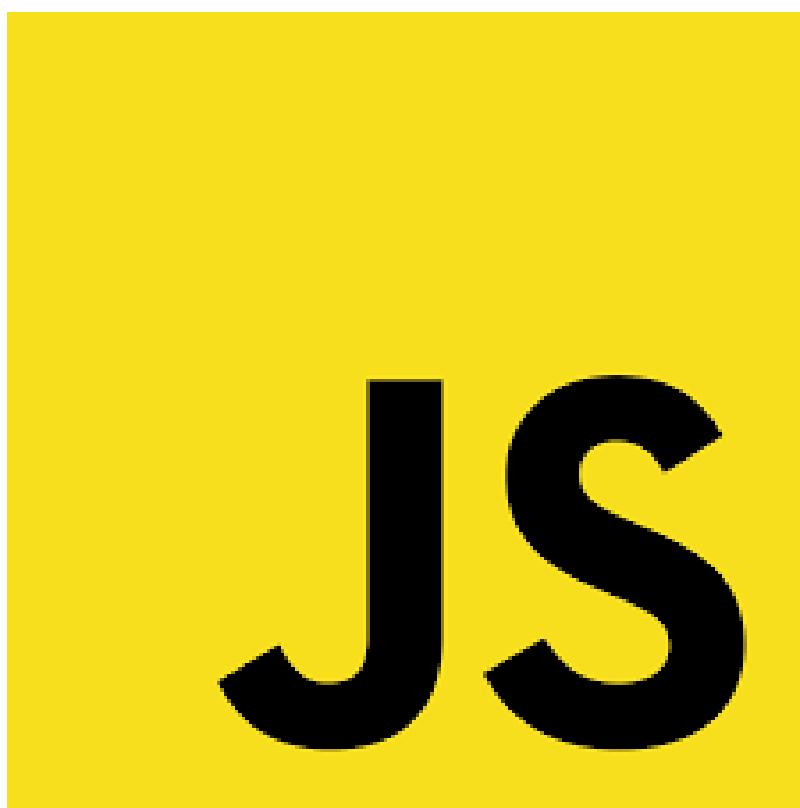


JavaScript Avanzado (Objetos)



Contenido

1.	Objetos definidos por el usuario	3
1.1.	Métodos y propiedades de los objetos	4
2.	Objetos nativos en JavaScript.....	8
2.1.	Global	8
2.1.	Math.....	10
2.2.	Number	11
2.3.	String	12
2.4.	Array	13
2.5.	Maps	16
2.6.	Date.....	17
2.7.	Registros y Tuplas (Records and Tuples).....	18
3.	Interacción con objetos del Navegador.	19
3.1.	Objeto Navigator	19
3.2.	Objeto Screen	21
3.3.	Objeto Window.....	21
3.4.	Objeto Document.....	25
3.5.	Objeto History	27
3.6.	Objeto Location	27
4.	Expresiones regulares	29
4.1.	Creación de una expresión regular	29
4.2.	Escribiendo un patrón de expresión regular	30
4.3.	Trabajando con Expresiones Regulares.....	34
	ANEXO I: Temporal	35
	ANEXO II: Resumen de tipos de objetos predefinidos.....	37

1. Objetos definidos por el usuario

JavaScript está basado en objetos (ojo!, no orientado a objetos). Un **objeto** es una colección de propiedades, y una propiedad es una asociación entre un nombre (o clave) y un valor. Un valor de una propiedad **puede ser una función**, en cuyo caso la propiedad es conocida como un método. Podemos definir nuestros propios objetos con una sintaxis sencilla.

Los objetos en JavaScript, como en tantos otros lenguajes de programación, pueden ser comparados con objetos tangibles de la vida real. Las propiedades de un objeto definen las características de un objeto; se puede acceder a ellas con una simple notación de puntos:

```
miObjeto.nombrePropiedad  
miObjeto.metodo([argumentos])
```

Podemos crear objetos usando dos tipos de sintaxis diferentes. La primera es usando el Objeto nativo JavaScript **Object** de la siguiente manera:

```
var miAuto = new Object();  
miAuto.marca = "Ford";  
miAuto.modelo = "Mustang";  
miAuto.año = 1969;
```

Los objetos son llamados a veces *arreglos asociativos*, ya que cada propiedad está asociada con un valor de cadena que puede ser utilizada para acceder a ella. Así, por ejemplo, puedes acceder y devolver las propiedades del objeto **miAuto** de la siguiente manera:

```
miAuto["marca"] = "Ford";  
miAuto["modelo"] = "Mustang";  
console.log (miAuto["año"]);
```

Otra forma de declarar un objeto es usando una sintaxis de la que, posteriormente, se ha derivado JSON. Esta sintaxis usa los símbolos '{' y '}' para delimitar los pares nombre_propiedad / valor de un objeto. Cada una de las propiedades de un objeto se separan por comas (,). Así, el ejemplo anterior se podría declarar como:

```
var miAuto = {marca:"Ford", modelo:"Mustang", año:1969};
```

Notar que el nombre de las propiedades no tiene por qué entrecorrillarse, ni tampoco los valores de las propiedades que no son de tipo **string**.

Podemos incluir cualquier tipo de valor en una propiedad de un objeto: arrays, cadenas, booleanos, funciones, ..., incluso otros objetos:

```
var obj = {  
    cursos: [1,2,3],
```

```
iniciado: false,  
alumnos: [{nombre:"Pepe",apellidos:"Pérez"}, {nombre:"Ana",apellidos:"Ruiz"}],  
bio: function () {  
    if (this.iniciado)  
        alert ("Hola, el curso actual ha empezado!");  
    else  
        alert ("Hola, el curso actual aún no ha empezado!");  
}  
}
```

Ejemplos de acceso al objeto anterior serían:

```
obj.bio; // Hola, el curso actual aún no ha empezado!  
obj.alumnos[0].nombre; // Pepe
```

1.1. Métodos y propiedades de los objetos

Lo primero que debemos saber es que podemos añadir las siguientes meta-propiedades a cada campo de un objeto:

- **enumerable**: Las propiedades enumerables son aquellas que pueden mostrarse repetidamente en un ciclo *for ... in/of* y con el método `Object.keys()`. Por defecto tiene el valor `true`.
- **configurable**: Puedo modificar el comportamiento de la propiedad, por lo que puedo hacerlos no enumerables, no escribibles o incluso no configurables. Estas propiedades son las únicas que se pueden eliminar con *delete*. Por defecto tiene el valor `true`.
- **writable**: Puedo modificar sus valores usando el operador de asignación (`=`). Por defecto tiene el valor `true`.
- **value**: Valor de la propiedad. Por defecto tiene el valor `undefined`.

```
Object.defineProperties(obj, {  
    "property1": {value: true, writable: true},  
    "property2": {value: "Hello", enumerable: false}  
    // etc. etc.  
});
```

Desde [MDN](#) podemos encontrar la descripción de las propiedades y métodos de los objetos. Mostramos a continuación algunos de ellos:

- **delete Object.campo**

Elimina el campo “*campo*” del objeto.

Para añadir nuevos campos solo tenemos que indicarlo:

```
obj.valor = 80;
```

- **Object.defineProperty(object name, ‘property name’, property descriptor)**

Otro método que nos permite añadir nuevas propiedades a un objeto:

```
Object.defineProperty(obj, 'beca', {enumerable:false, value:4500});
```

Si se usa Object.defineProperty() para definir una propiedad de un objeto, los valores por defecto de *configurable*, *enumerable* y *writable* se establecen a false.

También podemos usar **Object.defineProperties** para definir varias propiedades a la vez:

```
Object.defineProperties(obj, {  
    <Propiedad>: {  
        enumerable: false,  
        configurable: false,  
        writable: true,  
        value: <valor>  
    },  
    <Propiedad>: {  
        enumerable: true,  
        configurable: true,  
        writable: true,  
        value: <valor>  
    },  
    ...  
})
```

- Podemos definir las propiedades **Get** y **Set** y personalizarlas con una función personalizada de la siguiente forma:

```
let persona = {  
    nombre: 'Pepe',  
    apellido: 'Aguilar'  
}  
  
Object.defineProperty(persona, 'nombreCompleto', {  
    get: function () {  
        return this.nombre + ' ' + this.apellido;  
    },  
    set: function (value) {  
        let parts = value.split(' ');  
        if (parts.length == 2) {  
            this.nombre = parts[0];  
            this.apellido = parts[1];  
        } else {  
            throw 'Formato de nombre no válido!';  
        }  
    }  
})
```

```
});  
  
console.log(persona.nombreCompleto);  
persona.nombreCompleto = "Antonio Nogales";
```

Como principales métodos encontramos:

- **Object.keys()**

Devuelve una colección conteniendo los nombres de todas las propiedades enumerables propias de un objeto dado.

- **Object.getOwnPropertyNames()**

Devuelve una colección que contiene los nombres de todas las propiedades enumerables y no-enumerables propias de un objeto dado.

```
var a = {};  
  
Object.defineProperties(a, {  
    one: {enumerable: true, value: 'one'},  
    two: {enumerable: false, value: 'two'}  
});  
  
Object.keys(a); // ["one"]  
Object.getOwnPropertyNames(a); // ["one", "two"]
```

- **Object.getPrototypeOf()**

Devuelve el valor de la propiedad interna *prototype* del objeto indicado.

- **Object.preventExtensions()**

Impide que se puedan agregar nuevas propiedades al objeto. Esto no impide que se puedan seguir eliminando propiedades.

- **Object.isExtensible()**

Determina si está permitida la extensión de un objeto.

- **Object.seal()**

Previene que añadan o borren propiedades de un objeto, se dice que el objeto queda sellado. Similar a *preventExtensions* pero, además, marca todas las propiedades existentes como no-configurables. Así, por ejemplo, no se le podrá añadir o quitar la propiedad *enumerable* a los distintos campos.

- **Object.isSealed()**

Indica si un objeto está sellado, no se podrán modificar los valores de los campos.

- **Object.is(valor1, valor2)**

Compara dos valores para ver si se distinguen (p.e. el mismo). Se puede utilizar para valores y para objetos. Dos valores pueden ser iguales si tienen una de las siguientes propiedades:

- ambos son undefined
- ambos son null
- ambos son true o false
- ambos son strings y tienen la misma longitud con los mismos caracteres
- ambos son el mismo objeto
- ambos son números y
 - ambos +0 (mayores que 0)
 - ambos -0 (menores que 0)
 - ambos son NaN
 - o ambos no son cero o no son de tipo NaN y tienen el mismo valor.

Esta comparación no es igual a la que realiza el operador ==. El operador == aplica varias comprobaciones en ambos sentidos (si no tienen el mismo Type) antes de probar la igualdad (lo que resulta en comportamientos como "" ==), pero **Object.is** no obliga a ninguno de los valores. Ésta tampoco es igual a la que realiza el operador ===. El operador === (y el operador ==) trata los valores -0 y +0 como iguales, y además, trata Number.NaN como no igual a NaN.

Object.is() usado para objetos, solo devolverá *true* si ambos objetos **referencian a la misma posición de memoria**. Esto ocurre si hacemos una copia con obj1 = obj2. Para comparar dos objetos tendremos que implementar manualmente una función que use los métodos vistos más arriba.

- **Object.freeze()**

Congela un objeto: otro código no puede borrar o cambiar ninguna propiedad. Impide que se le agreguen nuevas propiedades; impide que se puedan eliminar las propiedades ya existentes; impide que dichas propiedades, o su capacidad de enumeración, configuración, o escritura, puedan ser modificadas; impide también que se pueda modificar su prototipo. Igual que *seal*, pero además no puedes modificar el valor de las propiedades.

- **Object.isFrozen()**

Indica si un objeto está congelado.

- **Object.hasOwn()**

Este método comprueba si una propiedad está incluida dentro de un objeto o existe dentro del objeto.

```
const miObjeto = {
    titutlo: "Master en Angular",
    descrip: "Aprende Angular desde cero",
    autor: "Pepe Robles",
```

```
    fecha_publi: 2022
};

Object.hasOwnProperty(miObjeto, "categoría"); // devuelve false
Object.hasOwnProperty(miObjeto, "autor"); //devuelve true
```

2. Objetos nativos en JavaScript

Según ECMA-262 los objetos se clasifican en:

1. **Nativos**: Son objetos proporcionados por una implementación de este estándar que son independientes del entorno. Eso quiere decir que cualquier lenguaje basado en ECMAScript debe implementarlos. Son:

- Array
- Boolean
- Date
- Error
- Evaluator
- Function
- Number
- Object
- RangeError
- ReferenceError
- RegExp
- String
- SyntaxError
- TypeError
- URIError

Cada vez que se quiere emplear uno de ellos hay que crear una instancia del objeto por medio de la palabra clave **new**. Por ejemplo:

```
let fecha = new Date();
```

2. **Incorporados** (en inglés built-in): Son, como los anteriores, objetos que tiene que proporcionar una implementación de ECMA-262 y que también son independientes del entorno, pero que deben estar presentes al iniciarse la ejecución de un programa de ECMAScript. Por eso no hace falta crear una instancia para acceder a ellos. Son:

Global y Math

3. **Anfitriones**: Es todo objeto definido por una implementación concreta de ECMAScript.

2.1. Global

Global es un objeto especial porque, como tal, no existe. Si, por ejemplo, lanzamos esta alerta:

```
alert(Math);  
recibimos [objectMath], pero con
```

```
alert(Global);
```

obtenemos un error que nos indica que `Global` no ha sido definido.

¿A qué se debe este misterio? Pues a que según ECMA-262 no puede existir ninguna función independiente de un objeto, y existen métodos —que como sabemos son funciones— que en realidad no pertenecen a ningún objeto concreto. Para “cobijar” estos métodos se define el objeto abstracto `Global`.

Más aún, si probamos `alert(Array);`, `alert(Date);`, `alert(String);`, o cualquier otro objeto nativo, las alertas se parecen a estas:

```
function Array(){
    [nativecode]
}

function Date(){
    [nativecode]
}

function String(){
    [nativecode]
}
```

Esto quiere decir que los objetos nativos son interpretados como funciones, es decir, métodos. ¿Y si son métodos, a qué objeto pertenecen? A **Global**.

No obstante, no hay que alarmarse con la revelación, dado que sólo nos interesan unos pocos métodos de este objeto:

Métodos del objeto Global	
Métodos	Descripción
<code>decodeURI()</code>	Decodifica un URI y lo convierte en una cadena literal.
<code>decodeURIComponent()</code>	Igual que <code>decodeURI</code> , pero para una porción de URI.
<code>encodeURI()</code>	Codifica una cadena literal convirtiéndola en un URI.
<code>encodeURIComponent()</code>	Igual que <code>encodeURI</code> , pero para una porción de URI.
<code>eval()</code>	Trata la cadena no como un literal, sino como una expresión de JavaScript y la intenta ejecutar.
<code>parseFloat()</code>	Convierte una cadena literal en un número con parte decimal.
<code>parseInt()</code>	Convierte una cadena literal en un número entero.

La diferencia entre `encodeURI` y `encodeURIComponent` es que no codifican los mismos caracteres. Así, `encodeURI()` no codificará: `~!@#$&*()=:/,;?+'` y `encodeURIComponent()` no codificará: `~!*()'.` Ejemplo:

```
const url = 'https://www.twitter.com'

console.log(encodeURI(url))          //https://www.twitter.com
console.log(encodeURIComponent(url))  //https%3A%2Fwww.twitter.com
```

```
const paramComponent = '?q=search'
console.log(encodeURIComponent(paramComponent)) //"%3Fq%3Dsearch"
console.log(url + encodeURIComponent(paramComponent))
//https://www.twitter.com%3Fq%3Dsearch
```

NOTA: eval() es una función peligrosa que ejecuta el código el cual es pasado como parámetro con los privilegios de quien llama. Si ejecuta eval() con una cadena de caracteres, ésta podría terminar ejecutando código malicioso dentro de la computadora del usuario con los permisos de su página o extensión web.

2.1. Math

Es un objeto incorporado, por lo que no es necesario declarar una nueva instancia del objeto para invocar sus propiedades o métodos.

```
var constante_Euler = objeto_matematico.E;
```

En el caso de este objeto, las propiedades son valores empleados constantemente en matemáticas:

Propiedades del objeto Math	
Propiedad	Descripción
E	Devuelve la constante de Euler.
LN2	Devuelve el logaritmo natural de 2.
LN10	Devuelve el logaritmo natural de 10.
LOG2E	Devuelve el logaritmo de la constante de Euler en base 2.
LOG10E	Devuelve el logaritmo de la constante de Euler en base 10.
PI	Devuelve el valor de π .
SQRT1_2	Devuelve la raíz cuadrada de 1/2.
SQRT2	Devuelve la raíz cuadrada de 2.

En cuanto a sus métodos, aquí están:

Métodos del objeto Math	
Método	Descripción
abs()	Devuelve el valor absoluto de un número.
acos()	Devuelve el arcocoseno de un valor (en radianes).
asin()	Devuelve el arcoseno de un valor (en radianes).
atan()	Devuelve la arcotangente de un valor (en radianes).
ceil()	Devuelve el entero mayor o igual que el valor pasado.
cos()	Devuelve el coseno de un valor (en radianes).
exp()	Devuelve el valor de la constante de Euler elevada al valor proporcionado.

floor()	Devuelve el entero menor o igual que el valor.
log()	Devuelve el logaritmo natural del valor.
max()	Devuelve el mayor de dos valores.
min()	Devuelve el menor de dos valores.
pow()	Eleva una base a una potencia.
random()	Devuelve un valor pseudoaleatorio entre 0 y 1.
round()	Devuelve un valor redondeado al entero más cercano.
sin()	Devuelve el seno de un valor (en radianes).
sqrt()	Devuelve la raíz cuadrada de un valor.
tan()	Devuelve la tangente de un valor (en radianes).

2.2. Number

Permite trabajar con valores numéricos.

```
var num = new Number(valor);
```

Propiedades:

Number.MAX_VALUE	El número más grande representable.
Number.MIN_VALUE	El número más pequeño representable.
Number.NaN	Valor especial "no es número" NaN.
Number.NEGATIVE_INFINITY	Valor especial para representar infinitos negativos; retorno de un desborde de pila overflow.
Number.POSITIVE_INFINITY	Valor especial para representar infinitos positivos; retorno de un desborde de pila overflow.
Number.prototype	Permite la adición de propiedades a un objeto Number.

Métodos:

Métodos de Number	
toExponential()	Retorna una cadena representando el número en notación exponencial.
toFixed()	Retorna una cadena representando el número en una notación de punto fijo. 1. Número de decimales a representar.
toLocaleString()	Retorna una cadena humanamente legible representando el número utilizando el ámbito local del entorno.
toPrecision()	Retorna una cadena representando el número en una notación de precisión de punto fijo. 1. Número de dígitos (enteros y decimales) con los que se va a representar.
toString()	Retorna una cadena representando el objeto especificado.
valueOf()	Retorna el valor primitivo de un objeto especificado.
isNaN()	Devuelve verdadero si el objeto no representa a un número válido. Falso en otro caso.
isFinite()	Determina si un valor es infinito.

Para conocer el tipo concreto de cada objeto en JavaScript usaremos ‘`typeof <variable>`’.

Ejemplo:

```
var x = 5
typeof x; // ⚡ Devuelve: number
```

2.3. String

Como en el caso de `Array`, la propiedad más importante de `String` es `length`, que es el número de caracteres de una cadena:

```
var cadena_literal = "I have become comfortably brilliant";
var longitud = cadena_literal.length;
// Ahora la variable longitud es igual a 35
// (los espacios en blanco también son caracteres)
```

Sus métodos destacados son:

Métodos del objeto String	
Método	Descripción
<code>charAt()</code>	Devuelve el carácter en un determinado índice de la cadena.
<code>charCodeAt()</code>	Devuelve el valor Unicode del carácter en un determinado índice de la cadena.
<code>fromCharCode()</code>	El inverso del anterior, convierte una cadena de valores Unicode en una literal.
<code>concat()</code>	Combina el texto de dos cadenas para crear una nueva.
<code>indexOf()</code>	Devuelve el índice de la primera coincidencia de un carácter dentro de una cadena, o -1 si no existe.
<code>lastIndexOf()</code>	Devuelve el índice de la última coincidencia de un carácter dentro de una cadena, o -1 si no existe.
<code>match()</code>	Busca las coincidencias de una expresión regular en una cadena literal. Devuelve las coincidencias, si se producen, en un array.
<code>matchAll()</code>	Devuelve un iterador que devuelve todos los grupos coincidentes uno tras otro.
<code>replace()</code>	Sustituye una expresión regular o cadena por una cadena dentro de otra cadena. Devuelve la nueva cadena. Parámetros: 1. Cadena o expresión a buscar. 2. Cadena o expresión por la que sustituir.
<code>replaceAll()</code>	Cambia todos los elementos de la cadena que coincidan con el primer parámetro por el valor del segundo parámetro.
<code>search()</code>	Igual que <code>match</code> , pero devuelve los índices de las coincidencias, o -1 si no se da ninguna.
<code>slice()</code>	Extrae una sección de una cadena. Devuelve la subcadena.
<code>split()</code>	Devuelve un array formado por subcademas de una cadena obtenida con un separador especificado o una expresión regular.
<code>substr()</code>	Devuelve los caracteres solicitados a partir de un índice. 1. Índice de inicio. Si es el único parámetro se extrae hasta el final.

	2. Número de elementos a extraer. (opcional)
toLowerCase()	Devuelve la cadena en minúsculas.
toUpperCase()	Devuelve la cadena en mayúsculas.
trim()	Elimina los espacios en blanco al principio o al final de una cadena.

Podemos mostrar cadenas de texto de varias formas:

```
var valor = 28
alert("Valor es " + valor) // Podemos usar " o ' (comillas simples o dobles)
```

Otra forma es la siguiente:

```
var valor = 28
alert(`Valor es ${valor}`) // Fíjate que este método usa comillas invertidas
```

2.4. Array

Como todos los objetos, Array tiene una serie de propiedades y de métodos.

De las propiedades, el que casi en un 90% de las ocasiones se va a emplear en exclusiva es *length*. Esta propiedad contiene el número de valores almacenados en la matriz:

```
var matriz = new Array();
matriz[0] = 'El primer elemento';
matriz[1] = 'El segundo elemento';
matriz[2] = 'El tercer elemento';

alert(matriz.length);
```

La alerta de este ejemplo nos mostraría el valor 3.

Hay que tener en cuenta que la longitud de una matriz es independiente de la cantidad de valores almacenados, sino que depende de aquel que tenga un índice mayor. Por ejemplo, si declaramos una matriz especificando inicialmente el número de ítems:

```
var matriz = new Array(100);
o asignamos un valor de esta manera:
```

```
var matriz = new Array();
matriz[99] = 'El nonagésimo noveno elemento';
```

La misma alerta de antes nos diría que la longitud de la matriz es 100 en ambos casos.

Otras formas de declarar un array vacío pueden ser:

```
let array1 = Array();
let array2 = [ ];
```

Dicho esto, se recogen ahora los métodos más comunes de Array:

Métodos del objeto Array		
Método	Descripción	¿Modifica la matriz?
pop()	Devuelve el último elemento de la matriz, y lo elimina de la misma.	Sí
push()	Añade un elemento al final de la matriz, y devuelve la nueva longitud.	Sí
reverse()	Invierte el orden de los elementos de la matriz.	Sí
shift()	Devuelve el primer elemento de la matriz, y lo elimina de la misma.	Sí
unshift()	Añade un elemento al principio de la matriz y devuelve la nueva longitud.	Sí
sort()	Ordena los elementos de una matriz.	Sí
splice()	Añade o elimina elementos de una matriz. Parámetros: 1. Posición para añadir o eliminar. 2. Número de elementos a borrar. (opcional) 3. Elementos a añadir. (opcional)	Sí
concat()	Devuelve una matriz con todos los elementos de las matrices y/o valores concatenados.	No
join()	Une todos los elementos de una matriz en una única cadena literal.	No
slice()	Extrae una sección de una matriz y la devuelve como una nueva matriz. 1. Índice de inicio. (opcional) 2. Índice de fin. (opcional)	No
add()	Añade un elemento al final del array. Se le pasa el elemento a añadir.	Si
delete()	Permite eliminar un elemento. Se le pasa el elemento a eliminar y devuelve true o false.	Si
toString()	Devuelve una cadena que representa a la matriz y a sus elementos.	No
indexOf()	Devuelve el primer índice del elemento que concuerda con el valor a comparar, o -1 si no existe. Admite un segundo parámetro opcional para indicar desde donde se empieza la búsqueda.	No
lastIndexOf()	Devuelve el último índice del elemento que concuerda con el valor a comparar, o -1 si no existe. Admite un segundo parámetro opcional para indicar desde donde se empieza la búsqueda.	No
at()	Permite seleccionar el índice de un array de manera flexible. Solo hay que pasarle como parámetro el índice del valor que queremos conseguir: Ejemplo: <code>[1,2,3,4,5].at(2) // devuelve 3 [1,2,3,4,5].at(-1) // devuelve 5</code>	No
map(func)	Permite recorrer el array y modificar los elementos presentes en él, retornando un nuevo array con la misma longitud que el original.	No

	<pre><code>const numbers = [65, 44, 12, 4]; const newArr = numbers.map(myFunction) function myFunction(num) { return num * 10; }</code></pre>	
filter(func)	<p>Recorre el array y retorna un nuevo array con aquellos elementos que pasen una determinada condición.</p> <pre><code>const ages = [32, 33, 16, 40]; ages.filter(checkAdult) // Devuelve [32, 33, 40] function checkAdult(age) { return age >= 18; }</code></pre>	No
forEach(func)	<p>Permite iterar el contenido de un array. Recibe un callback que toma como parámetro el elemento actual de la iteración y el índice del mismo.</p> <pre><code>const numbers = [65, 44, 12, 4]; numbers.forEach(myFunction) function myFunction(item, index, arr) { arr[index] = item * 10; }</code></pre>	Si
find()	<p>Recorre el array y retorna la primera coincidencia del elemento que se busca.</p> <pre><code>const ages = [3, 10, 18, 20]; function checkAge(age) { return age > 18; } function myFunction() { demo.innerHTML = ages.find(checkAge); }</code></pre>	No
some()	<p>Itera el array y retorna un booleano si como mínimo uno de los elementos presentes en el array pasa una condición determinada. Recibe un callback que se encargará de preguntar aquello que queremos dentro del array.</p> <pre><code>const ages = [3, 10, 18, 20]; ages.some(checkAdult); function checkAdult(age) { return age > 18; }</code></pre>	No
every()	<p>Es similar al some(), ya que itera el array y retorna un booleano. Pero esta vez, para que dicho booleano sea true todos los elementos</p>	No

	<p>del array deberán pasar la condición dada. Ejemplo:</p> <pre>const ages = [32, 33, 16, 40]; ages.every(checkAge) // Devuelve false function checkAge(age) { return age > 18; }</pre>	
includes()	Determina si un array incluye un determinado elemento y retorna un booleano según corresponda.	No
	<pre>let array = [12, 15, 18, 9]; array.includes(18); // Devuelve true</pre>	
findIndex()	Retorna el índice del primer elemento de un array que cumpla con la función de prueba proporcionada. En caso contrario devuelve -1.	No
	<pre>const ages = [3, 10, 18, 20]; ages.findIndex(checkAge) // Returns 3 function checkAge(age) { return age > 18; }</pre>	
isArray()	Determina si el valor pasado es un Array.	No

Como se ve en la tabla, algunos de los métodos modifican la matriz, mientras que otros sólo representan la matriz de una forma determinada u ofrecen información sobre ella. Muchos de estos métodos se pueden aplicar también a variables o constantes de tipo **string**.

Hasta ahora hemos trabajado con arrays unidimensionales, para usar arrays multidimensionales tendremos dos opciones:

1. Declarar un array de arrays.
2. Declarar directamente un array de varias dimensiones de la siguiente forma:

```
let tablaNotas = [[,,],[,,]];
```

Como puede verse, cada nivel de anidamiento de corchetes indica una dimensión del array, y el número de elementos separados por comas hace referencia al número de elementos de esa dimensión.

2.5. Maps

Los mapas son unas estructuras que nos permiten guardar colecciones de valores a los que se accede a través de una clave que es única. De esta forma guardamos colecciones de elementos clave-valor. Un mapa recuerda el orden de inserción original de las claves.

Para crear un mapa usaremos:

```
let phones = new Map ([
[678234876, "Elena"],
[623498734, "Jhon"],
[693939339, "Ana"],
[678901233, "Fran"]
]);
```

Para recorrer los mapas la mejor estructura es un *for of* usado de la siguiente forma:

```
for ([tlf, name] of phones)
    console.log("Teléfono de ${name} es ${tlf}");
```

Métodos del objeto Map	
Método	Descripción
keys()	Devuelve una clave del mapa
values()	Devuelve un valor del mapa
set()	Añade un elemento al mapa. Toma como argumentos la clave y el valor a añadir al mapa. También se usa para cambiar un valor.
get()	Obtiene el valor de una clave en un mapa. Requiere el valor de la clave.
delete()	Borra un elemento del mapa pasándole una clave válida.
has()	Busca un elemento en el mapa según una clave pasada como argumento. Devuelve true o false.

A primera vista puede parecer que un mapa y un objeto son la misma estructura, pero existen diferencias importantes:

Objeto	Mapa
No directamente iterable	Directamente iterable
No tiene una propiedad de tamaño	Tiene una propiedad de tamaño
Las claves deben ser cadenas (o símbolos)	Las claves pueden ser de cualquier tipo de datos
Las llaves no están bien ordenadas	Las claves se ordenan por inserción.
Tiene claves predeterminadas	No tiene claves predeterminadas

2.6. Date

El objeto **Date** contiene información sobre la fecha y la hora del sistema.

Sus métodos son:

Métodos del objeto Date

Método	Descripción
<code>parse()</code>	Convierte una cadena que representa una fecha en el número de milisegundos transcurridos desde las 12 en punto de la noche del 1 de enero de 1970.
<code>getDate()</code>	Devuelve el día del mes de acuerdo con la hora local.
<code>getDay()</code>	Devuelve el día de la semana de acuerdo con la hora local.
<code>getFullYear()</code>	Devuelve el año de acuerdo con la hora local.
<code>getHours()</code>	Devuelve la hora de acuerdo con la hora local.
<code>getMilliseconds()</code>	Devuelve el milisegundo de acuerdo con la hora local.
<code>getMinutes()</code>	Devuelve el minuto de acuerdo con la hora local.
<code>getMonth()</code>	Devuelve el mes de acuerdo con la hora local.
<code>getTime()</code>	Devuelve el número de milisegundos transcurridos desde las 12 en punto de la noche del 1 de enero de 1970.
<code>getSeconds()</code>	Devuelve el segundo de acuerdo con la hora local.
<code>getTimezoneOffset()</code>	Devuelve los minutos de diferencia entre la hora local y el UTC.
<code>getUTCDate()</code>	Devuelve el día del mes de acuerdo con la hora universal.
<code>getUTCDay()</code>	Devuelve el día de la semana de acuerdo con la hora universal.
<code>getUTCFullYear()</code>	Devuelve el año de acuerdo con la hora universal.
<code>getUTCHours()</code>	Devuelve la hora de acuerdo con la hora universal.
<code>getUTCMilliseconds()</code>	Devuelve el milisegundo de acuerdo con la hora universal.
<code>getUTCMinutes()</code>	Devuelve el minuto de acuerdo con la hora universal.
<code>getUTCMonth()</code>	Devuelve el mes de acuerdo con la hora universal.
<code>getUTCSeconds()</code>	Devuelve el segundo de acuerdo con la hora universal.
<code>toString()</code>	Convierte la fecha en una cadena literal que la representa.
<code>toDateString()</code>	Convierte una fecha en una cadena legible para seres humanos.
<code>toUTCString()</code>	Convierte una fecha en una cadena acorde con la convención UTC de Internet.
<code>toLocaleString()</code>	Convierte una fecha en una cadena acorde con la convención de la hora local.
<code>toLocaleDateString()</code>	Como el anterior, pero sólo para la porción de día/mes/año.
<code>toLocaleTimeString()</code>	Como el anterior, pero sólo para la porción de hora/minuto/segundo.
<code>valueOf()</code>	Devuelve el número de milisegundos transcurridos desde las 12 en punto de la noche del 1 de enero de 1970.

Podemos crear un nuevo objeto `Date` a través de su constructor de varias formas: sin valores, especificando cada valor del objeto (año, mes, día, hora, minutos, segundos y milisegundos), solamente indicando la fecha en milisegundos y, por último, a partir de un `String`.

NOTA: Para cada método que comienza con `get-` hay un equivalente con `set-`, que sirven para establecer los valores de la fecha en lugar de obtenerlos.

2.7. Registros y Tuplas (Records and Tuples)

Los registros y tuplas introducen estructuras de datos inmutables en JavaScript. Las tuplas son similares a los arrays, pero una vez creadas, sus valores no pueden ser modificados. Los registros son equivalentes a los objetos, pero también inmutables. Por ejemplo:

```
const heroes = #["Batman", "Superman", "Wonder Woman"];
const traitors = #{diane: false, paul: true};
```

Estas estructuras de datos son útiles para garantizar la inmutabilidad, lo que puede ayudar a evitar errores y a mejorar el rendimiento en ciertas aplicaciones.

3. Interacción con objetos del Navegador.

Existen objetos predefinidos en Javascript que permiten el control del navegador en sí mismo. Éstos son los más importantes.

3.1. Objeto Navigator

Permite identificar las características de la plataforma sobre la cual se está ejecutando la aplicación web. En concreto: tipo de navegador, versión y sistema operativo del usuario. Se suele usar para tomar decisiones sobre qué tipo de código ejecutar, ya que no todos los navegadores se comportan del mismo modo con el mismo código.

Propiedades de Navigator	Descripción
appName	Nombre del navegador. (Obsoleto)
appVersion	Versión del navegador. (Obsoleto)
userAgent	Cadena que identifica el navegador y el sistema operativo.
platform	Plataforma del sistema operativo (por ejemplo, "Win32").
language	Idioma preferido del navegador (por ejemplo, "es-ES").
languages	Array con los idiomas preferidos del usuario.
onLine	Devuelve <code>true</code> si el navegador está conectado a internet.
cookieEnabled	Devuelve <code>true</code> si las cookies están habilitadas.
hardwareConcurrency	Número de núcleos de CPU disponibles.
deviceMemory	Cantidad estimada de memoria RAM del dispositivo (en GB).
vendor	Fabricante del navegador (por ejemplo, "Google Inc.").

product	Generalmente devuelve "Gecko" (motor de navegador).
webdriver	Devuelve <code>true</code> si el navegador está controlado por un software de automatización.
maxTouchPoints	Número máximo de puntos táctiles que soporta el dispositivo.
mediaCapabilities	Acceso a capacidades de reproducción de medios (rendimiento, calidad, etc).
permissions	Objeto que permite consultar el estado de los permisos (geolocalización, cámara, etc).
serviceWorker	Permite registrar y gestionar Service Workers.
storage	Proporciona acceso a la API de almacenamiento (persistencia, etc).
bluetooth	Permite acceso a dispositivos Bluetooth.

Métodos de Navigator	Descripción
getBattery()	Devuelve una promesa con el estado de la batería del dispositivo.
getUserMedia(constraints)	Solicita acceso a cámara/micrófono según restricciones (constraints).
registerProtocolHandler(scheme, url, title)	Registra una app como manejador de un protocolo personalizado.
sendBeacon(url, data)	Envía datos de manera asíncrona y sin bloquear la carga de la página.
vibrate(pattern)	Hace vibrar al dispositivo según el patrón especificado.
requestMediaKeySystemAccess(keySystem, config)	Solicita acceso a un sistema de claves DRM para medios.
getGamepads()	Devuelve un array con los gamepads conectados al dispositivo.
share(data)	Activa la interfaz nativa para compartir contenido (data debe tener título, texto, URL, etc.).
canShare(data)	Devuelve <code>true</code> si el contenido puede ser compartido usando <code>navigator.share()</code> .
credentials.get(options)	Solicita credenciales guardadas (como contraseñas o tokens de federación).
credentials.store(credential)	Almacena las credenciales proporcionadas en el navegador.
credentials.preventSilentAccess()	Evita el acceso silencioso a las credenciales del usuario.
getBattery()	Devuelve una promesa con el estado de la batería del dispositivo.

3.2. Objeto Screen

Se corresponde a la pantalla utilizada por el usuario. Este objeto no posee métodos, solo propiedades de lectura.

Propiedad de Screen	Descripción
availHeight	Altura disponible de la pantalla para el uso de ventanas.
availWidth	Anchura disponible de la pantalla para el uso de ventanas.
colorDepth	Número de colores (en bits) que puede representar la pantalla.
height	Altura total de la pantalla
width	Anchura total de la pantalla
pixelDepth	Resolución de la pantalla expresada en bits por pixel.
orientation	Objeto que representa la orientación actual de la pantalla (type, angle).
availLeft (experimental)	Posición horizontal disponible desde el borde izquierdo (en sistemas multimonitor).
availTop (experimental)	Posición vertical disponible desde el borde superior (en sistemas multimonitor).

Métodos de Screen	Descripción
screen.orientation.lock(type)	Intenta bloquear la orientación de la pantalla al tipo especificado ("portrait" o "landscape"). Devuelve una Promise.
screen.orientation.unlock()	Libera la orientación bloqueada, permitiendo que el dispositivo gire automáticamente.

3.3. Objeto Window

Es uno de los objetos más importantes de JavaScript. A partir de él, podemos gestionar las ventanas del navegador y utilizar una serie de propiedades y métodos siguientes:

Propiedad	Descripción
closed	Indica si una variable que representa a una ventana se ha cerrado.
frames	El vector que contiene todos los iframes de la página. Se accede por su índice a partir de 0.

document	Objeto que contiene la página web que se está mostrando. Se verá con el objeto document .
history	Objeto historial de páginas visitadas. Se verá con el objeto history .
innerHeight	Tamaño en pixels del espacio donde se visualiza la página, en vertical.
innerWidth	Tamaño en pixels del espacio donde se visualiza la página, en horizontal.
outerHeight	Tamaño en pixels del espacio de toda la ventana, en vertical. Esto incluye las barras de desplazamiento, botones, etc.
outerWidth	Tamaño en pixels del espacio de toda la ventana, en horizontal. Esto incluye las barras de desplazamiento.
length	Numero de iframes de la ventana.
location	La URL del documento que se está visualizando. Podemos cambiar el valor de esta propiedad para movernos a otra página. Ver también la propiedad location del objeto document.
name	Nombre de la ventana. Lo asignamos cuando abrimos una nueva ventana.
opener	Corresponde al objeto Window que haya abierto una ventana nueva.
resizable	Indica si la ventana se puede redimensionar.
scrollX / pageXOffset	Cantidad de píxeles que la página ha sido desplazada horizontalmente.
scrollY / pageYOffset	Cantidad de píxeles que la página ha sido desplazada verticalmente.
parent	Referencia a la ventana contenedora si está dentro de un frame.
top	Referencia al marco más alto (raíz) de la jerarquía de ventanas.
self	Referencia a sí mismo (igual que window).

Métodos de window:

Método de window	Descripción
alert(texto)	Presenta una ventana de alerta donde se puede leer el texto que recibe por parámetro
prompt(mensaje)	Muestra una caja de diálogo para pedir un dato. Devuelve el dato que se ha escrito.
confirm(texto)	Muestra una ventana de confirmación y permite aceptar o rechazar.
home()	Ir a la página de inicio que haya configurada en el explorador.
back()	Ir una página atrás en el historial de páginas visitadas.

	Funciona como el botón de volver de la barra de herramientas.
forward()	Ir una página adelante en el historial de páginas visitadas. Como si pulsásemos el botón de adelante del navegador.
setInterval(sentencia,milisegundos)	Instrucción o función que será ejecutada indefinidamente en cada intervalo de tiempo indicado. 1. Nombre de la función a ejecutar. 2. Número de milisegundos que indican los intervalos en los que se ejecutará.
setTimeout(sentencia,milisegundos)	Sentencia o función que se ejecutará después de un tiempo de espera determinado en milisegundos.
clearInterval()	Elimina la ejecución de sentencias asociadas a un intervalo indicadas con el método setInterval().
clearTimeout()	Elimina la ejecución de sentencias asociadas a un tiempo de espera indicadas con el método setTimeout().
open(URL, nombre, especificaciones, reemplazo)	Abre una ventana secundaria del navegador.
close()	Cierra la ventana.
focus()	Coloca el foco de la aplicación en la ventana.
blur()	Quitar el foco de la ventana actual.
moveBy(pixelsX, pixelsY)	Mueve la ventana del navegador los pixels que se indican por parámetro hacia la derecha y abajo.
moveTo(pixelsX, pixelsY)	Mueve la ventana del navegador a la posición indicada en las coordenadas que recibe por parámetro.
resizeBy(pixelsAncho,pixelsAlto)	Redimensiona el tamaño de la ventana, añadiendo a su tamaño actual los valores indicados en los parámetros. El primero para la altura y el segundo para la anchura. Admite valores negativos si se desea reducir la ventana.
resizeTo(pixelsAncho,pixelsAlto)	Redimensiona la ventana del navegador para que ocupe el espacio en pixels que se indica por parámetro
scrollBy(pixelsX,pixelsY)	Hace un scroll del contenido de la ventana relativo a la posición actual.
scrollTo(pixelsX,pixelsY)	Hace un scroll de la ventana a la posición indicada por el parámetro. Este método se tiene que utilizar en lugar de scroll.
stop()	Como pulsar el botón de stop de la ventana del navegador.
print()	Como si pulsásemos el botón de imprimir del navegador.
requestIdleCallback(callback)	Ejecuta una función cuando el navegador está inactivo.
matchMedia(query)	Evalúa una media query CSS y devuelve un objeto con el resultado.

Nota: Especificaciones para el método window.open()

Syntax: **window.open(URL, name, specs)**

Valores:

Parámetro	Descripción																
URL	Opcional. Especifica la URL de la página a abrir. Si no se especifica ninguna se abrirá una ventana en blanco con about:blank.																
name	Opcional. Especifica el nombre de la ventana. Los valores soportados son: _blank – La URL se carga en una nueva ventana. Este es el valor por defecto. _parent – La URL se carga en el frame padre. _self – La URL reemplaza la actual página. _top – La URL reemplaza el frameset. (obsoleto) name – Nombre de la ventana (no confundir con el título).																
specs	Opcional. Es una lista de propiedades separada por comas y sin espacios en blanco. Estos son los valores que acepta: <table border="1"> <tr> <td>height=pixels</td><td>La altura de la ventana. El valor mínimo es 100. Ejemplo: "height=100"</td></tr> <tr> <td>left=pixels</td><td>La posición izquierda de la ventana. Los valores negativos no se permiten.</td></tr> <tr> <td>location=yes no 1 0</td><td>Si se muestra o no el campo de dirección. Solo para Opera.</td></tr> <tr> <td>menubar=yes no 1 0</td><td>Muestra o no la barra de menú.</td></tr> <tr> <td>scrollbars=yes no 1 0</td><td>Opciones para mostrar scroll en la ventana. Solamente para Firefox y Opera.</td></tr> <tr> <td>status=yes no 1 0</td><td>Muestra o no la barra de estado.</td></tr> <tr> <td>titlebar=yes no 1 0</td><td>Si debe o no mostrar la barra de título. Ignorado a menos que la aplicación sea una aplicación HTML o un cuadro de diálogo de confianza, es decir, que provenga del mismo servidor.</td></tr> <tr> <td>toolbar=yes no 1 0</td><td>Si debe o no mostrar la barra de herramientas del navegador. IE y Firefox solamente.</td></tr> </table>	height=pixels	La altura de la ventana. El valor mínimo es 100. Ejemplo: "height=100"	left=pixels	La posición izquierda de la ventana. Los valores negativos no se permiten.	location=yes no 1 0	Si se muestra o no el campo de dirección. Solo para Opera.	menubar=yes no 1 0	Muestra o no la barra de menú.	scrollbars=yes no 1 0	Opciones para mostrar scroll en la ventana. Solamente para Firefox y Opera.	status=yes no 1 0	Muestra o no la barra de estado.	titlebar=yes no 1 0	Si debe o no mostrar la barra de título. Ignorado a menos que la aplicación sea una aplicación HTML o un cuadro de diálogo de confianza, es decir, que provenga del mismo servidor.	toolbar=yes no 1 0	Si debe o no mostrar la barra de herramientas del navegador. IE y Firefox solamente.
height=pixels	La altura de la ventana. El valor mínimo es 100. Ejemplo: "height=100"																
left=pixels	La posición izquierda de la ventana. Los valores negativos no se permiten.																
location=yes no 1 0	Si se muestra o no el campo de dirección. Solo para Opera.																
menubar=yes no 1 0	Muestra o no la barra de menú.																
scrollbars=yes no 1 0	Opciones para mostrar scroll en la ventana. Solamente para Firefox y Opera.																
status=yes no 1 0	Muestra o no la barra de estado.																
titlebar=yes no 1 0	Si debe o no mostrar la barra de título. Ignorado a menos que la aplicación sea una aplicación HTML o un cuadro de diálogo de confianza, es decir, que provenga del mismo servidor.																
toolbar=yes no 1 0	Si debe o no mostrar la barra de herramientas del navegador. IE y Firefox solamente.																

	top=pixels	La posición de la parte superior de la ventana. Los valores negativos no se permiten.
	width=pixels	La anchura de la ventana. Valor mínimo es 100.

Valor de retorno:	Referencia a la nueva ventana o null si la llamada falla.
--------------------------	--

3.4. Objeto Document

Se refiere a los documentos que se cargan en la ventana del navegador. Con él es posible manipular las propiedades y el contenido de los elementos principales de las páginas web.

Cuenta con una serie de subobjetos como son los vínculos, puntos de anclaje, imágenes o formularios.

Propiedad	Descripción
documentElement	Nodo raíz del documento (normalmente <html>).
body	Nodo <body> del documento HTML.
head	Nodo <head> del documento HTML.
title	El texto dentro de la etiqueta <title>.
cookie	Nos permitirá crear, modificar y eliminar cookies. Lo veremos más adelante.
domain	Nombre del dominio del servidor de la página.
forms	Un array con todos los formularios de la página.
images	Cada una de las imágenes de la página introducidas en un array. (Javascript 1.1)
lastModified	La fecha de última modificación del documento.
links	Un array con cada uno de los enlaces de la página.
location	La URL del documento que se está visualizando.

referrer	La URL de la página.
tags	Estilos definidos a las etiquetas de HTML en la página web. (Javascript 1.2)
URL	Igual que location, pero es aconsejable utilizar location ya que URL no existe en todos los navegadores.
documentURI	URI del documento (similar a URL).
readyState	Estado de carga del documento: "loading", "interactive" o "complete".
activeElement	Elemento actualmente enfocado (por ejemplo, un <input>).
fullscreenElement	Elemento actualmente en pantalla completa (si lo hay).
characterSet	Codificación de caracteres del documento (normalmente "UTF-8").
scripts	Colección de todos los elementos <script>.
styleSheets	Lista de hojas de estilo cargadas.
children	Lista de nodos hijos del nodo raíz.

En la siguiente tabla se muestran los métodos más usuales del objeto **document**, **no se incluyen los métodos propios de acceso y modificación del DOM**, éstos se verán más adelante.

Método	Descripción
close()	Cierra un documento abierto con document.open().
open()	Abre el flujo del documento.
write()	Escribe dentro de la página web. Podemos escribir etiquetas HTML y texto normal.
writeln()	Escribe igual que el método write(), aunque coloca un salto de línea al final.
exitFullscreen()	Sale del modo de pantalla completa.
hasFocus()	Devuelve true si el documento tiene el foco del usuario.

3.5. Objeto History

Almacena las referencias de todos los sitios web visitados. No podremos acceder a los nombres de las direcciones URL visitadas, ya que son información privada del usuario.

History permite interactuar con el historial del usuario de forma controlada, **sin recargar la página**. Ideal para aplicaciones **SPA (Single Page Applications)**, history permite añadir, reemplazar y navegar por entradas del historial.

Propiedad	Descripción
current	Cadena que contiene la URL completa de la entrada actual en el historial.
next	Cadena que contiene la URL completa de la siguiente entrada en el historial.
length	Entero que contiene el número de entradas del historial (i.e., cuántas direcciones han sido visitadas).
previous	Cadena que contiene la URL completa de la anterior entrada en el historial.
scrollRestoration	Define si el navegador debe restaurar la posición de scroll al navegar ("auto" o "manual").

Método	Descripción
back()	Vuelve a cargar la URL del documento anterior dentro del historial.
forward()	Vuelve a cargar la URL del documento siguiente dentro del historial.
go(posición)	Vuelve a cargar la URL del documento especificado por posición dentro del historial. posición puede ser un entero, en cuyo caso indica la posición relativa del documento dentro del historial; o puede ser una cadena de caracteres, en cuyo caso representa toda o parte de una URL que esté en el historial
pushState(state, title, url)	Agrega una entrada al historial sin recargar la página. Útil para SPA.
replaceState(state, title, url)	Igual que pushState, pero reemplaza la entrada actual del historial.
state (getter)	Devuelve el objeto de estado actual (establecido con pushState o replaceState).

3.6. Objeto Location

Corresponde a la URL de la página web en uso. Sus principales funciones son las de consultar las diferentes partes que forman una URL, como por ejemplo el dominio, el protocolo o el puerto. De este modo, podemos extraer cada componente de la URL y trabajar con ellos de forma separada.

Gracias a este objeto podemos recargar una página, cargar una nueva o remplazar una por otra.

Propiedad	Descripción
hash	El contenido de la URL que se encuentra después del signo # (para los enlaces internos) Ejemplo: http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion <pre>location.hash = "#contacto"; // Esto mueve al ancla sin recargar</pre>
host	El nombre del servidor http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion host= www.ejemplo.com
hostname	La mayoría de las veces coincide con host, aunque en ocasiones, se eliminan las www del principio http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion hostname =www.ejemplo.com
href	La URL completa de la página actual http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion URL =http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion También es una propiedad de escritura, permite cambiar la URL.
pathname	Todo el contenido que se encuentra después del host http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion pathname =/ruta1/ruta2/pagina.html
port	Si se especifica en la URL, el puerto accedido http://www.ejemplo.com:8080/ruta1/ruta2/pagina.html#seccion port = 8080 La mayoría de URL no proporcionan un puerto, por lo que su contenido es vacío http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion port = (vacío)
protocol	El protocolo empleado por la URL, es decir, todo lo que se encuentra antes de las dos barras inclinadas // http://www.ejemplo.com/ruta1/ruta2/pagina.html#seccion protocol = http:
search	Todo el contenido que se encuentra tras el símbolo ?, es decir, la consulta o "query string" http://www.ejemplo.com/pagina.php?variable1=valor1&variable2=valor2 search =?variable1=valor1&variable2=valor2 <pre>location.search = "?modo=oscuro"; // Recarga con nuevos parámetros</pre>
origin	Origen completo de la URL (protocol + "//" + host). Solo lectura.

Método	Descripción
assign()	Carga un nuevo documento. Crea una entrada nueva en el Historial.
reload()	Carga de nuevo el documento actual
replace()	Sustituye la URL del documento actual por otra URL. No crea una entrada nueva en

el historial de navegación, sino que sustituye la entrada actual.

4. Expresiones regulares

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto. En JavaScript, las expresiones regulares también son objetos. Estos patrones son utilizados a través de los métodos **exec** y **test** de **RegExp**, así como los métodos **match**, **replace**, **search** y **split** de **String**. En este documento se describe el uso y funcionamiento de las expresiones regulares en JavaScript.

4.1. Creación de una expresión regular

Una expresión regular puede crearse de cualquiera de las siguientes dos maneras:

1. Utilizando una representación literal de la expresión:

```
var re = /ab+c/<modificador>;
```

La representación literal compila la expresión regular una vez que el script ha terminado de cargar. Es recomendable utilizar esta representación cuando la expresión regular permanecerá sin cambios durante la ejecución del script, ya que ofrece un mejor rendimiento.

2. Constructor del objeto **RegExp**:

```
var re = new RegExp("ab+c","<modificador>");
```

El uso del constructor ofrece una compilación de la expresión regular en tiempo de ejecución. Su uso es recomendable en aquellos escenarios en que el patrón de la expresión regular pueda cambiar durante la ejecución del script, o bien, se desconoce el patrón, dado que se obtiene desde otra fuente, cuando es suministrado por el usuario, por ejemplo.

Los **modificadores** en expresiones regulares en JavaScript son estos:

i	Ignora mayúsculas y minúsculas
g	Búsqueda global por todo el String. Especifica que la búsqueda debe encontrar todas las ocurrencias del patrón en lugar de sólo la primera aparición.
m	Búsqueda en cadenas con múltiples líneas. El modificador "m" solo afecta cuando usamos expresiones regulares con ^ o \$. Si está, encontrará coincidencias en cada línea

(\n). Recuerda que "\n" en un string que se asigna a un <p> o similar, el \n no aparecerá, aunque no se verá el salto de línea, eso ocurre solamente en <pre>.

4.2. Escribiendo un patrón de expresión regular

Un patrón de expresión regular se compone de caracteres simples, como /abc/, o una combinación de caracteres simples y especiales, como /ab*c/ o /Chapter (\d+).\.\d*/. El tercer ejemplo incluye paréntesis, los cuales se emplean como un recurso de memoria. La coincidencia encontrada por esta parte del patrón es almacenada para posterior uso.

Utilizando patrones simples

Los patrones simples se construyen con caracteres para los que se desea una coincidencia exacta. Por ejemplo, el patrón /abc/ coincidirá sólo con esta exacta secuencia y orden de caracteres ('abc'). Tal expresión tendría resultados en las cadenas "*Hi, do you know your abc's?*" y "*The latest airplane designs evolved from slabcraft.*" En ambos existe una coincidencia exacta con la subcadena 'abc'. No hay coincidencia en la cadena 'Grab crab' debido a que, a pesar de que contiene los caracteres 'a', 'b' y 'c', la secuencia 'abc' nunca aparece.

Utilizando caracteres especiales

Cuando la búsqueda de coincidencia requiere algo más que una coincidencia exacta, como por ejemplo el encontrar una o más 'b', o encontrar espacios en blanco, se incluyen en el patrón caracteres especiales. Por ejemplo, el patrón /ab*c/ coincidirá con cualquier secuencia de caracteres en la cual una 'a' preceda a cualquier cantidad de 'b' (cero o más) y sea inmediatamente seguida por una 'c'. En la cadena 'cbbabbbbcdabc,' el patrón coincidirá con la subcadena 'abbbbc'.

La siguiente tabla ofrece una lista completa de los caracteres especiales que pueden utilizarse en las expresiones regulares.

Caracteres especiales para expresiones regulares.	
Carácter	Significado
\	<p>Buscará coincidencias conforme a las siguientes reglas:</p> <p>Una barra invertida precediendo un carácter simple indica que éste debe ser interpretado como un carácter especial y no de forma literal. Por ejemplo, una 'b' sin '\' precediéndole coincidirá con cualquier 'b' minúscula en la cadena, sin embargo, '\b' no coincidirá con algún carácter en específico, representará el <u>delimitador especial de palabras</u>.</p> <p>Una barra invertida que precede a un carácter especial indica que éste deberá ser interpretado literalmente, esto es, como un carácter simple y no como un carácter especial. A esto se le denomina escapado. Por ejemplo, en el patrón '/a*/' el '*' indica que se deberá buscar una secuencia de 'a' cero o más veces, por el contrario, el cambiar el patrón a '/a*/', el carácter especial es interpretado como un carácter</p>

Caracteres especiales para expresiones regulares.	
Carácter	Significado
	simple, y cadenas como 'a*' harán coincidencia. No olvidar <i>escapar</i> la propia barra invertida al usarla en expresiones regulares con strings - RegExp("patrón") - ya que la \ es un carácter de escapado en strings.
^	Coincide con el principio de la entrada. Si la bandera de multilínea (g) está activada, también coincidirá inmediatamente después de un salto de línea. Por ejemplo, /^A/ no coincide con la 'A' en "an A", pero sí con la 'A' en "An E".
\$	Busca el final de la entrada. Si la bandera de multilínea se establece en true, también buscará inmediatamente antes de un carácter de salto de línea. Por ejemplo, la expresión /r\$/ no encontrará el carácter 'r' en la cadena "cenaremos", pero sí la encontrará en la cadena "cenar".
*	Busca el carácter precedente 0 (cero) o más veces. Es equivalente a {0,}.
*	Por ejemplo, la expresión /bo*/ encontrará la subcadena 'boooo' en la cadena "A ghost boooooed" y el carácter 'b' en la cadena "A bird warbled", pero no encontrará nada en la cadena "A goat grunted".
+	Busca el carácter precedente 1 o más veces. Es equivalente a {1,}.
+	Por ejemplo, la expresión /u+/ encontrará el carácter 'u' en la cadena "dulce" y todos los caracteres 'u' en la cadena "duuuuulce".
?	Busca el carácter precedente 0 (cero) o 1 (una) vez. Es equivalente a {0,1}.
?	Por ejemplo, la expresión /e?le?/ encontrará la subcadena 'el' en la cadena "angel" y la subcadena 'le' en la cadena "angle" y también el carácter 'l' en la cadena "oslo".
?	Si se utiliza inmediatamente después que cualquiera de los cuantificadores *, +, ?, o {}, hace que el cuantificador no sea expansivo (encontrando la menor cantidad posible de caracteres), en comparación con el valor predeterminado, que sí es expansivo (encontrando tantos caracteres como le sea posible). Por ejemplo, aplicando la expresión /\d+/ a la cadena "123abc" encuentra "123". Pero aplicando la expresión /\d+?/ a la misma cadena, encuentra solamente el carácter "1".
.	(El punto decimal) coincide con cualquier carácter, solo uno, excepto un carácter de nueva línea. Por ejemplo ,/.n/ coincide 'an' y 'on' en "nay, an apple is on the tree", pero no 'nay'.
(x)	Busca 'x' y recuerda la búsqueda, como el siguiente ejemplo lo muestra. Los paréntesis son llamados <i>paréntesis de captura</i> . <i>Por ejemplo, /(foo)/ encuentra y recuerda 'foo' en "foo bar."</i>

Caracteres especiales para expresiones regulares.	
Carácter	Significado
x(?=y)	Coincide con 'x' sólo si 'x' es seguida por 'y'. Esto se denomina previsión (lookahead, mirar adelante). Por ejemplo, /Jack(?=Sprat)/ coincide con 'Jack' solo si es seguido por 'Sprat'. /Jack(?=Sprat Frost)/coincide con 'Jack' solo si es seguido por 'Sprat' o 'Frost'. Sin embargo, ni 'Sprat' ni 'Frost' serán parte del resultado.
x(?!y)	Coincide con 'x' solo si 'x' no es seguida por 'y'. Es una previsión negativa. Por ejemplo, /\d+(?!\.)/ coincide con números solo si no vienen seguidos por un punto decimal. La expresión regular /\d+(?!\.)/.exec("3.141") coincide con '141' pero no con '3.141'.
x y	Coincide con 'x' o 'y'. Por ejemplo, /green red/ coincide con 'green' en "green apple" y 'red' en "red apple."
{n}	Coincide exactamente con n ocurrencias de la expresión. N debe ser un entero positivo. Por ejemplo, /a{2}/ no coincide con la 'a' en "candy," pero sí con las 2 a de "caandy," y las 2 primeras a en "caaandy."
{n,m}	Donde n y m son enteros positivos y n <= m. Coincide con al menos n y no más de m ocurrencias de la expresión. Si se omite m, no tiene límite de máximo. Por ejemplo, /a{1,3}/ no coincide con "cndy", pero sí con la 'a' en "candy," las primeras 2 a en "caandy," y las primeras 3 a en "aaaaaaaaandy". Note que en "aaaaaaaaandy", la coincidencia es "aaa", aunque la cadena contenga más a en ella.
[xyz]	Grupo de caracteres. Este tipo de patrón coincide con cada carácter dentro de los corchetes. Caracteres especiales como el punto (.) y el asterisco (*) no son especiales en un grupo, así que no necesitan ser escapados. Puede especificar un rango utilizando un guión, como en el siguiente ejemplo. El patrón [a-d], que equivale a [abcd], coincide con la 'b' en "brisket" y la 'c' in "city". El patrón /[a-z.]++ y /[\w.]++ coinciden con toda la cadena "test.i.ng".
[^xyz]	Grupo de caracteres negativo. Significa que coincide con cualquier cosa que no esté en los corchetes . Puede especificar rangos. Todo lo que funciona en el grupo de caracteres positivo funciona también aquí. Por ejemplo, [^abc] es lo mismo que [^a-c], y coincide con la 'r' en "brisket" y 'h' en "chop."
\b	Coincide con backspace (U+0008). Debe ir entre corchetes. (No confundir con \b.)
\b	Coincide con un <i>límite de palabra</i> . Un <i>límite de palabra</i> coincide con la posición donde un carácter de palabra no viene precedido o seguido por otro. Nótese que el

Caracteres especiales para expresiones regulares.	
Carácter	Significado
	<p>límite no estará incluido en la coincidencia. En otras palabras, la longitud del límite es cero. (No confundir con <code>[b]</code>.)</p> <p>Ejemplos:</p> <ul style="list-style-type: none"> <code>\bm/</code> coincide con la 'm' de "moon" ; <code>/oo\b/</code> no tiene coincidencias en "moon", porque las 'oo' están seguidas de una 'n' que es un carácter de palabra; <code>/oon\b/</code> coincide con 'oon' en "moon", porque 'oon' es el final de la cadena, por lo cual no va seguido de un carácter de palabra;
<code>\B</code>	<p>Coincide con un no-límite de palabra. Esto coincide con una posición donde el anterior y el siguiente carácter son del mismo tipo: ambos son o no son caracteres de palabra. El inicio y el final de una cadena se consideran <i>no palabras</i>.</p> <p>Por ejemplo, <code>\B..</code> coincide con 'oo' en "noonday", y <code>/y\B./</code> coincide con 'ye' en "possibly yesterday."</p>
<code>\d</code>	<p>Coincide con un carácter de número. Equivalente a [0-9].</p> <p>Por ejemplo, <code>\d/</code> or <code>/[0-9]/</code> coinciden con el '2' en "B2 is the suite number."</p>
<code>\D</code>	<p>Coincide con cualquier carácter no numérico. Equivalente a <code>[^0-9]</code>.</p> <p>Por ejemplo, <code>\D/</code> o <code>/[^0-9]/</code> coincide con la 'B' en "B2 is the suite number."</p>
<code>\f</code>	Coincide con un salto de página.
<code>\n</code>	Coincide con un salto de línea. Código ASCII = 10.
<code>\r</code>	Coincide con un retorno de carro. Código ASCII = 13.
<code>\s</code>	<p>Coincide con un <i>carácter de espacio</i>, entre ellos incluidos: espacio, tabulador, salto de página, salto de línea y retorno de carro.</p> <p>Por ejemplo, <code>\s\w*/</code> coincide con ' bar' en "foo bar.".</p>
<code>\S</code>	<p>Coincide con todo menos caracteres de espacio.</p> <p>Por ejemplo, <code>\S\w*/</code> coincide con 'foo' en "foo bar."</p>
<code>\t</code>	Coincide con tab (U+0009).
<code>\u{hhhh}</code> o <code>\u{hhhhh}</code>	Busca el carácter con el valor Unicode U+hhhh o U+hhhhh (dígitos hexadecimales). Sin las llaves, busca caracteres que coincidan con UTF-16. Ej: <code>\uD0F5</code> .
<code>\w</code>	<p>Coincide con cualquier carácter alfanumérico, incluyendo el guión bajo. Equivalente a <code>[A-Za-z0-9_]</code>.</p> <p>Por ejemplo, <code>\w/</code> coincide con 'a' en "apple," '5' en "\$5.28," y '3' en "3D."</p>
<code>\W</code>	Coincide con todo menos caracteres de palabra . Equivalente a <code>[^A-Za-z0-9_]</code> .

Usando paréntesis

El paréntesis alrededor de alguna parte del patrón de la expresión regular causa que, parte de la subcadena que coincide, sea recordada. Una vez recordada, puede ser llamada en otro uso.

Por ejemplo, el patrón `/Chapter (\d+)\.\d*/` ilustra caracteres de escape y especiales adicionales e indica que parte del patrón debe recordarse. Coincide con los caracteres 'Chapter' seguidos por uno o más números, luego un punto decimal, seguido por cualquier número 0 o más veces. Además, se utilizaron paréntesis para recordar el primer grupo de números.

Este patrón se encuentra dentro del string "Open Chapter 4.3, paragraph 6" y se memoriza el '4'. El patrón no se encuentra en "Chapter 3 and 4", porque no consigue el punto después del '3'.

En conclusión, unos paréntesis dentro de una expresión regular se usan para comprobar una sub expresión regular dentro de la expresión regular principal.

4.3. Trabajando con Expresiones Regulares

Las expresiones regulares se utilizan con los métodos de RegExp **test** y **exec** y con los métodos de String **match**, **matchAll**, **replace**, **replaceAll**, **search**, and **split**.

Método	Descripción
exec	<p>Un método RegExp que ejecuta una búsqueda por una coincidencia en una cadena.</p> <p>Si el método exec no encuentra ninguna coincidencia, devuelve null. Si encuentra una coincidencia, exec devuelve una matriz y las propiedades del objeto RegExp se actualizan para reflejar los resultados de la búsqueda. El elemento cero de la matriz contiene toda la coincidencia, mientras que los elementos 1 a n contienen las sub coincidencias que se producen dentro de la coincidencia. El comportamiento es idéntico al del método match sin establecer la marca global (g).</p> <p>La matriz devuelta por el método exec tiene tres propiedades: input, index. La propiedad input contiene la cadena completa en la que se busca. La propiedad index contiene la posición de la subcadena coincidente dentro de la cadena completa en la que se busca.</p> <p>La propiedad lastIndex de la variable que representa la expresión regular contiene la posición siguiente al último carácter de la coincidencia. Ej:</p> <pre>var er = /ab/g; er.exec("el abecedario es absurdo"); console.log(er.lastIndex)</pre> <p>Si se establece la marca global para una expresión regular, el método exec busca la cadena empezando en la posición indicada por el valor de lastIndex. Si no se establece la marca global, exec omite el valor de lastIndex y busca desde el principio de la cadena.</p>
test	Un método RegExp que verifica una coincidencia en una cadena. Devuelve true o false.
match	Un método String que ejecuta una búsqueda por una coincidencia en una

	cadena. Devuelve un array de información o null si no existe coincidencia alguna. En este caso no podemos usar lastIndex sobre la expresión regular.
matchAll	Un método de String que devuelve todos los grupos coincidentes uno tras otro. En este caso no podemos usar lastIndex sobre la expresión regular. Es equivalente a match con el modificador /g en su expresión regular.
search	Un método String que verifica una coincidencia en una cadena. Devuelve el índice de la coincidencia, o -1 si la búsqueda falla.
replace y replaceAll	Métodos de String que ejecutan una búsqueda por una coincidencia en una cadena, y reemplaza la subcadena encontrada con una subcadena de reemplazo.
split	Un método String que utiliza una expresión regular o una cadena fija para cortar una cadena y colocarlo en un array de subcademas.

Cuando se quiera saber si un patrón se encuentra en una cadena, habrá que utilizar los métodos `test` o `search`; para obtener más información (pero de ejecución más lenta) se utilizan los métodos `exec` o `match`. Si se utiliza `exec` o `match` y si se logra la coincidencia, estos métodos devuelven un array y actualizan las propiedades del objeto de la expresión regular asociada y también, aquellas del objeto de la expresión regular predefinida, `RegExp`.

En el siguiente ejemplo, el script usa el método `exec` para buscar coincidencias en un string.

```
var myRe = /d(b+)d/g;
var myArray = myRe.exec("cdbbdbbz");
```

Si quieras construir la expresión regular desde un String, otra alternativa es:

```
var myRe = new RegExp("d(b+)d", "g");
var myArray = myRe.exec("cdbbdbbz");
```

Por ultimo, un ejemplo usando el método `matchAll` de `String`:

```
const regexp = /[a-c]/g
const str = 'abc'
const iterator = str.matchAll(regexp)
```

ANEXO I: Temporal

En la encuesta State of JS 2022, la tercera respuesta más común a "¿Qué crees que falta actualmente en JavaScript?" fue una mejor gestión de las fechas.

Esto ha llevado a la propuesta **Temporal**, que ofrece un objeto global estándar para reemplazar el objeto **Date** y soluciona una serie de problemas que han causado mucho dolor a los desarrolladores al trabajar con fechas en JavaScript a lo largo de los años.

Trabajar con fechas en JavaScript es casi siempre una tarea temida; tener que lidiar con pequeñas pero exasperantes inconsistencias, como la locura de que los meses estén indexados a cero, pero los días del mes comiencen en 1.

La dificultad de las fechas ha resultado en la aparición de bibliotecas populares como Moment, Day.JS y date-fns para intentar solucionar los problemas. Sin embargo, la *API Temporal* tiene como objetivo solucionar todos los problemas de forma nativa.

Temporal admitirá múltiples zonas horarias y calendarios no gregorianos listos para usar, y proporcionará una API fácil de usar que hará que sea mucho más fácil analizar fechas a partir de cadenas. Además, todos los objetos temporales serán inmutables, lo que ayudará a evitar errores de cambio de fecha accidental.

Veamos algunos ejemplos de los métodos más útiles que ofrece la API

Temporal.Temporal.Now.Instant()

`Temporal.Now.Instant()` devolverá un objeto `DateTime` al nanosegundo más cercano. Puede especificar fechas particulares utilizando el método `from` de la siguiente manera:

```
const olympics = Temporal.Instant.from('2024-07-26T20:24:00+01:00');
```

Esto creará un objeto `DateTime` que representa el inicio de los Juegos Olímpicos de París a finales de este año a las 20:24 del 26 de julio de 2024 (UTC).

PlainDate()

Esto le permite crear solo una fecha, sin hora:

```
new Temporal.PlainDate(2024, 7, 26);  
Temporal.PlainDate.from('2024-07-26');
```

PlainTime()

Como complemento a `PlainDate()`, podemos usar esto para crear solo una hora sin fecha, usando `.PlainTime()`:

```
new Temporal.PlainTime(20, 24, 0);  
Temporal.PlainTime.from('20:24:00');
```

PlainMonthDay()

`PlainMonthDay()` es similar a `PlainDate`, pero solo devuelve el mes y el día sin información del año (útil para fechas que se repiten el mismo día todos los años, como el día de Navidad y el día de San Valentín):

```
const valentinesDay = Temporal.PlainMonthDay.from({ month: 2, day: 14 });
```

PlainYearMonth()

De manera similar, también existe PlainYearMonth que devolverá solo el año y el mes (útil para representar un mes completo de un año):

```
const march = Temporal.PlainYearMonth.from({ month: 3, year: 2024 });
```

Calculations

Hay una serie de cálculos que se pueden realizar con objetos temporales. Puede sumar y restar varias unidades de tiempo a un objeto de fecha:

```
const today = Temporal.Now.plainDateISO();  
const lastWeek = today.subtract({ days: 7});  
const nextWeek = today.add({ days: 7 });
```

Los métodos hasta y desde le permiten saber cuánto tiempo falta para una fecha determinada o desde que ocurrió la fecha. Por ejemplo, el siguiente código le dirá cuántos días faltan para los Juegos Olímpicos de París:

```
olympics.until().days  
valentinesDay.since().hours
```

Estos métodos devuelven un objeto Temporal.Duration que se puede utilizar para medir una cantidad de tiempo que tiene numerosas unidades diferentes y opciones de redondeo.

Más métodos e info en: <https://tc39.es/proposal-temporal/docs/>

ANEXO II: Resumen de tipos de objetos predefinidos

◆ 1. Objetos Fundamentales

Objeto	Descripción breve
Object	Objeto base del que heredan todos los demás objetos.

Function	Permite crear funciones.
Boolean	Representa un valor lógico: true o false.
Symbol	Tipo de dato único y no enumerable.
BigInt	Representa enteros grandes fuera del rango de Number.

📌 2. Objetos para Números y Fechas

Objeto	Descripción breve
Number	Manejo de números y constantes matemáticas.
Math	Métodos y constantes matemáticas (no es un constructor).
Date	Manipulación de fechas y horas.

📌 3. Objetos para Texto

Objeto	Descripción breve
String	Manejo de cadenas de texto.

📌 4. Objetos para Colecciones de Datos

Objeto	Descripción breve
Array	Lista ordenada de elementos.
Set	Colección de valores únicos.
Map	Colección de pares clave-valor.
WeakSet	Como Set, pero las referencias no evitan el garbage collection.
WeakMap	Como Map, pero con claves "débiles" (objetos).

📌 5. Objetos para Uso Estructurado / de Bajo Nivel

Objeto	Descripción breve
ArrayBuffer	Manipulación de datos binarios de bajo nivel.
DataView	Permite leer y escribir en buffers con control.
TypedArray	Arrays tipados como Int8Array, Float32Array, etc.

6. Objetos relacionados con Expresiones Regulares

Objeto	Descripción breve
RegExp	Manejo de expresiones regulares.

7. Objetos relacionados con Errores

Objeto	Descripción breve
Error	Objeto base para errores.
SyntaxError	Error de sintaxis.
ReferenceError	Acceso a una variable no definida.
TypeError	Uso incorrecto de un tipo de dato.
RangeError	Valor fuera de un rango permitido.
URIError	URI mal formada.
EvalError	Error relacionado con eval() (obsoleto, casi no se usa).

8. Internacionalización (ECMAScript Internationalization API)

Objeto	Descripción breve
Intl	API para soporte internacional: formatos de fecha, número, etc.
Intl.DateTimeFormat	Formateo de fechas localizadas.
Intl.NumberFormat	Formateo de números localizados.

9. Otros útiles

Objeto	Descripción breve
JSON	Parseo y conversión entre objetos y texto JSON.
Promise	Manejo de operaciones asíncronas.
Reflect	API de reflexión para manipular objetos.
Proxy	Intercepta operaciones sobre objetos (get, set, etc.).
console	Métodos de depuración (log, warn, error, etc.).
globalThis	Acceso universal al objeto global (compatible en cualquier entorno).