mdn web docs_

Hoisting

JavaScript **Hoisting** refers to the process whereby the interpreter appears to move the *declaration* of functions, variables, classes, or imports to the top of their scope, prior to execution of the code.

Hoisting is not a term normatively defined in the ECMAScript specification. The spec does define a group of declarations as *HoistableDeclaration*, but this only includes `function`, `function*`, `async function`, and `async function*` declarations. Hoisting is often considered a feature of `var` declarations as well, although in a different way. In colloquial terms, any of the following behaviors may be regarded as hoisting:

1. Being able to use a variable's value in its scope before the line it is declared. ("Value hoisting")
2. Being able to reference a variable in its scope before the line it is declared, without throwing a `ReferenceError`, but the value is always `undefined`. ("Declaration hoisting")
3. The declaration of the variable causes behavior changes in its scope before the line in which it is declared.
4. The side effects of a declaration are produced before evaluating the rest of the code that contains it.

The four function declarations above are hoisted with type 1 behavior; `var` declaration is hoisted with type 2 behavior; `let`, `const`, and `class` declarations (also collectively called *lexical declarations*) are hoisted with type 3 behavior; `import` declarations are hoisted with type 1 and type 4 behavior.

Some prefer to see `let`, `const`, and `class` as non-hoisting, because the temporal dead zone strictly forbids any use of the variable before its declaration. This dissent is fine, since hoisting is not a universally-agreed term. However, the temporal dead zone can cause other observable changes in its scope, which suggests there's some form of hoisting:

JS

```js
const x = 1;
{
  console.log(x); // ReferenceError
  const x = 2;
}
```

If the `const x = 2` declaration is not hoisted at all (as in, it only comes into effect when it's executed), then the `console.log(x)` statement should be able to read the `x` value from the upper scope. However, because the `const` declaration still "taints" the entire scope it's defined in, the `console.log(x)` statement reads the `x` from the `const x = 2` declaration instead, which is not yet initialized, and throws a `ReferenceError`. Still, it may be more useful to characterize lexical declarations as non-hoisting, because from a utilitarian perspective, the hoisting of these declarations doesn't bring any meaningful features.

Note that the following is not a form of hoisting:

JS

```js
{
  var x = 1;
}
console.log(x); // 1
```

There's no "access before declaration" here; it's simply because `var` declarations are not scoped to blocks.

For more information on hoisting, see:

- `var` / `let` / `const` hoisting — Grammar and types guide
- `function` hoisting — Functions guide
- `class` hoisting — Classes guide
- `import` hoisting — JavaScript modules