Handling JavaScript Exceptions

Lesson Objectives

When you complete this lesson, you will be able to:

- *throw and catch exceptions.*
- *use the finally clause to execute code regardless of what happens in the try/catch.*

As you have realized by now, there are plenty of things that can go wrong when writing JavaScript code! There are unexpected events, bugs, and mistyped data submitted by users. So far we've handled these *exceptional conditions* and *errors* either by ignoring them (not a good long-term plan!) or by testing with if/then/else statements to check for certain conditions.

JavaScript includes another way of handling *exceptions*: the *try/catch* statement. *Try/catch* lets you *try* some code, and, if something goes wrong, you can *catch* the error and do something about it.

In this lesson you'll learn how to use *try/catch* (and the optional *finally* part of this statement) to handle exceptions.

What Causes an Exception?

An exception is often caused by an error in your JavaScript that causes your code to stop executing. That is, it's an error from which the JavaScript interpreter can't easily recover. In a previous lesson, you wrote some code to see whether the result of calling `Date.parse()` on an invalid string was equal to `NaN`. Clearly, if you give `Date.parse()` an invalid string, that's an error (in this case, a user error for using the wrong date format), but it's not a fatal error. JavaScript is happy to set the result to `NaN` and proceed. Of course, later on in your code, the fact that `NaN` is not a real number might cause an exception, but that's another issue.

So what kind of code causes an exception that causes your code to stop running? Let's take a look at an example:

INTERACTIVE SESSION:

```
var myString = null;
myString.length;
```

Open a browser window, and open the JavaScript console (you may have to load a web page to be able to access the console. Any web page will do, including a previous file you've created in the course). Type in those two lines; you see a JavaScript Error like this (in Safari):

```
> var myString = null;
  undefined
> myString.length;
⊗ ▼Error
     line: 2
     message: "'null' is not an object (evaluating 'myString.length')"
     sourceId: 5089147688
   ▶ __proto__: Error
>
```

...or like this (in Firefox):

```
16:45:52.701 ◀ var myString = null;
16:45:52.703 ▶ undefined
16:45:55.999 ◀ myString.length;
16:45:56.001 ✕ TypeError: myString is null
```

Note Error messages in various browsers will differ slightly, but all browsers should give you an exception for this code, and a similar error message. Try different browsers to see what you get!

An error like this in your code will cause the code to stop executing. Let's try making an error in code loaded with an HTML page (rather than just at the console), and this time, we'll *try* it and *catch* the error with the *try/catch* statements. First, we'll create a super-simple HTML page, and then the JavaScript to create the error.

Create a new HTML file and type the code as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
<title>Exceptions</title>
  <meta charset="utf-8">
  <script src="ex.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
```

```
  </style>
</head>
<body>
</body>
</html>
```

Save the file in your work folder as *ex.html*. Next, create a new JavaScript file as shown:

CODE TO TYPE:

```
window.onload = init;

function init() {
    var myString = null;
    try {
        var len = myString.length;
        console.log("Len: " + len);
    }
    catch (ex) {
        console.log("TypeError: " + ex.message);
    }
}
```

Save this *as ex.js* in your work folder. Now, load *ex.html* into your browser, and open the JavaScript console and you see an error message like this (in Safari):

OBSERVE:

```
TypeError: 'null' is not an object (evaluating 'myString.length')
```

...or like this (in Firefox):

OBSERVE:

```
TypeError: myString is null
```

The error message generated from your JavaScript code is the same as those you saw using the console earlier.

Let's take a closer look at the *try/catch* statement:

OBSERVE:

```
function init() {
    var myString = null;
    try {
        var len = myString.length;
```

```
        console.log("Len: " + len);
    }
    catch (ex) {
        console.log("Error: " + ex.message);
    }
}
```

After setting `myString` to null, which we know will cause an error when we try to access the `length` property (because null doesn't have a length property), we start the *try/catch* block. We use { and } to delimit each part of the statement. These are required.

JavaScript will try to execute all the code in the try part of the statement. If it works and no exception is created, then the try ends normally, and the catch is *not* executed. So the flow of execution would continue below the catch. In this case, that means the function simply returns.

But if something goes wrong, and an exception is generated, as we know it will in this code, then as soon as the exception is generated, the flow of execution jumps from the try to the catch. In this case, that means we never see the `console.log()` message that displays the value of `len`.

JavaScript automatically generates a value for the exception and passes it into the catch clause (much like passing an argument to a function parameter). For errors generated internally by the JavaScript interpreter, like this one is, that value is typically an *Error* object. Here, we assume it is such an error, and we name that object `ex`, and access its message property to display in the console, with information about what the error was.

So in this code, we cause, or *raise* (as it's often called), an exception by attempting to access a property that doesn't exist, and we *catch* that exception so that it doesn't cause our program to stop running entirely. This is usually better for the application. If you handle the errors that are caused in your program gracefully, then the end user can continue using your application, whereas if your JavaScript stops running, that might cause your application to stop working altogether!

Throwing Exceptions

You might want to use the *try/catch* statement in situations where JavaScript might not raise an exception internally, but where you are testing for errors or exceptional conditions in your code. In that case, you can raise your own exceptions by using the *throw* statement. Let's expand our example just a bit and throw our own exception (and catch it, of course). Modify *ex.html* as shown:

CODE TO TYPE:

```
<!doctype html>
<html>
<head>
<title>Exceptions</title>
  <meta charset="utf-8">
  <script src="ex.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
    }
  </style>
</head>
<body>
    <h1>Enter a string</h1>
    <p id="stringInfo"></p>
    <p id="error"></p>
    <p id="msg"></p>
</body>
</html>
```

Save these changes, and then update *ex.js* as follows:

CODE TO TYPE:

```
window.onload = init;

function init() {
    var myString = null;
    try {
        var len = myString.length;
        console.log("Len: " + len);
    }
    catch (ex) {
        console.log("Error: " + ex.message);
    }
    var myString = prompt("Enter a string:");
    try {
        var len = myString.length;
        if (len == 0) {
            throw new Error("You didn't enter anything. Try again.");
        }
        else {
            displayLength(myString, len);
        }
    }
    catch (ex) {
        displayError(ex.message);
    }
```

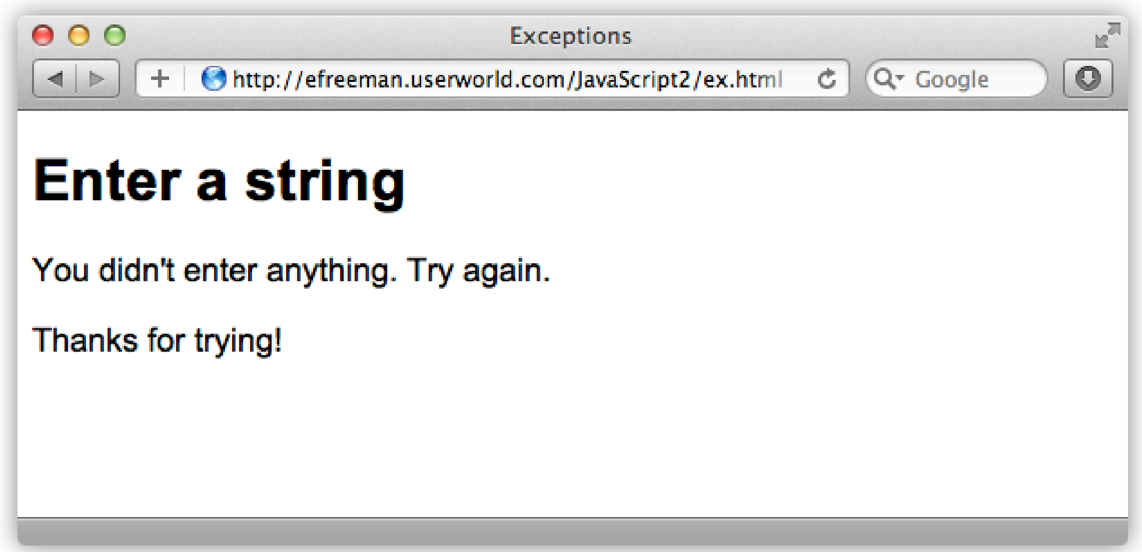
```
    finally {
        displayMessage("Thanks for trying!");
    }
}

function displayError(e) {
    var error = document.getElementById("error");
    error.innerHTML = e;
}

function displayLength(myString, len) {
    var stringInfo = document.getElementById("stringInfo");
    stringInfo.innerHTML = "The string '" + myString + "' has length:
" + len;
}

function displayMessage(m) {
    var msg = document.getElementById("msg");
    msg.innerHTML = m;
}
```

Save these changes, and open or refresh *ex.html* again in the browser. You'll be prompted to enter a string. Try entering a real string; see what happens. Now try clicking *OK* without entering anything, and see what happens.



Let's go through the code to see what's going on:

OBSERVE:

```
function init() {
    var myString = prompt("Enter a string:");
    try {
        var len = myString.length;
        if (len == 0) {
            throw new Error("You didn't enter anything. Try again.");
        }
        else {
            displayLength(myString, len);
        }
    }
    catch (ex) {
        displayError(ex.message);
    }
    finally {
        displayMessage("Thanks for trying!");
    }
}
```

Now, instead of setting `myString` to null, we're prompting the user to enter a string. Once we've done that, we try to get the length of the string. When you use `prompt()`, even if you don't enter anything, the value returned is still a string (in that case, it would be an empty string, ""), and getting the length of an empty string will result in 0, rather than an exception.

We can check to see if the length is 0, and if it is, we can throw our own exception. You can throw any value you want. In this case, we are throwing an *Error object* (just like JavaScript did in the previous example). As soon as we throw the Error, the code skips any other code in the try clause, and jumps to the catch clause. There, we pass the value of the message property from the Error object to the function `displayError()`, which displays that value in the web page. The value of message is the value we passed to the `Error()` constructor when we threw the Error.

If the length is greater than 0, we display the string and the length of the string by passing the two values to `displayLength()`.

The Finally Clause

We added on a finally clause in this example. This clause is executed whether the exception is thrown or not. In other words, if the length is 0, and we execute the catch clause, once the catch is complete, we execute the finally clause. If the length is greater than 0, we execute the

code in the try clause, skip the catch clause (because there's no exception), and execute the finally clause.

Finally allows you execute some code regardless of what happens in the *try/catch*, so it's handy for clean-up code, for example. In this case, all we do is display the same message whether the prompt is successful or not.

Using Exceptions and Try/Catch

In the previous lesson, we created a program to parse a string and convert it to a JavaScript *Date*, but we weren't checking to make sure the `Date.parse()` actually worked!

This is a good example of where we can use *try/catch* and throw an *exception* instead. Doing this will make the code more robust. It's a good way to handle this type of error. Let's give it a try. Edit *dates.js* as shown:

CODE TO TYPE:

```
window.onload = init;

function init() {
    var submit = document.getElementById("submit");
    submit.onclick = getDate;
}

function getDate() {
    var aDate;
    var aDateString = document.getElementById("aDate").value;
    if (aDateString == null || aDateString == "") {
        alert("Please enter a date");
        return;
    }
    var aDateMillis = Date.parse(aDateString);
    alert(aDateMillis);
    var aDate = new Date(aDateMillis);
    try {
        if (isNaN(aDateMillis)) {
            throw new Error("Date format error. Please enter the date
in the format MM/DD/YYYY, YYYY/MM/DD, or January 1, 2012");
        }
        else {
            aDate = new Date(aDateMillis);
        }
        var datetime = document.getElementById("datetime");
        datetime.innerHTML = aDate.toLocaleString();
    }
    catch (ex) {
        alert(ex.message);
```

```
    }
}
```

Save your changes, and open *dates.html* in your browser. Try entering a string using a format the program will recognize, and a string that it will not recognize. You see the same behavior you saw at the end of the previous lesson, but the way we handle the error (that is, the format that the program can't parse) is different. Now we use *try/catch* and throw an *exception* if we can't match the date format the user has entered.

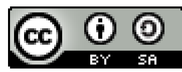
Notice that as soon as we throw the *Error* object when we can't match the format, the rest of the *try* clause is skipped, so we don't have to check to see if an error was generated. As soon as that *Error* is thrown, the code jumps directly to the *catch* clause. This is a convenient way to tell your program to, "stop everything we're doing and go here!" This technique can be really useful. It can also make the code a bit easier to read.

Remember that the *try* clause must always be matched with either a *catch* or a *finally*. Typically, you'll see *try/catch*, but you'll find *finally* will also come in handy. If you want to raise your own exceptions, you can use *throw* and throw a value that is caught by the *catch* clause parameter. You can throw any value you want. JavaScript typically throws the *Error* object.

You can do this too by creating a `new Error` object, and passing in the value for the *message* property. Always check the JavaScript documentation in a good reference to find out exactly which type of exception to expect for circumstances where JavaScript might not throw the *Error* object, so you know which kind of value to expect in your *catch* clause.

More about the material in this lesson

The Riverside JS Workshop would like to acknowledge the generosity of O'Reilly Media, Inc. for making this material available to us through the Creative Commons License. We would also like to acknowledge the great work of Elisabeth Robson who authored this content. She is one of our favorites. Elisabeth has written a number of *Head First* programming books for web developers. We recommend them highly. You can browse her work here.



Copyright © 1998-2015 O'Reilly Media, Inc.
This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License.
See http://creativecommons.org/licenses/by-sa/3.0/legalcode for more information.

The original source document has been altered. It has been edited to accommodate this format and enhance readability.