SCOPE

Cuando nos referimos al “scope” nos estamos refiriendo al alcance o al área donde una variable puede utilizarse, en este tema veremos cómo maneja JavaScript este concepto. En JavaScript el alcance de las variables sucede de una función y no de bloques (if, while, switch, etc.) como en lenguajes Java o C/C++; en otras palabras, si se define una variable dentro de un bloque condicional (if) esta variable se podrá utilizar en toda la función en cuestión y no solamente en el bloque definido. Veamos el siguiente ejemplo que demuestra lo que se acaba de mencionar:

```
if(true){
        var test ='Variable';
}
function testing(){
        var test = 'Variable SCOPE';
```

```
}

testing();

console.debug(test);
```

Dentro de la condición se ha definido la variable “test”, en lenguajes como Java ésta variable debería existir sólo dentro de la condición, pero en JavaScript no sucede de esta manera ya que esa variable ha sido definida en el “global scope” y no dentro del bloque condicional. Por otro lado, la variable que se definió dentro de la función “testing” sólo existe dentro de esa función. Es importante mencionar que las variables declaradas en el “global scope” son propiedades del objeto “window”, para comprobar esta afirmación basta con hacer lo siguiente:

```
var global = 'Es una variable global!';

console.debug(window.global);
```

Otro punto a tomar en cuenta es que cuando no se declaran las variables utilizando la palabra reservada “var” no importa si están dentro de una función o no, estas variables automáticamente serán definidas en el “global scope”.

```
function globalScopeFunction(){
        globalScope = 'Nueva variable';
}

globalScopeFunction();

console.debug(globalScope);

console.debug(window.globalScope);
```

¿Y para qué nos sirve el Scope?

Entender bien el concepto de scope nos ayudará a aumentar el nivel de seguridad ya que delimita quienes tienen acceso y quienes no a determinadas partes de nuestro código, también nos facilitará en la detección y disminución de errores, por ende, nuestro código será más robusto.

Tipos de Scope

En Javascript tenemos distintos tipos de Scopes:

Global Scope

Se dice que una variable se encuentra en el scope global cuando está declarada fuera de una función o de un bloque. Vamos a poder acceder a este tipo de variables desde cualquier parte de nuestro código, ya sea dentro o fuera de una función.

El objeto window es un ejemplo de scope global.

Debajo te dejo otro ejemplo de variables globales.

```
var alfajor = "Guaymallen";

// El código escrito en esta parte va a poder acceder a alfajor.

function myFunction() {

  // El código escrito en esta parte también va a poder acceder a alfajor.

}
```

Local Scope

Las variables que definimos dentro de una función son variables locales, es decir se encuentran en el Scope local. Esto significa que este tipo de variables van a vivir únicamente dentro de la función en donde las hayamos declarado y si intentamos accederlas fuera de ella, dichas variables no van a estar definidas.

Esto nos permite decidir si queremos una variable solo para una determinada función.

Un ejemplo de local scope sería el siguiente:

```
// El código escrito en esta parte NO va a poder acceder a la variable alfajor.

function myFunction() {

  var alfajor= "Guaymallen";

  // El código escrito acá si puede.

}
```

Global automática

Si asignamos un valor a una variable que no ha sido declarada, esta se convertirá automáticamente en una variable global.

```
myFunction();

// El código en esta parte va a poder acceder a la variable alfajor.

function myFunction() {

  alfajor = "Jorgito";

}
```

Block Scope

A diferencia del scope local este scope está limitado al bloque de código donde fue definida la variable. Desde ECMAScript 6 contamos con los keyword let y const los cuales nos permiten tener un scope de bloque, esto quiere decir que las variables solo van a vivir dentro del bloque de código correspondiente.

Un breve ejemplo de variables de bloque:

```
if (true) {

    // este bloque if no crea un scope

    // la variable nombre es global por el uso de la keyword 'var'

    var nombre = 'Juan';

    // preferencias se encuentra en el scope local por el uso de la keyword 'let'

    let preferencias = 'Codear';

    // skills también es una scope local por el uso de la keyword 'const'

    const skills = 'Java';

}

console.log(nombre); // logs 'Juan'

console.log(preferencias); // Uncaught ReferenceError: preferencias is not defined
```

```
console.log(skills); // Uncaught ReferenceError: skills is not defined
```

Lexical Scope

El lexical scope significa que en un grupo anidado de funciones, las funciones internas tienen acceso a las variables y otros recursos de su ámbito padre. Esto significa que las funciones hijas están vinculadas léxicamente al contexto de ejecución de sus padres.

Vamos con un ejemplo:

```
function myFunction() {
    var nombre = 'Juan';
    // no podemos acceder a preferencias desde acá
    function parent() {
        // nombre si es accesible desde acá.
        // preferencias en cambio, no es accesible.
        function child() {
            // tambien podemos acceder a nombre desde acá.
            var preferencias = 'Codear';
        }
    }
}
```