

Tema 2 - JavaScript

JS



Anexo 1: var vs let

Desarrollo web en entorno cliente
IES Pere Maria Orts I Bosch



Index

| | |
|---|---|
| Anexo bloque 1..... | 3 |
| “hoisting” de variables (declaración de variables con var)..... | 3 |
| var vs let..... | 3 |
| Referencia vs Copia..... | 4 |

Anexosemana1

“hoisting” de variables (declaración de variables con var)

La palabra “hoisting” define un particular comportamiento en la declaración de variables con la palabra reservada `var` en JS (por esto se recomienda usar `let` desde ES2015). Antes de ejecutar el código, JS lo analiza y realiza dos cosas:

- Carga la declaración de funciones en memoria (por tanto, pueden ser accesibles desde cualquier posición).
- Mueve la declaración de variables al principio de las funciones (si no es local a una función, la mueve al principio del programa/bloque principal).

Vamos a ver un ejemplo:

```
function printHello() {  
  console.log(hello);  
  var hello = "Hello World";  
}  
  
printHello(); // Esto imprimirá undefined
```

Debería imprimir un error porque estamos intentando imprimir la variable `hello` antes de que exista. En este caso imprime “undefined”, porque en el primer paso ha transformado el código internamente a este:

```
function printHello() {  
  var hello = undefined;  
  console.log(hello);  
  hello = "Hello World";  
}  
  
printHello();
```

var vs let

Usar `let` es la manera actualmente recomendada de declarar variables. La mayor ventaja respecto a `var` es que evita el *hoisting*. Si accedemos a una variable antes de ser declarada, no será *undefined*, sino que lanzará un error (comportamiento lógico).

```
'use strict';  
console.log(number); // Imprime undefined (hoisting)  
var number = 14;
```

```
'use strict';  
console.log(number); → Uncaught ReferenceError: number is not defined  
let number = 14;
```

Otra ventaja de `let` es que la variable es local al bloque donde se declara (no a

toda la función).

```
'use strict';
for(var i = 0; i < 10; i++) {
  console.log(i);
}
console.log(i); // Imprime 10 (i existe fuera del bucle)
```

```
'use strict';
for(let i = 0; i < 10; i++) {
  console.log(i);
}
console.log(i); → Uncaught ReferenceError: i is not defined
```

Podemos incluso crear un bloque artificialmente para crear un nuevo ámbito.

```
'use strict';
let number = 10;

{
  let number = 200; // number es local a este bloque
}

console.log(number); // Imprime 10
```

Vamos a ver un ejemplo que refleja un problema clásico de JavaScript cuando usamos var. En este caso crearemos funciones dentro del bucle que accedieran al contador del mismo para imprimirlo por consola:

```
'use strict';
let functions = [];
for(var i = 0; i < 10; i++) {
  functions.push(function() {
    console.log(i);
  });
}
functions[0](); // Imprime 10, (valor actual de i, y no el valor que tenía al crear la función)
functions[1](); // También imprime 10
```

Usando let, funciona como debería:

```
'use strict';
let functions = [];
for(let i = 0; i < 10; i++) {
  functions.push(function() {
    console.log(i);
  });
}
functions[0](); // Imprime 0 (en cada iteración se crea una nueva variable local i, que es la que usa)
functions[1](); // Imprime 1
```

Anexo 1: var vs let.

Programación con JavaScript
Cefire 2021
Autor: Arturo Bernal Mayordomo