

# UD 4

# Objetos predefinidos de JavaScript

DESARROLLO WEB EN ENTORNO CLIENTE

**Técnico de Grado Superior Desarrollo de Aplicaciones Web**

**2024-25**

# Contenidos

- Utilización de objetos.
  - Objetos nativos del lenguaje.
- Interacción con el navegador.
  - Objetos predefinidos asociados.
- Generación de texto y elementos HTML desde código.
- Creación y gestión de marcos.
  - Aplicaciones prácticas de los marcos.
- Gestión de la apariencia de la ventana.
- Creación de nuevas ventanas.
  - Comunicación entre ventanas.

# Recordamos... tipos de datos

## JavaScript Data Types

- String
- Number
- BigInt
- Boolean
- Undefined
- Null
- Symbol
- Object



# Tipos de datos... y ¡objetos!

## JavaScript Data Types

- String
- Number
- BigInt
- Boolean
- Undefined
- Null
- Symbol
- Object



Una variable en JavaScript puede albergar cualquier tipo de dato

Los tipos de datos en JavaScript son dinámicos

### Objetos predefinidos:

`objects, arrays, dates, maps, sets, intarrays, floatarrays, promises...`

### Objetos definidos por el usuario:

```
const person =  
  {nombre:"Iñigo", apellido:"Montoya"};
```

# Tipos de datos y objetos (ejemplo)

```
// Numbers:
let length = 16;
let weight = 7.5;

// Strings:
let color = "Yellow";
let lastName = "Johnson";

// Booleans
let x = true;
let y = false;

// Object:
const person = {firstName:"John", lastName:"Doe"};

// Array object:
const cars = ["Saab", "Volvo", "BMW"];

// Date object:
const date = new Date("2022-03-25");
```

# Recordamos... programación modular: funciones

```
function nombre([param[, param[, ...param]]) {  
    instrucciones  
}
```

**nombre**

El nombre de la función.

Se puede omitir, en ese caso la función se conoce como función anónima.

**param**

El nombre de un argumento que se pasará a la función. Una función puede tener hasta 255 argumentos.

**instrucciones**

Las instrucciones que forman el cuerpo de la función.

# Programación modular: funciones

Expresión “flecha”

```
([param] [, param]) => { instrucciones }  
param => expresión
```

~~nombre~~

param

El nombre de un argumento. Si no hay argumentos se tiene que indicar con ().

Para un único argumento no son necesarios los parentesis. (como foo => 1)

instrucciones

Múltiples instrucciones deben ser encerradas entre llaves. Una única expresión no necesita llaves.

La expresión es, así mismo, el valor de retorno implícito de esa función.

# Programación modular: funciones

- Para unos parámetros/argumentos de entrada la función devuelve una salida
- Los parámetros se convierten en variables locales -> **paso por valor**
- Una variable puede apuntar a una función, es decir, **una función es un objeto**

En JavaScript, las funciones son objetos de primera clase, es decir, son objetos y se pueden manipular y transmitir al igual que cualquier otro objeto. Concretamente son objetos `Function`.

Distinguir entre librerías (agrupación de funciones) y APIs (vinculadas a un recurso):

- Tus librerías (en Javascript)
- Librerías de terceros (añaden funcionalidad)
- APIs del Navegador (las ofrece el navegador)
- APIs de terceros (servicios externos)
- Frameworks JS. Paquetes HTML/CSS/JS para construir aplicaciones



# Programación modular: funciones

Constructor “Function”

```
new Function (arg1, arg2, ... argN, functionBody)
```

\* Llamar al constructor Function como una función, sin el operador new, tiene el mismo efecto que llamarlo como un constructor.

*arg1, arg2, ... argN*

Nombres de argumentos formales.

*Cuerpo de la función*

Una cadena conteniendo las instrucciones JavaScript que comprenden la definición de la función.

NO SE RECOMIENDA

# Programación modular: funciones

```
function multiply(x, y) {  
  return x * y;  
}
```

```
var multiply = function (x, y) {  
  return x * y;  
};
```

```
var multiply = new Function("x", "y", "return x * y;");
```

```
const multiply = (x, y) => x * y;
```

```
const multiply = (x, y) => {  
  return x * y;  
};
```

# Programación modular: funciones

¿Puede una función llamarse a sí misma?

Sí, hay varias maneras

## Reto 1:

Crea una función recursiva llamada `sumaArray` que reciba un array de números y devuelva la suma de todos los elementos del array.

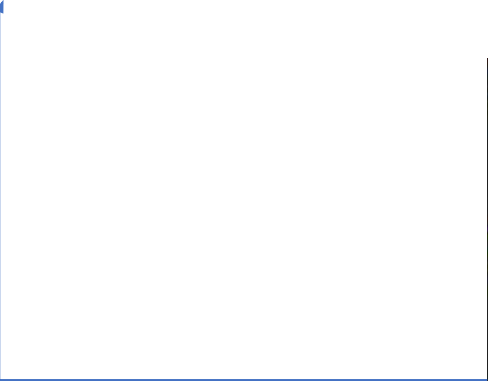
```
function sumaArray(arr) {  
  // Tu código aquí  
}  
  
console.log(sumaArray([1, 2, 3, 4, 5])); // Debe devolver 15
```

# Programación modular: funciones

## Reto 2:

Crea una función recursiva llamada `fibonacci` que devuelva el enésimo número en la serie de Fibonacci.

La serie de Fibonacci comienza con 0, y cada número siguiente es la suma de los dos anteriores. Por ejemplo: 0, 1, 1, 2, 3, 5, 8, 13....



```
function fibonacci(n) {  
    // Tu código aquí  
}  
  
console.log(fibonacci(6));
```

# Utilización de objetos

Cada objeto es una colección de pares clave-valor...  
...donde los valores pueden ser variables o funciones.

```
let persona = {  
  nombre: 'Juan',  
  edad: 30,  
  saludar: function() {  
    console.log(`Hola, mi nombre es ${this.nombre}`);  
  }  
};  
  
persona.saludar(); // "Hola, mi nombre es Juan"
```

Esta sintaxis se llama  
“interpolación de expresiones”  
+info

# Utilización de objetos

## Reto:

Crea un objeto llamado `coche` con las propiedades `marca`, `modelo` y `año`, y un método `arrancar` que imprima en la consola:  
"El coche [marca] [modelo] está arrancando".



Utiliza la interpolación de expresiones con `${ }`

# Objetos nativos del lenguaje

JavaScript viene con varios objetos nativos que ofrecen funcionalidad:

Math	□ <u>Métodos</u>
Date	□ <u>Métodos</u>
Array	□ <u>Métodos</u>
String	□ <u>Métodos</u>
...	

# Objetos nativos del lenguaje

## Reto Array:

Tienes una lista de productos con su nombre y precio. Crea una función que **filtre** los productos cuyo precio sea mayor a 20 y luego los **ordene** de mayor a menor precio.

```
let productos = [  
  { nombre: "Camiseta", precio: 25 },  
  { nombre: "Zapatos", precio: 50 },  
  { nombre: "Pantalón", precio: 15 },  
  { nombre: "Sombrero", precio: 30 },  
  { nombre: "Calcetines", precio: 5 }  
];
```



```
[  
  { nombre: "Zapatos", precio: 50 },  
  { nombre: "Sombrero", precio: 30 },  
  { nombre: "Camiseta", precio: 25 }  
]
```

## Reto Fechas:

Crea una función que reciba dos fechas (una fecha inicial y una final) y calcule cuántos días, horas, minutos y segundos han transcurrido entre ambas.

Invoca a la función con la fecha actual para saber cuánto ha pasado desde tu cumpleaños.



# Objetos nativos del lenguaje

## Reto:

Tienes dos listas de nombres.

```
let lista1 = ["Ana", "Pedro", "Maria", "Juan", "Ana", "Luis"];  
let lista2 = ["Pedro", "Laura", "Juan", "Mario", "Luis", "Ana"];
```

Crea una función que elimine los duplicados de ambas listas utilizando el objeto **Set**.

```
let set1 = new Set(lista1);  
let set2 = new Set(lista2);
```

```
let union = [...new Set([...set1, ...set2])];
```

# Interacción con el navegador

JavaScript proporciona objetos específicos `window`, `document`, `navigator`, `history`, que permiten interactuar directamente con el navegador.

```
alert("Hola! Esto es un mensaje desde window.alert.");  
console.log(window.innerWidth); // Anchura de la ventana
```

```
document.title = "Nuevo título de la página";  
console.log(document.getElementById("miElemento"));
```

```
console.log(navigator.userAgent); // Información del navegador
```

# Interacción con el navegador

## **Reto 1:**

Crea un botón en HTML que, al ser clicado, muestre un alert con el nombre del navegador que está utilizando el usuario.

## **Reto 2:**

Crea un script que muestre el ancho y alto actuales de la ventana del navegador en un elemento `<p>` de la página cada vez que la ventana sea redimensionada.

## **Reto 3:**

Crea una función que redirija al usuario a otra página web después de 5 segundos de cargar la página actual.

## **Reto 4:**

Crea un botón que, al hacer clic, abra una nueva ventana con dimensiones personalizadas (por ejemplo, 500x500) y cargue la página web del instituto en ella.

# Interacción con el navegador

## **Reto 5:**

Crea un botón en el que, al hacer clic, se cambie el título de la página web a otro texto, por ejemplo “Título nuevo”.

## **Reto 6:**

Crea un botón que verifique si el navegador del usuario está en línea o desconectado utilizando `navigator.onLine` y muestra el resultado en el body de la página.

## **Reto 7:**

Crea un botón que, al hacer clic, haga retroceder al usuario una página en su historial de navegación.

## **Reto 8:**

Crea un botón que obtenga la ubicación geográfica del usuario (si el navegador lo permite) utilizando `navigator.geolocation`.

# Generación de texto y HTML desde código

JavaScript puede generar y modificar elementos HTML para crear interfaces dinámicas.

```
let nuevoDiv = document.createElement('div');
nuevoDiv.innerHTML = "<p>Este es un nuevo párrafo</p>";
document.body.appendChild(nuevoDiv);
```

```
<ul id="listaFrutas"></ul>

<script>
let frutas = ['Manzana', 'Banana', 'Naranja'];
let lista = document.getElementById('listaFrutas');

frutas.forEach(function(fruta) {
    let li = document.createElement('li');
    li.textContent = fruta;
    lista.appendChild(li);
});
</script>
```

# Creación y gestión de marcos

Los marcos (frames) eran una forma antigua y popular de estructurar las páginas web en tiempos anteriores de desarrollo web (HTML 4).

Mediante un **frameset**, que permitía definir "marcos" o "frames" en la página web.

```
<frameset rows="20%, 80%">
  <frame src="menu.html">
  <frame src="contenido.html">
</frameset>
```

```
<frameset cols="25%, 75%">
  <frame src="barra_lateral.html">
  <frame src="contenido_principal.html">
</frameset>
```

```
<!DOCTYPE html>
<html>
<head>
  <title>Página con Frames</title>
</head>
<frameset cols="20%, 80%">
  <frame src="menu.html" name="menu">
  <frame src="contenido.html" name="contenido">
<noframes>
  Tu navegador no soporta frames.
</noframes>
</frameset>
</html>
```

# Creación y gestión de marcos

Por dificultades de navegación, de establecer como favoritos, de accesibilidad, de compatibilidad entre navegadores, de eficacia con el SEO, de gestión de estilos, etc. se “deprecán”.

HTML element: frame 📄

Usage % of all users ?  
Global 95.45%

Current aligned Usage relative Date relative Filtered All ⚙️

Chrome	Edge *	Safari	Firefox	Opera	IE ⚠️ *	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mini *	Opera Mobile *	UC Browser for Android	Android Browser *	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
				10-12.1								2.1-4.3				
4-129	12-129	3.1-17.6	2-130	15-113	6-10		3.2-17.6	4-24		12-12.1		4.4-4.4.4				2.5
130	130	18.0	131	114	11	129	18.0	25	all	80	15.5	129	130	14.9	13.52	3.1
131-133		18.1-TP	132-134				18.1									

Notes Test on a real browser Sub-features Feedback

This feature is deprecated/obsolete and **should not be used**.

See full reference on [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/frame).

# Creación y gestión de marcos

`<iframe>` "marcos inline" sustituye a los `frame`.

Son útiles para incrustar el contenido de otra documento en el HTML actual.

```
<iframe src="https://www.example.com" width="600" height="400"></iframe>
```

```
let miIframe = document.getElementsByTagName('iframe')[0];  
miIframe.src = "https://www.nuevaurl.com";
```

Una buena práctica es incluir siempre el atributo **title**, para los lectores de pantalla



# Creación y gestión de marcos

## Reto:

Crea un *iframe* en tu página que cargue la URL de la web del Sevilla FC y, mediante un botón, cambia la URL del *iframe* a la web del Betis.



### PROBLEMA

El servidor puede controlar la carga de la web en iframes ajenos mediante x-frame options y restringirlo a su dominio o denegarlo.

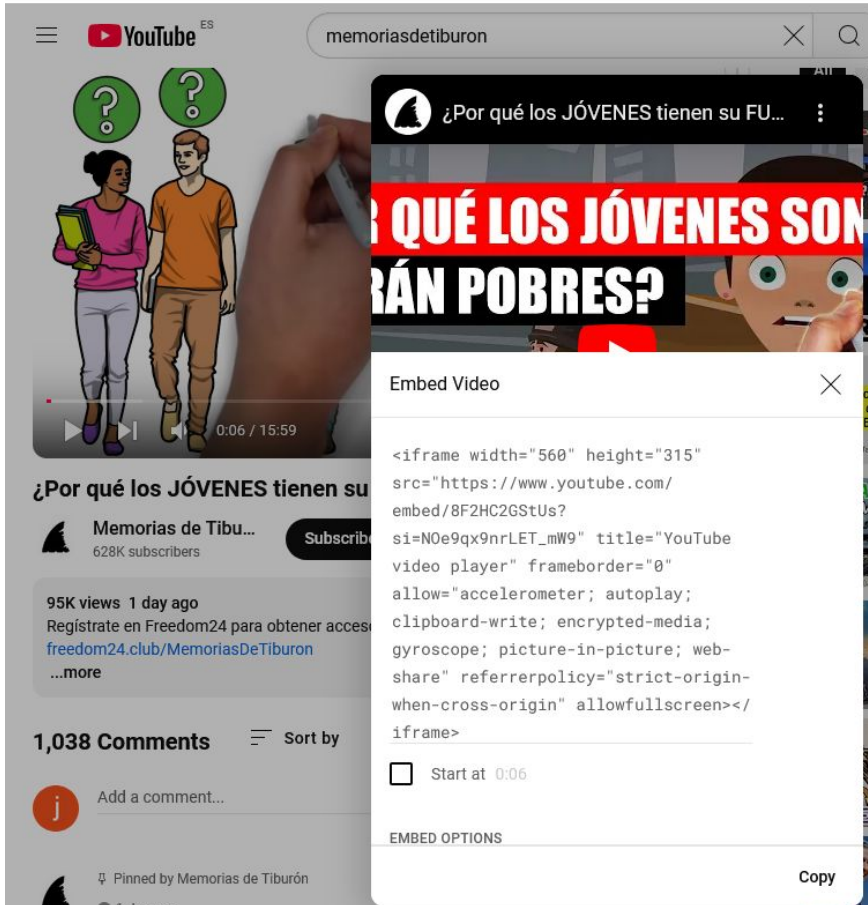
### Firefox no puede abrir esta página

Para proteger su seguridad, sevilla.fc.es no permitirá que Firefox muestre la página si otro sitio la ha incrustado.  
Para ver esta página, debe abrirla en una nueva ventana.

[Más información...](#)

☐ Informar de errores como esto ayuda a Mozilla a identificar y bloquear sitios maliciosos

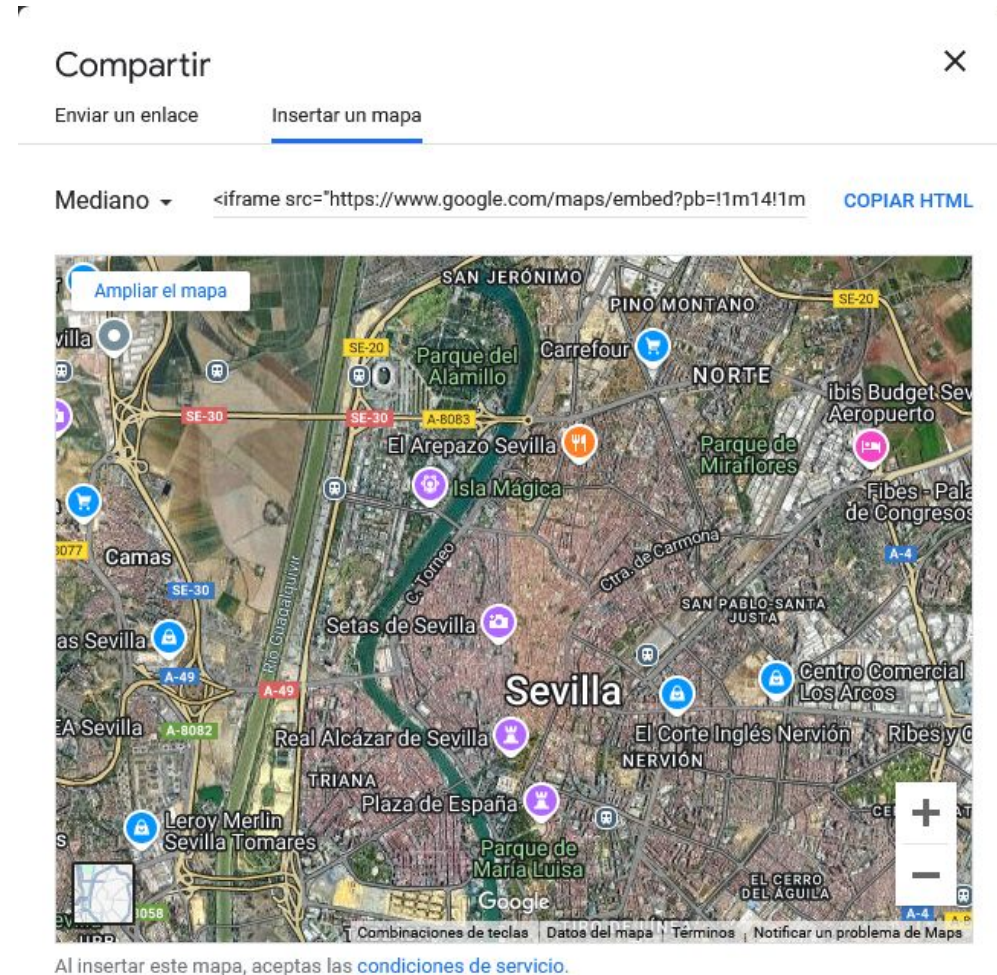
# Aplicaciones prácticas de los marcos



The screenshot shows a YouTube video player for the video "¿Por qué los JÓVENES tienen su FU...". The video title is partially visible as "¿POR QUÉ LOS JÓVENES SON RÁN POBRES?". The video player is at 0:06 / 15:59. An "Embed Video" dialog box is open, showing the following code:

```
<iframe width="560" height="315"
src="https://www.youtube.com/
embed/8F2HC2GStUs?
si=N0e9qx9nrLET_mW9" title="YouTube
video player" frameborder="0"
allow="accelerometer; autoplay;
clipboard-write; encrypted-media;
gyroscope; picture-in-picture; web-
share" referrerpolicy="strict-origin-
when-cross-origin" allowfullscreen></
iframe>
```

The dialog also includes a "Start at 0:06" checkbox and "EMBED OPTIONS" section. A "Copy" button is at the bottom right of the dialog.



The screenshot shows the Google Maps sharing interface. The "Compartir" (Share) button is selected, and the "Insertar un mapa" (Insert map) option is chosen. The map is centered on Sevilla, Spain. The embed code is displayed as:

```
<iframe src="https://www.google.com/maps/embed?pb=!1m14!1m
```

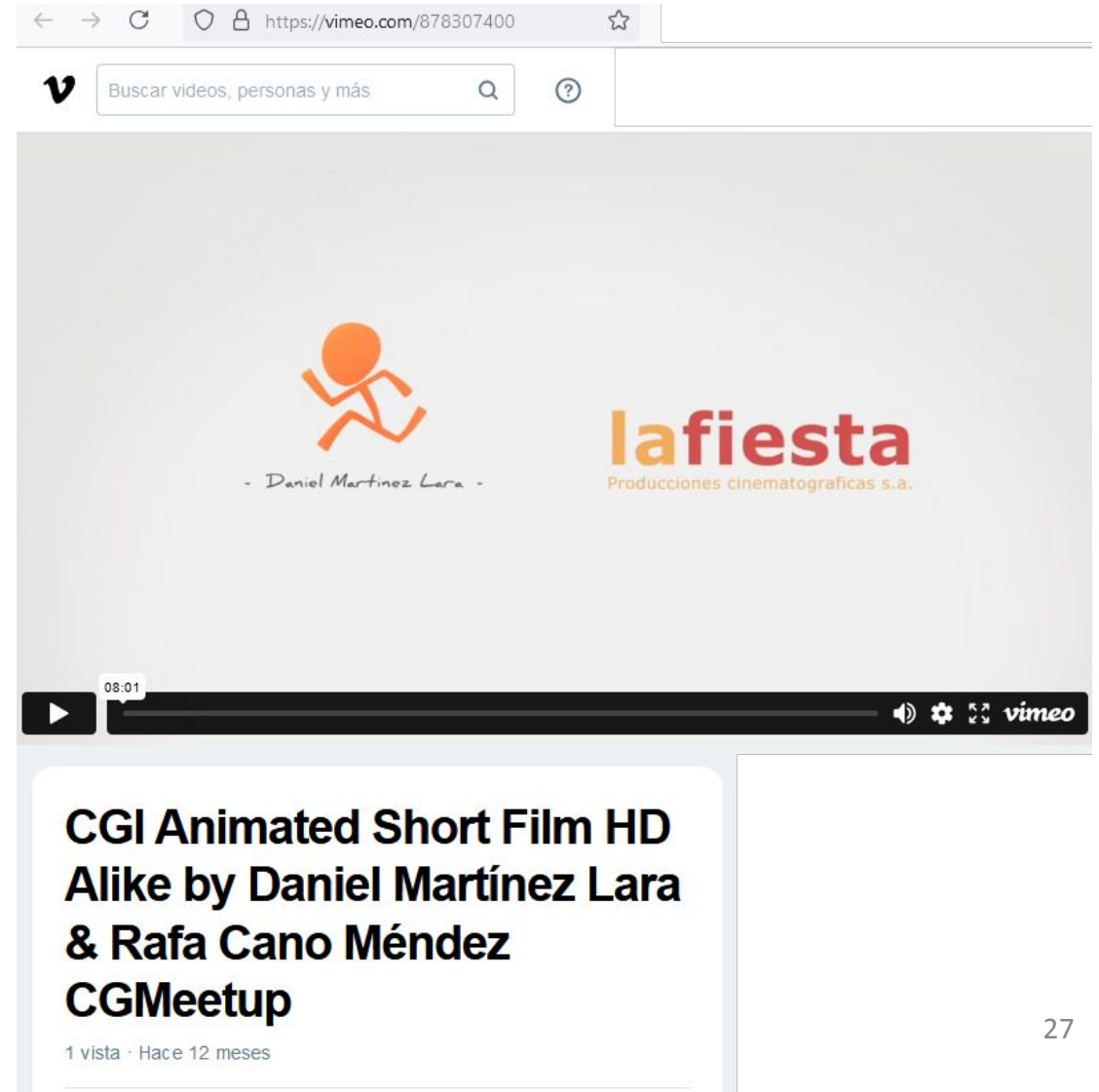
A "COPIAR HTML" (Copy HTML) button is next to the code. Below the map, there is a note: "Al insertar este mapa, aceptas las condiciones de servicio." (By inserting this map, you accept the terms of service.)

# Aplicaciones prácticas de los marcos

## Reto:

Creo una página web en la que incrustes un video de Vimeo.  
Observa el código que te ofrece la plataforma para incorporarlo en tu web.

Trata de personalizar la apariencia del video, por ejemplo, el tamaño.



# Aplicaciones prácticas de los marcos

Los iframes también se pueden usar para cargar documentos PDF directamente en la página web, sin necesidad de que el usuario descargue el archivo.

## **Reto:**

Usar un archivo disponible en la web y cargarlo dentro de un iframe en tu página web.

Prueba con un PDF, txt, ¿otro?

# Aplicaciones prácticas de los marcos

Los iframes pueden ser utilizados para ejecutar aplicaciones web *embebidas*, protegiendo la página principal del código o de acciones no deseadas que se pudieran realizar desde las aplicaciones externas.

A través del atributo **sandbox**, los iframes pueden restringir ciertos comportamientos: ejecución de scripts, envío de formularios, acceso a APIs...

## Reto:

Carga una página web que ejecute javascript dentro de un iframe y utiliza el atributo sandbox para restringir la ejecución de scripts.

Prueba con diferentes permisos y observa los resultados.

allow-scripts, allow-forms...

# Gestión de la apariencia de ventana

JavaScript permite modificar aspectos de la ventana del navegador  
...aunque estas características están limitadas por razones de seguridad.

```
window.resizeTo(800, 600); // Cambia el tamaño de la ventana  
window.moveTo(100, 100); // Mueve la ventana a la posición (100, 100)
```

`window.open` carga un recurso especificado en un contexto  
(una pestaña o una ventana) con un nombre.

No es posible cambiar el tamaño de una ventana o pestaña que no haya sido creada mediante `window.open()`.  
Tampoco es posible cambiar el tamaño cuando la ventana tiene varias pestañas.

# Gestión de la apariencia de ventana

La interfaz Screen representa una pantalla, la misma donde la ventana actual está siendo visualizada y es obtenida usando `window.screen`

Propiedades: `Screen.height`, `Screen.availHeight`... `Screen.onorientationchange`...

## **Reto:**

Redimensiona una ventana para que ocupe una cuarta parte de la pantalla disponible.



# Creación de ventanas

## Comunicación entre ventanas

Se pueden abrir nuevas ventanas utilizando `window.open()` y comunicarse entre ellas mediante `window.postMessage`.

### Reto:

Crea un botón que abra una nueva ventana con una página web diferente “receptora” y envíale un mensaje desde la ventana principal. En la ventana “receptora” se recibe y se muestra el mensaje.

### Ayuda:

- documentación -> [Window: postMessage\(\) method](#) y [Window: message event](#)
- uso de `targetOrigin '*'` para facilitar las cosas
- `esperar` un poco antes de enviar el mensaje para que la ventana esté ya cargada



# Programación funcional en JavaScript



Programación imperativa

Cocinar es un proceso en el que sigues una receta, modificas ingredientes y haces ajustes sobre la marcha. El resultado puede depender de cambios previos, lo que lo convierte en un proceso más variable.



Cuando presionas un botón (pasas un input), la máquina te da un producto (devuelve un output) sin alterar el estado de la máquina. Cada vez que pides el mismo producto, obtienes el mismo resultado y la máquina “no recuerda” interacciones anteriores.

Programación funcional

# Programación funcional en JavaScript



Programación funcional



Programación imperativa

# Programación funcional en JavaScript

## FUNDAMENTO: La **inmutabilidad**

significa que una vez que se asigna un valor a una variable, este valor no cambia; los datos no se modifican directamente, sino que se crean nuevos datos.

```
const numeros = [1, 2, 3];  
const numerosDuplicados = numeros.map(x => x * 2); // Crea un nuevo array  
  
console.log(numeros); // [1, 2, 3], el array original permanece igual  
console.log(numerosDuplicados); // [2, 4, 6]
```

# Programación funcional en JavaScript

## FUNDAMENTO: Las **funciones puras**

son funciones que cumplen con dos características principales:

- Siempre devuelve el mismo resultado para los mismos parámetros.
- No tiene efectos secundarios, no modifica nada externo ni variables globales, ni elementos DOM... nada

```
const sumar = (a, b) => a + b;
```

Pura

```
let total = 0;  
const sumarTotal = (num) => {  
  total += num; // Modifica la variable global "total", tiene efecto secundario  
  return total;  
};
```

Impura

# Programación funcional en JavaScript

## FUNDAMENTO: Las **funciones de orden superior**

son funciones que cumplen con al menos una de estas características:

- Recibe otra función como argumento.
- Devuelve una nueva función como resultado.

```
const numeros = [1, 2, 3, 4, 5];  
const duplicar = (n) => n * 2;  
  
const numerosDuplicados = numeros.map(duplicar);  
  
console.log(numerosDuplicados); // [2, 4, 6, 8, 10]
```



# Programación funcional en JavaScript

## FUNDAMENTO: Las **composición de funciones**

es el proceso de combinar varias funciones para crear una función más compleja

```
const doble = x => x * 2;  
const cuadrado = x => x ** 2;  
  
const dobleDelCuadrado = x => doble(cuadrado(x));  
  
console.log(dobleDelCuadrado(3)); // 18, equivalente a 2 * (3^2)
```

# Programación funcional en JavaScript

## FUNDAMENTO: Los **closures**

función que recuerda el ámbito en el que fue creada, incluso después de terminar.

Un closure "encierra" su contexto

función ac

expresado con function

```
const crearContador = () => {  
  let contador = 0;  
  return () => {  
    contador++;  
    return contador;  
  };  
};
```

```
function crearContador() {  
  let contador = 0; // Variable local al ámbito de crearContador  
  return function incrementar() {  
    contador += 1;  
    return contador;  
  };  
}
```

```
const contar = crearContador();  
console.log(contar()); // 1  
console.log(contar()); // 2  
console.log(contar()); // 3
```

# Programación funcional en JavaScript

## EJEMPLO:

map □ transformar un array aplicando una función a cada uno de sus elementos.

```
const numeros = [1, 2, 3, 4];  
const duplicados = numeros.map(n => n * 2);  
console.log(duplicados); // [2, 4, 6, 8]
```



# Programación funcional en JavaScript

## EJEMPLO:

filter □ filtrar elementos de un array basados en una condición.

```
const edades = [12, 18, 25, 10, 32, 19];  
const mayoresDe18 = edades.filter(edad => edad >= 18);  
console.log(mayoresDe18); // [18, 25, 32, 19]
```

# Programación funcional en JavaScript

## EJEMPLO:

reduce □ reducir un array a un solo valor.

```
const numeros = [1, 2, 3, 4];  
const sumaTotal = numeros.reduce((acumulador, n) => acumulador + n, 0);  
console.log(sumaTotal); // 10
```

# Programación funcional en JavaScript

## EJEMPLO:

usos variados de closures: encapsula el estado de forma *privada*

```
function crearContadorPrivado() {  
    let contador = 0;  
  
    return {  
        incrementar: function() { contador += 1; return contador; },  
        decrementar: function() { contador -= 1; return contador; },  
        obtenerContador: function() { return contador; }  
    };  
}  
  
const miContador = crearContadorPrivado();  
console.log(miContador.incrementar()); // 1  
console.log(miContador.incrementar()); // 2  
console.log(miContador.decrementar()); // 1  
console.log(miContador.obtenerContador()); // 1
```

# Programación funcional en JavaScript

## EJEMPLO:

usos variados de closures: crear funciones preconfiguradas

```
function saludar(prefijo) {  
  return function(nombre) {  
    return `${prefijo} ${nombre}!`;  
  };  
}  
  
const saludarEnIngles = saludar("Hello");  
console.log(saludarEnIngles("Alice")); // "Hello Alice!"  
  
const saludarEnEspanol = saludar("Hola");  
console.log(saludarEnEspanol("Carlos")); // "Hola Carlos!"
```

# Programación funcional en JavaScript

## EJEMPLO:

usos variados de closures: realizar acciones únicas

```
function crearUnaVez() {  
  let ejecutado = false;  
  return function() {  
    if (!ejecutado) {  
      ejecutado = true;  
      console.log("Ejecutando solo una vez");  
    } else {  
      console.log("Esta función ya fue ejecutada.");  
    }  
  };  
}  
  
const ejecutarUnaVez = crearUnaVez();  
ejecutarUnaVez(); // "Ejecutando solo una vez"  
ejecutarUnaVez(); // "Esta función ya fue ejecutada."
```

# Programación funcional en JavaScript

## **Reto 1:**

Dado un array de nombres, crea un nuevo array donde todos los nombres estén en mayúsculas.

## **Reto 2:**

Filtra los números pares de un array.

## **Reto 3:**

Usa reduce para sumar todos los números en un array.

