

Tema 3. Sistemas Web

Diferencias entre cláusulas var, let y const

1. ¿Qué obtendremos por pantalla?

```
var tester = "hey hi";
```

```
function newFunction() {  
  var hello = "hello";  
}
```

```
console.log(hello);
```

Un error: Uncaught ReferenceError: hello is not defined

La razón es que el alcance de la variable hello es local a la función newFunction. Fuera de ella, no existe.

2. ¿Es posible hacer esto?

```
var greeter = "hey hi";  
var greeter = "say Hello instead";
```

Sí, no obtendremos ningún error. La segunda declaración de variable con la cláusula var será ignorada (la declaración, no la asignación de valor que se ejecutará con normalidad, de tal forma que greeter tome el valor "say Hello Instead")

2.1 ¿Y esto otro?

```
let greet = "hey hi";  
let greet = "say Hello instead";
```

Obtendremos un error por intentar redeclarar la variable greet. La cláusula let protege al programador de sí mismo (por descuido ha intentando volver a declarar una variable dentro del mismo bloque) arrojando un error.

OJO: si pruebas a ejecutar las líneas anteriores desde la consola de las DevTools NO obtendrás ningún error. ¿Cómo es posible? Es una característica introducida por Chrome 80 (Julio de 2020) para facilitar el prototipado y la experimentación con código JavaScript en la consola. Ver: <https://umaar.com/dev-tips/214-redeclare-let-console/>

¿Cómo probarlo entonces? Tres opciones:

* en la consola de Firefox (donde se ciñe al estándar y no permite excepciones)

```
Eskaerarik ez

Iragazi irteera

>> let greet = "hey hi";
← undefined

>> let greet = "say Hello instead";
! ▶ Uncaught SyntaxError: redeclaration of let greet
    <anonymous> debugger eval code:1
    [Argibide gehiago]

>> |
```

- * creando una página web completa con código JavaScript y cargándola en el navegador
- * a través del intérprete de NodeJS

```
→ ejercicios node
Welcome to Node.js v18.6.0.
Type ".help" for more information.
> let greet = "hey hi";
undefined
> let greet = "say Hello instead";
Uncaught SyntaxError: Identifier 'greet' has already been declared
> greet
'hey hi'
> █
```

3. ¿Es posible hacer esto sin obtener errores? Y en caso afirmativo, ¿cuál es el nombre de esta característica de JavaScript? (Pista: busca la siguiente definición en inglés: “refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code.”)

```
console.log (greeter);
var greeter = "say hello";
```

Es posible, y no se arrojan errores. Pero no es recomendable. La característica es conocida como **"hoisting"**. El intérprete JavaScript, al detectar que estamos intentado usar una variable no declarada aún, busca dicha declaración (con var) en las siguientes instrucciones del bloque. Si la encuentra, la "sube" para que esté al alcance al intentar usarla. De ahí que el código anterior, realmente sea ejecutado así:

```
var greeter;  
console.log (greeter);  
greeter = "say hello";
```

4. ¿Y esto otro?

```
console.log(greeter2);  
let greeter2 = "say hello";
```

Arrojará un error (greeter2 no está definida) El hoisting NO ocurre con las variables declaradas con cláusula let.

5. ¿Qué se imprime al final de este código?

```
var ind = 0;  
for(var ind=3; ind<10; ind++); // bucle sin cuerpo asociado  
  
console.log(ind);
```

Imprimirá 10 por pantalla. Recordemos que el intento de redeclarar ind será omitido. Por tanto, el for estará reutilizando la variable ind del bloque superior.

6. ¿Qué se imprime al final de este código?

```
let x = 0;  
for(let x=3; x<10; x++); // bucle sin cuerpo asociado  
console.log(x);
```

Imprimirá 0 por pantalla. Ese for realmente es equivalente a este:

```
for(let x=3; x<10; x++){ }; donde el bloque del cuerpo esta vacío.
```

El alcance de las declaraciones dentro de la inicialización del for(let x=3....) es el bloque for { } (en este caso vacío) Dentro del bloque for el intérprete JavaScript debe elegir qué variable usar: el x de nivel superior o el x local. Siempre tendrá preferencia (shadowing de variables) la variable local.

Una vez terminado el for, la variable local x deja de existir (y la única x que existe es la que se declaró en el bloque principal del programa, donde x valía 0. De ahí que console.log(x) imprima 0 en pantalla.

6.1 ¿Y tras este código?

```
let y = 0;  
for(y=3; y<10; y++); // bucle sin cuerpo asociado. Do nothing  
  
console.log(y);
```

Imprimirá 10. Existe una única variable y en todo el código.

7. ¿Se puede hacer esto sin recibir errores?

```
let saludo = "Hola";  
let saludo = "Adiós";
```

No, recibiremos un error por intentar redeclarar la variable saludo (salvo que lo ejecutemos en la consola de las DevTools de Chrome. A partir de Chrome 80, la consola permite esta redeclaración, como se puede ver en este enlace: <https://umaar.com/dev-tips/214-redeclare-let-console/>)

8. ¿Y esto?

```
var saludo2 = "Hola";  
var saludo2 = "Adiós";
```

Este código es correcto. La segunda declaración de saludo2 será omitida.

9. ¿Hay algún error en el siguiente código? ¿qué obtenemos por pantalla?

```
const AGUR="agur!"  
AGUR="adiós!";
```

Obtendremos un error por intentar asignar un valor a una constante ya inicializada.

10. ¿Hay algún error? ¿qué obtenemos por pantalla?

```
const AGUR="agur!";  
if (true){  
  const AGUR="adiós!";  
}  
console.log(AGUR);
```

Ahí, la intuición nos dice que la misma constante no la podemos inicializar dos veces. Pero cuidado, como ocurría en el ejercicio 6, el alcance de un "CONST" es el bloque correspondiente. Aquí declaramos e inicializamos la primera constante AGUR en el bloque del programa principal. Luego, dentro del bloque if () la constante AGUR externa es visible, podemos usarla, pero también podemos declarar e inicializar otra constante (¡con el mismo nombre!). El intérprete tiene que decidir cuál usar : la constante local al bloque (AGUR="adiós!"). Al finalizar el bloque if() desaparece la constante AGUR local (pero se mantiene la constante externa). Por lo tanto, en este último console.log se escribirá "agur!".

11. ¿Hay algún error? ¿qué obtenemos por pantalla?

```
const diccionario = {  
  hola : "kaixo",  
  casa : "etxea"  
};  
diccionario.hola = "iepa!";  
console.log(diccionario.hola);
```

11.1 Y si ahora hacemos lo siguiente, ¿habrá algún error?

```
diccionario = []
```

El valor de los atributos del objeto diccionario pueden cambiarse, pero al ser una constante, no podremos asignar a diccionario otro objeto distinto.

Es decir:

diccionario.hola = "iepa!" funcionará, pero "diccionario = []" no (se le está intentando asignar un nuevo objeto)

Ejercicios sobre arrays

1. Sea el siguiente código que genera 3 objetos de tipo Punto y los inserta en un array:

```
function Point(x,y){  
    this.x = x;  
    this.y = y;  
}
```

```
let puntos1 = [new Point(5,0), new Point(11,1), new Point(2,2)];
```

Implementa un script que elimine del array aquellos puntos cuya coordenada x > 10.

Pista: el método splice() te será de utilidad...

Prueba tu solución con estos otros casos:

```
let puntos2 = [new Point(5,0), new Point(11,1), new Point(15,1), new Point(2,2)];
```

```
let puntos3 = [new Point(5,0), new Point(4,1), new Point(5,2), new Point(6,0), new Point(11,1),  
new Point(15,2)];
```

Si intentamos una solución ingenua, como la siguiente, funcionará solamente cuando no haya dos puntos consecutivos en el array que cumplan la condición:

```
function borrarMayores(array){  
    for (let i=0; i < array.length; i++)  
        if (array[i].x > 10)  
            array.splice(i, 1)  
}
```

Pero fallará por ejemplo con una entrada como la siguiente:

```
[ Point(1,3), Point(12, 4), Point(14, 2), Point(2,3) ]
```

¿Por qué?

El for pasará por el primer elemnto del array, verá que no cumple la condición, incrementará el valor de i e irá a analizar el siguiente punto.

Estaremos en esta situación:

```
[ Point(1,3), Point(12, 4), Point(14, 2), Point(2,3) ]  
          ^-----
```

En esta iteración, Point(12,4) cumple la condición, por lo que se eliminará del array (es decir, los elementos se han desplazado hacia la izquierda)

```
[ Point(1,3), Point(14, 2), Point(2,3) ]  
          ^-----
```

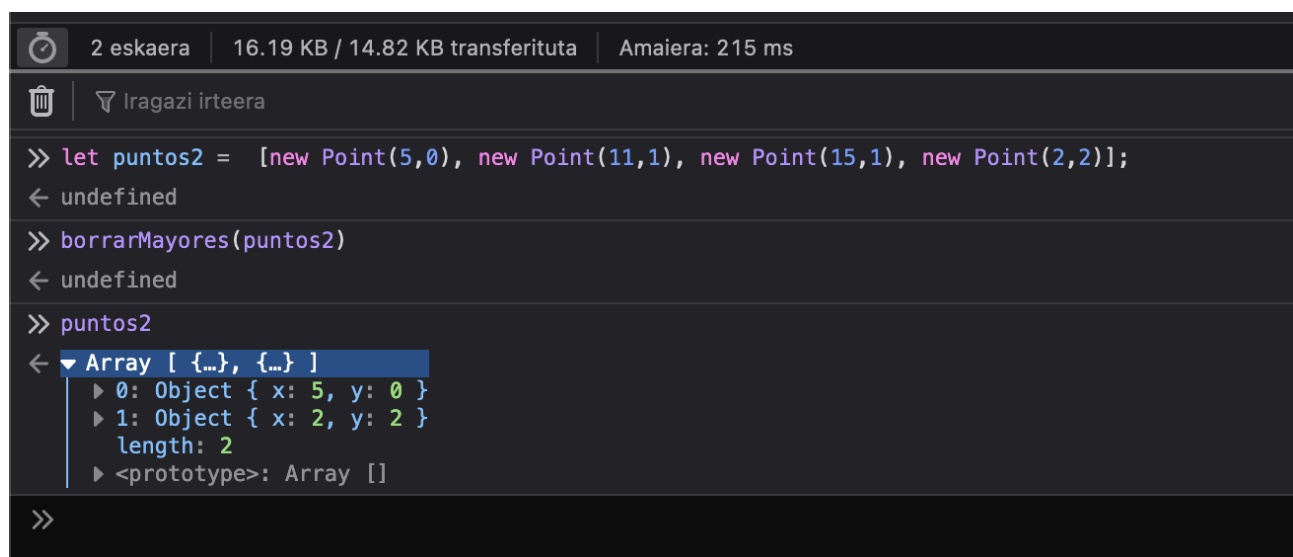
Ahora, el for vuelve a incrementar i (i++) y pasa a evaluar el siguiente elemento:

```
[ Point(1,3), Point(14, 2), Point(2,3) ]  
          ^-----
```

¡Error! Acabamos de dejarnos el punto 14,2 sin tratar :)

El problema es que al borrar un elemento, los que están a su derecha se desplazan y nuestra variable i deja de controlar qué elemento borrar a continuación. La solución es sencilla: en lugar de recorrer el array de izquierda a derecha, podemos recorrerlo de derecha a izquierda, evitando el error (al eliminar un elemento, los elementos a su izquierda no cambian su posición):

```
function borrarMayores(array){  
  for (let i= array.length - 1; i>=0; i--)  
    if (array[i].x > 10)  
      array.splice(i, 1)  
}
```



```
2 eskaera | 16.19 KB / 14.82 KB transferituta | Amaiera: 215 ms  
Iragazi irteera  
>> let puntos2 = [new Point(5,0), new Point(11,1), new Point(15,1), new Point(2,2)];  
< undefined  
>> borrarMayores(puntos2)  
< undefined  
>> puntos2  
< ▼ Array [ {...}, {...} ]  
  ▶ 0: Object { x: 5, y: 0 }  
  ▶ 1: Object { x: 2, y: 2 }  
    length: 2  
  ▶ <prototype>: Array []  
>>
```

2. Implementa un script para ordenar los puntos del array en función de su coordenada x, de menor a mayor. Pista: el método `sort()` te será de utilidad...

La función `sort()` del objeto¹ `Array`

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort permite ordenar un array.

```
const array1 = [1, 30, 4, 21, 100000];
array1.sort();
console.log(array1);
// expected output: Array [1, 100000, 21, 30, 4]
```

Ojo, como vemos, por defecto, `sort` convierte los elementos a string y los ordena en orden alfabético ascendente².

`sort()` admite como parámetro una función de ordenación (similar a un comparador Java). Para ordenar los puntos de nuestro array en orden ascendente, podemos crear el siguiente comparador:

```
function comparador(a, b){
    return a.x - b.x;
}
```

Devolverá un valor positivo si $a > b$; negativo si $a < b$; 0 si son iguales.

Ahora podremos ordenar los puntos en función de su atributo `x`:

```
let puntos2 = [new Point(5,0), new Point(11,1), new Point(15,1), new Point(2,2)];
puntos2.sort(comparador);
```

```
>> puntos2 = [new Point(5,0), new Point(11,1), new Point(15,1), new Point(2,2)];
< ▶ Array(4) [ {...}, {...}, {...}, {...} ]

>> puntos2.sort(comparador);
< ▼ Array(4) [ {...}, {...}, {...}, {...} ]
  ▶ 0: Object { x: 2, y: 2 }
  ▶ 1: Object { x: 5, y: 0 }
  ▶ 2: Object { x: 11, y: 1 }
  ▶ 3: Object { x: 15, y: 1 }
    length: 4
  ▶ <prototype>: Array []

>>
```

¹ En Java sería clase `Array`, pero en JavaScript todo son objetos (y `Array` es un built-in object)

² Realmente comparando la secuencia de valores UTF-16, pero para este caso puede simplificarse diciendo que el orden es alfabético ascendente