SOFT CLUB

Java Script

Lecture 2

SOFT CLUB

Table of Contents

SOFT CLUB

What is Scope in JavaScript ?

What is a scope in js ?

The scopeisthecurrentcontextofexecutioninwhichvaluesandexpressions are"visible" orcanbereferenced. Ifa variableorexpressionisnotinthecurrent scope, itwillnotbeavailableforuse. Scopescanalsobelayeredina hierarchy, sothatchildscopeshaveaccesstoparentscopes, butnotviceversa.

JavaScript has the following kinds of scopes:

- Global scope: The default scope for all code running in script mode.
- Function scope: The scope created with afunction.

Block scope: This scope restricts the variable that is declared inside a specific block, from access by the outside of the block.

- Module scope: The scope for code running in module mode.

The 3 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)

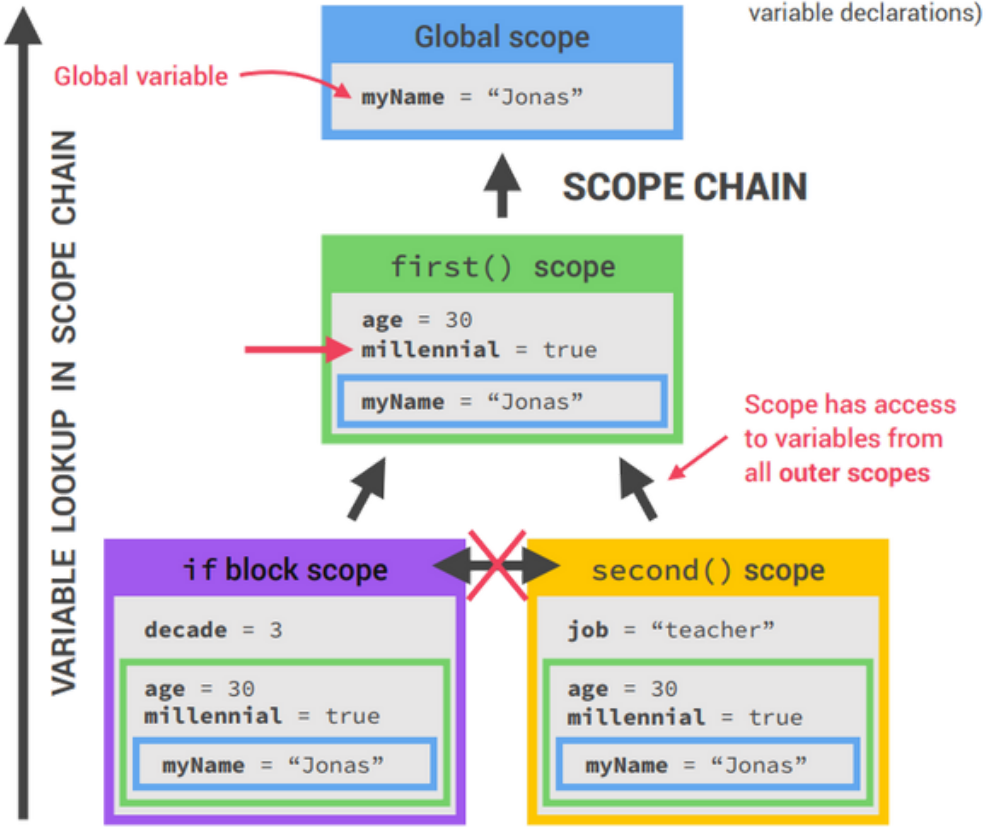
The scope chain

```
const myName = 'Jonas';

function first() {
  const age = 30;

  if (age >= 30) {  // true
    const decade = 3;
    var millenial = 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



SOFT CLUB

What is Hoisting in JavaScript ?

Hoisting in java script

- Hoisting isa JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.

  Hoisting in JavaScript isa behavior in which a function or a variable can be used before declaration.

```
a = 2;

var a;

console.log( a );
```

Temporal Dead Zone JS?

A variabledeclared withletorconst cannot be accessed untilitisdeclared withinitsscope.

	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

Hoisting –Variable(var)

There’s a temptation to think that all of the code you see in a JavaScript
program is interpreted line-by-line, top-down in order, as the program execute. While that is essentially true, there’s one part of that as-assumption that can lead to incorrect thinking about your program.

```
a = 2;

var a;

console.log( a );
```

Hoisting –function declaration

So, one way of thinking, sort of metaphorically, about this process, is that variable and function
declarations are “moved” from where they appear in the flow of the code to the top of the code. This gives rise to the name hoisting.

```
foo();

function foo() {
    console.log( a ); // undefined

    var a = 2;
}
```

The function foo’s declaration (which in this case includes the implied value of it as an actual function) is hoisted, such that the call on the first line is able to execute

Temporal dead zone, letand 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
```

WHY HOISTING?

- **Using functions before actual declaration;**
- `var` hoisting is just a byproduct.

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**

SOFT CLUB

Thanks!

Be happy and Smile

The END
Lecture 2