JavaScript Errors

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

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 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`:

JavaScript Scope

Scope determines the accessibility (visibility) of variables.

JavaScript variables have 3 types of scope:

- Block scope
- Function scope
- Global scope

Block Scope

Before ES6 (2015), JavaScript variables had only Global Scope and Function Scope.

ES6 introduced two important new JavaScript keywords: `let` and `const`.

These two keywords provide Block Scope in JavaScript.

Variables declared inside a { } block cannot be accessed from outside the block

Function Scope

JavaScript has function scope: Each function creates a new scope.

Variables defined inside a function are not accessible (visible) from outside the function.

Variables declared with `var`, `let` and `const` are quite similar when declared inside a function

Global Scope

Variables declared Globally (outside any function) have Global Scope.

Global variables can be accessed from anywhere in a JavaScript program.

Variables declared with `var`, `let` and `const` are quite similar when declared outside a block.

The let and const Keywords

Variables defined with `let` and `const` are hoisted to the top of the block, but not *initialized*.

Meaning: The block of code is aware of the variable, but it cannot be used until it has been declared.

Using a `let` variable before it is declared will result in a `ReferenceError`.

JavaScript this Keyword

object method, `this` refers to the object.

Alone, `this` refers to the global object.

In a function, `this` refers to the global object.

In a function, in strict mode, `this` is `undefined`.

In an event, `this` refers to the element that received the event.

Methods like `call()`, `apply()`, and `bind()` can refer `this` to any object.

this in a Method

When used in an object method, `this` refers to the object.

In the example on top of this page, `this` refers to the person object.

Because the fullName method is a method of the person object.

Explicit Function Binding

The `call()` and `apply()` methods are predefined JavaScript methods.

They can both be used to call an object method with another object as argument.

Function Borrowing

With the `bind()` method, an object can borrow a method from another object.

This example creates 2 objects (person and member).

The member object borrows the fullname method from the person object