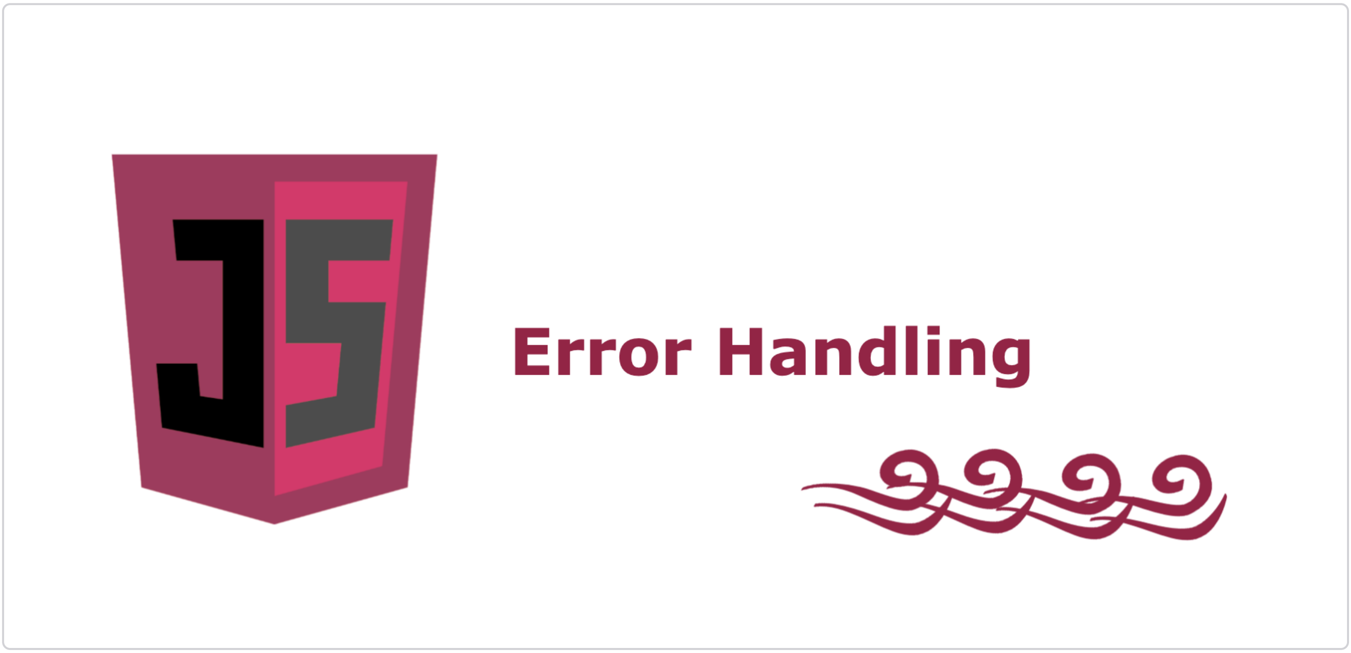
Error Handling

In this lesson, we will learn a key concept in JavaScript, error handling.
Let's begin!

WE'LL COVER THE FOLLOWING

- The `try-catch` construct
- Error types
- JavaScript error types
- Throwing exceptions



To write robust programs that resist common user errors and other issues in the usage context, you must endeavor to find and handle all potential errors. In most cases, validating input arguments takes the lion's share of this work, but this is not always enough.

There might be issues that cannot be checked in advance, and you'll observe the error only after an operation has been invoked. Most modern programing languages contain exception handling mechanism to cope with this situation.

The `try-catch` construct

JavaScript provides this machinery, too, it uses the `try-catch` exception handling block, just like the other curly-brace languages. The syntax of the statement block is this:

```
try {
  // --- Code that may cause an error
} catch (error) {
  // --- Activity to do when an error occurs
} finally {
  // --- Cleanup code
}
```

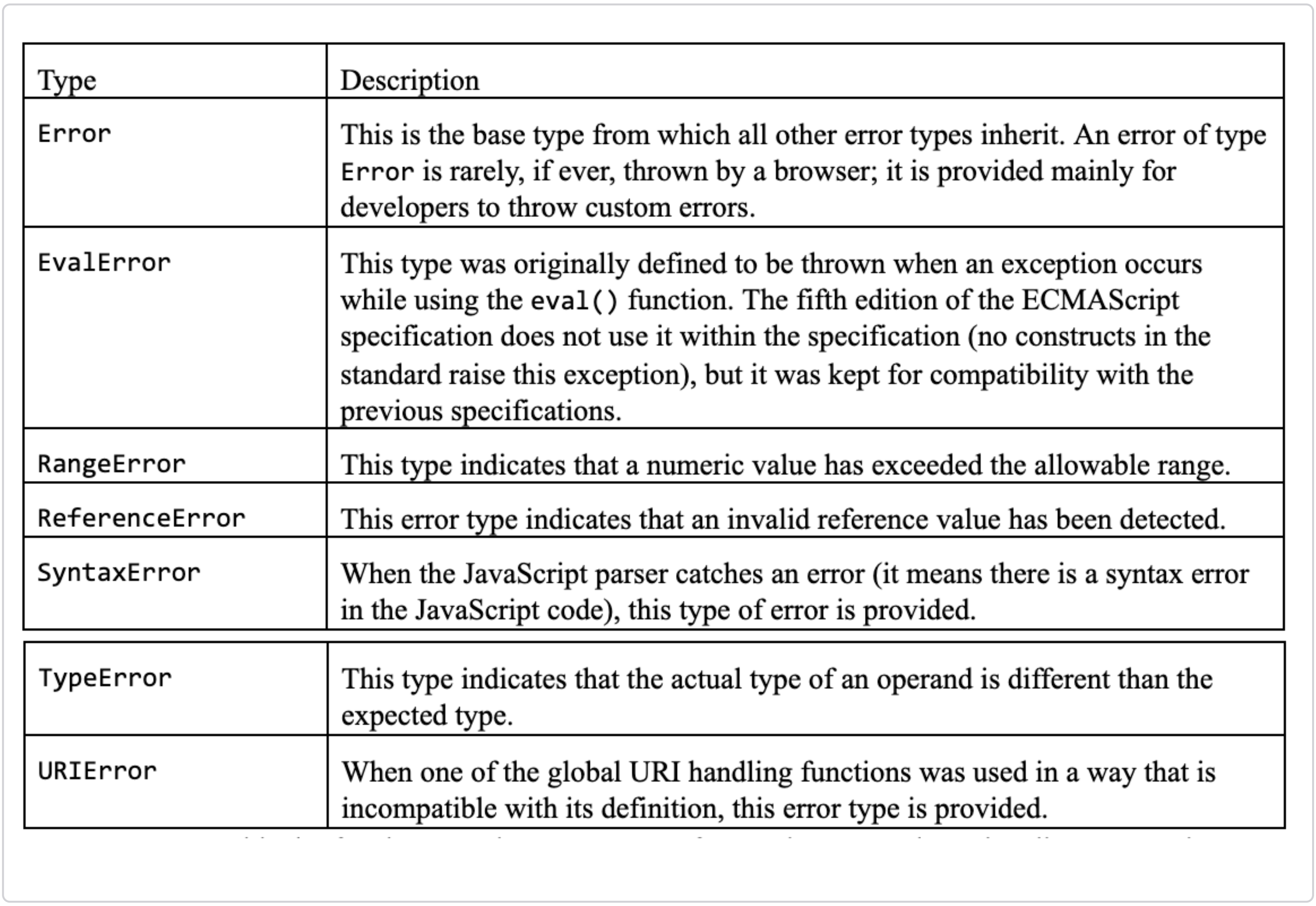
You enclose the code that may cause an error into the block between `try` and `catch`. When the error occurs, the execution is immediately transferred to the `catch` branch, where `error` is an object that carries information about the error. The code in the `finally` block is always executed independently whether the try block was completed or interrupted by an error. There is no way to prevent the `finally` block from running.

You can omit either the catch or the `finally` block, but not both. If you omit the `finally` block, you do not have the chance for cleanup activity, but you often do not need this. If you omit the `catch` block, you cannot specify code to handle the error, but you can be sure that your cleanup code runs.

Error types

There are several types of errors that may occur during code execution. The **ECMAScript** standard defines a set of object types to represent these errors, as summarized in the table below:

JavaScript error types



Type	Description
`Error`	This is the base type from which all other error types inherit. An error of type `Error` is rarely, if ever, thrown by a browser; it is provided mainly for developers to throw custom errors.
`EvalError`	This type was originally defined to be thrown when an exception occurs while using the `eval()` function. The fifth edition of the ECMAScript specification does not use it within the specification (no constructs in the standard raise this exception), but it was kept for compatibility with the previous specifications.
`RangeError`	This type indicates that a numeric value has exceeded the allowable range.
`ReferenceError`	This error type indicates that an invalid reference value has been detected.
`SyntaxError`	When the JavaScript parser catches an error (it means there is a syntax error in the JavaScript code), this type of error is provided.
`TypeError`	This type indicates that the actual type of an operand is different than the expected type.
`URIError`	When one of the global URI handling functions was used in a way that is incompatible with its definition, this error type is provided.

In many cases, a block of code may raise more types of exceptions. In order to handle an exception, you must know its type and traits.

The error variable of the catch clause points to the object describing the exception, and you can use it to branch the handler code, as shown in this code snippet:

```
try {
  myFunctionThatMayRaiseErrors();
} catch (error){
  if (error instanceof TypeError){
    // An unexpected type provided,
    // handle here
  } else if (error instanceof ReferenceError){
    // An invalid reference value
    // was used, handle here
  } else {
    // Other kind of error found,
    // handle here
  }
}
```

Throwing exceptions

An exception handling mechanism must allow catching and throwing exceptions. With the throw operator you can indicate an error that can be

caught with `try-catch`, as shown here:

```
throw new EvalError("Can't evaluate this.");
throw new RangeError("Posotive value expected.");
throw new ReferenceError("Can't handle this reference");
throw new SyntaxError("What's this: 'oumaer'?");
throw new TypeError("A RegExp instance expected");
throw new URIError("Can't find this resource");
```

Although the JavaScript core throws only the exceptions defined by the ECMAScript specification, you can throw any kind of object, as shown in this sample code:

```
throw "Hey! This is an error!"
throw 42;
throw {code: 42, message: "Disk access failure"};
```

With prototype chaining you can easily define your own error objects, as demonstrated by this short example:

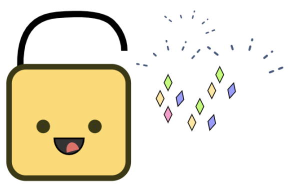
```
function NegativeNumberUsed() {
  this.name = "NegativeNumberFound";
  this.message = "A negative number found";
}

NegativeNumberUsed.prototype = new Error();

// ...
throw new NegativeNumberUsed();
```

Achievement unlocked!

Congratulations! You’ve learned error handling in JavaScript.



Great work! Give yourself a round of applause! :)

In the *next lesson*, we’ll get to know the `onerror` Event in detail.