

# Introducción a JS

Eduardo Aguilar  



**KEEPCODING**  
Tech School

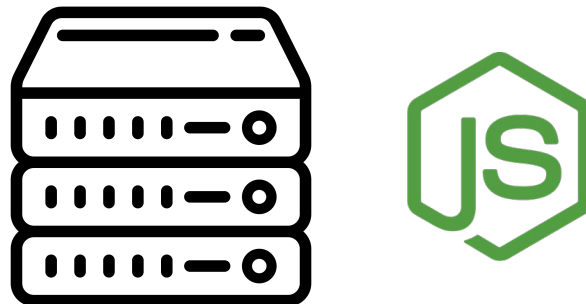


# Roadmap



1. Introducción a JS
2. Fundamentos de JS
  - a. Variables
  - b. Condicionales
  - c. Funciones
  - d. Bucles
3. Práctica

# Introducción a JS





# Fundamentos de JS - Variables

Definición de variables

var - vieja

let - lectura/escritura

const - lectura

# Fundamentos de JS - Variables

## Nombrado de variables

Existen dos limitaciones al nombrar variables en JavaScript:

- El nombre únicamente puede incluir letras, dígitos, o los símbolos \$ y \_.
- El primer carácter no puede ser un dígito.



# Fundamentos de JS - Variables

## Particularidades

- Son case sensitive
- Ojo con los nombres reservados

## Ejercicios

# Fundamentos de JS - Tipos de datos

En javaScript existen 8 tipos de datos básicos

- Number
- BigInt
- String
- Boolean
- null
- undefined
- Symbol
- Object

# Fundamentos de JS - Operadores

Operaciones soportadas:

- Suma +
- Resta -
- Multiplicación \*
- División /
- Resto %
- Exponenciación \*\*



# Fundamentos de JS - Comparaciones

- Mayor/menor que:  $a > b$  |  $a < b$
- Mayor/menor o igual que:  $a \geq b$  |  $a \leq b$
- Igual:  $a == b$  (ojo con la asignación!)
- Distinto:  $a != b$
- Igualdad/desigualdad estricta:  $===$  |  $!==$

## Ejercicios

# Fundamentos de JS - Sentencia If

La sentencia `if(...)` evalúa la condición de los paréntesis. Si el resultado es `true` ejecuta un bloque de código.

Si queremos ejecutar cierto código cuando la condición no se cumple, podemos concatenar el `if` con la sentencia `else`.

También podemos concatenar `if`'s con la sentencia `else if`.

## Ejercicios

# Fundamentos de JS - Operador ternario

En el momento en el que necesitamos asignar un valor a una variable dependiendo de una condición, es muy probable que el operador ternario '?' nos facilite la labor.

# Fundamentos de JS - Operadores lógicos

Existen cuatro operadores lógicos en JS:

- OR ||
- AND &&
- NOT !
- Nullish Coalescing ??

## Ejemplos

# Fundamentos de JS - While y for

Frecuentemente necesitaremos realizar operaciones de forma repetida, o recorrer una lista para hacer ciertas acciones. Para ello utilizaremos *while*, *do while* y *for*.

- While: mientras una condición sea verdadera, ejecutaremos un bloque de código.
- Do while: ejecutaremos un bloque de código y seguiremos ejecutando dicho código mientras una condición sea verdadera.
- for: repetiremos bloques de código a partir de un índice, una condición y un incremento del índice.

# Fundamentos de JS - While y for

```
while (condition) {  
  // code loop  
}
```

```
do {  
  // code loop  
} while (condition);
```

```
for (begin; condition; step) {  
  // ... code loop  
}
```

[Ejemplos](#)

# Fundamentos de JS - Switch

La sentencia switch nos va a permitir ejecutar distintos bloques de código según el valor de una variable.

```
switch(x) {  
  case 'value1':  
    [break]  
  case 'value2': // if (x === 'value2')  
    [break]  
  default:  
    ....  
}
```

[Ejemplo](#)

# Fundamentos de JS - Funciones

Una función es un bloque de código con una responsabilidad única que puede ejecutarse varias veces.

Para usar funciones, tenemos que hacer dos cosas:

- Declararlas
- Ejecutarlas



# Fundamentos de JS - Funciones

```
function printToConsole(value) {  
    console.log('debug: ', value);  
}
```

```
printToConsole(5);  
printToConsole('Hola!');
```

## Ejemplos

# Fundamentos de JS - Funciones

- Los valores pasados a una función son copiados como variables locales.
- Las funciones pueden acceder a variables de fuera, pero no al revés.
- Las funciones pueden retornar valores. Si no lo hacemos, el resultado es undefined.
- Debemos procurar que las funciones sólo utilicen variables locales a esa función, y no de fuera.

# Fundamentos de JS - Funciones flecha

Las funciones flecha son otra forma que tenemos en JavaScript para definir y utilizar funciones de una forma más ligera.

```
const printToConsole = value => console.log('debug: ', value)
```

## Ejemplos

# Fundamentos de JS - Objetos

```
{  
  key1: value,  
  key2: value,  
}
```

# Fundamentos de JS - Objetos

Formas de crear objetos:

```
let user = new Object();
```

```
let user = {}
```

# Fundamentos de JS - Objetos

Declarar, inicializar e interactuar con objetos:

```
let user = {  
  name: "John",  
  age: 30,  
};  
  
user.name // John  
user.age  // 30  
  
user['name'] // John  
user['age']  // 30  
  
delete user.name
```

[Ejemplos](#)

# Fundamentos de JS - Objetos

Referencias y copias de los objetos

```
let message = "Hello!";
```

```
let phrase = message;
```

```
let user = { name: "John" };
```

```
let admin = user;
```

# Fundamentos de JS - Objetos

¿Y qué problema hay si copio la referencia?

```
const obj1 = { name: 'John' };
```

```
const obj2 = obj1;
```

```
obj2.name = 'Edu'
```

```
console.log(obj1.name) // Edu
```



# Fundamentos de JS - Objetos

A ver, ¿entonces no podemos copiar un objeto en JS?

Sí, utilizando por ejemplo spread operator:

```
const obj1 = { name: 'John' };  
const obj2 = { ...obj1 };      // spread operator  
obj2.name = 'Edu'  
console.log(obj1.name)    // John  
console.log(obj2.name)    // Edu
```

# Fundamentos de JS - Objetos

Comparación entre objetos → comparación por referencia.

```
const a = 5;
```

```
const b = 5;
```

```
console.log(a === b) // true
```

```
const obj1 = { a: 5 };
```

```
const obj2 = { a: 5 };
```

```
console.log(obj1 === obj2) // false
```

[Ejemplos](#)

# Fundamentos de JS - Arrays

Los arrays son un tipo de dato de JS que nos permite almacenar un conjunto de datos. Por ejemplo:

```
const fruits = ["Apple", "Orange", "Lemon"];
```



# Fundamentos de JS - Arrays

Podemos acceder a los elementos del array mediante su índice.

```
console.log(fruits[0])           // Apple
```

```
fruits[0] = "banana"
```

```
console.log(fruits.length)      // 3
```

```
console.log(fruits[3])          // undefined
```

# Fundamentos de JS - Arrays

Los arrays nos ofrecen distintos métodos para poder consultar y operar con los datos de un array.

- `at(index)`
- `pop()`    `push(item)`
- `shift()`    `unshift(item)`

# Fundamentos de JS - Arrays

Podemos encontrarnos arrays de arrays. Es decir, arrays bidimensionales.

```
const matrix = [  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
];
```

# Fundamentos de JS - Arrays

[Métodos de los arrays más comunes](#)

[Todos los métodos de los arrays](#)

# Fundamentos de JS - Arrays

Para recorrer un array, podemos:

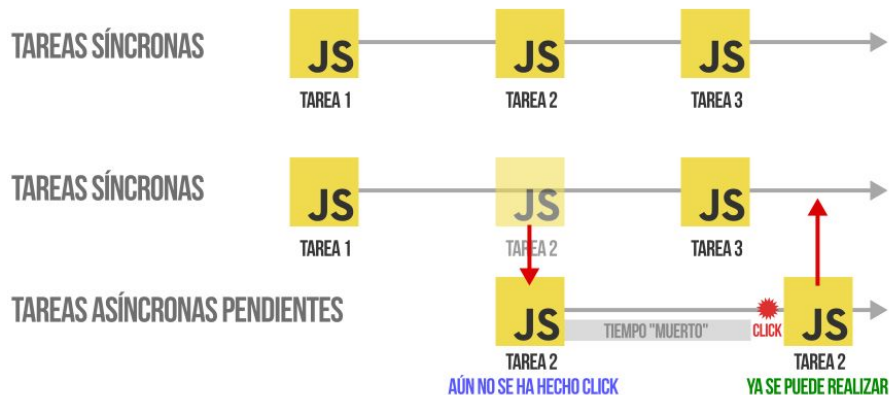
- Usar bucles de JS, como un for, while, for..of, etc.
- Utilizar métodos de los Arrays como forEach

```
for(let i = 0; i < array.length; i++) {    array.forEach(item => console.log(item))  
    console.log(i);  
}
```





# Fundamentos de JS - Asincronismo

Javascript permite ejecutar ciertas instrucciones de forma asíncrona. Esto quiere decir que nuestro intérprete de JS no esperará a que esa instrucción termine de ejecutarse y continuará ejecutando más líneas de JS.



# Fundamentos de JS - Asincronismo

Para controlar el asincronismo, tenemos varias opciones:

- Callbacks → obsoleto
- Promesas → 
- Async / await → 

# Fundamentos de JS - Promesas

Hay 2 involucrados en una promesa. Podemos identificarlos como productor y consumidor.

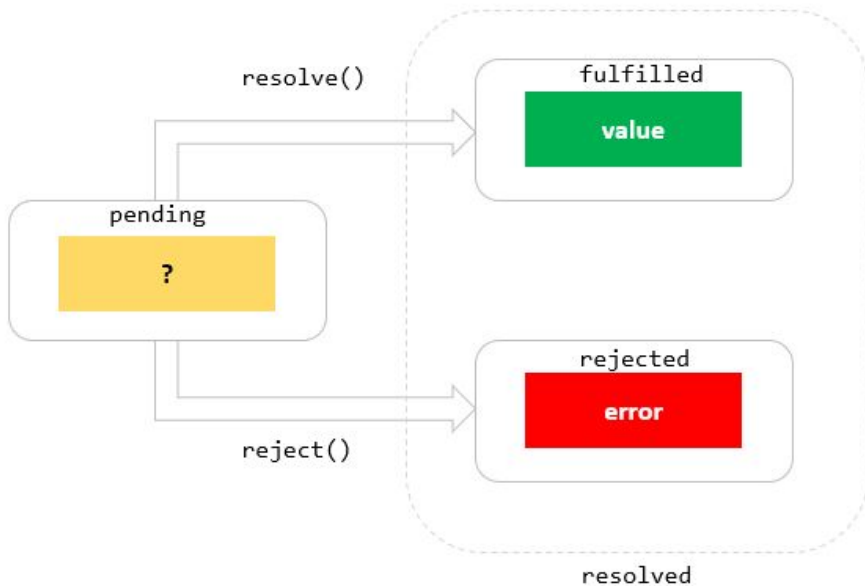
Productor: es quien gestiona un proceso asíncrono mediante una promesa y quien decide si la operación concluye bien o no.

Consumidor: es quien reacciona ante el cambio de estado de la promesa.

# Fundamentos de JS - Promesas

Los estados de una promesa son los siguientes

- pending
- fulfilled
- rejected



# Fundamentos de JS - Módulos

Los módulos son una forma de escribir código JS de manera organizada y reutilizable.

Cada módulo es un fichero de JavaScript que permite exportar definiciones que hagamos en su interior, como por ejemplo constantes, funciones, etc.

Por otra parte, también podremos importar en un módulo aquellas piezas que exporten otros.

## Ejemplos



# KEEPCODING

## Tech School

Madrid | Barcelona | Bogotá

**Datos de contacto**