Repaso Intensivo JS

Javier Ribal del Río

2025-12-12

Table of contents

Contenido

Repaso intensivo de JavaScript, tiene como objetivo dar una introducción del lenguaje, a usuarios que ya tengan experiencia previa programando.

Descargar archivo de código (click derecho guardar enlace como)

---

1 Variables y tipos básicos

1.1 Declaración de variables

En JavaScript moderno se utilizan **`let`** y **`const`** (evitar `var`).

- `let`: permite reasignación
- `const`: no permite reasignación
- `const` **NO** significa inmutable
- JavaScript es de **tipado dinámico**
- El tipo depende del valor, no de la variable

Tipos primitivos:

- `string`
- `number`
- `boolean`
- `null`
- `undefined`

```
let edad = 12;         // number
let altura = 1.8;      // number
let nombre = "Javier"; // string
let casado = false;    // boolean

let ordenador = null; // ausencia intencionada de valor
let direccion;        // undefined
```

1.2 Uso de `const`

```
const dni = "1235678L";
dni = "23"; // Error
```

1.2.1 `const` no significa inmutable

```
const dni = "12345678L";
dni = "87654321X"; // Error
```

Pero:

```
const arr = [1, 2, 3];
arr.push(4); // permitido
```

1.3 Tipado dinámico

- JavaScript es de **tipado dinámico**
- El tipo depende del valor, no de la variable
- Una variable puede cambiar de tipo durante la ejecución

```
let x = "hola";
x = 42;      // válido
x = true;    // válido
```

Este comportamiento se conoce como ***shadowing*** o cambio dinámico de tipo. En el ejemplo anterior solo existe una variable X cuyo tipo pasa de `Number` a `boolean`

2 `null` vs `undefined`

- `undefined`: variable declarada pero sin valor
- `null`: ausencia intencionada de valor

JavaScript distingue entre:

> "todavía no hay valor" y "no hay valor"

```
console.log(ordenador); // null
console.log(direccion); // undefined
```

3 Aritmética y operadores

3.1 Operadores aritméticos

```
let a = 12, b = 4;

a + b
a - b
a * b
a / b
a % b
```

```
a = a + b;
a += b;
a -= b;
a *= b;
```

3.1.1 Potencias

```
a**b  // Power
```

Incrementos:

```
b--; b++;
--b; ++b;
```

3.2 Strings y concatenación

```
a = "hola ";
b = "mundo";
a + b;
```

3.3 Comparación

- == compara valor (**evitar**)
- === compara valor y tipo (**usar**)

3.4 Operadores lógicos

- && AND
- || OR
- ! NOT

3.5 Comparaciones

- == compara solo valor (**evitar**)
- === compara valor y tipo (**usar siempre**, aunque sea innecesario)

```
5 == "5";   // true
5 === "5";  // false
```

3.6 Operadores lógicos

- && AND
- || OR
- ! NOT

```
(a > 0) && (b < 10)
```

4 Estructuras de control

4.1 Condicionales

```
if (a === b) {

} else if (a > b) {

} else {

}
```

4.2 Bucles

4.2.1 `while`

```
let x = 20;
while (x < 50) {
  x += 10;
}
```

4.2.2 `for`

```
for (let i = 0; i < 10; i++) {
  // iteración
}
```

5 Funciones

- Encapsulan lógica reutilizable
- Pueden recibir parámetros
- Pueden devolver valores
- Son ciudadanos de primera clase

5.1 Función clásica

```
function sumar2(num, num2 = 2) {
  num += num2;
  return num;
}
```

Paso de variables por **valor**:

```
let a = 7;
let b = sumar2(a, a); // 14
a; // 7
```

5.2 Funciones flecha y orden superior

```
const concatenarHola = input => input + " Hola";
concatenarHola("sdf");
```

```
function ejecuta(fun) {
  fun();
}

ejecuta(() => { console.log("hola"); });
```

5.3 Funciones propias de strings

```
let a = "jsadfsadf ";
a.trim();   // Suprimir espacios adicionales
a.split('a');    // Divide el string en un array c
a.length;
```

5.4 Funciones flecha

```
const cuadrado = n => n ** 2;
```

Forma extendida:

```
const cuadrado = (n) => {
  return n ** 2;
};
```

5.5 Funciones como ciudadanos de primera clase

Las funciones pueden:

- Asignarse a variables
- Pasarse como argumentos
- Devolverse como resultado

```
function ejecutar(f) {
  f();
}

ejecutar(() => console.log("Hola"));
```

6 Arrays (base)

- Lista ordenada
- Índices empiezan en 0
- Propiedad `length`

```
let cajon = [8, "hola", true, () => { return 7 }];
```

6.1 Acceso

```
cajon[0]  // 8
cajon[2] = "rino";       // modificamos la entrada 2
```

6.2 Adiciones

```
cajon.push(false);
cajon.unshift(0);
```

6.3 Eliminaciones

```
cajon.pop();
cajon.shift();
```

```
cajon.length;
```

7 Objetos

Un objeto representa una entidad mediante estructura **clave–valor**.

```
let gente = { pepe: 7, juanes: 8, andreas: 10, fran: [4, 2] };
```

7.1 Modificación

```
gente.andreas = 32;
gente.pepe++;
```

7.2 Acceso dinámico

```
let nombre = "pablo";

gente.nombre;      // undefined
gente[nombre] = 2; // añade propiedad
```

8 Referencias vs valores

8.1 Copia por valor

```
let a = 5;
let b = a;
b++;
```

`a` no cambia.

8.2 Copia por referencia

```
let e3 = ["juan", "pepe"];
let d = e3;

d.push("andrés");
```

Ambas variables apuntan al mismo array.

```
const poblacion = gente;
poblacion["julio"] = 8;
```

La copia por refencia se aplica a objetos y arrays

```
let x = ["juan", "pepe"];
let y = x;

y.push("andrés"); // Se añade en x e y pues son el mismo objeto
```

Ambas variables apuntan al mismo objeto.

9 Spread operator (...)

- Expande arrays u objetos
- Permite copiar y combinar
- Copia superficial

9.1 Arrays

Javier es añadido a `arr2`, pero no a `arr`.

```
let arr = ["Alice", "Bob", "Kevin"];
let arr2 = [...arr, "Javier"];
```

9.2 Objetos

Ocurre lo mismo con los objetos

```
const sociedad = { ...poblacion, julia: 12 };
```

10 Funciones de array (JS moderno)

Vamos a trabajar con un **ejemplo completo**, típico en programación funcional.

10.1 Datos de partida

```
let y = [2, 4, 6, 7];
```

10.2 Función `cuadrado`

Definimos una función que, dado un número, devuelve su cuadrado:

```
const cuadrado = n => n ** 2;
```

Es equivalente a:

```
const cuadrado = (n) => {
  return n ** 2;
};
```

10.3 `map`: transformar elementos

Aplicamos `map` para obtener un nuevo array con los cuadrados:

```
const y2 = y.map(cuadrado);
```

Resultado:

```
[4, 16, 36, 49]
```

10.4 `filter`: seleccionar elementos

Definimos una función que comprueba si un número es par:

```
const even = n => n % 2 === 0;
```

La usamos con `filter`:

```
const yEven = y.filter(even);
```

10.5 Encadenamiento (estilo declarativo)

Podemos encadenar ambas operaciones:

```
y
  .map(n => n ** 2)
  .filter(n => n % 2 === 0);
```

Este estilo es **declarativo**: describimos *qué* queremos hacer con los datos, no *cómo* recorrerlos.