

# JavaScript

---

Here we learn JavaScript, starting from scratch and go on to advanced concepts like OOP.  
We concentrate on the language itself here.

- An introduction
- JavaScript Fundamentals
- Code quality
- Objects: the basics
- Data types
- Advanced working with functions
- Object properties configuration
- Prototypes, inheritance
- Classes
- Error handling
- Promises, async/await
- Generators, advanced iteration
- Modules
- Miscellaneous

## What is JavaScript?

*JavaScript* was initially created to “make web pages alive”.

The programs in this language are called *scripts*. They can be written right in a web page’s HTML and run automatically as the page loads.

Scripts are provided and executed as plain text. They don’t need special preparation or compilation to run.

Today, JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called the JavaScript engine.

The browser has an embedded engine sometimes called a “JavaScript virtual machine”.

Different engines have different “codenames”. For example:

- V8 – in Chrome, Opera and Edge.
- SpiderMonkey – in Firefox.
- ...There are other codenames like “Chakra” for IE, “JavaScriptCore”, “Nitro” and “SquirrelFish” for Safari, etc.

## What can in-browser JavaScript do?

Modern JavaScript is a “safe” programming language. It does not provide low-level access to memory or the CPU, because it was initially created for browsers which do not require it.

JavaScript’s capabilities greatly depend on the environment it’s running in. For instance, Node.js supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.

In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the webserver.

For instance, in-browser JavaScript is able to:

- Add new HTML to the page, change the existing content, modify styles.
- React to user actions, run on mouse clicks, pointer movements, key presses.
- Send requests over the network to remote servers, download and upload files (so-called AJAX and COMET technologies).
- Get and set cookies, ask questions to the visitor, show messages.
- Remember the data on the client-side (“local storage”).

## What CAN'T in-browser JavaScript do?

JavaScript's abilities in the browser are limited to protect the user's safety. The aim is to prevent an evil webpage from accessing private information or harming the user's data.

Examples of such restrictions include:

JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS functions.

Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like “dropping” a file into a browser window or selecting it via an `<input>` tag.

There are ways to interact with the camera/microphone and other devices, but they require a user's explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the any third party.

## What CAN'T in-browser JavaScript do? Cont...

Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other page if they come from different sites (from a different domain, protocol or port).

This is called the “Same Origin Policy”. To work around that, both pages must agree for data exchange and must contain special JavaScript code that handles it. We'll cover that in the tutorial.

This limitation is, again, for the user's safety. A page from `http://anysite.com` which a user has opened must not be able to access another browser tab with the URL `http://gmail.com`, for example, and steal information from there.

JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that's a safety limitation.

Such limitations do not exist if JavaScript is used outside of the browser, for example on a server. Modern browsers also allow plugins/extensions which may ask for extended permissions.

## What makes JavaScript unique?

There are at least three great things about JavaScript:

- Full integration with HTML/CSS.
- Simple things are done simply.
- Supported by all major browsers and enabled by default.

JavaScript is the only browser technology that combines these three things.

That's what makes JavaScript unique. That's why it's the most widespread tool for creating browser interfaces.

That said, JavaScript can be used to create servers, mobile applications, etc.

## Summary

JavaScript was initially created as a browser-only language, but it is now used in many other environments as well.

Today, JavaScript has a unique position as the most widely-adopted browser language, fully integrated with HTML/CSS.

There are many languages that get “transpiled” to JavaScript and provide certain features. It is recommended to take a look at them, at least briefly, after mastering JavaScript.



## Developer console

Code is prone to errors. You will quite likely make errors... Oh, what am I talking about? You are *absolutely* going to make errors, at least if you're a human, not a robot.

But in the browser, users don't see errors by default. So, if something goes wrong in the script, we won't see what's broken and can't fix it.

To see errors and get a lot of other useful information about scripts, “developer tools” have been embedded in browsers.

Most developers lean towards Chrome or Firefox for development because those browsers have the best developer tools. Other browsers also provide developer tools, sometimes with special features, but are usually playing “catch-up” to Chrome or Firefox. So most developers have a “favorite” browser and switch to others if a problem is browser-specific.

Developer tools are potent; they have many features. To start, we'll learn how to open them, look at errors, and run JavaScript commands.

## Google Chrome

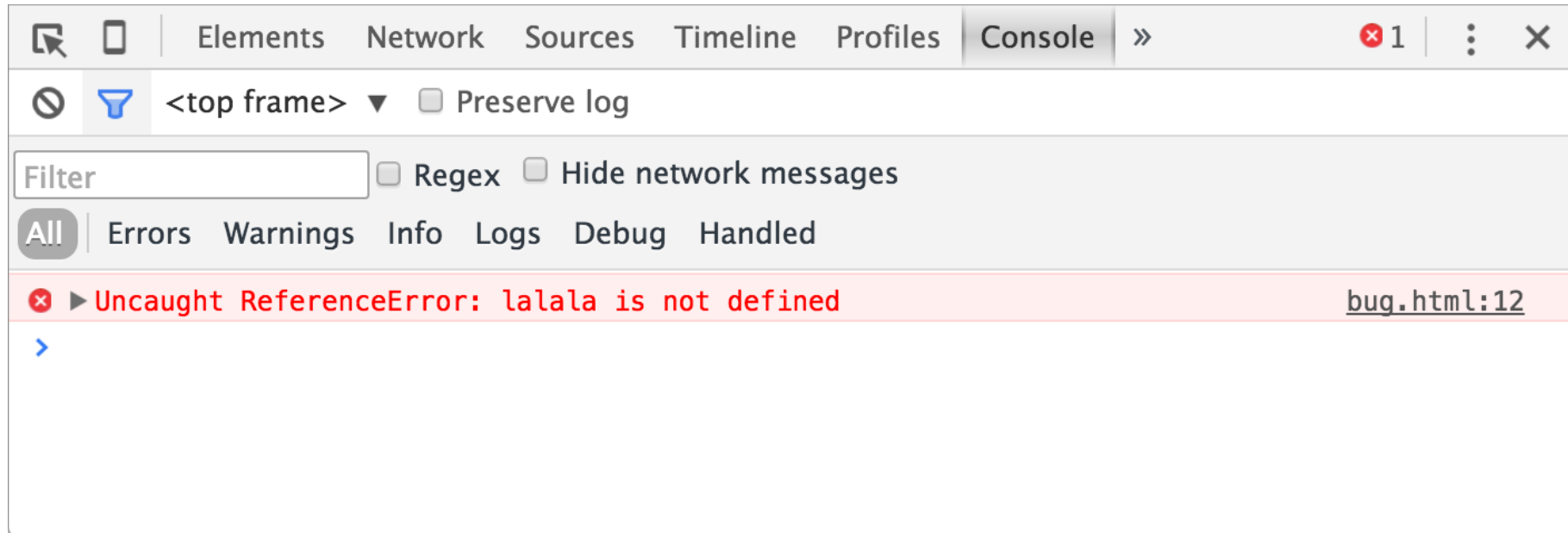
Open the page bug.html.

There's an error in the JavaScript code on it. It's hidden from a regular visitor's eyes, so let's open developer tools to see it.

Press F12 or, if you're on Mac, then Cmd+Opt+J.

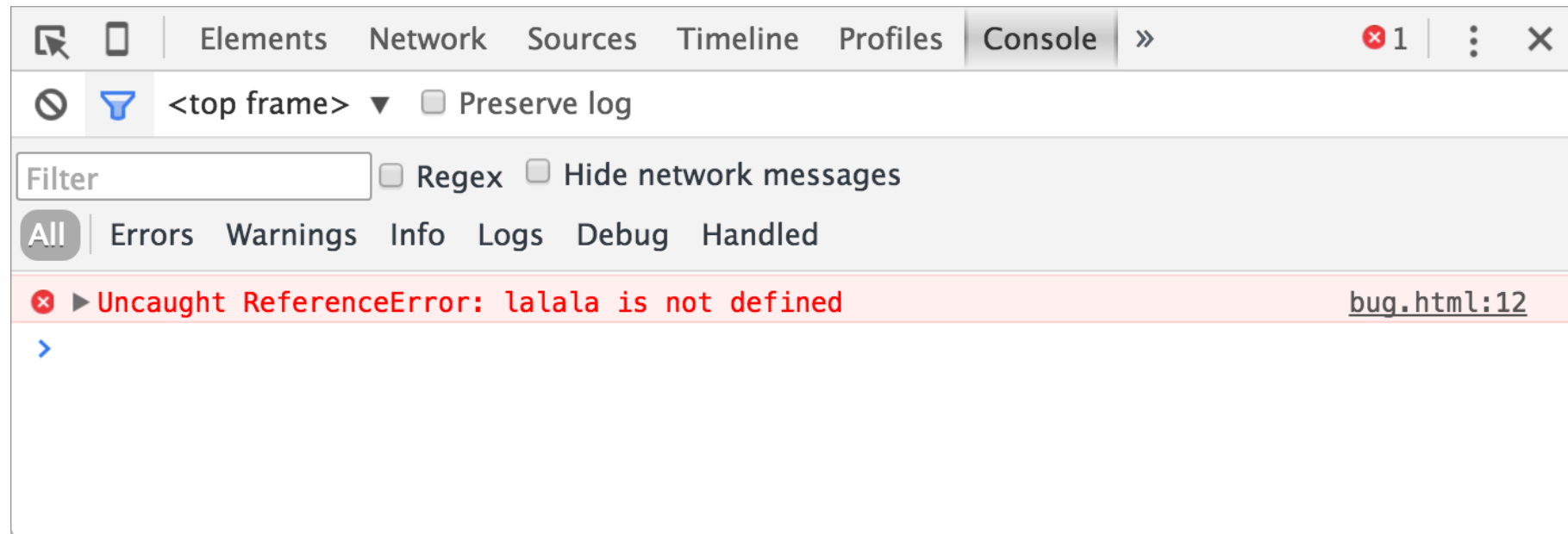
The developer tools will open on the Console tab by default.

It looks somewhat like this:



The exact look of developer tools depends on your version of Chrome. It changes from time to time but should be similar.

- Here we can see the red-colored error message. In this case, the script contains an unknown “lalala” command.
- On the right, there is a clickable link to the source `bug.html:12` with the line number where the error has occurred.



## **Firefox, Edge, and others**

Most other browsers use F12 to open developer tools.

The look & feel of them is quite similar. Once you know how to use one of these tools (you can start with Chrome), you can easily switch to another.

## **Safari**

Safari (Mac browser, not supported by Windows/Linux) is a little bit special here. We need to enable the “Develop menu” first.

Open Preferences and go to the “Advanced” pane. There’s a checkbox at the bottom:

