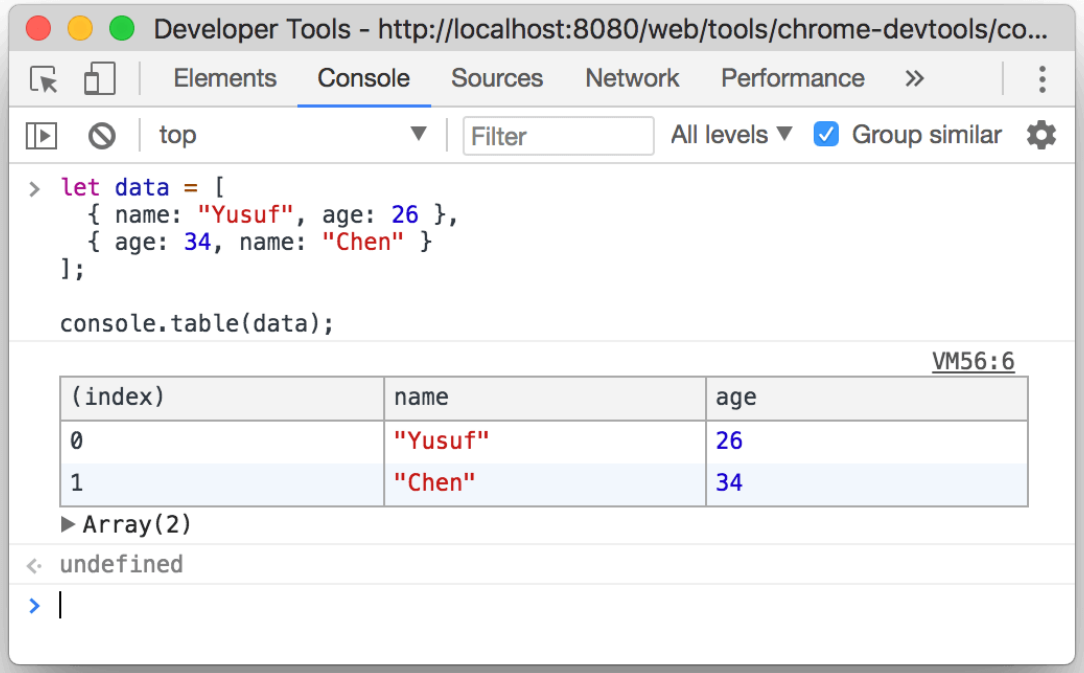
console.table(array)

Logs an array of objects as a table.

```
let data = [
  { name: "Yusuf", age: 26 },
```

```
  { age: 34, name: "Chen" }
];

console.table(data);
```



console.time([label])

Starts a new timer. Call **console.timeEnd()** to stop the timer and print the elapsed time to the Console.

```
console.time();
var arr = new Array(10000);
for (var i = 0; i < arr.length; i++) {
  arr[i] = new Object();
}
console.timeEnd();
// default: 3.696044921875ms
```

Pass an optional label to change the output text that precedes the elapsed time. Call **console.timeEnd()** with the same label to stop the timer.

```
console.time('total');
var arr = new Array(10000);
for (var i = 0; i < arr.length; i++) {
  arr[i] = new Object();
}
console.timeEnd('total');
// total: 3.696044921875ms
```

Use labels to run multiple timers at the same time.

```
console.time('total');
console.time('init arr');
var arr = new Array(10000);
console.timeEnd('init arr');
for (var i = 0; i < arr.length; i++) {
  arr[i] = new Object();
}
console.timeEnd('total');
// init arr: 0.0546875ms
// total: 2.5419921875ms
```

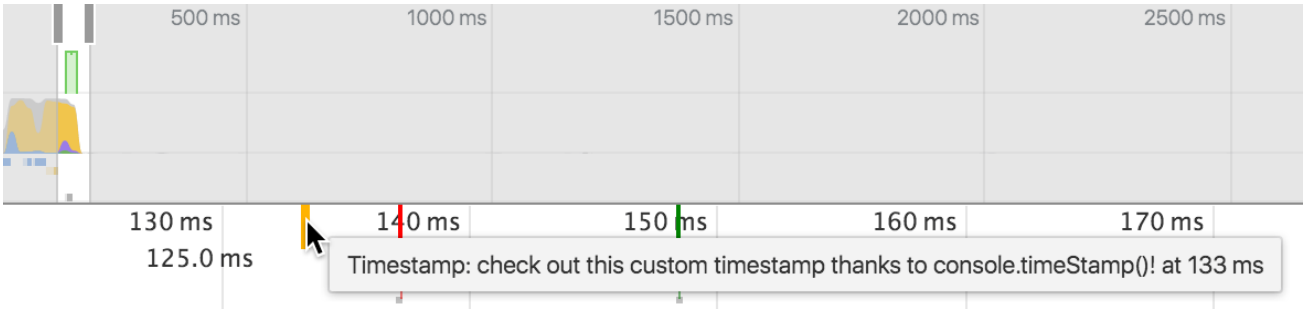
console.timeEnd([label])

Stops a timer. See `console.time()` for examples.

console.timeStamp([label])

Adds an event to the **Timeline** during a recording session.

```
console.timeStamp('check out this custom timestamp thanks to console.timeSt
```



Related Guides:

- Using the Timeline Tool

console.trace(object)

Prints a stack trace from the point where the method was called.

```
console.trace();
```

```
> function add(num) {
    if (num > 0) {
      // you can pass labels and objects to trace, too
      console.trace('recursion is fun:', num);
      return num + add(num - 1);
    } else {
      return 0;
    }
  }
<· undefined
> add(3);
  ▼ recursion is fun: 3                                  VM771:4
      add                    @ VM771:4
      (anonymous function) @ VM790:1
  ▼ recursion is fun: 2                                  VM771:4
      add                    @ VM771:4
      add                    @ VM771:5
      (anonymous function) @ VM790:1
  ▼ recursion is fun: 1                                  VM771:4
      add                    @ VM771:4
      add                    @ VM771:5
      add                    @ VM771:5
      (anonymous function) @ VM790:1
```

console.warn(object [, object, ...])

Prints a message like `console.log()`, but also displays a yellow warning icon next to the logged message.

```
console.warn('user limit reached!');
```

```
> console.warn('user limit reached!');
⚠ user limit reached!                                  VM1013:1
```


Last updated July 2, 2018.