## Const

const is a new keyword which declares a variable as *constant over time*.

**Declaring a const variable**

We can use const to declare a variable but unlike let and var it must be *immediately* initialised, with a value that can’t be changed afterwards.

If we try to declare it without initialising it we get a SyntaxError, like so:

const foo; // SyntaxError: Missing initializer in const declaration

If we try to change it after we have declared and initialised it we get a TypeError, like so:

const foo = 1;

foo = 2; // TypeError: Assignment to constant variable

**Block scoping**

Both let and const create variables that are block-scoped – they only exist within the innermost block that surrounds them.

function func() {

if (true) {

const tmp = 123;

}

console.log(tmp); // Uncaught ReferenceError: tmp is not defined

}

func();

**Immutable variable**

Variables created by

let

and

var

are

*mutable*

:

let foo = "foo";

foo = "moo";

console.log(foo);



*Mutable*

in this case means

*can change over time*

.

Variables created by const however are *immutable*, they don’t change over time, specifically the the const variable can’t *point to* another thing later on.

const foo = 'abc';

foo = 'def'; // TypeError: Assignment to constant variable

**Mutable Value**

There is one big pitfall with const however.

When we say "`const` variables are immutable" it only means that the variable always has to *point to* the same thing. It does not mean than the *thing* it points to can’t change over time.

For example, if the variable foo is a const that points to an object - we can’t make foo point to another object later on:

const foo = {};

foo = {}; // TypeError: Assignment to constant variable.

But we can however *mutate*, make changes to, the object foo points to, like so:

const foo = {};

foo['prop'] = "Moo"; // This works!

console.log(foo);

If we want the *value* of foo to be immutable we have to freeze it using Object.freeze(…).

When we freeze an object we can’t change it, we can’t add properties or change the values of properties, like so:

const foo = Object.freeze({});

foo.prop = 123;

console.log(foo.prop) // undefined

However by default the above doesn’t throw an error, it just *silently* ignores the issue. So in the example above it didn’t throw an error when we tried to set foo.prop = 123 but when we try to print out the value on the next line we just get undefined.

To force Object.freeze(…) to throw an error we must remember to be in "use strict" mode, like so:

"use strict";

const foo = Object.freeze({});

foo.prop = 123; // SyntaxError: Identifier 'foo' has already been declared

**Summary**

const lets us declare variables which don’t change over time, which are immutable.

The important *gotcha* with const is that the variable is immutable, but not the value, the thing the variable points to.

This means that if we declare an object as const, confusingly we can still change properties of the object later on.

To solve this and make an object immutable we use the Object.freeze(…) function which together with the "use strict"; param throws an error if we try to change the object.

**Listing**

*script.js*

'use strict';

/\* // Declaring a const variable

const foo; // SyntaxError: Missing initializer in const declaration

\*/

/\* // Declaring a const variable

const moo = 1;

moo = 2; // TypeError: `foo` is read-only

\*/

/\* // Block Scoping

function func() {

if (true) {

const tmp = 123;

}

console.log(tmp); // Uncaught ReferenceError: tmp is not defined

}

func();

\*/

const foo = Object.freeze({});

foo.prop = 1;

console.log(foo.prop);

*index.html*

<!

DOCTYPE html

>

<

html

>

<

head

>

<script src="script.js"></script>

<

/head

>

<

/html

>