

# Javascript-Behind the science

# HOISTING

👉 **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

# TDZ

```
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

## WHY HOISTING?

👉 Using functions before actual declaration;

## WHY TDZ?

👉 Makes it easier to avoid and catch errors: accessing variables before declaration is bad practice and should be avoided;

👉 Makes `const` variables actually work

# HOSITING PRACTICE

The image shows a web browser window with a video player and a JavaScript console. The video player has a green title bar that reads "How JavaScript Works Behind the Scenes". Below the video, the browser's developer tools are open, showing the "Console" tab. The console displays the following log entries:

- undefined script.js:46
- 5 script.js:55
- Uncaught ReferenceError: Cannot access 'addExpr' before initialization at script.js:56
- Live reload enabled. (index):55

The JavaScript code in the console is as follows:

```
script.js > addExpr
46 console.log(me);
47 // console.log(job);
48 // console.log(year);
49
50 var me = 'Jonas';
51 let job = 'teacher';
52 const year = 1991;
53
54 // Functions
55 console.log(addDecl(2, 3));
56 console.log(addExpr(2, 3));
57 console.log(addArrow(2, 3));
58
59 function addDecl(a, b) {
60   return a + b;
61 }
62
63 const addExpr = function (a, b) {
64   return a + b;
65 };
66
67 const addArrow = (a, b) => a + b;
68
```

# THIS KEYWORD

- 👉 **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
```

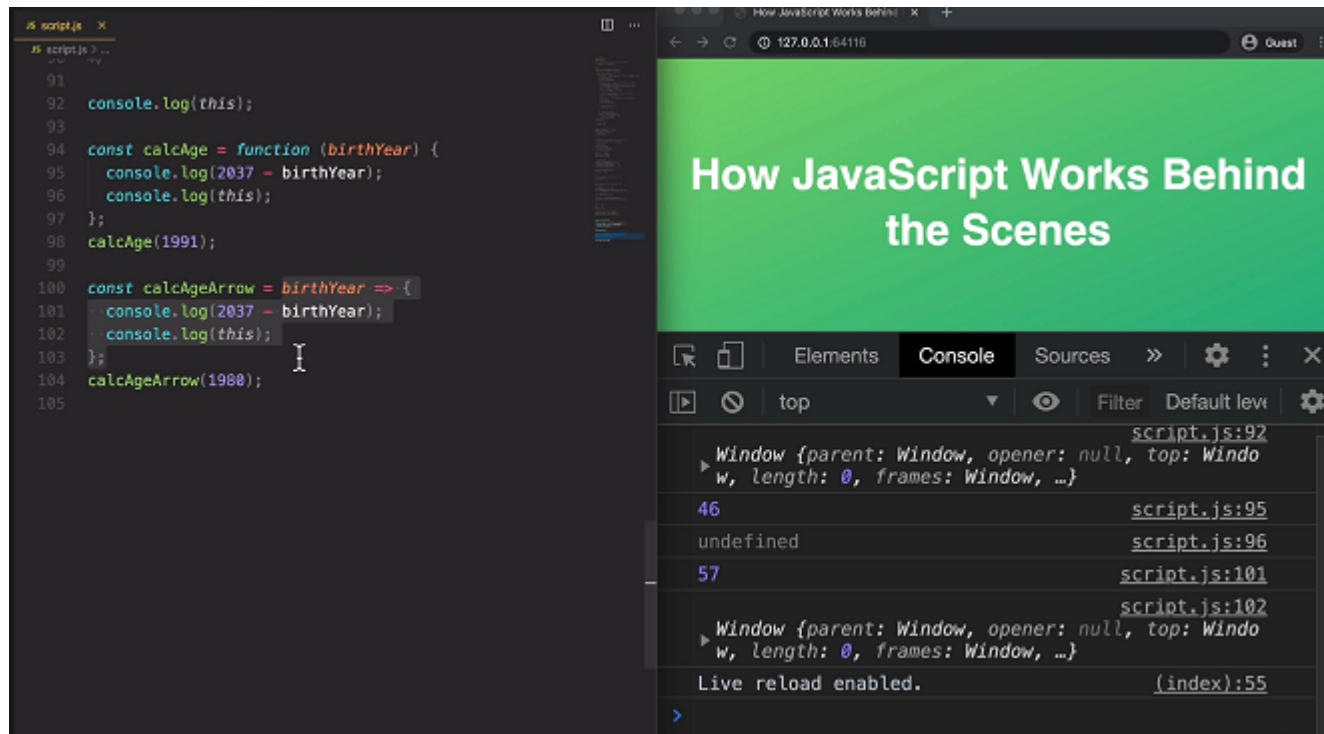
calcAge  
is method

jonas

1989

Way better than using  
jonas.year!

# THIS KEYWORD PRACTICE



The image shows a web browser window with the title "How JavaScript Works Behind the Scenes" and a green header. Below the header, the browser's developer tools are open, showing the "Console" tab. The console displays the output of the JavaScript code executed in the background. The code is written in a dark-themed editor on the left, showing a function `calcAge` and an arrow function `calcAgeArrow`. The console output shows the results of `calcAge(1991)` and `calcAgeArrow(1980)`, both returning a `Window` object. The console also shows the `Live reload enabled.` message.

```
script.js:92 console.log(this);
script.js:95 const calcAge = function (birthYear) {
script.js:96   console.log(2037 - birthYear);
script.js:97   console.log(this);
script.js:98 };
script.js:99 calcAge(1991);
script.js:100 const calcAgeArrow = birthYear => {
script.js:101   console.log(2037 - birthYear);
script.js:102   console.log(this);
script.js:103 };
script.js:104 calcAgeArrow(1980);
script.js:105
```

How JavaScript Works Behind the Scenes

Window {parent: Window, opener: null, top: Window, length: 0, frames: Window, ...}

46 script.js:95

undefined script.js:96

57 script.js:101

Window {parent: Window, opener: null, top: Window, length: 0, frames: Window, ...}

Live reload enabled. (index):55

# THIS KEYWORD IN NORMAL FUNCTION AND ARROW FUNCTION

```
<script>
var akash = {
  name: 'Akash',
  showTasks: function() {
    var _this = this;
    function display(){
      console.log(_this.name);
    }
    display();
  }
};
akash.showTasks();
</script>
```

```
<script>
var akash = {
  name: 'Akash',
  showTasks: function() {
    const display=()=>{
      console.log(this.name);
    }
    display();
  }
};
akash.showTasks();
</script>
```