Scope

The Complete Web Developer in 2018

The Complete Web Developer in 2018
Zero to Mastery
Andrei Neagoie
Lecture Notes by Stephanie

Scope - variable access

If we define a function aa( )…

```
> function aa() {
      console.log("Test!");
  }
```

We can execute aa( ) because it is within the ‘root scope’

```
> aa()
  Test!                                    console.js:35
```

The JavaScript default is that you are in the 'root scope' which is the 'window' object… the 'aa' function has been added to the 'window scope'.

Therefore, we can also execute it this way…

```
> window.aa()
  Test!                                    console.js:35
```

If we define function bb( )…

```
> function bb() {
      var a = "hello";
  }
```

Variable 'a' only lives within 'function bb'… the scope is inside of the function

```
> console.log(a);
✕ ▶Uncaught ReferenceError: a is not defined    VM6589:1
      at <anonymous>:1:13
```

We can think of it like this:

```
> function bb() {
      // child scope
      var a = "hello";
  }

  // root scope (window) = parent scope
  console.log(a);
✕ ▶Uncaught ReferenceError: a is not defined    VM7021:7
      at <anonymous>:7:13
```

We get an error because we cannot access the child scope from the parent scope.

However, functions have access to any variable in the root scope, because we can access the parent scope from the child scope

```
> // root scope (window) = parent scope
  var c = "Can I access this?";

  function dd() {
      // child scope
      console.log(c);
  }
```

'window.dd' exists and variable ‘c’ lives on the window object, meaning they both have the same parent…

```
> dd()
  Can I access this?                    console.js:35
```

In this example, we look at the child scope to log variable “c”. The function says, well I don't know what variable “c” is, but ask my parent. So then we ask the parent of the child. We continue up until the last parent, which is always the root scope.

Let's look at a different example…

```
> // root scope (window)
  var c = "Can I access this?";

  function ee() {
      // child scope
      c = "My function changed it!";
  }
```

Let's log "c"… we look at the root scope (parent)

```
> console.log(c);
  Can I access this?                    console.js:35
```

When we execute the function… we look at the child scope when calling the function…

```
> ee()
```

Now when we log "c"… (looking at the root scope (parent))

```
> console.log(c);
  My function changed it!               console.js:35
```

Exercise: Scope

Section 13, Lecture 132

It's time to code some javascript! Get your sublime text ready for this exercise, and use Google Chrome javascript console to test your code. You can find the exercise file and the solution file attached. Good luck!

PS - if you have any questions throughout this section, reach out to our student community in the **#js** or **#helpme** channel on Discord (Lecture 3 provides the link if you have not yet joined)!

```
>
// For all of these, what is the value of a
// when the function gets called with the
// alert()?
```

```
// #1
function q1() {
    var a = 5;
    if(a > 1) {
        a = 3;
    }
    alert(a);
}
q1();
// My answer:
//
// 3
```

```
//#2
var a = 0; // parent/root scope
function q2() {
    a = 5; // child scope
}

function q22() {
    alert(a); // child scope
}

q22();
// My answer:
// If you only run q22():
//
// 0
```

```
//#3
function q3() {
    window.a = "hello";
}


function q32() {
    alert(a);
}

q3();
q32();
// My answer:
// If you run q3() to add the property to the window
// and then run q32():
//
// "hello"
```

```
//#4
var a = 1; // parent scope
function q4() {
    var a = "test"; // child scope
    alert(a);
}

q4();
// My answer:
//
// "test"
```

```
//#5
var a = 2; // parent scope
if (true) {
    var a = 5; // child scope
    alert("if is true");  // child scope
}
alert(a);
// My answer:
//
// "if is true"
// 5
```

```
//#6
var a = 2; // parent scope
if (false) {
    var a = 5; // child scope
    alert("if is true");  // child scope
}
alert(a);
// My answer:
//
// 2
```

Scope of “let” and “const” (ES5/ES6 only)

With variable, we can only create a new scope within a function. But with 'let', we can create a new scope within an if-statement.

Use “let” to create scope:

```
> const player = 'bobby';
  let experience = 100;
  let wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      let wizardLevel = true; // scope is only within if statemt
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM188:5
inside2 true                      VM188:9
outside3 false                    VM188:12
```

Doesn’t work if we don’t say “let” within the if-statement:

```
> const player = 'bobby';
  let experience = 100;
  let wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      wizardLevel = true; // scope is root scope
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM42:5
inside2 true                      VM42:9
outside3 true                     VM42:12
```

“Const” will also create new scope…

```
> const player = 'bobby';
  let experience = 100;
  let wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      const wizardLevel = true; // scope is only within if statemt
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM829:5
inside2 true                      VM829:9
outside3 false                    VM829:12
```

If we use var:

```
> const player = 'bobby';
  let experience = 100;
  var wizardLevel = false;

  console.log("outside1",wizardLevel);

  if (experience > 90) {
      var wizardLevel = true; // scope is root scope
      console.log("inside2",wizardLevel);
  }

  console.log("outside3",wizardLevel);
```

```
outside1 false                    VM52:5
inside2 true                      VM52:9
outside3 true                     VM52:12
```