



Nuestro compromiso es con el *futuro*.

Introducción a la programación

Temario

- Diferencia entre var y **let**
- **Desestructuración** (destructuring)
- Operador **spread**

var y let

Tanto var como let **crean una nueva variable**. La principal diferencia entre ellos **es que** una variable creada por medio de la palabra reservada **let sólo tiene injerencia dentro del scope donde fue creada**, y no por fuera, como sí tiene var.

```
function prueba(){  
  var a = "Juan";  
  var b = "Tahiel";  
  
  if(true){  
    var a = "Pedro";  
    let b = "Mica";  
  }  
  
  console.log(a) // Pedro  
  console.log(b) // Tahiel  
}  
prueba();
```

var y let

La variable `i` no puede ser mostrada: JavaScript la marca como que **no está definida** porque, al haber sido creada con **let**, sólo existe dentro del scope del ciclo `for`.

```
function pruebaFor(){
  var array = [1, 2, 3];
  for(let i = 0; i < array.length; i++) {
    console.log(array[i])
  }
  console.log(i) // ReferenceError: i is not
defined
}
pruebaFor();
```

Destructuring

Las estructuras, como los arreglos y los objetos, se pueden **desestructurar** para poder seleccionar valores utilizando patrones de match o coincidencia.

```
var [a, b, c] = [1, 2, 3];  
  
console.log(a) // 1  
console.log(b) // 2  
console.log(c) // 3
```

Destructuring

La desestructuración también admite la **asignación por default**: podemos establecer el valor de **a**, en caso de que el arreglo que le pasemos esté provisoriamente vacío, evitando el **soft-fail**.

En caso de que el arreglo sí tenga un valor para asignarle, **se omite** el valor por default (también sirve para funciones).

```
// asignación por default
var [a = 1] = [];
console.log(a) // 1
```

```
var [a = 1] = [5];
console.log(a) // 5
```

Destructuring

Esto nos **ahorra tiempo**: de otra forma, deberíamos recorrer el arreglo y a cada posición asignarle los valores correspondientes.

```
var [nombre, edad] = ["Marcos", 30];  
console.log(nombre) // Marcos  
console.log(edad) // 30
```


Destructuring

Esto también nos permite **saltarnos elementos**.

Al nombre y al apellido se les asigna un valor, pero como el dato de la edad no nos interesa, dejamos **un espacio en blanco** en el arreglo de la izquierda.

```
var [nombre, , apellido] = ["Marcos", 30, "Rausch"];  
console.log(nombre) // Marcos  
console.log(apellido) // Rausch
```

Destructuring

¡También podemos aplicar destructuración sobre **objetos**!

Esto es extremadamente útil, puesto que nos permite obtener **únicamente las propiedades que nosotros le declaremos** a un objeto.

```
var { nombre, apellido } = {nombre: "Juan", apellido: "Aguirre" };  
console.log(nombre) // Juan  
console.log(apellido) // Aguirre
```

Destructuring

Como vemos, **sólo nos trae** el nombre, que es **lo que le pedimos** por medio de la función.

¿Qué pasaría si pedimos el nombre de una propiedad **que no existe**, o lo hiciéramos con un error de tipo?

```
const objetoGigante = {
  nombre: 'Juan',
  apellido: 'Aguirre',
  edad: 25,
  hobbies: ['fútbol', 'crossfit'],
  trabajo: 'desarrollador',
  nacionalidad: 'argentina',
  mascotas: true,
  altura: 1.80
}

function desestructurar({ nombre }) {
  console.log(nombre)
}

desestructurar(objetoGigante); // Juan
```

Spread operator...

Dependiendo de cómo lo invoquemos, hace una de dos cosas:

1. **Separa** elementos particulares de un arreglo
2. **Une** elementos separados dentro de un arreglo.

```
function unir(...y) {
  console.log(y);
}
unir(3, 4, "hola");
```

```
function desunir(x, y, z) {
  console.log(x + y + z);
}
desunir(...[1, 2, 3]); // 6
```



JS

¡Desafíos!

Muchas gracias!



ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar