The Pragmatic Programmers

Debugging TypeScript Applications

Build Web Apps That Don't Break

BETA

Andrey Ozornin

edited by Kelly Talbot

This extract shows the online version of this title, and may contain features (such as hyperlinks and colors) that are not available in the print version.

For more information, or to purchase a paperback or ebook copy, please visit https://www.pragprog.com.

CHAPTER 1

Printing and Interpreting Errors

If you see an error, you are blessed. A good error message tells you what's gone wrong and helps you find where the problem is. A good error message has references to the line of code where it occurred, a call stack, and context: call parameters, OS and browser versions, and so on. Even if your error is not exceptional and misses details, it still will point you toward a starting point, so you can figure out the rest.

In this chapter, we'll get you set up, and then we'll learn to interpret JavaScript errors and investigate them using an interactive debugger. We'll explore different error types.

To follow along, you need to know how to open the command line, how to browse developer tools, and how to switch between the JavaScript console and interactive debugger.

Setting Up for the Book

Chapter 3, Prioritizing Bugs, on page ? and Chapter 5, Designing Software That Doesn't Break, on page ? give foundational information and don't require a computer. But to get the most from the rest of this book, you should follow along with the exercises and install all the following software and tools, sorted from strictly necessary to optional:

- Chrome or another Chromium-based browser (Brave, Edge, Opera, Vivaldi). They all share the same platform: V8 as a Javascript engine and Blink as a rendering engine.
- Visual Studio Code: a free and powerful open-source text editor by Microsoft with plenty of useful extensions
- Node.js, JavaScript runtime, and npm

- Modern browsers based on engines other than Blink and V8:
  - Firefox, based on Gecko (rendering) and SpiderMonkey (JavaScript)
  - Safari, or Orion based on WebKit

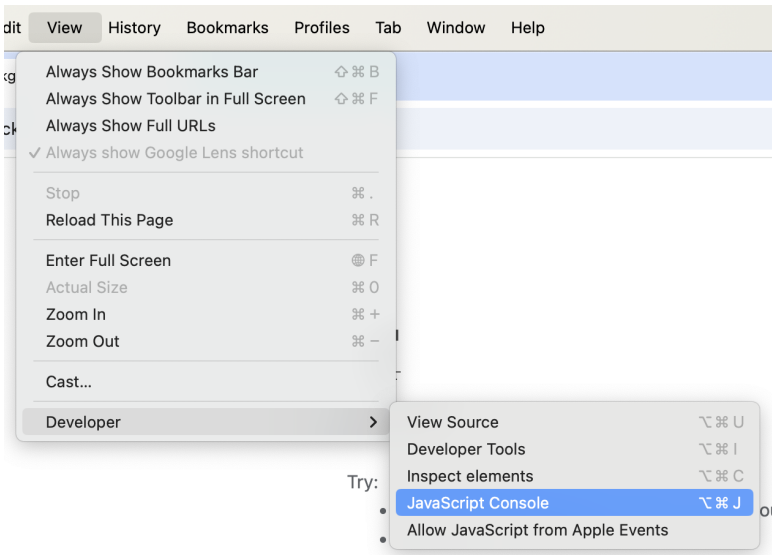
Mac, Linux, or Windows

You shouldn't have any trouble following this book using your platform of choice with one notable exception — hotkeys. The key combinations in this book will be provided for the Mac. Most of the time, if you're using Linux or Windows, it's just a matter of pressing Ctrl where you read ⌘ and Alt where you read ⌥, but sometimes keyboard patterns have bigger differences, and I'll do my best to identify them whenever that turns out to be the case.

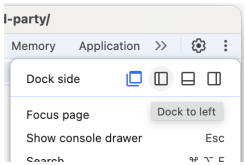
Chrome Developer Tools

We'll be using Google Chrome in this book, but feel free to use the browser of your choice. The developer tools will be mostly the same.

To open the JavaScript Console in Chromium, press ⌥-⌘-J or choose View → Developer → JavaScript Console in the top menu.

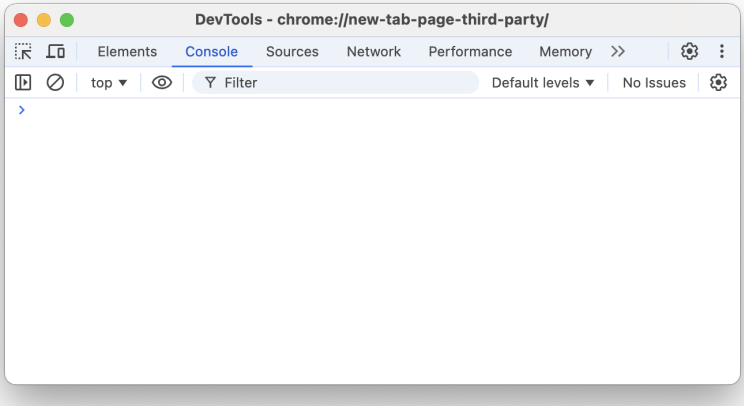


By default, Chrome developer tools open at the bottom of the screen. You can easily dock it to the right or left of the browser window, or even undock it to a separate window (which is especially convenient if you have a two-monitor setup).



Changing the developer tools' size and position will adaptively change the position and visibility of UI panels, so your screen won't look the same as this book's images, but the difference is merely cosmetic.

On the top of the developer tools, you'll find tabs.



You can go directly to the element inspector by pressing ⌥-⌘-C and opening wherever the developer tools panel was last open by pressing ⌥-⌘-I.

Installing VS Code and Extensions

The IDE used in this book is Visual Studio Code, which is open-source and developed and maintained primarily by Microsoft. It can work with git and TypeScript out of the box and lets you install plenty of third-party plug-ins from the marketplace: from C++ static analysis to project management tool integrations and AI-powered code companions. Note that the VS Code marketplace platform is controlled by Microsoft and the binary, unlike the source code, has Microsoft telemetry built in.

We will talk about setting up the developer infrastructure and productivity tools in Chapter 5, Designing Software That Doesn't Break, on page ?, but for now I'll list the plugins I recommend you to install and at least read marketplace descriptions of:

- ESLint
- Prettier
- Jest

If you don't like Visual Studio Code, you can use JetBrains WebStorm, which is free for non-commercial use and also has everything we need for this book out of the box.

Git and Repositories

This book's code samples are available on the pragprog website[1] and GitHub[2] for downloading. This book assumes that you have git. You can follow the instructions on the git website[3] to install it. Technically, you can work around git by downloading .zip archives from GitHub instead of cloning repositories, but git is the industry standard and it is good. Get it, really.

To clone a repository (download project files and the full history of changes to your local computer), you run `git clone`.

```
git clone https://github.com/oshibka404/xoxo.js.git
```

This command will clone a tic-tac-toe game written in JavaScript without external libraries or frameworks.

Node.js and NPM Packages

There are multiple ways of installing Node.js, depending on your operating system, preferred package manager, and habits. This book uses the one that works on both macOS and Linux. Go to nodejs.org, open the Download page, and enter the following in the terminal.

```
# installs nvm (Node Version Manager)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh | bash

# download and install Node.js (you may need to restart the terminal)
nvm install 24
```

This book uses Node v24.11, which is the most current LTS (long-time support) version. If you have another version of Node.js, you can and should use the node version manager to switch between versions. This book uses `nvm`, the most used of the alternatives at the time of this writing.

```
# to use the version we use in the book
nvm use 24
```

1. https://pragprog.com/titles/aodjs/
2. https://github.com
3. https://git-scm.com/downloads

If you have switched to a different version, you can go back to the latest LTS version by running:

```
nvm use --lts
```

This book uses node.js tools, such as package manager Yarn, test runner Jest, and static analyzer ESLint. Frameworks and libraries such as React, Lodash, and Astro will also be used. To install these packages from NPM, use this:

```
# Install Jest and save to developer dependencies
npm install jest --save-dev

# install React and save it to project dependencies
npm install react

# install a specific version of lodash (in this example, v3)
npm install lodash@3

# Install yarn and make it available globally
npm install --global yarn

# After installing a package globally, you can call it in command line directly:
yarn
```

Alternatives

This book mostly uses tools such as Node, NVM, NPM, ESLint, and Git, which are mature and widespread enough to not cease to exist in next couple of years. Still, alternatives exist that are in general faster, provide a better developer experience, or are based on a different paradigm. All of them are worth trying or reading about.

Command-line tools and npm packages include the following:

- Node.js: `bun`, `deno`
- NVM: `fnm`
- NPM: `yarn`, `pnpm` or, again, `bun`
- ESLint: Biome
- Git: `jj`
- Jest: vitest

Web tools and desktop software include the following:

- GitHub: CodeBerg
- VS Code: WebStorm

Checklist

Once you have the following installed, you're good to go:

- Chrome or another Chromium-based browser
- Visual Studio Code
- Node.js and npm
- git

If you choose to use alternative software while following this book, I assume you know what you're doing or are willing to do some research on your own. Now that everything's ready, you can start having some fun.

Getting Started with Debugging

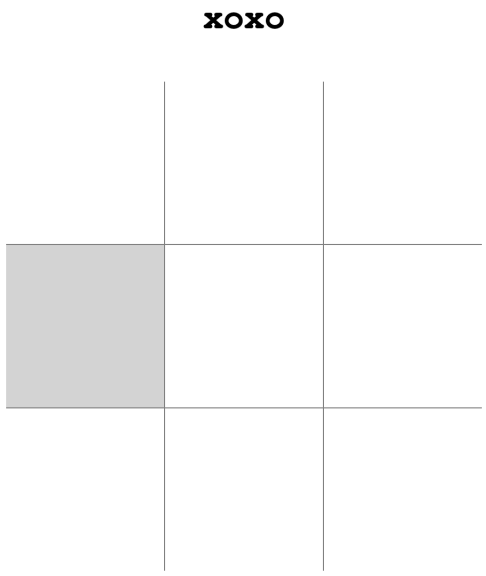
We'll start with debugging a production website rather than a locally running copy where you have full edit access to all the files. In the usual debugging process, this is the first step after a bug is reported by an external user: you open the same URL as they did and try to get the same results as they did. After you have managed to reproduce the problem, you start exploring to find the boundaries of the bug: Does the same thing happen on other pages? Does it happen on a local copy? On refresh? In a different browser? On a different OS? With another screen resolution? With a different zoom level?

There are many nuances to a good bug reproduction, as well as to a good bug report, and we'll talk about them in-depth in Chapter 4, Finding the Root Cause, on page ?. But first things first. We'll start investigating an easily reproduced error on a remote site using only a browser.

Fixing Common Errors

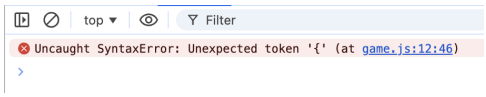
Open the tic-tac-toe game page[4] in a browser. The page opens, and the playing field is visible, but nothing happens when you click. Let's fix it.

4. https://oshibka404.github.io/xoxo.js/

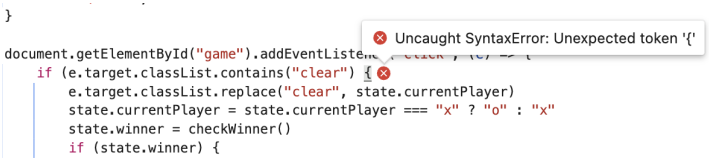


Fixing a SyntaxError with Code Overrides

Open the JavaScript console and see `Uncaught SyntaxError: Unexpected token '{'` written in red. `SyntaxError`, of all the error types, is the easiest to deal with. Usually, it means that you need to add or remove one character. Right next to it, you will see a clickable link to the line and column where JavaScript parsing failed:



Clicking on it will show you a code listing with the suspected line highlighted:



If the syntax mistake is not immediately clear from there, the typical sequence of actions to find it would be:

1. Count the opening and closing brackets of each kind around the error origin. (For example, { ([ ] } is missing a closing) bracket.)
2. Check that the order of closing brackets matches the order of the opening ones. (For example, ({ }) is valid, but ({) } is not.)

3. Check commas, colons, and semicolons. (For example, the declaration `function foo(a b)` misses a `,` between its arguments. It should be `function foo(a, b)`.)
4. Google the error text, having removed variable names and values from it.

In our case, the `if` condition is missing the closing parenthesis `)`. Note that `Unexpected token "{"` does not always mean that the `{` should not be there. Often it means that something that the code parser failed to find there should exist before it.

SyntaxError in Other Browsers

The JavaScript engine of WebKit-based browsers like Safari and Orion has an even more detailed description of `SyntaxError` than Chrome's V8. It not only tells you what unexpected character breaks the code parsing, but it also points out which character is expected, making the fix even easier for you. In our example, it would leave less ambiguity and print:

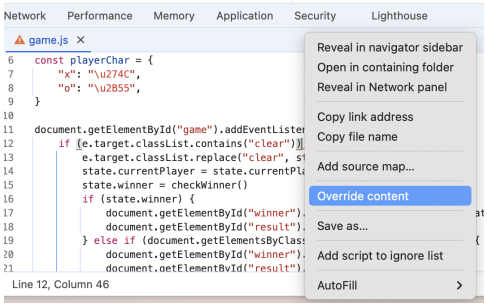
```
SyntaxError: Unexpected token '{'. Expected ')' to end an 'if' condition.
```

SpiderMonkey, the JavaScript engine of Firefox, provides a handy link called "[Learn More]" that leads to the MDN page with a description and common causes of this specific error.

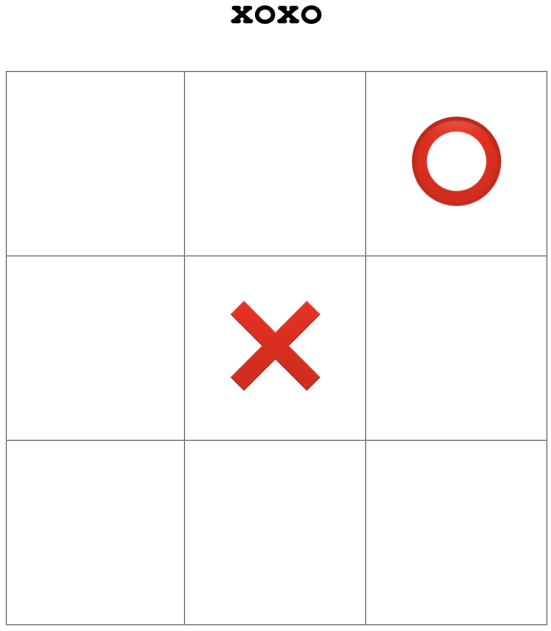
If we were debugging a page in local files, we would open the game.js file in a code editor, add the missing `)`, save the file, and refresh the page in the browser. But we are debugging a remote webpage with its source code on a server where we cannot easily upload a new version of a file and check whether it works. We will have to use local files on remote sites, and Chromium-based browsers allow us to do that using local overrides.

Local Overrides

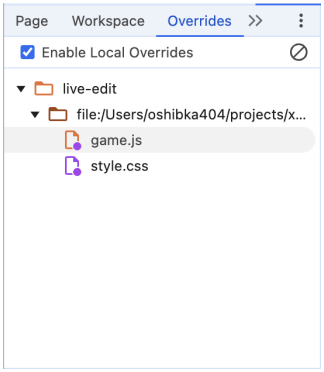
In the Sources tab, add the missing parenthesis, then right-click the code editor and select "Override content."



The browser will ask you to choose a folder to store the override files in (which can be any folder) and permit it to access the folder. Now refresh the page, click on any of the cells, and enjoy that the error is gone.



From now on, you can access your local overrides in the left sidebar of the Sources panel, in the “Overrides” tab. Going forward, all the style changes you make on the pages of this website will also be preserved between sessions, unless you explicitly disable this functionality or uncheck the “Enable Local Overrides” checkbox.



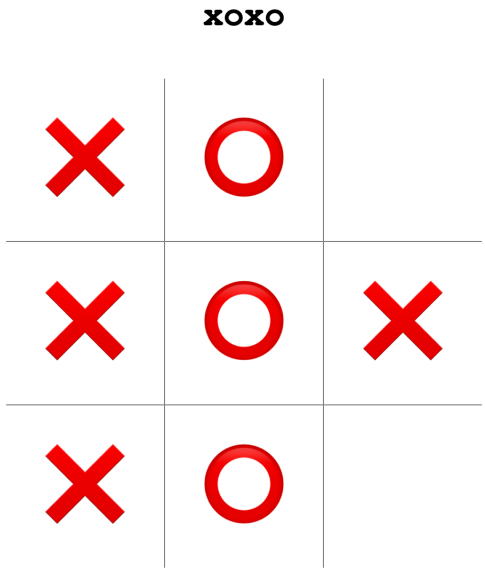
SyntaxErrors are not only the easiest errors to fix but also the easiest to prevent. Visual Studio Code, WebStorm, Zed, or any other modern web-oriented text editor will highlight these errors in no time. Sometimes, for various reasons, you might have to edit the code in web editors, using automated scripts, or in the console without the syntax highlighted, so the code editors won't help. If it happens often enough for the problem to be noticeable, you can set up a git hook validating code on commit and set up continuous integration (CI).

Syntax errors will be many times easier to find if you practice consistent formatting, and automation will help too. Use Prettier[5] or Biome[6] to validate or even auto-fix code formatting. It is helpful when you are on your own, and it's essential in a large team. We'll discuss developer infrastructure in more detail in Chapter 5, Designing Software That Doesn't Break, on page ?.

Fixing TypeError with Interactive Debugging

If you are from this planet and have played tic-tac-toe before, you will quickly notice that the game doesn't work. In a real tic-tac-toe game, when one of the players gets three crosses or circles in a row, column, or diagonal, the player wins and the game is supposed to finish. On our page, that doesn't happen, and the losing player can make moves even after being beaten.

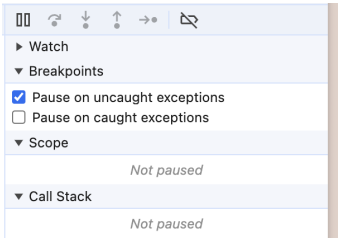
5. https://prettier.io/
6. https://biomejs.dev/



To start investigating why, let's take another look in the JavaScript console. You will find two unique error messages.

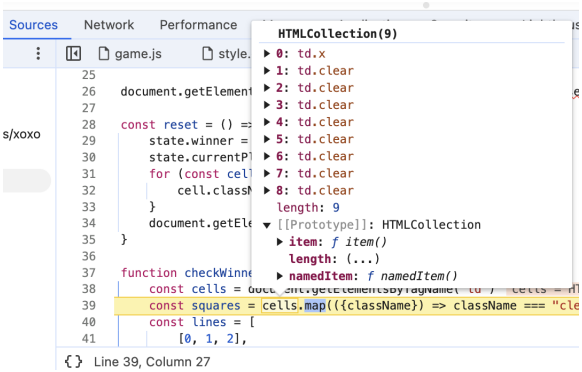
```
▸Uncaught ReferenceError: Cannot access 'reset' before initialization
    at game.js:26:62
▸Uncaught TypeError: document.getElementsByTagName(...).map is not a function
    at checkWinner (game.js:38:55)
    at HTMLTableElement.<anonymous> (game.js:15:24)
```

Let's investigate the error using interactive debugging, a powerful technique that we'll be using a lot in this book and will dive deep into in Chapter 6, Following the Data Flow, on page ?. With the page open, go to the Sources tab and enable the "Pause on uncaught exceptions" checkmark.



Click on a game cell to make a move. The execution pauses on the line where the error is happening, and the code listing automatically scrolls to that line. Now you have access to all the values of variables and properties that are in the visible area of the line and column where the exception is thrown. Hover over the cells variable. In the popup, you see that cells is an instance of HTMLCol-

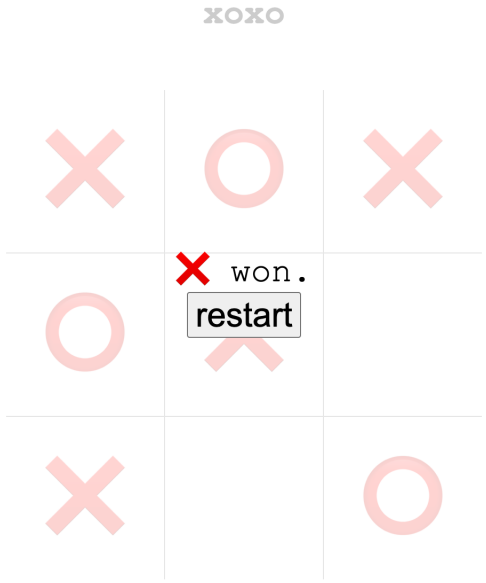
lection and has indexed properties that we are unsuccessfully trying to iterate through using map.



If you expand prototype to see what methods of HTMLCollection are available, you'll see that there is indeed no method called map. There are multiple options to actually map a collection of cells to an array of "x", "o" and "", depending on the cell class. For example, you can rewrite the method to use for ... of instead of map to iterate across all items or transform the HTMLCollection into an array and keep the rest of the logic as it is. Let's do the latter and replace cells.map with [...cells].map. HTMLCollection allows us to do this because it is an Iterable, which we can tell because it has a [[Symbol.Iterator]] in the prototype.

```
[...cells].map(({className}) => className === "clear" ? "" : className)
```

Edit and press Cmd+S to save the override, then refresh the page. Verify that the proper winner check works now.



On the Importance of Naming

Even without interactive debugging, it's often possible to figure out where to start investigating just by carefully reading the errors. In our case we have two errors: ReferenceError mentions something about reset which might be relevant somehow, but the TypeError has a checkWinner function in its call stack, and we know that there's a problem somewhere around checking whether there is a winner. If the function were named not checkWinner but doThings, we wouldn't have such a privilege.

TypeErrors can sometimes be tricky to track, but you can prevent them with technical measures, the best kind of prevention. Switch to TypeScript and don't circumvent its restrictions; they are there for a reason. We'll talk more about using and configuring TypeScript for your projects in Chapter 5, Designing Software That Doesn't Break, on page ?.

Fixing ReferenceError by Moving Code Blocks

The Restart button doesn't work. If you follow the same steps as we did earlier (either click the link in the JavaScript console or pause on exception and refresh the page), you will see that reset is not defined yet when it's being called.

There are two schools of thought regarding method ordering. One encourages the main procedure of a module, or an entry point, to be placed on top with

all of its subroutines declared below, in order of appearance. The other perspective prescribes every function in the module to be declared before it is referenced. Either approach is possible for this case. Whichever approach you choose, be consistent (at least within a single project), especially if you are not working alone. Following a rule consistently often matters more than whatever the rule actually is.

Your options to fix the error are:

1. Change the function expression to the function declaration: replace `const reset = () => {` with `function reset() {`. Function declarations hoist to the top of their scope and can be safely referenced from there.
2. Take the function as it is and move it above its usage.

Pick either option, save the file by pressing Cmd+S, and refresh the page.

Both actions can have side effects that are not relevant in this scenario but in many cases will be. Arrow functions, declared with the syntax `() => {}`, preserve the context where they are declared, whereas functions using the keyword `function` will use the context of the call. Be aware that if you change a function with `this` used in its body from an arrow function to a class method or vice versa, you are likely to break something.

Another important consideration is that execution order matters a lot, and if to fix a `ReferenceError` you do something more significant than swapping a function and a reference to it, double-check that the other code in the module doesn't rely on it being where it was.

That being said, congratulations! The tic-tac-toe game works. If you cloned the repository, you can copy the fixed file contents to the text editor, open `index.html` in a browser, and play around with the code.

Looking Up Errors

There are many other error types: `RangeError`, `URIError`, `AggregateError`, and dozens of different `TypeErrors` and `SyntaxErrors`. They all have lots in common within each type, and the techniques we learned in this chapter will help you to deal with any of them.

When you google the error, remember to delete your data and names of your variables from the request. Having them will only worsen the search result quality.

AI tools can be useful in interpreting and mitigating errors, but the burden of validation and the final judgment stays with you, the human. It is perfectly fine to apply a fix suggested by AI, but make sure you understand how the

fix works well; otherwise, you can't be sure that it is a good fix. "If you didn't fix it, it ain't fixed." What's more, if you end up breaking it worse, it needs to be fixed even more.

Mozilla Developer Network has a great JavaScript Error Reference[7] page listing all existing native JavaScript errors, their versions in different families of browsers, common causes, and solutions to them.

Wrapping Up

Now that you've completed this chapter, you should be able to trace the program execution flow by printing console messages without interrupting it. You can pause the program execution, inspect its state, and resume, using the interactive debugger. You are able to interpret JavaScript errors you see in the console.

In the next chapter, we'll use this knowledge of errors and logging to build reliable, easily debuggable software systems.

7. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Errors