02. SCOPE JAVASCRIPT

IES ESTACIÓ CURS 2021- 2022

Índex

1. Què és el Scope en JavaScript?

- El scope d'una variable fa referència al lloc on viurà o podrà ser accessible.

- Podríem dir també que scope és l'abast que determina l'accessibilitat de les variables en cada part del nostre codi.

2. Per a què ens serveix el Scope?

- Entendre bé el concepte de scope ens ajudarà a augmentar el nivell de seguretat ja que delimita els qui tenen accés i els qui no a determinades parts del nostre codi.

- També ens facilitarà en la detecció i disminució d'errors, per tant el nostre codi serà més robust.

3. Tipus de Scope.

- **3.1 Global Scope.**
  - Es diu que una variable es troba en el scope global quan està declarada fora d'una funció o d'un bloc.
  - Podrem accedir a aquest tipus de variables des de qualsevol part del nostre codi, ja siga dins o fora d'una funció.
  - L'objecte window és un exemple de scope global.
  - Exemple de global scope:

```
var x = "función externa declarada";


exampleFunction();

function exampleFunction() {
    console.log("funcion interna");
    console.log(x);
}

console.log("funcion externa");
console.log(x);
```

3. Tipus de Scope.

- **3.2 Local Scope.**
  - Les variables que definim dins d'una funció són variables locals, és a dir es troben en el Scope local.
  - Això significa que aquest tipus de variables viuran únicament dins de la funció on les hàgem declarades i si intentem accedir-les fora d'ella, aquestes variables no estaran definides.
  - Això ens permet decidir si volem una variable només per a una determinada funció.
  - Exemple de local scope:

```
function exampleFunction() {
    var x = "declarada dentro de la función"; // x solo se puede utilizar en exampleFunction
    console.log("funcion interna");
    console.log(x);
}

console.log(x);  // error
```

3. Tipus de Scope.

- **3.3 Global automàtica.**
  - Si assignem un valor a una variable que no ha sigut declarada, aquesta es convertirà automàticament en una variable global.
  - Exemple de global automàtica:

```
1.  myFunction();
2.
3.  // El código en esta parte va a poder acceder a la variable alfajor.
4.
5.  function myFunction() {
6.
7.    alfajor = "Jorgito";
8.
9.  }
```

3. Tipus de Scope.

- **3.4 Block Scope.**
  - A diferència del scope local el block scope està limitat al bloc de codi on va ser definida la variable.
  - Des de ECMAScript 6 comptem amb els keyword let i const els quals ens permeten tindre un scope de bloc, això vol dir que les variables només viuran dins del bloc de codi corresponent.
  - Exemple de block scope:

```
1.  if (true) {
2.      // este bloque if no crea un scope
3.
4.      // la variable nombre es global por el uso de la keyword 'var'
5.      var nombre = 'Juan';
6.
7.      // preferencias se encuentra en el scope local por el uso de la keyword 'let'
8.      let preferencias = 'Codear';
9.
10.     // skills también es una scope local por el uso de la keyword 'const'
11.     const skills = 'Java';
12. }
13.
14. console.log(nombre); // logs 'Juan'
15. console.log(preferencias); // Uncaught ReferenceError: preferencias is not defined
16. console.log(skills); // Uncaught ReferenceError: skills is not defined
```

3. Tipus de Scope.

- **3.5 Lexical Scope.**
  - El lexical scope significa que en un grup niat de funcions, les funcions internes tenen accés a les variables i altres recursos del seu àmbit pare.
  - Això significa que les funcions filles estan vinculades lèxicament al context d'execució dels seus pares.
  - Exemple de lexical scope:

```
1.  function myFunction() {
2.
3.      var nombre = 'Juan';
4.      // no podemos acceder a preferencias desde acá
5.
6.      function parent() {
7.
8.          // nombre si es accesible desde acá.
9.          // preferencias en cambio, no es accesible.
10.
11.         function child() {
12.
13.             // tambien podemos acceder a nombre desde acá.
14.             var preferencias = 'Codear';
15.         }
16.     }
17. }
```

4. ‘use strict’

- Podem activar el strict mode mitjançant la directive: “use *strict”;

- Aquesta directiva es troba disponible des de ECMAScript 5 i ens permet escriure codi més segur, ja que habilita més restriccions al nostre codi, per al nostre cas en particular per exemple, no podrem usar variables no declarades.

```
1.  "use strict";
2.
3.  mejorJugador = "Messi";       // Esto va a lanzar un error porque mejorJugador no está declarada
```