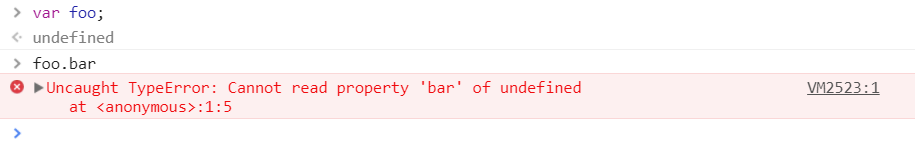
# Find and Fix

# C:\Users\admin\Pictures\Picture1.tif

# Uncaught TypeError: Cannot read property

This one occurs in Chrome when you read a property or call a method on an undefined object. You can test this very easily in the Chrome Developer Console.



This can occur for many reasons, but a common one is improper initialization of state while rendering the UI components. Let’s look at an example of how this can occur in a real-world app. We’ll pick React, but the same principles of improper initialization also apply to Angular, Vue or any other framework.

class Quiz extends Component {

componentWillMount() {

axios.get('/thedata').then(res => {

this.setState({items: res.data});

});

}

render() {

return (

<ul>

{this.state.items.map(item =>

<li key={item.id}>{item.name}</li>

)}

</ul>

);

}

}

There are two important things realize here:

1. A component’s state (e.g. this.state) begins life as undefined.
2. When you fetch data asynchronously, the component will render at least once before the data is loaded – regardless of whether it’s fetched in the constructor, componentWillMount or componentDidMount. When Quiz first renders, this.state.items is undefined. This, in turn, means ItemList gets items as undefined, and *you* get an error – "Uncaught TypeError: Cannot read property ‘map’ of undefined" in the console.

This is easy to fix. The simplest way: Initialize state with reasonable default values in the constructor.

class Quiz extends Component {

// Added this:

constructor(props) {

super(props);

// Assign state itself, and a default value for items

this.state = {

items: []

};

}

componentWillMount() {

axios.get('/thedata').then(res => {

this.setState({items: res.data});

});

}

render() {

return (

<ul>

{this.state.items.map(item =>

<li key={item.id}>{item.name}</li>

)}

</ul>

);

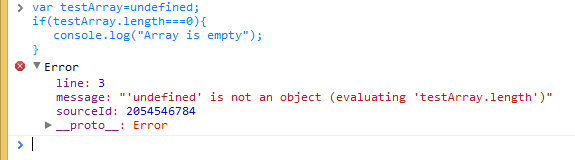
}

}

The exact code in your app might be different, but we hope we’ve given you enough of a clue to either fix or avoid this problem in your app. If not, keep reading because we’ll cover more examples for related errors below.

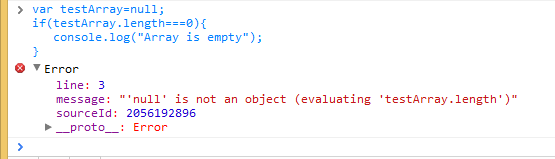
# 2.TypeError: ‘undefined’ is not an object (evaluating

This is an error that occurs in Safari when you read a property or call a method on an undefined object. You can test this very easily in the Safari Developer Console. This is essentially the same as the above error for Chrome, but Safari uses a different error message.

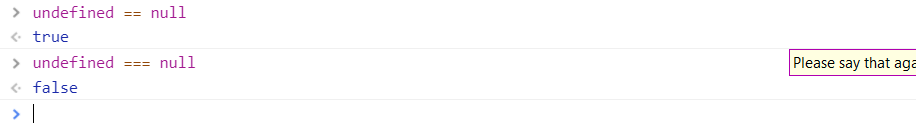


# 3.TypeError: null is not an object (evaluating

This is an error that occurs in Safari when you read a property or call a method on a null object. You can test this very easily in the Safari Developer Console.



Interestingly, in JavaScript, null and undefined are not the same, which is why we see two different error messages. Undefined is usually a variable that has not been assigned, while null means the value is blank. To verify they are not equal, try using the strict equality operator:



One way this error might occur in a real world example is if you try using a DOM element in your JavaScript before the element is loaded. That’s because the DOM API returns null for object references that are blank.

Any JS code that executes and deals with DOM elements should execute after the DOM elements have been created. JS code is interpreted from top to down as laid out in the HTML. So, if there is a tag before the DOM elements, the JS code within script tag will execute as the browser parses the HTML page. You will get this error if the DOM elements have not been created before loading the script.

In this example, we can resolve the issue by adding an event listener that will notify us when the page is ready. Once the addEventListener is fired, the init() method can make use of the DOM elements.

<script>

function init() {

var myButton = document.getElementById("myButton");

var myTextfield = document.getElementById("myTextfield");

myButton.onclick = function() {

var userName = myTextfield.value;

}

}

document.addEventListener('readystatechange', function() {

if (document.readyState === "complete") {

init();

}

});

</script>

<form>

<input type="text" id="myTextfield" placeholder="Type your name" />

<input type="button" id="myButton" value="Go" />

</form>

# 4.(unknown): Script error

The Script error occurs when an uncaught JavaScript error crosses domain boundaries in violation of the cross-origin policy. For example, if you host your JavaScript code on a CDN, any uncaught errors (errors that bubble up to the window.onerror handler, instead of being caught in try-catch) will get reported as simply "Script error" instead of containing useful information. This is a browser security measure intended to prevent passing data across domains that otherwise wouldn’t be allowed to communicate.

To get the real error messages, do the following:

**1. Send the Access-Control-Allow-Origin header**

Setting the Access-Control-Allow-Origin header to \* signifies that the resource can be accessed properly from any domain. You can replace \* with your domain if necessary: for example, Access-Control-Allow-Origin: www.example.com. However, handling multiple domains gets tricky, and may not be worth the effort if you’re using a CDN due to caching issues that may arise. See more [**here**](http://stackoverflow.com/questions/1653308/access-control-allow-origin-multiple-origin-domains).

Here are some examples on how to set this header in various environments:

***Apache***

In the folders where your JavaScript files will be served from, create an .htaccess file with the following contents:

Header add Access-Control-Allow-Origin "\*"

***Nginx***

Add the add\_header directive to the location block that serves your JavaScript files:

location ~ ^/assets/ {

add\_header Access-Control-Allow-Origin \*;

}

**HAProxy**

Add the following to your asset backend where JavaScript files are served from:

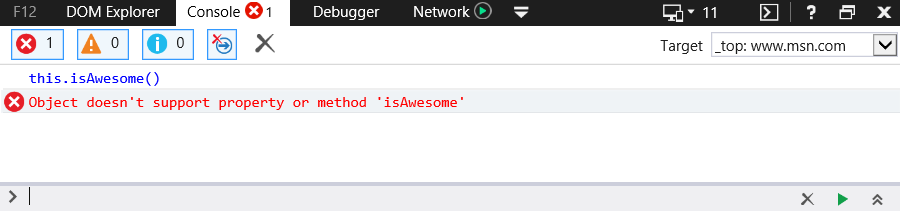
rspadd Access-Control-Allow-Origin:\ \*

**2. Set crossorigin="anonymous" on the script tag**

In your HTML source, for each of the scripts that you’ve set the Access-Control-Allow-Origin header for, set crossorigin="anonymous" on the SCRIPT tag. Make sure you verify that the header is being sent for the script file before adding the crossorigin property on the script tag. In Firefox, if the crossorigin attribute is present but the Access-Control-Allow-Origin header is not, the script won’t be executed.

# 5. TypeError: Object doesn’t support property

This is an error that occurs in IE when you call an undefined method. You can test this in the IE Developer Console.



This is equivalent to the error "TypeError: ‘undefined’ is not a function" in Chrome. Yes, different browsers can have different error messages for the same logical error.

This is a common problem for IE in web applications that employ JavaScript namespacing. When this is the case, the problem 99.9% of the time is IE’s inability to bind methods within the current namespace to the this keyword. For example, if you have the JS namespace Rollbar with the method isAwesome. Normally, if you are within the Rollbar namespace you can invoke the isAwesome method with the following syntax:

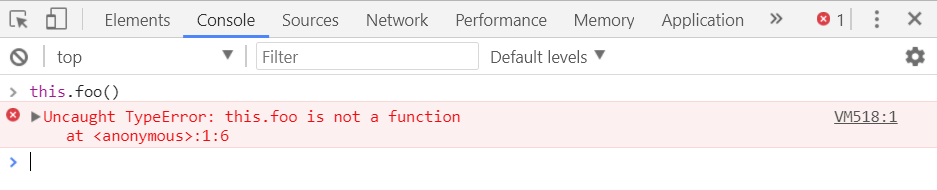
this.isAwesome();

Chrome, Firefox and Opera will happily accept this syntax. IE, on the other hand, will not. Thus, the safest bet when using JS namespacing is to always prefix with the actual namespace.

Rollbar.isAwesome();

# 6. TypeError: ‘undefined’ is not a function

This is an error that occurs in Chrome when you call an undefined function. You can test this in the Chrome Developer Console and Mozilla Firefox Developer Console.



As JavaScript coding techniques and design patterns have become increasingly sophisticated over the years, there’s been a corresponding increase in the proliferation of self-referencing scopes within callbacks and closures, which are a fairly common source of this/that confusion.

Consider this example code snippet:

function clearBoard(){

alert("Cleared");

}

document.addEventListener("click", function(){

this.clearBoard(); // what is “this” ?

});

If you execute the above code and then click on the page, it results in the following error "Uncaught TypeError: this.clearBoard is not a function". The reason is that the anonymous function being executed is in the context of the document, whereas clearBoard is defined on the window.

A traditional, old-browser-compliant solution is to simply save your reference to this in a variable that can then be inherited by the closure. For example:

var self=this; // save reference to 'this', while it's still this!

document.addEventListener("click", function(){

self.clearBoard();

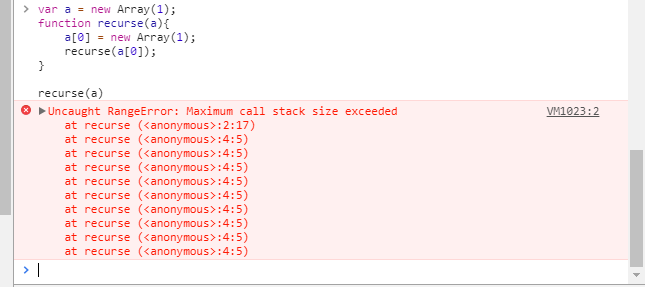
});

Alternatively, in the newer browsers, you can use the bind() method to pass the proper reference:

document.addEventListener("click",this.clearBoard.bind(this));

# 7. Uncaught RangeError

This is an error that occurs in Chrome under a couple of circumstances. One is when you call a recursive function that does not terminate. You can test this in the Chrome Developer Console.



It may also happen if you pass a value to a function that is out of range. Many functions accept only a specific range of numbers for their input values. For example, Number.toExponential(digits) and Number.toFixed(digits) accept digits from 0 to 100, and Number.toPrecision(digits) accepts digits from 1 to 100.

var a = new Array(4294967295); //OK

var b = new Array(-1); //range error

var num = 2.555555;

document.writeln(num.toExponential(4)); //OK

document.writeln(num.toExponential(-2)); //range error!

num = 2.9999;

document.writeln(num.toFixed(2)); //OK

document.writeln(num.toFixed(105)); //range error!

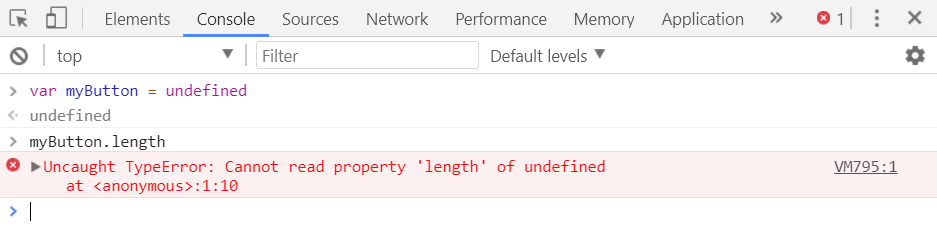
num = 2.3456;

document.writeln(num.toPrecision(1)); //OK

document.writeln(num.toPrecision(0)); //range error!

# 8.TypeError: Cannot read property ‘length’

This is an error that occurs in Chrome because of reading length property for an undefined variable. You can test this in the Chrome Developer Console.



You normally find length defined on an array, but you might run into this error if the array is not initialized or if the variable name is hidden in another context. Let’s understand this error with the following example.

var testArray= ["Test"];

function testFunction(testArray) {

for (var i = 0; i < testArray.length; i++) {

console.log(testArray[i]);

}

}

testFunction();

When you declare a function with parameters, these parameters become local ones. This means that even if you have variables with names testArray, parameters with the same names within a function will still be treated as **local.**

You have two ways to resolve your issue:

1. Remove parameters in the function declaration statement (it turns out you want to access those variables that are declared outside of the function, so you don’t need parameters for your function):

var testArray = ["Test"];

1. /\* Precondition: defined testArray outside of a function \*/
2. function testFunction(/\* No params \*/) {
3. for (var i = 0; i < testArray.length; i++) {
4. console.log(testArray[i]);
5. }
6. }
7. testFunction();

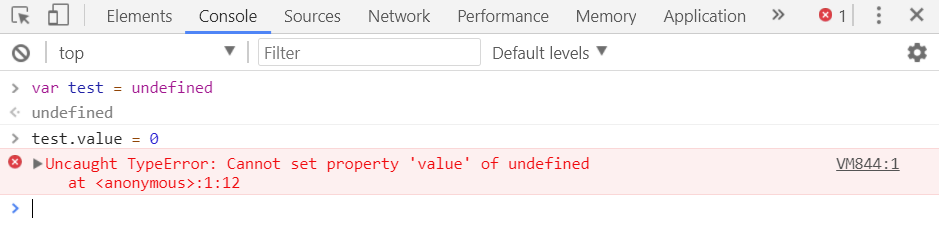
2.Invoke the function passing it the array that we declared:

1. var testArray = ["Test"];
2. var testArray = ["Test"];
3. function testFunction(testArray) {
4. for (var i = 0; i < testArray.length; i++) {
5. console.log(testArray[i]);
6. }
7. }
8. testFunction(testArray);

# 9. Uncaught TypeError: Cannot set property

When we try to access an undefined variable it always returns undefined and we cannot get or set any property of undefined. In that case, an application will throw “Uncaught TypeError cannot set property of undefined.”

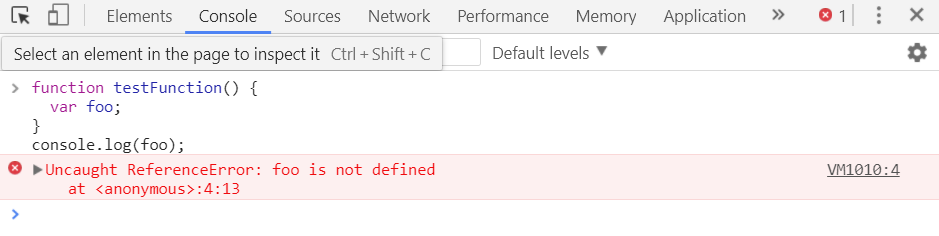
For example, in the Chrome browser:



If the test object does not exist, error will throw “Uncaught TypeError cannot set property of undefined.”

# 10. ReferenceError: event is not defined

This error is thrown when you try to access a variable that is undefined or is outside the current scope. You can test it very easily in Chrome browser.



If you’re getting this error when using the event handling system, make sure you use the event object passed in as a parameter. Older browsers like IE offer a global variable event, and Chrome automatically attaches the event variable to the handler. Firefox will not automatically add it. Libraries like jQuery attempt to normalize this behavior. Nevertheless, it’s best practice to use the one passed into your event handler function.

document.addEventListener("mousemove", function (event) {

console.log(event);

})