The Pragmatic Programmers

Debugging TypeScript Applications

Build Web Apps That Don't Break

BETA

Andrey Ozornin

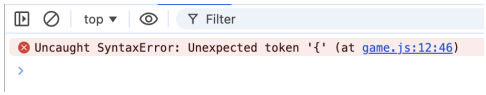
edited by Kelly Talbot

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.

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:

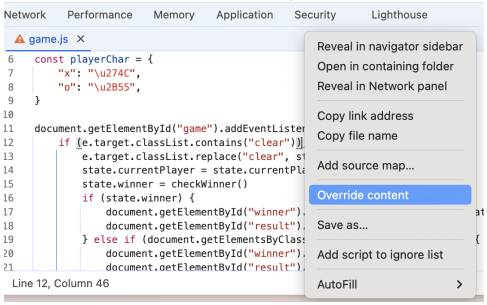
```
SyntaxError: Unexpected token '{'. Expected ')' to end an 'if' condition.
```

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.

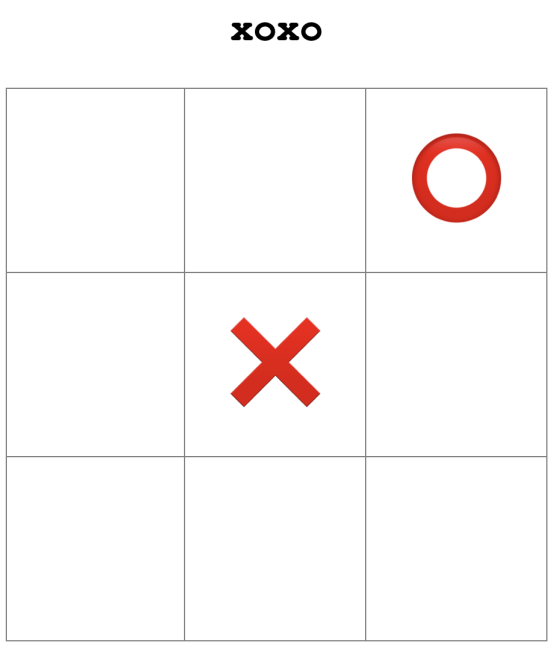
If we were debugging a page in local files, we would open the game.js file in a code editor, add the missing), save the file, and refresh the page in the browser. But we are debugging a remote webpage with its source code on a server where we cannot easily upload a new version of a file and check whether 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

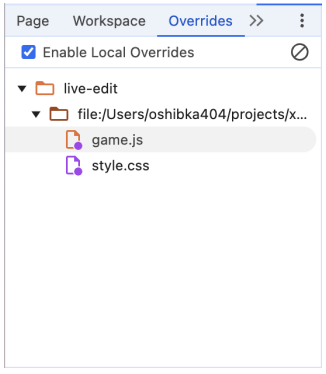
In the Sources tab, add the missing parenthesis, then right-click the code editor and select “Override content.”



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.



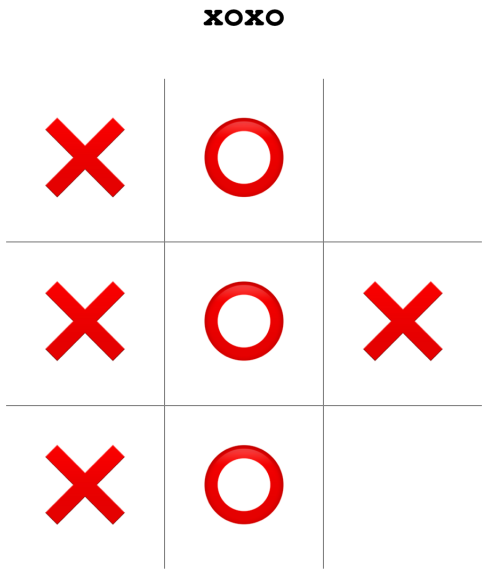
SyntaxErrors are not only the easiest errors to fix but also the easiest to prevent. Visual Studio Code, WebStorm, Zed, or any other modern web-oriented text editor will highlight these errors in no time. Sometimes, for various reasons, you might have to edit the code in web editors, using automated 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.

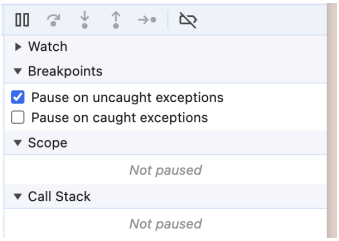


To start investigating why, let's take another look in the JavaScript console. You will find two unique error messages.

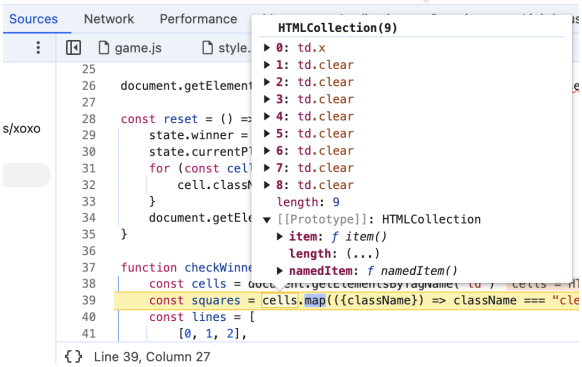
5. https://prettier.io/
6. https://biomejs.dev/

```
▶ 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.



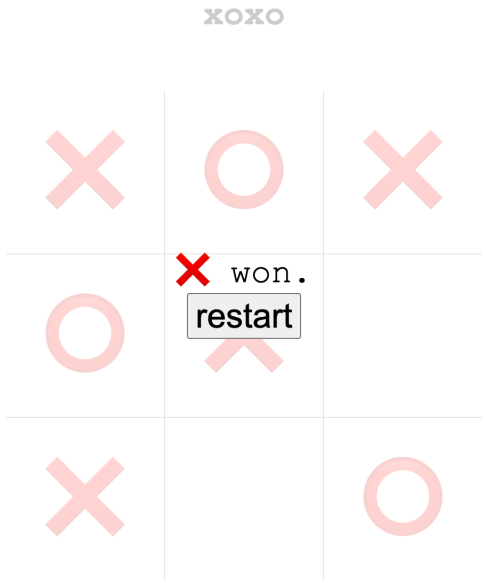
Click on a game cell to make a move. The execution pauses on the line where the error is happening, and the code listing automatically scrolls to that line. Now you have access to all the values of variables and properties that are in the visible area of the line and column where the exception is thrown.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