HOISTING IN JAVASCRIPT

👉 **Hoisting:** Makes some types of variables accessible/usable in the code before they are actually declared. "Variables lifted to the top of their scope".

BEHIND THE SCENES

Before execution, code is scanned for variable declarations, and for each variable, a new property is created in the **variable environment object**.

EXECUTION CONTEXT

- 👉 Variable environment
- ✅ Scope chain
- 👉 `this` keyword

	HOISTED? 👇	INITIAL VALUE 👇	SCOPE 👇
`function` declarations	✅ YES	Actual function	Block
`var` variables	✅ YES	`undefined`	Function
`let` and `const` variables	🚫 NO	`<uninitialized>`, TDZ	Block
`function` expressions and arrows		🤷 Depends if using `var` or `let/const`	

In strict mode. Otherwise: function!

Technically, yes. But not in practice

Temporal Dead Zone

46 people have written a note here.

TEMPORAL DEAD ZONE, LET AND CONST

```
const myName = 'Jonas';

if (myName === 'Jonas') {
  console.log(`Jonas is a ${job}`);
  const age = 2037 - 1989;
  console.log(age);
  const job = 'teacher';
  console.log(x);
}
```

TEMPORAL DEAD ZONE FOR **job** VARIABLE

👉 Different kinds of error messages:

```
ReferenceError: Cannot access 'job' before initialization
ReferenceError: x is not defined
```

46 people have written a note here.

HOW THE THIS KEYWORD WORKS

👉 **`this` keyword/variable:** Special variable that is created for every execution context (every function). Takes the value of (points to) the "owner" of the function in which the `this` keyword is used.

☝ `this` is **NOT** static. It depends on **how** the function is called, and its value is only assigned when the function **is actually called**.

EXECUTION CONTEXT

- ✅ Variable environment
- ✅ Scope chain
- 👉 `this` keyword

Method 👉 `this` = <Object that is calling the method>

Simple function call 👉 `this` = `undefined`

In strict mode! Otherwise: `window` (in the browser)

Don't get own `this`

Arrow functions 👉 `this` = <`this` of surrounding function (lexical `this`)>

Event listener 👉 `this` = <DOM element that the handler is attached to>

`new`, `call`, `apply`, `bind` 👉 *<Later in the course... ⏳>*

☝ `this` does **NOT** point to the function itself, and also **NOT** the its variable environment!

👉 Method example:

```
const jonas = {
  name: 'Jonas',
  year: 1989,
  calcAge: function() {
    return 2037 - this.year;
  }
};
jonas.calcAge(); // 48
```

`calcAge` is method

`jonas`

`1989`

Way better than using `jonas.year`!