# 1. Uncaught TypeError: Cannot Read Property

This is one of the most common JS errors in the list. It occurs when we try to access a property or a method in an undefined object.

**Fix**

Assign a reasonable value to the object during construction or initialization — do not keep it null or undefined.

**2. TypeError: ‘undefined’ Is Not an Object**

The error occurs when we call a property or method on an undefined object.

**Fix**

Assign a reasonable value to the object during construction or initialization — do not keep it undefined.

**3. TypeError: ‘null’ Is Not an Object**

The error occurs when we call a property or method on a nullobject. This is an error that, like the above, occurs only on Safari.

**Fix**

Assign a reasonable value to the object during construction or initialization — do not keep it null.

**4. (unknown): Script error**

The Script error occurs when an uncaught JavaScript error crosses domain boundaries in violation of the cross-origin policy.

**Fix**

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

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

**5 .TypeError: Object Doesn’t Support Property**

This is an error that occurs on IE when we call an undefined method. This is very much equivalent to the error “TypeError: ‘undefined’ is not a function” in Chrome.

**Fix**

This error is a common problem for IE. It is IE’s inability to bind methods within the current namespace to the **this** keyword .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.

6.Uncaught TypeError: 'undefined' is not a function

This is a common JavaScript error that happens when we try to call a function before it is defined. we get this error when we try to execute a function that is uninitialized or improperly initialized. It means that the expression did not return a function object.

**Fix**

Assign a reasonable value to the object during construction or initialization — do not keep it undefined.

**7.Maximum call stack size exceeded error**

This error is almost always means we have a problem with recursion in JavaScript code, as there isn't any other way in JavaScript to consume lots of stack. Sometimes calling a recursive function over and over again, causes the browser to send you Maximum call stack size exceeded error message as the memory that can be allocated for your use is not unlimited.

**Fix**

Be considerate while calling functions, also dry run is the best practice to prevent them. It's possible to cause infinite recursion in a fully promisified code, too. That can happen if the promises in a chain don't actually perform any asynchronous execution, in which case control never really returns to the event loop, even though the code otherwise appears to be asynchronous. That's when it's useful to wrap our recursive function call into a -

* setTimeout
* setImmediate or
* process.nextTick

Also, you can localize the issue by setting a breakpoint on RangeError type of exception, and then adjust the code appropriately. Moreover, we can managed to find the point that was causing the error by check the error details in the Chrome dev toolbar console , this will give the functions in the call stack, and guide us towards the recursion that's causing the error.

The Call Stack is what a program uses to keep track of method calls. The call stack is made up of stack frames — one for each method call. It is primarily used for function invocation (call). Since the call stack is single, function(s) execution, is done, one at a time, from top to bottom. It means that the call stack is synchronous.

When we enter a function, an entry for that function is pushed onto the Call stack and when we exit from the function, that same entry is popped from the Call Stack. Each method call creates its own stack frame, taking up space on the call stack. That's important because it can impact the space complexity of an algorithm, especially when we use recursion. if we provide too many arguments or caught inside any unhandled recursive call. we will encounter Maximum call stack size exceeded error.

**8. typeerror: cannot read property 'length'**

This error occurs when you define a normal variable instead of array variable.

**Fix**

We have to be carefully check before accessing a variable which is normal or array variable so as to avoid these errors.

**9. Uncaught TypeError: Cannot set property**

In JavaScript almost everything is an object, null and undefined are exception. 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.

**Fix**

Assign a reasonable value to the object during construction or initialization — do not keep it undefined.

**10. Reference Error: event is not defined**

This is an error that occurs, when we didn’t specify parameter to the event handlers.

**Fix**

Ensures that our event handlers are passed the event parameter.