Watch: 🧟 **Internals: Hoisting y Temporal Dead Zone**

Hoisting es como se suele explicar el funcionamiento al declarar variables y funciones. Tal vez habrás leído que JavaScript mueve la declaración arriba del scope. Es por esto que usar una variable antes de su declaración con `var` devuelve undefined:

```
console.log(one); // undefined
var one = 1;
console.log(one); // 1
```

El snippet anterior daría el mismo resultado si separamos declaración y inicialización y movemos la declaración arriba, tal y como pasaría si JavaScript moviera realmente nuestro código según esa definición de hoisting:

```
var one;
console.log(one); // undefined
one = 1;
console.log(one); // 1
```

En realidad, lo que entendemos como hoisting es que, a pesar de que JavaScript sea un lenguaje interpretado, tiene un paso de compilación, aunque pasa en el mismo navegador unas milésimas de segundo antes de su ejecución. Durante ese paso, el compilador guarda las variables en el scope, pero sin su valor. En el paso de ejecución, cuando el *engine* encuentra una variable, es capaz de encontrarla en el scope pese a que esta sea declarada después en nuestro código, porque ha habido el paso previo de compilación.

Si ejecutamos el código anterior con `let`, vamos a ver que se comporta diferente:

```
console.log(one); // Uncaught ReferenceError: can't access
lexical declaration 'one' before initialization
let one = 1;
console.log(one);
```

Se suele decir que `let` y `const` no *hoistean*, pero en realidad lo que pasa es que, durante el paso de compilación, se encuentran pero se guardan en el scope como **no inicializadas**. Si movemos la declaración arriba, el código funciona como esperamos ya que es en momento de ejecución que la variable `let` se inicializa como `undefined` en la línea 1:

```
let one;
console.log(one); // undefined
one = 1;
console.log(one); // 1
```

Enlaces relacionados

- MDN: Hoisting
- Hoisting in Modern JavaScript — let, const, and var
- MDN: Temporal Dead Zone