

NOTAS - JAVASCRIPT

Resumen

Este documento contiene notas de referencia sobre JavaScript en general.

Luis Carlos Salas Villalobos

e: luiscasalas16@gmail.com

w: <https://luiscasalas16.github.io>

Contenido

2	Fuentes.....	4
3	ECMAScript.....	4
4	Lenguaje	4
4.1	Comentarios	4
4.2	Consola.....	5
4.3	Importaciones	5
4.4	Tipos	5
4.4.1	Numbers.....	5
4.4.2	Strings.....	7
4.4.3	Dates.....	8
4.5	Variables.....	10
4.5.1	Var - Const - Let.....	10
4.5.2	Scopes.....	10
4.6	Operadores.....	11
4.6.1	Aritméticos	11
4.6.2	Asignación	11
4.6.3	Unarios	11
4.6.4	Comparación	12
4.6.5	Binarios.....	12
4.6.6	Lógicos.....	12
4.7	Arreglos	13
4.7.1	Constructores	13
4.7.2	Acceso	14
4.7.3	Insertar o Eliminar	14
4.7.4	Buscar	14
4.7.5	Modificar	14
4.7.6	Ordenar	14
4.7.7	Funciones	15
4.7.8	Otros.....	15
4.8	Objetos	15
4.9	Desestructuración	16
4.9.1	Arreglos	16

4.9.2	Objetos	17
4.9.3	Funciones	17
4.10	Conjuntos	18
4.11	Mapas.....	18
4.12	Json.....	18
4.13	Funciones	19
4.13.1	Declaración.....	19
4.13.2	Callbacks.....	19
4.13.3	Autoejecutables	20
4.13.4	Clausuras	20
4.13.5	Fechas.....	21
4.14	Valor vs Referencia.....	21
4.15	Condicionales	21
4.16	Bucles	21
4.17	Clases.....	21
4.17.1	Propiedades.....	22
4.17.2	Métodos	22
4.17.3	Gets y Sets	23
4.17.4	Constructores	23
4.17.5	Estáticos	24
4.17.6	Herencia	24
4.18	Módulos	25
4.18.1	Exportación	25
4.18.2	Importación.....	26
5	DOM	27
5.1	Seleccionar Elementos	27
5.1.1	Tradicionales	27
5.1.2	Modernos	27
5.2	Crear Elementos.....	28
5.2.1	Atributos.....	28
5.2.2	Fragmentos.....	28
5.3	Incorporar Elementos.....	28
5.3.1	Reemplazar.....	28

5.3.2	Insertar	29
5.3.3	Eliminar.....	29
5.4	CSS.....	29
5.5	Navegación.....	30
6	Eventos.....	31
6.1	Manejar Eventos	31
6.2	Eventos Nativos.....	32
6.3	Eventos Personalizados.....	33
6.4	Eventos Navegador	34
6.5	Emitir Eventos	36
6.6	Propagación.....	36
7	Expresiones Regulares.....	37
7.1	Definición	37
7.2	Propiedades y Banderas.....	37
7.3	Ejecución	38
7.4	Símbolos.....	39
8	Asincronía.....	40
8.1	Callbacks.....	40
8.2	Promesas.....	41
8.3	Async / Await.....	42
9	HTTP	43
9.1	Fetch.....	44
9.2	URL	45

1 FUENTES

Las fuentes de donde se ha obtenido la información en este documento son:

- <https://lenguajejs.com/javascript/>
- <https://www.w3schools.com/js/default.asp>
- <https://www.w3schools.com/jsref/default.asp>

2 ECMASCIPT

- The Original JavaScript ES1 ES2 ES3 (1997-1999)
- The First Main Revision ES5 (2009)
- The Second Revision ES6 (2015)
- Yearly Additions (2016, 2017, 2018, 2019, 2020)

Enfoque	Código escrito	Descripción
Conservador	ECMAScript 5	Incómodo de escribir. Anticuado. Compatible con navegadores nativamente.
Delegador	<i>Depende</i>	Cómodo. Rápido. Genera dependencia al framework/librería.
Evergreen	ECMAScript 6+	Cómodo. Moderno. No garantiza la compatibilidad en navegadores antiguos.
Transpilador	ECMAScript 6+	Cómodo. Moderno. Preparado para el futuro. Requiere preprocesado.

Independientemente del enfoque que se decida utilizar, el programador también puede utilizar **polyfills** o **fallbacks** para asegurarse de que ciertas características funcionarán en navegadores antiguos. También puede utilizar enfoques mixtos.

- Un **polyfill** no es más que una librería o código Javascript que actúa de « parche » o « relleno » para dotar de una característica que el navegador aún no posee, hasta que una actualización del navegador la implemente.
- Un **fallback** es algo también muy similar: un fragmento de código que el programador prepara para que en el caso de que algo no entre en funcionamiento, se ofrezca una alternativa.

3 LENGUAJE

3.1 COMENTARIOS

- No comentes detalles **redundantes**. No escribas lo que haces, escribe **por qué** lo haces.
- Mejor nombres de variables/funciones/clases descriptivas que comentarios descriptivos.
- Sé **conciso y concreto**. Resume. No escribas párrafos si no es absolutamente necesario.
- Intenta usar siempre el mismo **idioma** y **estilo** de comentarios.
- Con el tiempo, los comentarios no se suelen mantener (*modificar*), el código sí.

```
// Comentarios cortos de una sola línea. Suelen explicar la línea siguiente.  
var a = 1;  
  
var x = 45; // También se utilizan al final de una linea.  
  
/* Por otro lado, existen los comentarios múltiples de varias líneas consecutivas.  
Suelen utilizarse para explicaciones largas que requieren bastante  
espacio porque se mencionan gran cantidad de cosas :-) */
```

3.2 CONSOLA

Method	Description
<code>assert()</code>	Writes an error message to the console if a assertion is false
<code>clear()</code>	Clears the console
<code>count()</code>	Logs the number of times that this particular call to count() has been called
<code>error()</code>	Outputs an error message to the console
<code>info()</code>	Outputs an informational message to the console
<code>log()</code>	Outputs a message to the console
<code>table()</code>	Displays tabular data as a table
<code>time()</code>	Starts a timer (can track how long an operation takes)
<code>timeEnd()</code>	Stops a timer that was previously started by console.time()
<code>trace()</code>	Outputs a stack trace to the console
<code>warn()</code>	Outputs a warning message to the console

3.3 IMPORTACIONES

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

3.4 TIPOS

Tipo de dato	Descripción	Ejemplo básico
<code>NUMBER</code> Number	Valor numérico (enteros, decimales, etc...)	<code>42</code>
<code>BIGINT</code> BigInt	Valor numérico grande	<code>1234567890123456789n</code>
<code>STRING</code> String	Valor de texto (cadenas de texto, caracteres, etc...)	<code>'MZ'</code>
<code>BOOLEAN</code> Boolean	Valor booleano (valores verdadero o falso)	<code>true</code>
<code>UNDEFINED</code> undefined	Valor sin definir (variable sin inicializar)	<code>undefined</code>
<code>FUNCTION</code> Function	Función (función guardada en una variable)	<code>function() {}</code>
<code>SYMBOL</code> Symbol	Símbolo (valor único)	<code>Symbol(1)</code>
<code>OBJECT</code> Object	Objeto (estructura más compleja)	<code>{}</code>

3.4.1 Numbers

```
// Notación literal (preferida)
const number = 4;
const decimal = 15.8;
const legibleNumber = 5_000_000;
```

Propiedad o Método	Descripción
<code>NAN</code> Number.NaN	Es equivalente a <code>NaN</code> . Valor que no puede representarse como número.
<code>BOOLEAN</code> Number.isNaN(number)	Comprueba si <code>number</code> no es un número.

Método	Descripción
<code>NUMBER</code> Number.parseInt(text)	Convierte un <code>STRING</code> <code>text</code> en un <code>NUMBER</code> entero.
<code>NUMBER</code> Number.parseInt(text, radix)	Idem, pero el <code>STRING</code> tiene un número en base <code>NUMBER</code> <code>radix</code> .

Método	Descripción
<code>NUMBER Number.parseFloat(text)</code>	Convierte un <code>STRING</code> <code>text</code> en un <code>NUMBER</code> decimal.
<code>NUMBER Number.parseFloat(text, radix)</code>	Idem, pero el <code>STRING</code> tiene un número en base <code>NUMBER</code> <code>radix</code> .

Método	Descripción
<code>STRING Number.toString()</code>	Convierte un <code>NUMBER</code> en <code>STRING</code> .
<code>STRING Number.toString(radix)</code>	Convierte un <code>NUMBER</code> a la base <code>NUMBER</code> <code>radix</code> y lo devuelve como texto.

Método	Descripción	Ejemplo
<code>NUMBER Math.abs(x)</code>	Devuelve el <u>valor absoluto</u> de <code>x</code> .	$ x $
<code>NUMBER Math.sign(x) <small>ES2016</small></code>	Devuelve el signo del número: <code>1</code> positivo, <code>-1</code> negativo	
<code>NUMBER Math.exp(x)</code>	<u>Exponenciación</u> . Devuelve el número <code>e</code> elevado a <code>x</code> .	e^x
<code>NUMBER Math.expm1(x) <small>ES2016</small></code>	Equivalente a <code>Math.exp(x) - 1</code> .	$e^x - 1$
<code>NUMBER Math.max(a, b, c...)</code>	Devuelve el número más grande de los indicados por parámetro.	
<code>NUMBER Math.min(a, b, c...)</code>	Devuelve el número más pequeño de los indicados por parámetro.	
<code>NUMBER Math.pow(base, exp)</code>	<u>Potenciación</u> . Devuelve el número <code>base</code> elevado a <code>exp</code> .	<code>base^{exp}</code>
<code>NUMBER Math.sqrt(x)</code>	Devuelve la <u>raíz cuadrada</u> de <code>x</code> .	\sqrt{x}
<code>NUMBER Math.cbrt(x) <small>ES2015</small></code>	Devuelve la <u>raíz cúbica</u> de <code>x</code> .	$\sqrt[3]{x}$
<code>NUMBER Math.imul(a, b) <small>ES2016</small></code>	Equivalente a <code>a * b</code> , pero a nivel de bits.	
<code>NUMBER Math.clz32(x) <small>ES2016</small></code>	Devuelve el número de ceros a la izquierda de <code>x</code> en binario (32 bits).	

Método	Descripción
<code>NUMBER Math.random()</code>	Devuelve un número al azar entre <code>0</code> y <code>1</code> con 16 decimales.

Método	Descripción	Ejemplo
<code>NUMBER Math.log(x)</code>	Devuelve el <u>logaritmo natural</u> en base <code>e</code> de <code>x</code> .	$\log_e x \text{ o } \ln x$
<code>NUMBER Math.log10(x) <small>ES2016</small></code>	Devuelve el <u>logaritmo decimal</u> (en base 10) de <code>x</code> .	$\log_{10} x \text{ o } \log x$
<code>NUMBER Math.log2(x) <small>ES2016</small></code>	Devuelve el <u>logaritmo binario</u> (en base 2) de <code>x</code> .	$\log_2 x$
<code>NUMBER Math.log1p(x) <small>ES2016</small></code>	Devuelve el logaritmo natural de <code>(1+x)</code> .	$\log_e (1+x) \text{ o } \ln (1+x)$

Método	Descripción
<code>NUMBER Math.round(x)</code>	Devuelve <code>x</code> con <u>redondeo</u> (<u>el entero más cercano</u>)
<code>NUMBER Math.ceil(x)</code>	Devuelve <code>x</code> con <u>redondeo superior</u> (<u>el entero más alto</u>)
<code>NUMBER Math.floor(x)</code>	Devuelve <code>x</code> con <u>redondeo inferior</u> (<u>el entero más bajo</u>)
<code>NUMBER Math.fround(x) <small>ES2016</small></code>	Devuelve <code>x</code> con <u>redondeo</u> (<u>flotante con precisión simple</u>)
<code>NUMBER Math.trunc(x) <small>ES2016</small></code>	Trunca el número <code>x</code> (<u>devuelve sólo la parte entera</u>)

Método	Descripción
<code>NUMBER Math.sin(x)</code>	Seno de <code>x</code>
<code>NUMBER Math.asin(x)</code>	Arcoseno de <code>x</code>
<code>NUMBER Math.sinh(x) <small>ES2015</small></code>	Seno hiperbólico de <code>x</code>
<code>NUMBER Math.asinh(x) <small>ES2015</small></code>	Arcoseno hiperbólico de <code>x</code>
<code>NUMBER Math.cos(x)</code>	Coseno de <code>x</code>
<code>NUMBER Math.acos(x)</code>	Arcocoseno de <code>x</code>
<code>NUMBER Math.cosh(x) <small>ES2015</small></code>	Coseno hiperbólico de <code>x</code>
<code>NUMBER Math.acosh(x) <small>ES2015</small></code>	Arcocoseno hiperbólico de <code>x</code>
<code>NUMBER Math.tan(x)</code>	Tangente de <code>x</code>
<code>NUMBER Math.atan(x)</code>	Arcotangente de <code>x</code>
<code>NUMBER Math.tanh(x) <small>ES2015</small></code>	Tangente hiperbólica de <code>x</code>
<code>NUMBER Math.atanh(x) <small>ES2015</small></code>	Arcotangente hiperbólica de <code>x</code>
<code>NUMBER Math.atan2(x, y)</code>	Arcotangente del concreto de <code>x/y</code>
<code>NUMBER Math.hypot(a, b...) <small>ES2015</small></code>	Devuelve la raíz cuadrada de $a^2 + b^2 + \dots$

3.4.2 Strings

- **Comillas simples:** `'`
- **Comillas dobles:** `"`
- **Backticks:** ``` (ver más adelante, en *Interpolación de variables*)

Propiedad	Descripción
<code>NUMBER .length</code>	Devuelve el número de caracteres totales del texto.

Método	Descripción
<code>STRING .charAt(pos)</code>	Devuelve el carácter de la posición <code>pos</code> . Similar al operador <code>[]</code> .
<code>NUMBER .indexOf(text)</code>	Devuelve la primera posición del texto <code>text</code> .
<code>NUMBER .indexOf(text, from)</code>	Idem al anterior, partiendo desde la posición <code>from</code> .
<code>NUMBER .lastIndexOf(text)</code>	Devuelve la última posición del texto <code>text</code> .
<code>NUMBER .lastIndexOf(text, from)</code>	Idem al anterior, partiendo desde <code>from</code> hacia el inicio.

Método	Descripción
<code>STRING .repeat(num) <small>ES2015</small></code>	Devuelve el <code>STRING</code> repetido <code>num</code> veces.
<code>STRING .substring(start, end)</code>	Devuelve el <code>substring</code> desde la posición <code>start</code> hasta <code>end</code> .
<code>STRING .substr(start, size)</code>	Devuelve el <code>substring</code> desde la posición <code>start</code> hasta <code>start+size</code> .
<code>STRING .slice(start, end)</code>	Idem a <code>.substr()</code> con <u>leves diferencias</u> .

Método	Descripción
<code>ARRAY .split(text)</code>	Separa el texto en varias partes, usando <code>STRING text</code> como separador.
<code>ARRAY .split(text, limit)</code>	Idem, pero crea como máximo <code>NUMBER limit</code> fragmentos.
<code>ARRAY .split(regexp)</code>	Separa el texto usando la <code>REGEXP regexp</code> como separador.
<code>ARRAY .split(regexp, limit)</code>	Idem, pero crea como máximo <code>NUMBER limit</code> fragmentos.

Método	Descripción
<code>BOOLEAN .startsWith(text, from) <small>ES2018</small></code>	Comprueba si el texto comienza por <code>text</code> .
<code>BOOLEAN .endsWith(text, to) <small>ES2018</small></code>	Comprueba si el texto termina por <code>text</code> .
<code>BOOLEAN .includes(text, from) <small>ES2019</small></code>	Comprueba si el texto contiene el subtexto <code>text</code> .

Método	Descripción
<code>NUMBER .search(regexp)</code>	Busca un patrón que encaje con <code>regexp</code> y devuelve la <code>NUMBER</code> posición encontrada.
<code>ARRAY .match(regexp)</code>	Idem a la anterior, pero devuelve las coincidencias encontradas.
<code>ARRAY .matchAll(regexp) <small>ES2022</small></code>	Idem a la anterior, pero devuelve un iterador para iterar por cada coincidencia.

Método	Descripción
<code>STRING .replace(text, newText)</code>	Reemplaza la primera aparición del <code>STRING text</code> por <code>newText</code> .
<code>STRING .replace(regexp, newText)</code>	Idem, pero busca a partir de una <code>REGEXP</code> en lugar de un <code>STRING</code> .
<code>STRING .replaceAll(text, newText) <small>ES2021</small></code>	Reemplaza todas las apariciones del texto <code>text</code> por <code>newText</code> .
<code>STRING .replaceAll(regexp, newText) <small>ES2021</small></code>	Idem, pero busca a partir de una <code>REGEXP</code> en lugar de un <code>STRING</code> .

Método	Descripción
<code>STRING .toLowerCase()</code>	Devuelve el <code>STRING</code> transformado a minúsculas.
<code>STRING .toUpperCase()</code>	Devuelve el <code>STRING</code> transformado a mayúsculas.
<code>STRING .padStart(size, text) <small>ES2017</small></code>	Devuelve el <code>STRING</code> rellenando el inicio con <code>text</code> hasta llegar al tamaño <code>size</code> .
<code>STRING .padEnd(size, text) <small>ES2017</small></code>	Devuelve el <code>STRING</code> rellenando el final con <code>text</code> hasta llegar al tamaño <code>size</code> .
<code>STRING .trimStart() <small>ES2017</small></code>	Devuelve el <code>STRING</code> eliminando espacios a la izquierda del texto.
<code>STRING .trimEnd() <small>ES2017</small></code>	Devuelve el <code>STRING</code> eliminando espacios a la derecha del texto.
<code>STRING .trim()</code>	Devuelve el <code>STRING</code> eliminando espacios a la izquierda y derecha del texto.

```
const firstWord = "frase";
const secondWord = "concatenada";

"Una " + firstWord + " bien " + secondWord; // 'Una frase bien concatenada'
```

```
const firstWord = "frase";
const secondWord = "concatenada";

`Una ${firstWord} mejor ${secondWord}`; // 'Una frase mejor concatenada'
```

```
const magicalWord = `<strong>Magical Word</strong>`;
const template = `
  <div class="container">
    ${magicalWord}
  </div>
`;
```

3.4.3 Dates

Constructor	Descripción
<code>DATE new Date()</code>	Obtiene la fecha del momento actual.
<code>DATE new Date(str)</code>	Convierte el texto con formato <code>YYYY/MM/DD HH:MM:SS</code> a una fecha.
<code>DATE new Date(numero)</code>	Convierte el <code>número</code> (en formato <code>Tiempo UNIX</code>) a una fecha UTC.

```
// Obtenemos la fecha actual y la guardamos en la variable date
const date = new Date();

// Obtenemos la fecha 30 de Enero de 2018, a las 23h 30m 14seg
const date = new Date("2018/01/30 23:30:14");

// Obtenemos la fecha del juicio final a partir de un timestamp o Tiempo UNIX
const date = new Date(872817240000);
```

El **Tiempo UNIX** (*o UNIX timestamp*) es un formato numéricico utilizado para calcular una fecha en UNIX. Es una forma poco práctica y legible para humanos, pero muy eficiente en términos informáticos. Se trata de un número que representa la cantidad de **segundos** transcurridos desde la fecha **1/1/1970, a las 00:00:00**.

Método	Descripción
<code>NUMBER Date.now()</code>	Devuelve el Tiempo UNIX de la fecha actual. Equivalente a <code>+new Date()</code> .
<code>NUMBER Date.parse(str)</code>	Convierte un STRING de fecha a Tiempo UNIX . Equivalente a <code>+new Date(str)</code> .

Método	Descripción
<code>NUMBER .getDay()</code>	Devuelve el dia de la semana: OJO: 0 Domingo, 6 Sábado.
<code>NUMBER .getFullYear()</code>	Devuelve el año con 4 cifras.
<code>NUMBER .getMonth()</code>	Devuelve la representación interna del mes. OJO: 0 Enero - 11 Diciembre.
<code>NUMBER .getDate()</code>	Devuelve el dia del mes.
<code>NUMBER .getHours()</code>	Devuelve la hora. OJO: Formato militar; 23 en lugar de 11 .
<code>NUMBER .getMinutes()</code>	Devuelve los minutos.
<code>NUMBER .getSeconds()</code>	Devuelve los segundos.
<code>NUMBER .getMilliseconds()</code>	Devuelve los milisegundos.
<code>NUMBER .getTime()</code>	Devuelve el UNIX Timestamp : segundos transcurridos desde 1/1/1970 .
<code>NUMBER .getTimezoneOffset()</code>	Diferencia horaria (en min) de la hora local respecto a UTC (ver más adelante).

Método	Descripción
<code>NUMBER .setFullYear(year)</code>	Altera el año de la fecha, cambiándolo por year . Formato de 4 dígitos.
<code>NUMBER .setMonth(month)</code>	Altera el mes de la fecha, cambiándolo por month . Ojo: 0-11 (Ene-Dic).
<code>NUMBER . setDate(day)</code>	Altera el dia de la fecha, cambiándolo por day .
<code>NUMBER .setHour(hour)</code>	Altera la hora de la fecha, cambiándola por hour .
<code>NUMBER .setMinutes(min)</code>	Altera los minutos de la fecha, cambiándolos por min .
<code>NUMBER .setSeconds(sec)</code>	Altera los segundos de la fecha, cambiándolos por sec .
<code>NUMBER .setMilliseconds(ms)</code>	Altera los milisegundos de la fecha, cambiándolos por ms .

Método	Descripción
Formato por defecto	<code>Fri Aug 24 2018 00:23:31 GMT+0100</code>
<code>STRING .toDateString()</code>	Devuelve formato sólo de fecha: <code>Fri Aug 24 2018</code>
<code>STRING .toLocaleDateString()</code>	Idem al anterior, pero en el formato regional actual: <code>24/08/2018</code>
<code>STRING .toTimeString()</code>	Devuelve formato sólo de hora: <code>00:23:24 GMT+0100 ...</code>
<code>STRING .toLocaleTimeString()</code>	Idem al anterior, pero en el formato regional actual: <code>0:26:37</code>
<code>STRING .toISOString()</code>	Devuelve la fecha en el formato <code>ISO 8601 : 2018-08-23T23:27:29.380Z</code>
<code>STRING . toJSON()</code>	Idem al anterior, pero asegurándose que será compatible con JSON.
<code>STRING .toUTCString()</code>	Devuelve la fecha, utilizando UTC (ver más adelante).

Objeto	Descripción
<code>Intl.DateTimeFormat</code>	Crea un objeto de formato con las preferencias de tu región (o la región indicada).

Método	Descripción
STRING <code>.format(date)</code>	Formatea la fecha <code>date</code> con la configuración de región iniciada.
ARRAY <code>.formatToParts(date)</code>	Idem, dividiendo sus partes en un array de objetos.
STRING <code>.formatRange(a, b)</code>	Crea un rango con las fechas <code>a-b</code> usando la configuración de región.
ARRAY <code>.formatRangeToParts(a, b)</code>	Idem, pero divide sus partes en un array de objetos.
OBJECT <code>.resolvedOptions()</code>	Devuelve las opciones de región definidas en la instancia.

```
const esDate = new Intl.DateTimeFormat("es").format(date);
"30/1/2021"

const enDate = new Intl.DateTimeFormat("en-US").format(date);
"1/30/2021"
```



3.5 VARIABLES

3.5.1 Var - Const - Let

- **var:** The most commonly used variable in JavaScript is var. It can be redeclared and its value can be reassigned, but only inside the context of a function. When the JavaScript code is run, variables defined using var are moved to the top. An example of a variable declared using the "var" keyword in JavaScript is shown below:

```
var x = 140; // variable x can be reassigned a new value and also redeclared
```

- **const:** const variables in JavaScript cannot be used before they appear in the code. They can neither be reassigned values, that is, their value remains fixed throughout the execution of the code, nor can they be redeclared. An example of a variable declared using the "const" keyword in JavaScript is shown below:

```
const x = 5; // variable x cannot be reassigned a new value or redeclared
```

- **let:** The let variable, like const, cannot be redeclared. But they can be reassigned a value. An example of a variable declared using the "let" keyword in JavaScript is shown below:

```
let x = 202; // variable x cannot be redeclared but can be reassigned a new value
```

OJO: Las mayúsculas y minúsculas en los nombres de las variables de Javascript **importan**. No es lo mismo una variable llamada **precio** que una variable llamada **Precio**, pueden contener valores diferentes.



3.5.2 Scopes

1. **Scope:** The accessibility or visibility of variables in JavaScript is referred to as scope. That is, which sections of the program can access a given variable and where the variable can be seen. There are usually three types of scopes:

- **Global Scope:** The global scope includes any variable that is not contained within a function or block (a pair of curly braces). Global scope variables can be accessed from anywhere in the program. An example showing the global scope of a variable is given below:

```
var hello = 'Hello!';
function sayHello() {
  console.log(hello);
}
// 'Hello!' gets logged
sayHello();
```

- **Local or Function Scope:** Variables declared inside a function are local variables. They can only be accessed from within that function; they are not accessible from outside code. An example showing local scope of a variable is given below:

```
function sayHello() {
  var hello = 'Hello!';
  console.log(hello);
}
// 'Hello!' gets logged
sayHello();
```

```
console.log(hello); // Uncaught ReferenceError: hello is not defined
```

- **Block Scope:** Unlike var variables, let and const variables can be scoped to the nearest pair of curly brackets in ES6. They can't be reached from outside that pair of curly braces, which means they can't be accessed from the outside. An example showing the block scope of a variable is given below:

```
{
  let hello = 'Hello!';
  var language = 'Hindi';
  console.log(hello); // 'Hello!' gets logged
}
console.log(language); // 'Hindi' gets logged
console.log(hello); // Uncaught ReferenceError: hello is not defined
```

2. Scope Chain: When a variable is used in JavaScript, the JavaScript engine searches the current scope for the variable's value. If it can't find the variable in the inner scope, it will look into the outer scope until it finds it or reaches the global scope.

If it still can't identify the variable, it will either return an error or implicitly declare the variable in the global scope (if not in strict mode). Let us take into consideration the following example:

```
let a = 'a';
function foo() {
  let b = 'b';
  console.log(b); // 'b' gets logged
  console.log(a); // 'a' gets logged
  randomNumber = 33;
  console.log(randomNumber); // 33 gets logged
}
foo();
```

When the function `foo()` is called, the JavaScript engine searches for the `'b'` variable in the current scope and finds it. Then it looks for the `'a'` variable in the current scope, which it can't find, so it moves on to the outer scope, where it finds it (i.e. global scope).

3.6 OPERADORES

3.6.1 Aritméticos

Nombre	Operador	Descripción
Suma	<code>a + b</code>	Suma el valor de <code>a</code> al valor de <code>b</code> .
Resta	<code>a - b</code>	Resta el valor de <code>b</code> al valor de <code>a</code> .
Multiplicación	<code>a * b</code>	Multiplica el valor de <code>a</code> por el valor de <code>b</code> .
División	<code>a / b</code>	Divide el valor de <code>a</code> entre el valor de <code>b</code> .
Módulo	<code>a % b</code>	Devuelve el resto de la división de <code>a</code> entre <code>b</code> .
Exponenciación	<code>a ** b</code>	Eleva <code>a</code> a la potencia de <code>b</code> , es decir, a^b . Equivalente a <code>Math.pow(a, b)</code> .

3.6.2 Asignación

Nombre	Operador	Descripción
Asignación	<code>c = a + b</code>	Asigna el valor de la parte derecha (<i>en este ejemplo, una suma</i>) a <code>c</code> .
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> .

3.6.3 Unarios

Nombre	Operador	Descripción
Incremento	<code>a++</code>	Usa el valor de <code>a</code> y luego lo incrementa. También llamado postincremento .
Decremento	<code>a--</code>	Usa el valor de <code>a</code> y luego lo decremente. También llamado postdecremento .
Incremento previo	<code>++a</code>	Incrementa el valor de <code>a</code> y luego lo usa. También llamado preincremento .
Decremento previo	<code>--a</code>	Decrementa el valor de <code>a</code> y luego lo usa. También llamado predecremento .
Resta unaria	<code>-a</code>	Cambia de signo (niega) a <code>a</code> .

3.6.4 Comparación

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

```

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 -número-)
"5" === 5   // false  (no son idénticos, coincide su valor, pero no su tipo de dato -string y número-)

```

3.6.5 Binarios

Nombre	Operador	Descripción
Operador AND	<code>a & b</code>	Devuelve 1 si ambos operandos son 1 .
Operador OR	<code>a b</code>	Devuelve 1 si al menos un operando es 1 .
Operador XOR (OR exclusivo)	<code>a ^ b</code>	Devuelve 1 si ambos operandos son diferentes.
Operador NOT (unario)	<code>~a</code>	Invierte los bits del operando (por ejemplo, 000101 pasa a 111010). Trunca a 32 bits.
Operador LEFTSHIFT	<code>a << b</code>	Desplazamiento de bits hacia la izquierda. Ej: 11 (3) pasa a 110 (6).
Operador RIGHTSHIFT	<code>a >> b</code>	Desplazamiento de bits hacia la derecha. Ej: 11 (3) pasa a 1 (1).
Operador RIGHTSHIFT sin signo	<code>a >>> b</code>	Desplazamiento de bits hacia la derecha, como un operador sin signo.

a	b	AND	OR	XOR	NOT AND	NOT OR	NOT XOR
---	---	---	---	---	---	---	---
0	0	0	0	0	1	1	1
0	1	0	1	1	1	0	0
1	0	0	1	1	1	0	0
1	1	1	1	0	0	0	1

3.6.6 Lógicos

Nombre	Operador	Descripción
Operador lógico AND	<code>a && b</code>	Devuelve a si es false , sino devuelve b .
Operador ternario <code>?:</code>	<code>a ? b : c</code>	Si a es true , devuelve b , sino devuelve c .
Operador lógico OR	<code>a b</code>	Devuelve a si es true , sino devuelve b .
Operador lógico Nullish coalescing	<code>a ?? b</code>	Devuelve b si a es null o undefined , sino devuelve a .
Operador de asignación lógica nula <code>??=</code>	<code>a ??= b</code>	Es equivalente a <code>a ?? (a = b)</code>
Operador de encadenamiento opcional <code>?.</code>	<code>data?.name</code>	Permite intentar acceder a una propiedad, aunque su padre no exista.
Operador unario lógico NOT	<code>!a</code>	Invierte el valor. Si es true devuelve false y viceversa.

```

0 && undefined      // 0 (se evalua como false && false, devuelve el primero)
undefined && 0       // undefined (se evalua como false && false, devuelve el primero)
55 && null          // null (se evalua como true && false, devuelve el segundo)
null && 55            // null (se evalua como false && true, devuelve el primero)
44 && 20              // 20 (se evalua como true && true, devuelve el segundo)

```

```

45 && "OK"           // "OK"
false && "OK"          // false

const doTask = () => "OK!"; // Creamos función que devuelve "OK!"
isCorrect && doTask()     // Si isCorrect es true, ejecuta doTask()

```

```

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)

```

```

const userName = name || "Unknown name";
"Manz" || "Unknown name"    // "Manz"
null || "Unknown name"      // "Unknown name"
false || "Unknown name"     // "Unknown name"
undefined || "Unknown name" // "Unknown name"
0 || "Unknown name"         // "Unknown name"

```

Tips

```

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

// Con asignación lógica nula
x ??= 50;

```

```

user.attrs.power      // TypeError: Cannot read properties of undefined (reading 'power')
user.attrs.life       // TypeError: Cannot read properties of undefined (reading 'life')

user.attrs && user.attrs.life // Evitamos el error (comprobamos si existe attrs antes de continuar)

```

```

user.attrs?.power      // undefined
user.attrs?.life       // undefined

```

3.7 ARREGLOS

3.7.1 Constructores

Constructor	Descripción
ARRAY <code>new Array(NUMBER size)</code>	Crea un array vacío de tamaño <code>size</code> . Sus valores no están definidos, pero son <code>UNDEFINED</code> .
ARRAY <code>new Array(e1, e2...)</code>	Crea un array con los elementos indicados.
ARRAY <code>[e1, e2...]</code>	Simplemente, los elementos dentro de corchetes: [] Notación preferida .

3.7.2 Acceso

Forma	Descripción
NUMBER <code>.length</code>	Propiedad que devuelve el número de elementos del array.
OBJECT <code>[pos]</code>	Operador que devuelve (o modifica) el elemento número <code>pos</code> del array.
OBJECT <code>.at(pos)</code> ES2022	Método que devuelve el elemento en la posición <code>pos</code> . Números negativos en orden inverso.

3.7.3 Insertar o Eliminar

Método	Descripción
NUMBER <code>.push(e1, e2, e3...)</code>	Añade uno o varios elementos al final del array. Devuelve el tamaño del array.
OBJECT <code>.pop()</code>	Elimina el último elemento del array. Devuelve dicho elemento.
NUMBER <code>.unshift(e1, e2, e3...)</code>	Añade uno o varios elementos al inicio del array. Devuelve el tamaño del array.
OBJECT <code>.shift()</code>	Elimina el primer elemento del array. Devuelve dicho elemento.

3.7.4 Buscar

Método	Descripción
BOOLEAN <code>.includes(element)</code> ES2016	Comprueba si <code>element</code> está incluido en el array.
BOOLEAN <code>.includes(element, from)</code> ES2016	Idem, pero partiendo desde la posición <code>from</code> del array.
NUMBER <code>.indexOf(element)</code>	Devuelve la posición de la primera aparición de <code>element</code> . Devuelve <code>-1</code> si no existe.
NUMBER <code>.indexOf(element, from)</code>	Idem, pero partiendo desde la posición <code>from</code> del array.
NUMBER <code>.lastIndexOf(element)</code>	Devuelve la posición de la última aparición de <code>element</code> . Devuelve <code>-1</code> si no existe.
NUMBER <code>.lastIndexOf(element, from)</code>	Idem, pero partiendo desde la posición <code>from</code> del array.

3.7.5 Modificar

Método	Descripción
ARRAY <code>.slice(start, end)</code>	Devuelve los elementos desde la posición <code>start</code> hasta <code>end</code> (excluido).
ARRAY <code>.splice(start, size)</code>	Altera el array, eliminando <code>size</code> elementos desde posición <code>start</code> . Muta
ARRAY <code>.splice(start, size, e1, e2...)</code>	Idem. Además inserta <code>e1, e2...</code> en la posición <code>start</code> . Muta
ARRAY <code>.copyWithin(pos, start, end)</code> ES2018	Altera el array, modificando en <code>pos</code> y copiando los ítems desde <code>start</code> a <code>end</code> . Muta
ARRAY <code>.fill(element, start, end)</code> ES2018	Altera los elementos del <code>ARRAY</code> desde <code>start</code> hasta <code>end</code> . Muta

3.7.6 Ordenar

Método	Descripción
ARRAY <code>.reverse()</code>	Invierte el orden de elementos del array.
ARRAY <code>.sort()</code>	Ordena los elementos del array bajo un criterio de ordenación alfabética .
ARRAY <code>.sort(criterio)</code>	Ordena los elementos del array bajo un criterio de ordenación determinado por <code>FUNCTION criterio</code> .

3.7.7 Funciones

Método	Descripción
<code>UNDEFINED .forEach(f)</code>	Ejecuta la función definida en <code>f</code> por cada uno de los elementos del array.
Comprobaciones	
<code>BOOLEAN .every(f)</code>	Comprueba si todos los elementos del array cumplen la condición de <code>f</code> .
<code>BOOLEAN .some(f)</code>	Comprueba si al menos un elemento del array cumple la condición de <code>f</code> .
Transformadores y filtros	
<code>ARRAY .map(f)</code>	Construye un array con lo que devuelve <code>f</code> por cada elemento del array.
<code>ARRAY .filter(f)</code>	Filtrá un array y se queda sólo con los elementos que cumplen la condición de <code>f</code> .
<code>OBJECT .flatMap(level)</code>	Aplana el array al nivel <code>level</code> indicado.
<code>OBJECT .flatMap(f)</code>	Aplana cada elemento del array, transformándolo según <code>f</code> . Equivale a <code>.map().flatMap(1)</code> .
Búsquedas	
<code>NUMBER .findIndex(f) ES2015</code>	Devuelve la posición del elemento que cumple la condición de <code>f</code> .
<code>OBJECT .find(f) ES2015</code>	Devuelve el elemento que cumple la condición de <code>f</code> .
<code>OBJECT .findLastIndex(f)</code>	Idem a <code>findIndex()</code> , pero empezando a buscar desde el último elemento al primero.
<code>OBJECT .findLast(f)</code>	Idem a <code>find()</code> , pero empezando a buscar desde el último elemento al primero.
Acumuladores	
<code>OBJECT .reduce(f, initial)</code>	Ejecuta <code>f</code> con cada elemento (de izq a der), acumulando el resultado.
<code>OBJECT .reduceRight(f, initial)</code>	Idem al anterior, pero en orden de derecha a izquierda.

3.7.8 Otros

Método	Descripción
<code>ARRAY Array.from(obj) ES2015</code>	Intenta convertir el <code>obj</code> en un array.
<code>ARRAY Array.from(obj, fmap) ES2015</code>	Idem, pero ejecuta la función <code>fmap</code> por cada elemento. Equivalente a <code>.map()</code>
<code>ARRAY Array.from({ length: size })</code>	Crea un array a partir de un <code>OBJECT</code> de tamaño <code>size</code> , relleno de <code>UNDEFINED</code>
<code>ARRAY .concat(e1, e2, e3...)</code>	Devuelve los elementos pasados por parámetro concatenados al final del array.
<code>STRING .join(sep)</code>	Une los elementos del array mediante separadores <code>sep</code> en un <code>STRING</code> .

3.8 OBJETOS

```
const player = {
  name: "Manz",
  life: 99,
  power: 10,
};
```

```
// Notación con puntos (preferida)
console.log(player.name); // Muestra "Manz"
console.log(player.life); // Muestra 99

// Notación con corchetes
console.log(player["name"]); // Muestra "Manz"
console.log(player["life"]); // Muestra 99
```

```
// FORMA 1: A través de notación con puntos
const player = {};

player.name = "Manz";
player.life = 99;
player.power = 10;

// FORMA 2: A través de notación con corchetes
const player = {};

player["name"] = "Manz";
player["life"] = 99;
player["power"] = 10;
```

```
const user = {
  name: "Manz",
  talk: function() { return "Hola"; }
};

user.name;      // Es una variable (propiedad), devuelve "Manz"
user.talk();    // Es una función (método), se ejecuta y devuelve "Hola"
```

Método	Descripción
ARRAY <code>Object.keys(obj)</code> <small>ES2015</small>	Itera el <code>obj</code> y devuelve sus propiedades o keys .
ARRAY <code>Object.values(obj)</code> <small>ES2015</small>	Itera el <code>obj</code> y devuelve los valores de sus propiedades.
ARRAY <code>Object.entries(obj)</code> <small>ES2015</small>	Itera el <code>obj</code> y devuelve un <code>ARRAY</code> con los pares [key, valor] .
OBJECT <code>Object.fromEntries(array)</code> <small>ES2015</small>	Construye un objeto con un array de pares [key, valor] .

3.9 DESESTRUCTURACIÓN

3.9.1 Arreglos

```
const elements = [5, 2];
const [first, last] = elements;    // first = 5, last = 2

const elements = [5, 4, 3, 2];
const [first, second] = elements; // first = 5, second = 4, rest = discard

const elements = [5, 4, 3, 2];
const [first, , third] = elements; // first = 5, third = 3, rest = discard

const elements = [4];
const [first, second] = elements; // first = 4, second = undefined
```

```
const debug = (...param) => console.log(param);
debug(1, 2, 3, 4, 5);

// [1, 2, 3, 4, 5]
```

```
const elements = [5, 4, 3, 2];
const [first, ...rest] = elements; // first = 5, rest = [4, 3, 2]
```

3.9.2 Objetos

```
const user = {  
    name: "Manz",  
    role: "streamer",  
    life: 99  
}  
  
const { name, role, life } = user;  
  
console.log(name);  
console.log(role, life);
```

```
const { name, role: type, life } = user;  
  
console.log({ name, type, life });
```

```
const { name, role = "normal user", life = 100 } = user;  
  
console.log({ name, role, life });
```

```
// Extraemos propiedad attributes (objeto con 3 propiedades)  
const { attributes } = user;  
console.log(attributes);  
  
// Extraemos propiedad height (number)  
const { attributes: { height } } = user;  
console.log(height);  
  
// Extraemos propiedad height (number) y la cambiamos por nombre size  
const { attributes: { height: size } } = user;  
console.log(size);
```

```
const user = {  
    name: "Manz",  
    role: "streamer",  
    life: 99  
}  
  
const { name, ...rest } = user; // name = 'Manz', rest = {role: 'streamer', life: 99}
```

3.9.3 Funciones

```
const user = {  
    name: "Manz",  
    role: "streamer",  
    life: 99  
}  
  
function show({ name, role, life }) {  
    const stars = "★".repeat(life / 20);  
    return `Nombre: ${name} (${role}) ${stars}`;  
}  
  
show(user); // "Nombre: Manz (streamer) ★★★★"
```

3.10 CONJUNTOS

```
const set = new Set();           // Set({})
const set = new Set([5, 6, 7, 8, 9]); // Set({5, 6, 7, 8, 9}) (Conjunto con 5 elementos)
const set = new Set([5, 5, 7, 8, 9]); // Set({5, 7, 8, 9}) (Conjunto con 4 elementos)

set.constructor.name;           // "Set"
```

Propiedad o Método	Descripción
NUMBER <code>.size</code>	Propiedad que devuelve el número de elementos que tiene el conjunto.
SET <code>.add(element)</code>	Añade un elemento al conjunto (si no está repetido) y devuelve el set. Muta
BOOLEAN <code>.has(element)</code>	Comprueba si <code>element</code> ya existe en el conjunto. Devuelve si existe.
BOOLEAN <code>.delete(element)</code>	Elimina el <code>element</code> del conjunto. Devuelve si lo eliminó correctamente.
<code>.clear()</code>	Vacia el conjunto completo.

3.11 MAPAS

```
const map = new Map();           // Map({})
const map = new Map([[1, "uno"]]); // Map({ 1=>"uno" })
const map = new Map([[1, "uno"], [2, "dos"], [3, "tres"]]); // Map({ 1=>"uno", 2=>"dos", 3=>"tres" })

map.constructor.name;           // "Map"
```

Propiedad o Método	Descripción
NUMBER <code>.size</code>	Propiedad que devuelve el número de elementos que tiene el mapa.
MAP <code>.set(key, value)</code>	Establece o modifica la clave <code>key</code> con el valor <code>value</code> . Muta
BOOLEAN <code>.has(key)</code>	Comprueba si <code>key</code> ya existe en el mapa y devuelve si existe o no.
OBJECT <code>.get(key)</code>	Obtiene el valor de la clave <code>key</code> del mapa.
BOOLEAN <code>.delete(key)</code>	Elimina el elemento con la clave <code>key</code> del mapa. Devuelve si lo eliminó correctamente.
<code>.clear()</code>	Vacia el mapa completamente.

3.12 JSON

```
{
  "name": "Manz",
  "life": 3,
  "totallife": 6
  "power": 10,
  "dead": false,
  "props": ["invisibility", "coding", "happymood"],
  "senses": {
    "vision": 50,
    "audition": 75,
    "taste": 40,
    "touch": 80
  }
}
```

- Las propiedades del objeto deben estar entrecomilladas con «comillas dobles»
- Los textos `STRING` deben estar entrecomillados con «comillas dobles»
- Sólo se puede almacenar tipos como `STRING`, `NUMBER`, `OBJECT`, `ARRAY`, `BOOLEAN` o `null`.
- Tipos de datos como `FUNCTION`, `DATE`, `REGEXP` u otros, no es posible almacenarlos en un JSON.
- Tampoco es posible añadir **comentarios** en un JSON.

Método	Descripción
<code>OBJECT JSON.parse(str)</code>	Convierte el texto <code>STRING str</code> (si es un JSON válido) a un objeto y lo devuelve.
<code>STRING JSON.stringify(obj)</code>	Convierte un objeto Javascript <code>OBJECT obj</code> a su representación JSON y la devuelve.
<code>STRING JSON.stringify(obj, props)</code>	Idem al anterior, pero filtra y mantiene solo las propiedades del <code>ARRAY props</code> .
<code>STRING JSON.stringify(obj, props, spaces)</code>	Idem al anterior, pero indenta el JSON a <code>NUMBER spaces</code> espacios.

3.13 FUNCIONES

3.13.1 Declaración

Constructor	Descripción
<code>FUNCTION function nombre(p1, p2...) { }</code>	Crea una función mediante declaración .
<code>FUNCTION var nombre = function(p1, p2...) { }</code>	Crea una función mediante expresión .

```
function saludar() {
  return "Hola";
}

saludar(); // 'Hola'
typeof saludar; // 'function'
```

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

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

3.13.2 Callbacks

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

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

fA(fB);
```

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

// fError = Función Error (callback)
const fError = function () {
  console.error("Error");
};

// fA = Función A
const fA = function (callback, callbackError) {
  const n = ~~(Math.random() * 5);
  if (n > 2) callback();
  else callbackError();
};

fA(fB, fError); // Si ejecutamos varias veces, algunas darán error y otras no
```

```
// fA = Función A
const fA = function (callback, callbackError) {
  const n = ~~(Math.random() * 5);
  if (n > 2) callback();
  else callbackError();
};

fA(
  function () {
    console.log("Función B ejecutada.");
  },
  function () {
    console.error("Error");
  }
);
```

3.13.3 Autoejecutables

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

// Función autoejecutable con parámetros
(function (name) {
  console.log(`Hola, ${name}`);
})(“Manz”);
```

3.13.4 Clausuras

```
// Clausura: Función incr()
const incr = (function () {
  let num = 0;
  return function () {
    num++;
    return num;
  };
})();

typeof incr; // 'function'
incr(); // 1
incr(); // 2
incr(); // 3
```

3.13.5 Fechas

```
const func = function () {
    return "Función tradicional.";
};

const func = () => {
    return "Función flecha.";
};
```



- Si el cuerpo de la función sólo tiene una línea, podemos omitir las llaves (`{}`).
- Además, en ese caso, automáticamente se hace un `return` de esa única línea, por lo que podemos omitir también el `return`.
- En el caso de que la función no tenga parámetros, se indica como en el ejemplo anterior: `() =>`.
- En el caso de que la función tenga un solo parámetro, se puede indicar simplemente el nombre del mismo: `e =>`.
- En el caso de que la función tenga 2 ó más parámetros, se indican entre paréntesis: `(a, b) =>`.
- Si queremos devolver un objeto, que coincide con la sintaxis de las llaves, se puede englobar con paréntesis: `({name: 'Manz'})`.

```
const func = () => "Función flecha."; // 0 parámetros: Devuelve "Función flecha"
const func = (e) => e + 1; // 1 parámetro: Devuelve el valor de e + 1
const func = (a, b) => a + b; // 2 parámetros: Devuelve el valor de a + b
```



3.14 VALOR VS REFERENCIA

3.15 CONDICIONALES

Estructura de control	Descripción
<code>If</code>	Condición simple: Si ocurre algo, haz lo siguiente...
<code>If/else</code>	Condición con alternativa: Si ocurre algo, haz esto, sino, haz lo esto otro...
<code>:</code>	Operador ternario: Equivalente a <code>If/else</code> , método abreviado.
<code>Switch</code>	Estructura para casos específicos: Similar a varios <code>If/else</code> anidados.

3.16 BUCLES

Tipo de bucle	Descripción
<code>while</code>	Bucles simples.
<code>for</code>	Bucles clásicos por excelencia.
<code>do..while</code>	Bucles simples que se realizan siempre como mínimo una vez.
<code>for..in</code>	Bucles sobre posiciones de un array. Los veremos más adelante.
<code>for..of</code>	Bucles sobre elementos de un array. Los veremos más adelante.

3.17 CLASES

```
// Declaración de una clase (de momento, vacía)
class Animal {}

// Crear (instanciar) un objeto basada en una clase
const pato = new Animal();
```



Más adelante utilizaremos mucho la palabra clave **this**. Esta es una palabra clave que se utiliza mucho dentro de las clases para hacer referencia al objeto instanciado. Ojo, que hace referencia al **objeto instanciado** y no a la clase:

```
class Animal {
    name;           // Propiedad (variable de clase sin valor definido)

    constructor(name) {
        this.name = name; // Hacemos referencia a la propiedad name del objeto instanciado
    }
}
```

Elemento	Descripción	Más información
Propiedad	Variable que existe dentro de una clase. Puede ser pública o privada.	Ver propiedades
Propiedad pública	Propiedad a la que se puede acceder desde fuera de la clase.	
Propiedad privada <small>ES2020</small>	Propiedad a la que no se puede acceder desde fuera de la clase.	
Propiedad computada	Función para acceder a una propiedad con modificaciones (getter/setter).	
Método	Función que existe dentro de una clase. Puede ser pública o privada.	Ver métodos
Método público	Método que se puede ejecutar desde dentro y fuera de la clase.	
Método privado <small>ES2020</small>	Método que sólo se puede ejecutar desde dentro de la clase.	
Constructor	Método especial que se ejecuta automáticamente cuando se crea una instancia.	
Método estático	Método que se ejecuta directamente desde la clase, no desde la instancia.	
Inicializador estático <small>ES2022</small>	Bloque de código que se ejecuta al definir la clase, sin necesidad de instancia.	

3.17.1 Propiedades

Nombre	Sintaxis	Descripción
Propiedad pública	<code>name</code> o <code>this.name</code>	Se puede acceder a la propiedad desde dentro y fuera de la clase.
Propiedad privada	<code>#name</code> o <code>this.#name</code>	Se puede acceder a la propiedad sólo desde dentro de la clase.

```
class Personaje {
    name;
    energy = 10;

    constructor(name) {
        this.name = name;
    }
}
```

```
class Personaje {
    #name;
    energy = 10;

    constructor(name) {
        this.#name = name;
    }
}
```

3.17.2 Métodos

Nombre	Sintaxis	Descripción
Método público	<code>name()</code> o <code>this.name()</code>	Se puede acceder al método desde dentro y fuera de la clase.
Método privado	<code>#name()</code> o <code>this.#name()</code>	Se puede acceder al método sólo desde dentro de la clase.

```

class Personaje {
  name = "Mario";

  constructor() {
    this.#hablar();
  }

  #hablar() {
    console.log("It's me, Mario!");
  }
}

const mario = new Personaje();      // It's me, Mario! (se ha accedido a #hablar() desde dentro de la clase)

// Da error, no se permite acceder a un método privado desde fuera de la clase
// Uncaught SyntaxError: Private field '#hablar' must be declared in an enclosing class
mario.#hablar();

// Da error, el método hablar() no existe, ya que el nombre del método es #hablar()
// Uncaught TypeError: mario.hablar is not a function
mario.hablar();

```

3.17.3 Gets y Sets

```

class Personaje {
  name;
  energy;

  constructor(name, energy = 10) {
    this.name = name;
    this.energy = energy;
  }

  get status() {
    return '★'.repeat(this.energy);
  }

  set status(stars) {
    this.energy = stars.length;
  }
}

const mario = new Personaje("Mario");
mario.energy // 10
mario.status = '★★★★'
mario.energy // 3
mario.status // '★★★'

```

3.17.4 Constructores

```

class Animal {
  constructor() {
    console.log("Ha nacido un pato. 🐥");
  }

  hablar() {
    return "Cuak";
  }
}

// Creación de instancia/objeto
const pato = new Animal(); // 'Ha nacido un pato' (Se ha ejecutado implicitamente el constructor)
pato.hablar();           // 'Cuak' (Se ha ejecutado explicitamente el método hablar)

```

3.17.5 Estáticos

```
class Animal {
  static despedirse() {
    return "Adiós";
  }

  hablar() {
    return "Cuak";
  }
}

Animal.despedirse();      // Método estático (no requiere instancia): 'Adiós'
Animal.hablar();          // Uncaught TypeError: Animal.hablar is not a function

const pato = new Animal(); // Creamos una instancia

pato.despedirse();        // Uncaught TypeError: pato.despedirse is not a function
pato.hablar();            // Método no estático (requiere instancia): 'Cuak'
```

```
class Animal {
  static {
    console.log("Bloque inicializado");
  }

  constructor() {
    console.log("Constructor ejecutado");
  }
} // <-- Aquí nos aparece "Bloque inicializado"

const pato = new Animal(); // <-- Tras el new Animal(), aparece "Constructor ejecutado"
```

3.17.6 Herencia

```
// Clase padre
class Forma {
  constructor() {
    console.log("Soy una forma geométrica.");
  }
}

// Clase hija
class Cuadrado extends Forma {
  constructor() {
    super();
    console.log("Soy un cuadrado.");
  }
}
```

```
class Padre {
  soloPadre() { console.log("Tarea en el padre..."); }
  padreHijo() { console.log("Tarea en el padre..."); }
  sobreHijo() { console.log("Tarea en el padre..."); }
}

class Hijo extends Padre {
  padreHijo() {
    super.padreHijo();
    console.log("Tarea en el hijo...");
  }

  soloHijo() { console.log("Tarea en el hijo..."); }
  sobreHijo() { console.log("Tarea en el hijo..."); }
}
```

Método	Clase Padre	Clase Hija	¿Se ejecuta el método en una instancia de la clase hija?
<code>soloPadre()</code>	✓	✗	Se ejecuta porque se hereda el método del padre hacia el hijo.
<code>soloHijo()</code>	✗	✓	Se ejecuta porque simplemente existe en el hijo.
<code>padreHijo()</code>	✓	✓	Se ejecutan ambos porque super llama al padre primero.
<code>sobreHijo()</code>	✓	✓	Se ejecuta sólo el hijo, porque sobrescribe el heredado del padre.

3.18 MÓDULOS

Declaración	Descripción
<code>export</code>	Pone los datos indicados (variables, funciones, clases...) a disposición de otros ficheros
<code>import</code>	Incorpora datos (variables, funciones, clases...) desde otros ficheros <code>.js</code> al código actual.
<code>import()</code>	Permite importar módulos de forma más flexible, en tiempo real (imports dinámicos).

```
// Fichero constants.js
export const magicNumber = 42;
```

```
// Fichero index.js
import { magicNumber } from "./constants.js";

console.log(magicNumber); // 42
```

3.18.1 Exportación

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, n2, 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 por defecto .

```
export let number = 42;           // Se añade la variable number al módulo
export const hello = () => "Hello!"; // Se añade la función hello al módulo
export class CodeBlock { };       // Se añade la clase vacía CodeBlock al módulo
```

```
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
```

```

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

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

```

```

// CASO 1: Exporta todo lo exportado en el fichero math.js (abs, min, max, random)
export * from "./math.js";

```

```

// CASO 2: Exporta sólo abs, min y max del fichero math.js
export { abs, min, max } from "./math.js";

```

```

// CASO 3: Exporta todo lo exportado en el fichero math.js en un objeto con nombre
export const number = 42;
export * as math from "./math.js";

```

3.18.2 Importación

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

- `./math.js`: El fichero `math.js` en la carpeta actual.
- `../math.js`: El fichero `math.js` en la carpeta padre.
- `/math.js`: El fichero `math.js` en la raíz del proyecto.
- `"https://web.com/math.js"`: El fichero `math.js` está alojado en una web.

nombre

```

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

```

masiva

```

import * as module from "./file.js";

```

codigo

```

import "./math.js";

```

estáticos vs dinámicos

Los **import estáticos** son muy útiles, pero tienen algunas desventajas si se presentan ciertos casos específicos. Los más frecuentes suelen ser los siguientes:

- Queremos importar un módulo si se cumple una determinada **condición**
- Queremos importar un módulo **interpolando** variables o constantes
- Queremos importar un módulo **dentro de un ámbito** específico
- Queremos importar un módulo desde un **script normal** (*sin type="module"*)
- Queremos importar un fichero javascript (*sin módulo*) y ejecutarlo bajo demanda

En cada uno de estos casos, no se puede utilizar el **import estático**, pero si el **import dinámico**:

```
// Opción 1: Se carga functions.js si se cumple la condición
if (number > 42) {
  import("./functions.js")
    .then(module => module.func());
}

// Opción 2: Se carga functions.js interpolando la constante
const filename = "functions";
import(`./${filename}.js`)
  .then(module => module.func());

// Opción 3: Se carga additional.js sólo si el usuario hace click en el botón
const button = document.querySelector("button.info");
button.addEventListener("click", () => import("additional.js"), { once: true });
```

4 DOM

El objeto document

En Javascript, la forma de acceder al DOM es a través de un objeto llamado **document**, que representa el árbol DOM de la página o pestaña del navegador donde nos encontramos. En su interior pueden existir varios tipos de elementos, pero principalmente serán **ELEMENT** o **NODE**:

- **ELEMENT** no es más que la representación genérica de una etiqueta: **HTMLElement**.
- **NODE** es una unidad más básica, la cuál puede ser **ELEMENT** o un **nodo de texto**.

4.1 SELECCIONAR ELEMENTOS

4.1.1 Tradicionales

Métodos de búsqueda	Descripción
ELEMENT <code>.getElementById(id)</code>	Busca el elemento HTML con el id id . Si no, devuelve NULL .
ARRAY <code>.getElementsByClassName(class)</code>	Busca elementos con la clase class . Si no, devuelve [] .
ARRAY <code>.getElementsByTagName(name)</code>	Busca elementos con atributo name name . Si no, devuelve [] .
ARRAY <code>.getElementsByTagName(tag)</code>	Busca elementos tag . Si no encuentra ninguno, devuelve [] .

4.1.2 Modernos

Método de búsqueda	Descripción
ELEMENT <code>.querySelector(sel)</code>	Busca el primer elemento que coincide con el selector CSS sel . Si no, NULL .
ARRAY <code>.querySelectorAll(sel)</code>	Busca todos los elementos que coinciden con el selector CSS sel . Si no, [] .

Al realizar una búsqueda de elementos y guardarlos en una variable, podemos realizar la búsqueda posteriormente sobre esa variable en lugar de hacerla sobre **document**. Esto permite realizar búsquedas acotadas por zonas, en lugar de realizarlo siempre sobre **document**, que buscará en todo el documento HTML.



4.2 CREAR ELEMENTOS

Métodos	Descripción
<code>ELEMENT .createElement(tag, options)</code>	Crea y devuelve el elemento HTML definido por el <code>STRING tag</code> .
<code>NODE .createComment(text)</code>	Crea y devuelve un nodo de comentarios HTML <code><!-- text --></code> .
<code>NODE .createTextNode(text)</code>	Crea y devuelve un nodo HTML con el texto <code>text</code> .
<code>NODE .cloneNode(deep)</code>	Clona el nodo HTML y devuelve una copia. <code>deep</code> es <code>false</code> por defecto.
<code>BOOLEAN .isConnected</code>	Indica si el nodo HTML está insertado en el documento HTML.

4.2.1 Atributos

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

Métodos	Descripción
<code>BOOLEAN hasAttributes()</code>	Indica si el elemento tiene atributos HTML.
<code>BOOLEAN hasAttribute(attr)</code>	Indica si el elemento tiene el atributo HTML <code>attr</code> .
<code>ARRAY getAttributeNames()</code>	Devuelve un <code>ARRAY</code> con los atributos del elemento.
<code>STRING getAttribute(attr)</code>	Devuelve el valor del atributo <code>attr</code> del elemento o <code>NULL</code> si no existe.
<code>UNDEFINED removeAttribute(attr)</code>	Elimina el atributo <code>attr</code> del elemento.
<code>UNDEFINED setAttribute(attr, value)</code>	Añade o cambia el atributo <code>attr</code> al valor <code>value</code> .
<code>NODE getAttributeNode(attr)</code>	Idem a <code>getAttribute()</code> pero devuelve el atributo como <code>nodo</code> .
<code>NODE removeAttributeNode(attr)</code>	Idem a <code>removeAttribute()</code> pero devuelve el atributo como <code>nodo</code> .
<code>NODE setAttributeNode(attr, value)</code>	Idem a <code>setAttribute()</code> pero devuelve el atributo como <code>nodo</code> .

4.2.2 Fragmentos

Métodos	Descripción
<code>OBJECT document.createDocumentFragment()</code>	Crea un fragmento aislado (<i>sin padre</i>).

- No tiene elemento padre. Está aislado de la página o documento.
- Es mucho más simple y ligero (*mejor rendimiento*).
- Si necesitamos hacer cambios consecutivos, no afecta al `reflow` (*re pintado de un documento*).

4.3 INCORPORAR ELEMENTOS

4.3.1 Reemplazar

Propiedades	Descripción
<code>STRING .nodeName</code>	Devuelve el nombre del nodo (etiqueta si es un elemento HTML). Sólo lectura.
<code>STRING .textContent</code>	Devuelve el contenido de texto del elemento. Se puede asignar para modificar.
<code>STRING .innerHTML</code>	Devuelve el contenido HTML del elemento. Se puede usar asignar para modificar.
<code>STRING .outerHTML</code>	Idem a <code>.innerHTML</code> pero incluyendo el HTML del propio elemento HTML.

```
const data = document.querySelector(".data");
data.innerHTML = "<h1>Tema 1</h1>";

data.textContent; // "Tema 1"
data.innerHTML; // "<h1>Tema 1</h1>"
data.outerHTML; // "<div class='data'><h1>Tema 1</h1></div>"
```

4.3.2 Insertar

Métodos	Descripción
<code>NODE .appendChild(node)</code>	Añade como hijo el nodo <code>node</code> . Devuelve el nodo insertado.
<code>ELEMENT .insertAdjacentElement(pos, elem)</code>	Inserta el elemento <code>elem</code> en la posición <code>pos</code> . Si falla, <code>NULL</code> .
<code>UNDEFINED .insertAdjacentHTML(pos, str)</code>	Inserta el código HTML <code>str</code> en la posición <code>pos</code> .
<code>UNDEFINED .insertAdjacentText(pos, text)</code>	Inserta el texto <code>text</code> en la posición <code>pos</code> .
<code>NODE .insertBefore(new, node)</code>	Inserta el nodo <code>new</code> antes de <code>node</code> y como hijo del nodo actual.

```
const div = document.createElement("div"); // <div></div>
div.textContent = "Ejemplo"; // <div>Ejemplo</div>

const app = document.querySelector("#app"); // <div id="app">App</div>

app.insertAdjacentElement("beforebegin", div);
// Opción 1: <div>Ejemplo</div> <div id="app">App</div>

app.insertAdjacentElement("afterbegin", div);
// Opción 2: <div id="app"> <div>Ejemplo</div> App</div>

app.insertAdjacentElement("beforeend", div);
// Opción 3: <div id="app">App <div>Ejemplo</div> </div>

app.insertAdjacentElement("afterend", div);
// Opción 4: <div id="app">App</div> <div>Ejemplo</div>
```

```
app.insertAdjacentElement("beforebegin", div);
// Opción 1: <div>Ejemplo</div> <div id="app">App</div>

app.insertAdjacentHTML("beforebegin", '<p>Hola</p>');
// Opción 2: <p>Hola</p> <div id="app">App</div>

app.insertAdjacentText("beforebegin", "Hola a todos");
// Opción 3: Hola a todos <div id="app">App</div>
```

4.3.3 Eliminar

Métodos	Descripción
<code>UNDEFINED .remove()</code>	Elimina el propio nodo de su elemento padre.
<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> .

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

div.isConnected; // true
div.remove();
div.isConnected; // false
```

4.4 CSS

Propiedad	Descripción
<code>STRING .className</code>	Acceso directo al valor del atributo HTML <code>class</code> . También se puede asignar.
<code>OBJECT .classList</code>	Objeto especial para manejar clases CSS. Contiene métodos y propiedades de ayuda.

La propiedad `.className` viene a ser la modalidad directa y rápida de utilizar el getter `.getAttribute("class")` y el setter `.setAttribute("class", v)`. Veamos un ejemplo utilizando estas propiedades y métodos y su equivalencia:

```
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 brillo tema-oscuro";
div.setAttribute("class", "element brillo tema-oscuro");
```

Trabajar con `.className` tiene una limitación cuando trabajamos con **múltiples clases CSS**, y es que puedes querer realizar una manipulación sólo en una clase CSS concreta, dejando las demás intactas. En ese caso, modificar clases CSS mediante una asignación `.className` se vuelve poco práctico.

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

4.5 NAVEGACIÓN

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

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

```

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

```

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

```

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

```

5 EVENTOS

5.1 MANEJAR EVENTOS

Forma	Ejemplo
Mediante atributos HTML	<code><button onClick="...></button></code>
Mediante propiedades Javascript	<code>.onClick = function() { ... }</code>
Mediante addEventListener()	<code>.addEventListener("click", ...)</code>

Método	Descripción
<code>.removeEventListener(STRING event, FUNCTION func)</code>	Elimina la funcionalidad <code>func</code> asociada al evento <code>event</code> .

Manejo eventos con funciones.

```

class EventManager {
  constructor(element) {
    element.addEventListener("click", () => this.sendMessage());
  }

  sendMessage() {
    alert("Has hecho click en el botón");
    console.log(this); // this = referencia a EventManager
  }
}

const button = document.querySelector("button");
const eventManager = new EventManager(button);

```

Manejo eventos con objetos.

```

const button = document.querySelector("button");
const eventManager = {
  handleEvent: function(ev) {
    if (ev.type === "click") {
      alert("¡Has hecho click!");
    } else if (ev.type === "mouseleave") {
      alert("¡Has abandonado el botón!");
    }
  }
}
button.addEventListener("click", eventManager);
button.addEventListener("mouseleave", eventManager);

```

Manejo eventos con clases.

```

<button>Click me!</button>

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

class EventManager {
  handleEvent(ev) {
    if (ev.type === "click") {
      this.onClick(ev.type, ev.target);
    } else if (ev.type === "mouseleave") {
      this.onLeave(ev.type, ev.target);
    }
  }

  onClick(type, element) {
    alert("¡Has hecho click!");
    console.log({ type, element });
  }

  onLeave(type, element) {
    alert("¡Has abandonado el botón!");
    console.log({ type, element });
  }
}

const eventManager = new EventManager();
button.addEventListener("click", eventManager);
button.addEventListener("mouseleave", eventManager);
</script>

```

5.2 EVENTOS NATIVOS

```

const button = document.querySelector("button");
button.addEventListener("click", (event) => {
  const { type, timeStamp, isTrusted } = event;
  console.log({ type, timeStamp, isTrusted });
});

```

Propiedad	Descripción	
STRING <code>.type</code>	Indica el tipo de evento en cuestión.	
NUMBER <code>.timeStamp</code>	Hora en milisegundos en la que se creó el evento.	
BOOLEAN <code>.isTrusted</code>	Indica si es un evento real de un usuario o uno enviado manualmente con <code>.dispatchEvent()</code> .	

Propiedad o Método	Valor por defecto	Descripción
Propiedad		
BOOLEAN <code>.defaultPrevented</code>	<code>false</code>	Indica si el comportamiento por defecto se ha evitado.
Métodos		
<code>.preventDefault()</code>	Evita que se realice el comportamiento por defecto del evento.	

5.3 EVENTOS PERSONALIZADOS

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

En lugar de `CustomEvent` también se puede indicar simplemente `Event` (*o alguno de sus objetos derivados*). La diferencia radica en que `CustomEvent` se suele utilizar cuando queremos añadir **datos personalizados**, como vamos a hacer a continuación en las opciones.

Nombre

- Los **eventos** son *case sensitive*, por lo que es preferible usar todo en minúsculas.
- Evita **camelCase**, que suele inducir a dudas. Si has elegido todo en minúsculas, puedes optar a usar **kebab-case**.
- Usa namespaces y elegir un separador: Por ejemplo, `user:data-message` o `user .data-message`.

Opciones

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

Ejemplo

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

5.4 EVENTOS NAVEGADOR

Event	Occurs When	Belongs To
<code>abort</code>	The loading of a media is aborted	UiEvent , Event
<code>afterprint</code>	A page has started printing	Event
<code>animationend</code>	A CSS animation has completed	AnimationEvent
<code>animationiteration</code>	A CSS animation is repeated	AnimationEvent
<code>animationstart</code>	A CSS animation has started	AnimationEvent
<code>beforeprint</code>	A page is about to be printed	Event
<code>beforeunload</code>	Before a document is about to be unloaded	UiEvent , Event
<code>blur</code>	An element loses focus	FocusEvent
<code>canplay</code>	The browser can start playing a media (has buffered enough to begin)	Event
<code>canplaythrough</code>	The browser can play through a media without stopping for buffering	Event
<code>change</code>	The content of a form element has changed	Event
<code>click</code>	An element is clicked on	MouseEvent
<code>contextmenu</code>	An element is right-clicked to open a context menu	MouseEvent
<code>copy</code>	The content of an element is copied	ClipboardEvent
<code>cut</code>	The content of an element is cutted	ClipboardEvent
<code>dblclick</code>	An element is double-clicked	MouseEvent
<code>drag</code>	An element is being dragged	DragEvent
<code>dragend</code>	Dragging of an element has ended	DragEvent
<code>dragenter</code>	A dragged element enters the drop target	DragEvent
<code>dragleave</code>	A dragged element leaves the drop target	DragEvent
<code>dragover</code>	A dragged element is over the drop target	DragEvent
<code>dragstart</code>	Dragging of an element has started	DragEvent
<code>drop</code>	A dragged element is dropped on the target	DragEvent
<code>durationchange</code>	The duration of a media is changed	Event
<code>ended</code>	A media has reached the end ("thanks for listening")	Event
<code>error</code>	An error has occurred while loading a file	ProgressEvent , UiEvent , Event
<code>focus</code>	An element gets focus	FocusEvent
<code>focusin</code>	An element is about to get focus	FocusEvent
<code>focusout</code>	An element is about to lose focus	FocusEvent
<code>fullscreenchange</code>	An element is displayed in fullscreen mode	Event
<code>fullscreenerror</code>	An element can not be displayed in fullscreen mode	Event
<code>hashchange</code>	There have been changes to the anchor part of a URL	HashChangeEvent
<code>input</code>	An element gets user input	InputEvent , Event
<code>invalid</code>	An element is invalid	Event
<code>keydown</code>	A key is down	KeyboardEvent
<code>keypress</code>	A key is pressed	KeyboardEvent
<code>keyup</code>	A key is released	KeyboardEvent
<code>load</code>	An object has loaded	UiEvent , Event
<code>loadeddata</code>	Media data is loaded	Event
<code>loadedmetadata</code>	Meta data (like dimensions and duration) are loaded	Event
<code>loadstart</code>	The browser starts looking for the specified media	ProgressEvent
<code>message</code>	A message is received through the event source	Event

Event	Occurs When	Belongs To
<code>mousedown</code>	The mouse button is pressed over an element	MouseEvent
<code>mouseenter</code>	The pointer is moved onto an element	MouseEvent
<code>mouseleave</code>	The pointer is moved out of an element	MouseEvent
<code>mousemove</code>	The pointer is moved over an element	MouseEvent
<code>mouseover</code>	The pointer is moved onto an element	MouseEvent
<code>mouseout</code>	The pointer is moved out of an element	MouseEvent
<code>mouseup</code>	A user releases a mouse button over an element	MouseEvent
<code>mousewheel</code>	Deprecated. Use the <code>wheel</code> event instead	WheelEvent
<code>offline</code>	The browser starts working offline	Event
<code>online</code>	The browser starts working online	Event
<code>open</code>	A connection with the event source is opened	Event
<code>pagehide</code>	User navigates away from a webpage	PageTransitionEvent
<code>pageshow</code>	User navigates to a webpage	PageTransitionEvent
<code>paste</code>	Some content is pasted in an element	ClipboardEvent
<code>pause</code>	A media is paused	Event
<code>play</code>	The media has started or is no longer paused	Event
<code>playing</code>	The media is playing after being paused or buffered	Event
<code>popstate</code>	The window's history changes	PopStateEvent
<code>progress</code>	The browser is downloading media data	Event
<code>ratechange</code>	The playing speed of a media is changed	Event
<code>resize</code>	The document view is resized	UIEvent , Event
<code>reset</code>	A form is reset	Event
<code>scroll</code>	An scrollbar is being scrolled	UIEvent , Event
<code>search</code>	Something is written in a search field	Event
<code>seeked</code>	Skipping to a media position is finished	Event
<code>seeking</code>	Skipping to a media position is started	Event
<code>select</code>	User selects some text	UIEvent , Event
<code>show</code>	A <code><menu></code> element is shown as a context menu	Event
<code>stalled</code>	The browser is trying to get unavailable media data	Event
<code>storage</code>	A Web Storage area is updated	StorageEvent
<code>submit</code>	A form is submitted	Event
<code>suspend</code>	The browser is intentionally not getting media data	Event
<code>timeupdate</code>	The playing position has changed (the user moves to a different point in the media)	Event
<code>toggle</code>	The user opens or closes the <code><details></code> element	Event
<code>touchcancel</code>	The touch is interrupted	TouchEvent
<code>touchend</code>	A finger is removed from a touch screen	TouchEvent
<code>touchmove</code>	A finger is dragged across the screen	TouchEvent
<code>touchstart</code>	A finger is placed on a touch screen	TouchEvent
<code>transitionend</code>	A CSS transition has completed	TransitionEvent
<code>unload</code>	A page has unloaded	UIEvent , Event
<code>volumechange</code>	The volume of a media is changed (includes muting)	Event
<code>waiting</code>	A media is paused but is expected to resume (e.g. buffering)	Event
<code>wheel</code>	The mouse wheel rolls up or down over an element	WheelEvent

5.5 EMITIR EVENTOS

```
const event = new CustomEvent("user:message", {  
    detail: {  
        from: "Manz",  
        message: "Hello!"  
    }  
});  
  
const button = document.querySelector("button");  
button.addEventListener("click", () => {  
    button.dispatchEvent(event);  
});
```

```
const root = document.querySelector(".root");  
const button = document.querySelector("button");  
  
button.addEventListener("click", () => {  
    button.dispatchEvent(new CustomEvent("user:message", {  
        detail: {  
            name: "Manz"  
        },  
        bubbles: true,  
    }));  
});  
  
root.addEventListener("user:message", (event) => {  
    const name = event.detail.name;  
    const number = event.target.dataset.number;  
    console.log(`Message received from ${name} (${number})`);  
});
```

```
root.addEventListener("user:message", (event) => {  
    const name = event.detail.name;  
    const number = event.target.dataset.num;  
    console.log(`Message received from ${name} (${number})`);  
}, { capture: true });
```

5.6 PROPAGACIÓN

Propiedad o Método	Descripción
Propiedades	
<code>BOOLEAN .bubbles</code>	Indica si el evento se propagará hacia contenedores padres o se detendrá en el elemento emitido.
<code>BOOLEAN .composed</code>	Indica si el evento puede atravesar un Shadow DOM en su propagación, o no.
Destino del evento	
<code>ELEMENT .target</code>	Indica el elemento objetivo (donde se hizo el <code>dispatchEvent()</code>).
<code>ELEMENT .currentTarget</code>	Indica el elemento actual donde se ha escuchado el evento.
Método	
<code>ARRAY .composedPath()</code>	Muestra el camino de elementos por donde se ha propagado el evento.

- Si el flag **bubbles** está desactivado, el evento se emite a `<button>` y se detiene ahí.
- Si el flag **bubbles** está activado, el evento se emite a `<button>`, luego a su contenedor padre, y así sucesivamente.
- Si el flag **composed** está desactivado, el evento se detendrá al encontrar un [Shadow DOM](#).
- Si el flag **composed** está activado, el evento no se detendrá si encuentra un [Shadow DOM](#).

Propiedad o Método	Valor por defecto	Descripción
Propiedad		
<code>BOOLEAN .cancelable</code>	<code>true</code>	Indica si es posible cancelar el evento.
Métodos		
<code>.stopPropagation()</code>		Detiene la propagación en el evento en cuestión.
<code>.stopImmediatePropagation()</code>		Detiene la propagación en todos los eventos del mismo tipo.

6 EXPRESIONES REGULARES

6.1 DEFINICIÓN

Constructor	Descripción
<code>REGEXP new RegExp(regex, flags)</code>	Crea una nueva expresión regular a partir de <code>regexp</code> con los <code>flags</code> indicados.
<code>REGEXP /regexp flags</code>	Simplemente, la expresión regular <code>regexp</code> entre barras / Notación preferida .

```
// Notación literal (forma preferida)
const regexp = /.a.o/i;

// Notación de objeto
const regexp = new RegExp(".a.o", "i");
const regexp = new RegExp(/.a.o/, "i");
```



En Javascript, se prefiere utilizar las barras / para delimitar una expresión regular en una variable. Se trata de una forma más corta y compacta que evita tener que escribir el `new` del objeto [REGEXP](#).



6.2 PROPIEDADES Y BANDERAS

Propiedades	Descripción
<code>STRING .source</code>	Devuelve la expresión regular original definida (<i>sin los flags</i>).
<code>STRING .flags</code>	Devuelve los <code>flags</code> activados en la expresión regular.

Propiedades	Flag	Descripción
<code>BOOLEAN .global</code>	<code>g</code>	Búsqueda global. Permite seguir buscando coincidencias en lugar de pararse al encontrar una.
<code>BOOLEAN .ignoreCase</code>	<code>i</code>	Le da igual mayúsculas y minúsculas. Se suele denominar insensible a mayúsculas/minúsculas .
<code>BOOLEAN .multiline</code>	<code>m</code>	Multilínea. Permite a ^ y \$ tratar los finales de línea \r o \n.
<code>BOOLEAN .unicode</code>	<code>u</code>	Unicode. Interpreta el patrón como un código de una secuencia Unicode.
<code>BOOLEAN .sticky</code>	<code>y</code>	Sticky. Busca sólo desde la posición indicada por <code>lastIndex</code> .
<code>BOOLEAN .dotAll</code>	<code>s</code>	Establece si \n, \r, separación de párrafo o separación de línea deberían considerarse en los .
<code>BOOLEAN .hasIndices</code>	<code>d</code>	Establece si al ejecutar un <code>.exec()</code> el resultado deberá tener propiedad <code>.indices</code> .

6.3 EJECUCIÓN

Métodos

Método	Descripción
<code>BOOLEAN test(text)</code>	Comprueba si la expresión regular «casa» con el texto <code>text</code> pasado por parámetro.
<code>ARRAY exec(text)</code>	Ejecuta una búsqueda en el texto <code>text</code> . Devuelve <code>ARRAY</code> con capturas de lo que coincide.

Patrones

Formato	Descripción
<code>(x)</code>	El patrón <code>x</code> incluido dentro de paréntesis se captura y se guarda.
<code>(?:x)</code>	Si incluimos <code>?:</code> al inicio del patrón en los paréntesis, no se captura ese patrón.
<code>x(?:y)</code>	Busca sólo si <code>x</code> está seguido de <code>y</code> .
<code>x(?:!y)</code>	Busca sólo si <code>x</code> no está seguido de <code>y</code> .

Resultado

Propiedad	Descripción
<code>.length</code>	Como array, se puede consultar la longitud (coincidencia completa + capturas)
<code>.groups</code>	Crea un objeto con los resultados de parentezas nombradas (ver más adelante)
<code>.index</code>	Posición del <code>STRING</code> donde se encontró la ocurrencia.
<code>.input</code>	Texto <code>STRING</code> original pasado por parámetro a <code>.exec()</code> .
<code>.indices</code>	Si se usa el flag <code>d</code> , se incluye un <code>ARRAY</code> con las posiciones inicial y final de las coincidencias del <code>ARRAY</code> .

Por Posición

```
const text = "Hola Manz. El formato adecuado es 2022-08-15. Ignoraremos fechas en el formato 15-08-2022.";
const regexp = /([0-9]{4})-([0-9]{2})-([0-9]{2})/gd;

regexp.global; // true (el flag global está activado)

const result = regexp.exec(text); // ["2022-08-15", "2022", "08", "15"] index: 34
regexp.exec(text); // null
```

```
result // ["2022-08-15", "2022", "08", "15"]
result.length // 4
result.index // 34
result.input == text // true

regexp.hasIndices // true
result.indices // [[34, 44], [34, 38], [39, 41], [42, 44]]
```

Por Nombre

```
const text = `Hola Manz. Son las 13:33:02, a las 18:45:00 te avisaré para que inicies stream.`;
const regexp = /(?<hours>[0-9]{2}):(?<mins>[0-9]{2}):(?<secs>[0-9]{2})/gd

const result = regexp.exec(text); // ["13:33:02", "13", "33", "02"]
```

```
result.groups // { hours: "13", mins: "33", secs: "02" }
result.index // 19
result.indices // [[19, 27], [19, 21], [22, 24], [25, 27]]
result.indices.groups // { hours: [19, 21], mins: [22, 24], secs: [25, 27] }
```

6.4 SÍMBOLOS Generales

Carácter especial	Descripción
.	Comodín, significa cualquier carácter (letra, número, símbolo...), pero que ocupe sólo 1 carácter .
\	Precedido de un carácter especial, lo invalida (se llama «escapar»).

Rangos

Carácter especial	Descripción
[]	Rango de caracteres. Cualquiera de los caracteres del interior de los corchetes.
[^]	Que no exista cualquiera de los caracteres del interior de los corchetes.
	Establece una alternativa: lo que está a la izquierda o lo que está a la derecha.

Conjunto

Carácter especial	Alternativa	Descripción
[0-9]	\d	Un dígito del 0 al 9.
[^0-9]	\D	No existe un dígito del 0 al 9.
[A-Z]		Letra mayúscula de la A a la Z. Excluye ñ o letras acentuadas.
[a-z]		Letra minúscula de la a a la z. Excluye ñ o letras acentuadas.
[A-Za-z0-9]	\w	Carácter alfanumérico (letra mayúscula, minúscula o dígito).
[^A-Za-z0-9]	\W	No existe carácter alfanumérico (letra mayúscula, minúscula o dígito).
[\t\r\n\f]	\s	Carácter de espacio en blanco (espacio, TAB, CR, LF o FF).
[^ \t\r\n\f]	\S	No existe carácter de espacio en blanco (espacio, TAB, CR, LF o FF).
	\xN	Carácter hexadecimal número N.
	\uN	Carácter Unicode número N.

Anclas

Carácter especial	Descripción
^	Ancla. Delimita el inicio del patrón. Significa empieza por .
\$	Ancla. Delimita el final del patrón. Significa acaba en .
\b	Límite de una palabra separada por espacios, puntuación o inicio/final.
\B	Opuesta al anterior: Posición entre 2 caracteres alfanuméricos o no alfanuméricos.

Cuantificadores

Carácter especial	Descripción
*	El carácter anterior puede aparecer 0 o más veces.
+	El carácter anterior puede aparecer 1 o más veces.
?	El carácter anterior puede aparecer o no (es opcional).
{n}	El carácter anterior aparece n veces.
{n,}	El carácter anterior aparece n o más veces.
{n,m}	El carácter anterior aparece de n a m veces.

7 ASÍNCRONÍA

Método	Descripción	Tema
Mediante callbacks	Probablemente, la forma más clásica de gestionar la asíncronía en Javascript.	Ver Callbacks
Mediante promesas	Una forma más moderna y actual de gestionar la asíncronía.	Ver Promesas
Mediante async/await	Seguimos con promesas, pero con <code>async/await</code> añadimos más azúcar sintáctico.	Ver Async/Await

7.1 CALLBACKS

```
setTimeout([FUNCTION] callback, [NUMBER] time).
```

Ejemplo

```
/* Implementación con callbacks */
const doTask = (iterations, callback) => {
  const numbers = [];
  for (let i = 0; i < iterations; i++) {
    const number = 1 + Math.floor(Math.random() * 6);
    numbers.push(number);
    if (number === 6) {
      /* Error, se ha sacado un 6 */
      callback({
        error: true,
        message: "Se ha sacado un 6"
      });
      return;
    }
  }
  /* Termina bucle y no se ha sacado 6 */
  return callback(null, {
    error: false,
    value: numbers
  });
}
```

```
doTask(10, function(err, result) {
  if (err) {
    console.error("Se ha sacado un ", err.message);
    return;
  }
  console.log("Tiradas correctas: ", result.value);
});
```

7.2 PROMESAS



Métodos	Descripción
<code>.then(function resolve)</code>	Ejecuta la función callback <code>resolve</code> cuando la promesa se cumple.
<code>.catch(function reject)</code>	Ejecuta la función callback <code>reject</code> cuando la promesa se rechaza.
<code>.then(function resolve, function reject)</code>	Método equivalente a las dos anteriores en el mismo <code>.then()</code> .
<code>.finally(function end)</code>	Ejecuta la función callback <code>end</code> tanto si se cumple como si se rechaza.

```
fetch("/robots.txt").then(function(response) {  
    /* Código a realizar cuando se cumpla la promesa */  
});
```

```
fetch("/robots.txt")  
.then(function(response) {  
    /* Código a realizar cuando se cumpla la promesa */  
})  
.catch(function(error) {  
    /* Código a realizar cuando se rechaza la promesa */  
});
```

```
fetch("/robots.txt")  
.then(response => {  
    return response.text(); // Devuelve una promesa  
})  
.then(data => {  
    console.log(data);  
})  
.catch(error => { /* Código a realizar cuando se rechaza la promesa */});
```

```
fetch("/robots.txt")  
.then(response => {  
    return response.text(); // Devuelve una promesa  
})  
.then(data => {  
    console.log(data);  
})  
.catch(error => { /* Código a realizar cuando se rechaza la promesa */});
```

```

fetch("/robots.txt")
  .then(response => response.text())
  .then(data => console.log(data))
  .finally(() => console.log("Terminado."))
  .catch(error => console.error(error));

```

Grupos

Métodos	Descripción
<code>Promise.all(ARRAY list)</code>	Acepta sólo si todas las promesas del <code>ARRAY</code> se cumplen.
<code>Promise.allSettled(ARRAY list)</code> <small>ES2020</small>	Acepta sólo si todas las promesas del <code>ARRAY</code> se cumplen o rechazan.
<code>Promise.any(OBJECT value)</code> <small>ES2021</small>	Acepta con el valor de la primera promesa del <code>ARRAY</code> que se cumpla.
<code>Promise.race(OBJECT value)</code>	Acepta o rechaza dependiendo de la primera promesa que se procese.
<code>Promise.resolve(OBJECT value)</code>	Devuelve un valor envuelto en una promesa que se cumple directamente.
<code>Promise.reject(OBJECT value)</code>	Devuelve un valor envuelto en una promesa que se rechaza directamente.

Inmediatas

Mediante los métodos estáticos `Promise.resolve()` y `Promise.reject()` podemos devolver una promesa cumplida o rechazada respectivamente sin necesidad de crear una promesa con `new Promise()`. Esto puede ser interesante en algunos casos, aunque rara vez solemos utilizarlo hoy en día.

```

const doTask = () => {
  const number = 1 + Math.floor(Math.random() * 6);
  return (number % 2 === 0) ? Promise.resolve(number) : Promise.reject(number);
}

```

Ejemplo

```

/* Implementación con promesas */
const doTask = (iterations) => new Promise((resolve, reject) => {
  const numbers = [];
  for (let i = 0; i < iterations; i++) {
    const number = 1 + Math.floor(Math.random() * 6);
    numbers.push(number);
    if (number === 6) {
      reject({
        error: true,
        message: "Se ha sacado un 6"
      });
    }
    resolve({
      error: false,
      value: numbers
    });
  }
});

```

```

doTask(10)
  .then(result => console.log("Tiradas correctas: ", result.value))
  .catch(err => console.error("Ha ocurrido algo: ", err.message));

```

7.3 ASYNC / AWAIT

Async

```
async function funcion_asincrona() {  
    return 42;  
}
```

```
const funcion_asincrona = async () => 42;
```

```
funcion_asincrona().then(value => {  
    console.log("El resultado devuelto es: ", value);  
});
```

Await

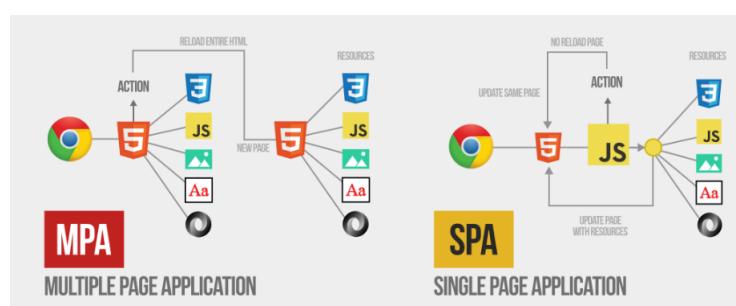
```
const funcion_asincrona = async () => 42;  
  
const value = funcion_asincrona(); // Promise { <fulfilled>: 42 }  
const asyncValue = await funcion_asincrona(); // 42
```

Ejemplo

```
const doTask = async (iterations) => {  
    const numbers = [];  
    for (let i = 0; i < iterations; i++) {  
        const number = 1 + Math.floor(Math.random() * 6);  
        numbers.push(number);  
        if (number == 6) {  
            return {  
                error: true,  
                message: "Se ha sacado un 6"  
            };  
        }  
    }  
    return {  
        error: false,  
        value: numbers  
    };  
}
```

```
const resultado = await doTask(10); // Devuelve un objeto, no una promesa
```

8 HTTP



8.1 FETCH

```
// Petición HTTP
fetch("/robots.txt")
  .then(response => {
    if (response.ok)
      return response.text()
    else
      throw new Error(response.status);
  })
  .then(data => {
    console.log("Datos: " + data);
  })
  .catch(err => {
    console.error("ERROR: ", err.message)
  });
});
```

Parametros

Campo	Descripción
STRING <code>method</code>	Método HTTP de la petición. Por defecto, <code>GET</code> . Otras opciones: <code>HEAD</code> , <code>POST</code> , etc...
OBJECT <code>body</code>	Cuerpo de la petición HTTP. Puede ser de varios tipos: <code>String</code> , <code>FormData</code> , <code>Blob</code> , etc...
OBJECT <code>headers</code>	Cabeceras HTTP. Por defecto, <code>{}</code> .
STRING <code>credentials</code>	Modo de credenciales. Por defecto, <code>omit</code> . Otras opciones: <code>same-origin</code> e <code>include</code> .

Headers

Método	Descripción
BOOLEAN <code>.has(STRING name)</code>	Comprueba si la cabecera <code>name</code> está definida.
STRING <code>.get(STRING name)</code>	Obtiene el valor de la cabecera <code>name</code> .
.set(<code> STRING name, STRING value</code>)	Establece o modifica el valor <code>value</code> a la cabecera <code>name</code> .
.append(<code> STRING name, STRING value</code>)	Añade un nuevo valor <code>value</code> a la cabecera <code>name</code> .
.delete(<code> STRING name</code>)	Elimina la cabecera <code>name</code> .

```
const headers = new Headers();
headers.set("Content-Type", "application/json");
headers.set("Content-Encoding", "br");
```

Response

Propiedad	Descripción
NUMBER <code>.status</code>	Código de error HTTP de la respuesta (100-599).
STRING <code>.statusText</code>	Texto representativo del código de error HTTP anterior.
BOOLEAN <code>.ok</code>	Devuelve <code>true</code> si el código HTTP es <code>200</code> (o empieza por <code>2</code>).
OBJECT <code>.headers</code>	Cabeceras de la respuesta.
STRING <code>.url</code>	URL de la petición HTTP.

Método	Descripción
STRING <code>.text()</code>	Devuelve una promesa con el texto plano de la respuesta.
OBJECT <code>.json()</code>	Idem, pero con un objeto <code>json</code> . Equivalente a usar <code>JSON.parse()</code> .
OBJECT <code>.blob()</code>	Idem, pero con un objeto <code>Blob</code> (binary large object).
OBJECT <code>.arrayBuffer()</code>	Idem, pero con un objeto <code>ArrayBuffer</code> (buffer binario puro).
OBJECT <code>.formData()</code>	Idem, pero con un objeto <code>FormData</code> (datos de formulario).
OBJECT <code>.clone()</code>	Crea y devuelve un clon de la instancia en cuestión.
OBJECT <code>Response.error()</code>	Devuelve un nuevo objeto <code>Response</code> con un error de red asociado.
OBJECT <code>Response.redirect(url, code)</code>	Redirige a una <code>url</code> , opcionalmente con un <code>code</code> de error.

Ejemplos

```
// Petición HTTP
fetch("/robots.txt")
  .then(response => {
    if (response.ok)
      return response.text()
    else
      throw new Error(response.status);
  })
  .then(data => {
    console.log("Datos: " + data);
  })
  .catch(err => {
    console.error("ERROR: ", err.message)
  });

```

```
const request = async (url) => {
  const response = await fetch(url);
  if (!response.ok)
    throw new Error("WARN", response.status);
  const data = await response.text();
  return data;
}

const resultOk = await request("/robots.txt");
const resultError = await request("/nonExistentFile.txt");
```

8.2 URL

```
const url = new URL("https://manz.dev/");
```



```
const url = new URL("https://sub.manz.dev/path/page.html#anchor");
const local = new URL("http://manz:12345@localhost:8000/path/file.mp3");
```

Propiedad	Descripción	Ejemplo url	Ejemplo local
STRING .protocol	Protocolo de comunicación usado.	<code>https:</code>	<code>http:</code>
STRING .hostname	Dominio completo (subdominio + dominio).	<code>sub.manz.dev</code>	<code>localhost</code>
STRING .host	Dominio completo + puerto.	<code>sub.manz.dev</code>	<code>localhost:8000</code>
STRING .origin	Origen (Protocolo + host)	<code>https://sub.manz.dev</code>	<code>http://localhost:8000</code>
STRING .username	Nombre de usuario que ha accedido.	Vacio	<code>manz</code>
STRING .password	Contraseña del usuario que ha accedido.	Vacio	<code>12345</code>
STRING .port	Puerto en escucha de la web.	Vacio	<code>8000</code>
STRING .pathname	Ruta completa (ruta + documento)	<code>/path/page.html</code>	<code>/path/file.mp3</code>
STRING .hash	Ancla (lo explicamos más adelante)	<code>#anchor</code>	Vacio
STRING .href	Esta propiedad devuelve la URL completa (query strings incluidas, ver más adelante).		

Opcionales



Query String

