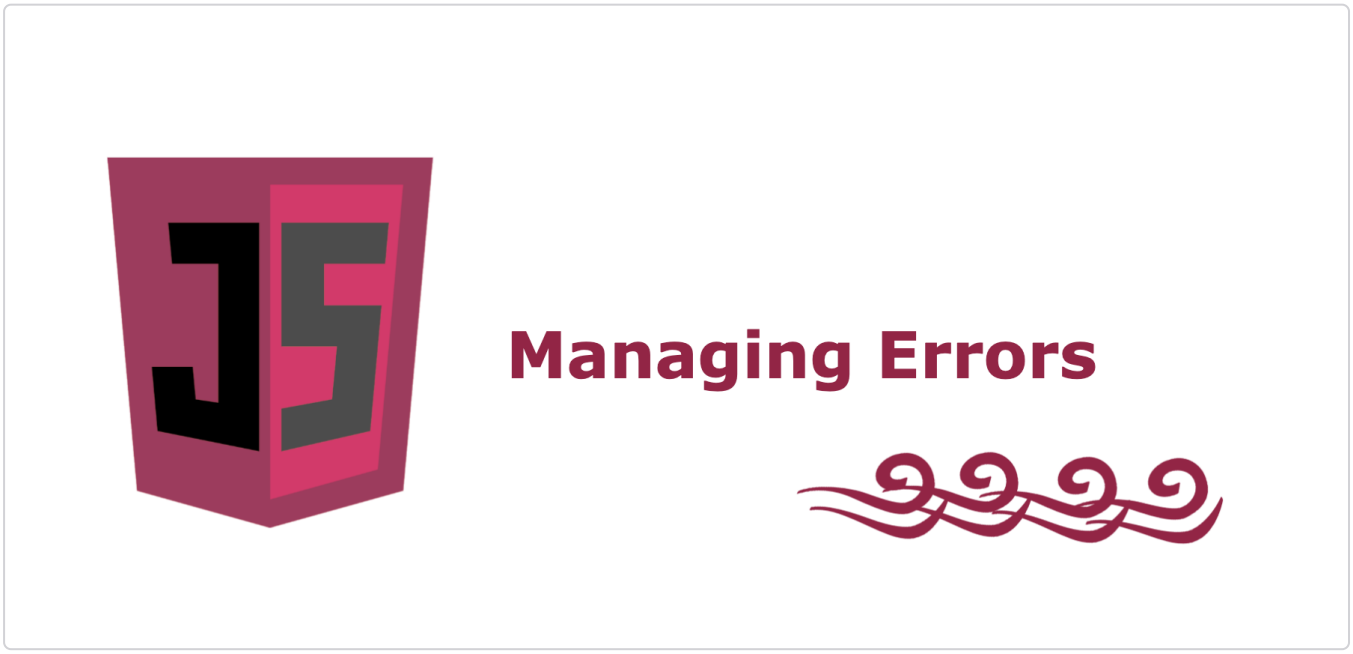
Managing Errors

In this lesson, we learn how to manage and cater to errors that arise in JavaScript code.
Let's begin!

WE'LL COVER THE FOLLOWING

- Preventing and handling errors



Preventing and handling errors is a topic that deserves a full book dedicated to it. Here, I give you a brief overview and some useful advice.

Preventing and handling errors

You must have an error-handling strategy for the JavaScript layer of a web application, because any JavaScript error can cause a web page to become unusable. The basis of this strategy should be understanding when and why errors occur. Most users are not technical and can easily get confused when something doesn’t work as expected. In most cases when they face an error, they reload the page or try to navigate away and then back. If the problem does not get fixed after their attempts, they get frustrated and stop using your app.

As the developer, you should create robust applications that prevent users from making errors and, whenever possible, catch potential issues before they become errors.

You should design and implement the UI so that it immediately indicates potential errors like specifying invalid values or options. If users provide invalid data, you must respond with helpful, non-technical messages that lead the user to the right way to correct what they misused. For example, users get frustrated with an error message “Use the `‘\[A-Z]{3}-\d{3}’` format to specify registration number”, because they do not understand it.

Instead, use the “Registration number must be three letters, followed by a dash, and closed by three decimal digits” message to help them.

If possible, check any data before sending it to a web server. Although a robust application must check all input data at server side and while omitting validation at the client side must not hurt the business logic, checking data in the browser will save resources at the server side, provided invalid data is not sent to the server.

Since JavaScript is loosely typed, and its function arguments are not verified by the engine, many errors will come to light only when the code is executed. There are a number of errors that might be caused by these peculiarities of JavaScript.

Due to the loosely-typed nature of JavaScript, variables and function arguments aren’t compared to ensure that the correct type of data is being used. Data type errors most often occur as a result of unexpected values being passed into a function. For example, this code snippet causes an error in the highlighted line:

```
function splitName(str) {
  var posComma = str.indexOf('=');
  if (posComma > 1) {
    return str.substring(0, posComma);
  }
  return "";
}

var name1 = splitName("key=value");
```

```
console.log(name1);



var name2 = splitName(123);
console.log(name2);
```

The `splitName()` function implicitly expects a string. When you pass a number, as the code does in the highlighted line, the execution fails because a number does not have an `indexOf()` function.

It is up to you as the developer to do an appropriate amount of data type checking to ensure that an error will not occur. The JavaScript language has a number of constructs that automatically change the type of a value or an expression. This behavior leads to coercion errors that are not easy to catch.

For example, the `getKeyValuePair()` function in this example has an if statement that may lead to coercion error:

```
function getKeyValuePair(key, value) {
  var result = key + "=";
  if (value) {
    result += value;
  }
  return result;
}

var pair1 = getKeyValuePair("key");
var pair2 = getKeyValuePair("key", 13);
var pair3 = getKeyValuePair("key", 0);

console.log(pair1); // key=
console.log(pair2); // key=13
console.log(pair3); // key
```

The aim of the if statement is to check whether the value argument is provided. If it is not, the value evaluates to undefined and is coerced by the JavaScript engine to false in the condition of the if statement.

The issue is that there are other values that are also coerced to false, such as the number 0,so the method provides faulty operation.

Communication with the server side also provides an opportunity for errors to occur including malformed URLs, network issues, timeouts, etc.

> 📜**NOTE:** Most web applications log user activities and other system events at the server side in files or databases. For diagnostic purposes you may use the same services to log JavaScript errors. There are many logging patterns, search the web for “JavaScript error logging” with your favorite search engine.

In the *next lesson*, we will summarize what we have learned in this chapter.

See you there! :)