JavaScript Errors

❮ Previous Next ❯

Throw, and Try...Catch...Finally

The `try` statement defines a code block to run (to try).

The `catch` statement defines a code block to handle any error.

The `finally` statement defines a code block to run regardless of the result.

The `throw` statement defines a custom error.

Errors Will Happen!

When executing JavaScript code, different errors can occur.

Errors can be coding errors made by the programmer, errors due to wrong input, and other unforeseeable things.

Example

In this example we misspelled "alert" as "adddlert" to deliberately produce an error:

☑ Dark mode

```
<script>
try {
  adddlert("Welcome guest!");
}
catch(err) {
  document.getElementById("demo").innerHTML = err.message;
}
</script>
```

Try it Yourself »

JavaScript catches **adddlert** as an error, and executes the catch code to handle it.

JavaScript try and catch

The `try` statement allows you to define a block of code to be tested for errors while it is being executed.

The `catch` statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements `try` and `catch` come in pairs:

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
```

ADVERTISEMENT

☑ Dark mode

JavaScript Throws Errors

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will **throw an exception (throw an error)**.

JavaScript will actually create an **Error object** with two properties: **name** and **message**.

The throw Statement

The `throw` statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

The exception can be a JavaScript `String`, a `Number`, a `Boolean` or an `Object`:

```
throw "Too big";    // throw a text
throw 500;          // throw a number
```

☑ Dark mode

Input Validation Example

This example examines input. If the value is wrong, an exception (err) is thrown.

The exception (err) is caught by the catch statement and a custom error message is displayed:

```
<!DOCTYPE html>
<html>
<body>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="myFunction()">Test Input</button>
<p id="p01"></p>

<script>
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```

☑ Dark mode

HTML Validation

The code above is just an example.

Modern browsers will often use a combination of JavaScript and built-in HTML validation, using predefined validation rules defined in HTML attributes:

```
<input id="demo" type="number" min="5" max="10" step="1">
```

You can read more about forms validation in a later chapter of this tutorial.

The finally Statement

The `finally` statement lets you execute code, after try and catch, regardless of the result:

Syntax

```
try {
  Block of code to try
}
catch(err) {
  Block of code to handle errors
}
finally {
  Block of code to be executed regardless of the try / catch result
}
```

✓ Dark mode

```
function myFunction() {
  const message = document.getElementById("p01");
  message.innerHTML = "";
  let x = document.getElementById("demo").value;
  try {
    if(x == "") throw "is empty";
    if(isNaN(x)) throw "is not a number";
    x = Number(x);
    if(x > 10) throw "is too high";
    if(x < 5) throw "is too low";
  }
  catch(err) {
    message.innerHTML = "Error: " + err + ".";
  }
  finally {
    document.getElementById("demo").value = "";
  }
}
```

Try it Yourself »

The Error Object

JavaScript has a built in error object that provides error information when an error occurs.

The error object provides two useful properties: name and message.

Error Object Properties

Property	Description
name	Sets or returns an error name

☑ Dark mode

Error Name Values

Six different values can be returned by the error name property:

Error Name	Description
EvalError	An error has occurred in the eval() function
RangeError	A number "out of range" has occurred
ReferenceError	An illegal reference has occurred
SyntaxError	A syntax error has occurred
TypeError	A type error has occurred
URIError	An error in encodeURI() has occurred

The six different values are described below.

Eval Error

An `EvalError` indicates an error in the eval() function.

Newer versions of JavaScript do not throw EvalError. Use SyntaxError instead.

Range Error

A `RangeError` is thrown if you use a number that is outside the range of legal values.

For example: You cannot set the number of significant digits of a number to 500.

☑ Dark mode

```
let num = 1;
try {
  num.toPrecision(500);    // A number cannot have 500 significant digits
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Try it Yourself »

Reference Error

A `ReferenceError` is thrown if you use (reference) a variable that has not been declared:

Example

```
let x = 5;
try {
  x = y + 1;    // y cannot be used (referenced)
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Try it Yourself »

Syntax Error

A `SyntaxError` is thrown if you try to evaluate code with a syntax erroı

☑ Dark mode

```
try {
  eval("alert('Hello)");   // Missing ' will produce an error
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Try it Yourself »

Type Error

A `TypeError` is thrown if you use a value that is outside the range of expected types:

Example

```
let num = 1;
try {
  num.toUpperCase();   // You cannot convert a number to upper case
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Try it Yourself »

URI (Uniform Resource Identifier) Error

A `URIError` is thrown if you use illegal characters in a URI function:

✓ Dark mode

```
try {
  decodeURI("%%%");   // You cannot URI decode percent signs
}
catch(err) {
  document.getElementById("demo").innerHTML = err.name;
}
```

Try it Yourself »

Non-Standard Error Object Properties

Mozilla and Microsoft defines some non-standard error object properties:

fileName (Mozilla)
lineNumber (Mozilla)
columnNumber (Mozilla)
stack (Mozilla)
description (Microsoft)
number (Microsoft)

Do not use these properties in public web sites. They will not work in all browsers.

Complete Error Reference

For a complete reference of the Error object, go to our Complete JavaScript Error Reference.

❮ Previous Next ❯

☑ Dark mode