**try...catch**

The **try...catch** statement is comprised of a try block and either a catch block, a finally block, or both. The code in the try block is executed first, and if it throws an exception, the code in the catch block will be executed. The code in the finally block will always be executed before control flow exits the entire construct.

[**Try it**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#try_it)

[**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#syntax)

try {

tryStatements

} catch (exceptionVar) {

catchStatements

} finally {

finallyStatements

}

Copy to Clipboard

tryStatements

The statements to be executed.

catchStatements

Statement that is executed if an exception is thrown in the try-block.

exceptionVar Optional

An optional identifier to hold the caught exception for the associated catch block. If the catch block does not utilize the exception's value, you can omit the exceptionVar and its surrounding parentheses, as catch {...}.

finallyStatements

Statements that are executed before control flow exits the try...catch...finally construct. These statements execute regardless of whether an exception was thrown or caught.

[**Description**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#description)

The try statement always starts with a try block. Then, a catch block or a finally block must be present. It's also possible to have both catch and finally blocks. This gives us three forms for the try statement:

* try...catch
* try...finally
* try...catch...finally

Unlike other constructs such as [if](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else) or [for](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for), the try, catch, and finally blocks must be *blocks*, instead of single statements.

try doSomething(); // SyntaxError

catch (e) console.log(e);

A catch-block contains statements that specify what to do if an exception is thrown in the try-block. If any statement within the try-block (or in a function called from within the try-block) throws an exception, control is immediately shifted to the catch-block. If no exception is thrown in the try-block, the catch-block is skipped.

The finally block will always execute before control flow exits the try...catch...finally construct. It always executes, regardless of whether an exception was thrown or caught.

You can nest one or more try statements. If an inner try statement does not have a catch-block, the enclosing try statement's catch-block is used instead.

You can also use the try statement to handle JavaScript exceptions. See the [JavaScript Guide](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide) for more information on JavaScript exceptions.

[**Unconditional catch-block**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#unconditional_catch-block)

When a catch-block is used, the catch-block is executed when any exception is thrown from within the try-block. For example, when the exception occurs in the following code, control transfers to the catch-block.

try {

throw "myException"; // generates an exception

} catch (e) {

// statements to handle any exceptions

logMyErrors(e); // pass exception object to error handler

}

Copy to Clipboard

The catch-block specifies an identifier (e in the example above) that holds the value of the exception; this value is only available in the [scope](https://developer.mozilla.org/en-US/docs/Glossary/Scope) of the catch-block.

[**Conditional catch-blocks**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#conditional_catch-blocks)

You can create "Conditional catch-blocks" by combining try...catch blocks with if...else if...else structures, like this:

try {

myroutine(); // may throw three types of exceptions

} catch (e) {

if (e instanceof TypeError) {

// statements to handle TypeError exceptions

} else if (e instanceof RangeError) {

// statements to handle RangeError exceptions

} else if (e instanceof EvalError) {

// statements to handle EvalError exceptions

} else {

// statements to handle any unspecified exceptions

logMyErrors(e); // pass exception object to error handler

}

}

Copy to Clipboard

A common use case for this is to only catch (and silence) a small subset of expected errors, and then re-throw the error in other cases:

try {

myRoutine();

} catch (e) {

if (e instanceof RangeError) {

// statements to handle this very common expected error

} else {

throw e; // re-throw the error unchanged

}

}

Copy to Clipboard

[**The exception identifier**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#the_exception_identifier)

When an exception is thrown in the try-block, exception\_var (i.e., the e in catch (e)) holds the exception value. You can use this identifier to get information about the exception that was thrown. This identifier is only available in the catch-block's [scope](https://developer.mozilla.org/en-US/docs/Glossary/Scope). If you don't need the exception value, it could be omitted.

function isValidJSON(text) {

try {

JSON.parse(text);

return true;

} catch {

return false;

}

}

Copy to Clipboard

[**The finally-block**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#the_finally-block)

The finally block contains statements to execute after the try block and catch block(s) execute, but before the statements following the try...catch...finally block. Control flow will always enter the finally block, which can proceed in one of the following ways:

* Immediately before the try block finishes execution normally (and no exceptions were thrown);
* Immediately before the catch block finishes execution normally;
* Immediately before a control-flow statement (return, throw, break, continue) is executed in the try block or catch block.

If an exception is thrown from the try block, even when there's no catch block to handle the exception, the finally block still executes, in which case the exception is still thrown immediately after the finally block finishes executing.

The following example shows one use case for the finally-block. The code opens a file and then executes statements that use the file; the finally-block makes sure the file always closes after it is used even if an exception was thrown.

openMyFile();

try {

// tie up a resource

writeMyFile(theData);

} finally {

closeMyFile(); // always close the resource

}

Copy to Clipboard

Control flow statements (return, throw, break, continue) in the finally block will "mask" any completion value of the try block or catch block. In this example, the try block tries to return 1, but before returning, the control flow is yielded to the finally block first, so the finally block's return value is returned instead.

function doIt() {

try {

return 1;

} finally {

return 2;

}

}

doIt(); // returns 2

Copy to Clipboard

It is generally a bad idea to have control flow statements in the finally block. Only use it for cleanup code.

[**Examples**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#examples)

[**Nested try-blocks**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#nested_try-blocks)

First, let's see what happens with this:

try {

try {

throw new Error("oops");

} finally {

console.log("finally");

}

} catch (ex) {

console.error("outer", ex.message);

}

// Logs:

// "finally"

// "outer" "oops"

Copy to Clipboard

Now, if we already caught the exception in the inner try-block by adding a catch-block:

try {

try {

throw new Error("oops");

} catch (ex) {

console.error("inner", ex.message);

} finally {

console.log("finally");

}

} catch (ex) {

console.error("outer", ex.message);

}

// Logs:

// "inner" "oops"

// "finally"

Copy to Clipboard

And now, let's rethrow the error.

try {

try {

throw new Error("oops");

} catch (ex) {

console.error("inner", ex.message);

throw ex;

} finally {

console.log("finally");

}

} catch (ex) {

console.error("outer", ex.message);

}

// Logs:

// "inner" "oops"

// "finally"

// "outer" "oops"

Copy to Clipboard

Any given exception will be caught only once by the nearest enclosing catch-block unless it is rethrown. Of course, any new exceptions raised in the "inner" block (because the code in catch-block may do something that throws), will be caught by the "outer" block.

[**Returning from a finally-block**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#returning_from_a_finally-block)

If the finally-block returns a value, this value becomes the return value of the entire try-catch-finally statement, regardless of any return statements in the try and catch-blocks. This includes exceptions thrown inside of the catch-block:

(() => {

try {

try {

throw new Error("oops");

} catch (ex) {

console.error("inner", ex.message);

throw ex;

} finally {

console.log("finally");

return;

}

} catch (ex) {

console.error("outer", ex.message);

}

})();

// Logs:

// "inner" "oops"

// "finally"

Copy to Clipboard