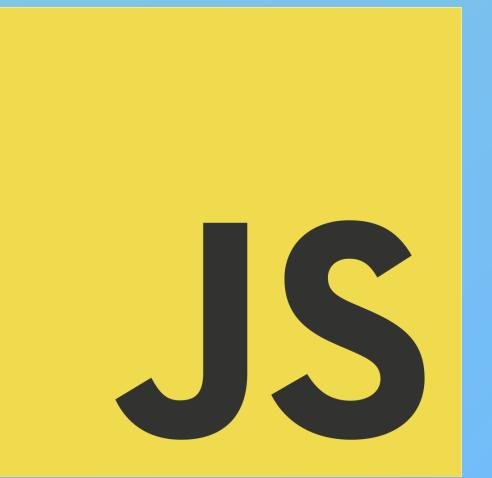


{JS}

Clase 06: Closures, Hoisting y Scope  JS

<índice>

Clousures, Hoisting y Scope

Scope

Hoisting

Clousures

Scope

Scope

Podemos definirlo como
El contexto actual de ejecución

Scope

El contexto en el que los valores y las expresiones son "visibles" o pueden ser referenciados. Si una variable u otra expresión no está "en el Scope-alcance actual", entonces no está disponible para su uso. Los Scope también se pueden superponer en una jerarquía, de modo que los Scope secundarios tengan acceso a los ámbitos primarios, pero no al revés.

Scope

Una función sirve como un cierre en JavaScript y, por lo tanto, crea un ámbito, de modo que (por ejemplo) no se puede acceder a una variable definida exclusivamente dentro de la función desde fuera de la función o dentro de otras funciones. Por ejemplo, lo siguiente no es válido:

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
```

Scope

Sin embargo, el siguiente código es válido debido a que la variable se declara fuera de la función, lo que la hace global:

```
var x = "función externa declarada";  
  
exampleFunction();  
  
function exampleFunction() {  
    console.log("funcion interna");  
    console.log(x);  
}  
  
console.log("funcion externa");  
console.log(x);
```

Scope

¿Qué es la cadena de scopes?

Scope

¿Una variable puede estar
definida dos veces?

Tipos de scopes

- Scope global
- Scope local
 - scope de función
 - scope de bloque (hoisting)

Scope

¿Mejor usar scope global
o scope local?

Siempre se recomienda usar el scope más
reducido posible. Ahorramos memoria

Scope

Hagamos este ejercicio

```
1 var fruta = 'manzana';
2
3 function comer() {
4     var fruta = 'banana';
5
6     function lavar() {
7         console.log(`Lavando ${fruta}`);
8     }
9
10    lavar();
11    console.log(`Comiendo ${fruta}`);
12}
13
14 comer();
```

Scope

```
var fruta = 'manzana';

function comer() {
    var fruta = 'banana';

    function lavar() {
        console.log(`Lavando ${window.fruta}`);
    }

    lavar();
    console.log(`Comiendo ${fruta}`);
}

comer();
```

¿Cómo
accedemos a las
variables
globales?

Scope

¿Cómo vemos los
diferentes scopes en
nuestro navegador?

Scope

The screenshot shows the Chrome DevTools interface with the "Sources" tab selected. A file named "prueba.js" is open, containing the following code:

```
1 var fruta = 'manzana';
2
3 function comer() {
4     var fruta = 'banana'; fruta = "banana"
5
6     function lavar() { lavar = f lavar()
7         console.log(`Lavando ${window.fruta}`)
8     }
9
10    lavar(); lavar = f lavar()
11    console.log(`Comiendo ${fruta}`)
12}
13
14 comer();
15
16
17
18
19
20
```

The code highlights several variable assignments and function definitions. The line `fruta = "banana"` is highlighted in brown. The line `lavar = f lavar()` is also highlighted in brown. The line `console.log(`Comiendo ${fruta}`)` is highlighted in blue.

In the bottom right corner of the DevTools window, there is a yellow rounded rectangle containing the text "Debugger paused".

The right sidebar of the DevTools shows the following sections:

- Threads
- Watch
- Breakpoints
- Pause on uncaught exceptions
- Pause on caught exceptions
- Scope
 - Local
 - this: Window
 - fruta: "banana"
 - lavar: f lavar()
 - Global
 - 0: Window {window: Window, self: Wi
 - JSCompiler_renameProperty: f (t,e)
 - alert: f alert()
 - atob: f atob()
 - blur: f blur()
 - btoa: f btoa()
 - caches: CacheStorage {}

Hoisting

Hoisting

Característica rara y poco intuitiva

Hoisting

¿Qué hace esto?

```
1 console.log(firstName);
2 var firstName = 'Bienve';
```

Hoisting

Hoisting = Elevación

Cada vez que declaremos variables con var, estas se elevarán al inicio de su scope.

Es como si las declaráramos siempre al principio.

Pero solo las declaraciones, no las asignaciones.

Hoisting

Solo las variables declaradas
con var sufren de hoisting

Hoisting

Ojo, las funciones también. Depende de como sean declaradas.

¿Os acordáis de las formas de definir una función?

Declarar funciones

Funciones por declaración

Probablemente, la forma más popular de estas tres, y a la que estaremos acostumbrados si venimos de otros lenguajes de programación, es la primera, a la creación de funciones por declaración.

Declarar funciones

```
function saludar() {  
    return "Hola";  
}  
  
saludar(); // 'Hola'  
typeof saludar; // 'function'
```

Declarar funciones

```
1  saludar();  
2  
3  function · saludar() · {  
4      ··· console.log( 'Hola' );  
5  }  
6
```

Declarar funciones

Funciones por expresión

Sin embargo, en Javascript es muy habitual encontrarse códigos donde los programadores «guardan funciones» dentro de variables, para posteriormente «ejecutar dichas variables»

Declarar funciones

```
// El segundo "saludar" (nombre de la función) se suele omitir: es redundante
const saludar = function saludar() {
    return "Hola";
};

saludo(); // 'Hola'
```

Declarar funciones

```
1  saludo();  
2  
3  const saludo = function () {  
4      console.log('Hola')  
5  }
```

Hoisting

¿Qué sufre de hoisting?

`var`

funciones por declaración

De las cosas más raras de JS

Reaso

	<i>var</i>	<i>let</i>	<i>const</i>
<i>compatibilidad</i>	✓	BABEL	BABEL
<i>scope (ámbito)</i>	función	bloque	bloque
<i>re-asignación</i>	✓	✓	✗
<i>re-declaración</i>	✓	✗	✗
<i>declaración sin valor inicial</i>	✓	✓	✗
<i>propiedad del obj. global</i>	✓	✗	✗
<i>hoisting</i>	declaración	TDZ	TDZ

Closures

Clousures

Necesitamos tres cosas

1. Función anidada
2. Variable descrita en la función padre que sea utilizada por la función anidada
3. Invocar a la función interna desde otro scope

Clousures

```
1  function · crearContador( ) · {  
2      · · let · contador · = · 0;  
3  
4      · · return · function · incrementar( ) · {  
5          · · · | · contador · += · 1;  
6          · · · return · contador;  
7      · · }  
8  }  
9  
10 const · c1 · = · crearContador( );  
11
```

Clousures

Vamos a incrementar su funcionalidad
decrementar
obtener valor

Clousures

Vamos a incrementar su funcionalidad
contador personalizado

<Despedida>

Email

bienvenidosaez@gmail.com

Instagram

@bienvenidosaez

Youtube

youtube.com/bienvenidosaez