

Javascript-Behind the science

JavaScript strict mode

- JavaScript is a loosely typed (dynamic) scripting language
- JavaScript allows strictness of code using "use strict" with ECMAScript 5 or later.
- Write "use strict" at the top of JavaScript code or in a function

```
"use strict";
```

```
var x = 1; // valid in strict mode  
y = 1; // invalid in strict mode
```

```
<script>  
  "use strict";  
  y = 4;  
  console.log(y);  
  
  function add(a,a)  
  {  
    return a+a;  
  }  
  add(22,22);  
</script>
```

SCOPE CHAIN

- 👉 **Scoping:** How our program's variables are **organized** and **accessed**. *"Where do variables live?" or "Where can we access a certain variable, and where not?"*;
- 👉 **Lexical scoping:** Scoping is controlled by **placement** of functions and blocks in the code;
- 👉 **Scope:** Space or environment in which a certain variable is **declared** (*variable environment in case of functions*). There is **global** scope, **function** scope, and **block** scope;
- 👉 **Scope of a variable:** Region of our code where a certain variable can be **accessed**.

TYPES OF SCOPE

GLOBAL SCOPE

```
const me = 'Jonas';  
const job = 'teacher';  
const year = 1989;
```

- ✚ Outside of **any** function or block
- ✚ Variables declared in global scope are accessible **everywhere**

FUNCTION SCOPE

```
function calcAge(birthYear) {  
  const now = 2037;  
  const age = now - birthYear;  
  return age;  
}  
  
console.log(now); // ReferenceError
```

- ✚ Variables are accessible only **inside function**, NOT outside
- ✚ Also called local scope

BLOCK SCOPE (ES6)

```
if (year >= 1981 && year <= 1996) {  
  const millenial = true;  
  const food = 'Avocado toast';  
} ← Example: if block, for loop block, etc.  
  
console.log(millenial); // ReferenceError
```

- ✚ Variables are accessible only **inside block** (block scoped)
- ⚠ **HOWEVER**, this only applies to **let** and **const** variables!
- ✚ Functions are **also block scoped** (only in strict mode)

SCOPE CHAIN

```
const myName = 'Jonas';

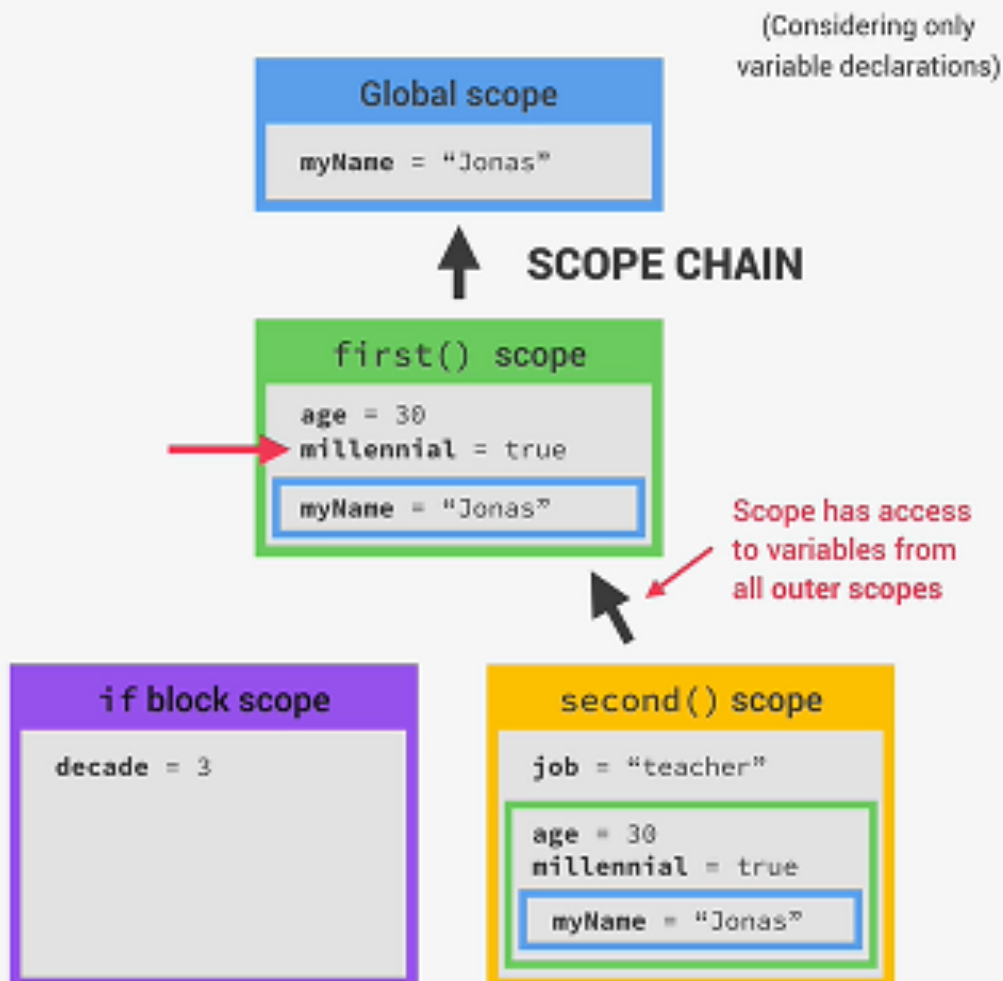
function first() {
  const age = 30;
  if (age >= 30) { // true
    const decade = 3;
    var millennial = true;
  }
  function second() {
    const job = 'teacher';
    console.log(`$myName is a $age-old ${job}`);
    // Jonas is a 30-old teacher
  }
  second();
}

first();
```

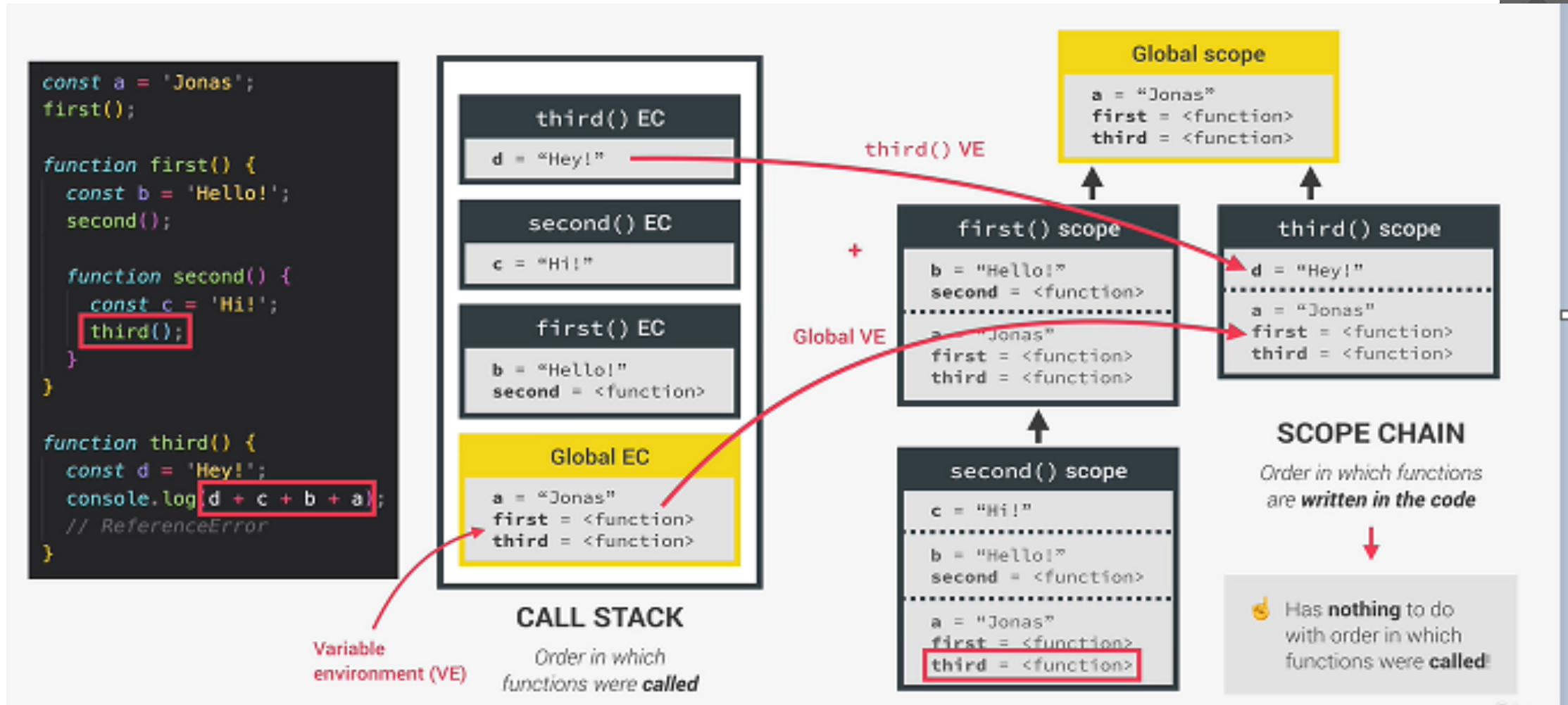
let and const are **block-scoped**

var is **function-scoped**

Variables not in current scope



SCOPE CHAIN VS CALL STACK



SCOPE PRACTICE

```
5
6 function printAge() {
7   let output = `${firstName}, you are ${age}, born
8   in ${birthYear}`;
9   console.log(output);
10
11   if (birthYear >= 1981 && birthYear <= 1996) {
12     var millennial = true;
13     // Creating NEW variable with same name as
14     // outer scope's variable
15     const firstName = 'Steven';
16
17     // Reassigning outer scope's variable
18     output = 'NEW OUTPUT!';
19
20     const str = `Oh, and you're a millennial, $
21     {firstName}`;
22     console.log(str);
23
24     function add(a, b) {
25       return a + b;
26     }
27
28     // console.log(str);
29     console.log(millennial);
30     // console.log(add(2, 3));
31     console.log(output);
32   }
33 }
34 printAge();
```

How JavaScript Works Behind the Scenes

	Elements	Console	Sources	»	⚙	⋮	✕
▶	🔍	top	▼	👁	Filter	Default level	⚙
		Jonas, you are 46, born in 1991				script.js:8	
		Oh, and you're a millennial, Steven				script.js:19	
		true				script.js:26	
		NEW OUTPUT!				script.js:28	
		Live reload enabled.				(index):55	
		>					