

**CONQUER**

**BLOCKS**

**JAVASCRIPT**





# {JS}

JS

Clase 01

CONQUERBLOCKS

<Índice>

## Introducción a JS

¿Qué es JS?

---

¿Cómo hacer funcionar JavaScript?

---

Conceptos básicos del lenguaje

---

¿Qué hace diferente a JavaScript?

---

Anatomía de una variable y convención  
de nombres

---

CONQUERBLOCKS

## ¿Qué es JS?

CONQUERBLOCKS

## ¿Qué es JS?

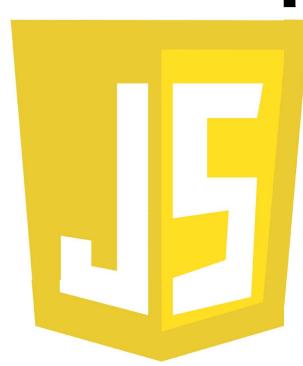
**HTML**



**CSS**



**JavaScript**



CONQUERBLOCKS

## ¿Qué es JS?

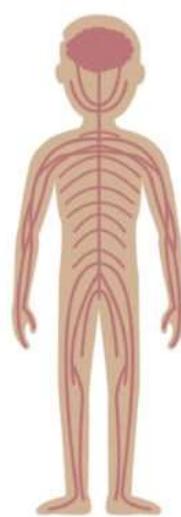
HTML the Skeleton



CSS the Skin



Javascript the Brain



CONQUERBLOCKS

## ¿Qué es JS?

### What's the Difference?



**HTML**

Hypertext Markup Language

*Create the structure*

- Controls the layout of the content
- Provides structure for the web page design
- The fundamental building block of any web page



**CSS**

Cascading Style Sheet

*Stylize the website*

- Applies style to the web page elements
- Targets various screen sizes to make web pages responsive
- Primarily handles the "look and feel" of a web page



**Javascript**

*Increase interactivity*

- Adds interactivity to a web page
- Handles complex functions and features
- Programmatic code which enhances functionality

CONQUERBLOCKS

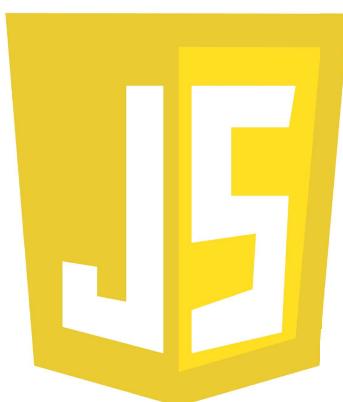
## ¿Qué es JS?

- Es el responsable de controlar toda la interactividad del usuario en la web
- Trabajes en lo que trabajes en **web** debes conocer JS si o sí
- Toda la web está hecha HTML + CSS + JS
- Son los cimientos de toda web

CONQUERBLOCKS

## ¿Qué es JS?

# JavaScript



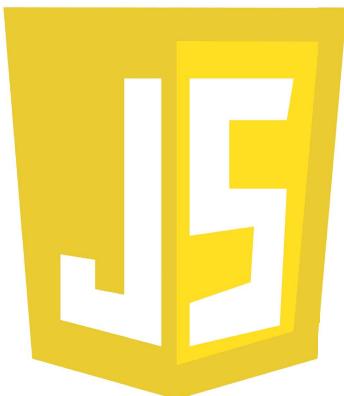
### Características

1. Texto plano - Cualquier editor
2. Lenguaje nativo de los navegadores
3. No necesita ser compilado ni transpilado
4. SI ES UN LENGUAJE DE PROGRAMACIÓN

CONQUERBLOCKS

## ¿Qué es JS?

# JavaScript



JavaScript fue desarrollado originalmente por

Brendan Eich de Netscape con el nombre de

Mocha, el cual fue renombrado posteriormente a

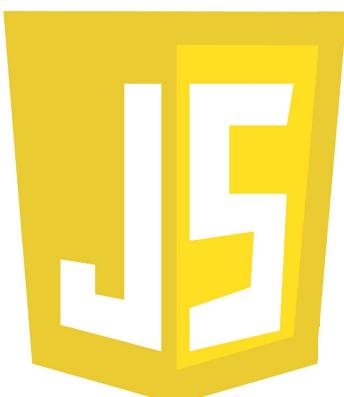
LiveScript, para finalmente quedar como

JavaScript.

**CONQUERBLOCKS**

## ¿Qué es JS?

# JavaScript



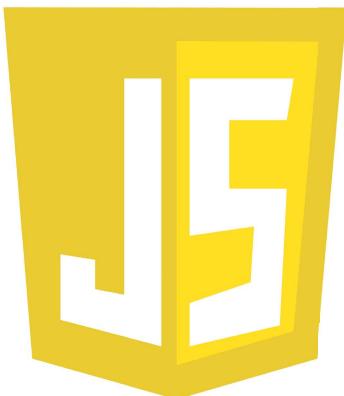
**Imperativo y estructurado**

- JavaScript es compatible con gran parte de la estructura de programación de C (por ejemplo, sentencias if, bucles for, sentencias switch, etc.).
- JavaScript difiere sobre todo en el ámbito o scope de las variables

**CONQUERBLOCKS**

## ¿Qué es JS?

# JavaScript



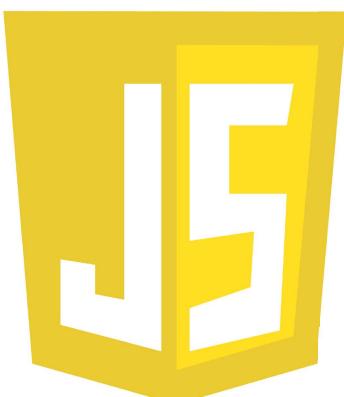
### Tipado dinámico

- Como en la mayoría de lenguajes de scripting, el tipo está asociado al valor, no a la variable.
- Por ejemplo, una variable x en un momento dado puede estar ligada a un número y más adelante, religada a una cadena.

CONQUERBLOCKS

## ¿Qué es JS?

# JavaScript



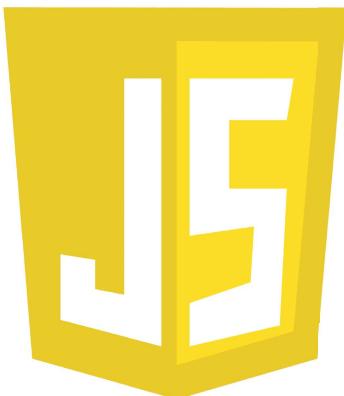
### Objetual

- JavaScript está formado casi en su totalidad por objetos.
- Los objetos en JavaScript son arrays asociativos, mejorados con la inclusión de prototipos (ver más adelante). Los nombres de las propiedades de los objetos son claves de tipo cadena: obj.x = 10 y obj['x'] = 10 son equivalentes,

CONQUERBLOCKS

## ¿Qué es JS?

# JavaScript



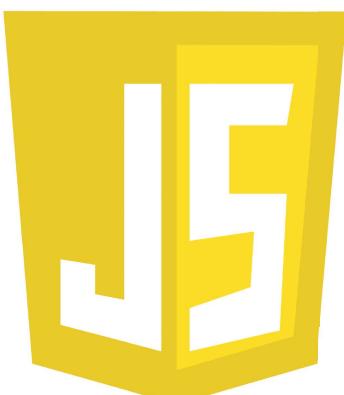
### Evaluación en tiempo de ejecución

- JavaScript incluye la función eval que permite evaluar expresiones expresadas como cadenas en tiempo de ejecución.
- Por ello se recomienda que eval sea utilizado con precaución y que se opte por utilizar la función

CONQUERBLOCKS

## ¿Qué es JS?

# JavaScript



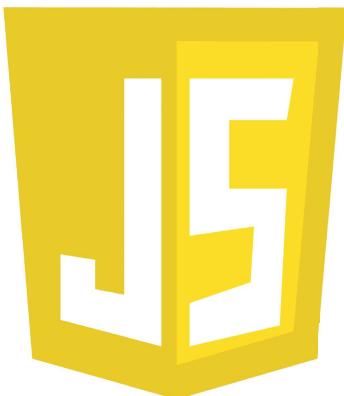
### Funciones de primera clase

- En JavaScript a las funciones se les llaman ciudadanos de primera clase y es que hasta las funciones son objetos con propiedades y métodos

CONQUERBLOCKS

## ¿Qué es JS?

# JavaScript



### Entornos de ejecución

- JavaScript normalmente depende del entorno en el que se ejecute para ofrecer objetos y métodos por los que los scripts pueden interactuar con el "mundo exterior".
- De hecho, depende del entorno para ser capaz de proporcionar la capacidad de incluir o importar scripts (por ejemplo, en HTML por medio del tag <script>). (Esto no es una característica del lenguaje, pero es común en la mayoría de las implementaciones de JavaScript.)

CONQUERBLOCKS

## ¿Cómo hacer funcionar JS?

CONQUERBLOCKS

## ¿Cómo hacer funcionar JS?

- Consola del navegador
- Etiqueta script dentro de html
- Script externo
- nodejs desde la terminal

### Ejecuta JS

CONQUERBLOCKS

## ¿Cómo hacer funcionar JS?

### Consola del navegador

- El clásico primer ejemplo cuando se comienza a programar, es crear un programa que muestre por pantalla un texto, generalmente el texto «Hola Mundo».
- También podemos realizar, por ejemplo, operaciones numéricas. En la consola Javascript podemos hacer esto de forma muy sencilla:

CONQUERBLOCKS

# ¿Cómo hacer funcionar JS?

## Consola del navegador

```
console.log("Hola Mundo");  
console.log(2 + 2);
```

CONQUERBLOCKS

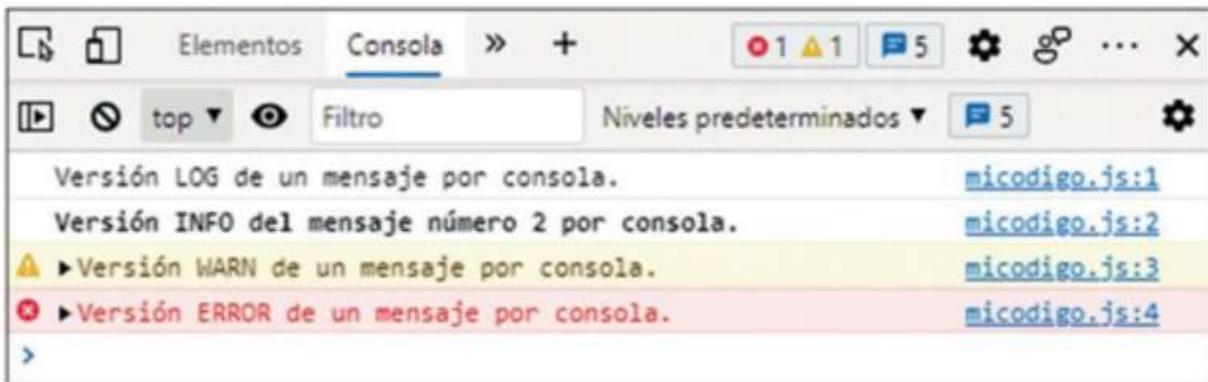
# ¿Cómo hacer funcionar JS?

## Consola del navegador

Función	Descripción
console.log()	Muestra la información proporcionada en la consola Javascript.
console.info()	Equivalente al anterior. Se utiliza para mensajes de información.
console.warn()	Muestra información de advertencia. Aparece en amarillo.
console.error()	Muestra información de error. Aparece en rojo.
console.clear()	Limpia la consola. Equivalente a pulsar <b>CTRL + L</b> o escribir <code>clear()</code> .

CONQUERBLOCKS

## ¿Cómo hacer funcionar JS?



CONQUERBLOCKS

## ¿Cómo hacer funcionar JS?

### Etiqueta script en nuestro html

- Este método de escribir scripts se denomina Javascript en línea (inline), y significa que el Javascript está escrito directamente en el código HTML. Nos puede servir como ejemplo inicial, pero no es la forma recomendable de escribirlo, ya que lo ideal es separar el código HTML del código Javascript

CONQUERBLOCKS

## ¿Cómo hacer funcionar JS?

### Etiqueta script en nuestro html

```
<html>
  <head>
    <title>Título de la página</title>
    <script>
      console.log("¡Hola!");
    </script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

CONQUERBLOCKS

## ¿Cómo hacer funcionar JS?

### Script externo

- Esta otra forma de incluir Javascript en una página tiene la ventaja de, en el caso de necesitar incluir el código Javascript desde varios documentos HTML, no tendremos que volver a escribir dicho código, sino simplemente referenciar el nombre del mismo archivo Javascript a incluir en todas las páginas HTML.

CONQUERBLOCKS

# ¿Cómo hacer funcionar JS?

## Script externo

```
<html>
  <head>
    <title>Título de la página</title>
    <script src="js/index.js"></script>
  </head>
  <body>
    <p>Ejemplo de texto.</p>
  </body>
</html>
```

CONQUERBLOCKS

# ¿Cómo hacer funcionar JS?

## Script externo Dónde ubicar la etiqueta script

Ubicación	¿Cómo descarga el archivo Javascript?	Estado de la página
En <head>	ANTES de empezar a dibujar la página.	Página aún no dibujada.
En <body>	DURANTE el dibujado de la página.	Dibujada hasta donde está la etiqueta <script>.
Antes de </body>	DESPUÉS de dibujar la página.	Dibujada al 100%.

CONQUERBLOCKS

# ¿Cómo hacer funcionar JS?

**Etiqueta script**

CONQUERBLOCKS

# ¿Cómo hacer funcionar JS?

**Ejecución con nodejs**

CONQUERBLOCKS

## ¿Qué hace diferente a JS?

CONQUERBLOCKS

## ¿Qué hace diferente a JS?

- Se puede ejecutar en el front y back
- Sabe manipular el DOM (lista 50 elementos en html por ejemplo)
- No es el mejor lenguaje para aprender a programar
- Se partirá de todo lo visto en los módulos de programación

CONQUERBLOCKS

## ¿Qué hace diferente a JS?

### ECMAScript

A lo largo de los años, Javascript ha ido sufriendo modificaciones que los navegadores han ido implementando para acomodarse a la última versión de ECMAScript cuanto antes. La lista de versiones de ECMAScript aparecidas hasta el momento son las siguientes, donde encontramos las versiones enmarcadas en lo que podemos considerar el pasado de Javascript

CONQUERBLOCKS

## ¿Qué hace diferente a JS?

Ed.	Fecha	Nombre formal / informal	Cambios significativos
1	JUN/1997	ECMAScript 1997 (ES1)	Primera edición
2	JUN/1998	ECMAScript 1998 (ES2)	Cambios leves
3	DIC/1999	ECMAScript 1999 (ES3)	RegExp, try/catch, etc...
4	AGO/2008	ECMAScript 2008 (ES4)	Versión abandonada.
5	DIC/2009	ECMAScript 2009 (ES5)	Strict mode, JSON, etc...
5.1	DIC/2011	ECMAScript 2011 (ES5.1)	Cambios leves

CONQUERBLOCKS

## ¿Qué hace diferente a JS?

A partir del año 2015, se marcó un antes y un después en el mundo de Javascript, estableciendo una serie de cambios que lo transformarían en un lenguaje moderno, partiendo desde la especificación de dicho año, hasta la actualidad:

Ed.	Fecha	ECMAScript	Cambios significativos
6	JUN/2015	ES2015	Clases, módulos, hashmaps, sets, for of, proxies...
7	JUN/2016	ES2016	Array includes(), Exponenciación **
8	JUN/2017	ES2017	Async/await
9	JUN/2018	ES2018	Rest/Spread operator, Promise.finally()...
10	JUN/2019	ES2019	flat, flatMap, trimStart(), optional error catch...
11	JUN/2020	ES2020	Dynamic imports, BigInt, Promise.allSettled

CONQUERBLOCKS

## Anatomía de una variable

CONQUERBLOCKS

## Anatomía de una variable

¿Qué es una variable?

CONQUERBLOCKS

## Anatomía de una variable

- Declaración
- Asignación
- Nombres descriptivos
- Evitar ambigüedades
- Evitar palabras prohibidas
- Evitar símbolos de puntuación

CONQUERBLOCKS

# Anatomía de una variable

- Ser descriptivos
- Usar camelcase
- Algunos consejos
- Más consejos

CONQUERBLOCKS

# Anatomía de una variable

```
// Good
const isActive = false

// Bad
const is-active = false
const is_active = false
```

```
// Good
const article = {title: 'Some title'}
const isAlreadyPaid = true

// Bad
const a = {title: 'Some title'}
const paid = true
```

```
// Good
const isLoggedIn = true
const shouldBeRendered = true

// Bad
const logged = true
const areDefined = true
```

CONQUERBLOCKS

## Anatomía de una variable

Declaremos nuestras primeras variables

CONQUERBLOCKS

## Anatomía de una variable

¿let, var o const?

CONQUERBLOCKS

## Anatomía de una variable

¿Qué podemos guardar en las variables?

CONQUERBLOCKS

«Despedida»

Email

[bienvenidosaez@gmail.com](mailto:bienvenidosaez@gmail.com)

Instagram

[@bienvenidosaez](https://www.instagram.com/bienvenidosaez)

Youtube

[youtube.com/bienvenidosaez](https://www.youtube.com/bienvenidosaez)

CONQUERBLOCKS



# {JS}

JS

Clase 02

CONQUERBLOCKS

<Índice>

## Conceptos básicos del lenguaje

Sentencias, Bloques e Identificadores

---

Comentarios

---

Tipos de datos básicos y typeof

---

Declaración e inicialización

---

CONQUERBLOCKS

# **Sentencias, Bloques e identificadores**

**CONQUERBLOCKS**

## **Identificadores**

**CONQUERBLOCKS**

## Identificadores

En JavaScript, los identificadores son nombres utilizados para identificar variables, funciones, clases y otros elementos en el código. Aquí hay algunas reglas y convenciones importantes sobre los identificadores en JavaScript

CONQUERBLOCKS

## Identificadores

### Reglas Básicas

- Deben comenzar con una letra (A-Z, a-z), un guion bajo (\_) o un signo de dólar (\$).
- Los caracteres siguientes pueden ser letras, dígitos (0-9), guiones bajos o signos de dólar.
- No se permiten caracteres especiales o espacios.

CONQUERBLOCKS

## Identificadores

### Sensibilidad a Mayúsculas y Minúsculas:

- JavaScript es sensible a mayúsculas y minúsculas. Por ejemplo, variable, Variable, y VARIABLE son tres identificadores diferentes.

CONQUERBLOCKS

## Identificadores

### Palabras Reservadas:

- Hay ciertas palabras que no se pueden usar como identificadores porque están reservadas por el lenguaje, como if, for, let, const, etc.

CONQUERBLOCKS

## Identificadores

### Estilo de Nomenclatura:

- Camel Case: Usado comúnmente para nombrar variables y funciones (ejemplo: miVariable, calcularEdad).
- Pascal Case: A menudo utilizado para clases (ejemplo: Producto, CocheElectrico).
- Snake Case: Menos común en JavaScript, pero usado a veces para constantes (ejemplo: MAX\_VALOR, URL\_BASE).

CONQUERBLOCKS

## Identificadores

### Identificadores Descriptivos:

- Se recomienda usar nombres descriptivos para hacer el código más legible (ejemplo: sumaTotal, nombreUsuario en lugar de s, n).

CONQUERBLOCKS

## Identificadores

### Identificadores Largos:

- No hay un límite de longitud específico para los identificadores, pero es una buena práctica mantenerlos de una longitud manejable.

CONQUERBLOCKS

## Sentencias

CONQUERBLOCKS

## Sentencias

En JavaScript, una sentencia es la unidad mínima de ejecución. Es una instrucción que le dice al navegador qué hacer. Cada sentencia en

JavaScript puede llevar a cabo una acción, como declarar una variable, asignar un valor, ejecutar una función, controlar el flujo del programa mediante bucles y condicionales, y más.

CONQUERBLOCKS

## Sentencias

### Estructura básica

Una sentencia puede ser tan simple como una única palabra clave o variable, o tan compleja como una serie de llamadas de funciones y operaciones.

CONQUERBLOCKS

# Sentencias

## Terminación

Por lo general, las sentencias se terminan con un punto y coma (;). Sin embargo, debido a la Inserción Automática de Punto y Coma (ASI) en JavaScript, no siempre es necesario, aunque a menudo se recomienda para evitar errores sutiles.

CONQUERBLOCKS

# Sentencias

## Ejemplos

- Declaración de Variables: `let x = 5;`
- Asignación: `x = 10;`
- Funciones: `function miFuncion() { ... }`
- Condicionales: `if (x > 5) { ... }`
- Bucles: `for (let i = 0; i < 10; i++) { ... }`
- Expresiones: `console.log(x);`

CONQUERBLOCKS

## Bloque

CONQUERBLOCKS

## Bloque

En JavaScript, un bloque de código es una sección del código fuente que está delimitada por llaves { }. Estos bloques definen un ámbito o contexto en el cual se agrupan múltiples sentencias de JavaScript. Los bloques de código son **fundamentales** en la estructura y organización del código en JavaScript, y son utilizados en varios contextos

CONQUERBLOCKS

## Bloque

En JavaScript, un bloque de código es una sección del código fuente que está delimitada por llaves { }. Estos bloques definen un ámbito o contexto en el cual se agrupan múltiples sentencias de JavaScript. Los bloques de código son **fundamentales** en la estructura y organización del código en JavaScript, y son utilizados en varios contextos

CONQUERBLOCKS

## Bloque

### Agrupación

Permiten agrupar varias sentencias juntas para que puedan ser tratadas como una sola unidad.

Esto es especialmente útil en estructuras de control de flujo como condicionales (if, else) y bucles (for, while).

CONQUERBLOCKS

## Bloque

### Ámbito de variables (Scope)

Las variables declaradas dentro de un bloque de código tienen un ámbito local. Esto significa que están accesibles únicamente dentro de ese bloque. Por ejemplo, una variable declarada dentro de un bloque if no es accesible fuera de ese bloque.

CONQUERBLOCKS

## Bloque

### Estructuras de control

En estructuras de control de flujo, los bloques de código definen las sentencias que se ejecutan en condiciones específicas (como en if, else) o en cada iteración de un bucle (for, while).

CONQUERBLOCKS

# Bloque

## Funciones

El ejemplo más claro de bloque de código

CONQUERBLOCKS

# Bloque

```
if (condicion) {  
    // Bloque de código para 'if'  
    console.log("Condición cumplida");  
}
```

```
for (let i = 0; i < 10; i++) {  
    // Bloque de código para 'for'  
    console.log(i);  
}
```

```
function miFuncion() {  
    // Bloque de código para la función  
    console.log("Hola, mundo!");  
}
```

CONQUERBLOCKS

## Bloque

### Conclusión

Los bloques de código son una herramienta fundamental en JavaScript para organizar y controlar el flujo de ejecución del programa

CONQUERBLOCKS

## Comentarios

CONQUERBLOCKS

## Comentarios

En JavaScript, los comentarios son fragmentos de texto dentro del código fuente que son ignorados por el intérprete de JavaScript.

Existen dos tipos de comentarios en JavaScript:

Son útiles para explicar qué hace el código, hacer anotaciones, o temporalmente deshabilitar ciertas partes del código durante el desarrollo y la depuración.

CONQUERBLOCKS

## Comentarios

### Existen dos tipos de comentarios en JavaScript

Comentarios de una sola línea

Comentarios en varias líneas

CONQUERBLOCKS

## Comentarios

### Comentarios de línea

Comienzan con `//`. Todo el texto que sigue a `//` en esa línea es tratado como un comentario

```
// Esto es un comentario de una sola línea  
var x = 5; // Este también es un comentario, después de la sentencia
```

CONQUERBLOCKS

## Comentarios

### Comentarios de bloque

Comienzan con `/*` y terminan con `*/`. Todo el texto entre `/*` y `*/` es tratado como un comentario, independientemente de cuántas líneas abarque.

```
/* Este es un comentario  
que abarca varias  
líneas */  
var y = 10;
```

CONQUERBLOCKS

## Comentarios

Los comentarios de varias líneas son especialmente útiles para comentar secciones más grandes de código o para añadir descripciones más detalladas. Además, permiten "comentar" temporalmente grandes bloques de código que no se desean ejecutar durante ciertas fases del desarrollo.

CONQUERBLOCKS

## Comentarios

Además se pueden utilizar para linters y programas de post-procesado de Javascript

CONQUERBLOCKS

## Comentarios

Un programador pasa el 80% del tiempo leyendo y el 20% escribiendo, no lo olvides

CONQUERBLOCKS

## Tipos de datos básicos

CONQUERBLOCKS

## Tipos de datos básicos

### Tipos de datos básicos

- string
- number
- boolean
- undefined
- null
- bigint
- symbol

CONQUERBLOCKS

## Tipos de datos básicos

### String

Utilizado para representar datos textuales. Se compone de una secuencia de caracteres y puede ser definido utilizando comillas simples, dobles o acentos graves (para plantillas literales).

Ejemplo: var nombre = "Alice";

Comillas dobles, simple y backticks

Lo estudiaremos en profundidad

CONQUERBLOCKS

# Tipos de datos básicos

## Number

Representa tanto enteros como números de punto flotante.

JavaScript utiliza una representación de punto flotante de doble precisión para todos sus números.

Ejemplo: var edad = 25;

Lo estudiaremos en profundidad

CONQUERBLOCKS

# Tipos de datos básicos

## Boolean

Este tipo tiene dos valores posibles: true (verdadero) y false (falso). Es útil en operaciones lógicas y toma de decisiones en el flujo de control.

Ejemplo: var esMayorDeEdad = true;

Lo estudiaremos en profundidad

CONQUERBLOCKS

# Tipos de datos básicos

## Undefined

Se utiliza para indicar una variable que ha sido declarada pero aún no se le ha asignado un valor.

Ejemplo: var esMayorDeEdad;

CONQUERBLOCKS

# Tipos de datos básicos

## Undefined

Se produce cuando:

- Al declarar una variable pero no asignarle un valor.
- Al intentar acceder a una propiedad que no existe en un objeto.
- Al intentar acceder a un elemento que no existe en un arreglo.
- Cuando una función no tiene una sentencia return explícita, devuelve undefined.

CONQUERBLOCKS

# Tipos de datos básicos

## Null

Es un tipo que tiene un único valor: null.

Se utiliza para representar la ausencia intencional de un valor de objeto. En el campo de la programación el valor null siempre hace referencia a una dirección inválida

Ejemplo: var esMayorDeEdad = null;

CONQUERBLOCKS

# Tipos de datos básicos

A los programadores principiantes de JavaScript les cuesta entender al diferencia entre undefined y null, puesto que ambos tipos de datos significan ausencia de valor. No obstante, existen matices que los convierten en tipos de datos completamente distintos.

CONQUERBLOCKS

## Tipos de datos básicos

Este es el más importante: undefined significa que no hay valor porque aún no se ha definido; en cambio, null significa que no hay valor porque así lo ha indicado expresamente el programador.

CONQUERBLOCKS

## Tipos de datos básicos

Entender la diferencia entre undefined y null es crucial para manejar adecuadamente los estados de las variables y los errores en JavaScript.

CONQUERBLOCKS

# Tipos de datos básicos

## BigInt

Introducido en versiones recientes de JavaScript, este tipo permite trabajar con números enteros muy grandes que superan el límite de los números del tipo Number.

Ejemplo: var numeroGrande = 1234567890123458901234567890n;

CONQUERBLOCKS

# Tipos de datos básicos

## Symbol

Symbol: Introducido en ECMAScript 2015, es un tipo de datos cuyas instancias son únicas e inmutables. Son útiles para crear identificadores únicos para propiedades de objetos.

Ejemplo: let miSímbolo = Symbol('mi identificador único');

CONQUERBLOCKS

# Tipos de datos básicos

Demos

CONQUERBLOCKS

# Tipos de datos básicos

¿Qué tipo de dato tiene  
una variable?

CONQUERBLOCKS

## Tipos de datos básicos

### **typeof**

es utilizado para determinar el tipo de una variable o expresión. Esto es especialmente útil en JavaScript debido a su naturaleza de tipado dinámico, donde el tipo de una variable puede cambiar en tiempo de ejecución.

CONQUERBLOCKS

## Typeof

### **typeof**

- Sintaxis: `typeof operando`, donde operando es la variable o expresión cuyo tipo se quiere conocer.
- Retorno: Devuelve una cadena de texto que indica el tipo del operando.

CONQUERBLOCKS

# Typeof

## typeof

Valores posibles de retorno:  
undefined, boolean, number, string, symbol,  
object o function.

CONQUERBLOCKS

# Typeof

## importancia de typeof

1. Depuración y Validación: Permite a los desarrolladores verificar el tipo de las variables durante la depuración y validar tipos de datos antes de realizar operaciones específicas.
2. Evitar Errores: Al verificar el tipo de una variable antes de operar con ella, se pueden prevenir errores comunes como intentar realizar operaciones inadecuadas para un tipo de dato específico.

CONQUERBLOCKS

# Typeof

## limitaciones de typeof

- No puede diferenciar entre un objeto y un arreglo, o entre un objeto y null. Para estos casos, se suelen utilizar otros métodos como `Array.isArray()` para arreglos o comparaciones directas con null.
- No puede identificar tipos de objetos más específicos (como instancias de clases personalizadas). Para eso usaremos `instanceof` más adelante

CONQUERBLOCKS

# Typeof

**Nos quedan por ver los tipos  
no primitivos**

CONQUERBLOCKS

## Declaración e inicialización

CONQUERBLOCKS

## Declaración e inicialización

En JavaScript, existen principalmente tres formas de declarar variables, cada una con sus propias características y niveles de alcance (scope).

Estas son var, let, y const.

¿Os acordáis de lo que era un bloque de código?

CONQUERBLOCKS

# Declaración e inicialización

## **var**

- var es la forma más antigua de declarar variables en JavaScript.
- Tiene un alcance (scope) de función; es decir, una variable declarada con var está disponible en toda la función en la que fue declarada, o globalmente si se declara fuera de una función.
- Permite la redeclaración de la misma variable en el mismo ámbito.
- Tiene lo que se conoce como "hoisting" (elevación), lo que significa que se puede usar la variable antes de su declaración en el código.

CONQUERBLOCKS

# Declaración e inicialización

## **let**

- Introducido en ECMAScript 2015 (ES6), let permite declarar variables con alcance de bloque (block scope), lo que significa que la variable solo existe dentro del bloque en el que fue declarada.
- No permite la redeclaración de la misma variable dentro del mismo bloque.
- Menos propenso a errores que var debido a su alcance de bloque y no permite el uso de la variable antes de su declaración.

CONQUERBLOCKS

## Declaración e inicialización

### **const**

- También introducido en ES6, const se utiliza para declarar constantes.
- Tiene un alcance de bloque, similar a let.
- No permite la redeclaración ni la reasignación de la variable.
- Debe ser inicializada en el momento de su declaración.
- Aunque una variable declarada con const es inmutable en términos de reasignación de su valor primitivo, si se trata de un objeto, las propiedades de ese objeto pueden ser modificadas.

CONQUERBLOCKS

## Declaración e inicialización

### **var, let o const**

- var se usa cada vez menos en el desarrollo moderno de JavaScript debido a sus peculiaridades y potenciales problemas con el alcance de función y hoisting.
- let es preferible cuando se necesita una variable cuyo valor va a cambiar, como contadores en bucles, o valores que se reasignan en un bloque de código.
- const es la mejor opción para declarar variables que no deben cambiar después de su asignación inicial. Esto mejora la legibilidad del código y reduce la posibilidad de errores inesperados.

CONQUERBLOCKS

## Anatomía de una variable

Mi consejo, no uses var salvo que quieras variables globales y entiendas bien cómo funcionan

CONQUERBLOCKS

## Anatomía de una variable

¿Qué podemos guardar en las variables?

CONQUERBLOCKS

# Ejercicios

CONQUERBLOCKS

## Ejercicios de tipos primitivos

1. Declara una variable saludo y asígnale el texto "Hola Mundo".
2. Declara una variable edad y asígnale tu edad.
3. Declara una variable estaSoleado y asigna un valor booleano dependiendo si está soleado o no.
4. Declara una variable valorNulo y asígnale el valor null.
5. Declara una variable sinDefinir sin asignarle un valor.
6. Declara una variable numeroGrande y asígnale un número entero grande usando BigInt.
7. Declara una variable mensaje y asígnale el texto 'Aprendiendo JavaScript' usando comillas simples.
8. Declara una variable precio y asígnale un valor decimal, por ejemplo, el precio de un artículo.
9. Declara una variable estaLloviendo y asigna el valor opuesto a estaSoleado.
10. Declara una variable temperatura y asígnale un número negativo para representar una temperatura bajo cero.

# «Despedida»

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**

**CONQUERBLOCKS**



# {JS}

JS

Clase 03

CONQUERBLOCKS

<Índice>

## Operadores y funciones en Js

Operadores básicos

---

Operadores avanzados

---

¿Qué es una función?

---

CONQUERBLOCKS

# Operadores básicos

CONQUERBLOCKS

## Tipos de operadores

- Aritméticos
- Asignación
- Unarios
- Comparacion
- Binarios

CONQUERBLOCKS

## Operadores aritméticos

Se utilizan para realizar operaciones matemáticas

CONQUERBLOCKS

## Operadores aritméticos

Los típicos

+ - \* / % \*\*

CONQUERBLOCKS

# Operadores de asignación

Estos operadores nos permiten asignar información a diferentes constantes o variables a través del símbolo `=`, lo cuál es bastante lógico pues así lo hacemos en matemáticas.

CONQUERBLOCKS

# Operadores de asignación

Nombre	Operador	Descripción
Asignación	<code>c = a + b</code>	Asigna el valor de la parte derecha
Suma y asignación	<code>a += b</code>	Es equivalente a <code>a = a + b</code> .
Resta y asignación	<code>a -= b</code>	Es equivalente a <code>a = a - b</code> .
Multiplicación y asignación	<code>a *= b</code>	Es equivalente a <code>a = a * b</code> .
División y asignación	<code>a /= b</code>	Es equivalente a <code>a = a / b</code> .
Módulo y asignación	<code>a %= b</code>	Es equivalente a <code>a = a % b</code> .
Exponenciación y asignación	<code>a **= b</code>	Es equivalente a <code>a = a ** b</code> .

CONQUERBLOCKS

## Operadores unarios

Los operadores unarios se diferencian de otros operadores porque actúan sobre un solo valor o variable, en lugar de dos operandos. Esto significa que realizan operaciones utilizando únicamente un elemento almacenado en una variable.

CONQUERBLOCKS

## Operadores unarios

Nombre	Operador	Descripción
Incremento	<code>a++</code>	Usa el valor de <code>a</code> y luego lo incrementa. También llamado <b>postincremento</b> .
Decremento	<code>a--</code>	Usa el valor de <code>a</code> y luego lo decrementa. También llamado <b>postdecremento</b> .
Incremento previo	<code>++a</code>	Incrementa el valor de <code>a</code> y luego lo usa. También llamado <b>preincremento</b> .
Decremento previo	<code>--a</code>	Decrementa el valor de <code>a</code> y luego lo usa. También llamado <b>predecremento</b> .

CONQUERBLOCKS

## Operadores unarios

Probemos:

```
let a = 0;           let a = 0;  
  
while (a < 5) {      while (a < 5) {  
    console.log(a, a++, a);  console.log(a, ++a, a);  
}  
}
```

CONQUERBLOCKS

## Operadores de comparación

Los operadores de comparación se emplean en la programación, frecuentemente dentro de una estructura condicional como un 'if', aunque también pueden usarse en otros contextos, para efectuar verificaciones. Estas expresiones comparativas retornan un valor booleano, que puede ser verdadero (true) o falso (false).

CONQUERBLOCKS

# Operadores de comparación

Nombre	Operador	Descripción
Operador de igualdad ==	<code>a == b</code>	Comprueba si el <b>valor</b> de <b>a</b> es igual al de <b>b</b> . <b>No comprueba tipo de dato</b> .
Operador de desigualdad !=	<code>a != b</code>	Comprueba si el <b>valor</b> de <b>a</b> no es igual al de <b>b</b> . <b>No comprueba tipo de dato</b> .
Operador mayor que >	<code>a &gt; b</code>	Comprueba si el valor de <b>a</b> es mayor que el de <b>b</b> .
Operador mayor/igual que >=	<code>a &gt;= b</code>	Comprueba si el valor de <b>a</b> es mayor o igual que el de <b>b</b> .
Operador menor que <	<code>a &lt; b</code>	Comprueba si el valor de <b>a</b> es menor que el de <b>b</b> .
Operador menor/igual que <=	<code>a &lt;= b</code>	Comprueba si el valor de <b>a</b> es menor o igual que el de <b>b</b> .
Operador de identidad ===	<code>a === b</code>	Comprueba si el <b>valor y el tipo de dato</b> de <b>a</b> es igual al de <b>b</b> .
Operador no idéntico !==	<code>a !== b</code>	Comprueba si el <b>valor y el tipo de dato</b> de <b>a</b> no es igual al de <b>b</b> .

CONQUERBLOCKS

# Operadores de comparación

## Probemos

```
5 == 5      // true    (ambos son iguales, coincide su valor)  
"5" == 5    // true    (ambos son iguales, coincide su valor)  
5 === 5     // true    (ambos son idénticos, coincide su valor y su tipo de dato)  
"5" === 5   // false   (no son idénticos, coincide su valor, pero no su tipo de dato)
```

CONQUERBLOCKS

## Operadores binarios

En Javascript no son muy utilizados, pero existen los operadores binarios que funcionan a nivel de bit. Es decir, con variables que solo pueden tomar valores 0 y 1.

Existen operadores para desplazar bits, y algunas lógicas.

CONQUERBLOCKS

## Operadores avanzados

CONQUERBLOCKS

# Operadores lógicos

## Operadores lógicos

Son los que podemos ver en cualquier lenguaje de programación, aunque en Javascript, tienen sus particularidades propias

CONQUERBLOCKS

# Operadores lógicos

## Operador AND

Sigue la misma tabla de verdad de todos los lenguajes de programación

Operador AND		
Condición 1	Condición 2	Resultado
FALSO	FALSO	FALSO
FALSO	VERDADERO	FALSO
VERDADERO	FALSO	FALSO
VERDADERO	VERDADERO	VERDADERO

CONQUERBLOCKS

# Operadores lógicos

## Operador AND

Pero en Javascript tiene otra particularidad cuando lo ejecutamos entre dos variables.

Devolverá el primer valor si es false, o el segundo valor si el primero es true. Esto se puede leer de forma que «devuelve b si a y b son verdaderos, sino a».

CONQUERBLOCKS

# Operadores lógicos

```
0 && undefined      // 0
undefined && 0        // undefined
55 && null          // null
null && 55            // null
44 && 20              // 20
45 && "OK"            // "OK"
false && "OK"          // false
```

CONQUERBLOCKS

# Operadores lógicos

## Operador OR

Si operamos con booleanos, sigue la tabla de verdad de toda la vida del operador OR.

Operador OR		
Condición 1	Condición 2	Resultado
FALSO	FALSO	FALSO
FALSO	VERDADERO	VERDADERO
VERDADERO	FALSO	VERDADERO
VERDADERO	VERDADERO	VERDADERO

CONQUERBLOCKS

# Operadores lógicos

## Operador OR

Si no son booleanos funcionará de la siguiente manera

Devolverá el primer valor si es true, o el segundo valor si el primero es false. Esto se puede leer de forma que «devuelve a (si es verdadero), o si no, b».

CONQUERBLOCKS

# Operadores lógicos

```
0 || null      // null (se evalua como false || false, devuelve el segundo)  
44 || undefined // 44 (se evalua como true || false, devuelve el primero)  
0 || 17        // 17 (se evalua como false || true, devuelve el segundo)  
4 || 10        // 4 (se evalua como true || true, devuelve el primero)
```

CONQUERBLOCKS

# Operadores lógicos

```
"Conquer" || "Unknown name"    // "Conquer"  
null || "Unknown name"         // "Unknown name"  
false || "Unknown name"        // "Unknown name"  
undefined || "Unknown name"   // "Unknown name"  
0 || "Unknown name"           // "Unknown name"
```

CONQUERBLOCKS

# Operadores lógicos

## Operador Ternario

El operador ternario en JavaScript es un operador condicional que es una versión abreviada de la declaración if-else. Consiste en tres partes y se utiliza para asignar o retornar un valor basado en una condición

```
condición ? expresión1 : expresión2
```

CONQUERBLOCKS

# Operadores lógicos

## Operador Ternario

```
let edad = 20;  
let mensaje = edad >= 18 ? 'Mayor de edad' : 'Menor de edad';  
console.log(mensaje);
```

CONQUERBLOCKS

# Operadores lógicos

## Operador Ternario

```
// Sin operador ternario
let role;
if (name === "B3") {
  role = "Mago";
} else {
  role = "Novato";
}

// Con operador ternario
const role = name === "B3" ? "Mago" : "Novato";
```

CONQUERBLOCKS

# Operadores lógicos

## Operador Nullish coalescing ??

El operador a ?? b devuelve b sólo cuando a es undefined o null. De lo contrario devuelve a

CONQUERBLOCKS

# Operadores lógicos

```
42 || 50          // 42
42 ?? 50         // 42 (ambos se comportan igual)
false || 50       // 50
false ?? 50       // false
0 || 50          // 50
0 ?? 50          // 0
null || 50        // 50
null ?? 50        // 50
undefined || 50    // 50
undefined ?? 50    // 50
```

CONQUERBLOCKS

# Operadores lógicos

## Asignación lógica nula ??=

Esto se usa en Javascript por ciertos tipos de operaciones

Existen ciertos casos donde, si una variable tiene valores null o undefined (valores nullish) y sólo en esos casos, queremos cambiar su valor.

CONQUERBLOCKS

# Operadores lógicos

```
// Sin asignación lógica nula  
if (x === null || x === undefined)  
{  
    x = 50;  
}  
  
// Con asignación lógica nula  
x ??= 50;
```

CONQUERBLOCKS

# Operadores lógicos

## Operador lógico NOT

Es un operador unario

Devuelve el valor negado, es decir, lo convierte a booleano y devuelve su negación.

CONQUERBLOCKS

# Operadores lógicos

```
!true          // false
!false         // true
!!true         // true
!!false        // false
!!!true        // false
!5             // false
!0             // true
!" "           // true (se evalua como !0, que es !false)
!(10 || 23)    // false (se evalua como !10, que es !true)
```

CONQUERBLOCKS

# Primer acercamiento a funciones

CONQUERBLOCKS

# Funciones

¿Qué son y para qué se utilizan?

CONQUERBLOCKS

# Funciones

Diferencia entre una función y un método

CONQUERBLOCKS

# Funciones

```
function saludar(nombre = '') {  
  console.log("Hola " + nombre);  
}  
  
saludar();  
saludar('Bienve');  
console.log(typeof saludar);|
```

CONQUERBLOCKS

# Tipos de datos básicos

```
1 function saludar(nombre = '') {  
2   return ("Hola " + nombre);  
3 }  
  
4  
5 saludar();  
6 console.log(saludar());  
7 console.log(saludar("Bienve"));|
```

CONQUERBLOCKS

# Typeof

**En JS las funciones son  
objetos, y son de primer nivel**

**Daremos una clase completa  
sobre funciones por su  
peculiaridad**

CONQUERBLOCKS

**Ejercicios en el repositorio**

CONQUERBLOCKS

# «Despedida»

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**

**CONQUERBLOCKS**



# {JS}

Clase 04

JS

<Índice>

## Funciones en Javascript

¿Qué es una función?

---

Invocación de una función

---

Return o no return

---

Parámetros

---

Paso por referencia y por valor

---

Cómo declarar funciones

---

## ¿Qué es una función?

## ¿Qué es una función?

- ¿Qué es una función?
- Ventajas del uso de funciones

## ¿Qué es una función?

¿Qué es una función?

Bloques de código ejecutable a los que podemos pasar parámetros y operar con algo.

Además, las funciones podrán devolvernos un resultado

## ¿Qué es una función?

Ventajas del uso de funciones

## ¿Qué es una función?

### Reutilización de Código

Las funciones permiten reutilizar código, evitando la duplicidad. Esto significa que puedes escribir una función para realizar una tarea específica una vez y luego llamarla desde diferentes partes del programa tantas veces como sea necesario.

## ¿Qué es una función?

### Organización y Legibilidad

Las funciones ayudan a organizar el código en bloques lógicos y manejables. Esto hace que el código sea más fácil de leer y entender, lo que es especialmente útil cuando se trabaja en proyectos grandes o en equipo.

## ¿Qué es una función?

### **Facilidad de Mantenimiento**

Si necesitas modificar la lógica de un proceso específico que se encuentra en una función, solo tienes que hacerlo en un lugar. Esto simplifica el mantenimiento y reduce el riesgo de errores.

## ¿Qué es una función?

### **Abstracción y Enfoque**

Las funciones permiten abstraer detalles de implementación. Puedes usar una función sin necesidad de saber cómo realiza su tarea internamente. Esto te permite centrarte en lo que hace la función y no en cómo lo hace.

## ¿Qué es una función?

### Reducción de Errores

Al tener un código organizado en funciones, es menos probable que cometas errores.

Además, si un error ocurre en una función, generalmente es más fácil de localizar y corregir.

## ¿Qué es una función?

### Facilidad para Pruebas

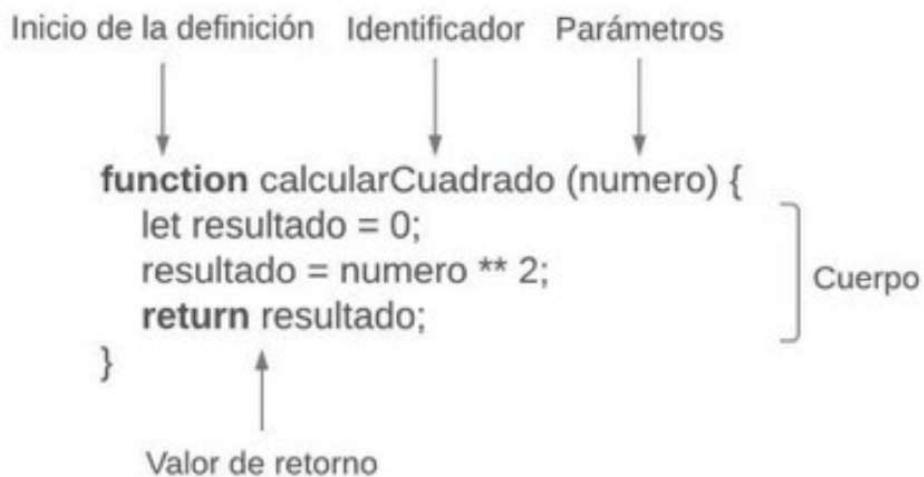
Las funciones hacen más fácil probar tu código. Puedes probar cada función individualmente, lo que ayuda a identificar y solucionar problemas de manera más eficiente.

## ¿Qué es una función?

### Facilidad para Pruebas

Las funciones hacen más fácil probar tu código. Puedes probar cada función individualmente, lo que ayuda a identificar y solucionar problemas de manera más eficiente.

## ¿Qué es una función?



## ¿Qué es una función?

### Ejemplos

- Función saludar
- Función multiplicar sin parámetros

## Invocación de una función

## ¿Qué es una función?

¿Cómo ejecutamos o llamamos a una función?

**Return o no return**

## Return o no return

- Ejemplo de una función sin return
- ¿Qué devuelve una función sin return?
- Ejemplo de una función con return

## Parámetros

## Parámetros

- Definición de parámetro
- ¿Para qué sirve un parámetro?
- Múltiples parámetros
- Orden de parámetros
- Parámetros por defecto (ES6+)
- Arguments (argumentos variables)

## Paso por referencia y por valor

## Parámetros

- Ámbito de variables y tipos de parámetros
- ¿Qué es el paso por valor?
- ¿Qué es el paso por referencia?
- ¿Cómo lo hace Javascript?

## Parámetros

### Ámbito de variables y tipos de parámetros

```
var mensaje = "Fuera de la función";
function mostrarAnuncio() {
    var mensaje = "Dentro de la función";
    console.log(mensaje);
}
mostrarAnuncio();
console.log(mensaje);
```

## Parámetros

### Ámbito de variables y tipos de parámetros

```
function mostrarAnuncio() {  
    var mensaje = "Dentro de la función";  
    console.log(mensaje);  
}  
mostrarAnuncio();  
console.log(mensaje);
```

## Parámetros

Nota mental, evitar el uso de variables globales

## **Ámbito de una función**

## **Ámbito de una función**

¿Qué es el **ámbito** de una función?

## Ámbito de una función

¿Os acordáis de let, const y var?

Ahora cobra más sentido

## Ámbito de una función

```
1 var valor = "global";
2 function funcionlocal() {
3   var valor = "local";
4   return valor;
5 }
6 console.log(valor); // "global"
7 console.log(funcionLocal()); // "local"
8 console.log(valor); // "global"
```

## Ámbito de una función

Es por esto por lo que se recomienda encarecidamente dejar de usar var y usar let y const

## Formas de declarar funciones en Javascript

## Declarar funciones

### Funciones por declaración

Probablemente, la forma más popular de estas tres, y a la que estaremos acostumbrados si venimos de otros lenguajes de programación, es la primera, a la creación de funciones por declaración.

## Declarar funciones

```
function saludar() {  
    return "Hola";  
}  
  
saludar(); // 'Hola'  
typeof saludar; // 'function'
```

## Declarar funciones

### Funciones por expresión

Sin embargo, en Javascript es muy habitual encontrarse códigos donde los programadores «guardan funciones» dentro de variables, para posteriormente «ejecutar dichas variables»

## Declarar funciones

```
// El segundo "saludar" (nombre de la función) se suele omitir: es redundante
const saludo = function saludar() {
    return "Hola";
};

saludo(); // 'Hola'
```

## Declarar funciones

Con este nuevo enfoque, estamos creando una función en el interior de una variable, lo que nos permitirá posteriormente ejecutar la variable (como si fuera una función). Observa que el nombre de la función (en este ejemplo: saludar) pasa a ser inútil, ya que si intentamos ejecutar saludar() nos dirá que no existe y si intentamos ejecutar saludo() funciona correctamente.

## Declarar funciones

### Funciones como objetos

Como curiosidad, debes saber que se pueden declarar funciones como si fueran objetos. Sin embargo, es un enfoque que no se suele utilizar en producción.

Simplemente es interesante saberlo para darse cuenta que en Javascript todo pueden ser objetos:

## Declarar funciones

```
const saludar = new Function("return 'Hola';");  
saludar(); // 'Hola'
```

## Declarar funciones

### Funciones anónimas

Las funciones anónimas o funciones lambda son un tipo de funciones que se declaran sin nombre de función y se alojan en el interior de una variable y haciendo referencia a ella cada vez que queramos utilizarla:

## Declarar funciones

```
// Función anónima "saludo"
const saludo = function () {
    return "Hola";
};

saludo; // f () { return 'Hola'; }
saludo(); // 'Hola'
```

## Declarar funciones

### Callbacks

A grandes rasgos, un callback (llamada hacia atrás) es pasar una función B por parámetro a una función A, de modo que la función A puede ejecutar esa función B de forma genérica desde su código, y nosotros podemos definirlas desde fuera de dicha función

## Declarar funciones

```
// fB = Función B
const fB = function () {
    console.log("Función B ejecutada.");
};

// fA = Función A
const fA = function (callback) {
    callback();
};

fA(fB);
```

## Declarar funciones

### **Funciones autoejecutables o IIFE**

Pueden existir casos en los que necesites crear una función y ejecutarla sobre la marcha. En Javascript es muy sencillo crear funciones autoejecutables.

```
// Función autoejecutable
(function () {
    console.log("Hola!!!");
})();
```

# Declarar funciones

## Funciones anidadas

Dentro del bloque de definición de una función pueden declararse a su vez otras funciones.

Si una función, devuelve otra función, para ejecutarla deberemos utilizar el doble ()() a no ser que se ejecute dentro de la primera

# Declarar funciones

## Funciones anidadas

```
1 function funcionExterna(){
2     var variableExterna = "Hola, soy una variable de la función externa";
3
4     function funcionInterna(){
5         var variableInterna = "y yo soy una variable de la función interna.";
6         console.log(variableExterna + variableInterna); // Accede a ambas variables
7     }
8
9     funcionInterna(); // Llamamos a la función interna dentro de la función externa
10 }
11
12 funcionExterna(); // Llamamos a la función externa
```

# Declarar funciones

## Funciones anidadas devolviendo la interna

```
1 function funcionExterna(){
2     var variableExterna = "Hola, soy una variable de la función externa";
3
4     function funcionInterna(){
5         var variableInterna = "y yo soy una variable de la función interna.";
6         console.log(variableExterna + variableInterna); // Accede a ambas variables
7     }
8
9     return funcionInterna; // Llamamos a la función interna dentro de la función externa
10 }
11
12 funcionExterna(); // Llamamos a la función externa
```

¿Qué es un Clousure?

## ¿Qué es un Closure?

Los closures pueden ser utilizados para crear estructuras de datos privadas y controlar el acceso a ciertas variables, manteniendo una interfaz limpia y bien definida.

## ¿Qué es un Closure?

Se crean cada vez que una función es creada

## ¿Qué es un Closure?

A veces se usan sin saberlo.

Cada vez que una función cualquiera accede a una variable fuera de su contexto, estás usando un closure

## ¿Qué es un Closure?

```
// Ejemplo de Closure con un contador sin usar clases
function crearContador() {
  let contador = 0; // Variable que mantiene el estado del contador

  return {
    incrementar: function() {
      contador++;
      console.log(contador);
    },
    decrementar: function() {
      contador--;
      console.log(contador);
    },
    obtenerValor: function() {
      return contador;
    }
  };
}
```

## ¿Qué es un Clousure?

- La función crearContador define una variable contador y retorna un objeto con tres métodos: incrementar, decrementar, y obtenerValor.
- contador es privada para el ámbito de crearContador y solo puede ser accedida y modificada a través de estos métodos.
- Al llamar a crearContador, se crea un nuevo closure que mantiene su propio estado privado para contador.
- Puedes usar miContador para interactuar con este contador, incrementando, decrementando y obteniendo su valor actual.

<Despedida>

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

Clase 04

JS

<Índice>

**Sentencias de control de flujo**

Tipo de dato string

---

Tipo de dato number

---

Condicional simple y complejo, Switch

---

Bucles I: While, Do /While y For

---

Ejercicios

---

## Tipo de dato string

### Tipo de dato string

- Creación de strings
- Concatenación
- Strings: literals y values
- Caracteres de escape
- Strings largos
- Métodos

## Tipo de dato string

Métodos y propiedades útiles

length	slice	repeat
includes	toUpperCase	trim
indexOf	toLowerCase	split
startsWith	replace	
endsWith	replaceAll	

## Number

## Number

- Enteros y decimales
- Notación científica 5e3
- Infinitos y NaN

## Number

Métodos y propiedades útiles

toFixed

Math.random

Math.round

Number.MAX\_VALUE

Math.floor

Number.MIN\_VALUE

Math.ceil

parseInt

Math.abs

parseFloat

## Condicionales

### Condicionales

- Operadores de comparación
- Booleanos

## Condicionales

if

if - else

if - else if - else

## Condicionales

condicional ternario

condición ? valor\_si\_verdaero : valor\_si\_falso

## Condicionales

switch: break y default

## Bucles I

## Bucles

- While
- Do – While
- For

Ejercicios en el repositorio

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

## Clase 06: Closures, Hoisting y Scope JS

<Índice>

**Closures, Hoisting y Scope**

---

Scope

---

Hoisting

---

Closures

---

## Scope

## Scope

Podemos definirlo como  
El contexto actual de ejecución

## Scope

El contexto en el que los valores y las expresiones son "visibles" o pueden ser referenciados. Si una variable u otra expresión no está "en el Scope-alcance actual", entonces no está disponible para su uso. Los Scope también se pueden superponer en una jerarquía, de modo que los Scope secundarios tengan acceso a los ámbitos primarios, pero no al revés.

## Scope

Una función sirve como un cierre en JavaScript y, por lo tanto, crea un ámbito, de modo que (por ejemplo) no se puede acceder a una variable definida exclusivamente dentro de la función desde fuera de la función o dentro de otras funciones. Por ejemplo, lo siguiente no es válido:

## Scope

```
function exampleFunction() {  
    var x = "declarada dentro de la función"; // x  
    solo se puede utilizar en exampleFunction  
    console.log("funcion interna");  
    console.log(x);  
}  
  
console.log(x); // error
```

## Scope

Sin embargo, el siguiente código es válido debido a que la variable se declara fuera de la función, lo que la hace global:

```
var x = "función externa declarada";  
  
exampleFunction();  
  
function exampleFunction() {  
    console.log("funcion interna");  
    console.log(x);  
}  
  
console.log("funcion externa");  
console.log(x);
```

## Scope

¿Qué es la cadena de scopes?

## Scope

¿Una variable puede estar  
definida dos veces?

## Scope

### Tipos de scopes

- Scope global
- Scope local
  - scope de función
  - scope de bloque (hoisting)

## Scope

¿Mejor usar scope global  
o scope local?

Siempre se recomienda usar el scope más  
reducido posible. Ahorramos memoria

# Scope

Hagamos este ejercicio

```
1  var fruta = 'manzana';
2
3  function comer() {
4    var fruta = 'banana';
5
6    function lavar() {
7      console.log(`Lavando ${fruta}`);
8    }
9
10   lavar();
11   console.log(`Comiendo ${fruta}`);
12 }
13
14 comer();
15
```

# Scope

```
var fruta = 'manzana';

function comer() {
  var fruta = 'banana';

  function lavar() {
    console.log(`Lavando ${window.fruta}`);
  }

  lavar();
  console.log(`Comiendo ${fruta}`);
}

comer();
```

¿Cómo  
accedemos a las  
variables  
globales?

# Scope

¿Cómo vemos los diferentes scopes en nuestro navegador?

# Scope

The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. A file named 'prueba.js' is open, containing the following code:

```
1 var fruta = 'manzana';
2
3 function comer() {
4     var fruta = 'banana'; fruta = "banana"
5
6     function lavar() { lavar = f lavar()
7         console.log(`Lavando ${window.fruta}`)
8     }
9
10    lavar(); lavar = f lavar()
11    console.log(`Comiendo ${fruta}`)
12 }
13
14 comer();
```

The code is highlighted with syntax coloring. A yellow box highlights the line `console.log(`Comiendo \${fruta}`)`.

In the bottom right corner of the code editor, there is a status bar that says '(1) Debugger paused'.

To the right of the code editor is the DevTools sidebar. It shows the following state:

- Debugger status: '(1) Debugger paused'
- Threads
- Watch
- Breakpoints
- Pause options:  Pause on uncaught exceptions,  Pause on caught exceptions
- Scope
- Local scope (under Window):
  - this: Window
  - fruta: "banana"
  - lavar: f lavar()
- Global scope (under Window):
  - 0: Window {window: Window, self: Wi
  - JSCompiler\_renameProperty: f (t,e)
  - alert: f alert()
  - atob: f atob()
  - blur: f blur()
  - btoa: f btoa()
  - caches: CacheStorage {}

## Hoisting

## Hoisting

Característica rara y poco intuitiva

# Hoisting

¿Qué hace esto?

```
1 console.log(firstName);
2 var firstName = 'Bienve';
```

# Hoisting

Hoisting = Elevación

Cada vez que declaremos variables con var, estas se elevarán al inicio de su scope.

Es como si las declaráramos siempre al principio.

**Pero solo las declaraciones, no las asignaciones.**

## Hoisting

Solo las variables declaradas  
con var sufren de hoisting

## Hoisting

Ojo, las funciones también. Depende de  
como sean declaradas.  
¿Os acordáis de las formas de definir  
una función?

## Declarar funciones

### Funciones por declaración

Probablemente, la forma más popular de estas tres, y a la que estaremos acostumbrados si venimos de otros lenguajes de programación, es la primera, a la creación de funciones por declaración.

## Declarar funciones

```
function saludar() {  
    return "Hola";  
}  
  
saludar(); // 'Hola'  
typeof saludar; // 'function'
```

## Declarar funciones

```
1  saludar();  
2  
3  function saludar() {  
4    console.log('Hola');  
5  }  
6
```

## Declarar funciones

### Funciones por expresión

Sin embargo, en Javascript es muy habitual encontrarse códigos donde los programadores «guardan funciones» dentro de variables, para posteriormente «ejecutar dichas variables»

## Declarar funciones

```
// El segundo "saludar" (nombre de la función) se suele omitir: es redundante
const saludo = function saludar() {
    return "Hola";
};

saludo(); // 'Hola'
```

## Declarar funciones

```
1  saludo();
2
3  const ·saludo· = ·function·( )··{·
4      ···console.log('Hola')
5  }
```

# Hoisting

## ¿Qué sufre de hoisting?

var  
funciones por declaración

De las cosas más raras de JS

## Repasso

	var	let	const
compatibilidad	✓	BABEL	BABEL
scope (ámbito)	función	bloque	bloque
re-asignación	✓	✓	✗
re-declaración	✓	✗	✗
declaración sin valor inicial	✓	✓	✗
propiedad del obj. global	✓	✗	✗
hoisting	declaración	TDZ	TDZ

## Clousures

## Clousures

### Necesitamos tres cosas

1. Función anidada
2. Variable descrita en la función padre que sea utilizada por la función anidada
3. Invocar a la función interna desde otro scope

## Clousures

```
1  function crearContador() {
2    let contador = 0;
3
4    return function incrementar() {
5      contador += 1;
6      return contador;
7    }
8  }
9
10 const c1 = crearContador();
11
```

## Clousures

Vamos a incrementar su funcionalidad

decrementar

obtener valor

## **Clousures**

Vamos a incrementar su funcionalidad  
contador personalizado

**<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**

# {JS}

Clase 08

JS

<Índice>

## Introducción a Arrays y Objetos

Tipos por referencia

---

Arrays

---

Objetos

---

For of

---

For in

---

Continue y Break

---

**Tipos por referencia**

**Arrays**

## Objetos

## Arrays: For of vs For

**Objetos: For in**

**Objetos: For in**

## **Continue y break**

**<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

Clase 09

JS

<Índice>

## Arrow Functions

¿Qué son?

---

Diferencias con funciones originales

---

Definición de Arrow Functions

---

**¿Qué son?**

**Diferencias con funciones originales**

## Diferencias

- No tiene la palabra function
- No tiene acceso a this (ya hablaremos más adelante sobre this)
- No tiene el objeto arguments como en las funciones tradicionales
- Siempre son anónimas
- No son afectadas por el Hoisting, es decir, no pueden llamarse antes de declararse

<Despedida>

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

Clase 10

JS

<Índice>

## Funciones útiles para Arrays I

Agregar elementos

---

Eliminar elementos

---

Vaciar arrays

---

Combinar y dividir

---

Join y order

---

**Agregar elementos**

**Eliminar elementos**

**Vaciar arrays**

**Combinar**

**Dividir**

**Join**

## **Order**

**<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

Clase 11

JS

<Índice>

## Funciones útiles para Arrays II

Ordenar arrays de referencias

---

Búsqueda de primitivos

---

Búsqueda de referencias

---

Spread operator

---

**order array de referencias**

**Búsqueda de primitivos  
includes, indexOf, lastIndexOf**

## Búsqueda de objetos por referencia `find`

`Spread operator`

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**

# {JS}

Clase 12

JS

<Índice>

## Funciones útiles para Arrays II

Spread operator

---

Every

---

Some

---

Filter

---

Map

---

Reduce

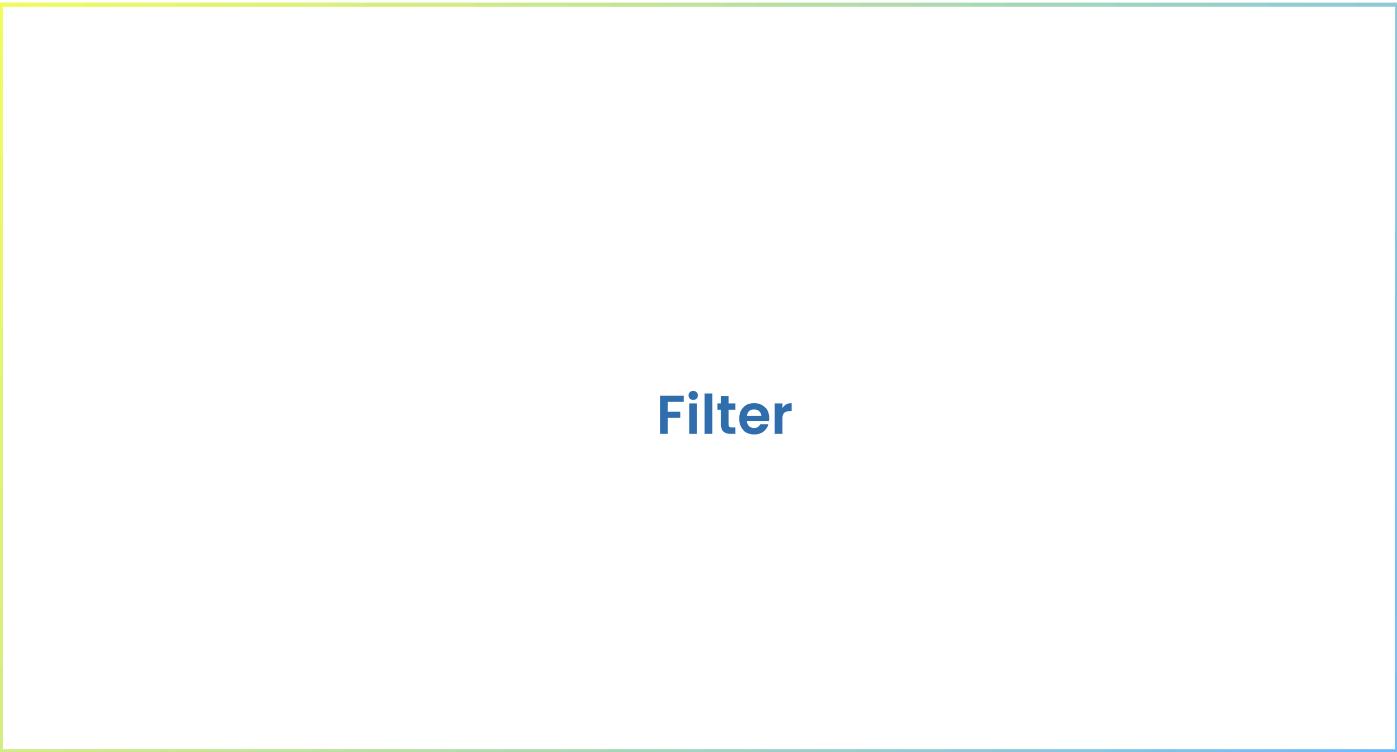
---

**Every**

**Some**



**Map**



**Filter**

## **Reduce**

**<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**

# {JS}

Clase 14

JS

<Índice>

Prototipos

Repaso de la herencia

---

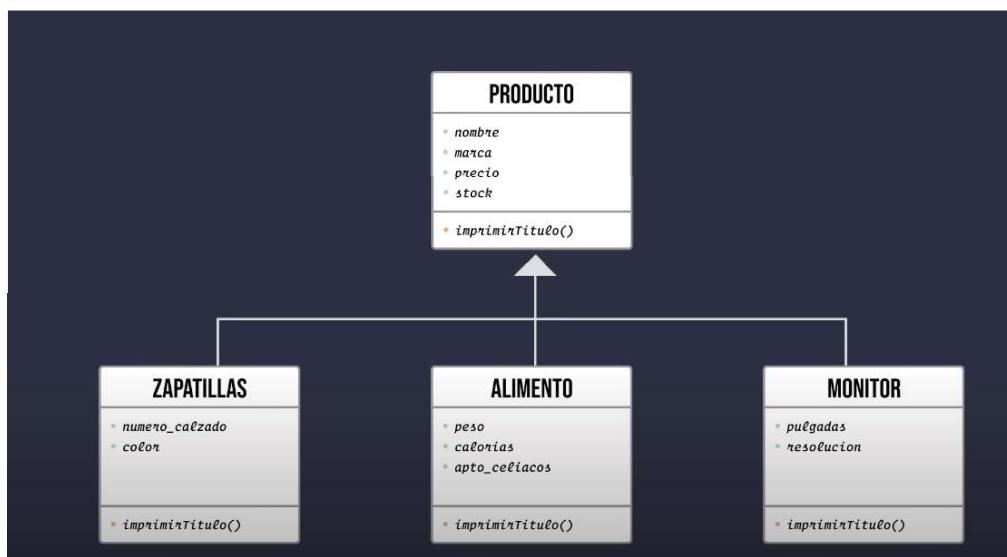
Prototipos

---

# Herencia

## Herencia

- Herencia por clases típica



## Herencia

### Ventajas de la herencia

- Reutilización de código
- Mantenimiento más sencillo
- Jerarquía clara y estructurada
- Extensibilidad
- Polimorfismo
- Abstracción y encapsulamiento
- Modelado del mundo real

## Herencia

¿Cómo funciona la herencia por prototipos en JS?

No funciona como  
tradicionalmente estamos  
acostumbrados. Veremos las  
clases de JS en otra clase

## Prototipos

## Prototipo

¿Qué es un prototipo?

Es como un delegado. Alguien a quien le damos una responsabilidad y la confiamos en él

## Prototipo

Veámoslo en código

## Prototipo

En JS no se denomina herencia normal, si no que se denomina herencia por prototipos.

O mejor todavía **delegación de objetos**

## Prototipo

Los prototipos son dinámicos ya que son objetos normales en JS como los que usamos normalmente.

La búsqueda de métodos y propiedades se hace en tiempo de ejecución

## Prototipo

Añadimos propiedades a un prototipo

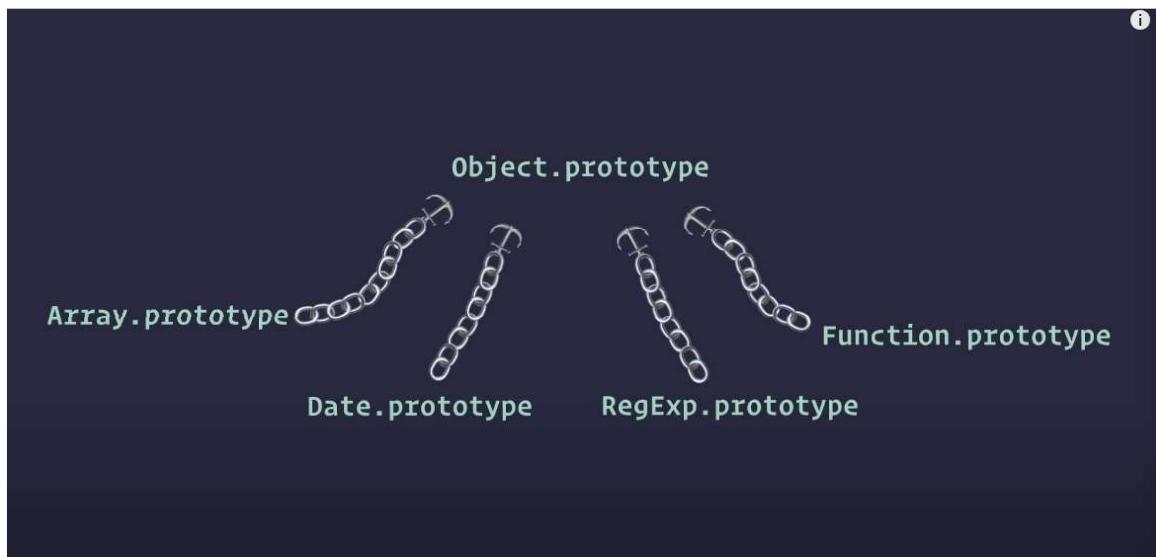
## Prototipo

Añadimos propiedad a un objeto que comparte con su prototipo

## Prototipo

Reescribir propiedades de un objeto  
`toString`

# Prototipo



# Prototipo

El final de la cadena: null

## Prototipo

Accedemos a una propiedad de un objeto  
¿Qué ocurre por detrás?

## Prototipo

La cadena de prototipos solo ocurre  
cuando queremos acceder a una  
propiedad o método.

Si sobrescribo una propiedad solo se hace  
en el objeto en el que la hago

## Prototipo

¿Podemos saber si un objeto es un prototipo de otro objeto?

## Prototipo

### ¿Cómo definimos prototipos?

- Objetos literales: funciones, objetos o arrays: crear un objeto
- `Object.create(proto)`
- Funciones creadoras o clase (otra clase)
- `setPrototypeOf` y `__proto__` (no usar)

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

## Clase 16

JS

<Índice>

### Clases en JS

Definición de clases

Métodos privados

Métodos estáticos

Getters y setters

Herencia

## “Clases” en Javascript

- Introducidas en ECMAScript 2015 o ES6
- Recordemos que JS se basa en prototipos
- En prototipos existe la “herencia” o delegación
- Sugar sintax...
- Al final no dejan de ser funciones constructoras
- Ventajas de la POO

## Definición de clases

- Recordemos las funciones constructoras
- Declaración de una “instancia” de función constructora
- ¿Dónde viven los métodos en un objeto de función constructora?
- Añadámoslo al prototipo de usuario
- Comprobemos donde vive ahora la función

## Nuestra primera clase

- Convertámoslo en clase
- ¿Qué es el constructor?
- Declaremos nuestro primer objeto
- Declaremos el primer método saludar
- ¿Dónde vive?
- Arreglémoslo
- ¿Dónde vive ahora el método?

## Métodos o propiedades privados

- ¿Cómo hacíamos atributos privados en funciones constructoras?
- ¿Cómo lo hacemos con clases?
- Declaremos nuestra primera propiedad privada
- Peculiaridad de Chrome para devs
- La importancia de la visibilidad o no de propiedades
- Funciona para propiedades y métodos

## Métodos estáticos

- ¿Qué son?
- Ya lo hemos usado
- ¿Dónde viven?
- Hagamos un ejemplo
- Sirve tanto para métodos como para propiedades

## Getters y Setters

- ¿Qué son?
- Su importancia
- Su uso, validación por ejemplo
- Ejemplo, edad de un alumno

## **Herencia**

- ¿Qué es?
- Su importancia
- Ejemplo de uso
- Llamar al constructor del padre
- Sobreescribir métodos, saludar

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

Clase 19

JS

<Índice>

## Prototipos

¿Qué es un módulo?

---

¿Qué problema resuelven?

---

Nuestro primer export

---

Nuestro primer import

---

Probemos con vite

---

## ¿Qué es un módulo?

A partir de ECMAScript se introduce una característica nativa denominada Módulos ES (ESM), que permite la importación y exportación de fragmentos de datos entre diferentes ficheros

Javascript, eliminando las desventajas que teníamos hasta ahora y permitiendo trabajar de forma más flexible en nuestro código Javascript.

## ¿Qué problema resuelve?

Uno de los principales problemas que ha ido arrastrando Javascript desde sus inicios es la dificultad de organizar de una forma adecuada una aplicación grande, con muchas líneas de código. En muchos lenguajes de programación, cuando un programa crece, se comienza a estructurar en funciones. Posteriormente, se traslada a clases, que contienen variables (propiedades) y funciones (métodos). De esta forma organizamos de forma más lógica el código de nuestro programa. Sin embargo, no será suficiente.

# Import y Export

Declaración	Descripción
export	Pone los datos indicados (variables, funciones, clases...) a disposición de otros ficheros
import	Incorpora datos (variables, funciones, clases...) desde otros ficheros .js al código actual.

## Antes de usar módulos

```
<script type="module">
    import { nombre } from "./file.js";
</script>
```

# Nuestro primer export

Forma	Descripción
<code>export ...</code>	Declara un elemento o dato, a la vez que lo añade al módulo de exportación.
<code>export { name }</code>	Añade el elemento <code>name</code> al módulo de exportación.
<code>export { name as newName }</code>	Añade el elemento <code>name</code> al módulo de exportación con el nombre <code>newName</code> .
<code>export { n1, n2, n3... }</code>	Añade los elementos indicados ( <code>n1</code> , <code>n2</code> , <code>n3</code> ...) al módulo de exportación.
<code>export * from "./file.js"</code>	Añade todos los elementos del módulo de <code>file.js</code> al módulo de exportación.
<code>export default ...</code>	Declara un elemento y lo añade como módulo de exportación <b>por defecto</b> .

# Nuestro primer export

```
let number = 42;
const hello = () => "Hello!";
const goodbye = () => "¡Adiós!";
class CodeBlock { }

export { number };           // Se crea un módulo y se añade number
export { hello, goodbye as bye }; // Se añade saludar y despedir al módulo
export { hello as greet };    // Se añade otroNombre al módulo
```

## Nuestro primer export

```
let number = 42;
const hello = () => "Hello!";
const goodbye = () => "¡Adiós!";
class CodeBlock { };

export {
  number,
  hello,
  goodbye as bye,
  hello as greet
};
```

## Nuestro primer import

Forma	Descripción
import { nombre } from "./file.js"	Importa el elemento <b>nombre</b> de file.js.
import { nombre as newName } from "./file.js"	Importa el elemento <b>nombre</b> de file.js como <b>newName</b> .
import { n1, n2... } from "./file.js"	Importa los elementos indicados desde file.js.
import nombre from "./file.js"	Importa el elemento <b>por defecto</b> de file.js como <b>nombre</b> .
import * as name from "./file.js"	Importa todos los elementos de file.js en el objeto <b>name</b> .
import "./file.js"	Ejecuta el código de file.js. No importa ningún elemento.
import { name } from "https://web.com/file.js"	Descarga el fichero e importa el elemento <b>name</b> de su módulo.

## Nuestro primer import

```
import { nombre } from "./file.js";
import { number, element } from "./file.js";
import { brand as brandName } from "./file.js";
```

## Nuestro primer import

```
import nombre from "./math.js";
```

## **Probemos con Vite**

**<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

## Clase 20

JS

# <Índice>

## Prototipos

Manipulación del DOM

---

¿Qué es un selector?

---

`getElementById`, `getElementsByName`

---

`getElementsByTagName` y `getElementsByName`

---

`querySelector` y `querySelectorAll`

---

`HTMLCollection` vs `NodeList` => Solución

---

## **Manipulación del DOM**

**¿Qué es un selector?**

**getElementById**

**getElementsByClassName**

**getElementsByTagName**

**getElementsByName**

**querySelector**

**querySelectorAll**

## Búsquedas acotadas

## HTMLCollection vs NodeList

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

## Clase 21

JS

<Índice>

### Creación de elementos del DOM

Repasemos el DOM

---

`createElement`

---

`createComment`

---

`createTextNode`

---

`cloneNode`

---

`isConnected`

---

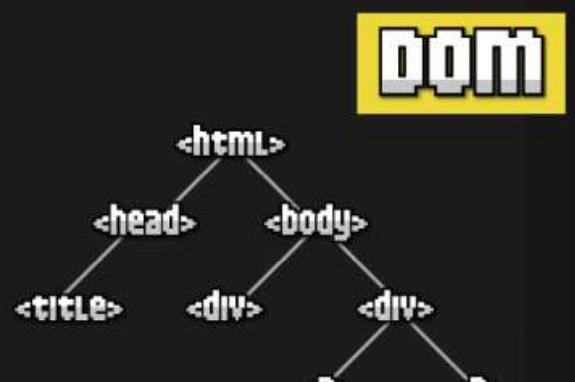
`Primer appendChild y Contenido`

## Repasemos el DOM

## Repasemos el DOM

```
<html>
<head>
  <title>Title</title>
</head>
<body>
  <div></div>
  <div>
    <p>Párrafo 1</p>
    <p>Párrafo 2</p>
  </div>
</body>
</html>
```

**HTML**



# Repasemos el DOM

## Tipos de elementos

Tipo Element

Tipo Node

# Repasemos el DOM

Tipo de dato genérico	Tipo específico	Etiqueta	Descripción	+ info
ELEMENT HTMLElement	HTMLDivElement	<div>	Etiqueta divisoria (en bloque).	<a href="#">Elemento &lt;div&gt;</a>
ELEMENT HTMLElement	HTMLSpanElement	<span>	Etiqueta divisoria (en línea).	<a href="#">Elemento &lt;span&gt;</a>
ELEMENT HTMLElement	HTMLImageElement	<img>	Imagen.	<a href="#">Elemento &lt;img&gt;</a>
ELEMENT HTMLElement	HTMLAudioElement	<audio>	Contenedor de audio.	<a href="#">Elemento &lt;audio&gt;</a>

## createElement

### Create Element

```
const div = document.createElement("div");      // Creamos un <div></div>
const span = document.createElement("span");    // Creamos un <span></span>
const img = document.createElement("img");      // Creamos un <img>
```

**createComment**

**createTextNode**

**cloneNode**

**isConnected**

**createDocumentFragment**

**Primera aproximación a contenido**

## **Primer testeo de textContent y innerHTML**

**Nuestro primer appendChild**

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

Clase 22

JS

<Índice>

## Gestión de atributos del DOM

Repasemos los atributos

---

Acceso a atributos básicos

---

Acceso a cualquier atributo

---

Modificar atributos

---

Casos especiales

---

`toggleAttribute`

## Repasemos los atributos

### Repasemos los atributos

```
<div class="container" data-attr="value">
  <button disabled>Avisar</button>
</div>
```

Atributo – Valor – Booleano

## **Repasemos los atributos**

Atributos básicos

Atributos específicos

Atributos para todas las etiquetas

Cuidado con el class: tiene sus propios métodos que ya veremos

## **Acceso a atributos básicos**

```
const element = document.querySelector("div");    // <div class="container"></div>  
  
element.id = "page";                // <div id="page" class="container"></div>  
element.style = "color: red";      // <div id="page" class="container" style="color: red"></div>  
element.className = "data";        // <div id="page" class="data" style="color: red"></div>
```

## Acceso a cualquier atributo

Métodos	Descripción
<b>BOOLEAN</b> hasAttributes()	Indica si el elemento tiene atributos HTML.
<b>BOOLEAN</b> hasAttribute(attr)	Indica si el elemento tiene el atributo HTML attr.
<b>ARRAY</b> getAttributeNames()	Devuelve un <b>ARRAY</b> con los atributos del elemento.
<b>STRING</b> getAttribute(attr)	Devuelve el valor del atributo attr del elemento o <b>NULL</b> si no existe.

```
<div id="page" class="info data dark" data-number="5"></div>
```



```
const element = document.querySelector("#page");

element.hasAttributes();           // true (tiene 3 atributos)
element.hasAttribute("data-number"); // true (data-number existe)
element.hasAttribute("disabled");   // false (disabled no existe)

element.getAttributeNames();        // ["id", "data-number", "class"]
element.getAttribute("id");         // "page"
```



## Modificar atributos

Métodos	Descripción
setAttribute(attr, value)	Añade o cambia el atributo <code>attr</code> al valor <code>value</code> del elemento HTML.
toggleAttribute(attr, force)	Añade atributo <code>attr</code> si no existe, si existe lo elimina.
removeAttribute(attr)	Elimina el atributo <code>attr</code> del elemento HTML.

```
<div id="page" class="info data dark" data-number="5"></div>
```



```
const element = document.querySelector("#page");

element.setAttribute("data-number", "10");    // Cambiar data-number a 10
element.removeAttribute("id");                // Elimina el atributo id
element.setAttribute("id", "page");           // Vuelve a añadirlo
```



## Casos especiales: booleanos

Hay que hablar de un caso especial, que es el que comentamos en el que podemos establecer atributos HTML que son booleanos, es decir, que no tienen indicado ningún valor.

## Casos especiales: booleanos

Si esto lo hacemos con el método `setAttribute()` y le indicamos un booleano, no tendremos exactamente lo que buscamos. Recuerda que los atributos HTML son siempre de tipo string.

## Casos especiales: booleanos

```
const button = document.querySelector("button");

button.setAttribute("disabled", true);    // ✗ <button disabled="true">Clickme!</button>
button.disabled = true;                  // ✓ <button disabled>Clickme!</button>
button.setAttribute("disabled", "");     // ✓ <button disabled>Clickme!</button>
```

## Casos especiales: booleanos

Recuerda que atributo HTML no es lo mismo que propiedad Javascript, aunque muchos frameworks o librerías Javascript simplifican para que parezcan que son la misma cosa.

**toggleAttribute**

Para atributos booleanos es más fácil usar:  
**toggleAttribute**

```
button.toggleAttribute("disabled");           // Como ya existe "disabled", lo elimina  
button.toggleAttribute("hidden");            // Como no existe "hidden", lo añade
```

<Despedida>

Email

[bienvenidosaez@gmail.com](mailto:bienvenidosaez@gmail.com)

Instagram

[@bienvenidosaez](#)

Youtube

[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)

# {JS}

Clase 23

JS

## API de classList y dataset

Gestión de clases

---

Gestión de data attributes

---

<Índice>

## Repasemos los atributos

### Repasemos los atributos

```
<div class="container" data-attr="value">
  <button disabled>Avisar</button>
</div>
```

Atributo – Valor – Booleano

## Repasemos los atributos

Atributos básicos

Atributos específicos

Atributos para todas las etiquetas

Cuidado con el class: tiene sus propios métodos que ya veremos

## Acceso a atributos básicos

```
const element = document.querySelector("div");    // <div class="container"></div>  
  
element.id = "page";                // <div id="page" class="container"></div>  
element.style = "color: red";      // <div id="page" class="container" style="color: red"></div>  
element.className = "data";        // <div id="page" class="data" style="color: red"></div>
```

## Acceso a cualquier atributo

Métodos	Descripción
<b>BOOLEAN</b> hasAttributes()	Indica si el elemento tiene atributos HTML.
<b>BOOLEAN</b> hasAttribute(attr)	Indica si el elemento tiene el atributo HTML attr.
<b>ARRAY</b> getAttributeNames()	Devuelve un <b>ARRAY</b> con los atributos del elemento.
<b>STRING</b> getAttribute(attr)	Devuelve el valor del atributo attr del elemento o <b>NULL</b> si no existe.

## classList

## classList

### className y classList

la propiedad `.className` viene a ser la modalidad directa y rápida de utilizar el getter `.getAttribute("class")` y el setter `.setAttribute("class", value)`.

CONQUERBLOCKS

## classList

```
const div = document.querySelector(".element");

// Obtener clases CSS
div.className; // "element shine dark-theme"
div.getAttribute("class"); // "element shine dark-theme"

// Modificar clases CSS
div.className = "element shine light-theme";
div.setAttribute("class", "element shine light-theme");
```

CONQUERBLOCKS

# classList

<b>ARRAY</b>	.classList	Devuelve la lista de clases del elemento HTML.
<b>NUMBER</b>	.classList.length	Devuelve el número de clases del elemento HTML.
<b>STRING</b>	.classList.item(n)	Devuelve la clase número <b>n</b> del elemento HTML. <b>NULL</b> si no existe.
<b>BOOLEAN</b>	.classList.contains(clase)	Indica si la <b>clase</b> existe en el elemento HTML.
Acciones sobre clases		
	.classList.add(c1, c2, ...)	Añade las clases <b>c1, c2...</b> al elemento HTML.
	.classList.remove(c1, c2, ...)	Elimina las clases <b>c1, c2...</b> del elemento HTML.
<b>BOOLEAN</b>	.classList.toggle(clase)	Si la <b>clase</b> no existe, la añade. Si no, la elimina.
<b>BOOLEAN</b>	.classList.toggle(clase, expr)	Si <b>expr</b> es <b>true</b> , añade la clase. Si es <b>false</b> , la elimina.
<b>BOOLEAN</b>	.classList.replace(old, new)	Reemplaza la clase <b>old</b> por la clase <b>new</b> .

CONQUERBLOCKS

# data attributes

## classList

Algo parecido pasa con los data attributes  
elemento.dataset

CONQUERBLOCKS

## classList

Obtener atributos

```
// Usando getAttribute
var valor = document.getElementById('miElemento').getAttribute('data-mi-dato');

// Usando dataset
var valor = document.getElementById('miElemento').dataset.miDato;
```

CONQUERBLOCKS

# classList

## Setear atributos

```
// Usando setAttribute  
document.getElementById('miElemento').setAttribute('data-mi-dato', 'nuevoValor');  
  
// Usando dataset  
document.getElementById('miElemento').dataset.miDato = 'nuevoValor';
```

CONQUERBLOCKS

# classList

## Eliminar atributos

```
document.getElementById('miElemento').removeAttribute('data-mi-dato');
```

CONQUERBLOCKS

# classList

## Notas importantes:

- Los nombres de los data atributos se convierten a camelCase cuando se acceden a través de dataset. Por ejemplo, data-mi-dato se convierte en miDato en el dataset.
- Los data atributos son útiles para almacenar información adicional que no se utiliza para otros fines, como estilos o scripts. No deben utilizarse para almacenar información que debería estar visible o accesible a los usuarios o para otra funcionalidad que HTML ya proporciona por otros medios.

CONQUERBLOCKS

<Despedida>

Email

[bienvenidosaez@gmail.com](mailto:bienvenidosaez@gmail.com)

Instagram

[@bienvenidosaez](https://www.instagram.com/bienvenidosaez)

Youtube

[youtube.com/bienvenidosaez](https://www.youtube.com/bienvenidosaez)

# {JS}

Clase 24

JS

<Índice>

**Insertar y movernos por el DOM**

**Propiedades de un elemento**

---

**Insertar elementos en el DOM**

---

**Navegar por los elementos del DOM**

## **Propiedades de un elemento**

### **Propiedades de un elemento**

`.textContent`

Sirve para obtener y asignar el contenido de texto de un elemento

## Propiedades de un elemento

### .innerText

Obtiene solo el texto visible después de renderizar el documento y aplicar los estilos CSS

## Propiedades de un elemento

### .innerHTML

Sirve para obtener o cambiar el contenido de una etiqueta en formato HTML. Esta propiedad si renderiza código HTML

## Propiedades de un elemento

### .innerHTML

Ten mucho cuidado a la hora de insertar contenido HTML utilizando .innerHTML puesto que si añades contenido que provenga del usuario sin revisarlo, podrían insertar HTML que realice acciones dañinas como inyección de código malicioso.

## Propiedades de un elemento

### .setHTML

“Sanitiza” previamente el contenido antes de introducirlo.

**Experimental**

## Propiedades de un elemento

`.outerHTML`

Muy similar a `innerHTML` pero esta propiedad devuelve también la etiqueta sobre la que estamos ejecutando

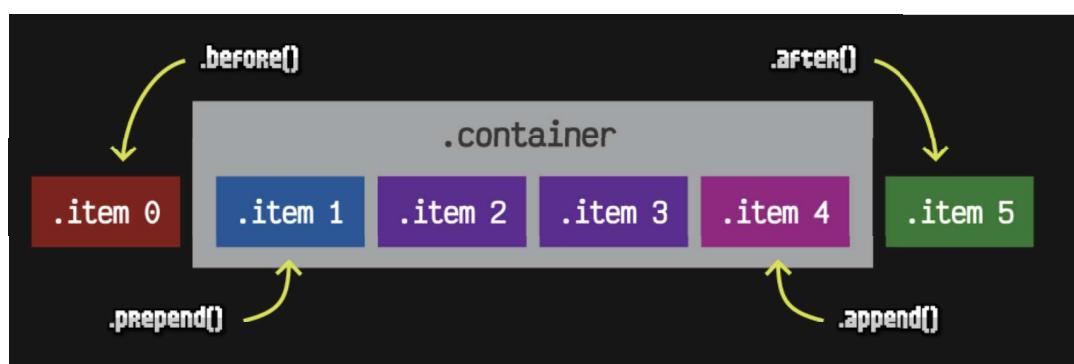
## Insertar elementos en el DOM API de nodos

# API de nodos

Métodos	Descripción
<code>NODE .appendChild(node)</code>	Añade como hijo el nodo <code>node</code> . Devuelve el nodo insertado.
<code>NODE .removeChild(node)</code>	Elimina y devuelve el nodo hijo <code>node</code> .
<code>NODE .replaceChild(new, old)</code>	Reemplaza el nodo hijo <code>old</code> por <code>new</code> . Devuelve <code>old</code> .
<code>NODE .insertBefore(new, node)</code>	Inserta el nodo <code>new</code> antes de <code>node</code> y como hijo del nodo actual.
<code>NODE .insertBefore(new, NULL )</code>	Inserta el nodo <code>new</code> después del último nodo hijo. Equivale a <code>.appendChild()</code> .

## Insertar elementos en el DOM API de elementos

Métodos	Descripción
.before()	Añade el nuevo elemento justo antes.
.after()	Añade el nuevo elemento justo después.
.prepend()	Se añade el nuevo elemento antes del primer hijo.
.append()	Se añade el nuevo elemento después del último hijo.
.replaceChildren()	Elimina todos los hijos y los sustituye por el nuevo elemento.
.replaceWith()	Se sustituye por el nuevo elemento.
.remove()	Elimina el propio elemento.



# Insertar elementos en el DOM

## API de inserción adyacente

Métodos	Descripción
<code>ELEMENT .insertAdjacentElement(position, element)</code>	Inserta el <code>element</code> en la posición <code>position</code> . Si falla, <code>NULL</code> .
<code>.insertAdjacentHTML(position, htmlCode)</code>	Inserta el código HTML de <code>htmlCode</code> en la posición <code>position</code> .
<code>.insertAdjacentText(position, text)</code>	Inserta el texto <code>text</code> en la posición <code>position</code> .



**CONQUERBLOCKS**

```
const container = document.querySelector(".container");

// Creamos un nuevo <div>Ejemplo</div>
const div = document.createElement("div");
div.textContent = "Ejemplo";

container.insertAdjacentElement("beforebegin", div);
// A) <div>Ejemplo</div> <div class="container">container</div>

container.insertAdjacentElement("afterbegin", div);
// B) <div class="container"> <div>Ejemplo</div> container</div>

container.insertAdjacentElement("beforeend", div);
// C) <div class="container">container <div>Ejemplo</div> </div>

container.insertAdjacentElement("afterend", div);
// D) <div class="container">App</div> <div>Ejemplo</div>
```

**CONQUERBLOCKS**

# Navegar por el DOM por elementos

Propiedades de elementos HTML	Descripción
<b>ARRAY</b> children	Devuelve una lista de elementos HTML hijos.
<b>ELEMENT</b> parentElement	Devuelve el padre del elemento o <b>NULL</b> si no tiene.
<b>ELEMENT</b> firstElementChild	Devuelve el primer elemento hijo.
<b>ELEMENT</b> lastElementChild	Devuelve el último elemento hijo.
<b>ELEMENT</b> previousElementSibling	Devuelve el elemento hermano anterior o <b>NULL</b> si no tiene.
<b>ELEMENT</b> nextElementSibling	Devuelve el elemento hermano siguiente o <b>NULL</b> si no tiene.

```
<html>
  <body>
    <div id="app">
      <div class="header">
        <h1>Titular</h1>
      </div>
      <p>Párrafo de descripción</p>
      <a href="/">Enlace</a>
    </div>
  </body>
</html>
```

```
document.body.children.length; // 1
document.body.children; // <div id="app">
document.body.parentElement; // <html>

const app = document.querySelector("#app");

app.children; // [div.header, p, a]
app.firstElementChild; // <div class="header">
app.lastElementChild; // <a href="/">

const a = app.querySelector("a");

a.previousElementSibling; // <p>
a.nextElementSibling; // null
```

CONQUERBLOCKS

## Navegar por el DOM por nodos

Propiedades de nodos HTML	Descripción
<b>ARRAY</b> childNodes	Devuelve una lista de nodos hijos. Incluye nodos de texto y comentarios.
<b>NODE</b> parentNode	Devuelve el nodo padre del nodo o <b>NULL</b> si no tiene.
<b>NODE</b> firstChild	Devuelve el primer nodo hijo.
<b>NODE</b> lastChild	Devuelve el último nodo hijo.
<b>NODE</b> previousSibling	Devuelve el nodo hermano anterior o <b>NULL</b> si no tiene.
<b>NODE</b> nextSibling	Devuelve el nodo hermano siguiente o <b>NULL</b> si no tiene.

CONQUERBLOCKS

```
document.body.childNodes.length; // 3
document.body.childNodes;      // [text, div#app, text]
document.body.parentNode;      // <html>

const app = document.querySelector("#app");

app.childNodes;                // [text, div.header, text, p, text, a, text]
app.firstChild.textContent;    // "
app.lastChild.textContent;    // "

const a = app.querySelector("a");

a.previousSibling;            // #text
a.nextSibling;                // #text
```

CONQUERBLOCKS

## Ejercicios

CONQUERBLOCKS

«Despedida»

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

Clase 26

JS

<Índice>

## Prototipos

¿Qué es un evento?

---

¿Cómo capturarlos?

---

Eventos mediante HTML

---

Eventos mediante Javascript

---

Eventos con addEventListener

---

El objeto Event

---

## ¿Qué es un evento?

## ¿Qué es un evento?

Los eventos son el idioma en el que JavaScript entiende las acciones del usuario. Es decir, son el mecanismo por el que se consigue establecer una comunicación bidireccional y en tiempo real entre la aplicación y los usuarios.

Tal y como su nombre indica, un evento no es más que un suceso que se ha producido en la página, normalmente provocado por la acción del usuario.

## ¿Qué es un evento?

Lo más interesante de este mecanismo es que es asíncrono, por tanto, no hay que esperar por él. Simplemente se prepara el código para que cuando se produzca el evento el programa lo capture y ejecute la lógica que interese.

Los eventos se asocian a elementos concretos del DOM. En el argot de los programadores se dice que «escuchamos» a ese elemento, o lo que es lo mismo, tenemos un programa preparado para capturar el evento que lanza ese elemento.

## ¿Cómo los capturamos?

## ¿Cómo los capturamos?

Mediante HTML

Mediante Javascript

Mediante addEventListener

**Mediante HTML**

```
<script>
  function doTask() {
    alert("Hello!");
  }
</script>

<button onClick="doTask()">Saludar</button>
```

```
<script src="tasks.js"></script>
<button onClick="doTask()">Saludar</button>
```

## ¿problemas?

Ahora aparece un nuevo problema que quizás puede que aún no sea muy evidente. En nuestro <button> estamos haciendo referencia a una función llamada doTask() que, aparentemente, confiaremos en que se encuentra declarada dentro del fichero tasks.js.

Esto podría convertirse en un problema, si posteriormente, o dentro de cierto tiempo, nos encontramos modificando código en el fichero tasks.js y le cambiamos el nombre a la función doTask(), ya que podríamos olvidar que hay una llamada a una función Javascript en uno (o varios) ficheros .html.

## Mediante Javascript

Existe una forma de gestionar eventos Javascript sin necesidad de hacerlo desde nuestros ficheros .html. No obstante, se trata de una «trampa», puesto que seguimos haciéndolo desde HTML, sólo que ese HTML se crea desde Javascript, y nos permite llevarlo a los ficheros .js.

```
<button>Saludar</button>

<script>
const button = document.querySelector("button");
button.onclick = function() {
    alert("Hello!");
}
</script>
```

Observa que en este caso, en lugar de añadir el atributo onClick a nuestro <button>, lo que hacemos es localizarlo mediante querySelector(). Esto podríamos hacerlo mediante una clase, pero en este ejemplo lo hemos hecho directamente mediante el botón, para simplificar.

¿problemas?

Los mismos que antes  
es similar a =>

```
<button>Saludar</button>

<script>
const button = document.querySelector("button");
const doTask = () => alert("Hello!");
button.setAttribute("onclick", "doTask()");
</script>
```

A grandes rasgos, se trata de una forma alternativa a gestionar los eventos Javascript desde HTML, pero creando el HTML mediante la API del DOM de Javascript.

El siguiente método es muuuuchoo mejor

**Mediante addEventListener**

Con el método `.addEventListener()` permite añadir una escucha del evento indicado (primer parámetro), y en el caso de que ocurra, se ejecutará la función asociada indicada (segundo parámetro)

```
const button = document.querySelector("button");
button.addEventListener("click", function() {
    alert("Hello!");
});
```

## Más organizado

```
const button = document.querySelector("button");
function action() {
  alert("Hello!");
};
button.addEventListener("click", action);
```

## Con Arrow Functions

```
const button = document.querySelector("button");
const action = () => alert("Hello!");
button.addEventListener("click", action);
```

## Con múltiples listener

```
<button>Saludar</button>

<style>
    .red { background: red }
</style>

<script>
const button = document.querySelector("button");
const action = () => alert("Hello!");
const toggle = () => button.classList.toggle("red");

button.addEventListener("click", action);          // Hello message
button.addEventListener("click", toggle);         // Add/remove red CSS
</script>
```

## Eliminando EventListener

Se usa para para eliminar un listener que se ha añadido previamente al elemento. Para ello es muy importante indicar la misma función que añadimos con el `.addEventListener()` y no una función diferente que haga lo mismo que la primera.

```
<button>Saludar</button>

<style>
    .red { background: red }
</style>

<script>
const button = document.querySelector("button");
const action = () => alert("Hello!");
const toggle = () => button.classList.toggle("red");

button.addEventListener("click", action);          // Add listener
button.addEventListener("click", toggle);         // Toggle red CSS
button.removeEventListener("click", action);       // Delete listener
</script>
```

Si la funcionalidad no la tenemos en una función y hemos creado una función anónima para ello, no podremos usar `removeEventListener` porque serán funciones diferentes

## El objeto Event

Para utilizar este objeto, que está asociado a cualquier evento, simplemente hay que indicarlo como parámetro del callback que gestiona el evento. Así, puede reescribirse el listener anterior para hacer uso de él

# Propiedades del objeto Evento

Propiedad	Utilidad
<code>altKey</code>	Devuelve si la tecla [Alt] fue pulsada durante el evento.
<code>button</code>	Devuelve el botón del ratón que activó el evento: <ul style="list-style-type: none"><li>■ 0: botón principal.</li><li>■ 1: botón central.</li><li>■ 2: botón secundario.</li><li>■ 3 y 4: cuarto y quinto botones (si los hubiera).</li></ul>
<code>charCode</code>	Contiene el valor Unicode de la tecla que se pulsó (evento <code>keypress</code> ).
<code>clientX</code>	Coordenada X del ratón con respecto a la ventana.
<code>clientY</code>	Coordenada Y del ratón con respecto a la ventana.
<code>ctrlKey</code>	Devuelve si la tecla [Ctrl] fue pulsada durante el evento.
<code>pageX</code>	Coordenada X del evento, relativa al documento completo.
<code>pageY</code>	Coordenada Y del evento, relativa al documento completo.
<code>screenX</code>	Coordenada X del evento con respecto a la pantalla.
<code>screenY</code>	Coordenada Y del evento con respecto a la pantalla.
<code>shiftKey</code>	Devuelve si la tecla [Mayús] fue pulsada durante el evento.
<code>target</code>	Referencia al elemento que lanzó el evento.
<code>timeStamp</code>	Devuelve el momento en el que se creó el evento.
<code>type</code>	Nombre del evento.

**DOMContentLoaded**

Investiga...

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

Clase 27

JS

<Índice>

## Índice

[Eventos de formulario](#)

---

[Eventos de ratón](#)

---

[Eventos de teclado](#)

---

[Eventos de arrastrar y soltar](#)

---

[Eventos de reproducción multimedia](#)

---

[Otros eventos](#)

## Eventos de formulario

### Eventos de formulario

Los formularios representan las auténticas entradas de datos de las aplicaciones web.

Más allá de capturar un clic en un elemento del DOM, existe todo un universo de controles para interactuar con el usuario

## Eventos de formulario

Desde el punto de vista del DOM, todos los elementos de un formulario normalmente se encuentran anidados en el elemento `<form>`. Aun así, el objeto `document` incluye una propiedad especial, llamada `forms`, que contiene todos los formularios de un documento.

## Eventos de formulario

De esta forma, el primer formulario que encuentra estará localizable en `forms[0]`, el segundo en `forms[1]` y así sucesivamente, por lo que no es necesario utilizar el habitual `document.getElementsByTagName("form")` para trabajar con ellos.

# Eventos de formulario

Propiedades y métodos del objeto forms

Propiedad	Utilidad
<code>action</code>	Contiene la URL que recibirá los datos del formulario.
<code>elements</code>	Contiene todos los controles del formulario.
<code>length</code>	Número de controles del formulario.
<code>method</code>	{GET POST} en función del método elegido para enviarlo.
<code>enctype</code>	Tipo de codificación de los datos del formulario.
<code>acceptCharset</code>	Conjunto de caracteres del formulario.
<code>submit()</code>	Envía los datos del formulario a la URL de <code>action</code> usando <code>method</code> .
<code>reset()</code>	Devuelve el formulario a su estado inicial.
<code>onX</code>	Todos los eventos asociados al formulario, siendo <i>X</i> el nombre del evento: <code>onAbort</code> , <code>onBlur</code> , <code>onCancel</code> , <code>onClick</code> ...

# Eventos de formulario

Hagamos un ejemplo:

Crea un formulario con un solo control, una dirección de correo electrónico (usa el tipo texto). Además, incluye dos enlaces, uno para enviar el formulario y otro para reiniciarlo. Los enlaces no deben funcionar como enlaces, sino como botones que lancen los eventos submit y reset.

Antes de permitir el envío del formulario es necesario comprobar que el correo electrónico introducido contiene una @.

# Eventos de formulario

Trabajemos con los elementos de un formulario.

Vamos a ver las propiedades que tiene cada elemento de formulario con las que podemos interactuar

# Eventos de formulario

Propiedad	Utilidad
<code>accept</code>	Tipos de archivos que se permiten en un control de tipo <code>file</code> .
<code>autocomplete</code>	Si el valor del control puede ser autocompletado por el navegador.
<code>name</code>	Nombre del control.
<code>type</code>	Tipo de control.
<code>value</code>	Valor actual del control.
<code>checked</code>	<code>true</code> si el control está activado, <code>false</code> si no lo está.
<code>defaultChecked</code>	Valor predeterminado de la propiedad <code>checked</code> .
<code>disabled</code>	<code>true</code> si el control está deshabilitado, <code>false</code> si está habilitado.
<code>hidden</code>	El control es invisible para el usuario, pero no para el programador.
<code>readonly</code>	<code>true</code> si el control es de solo lectura (no modificable), <code>false</code> si no lo es.
<code>required</code>	<code>true</code> si proporcionarle un valor al control es obligatorio, <code>false</code> si no lo es.
<code>maxLength</code>	Anchura máxima del texto.
<code>min</code>	Valor mínimo para el control.
<code>max</code>	Valor máximo para el control.
<code>pattern</code>	Una expresión regular contra la que el valor del control es evaluado.

# Eventos de formulario

Propiedad	Utilidad
<code>placeholder</code>	Una pista para el usuario sobre lo que debe introducir en el control.
<code>size</code>	Tamaño inicial del control.
<code>step</code>	Tamaño del cambio en el valor de un control.
<code>selectionStart</code>	En una selección de texto la posición del primer carácter seleccionado.
<code>selectionEnd</code>	En una selección de texto, la posición del último carácter seleccionado.

# Eventos de formulario

Ejemplo:

Capturar el envío de un formulario con mensaje de error

Por ejemplo, crear un formulario para validar una edad entre unos valores. Si no está, mostrar error y si está, enviarlo.

Explicación del preventDefault

## Eventos de formulario

¿Dónde es mejor validar? ¿En el front o en el back?

## Eventos de formulario

Evento change de los elementos de un formulario

## Eventos de formulario

Evento focus y blur

## Eventos de ratón

## Eventos de ratón

Evento	Funcionamiento
<b>click</b>	Hacer clic sobre el botón principal del dispositivo.
<b>dblclick</b>	Hacer doble clic sobre el botón principal del dispositivo.
<b>mousedown</b>	Cuando se pulsa y justo antes de soltarlo, se lanza el evento.
<b>mouseup</b>	El evento se lanza cuando se suelta el botón.
<b>mousenter</b>	El evento se lanza cuando el puntero se sitúa sobre el elemento que captura el evento.
<b>mouseleave</b>	El evento se lanza cuando el puntero deja de situarse sobre el elemento que captura el evento.
<b>mousemove</b>	Mientras se está dentro del elemento, el evento se lanza cada vez que se mueve el puntero.
<b>mouseover</b>	El evento se lanza cuando el puntero se sitúa sobre el elemento que lo captura o sobre cualquiera de sus hijos.
<b>mouseout</b>	El evento se lanza cuando el puntero deja de situarse sobre el elemento que lo captura o sobre cualquiera de sus hijos.
<b>contextmenu</b>	El evento se lanza cuando se solicita un menú contextual.

## Eventos de ratón

Ejemplo:

Crea un programa que a partir de una nutrida variedad de elementos HTML los coloree con colores aleatorios cada vez que se coloque el ratón sobre ellos, y los vuelva a colorear de blanco cuando el ratón los abandone.

## Eventos de ratón

Ejemplo:

Crea un programa que no permita que se abra el menú contextual sobre ningún elemento de la web.

## Eventos de teclado

## Eventos de teclado

Evento	Funcionamiento
keypress	El evento se lanza tras pulsar y soltar una tecla.
keydown	El evento se lanza tras pulsar y antes de soltar la tecla.
keyup	El evento se lanza tras soltar la tecla.

Al trabajar con eventos de teclado, el objeto de evento contiene información que puede resultar de mucho interés. Por ejemplo, una de las propiedades disponibles en el objeto es `key`, que almacena información de la tecla pulsada, con independencia de si se pulsaron directamente, o en combinación con [Mayús], [AltGr], [Ctri] o cualquier otra.

## Eventos de teclado

Valor	Funcionamiento	Constante asociada
0	Teclado estándar	DOM_KEY_LOCATION_STANDARD
1	Parte izquierda	DOM_KEY_LOCATION_LEFT
2	Parte derecha	DOM_KEY_LOCATION_RIGHT
3	Teclado numérico	DOM_KEY_LOCATION_NUMPAD

Además, también podría resultar de interés cuando existen varias teclas para la misma función (como [Mayús] o [Ctri]) en distintas zonas del teclado, cuál de ellas se pulsó. Para ello, se puede recurrir a la propiedad `location`

## **Eventos de reproducción multimedia**

### **Eventos de reproducción multimedia**

Son aquellos eventos que pueden capturarse cuando se reproducen contenidos multimedia, sean del tipo que sean. Ojo, solo de elementos nativos HTML

# Eventos de reproducción multimedia

Evento	Momento en el que se activa
<code>canplay</code>	Cuando se detecta que el contenido se puede comenzar a reproducir.
<code>canplaythrough</code>	Cuando se estima que el contenido tiene cargados datos suficientes como para poder reproducirse.
<code>durationchange</code>	Cuando se modifica el atributo <code>duration</code> del medio.
<code>emptied</code>	Cuando se vacía de datos el medio.
<code>ended</code>	Cuando se detiene la reproducción, sin intervención del usuario.
<code>loadeddata</code>	Cuando se ha cargado la primera imagen de un vídeo.
<code>loadedmetadata</code>	Cuando se han cargado los metadatos del medio.
<code>pause</code>	Cuando se pausa el medio de reproducción.
<code>play</code>	Cuando se reanuda la reproducción tras una pausa.
<code>playing</code>	Cuando el medio está listo para reproducirse tras una parada.
<code>ratechange</code>	Cuando se modifica el <code>rate</code> del video en reproducción.
<code>seeked</code>	Cuando finaliza la búsqueda en el medio.
<code>seeking</code>	Cuando se inicia la búsqueda en el medio.
<code>stalled</code>	Cuando se produce un fallo en la carga, pero la carga continúa.
<code>suspend</code>	Cuando se suspende la carga del medio.
<code>timeupdate</code>	Cuando se modifica el valor de <code>currentTime</code> del medio.
<code>volumechange</code>	Cuando se modifica el volumen.
<code>waiting</code>	Cuando la reproducción se detiene por falta de datos.

## Otros eventos

Evento	Momento en el que se activa
<b>Carga de elementos</b>	
<b>abort</b>	Cuando se cancela la carga de un elemento.
<b>DOMContentLoaded</b>	Cuando se ha cargado el documento HTML.
<b>error</b>	Cuando se produce un error en la carga.
<b>load</b>	Cuando se ha cargado el documento y los elementos externos que incorpora: imágenes, hojas de estilo...
<b>progress</b>	Cuando se está produciendo la carga.
<b>readystatechange</b>	Cuando se modifica el valor del atributo <b>readystate</b> .

Ventana	
<b>scroll</b>	Cuando se desplaza la ventana por medio de las barras de desplazamiento.
<b>resize</b>	Cuando se modifica el tamaño de la ventana.

Impresión	
<b>afterprint</b>	Cuando ha comenzado la impresión o se ha cerrado la previsualización de la impresión.
<b>beforeprint</b>	Cuando va a comenzar la impresión o se ha abierto la previsualización de la impresión.
Portapapeles	
<b>copy</b>	Cuando se va a copiar justo antes de ejecutar la acción.
<b>cut</b>	Cuando se va a cortar justo antes de ejecutar la acción.
<b>paste</b>	Cuando se va a pegar justo antes de ejecutar la acción.

# <Despedida>

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

Clase 29

JS

Eventos personalizados

## Eventos personalizados

Crear un evento personalizado en Javascript es muy sencillo. Se basa en crear una instancia del objeto CustomEvent, al cuál le pasaremos un con el nombre que le pondremos a nuestro evento. Como segundo parámetro le indicaremos un de opciones, que explicaremos más adelante.

```
const messageEvent = new CustomEvent("message", options);
```

## Eventos personalizados

### Nombre

En ejemplos sencillos no suele importar demasiado, pero una buena práctica a largo plazo es comenzar eligiendo una buena convención de nombres para los nombres de eventos, que sea «autoexplicativo» en cuanto la acción que vamos a realizar y a la vez sea coherente y fácil de recordar.

# Eventos personalizados

## Opciones

El segundo parámetro del CustomEvent es un `Object` donde podremos especificar varios detalles en relación al comportamiento o contenido del evento.

Opciones	Valor inicial	Descripción
<code>OBJECT</code> <code>detail</code>	<code>null</code>	Objeto que contiene la <a href="#">información</a> que queremos transmitir.
<code>BOOLEAN</code> <code>bubbles</code>	<code>false</code>	Indica si el evento debe <a href="#">burbujear</a> en el DOM «hacia la superficie» o no.
<code>BOOLEAN</code> <code>composed</code>	<code>false</code>	Indica si la propagación puede atravesar <a href="#">Shadow DOM</a> o no. <a href="#">Ver WebComponents</a>
<code>BOOLEAN</code> <code>cancelable</code>	<code>false</code>	Indica si el comportamiento se puede cancelar con <code>.preventDefault()</code> .

# Eventos personalizados

## Opciones

El segundo parámetro del CustomEvent es un `Object` donde podremos especificar varios detalles en relación al comportamiento o contenido del evento.

Opciones	Valor inicial	Descripción
<code>OBJECT</code> <code>detail</code>	<code>null</code>	Objeto que contiene la <a href="#">información</a> que queremos transmitir.
<code>BOOLEAN</code> <code>bubbles</code>	<code>false</code>	Indica si el evento debe <a href="#">burbujear</a> en el DOM «hacia la superficie» o no.
<code>BOOLEAN</code> <code>composed</code>	<code>false</code>	Indica si la propagación puede atravesar <a href="#">Shadow DOM</a> o no. <a href="#">Ver WebComponents</a>
<code>BOOLEAN</code> <code>cancelable</code>	<code>false</code>	Indica si el comportamiento se puede cancelar con <code>.preventDefault()</code> .

## Eventos personalizados

```
const MessageEvent = new CustomEvent("user:data-message", {  
    detail: {  
        from: "Manz",  
        message: "Hello!"  
    },  
    bubbles: true,  
    composed: true  
});
```

## Emisión de eventos

## Eventos de formulario

Además de capturar eventos realizados por los usuarios, también podemos lanzarlos de forma programática

¿Para qué puede servir esto?

Con esto podemos simular eventos de usuarios o gestionar eventos personalizados

## Emisión de eventos

```
<button>Click me</button>
<span class="text">Hover me</span>

<script>
const button = document.querySelector("button");
const text = document.querySelector(".text");

button.addEventListener("click", () => alert("Has pulsado el botón"));

text.addEventListener("mouseenter", () => {
  const event = new Event("click");
  button.dispatchEvent(event);
});
</script>
```

## Emisión de eventos

```
<button id="botonRegistro">Registrar Usuario</button>
<div id="mensajeBienvenida" style="display:none;">¡Bienvenido al sistema!</div>

<script>
  // Paso 1: Crear el manejador de eventos para escuchar el evento personalizado
  document.addEventListener('usuarioRegistrado', function(evento) {
    console.log('Evento personalizado recibido:', evento);
    console.log('Detalles del evento:', evento.detail);

    document.getElementById('mensajeBienvenida').style.display = 'block';
  });

  // Paso 2: Crear y lanzar el evento personalizado al hacer clic en el botón
  document.getElementById('botonRegistro').addEventListener('click', function() {
    // Crear un evento personalizado con algunos detalles
    var eventoPersonalizado = new CustomEvent('usuarioRegistrado', {
      detail: {
        nombreUsuario: 'Juan',
        motivo: 'Registro completado'
      }
    });

    // Lanzar el evento
    document.dispatchEvent(eventoPersonalizado);
  });
</script>
```

## Propagación de eventos

## Propagación de eventos

El concepto de propagación de eventos es muy importante para entender todo el ecosistema de utilidades que rodean a la gestión de eventos.

La propagación hace referencia a que los eventos pueden gestionarse desde un elemento más profundo hasta la superficie, por eso se denomina comportamiento de burbuja.

## Propagación de eventos

Dicho de otra forma, en una jerarquía de elementos, pueden capturarse eventos desde el nivel más profundo hasta el nivel más superficial, cuando se activa el evento más profundo.

## Propagación de eventos

Veamos en qué orden se ejecutan los eventos click.  
A esto le llamamos el comportamiento de burbuja

## Propagación de eventos

¿cómo cambiamos este comportamiento?  
Con el tercer parámetro de addEventListener  
Probemos de nuevo

## Propagación de eventos

¿Qué pasa cuando no queremos que el evento se propague y muera en el elemento en el que se ha producido?

stopPropagation al rescate

Ejemplo con el número de veces de ejecución por ejemplo

## Propagación de eventos

Ejemplos

1. Prevenir la Propagación de un Evento de Clic en un Botón  
Dentro de un Div
2. Detener la Propagación de un Evento de Menú  
Contextual
3. Evitar que un Evento Keydown se Propague
4. Detener la Propagación de un Evento en un Manejador  
de Eventos Delegado

# Cancelación de eventos

## 1. Prevenir la Propagación de un Evento de Clic en un Botón Dentro de un Div

```
<div id="divPadre" style="padding: 20px; background-color: #f0f0f0;">
    Haz clic en cualquier lugar de este div.
    <button id="botonHijo">Haz clic en mí</button>
</div>
<script>
    document.getElementById('divPadre').addEventListener('click', function() {
        alert("Clic en el div padre!");
    });

    document.getElementById('botonHijo').addEventListener('click', function(evento) {
        evento.stopPropagation();
        alert("Clic en el botón hijo!");
    });
</script>
```

# Cancelación de eventos

## 2. Detener la Propagación de un Evento de Menú Contextual

```
<div id="divExterno" style="padding: 20px; background-color: lightgray;">
    Div externo (clic derecho aquí mostrará el menú contextual predeterminado)
    <div id="divInterno" style="padding: 20px; background-color: lightblue;">
        Div interno (clic derecho aquí no propagará el evento)
    </div>
</div>
<script>
    document.getElementById('divInterno').addEventListener('contextmenu', function(evento) {
        evento.stopPropagation();
        alert("Menú contextual personalizado!");
        // Aquí podrías abrir tu propio menú contextual
        evento.preventDefault(); // Prevenir el menú contextual predeterminado del navegador
    });
</script>
```

## Cancelación de eventos

### 3. Evitar que un Evento Keydown se Propague

```
<input type="text" id="miInput" placeholder="Escribe algo aquí...">
<script>
  document.getElementById('miInput').addEventListener('keydown', function(evento) {
    if(evento.key === 'Enter') {
      evento.stopPropagation();
      alert("Presionaste Enter, pero el evento no se propagará.");
    }
  });
</script>
```

## Cancelación de eventos

### 4. Detener la Propagación de un Evento en un Manejador de Eventos Delegado

```
<ul id="listaPadre">
  <li>Ítem 1</li>
  <li id="itemEspecial">Ítem Especial (clic detiene la propagación)</li>
  <li>Ítem 3</li>
</ul>
<script>
  document.getElementById('listaPadre').addEventListener('click', function() {
    alert("Clic en un ítem de la lista!");
  });

  document.getElementById('itemEspecial').addEventListener('click', function(evento) {
    evento.stopPropagation();
    alert("Clic en el ítem especial!");
  });
</script>
```

## **Cancelación de eventos**

### **Cancelación de eventos**

Para cancelar un evento se debe diferenciar entre dos escenarios: eventos predeterminados y eventos personalizados.

## Cancelación de eventos

Con esto conseguiremos que no se ejecute el comportamiento por defecto de un evento como por ejemplo submit, los enlaces, etc...

1. Prevenir el Envío Automático de un Formulario
2. Prevenir que un Enlace Navegue a una URL
3. Prevenir el Menú Contextual del Botón Derecho del Ratón
4. Controlar el Comportamiento de Teclas Específicas

## Cancelación de eventos

### 1. Prevenir el Envío Automático de un Formulario

```
<form id="miFormulario">
  <input type="text" name="nombre" required>
  <input type="submit" value="Enviar">
</form>
<script>
  document.getElementById('miFormulario').addEventListener('submit', function(evento) {
    evento.preventDefault(); // Previene el envío del formulario
    // Aquí tu lógica de validación
    if(/* validación exitosa */) {
      // Eventualmente, podrías enviar el formulario programáticamente
      // evento.target.submit();
    } else {
      alert("La validación falló.");
    }
  });
</script>
```

## Cancelación de eventos

### 2. Prevenir que un Enlace Navegue a una URL

```
<a href="https://ejemplo.com" id="miEnlace">Haz clic en mí</a>
<script>
  document.getElementById('miEnlace').addEventListener('click', function(evento) {
    evento.preventDefault(); // Previene la navegación
    console.log("El enlace fue clickeado, pero no te llevará a ejemplo.com");
  });
</script>
```

## Cancelación de eventos

### 3. Prevenir el Menú Contextual del Botón Derecho del Ratón

```
<div id="miElemento" style="width: 200px; height: 200px; background-color: lightblue;">
  Haz clic derecho sobre mí
</div>
<script>
  document.getElementById('miElemento').addEventListener('contextmenu', function(evento) {
    evento.preventDefault(); // Previene la aparición del menú contextual
    alert("El menú contextual fue deshabilitado en este elemento.");
  });
</script>
```

## Cancelación de eventos

### 4. Controlar el Comportamiento de Teclas Específicas

```
<script>
  window.addEventListener('keydown', function(evento) {
    if(evento.code === "Space") {
      evento.preventDefault(); // Previene el scroll por la tecla Espacio
      console.log("Se presionó la tecla Espacio, pero no hará scroll en la página.");
    }
  });
</script>
```

## Cancelación de eventos

### 5. Prevenir la Selección de Texto

```
<div id="noSelectable">Intenta seleccionarme</div>
<script>
  document.getElementById('noSelectable').addEventListener('mousedown', function(evento) {
    evento.preventDefault(); // Previene la selección de texto
  });
</script>
```

## Cancelación de eventos

Con esto conseguiremos que no se ejecute el comportamiento por defecto de un evento como por ejemplo submit, los enlaces, etc...

## Cancelación de eventos

Sin embargo, lo que se ha hecho es cancelar el comportamiento por defecto, no el evento en sí. Para anular completamente un evento se recurre a `removeEventListener()`. Además, hay que tener en cuenta que la anulación no puede hacerse cuando la definición del evento se hace con una función anónima, solo es posible cuando se utilizan funciones con identificador.

## **target vs currentTarget**

### **target vs currentTarget**

En el contexto de los eventos en JavaScript, target y currentTarget son propiedades del objeto evento que ofrecen información sobre el elemento que desencadenó el evento y el elemento que actualmente está manejando el evento, respectivamente. Aunque a primera vista puedan parecer similares, tienen propósitos distintos. Veamos las diferencias:

# target vs currentTarget

## event.target

- **Definición:** event.target se refiere al elemento que fue el objetivo original del evento. Es decir, el elemento en el que el evento se originó realmente. Si tienes un botón dentro de un div y haces clic en el botón, event.target sería el botón, independientemente de en qué elemento se haya registrado el manejador de eventos.
- **Uso Común:** Es especialmente útil en la delegación de eventos, donde un solo manejador de eventos en un elemento padre se utiliza para manejar eventos de múltiples elementos hijos. event.target te permite determinar cuál de los elementos hijos desencadenó el evento.

# target vs currentTarget

## event.currentTarget

- **Definición:** event.currentTarget se refiere al elemento en el que actualmente se está manejando el evento. A diferencia de event.target, que señala el origen del evento, event.currentTarget apunta al elemento que tiene el manejador de eventos que actualmente está procesando el evento.
- **Uso Común:** event.currentTarget es útil cuando el mismo manejador de eventos se ha adjuntado a múltiples elementos y necesitas saber cuál de ellos está procesando el evento actualmente. Esto es común en los casos donde se agrega el mismo manejador de eventos a varios elementos para evitar la duplicación de código.

# target vs currentTarget

```
<div id="divPadre">
  <button id="botonHijo">Haz clic en mí</button>
</div>
<script>
  document.getElementById('divPadre').addEventListener('click', function(evento) {
    console.log('event.target:', evento.target); // Será el botón, si se cliquea el botón
    console.log('event.currentTarget:', evento.currentTarget); // Siempre será divPadre
  });
</script>
```

«Despedida»

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

Clase 33

JS

<Índice>

## Índice

Cookies

---

LocalStorage

---

SessionStorage

---

## Cookies

Teoría

## Ejercicio

Vamos a crear nuestro propio aviso y petición de permiso de Cookies

- Debe tener dos botones, aceptar y cancelar
- Si el usuario marca en aceptar, cargará un JS de analítica que lo único que hace es un console.log. En cada petición se volverá a cargar ese js. No volverá a preguntar tras pasados 1 mes
- Si el usuario pulsa en cancelar, no se cargará dicho archivo js
- Y no debería preguntar más hasta pasadas 24 horas

## LocalStorage

## Teoría

## **SessionStorage**

## Teoría

## Ejercicio

Vamos a hacer que nuestro anterior ejercicio de las parejas, sea persistente aunque recarguemos la ventana. Esto quiere decir, que el juego siga por donde iba aunque saltamos y volvamos a entrar en nuestro juego.

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**

# {JS}

Clase 37

JS

<Índice>

## Índice

Promesas

---

Resolve y Reject

---

Then, catch, finally

---

Cadena de promesas

---

Métodos estáticos

---

## Promesas

### Promesas

Son utilizadas sobre todo para hacer peticiones a otros servidores

## Promesas

Así obtenemos datos de otros lugares y los mostramos en el nuestro

## Promesas

Representan el estado de una petición  
Pendiente – Pending  
Rechazada – Rejected  
Terminada – Fullfilled

## Promesas

Se usan para  
Peticiones a servidores  
Peticiones a bases de datos  
Webworkers

## Promesas

Se usan para  
Peticiones a servidores  
Peticiones a bases de datos  
Webworkers

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

Clase 38

JS

<Índice>

## Índice

Promesas en serie

---

Promesas en paralelo: all

---

Promesas en paralelo: race

---

Promesas en paralelo: any

---

Promesas en paralelo: allSettled

---

Argumentos a una promesa

---

## **Promesas en serie**

### **Promesas en serie**

Son utilizadas cuando una promesa depende del resultado de la anterior

## Promesas en paralelo all

all

Son utilizadas cuando son independientes y queremos ejecutar algo cuando todas terminen

## Promesas en paralero race

race

Resuelve cuando la primera  
promesa de las que le pasamos  
se resuelve

## Promesas en paralero any

race

Se resuelve cuando alguna de las promesas de las que le pasamos se resuelve. Si alguna es rechazada la ignora y sigue con las demás

## Promesas en paralelo allSettled

**race**

Se resuelve cuando todas las promesas de las que le pasamos se resuelven o rechazan

## Argumentos

«Despedida»

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**[youtube.com/bienvenidosaez](https://youtube.com/bienvenidosaez)**



# {JS}

Clase 39

JS

<Índice>

Índice

API

---

Fetch

---

Async y Await

---

**API**

**API**

<https://jsonplaceholder.typicode.com/>

race

myfakeapi.com

API

Listar => GET /todos/1

Crear => POST /todos

Modificar => PUT /todos/1

Borrar => DELETE /todos/1

## Fetch

## Fetch

Fetch será utilizado para pedir o enviar datos a un servidor desde nuestro código javascript

## Fetch

Fetch siempre devuelve una  
promesa resuelta  
(cuidado con esto)

## Fetch

Fetch necesita dos parámetros  
url y un objeto con opciones que  
veremos posteriormente

## Fetch

Podemos usar then, catch y  
todo lo aprendido con las  
promesas

## Fetch

Hagamos nuestro primer fetch

## async y await

### async y await

La declaración de función async define una función asíncrona que devuelve un objeto, lo cual permite a un programa correr una función sin congelar todo la compilación

## async y await

Dada que la finalidad de las funciones `async/await` es simplificar el comportamiento del uso síncrono de promesas, se hace más fácil escribir promesas.

## async y await

Nos evita tener que encadenar con `.then` si no que usamos un sistema más tradicional y lógico ante nuestros ojos

## async y await

“Bloqueamos” hasta que son resueltas para continuar dentro de una función

## async y await

Lo que hace await es detener la ejecución y no continuar. Se espera a que se resuelva la promesa, y hasta que no lo haga, no continua. A diferencia del .then(), aquí tenemos un código bloqueante.

## **async y await**

Sobre todo viene a resolver  
el callback hell

## **async y await**

Probemos

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**



# {JS}

Clase 40

JS

<Índice>

## Índice

[Nuestro primer Post con Fetch](#)

---

[Crear Todo](#)

---

[Crear Todo v2](#)

---

**API**

**API**

<https://jsonplaceholder.typicode.com/>

## API

Listar => GET /todos/1

Crear => POST /todos

Modificar => PUT /todos/1

Borrar => DELETE /todos/1

<Despedida>

Email

[bienvenidosaez@gmail.com](mailto:bienvenidosaez@gmail.com)

Instagram

[@bienvenidosaez](https://www.instagram.com/bienvenidosaez)

Youtube

[youtube.com/bienvenidosaez](https://www.youtube.com/bienvenidosaez)



# {JS}

Clase 41

JS

<Índice>

## Índice

[Autenticación y Autorización](#)

---

[Autenticación basada en sesiones](#)

---

[Autenticacion basada en tokens](#)

---

[Sesiones vs Tokens](#)

---

[JWT](#)

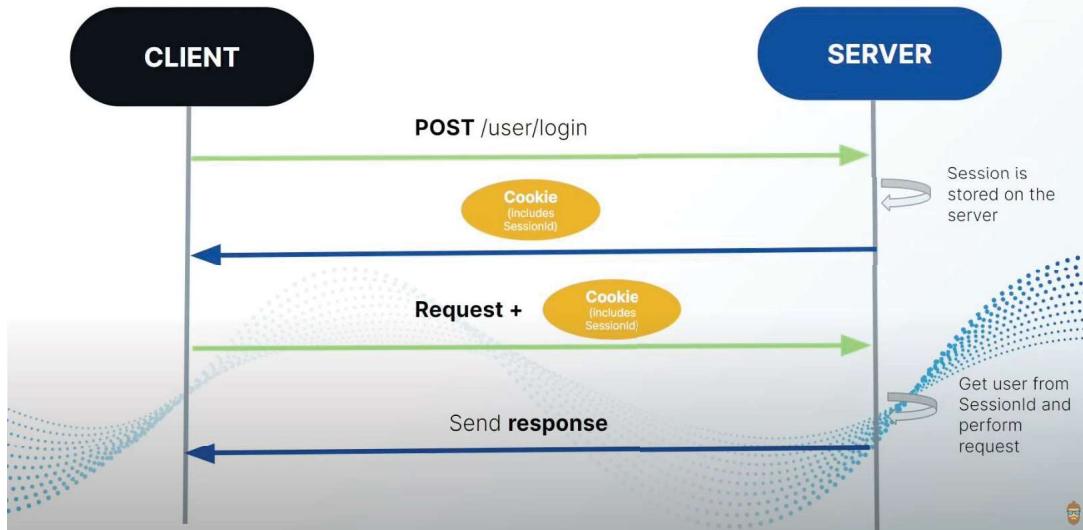
---

## **Autenticación vs Autorización**

### **Autenticación basada en sesiones**

# API

## Session Based Authentication



# API

## Microservices example: Uber



# API

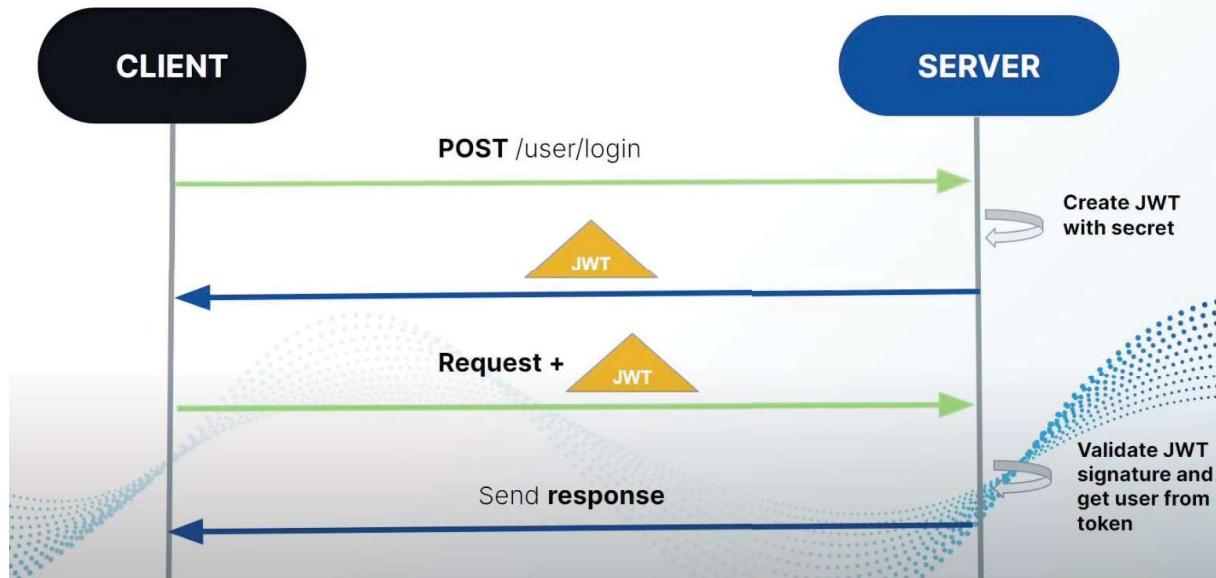
## Microservices example: Uber



## Autenticación basada en tokens

# API

## Token Based Authentication

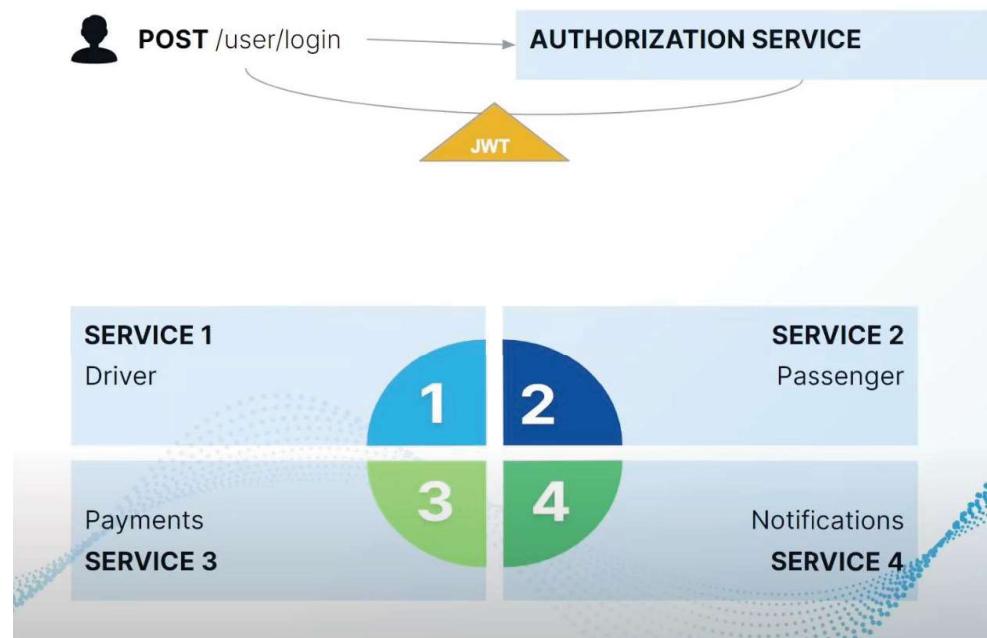


# API

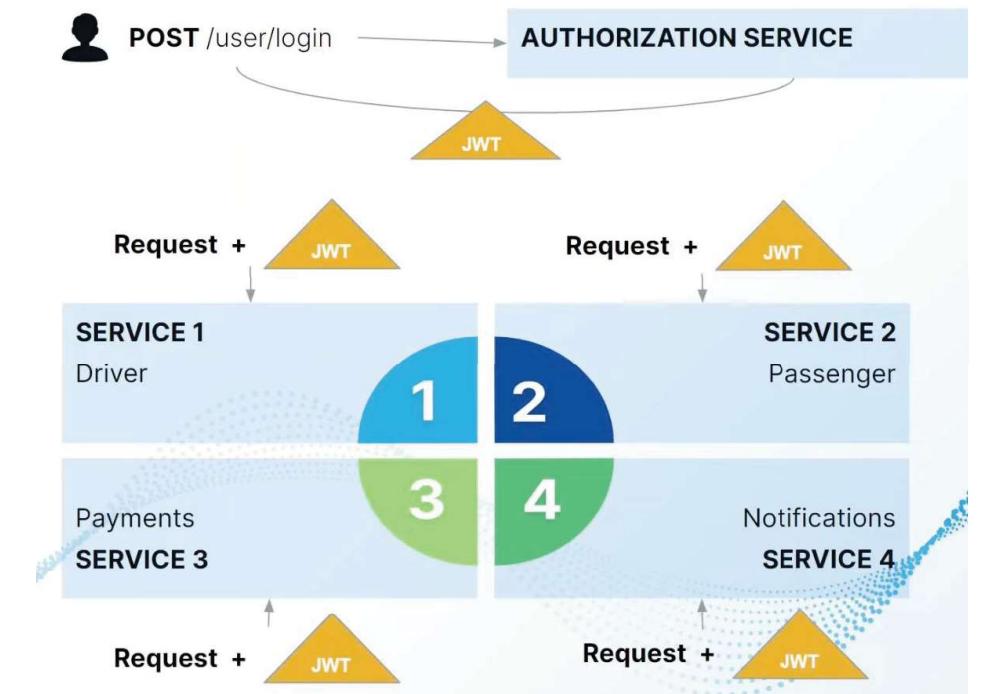
## Microservices example: Uber



# API



# API



**API**

Cifrado

**Sesiones VS JWT**

## Sesiones VS JWT

Entendamos cómo funciona HTTP

## Sesiones VS JWT

La autenticación por sesiones y la autenticación con tokens JWT (JSON Web Tokens) son dos métodos comunes para mantener el estado de la sesión de un usuario en aplicaciones web. Ambos tienen sus propias características, ventajas e inconvenientes.

Aquí te explico las principales diferencias entre ambos métodos

## Sesiones VS JWT

Diferencias principales

## Sesiones VS JWT

Almacenamiento de estado

## Sesiones VS JWT

- Sesiones: La autenticación por sesiones mantiene el estado del usuario en el servidor. Se crea un archivo o un registro en una base de datos para cada sesión donde se almacena la información del usuario. El cliente solo almacena un identificador de sesión (generalmente en una cookie), que el servidor utiliza para recuperar el estado almacenado.

## Sesiones VS JWT

- JWT: La autenticación JWT es stateless (sin estado). No se almacena información del usuario en el servidor. En cambio, el servidor genera un token que contiene todos los datos necesarios, codificados y posiblemente cifrados, que el cliente envía en cada solicitud. El servidor lee y verifica este token para obtener la información del usuario.

## Sesiones VS JWT

Escalabilidad

## Sesiones VS JWT

- Sesiones: Como el estado se almacena en el servidor, la escalabilidad puede ser un problema, especialmente en aplicaciones distribuidas donde se necesitan mecanismos como el sticky session o bases de datos de sesión compartidas.

## Sesiones VS JWT

- JWT: Es más escalable en sistemas distribuidos ya que el servidor no necesita mantener el estado de la sesión. Cada solicitud contiene toda la información necesaria en el token.

## Sesiones VS JWT

Seguridad

## Sesiones VS JWT

- Sesiones: Son relativamente seguras si se implementan correctamente, como configurar las cookies de sesión para que sean HttpOnly (no accesibles por JavaScript) y Secure (transmitidas solo a través de HTTPS).

## Sesiones VS JWT

- JWT: Puede ser vulnerable si no se maneja adecuadamente. La información sensible en el token debe ser cifrada, no solo codificada. Los tokens son susceptibles a ataques si se interceptan, ya que contienen toda la información necesaria para autenticarse.

## Sesiones VS JWT

### Ventajas de las Sesiones

## Sesiones VS JWT

- Fácil de implementar con muchas frameworks y bibliotecas.
- Más control sobre la sesión, ya que el servidor puede invalidar sesiones fácilmente.
- La información del usuario no se expone al cliente, solo se almacena un ID de sesión.

## Sesiones VS JWT

Inconvenientes de las sesiones

## Sesiones VS JWT

- Requiere más recursos del servidor, ya que necesita almacenar información de la sesión.
- Menos eficiente en aplicaciones distribuidas a menos que se implementen soluciones adicionales para la gestión de sesiones.

## Sesiones VS JWT

### Ventajas de JWT

## Sesiones VS JWT

- No requiere almacenamiento de estado en el servidor, lo que simplifica la arquitectura en sistemas distribuidos.
- Facilita el control de acceso y la autorización en diferentes servicios y microservicios.
- Puede ser utilizado en diferentes tipos de clientes, como aplicaciones móviles, web y de escritorio.

## Sesiones VS JWT

### Inconvenientes de JWT

## Sesiones VS JWT

- Requiere un manejo cuidadoso de la seguridad, especialmente en la generación y almacenamiento del token.
- Los tokens no pueden ser invalidados fácilmente. Una vez emitidos, son válidos hasta que expiren.
- La información del token puede volverse obsoleta si los datos del usuario cambian y el token aún no ha expirado.

<https://jwt.io/>

¿Probamos?

# **<Despedida>**

Email

**bienvenidosaez@gmail.com**

Instagram

**@bienvenidosaez**

Youtube

**youtube.com/bienvenidosaez**

