STRINGS (CADENAS DE TEXTO)

Las cadenas de texto almacenan un conjunto de caracteres. Una cadena de texto es cualquier texto que se encuentra entre comillas simples o dobles.

```
var coche = "Volvo XC60";  // Comillas dobles

var coche = 'Volvo XC60';  // Comillas simples
```

Todos los objetos que almacenan texto se consideran elementos del objeto 'String' por lo que pueden utilizar todas sus propiedades y métodos disponibles. Una propiedad muy utilizada es '***length***' que devuelve el tamaño de una cadena de texto.

```
var txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  // Texto guardado en variable

var tamaño = txt.length; // 'Tamaño' almacena el tamaño de 'txt'
```

Si las comillas simples o dobles se quieren utilizar como caracteres y no como elementos de apertura o cierre es preciso utilizar el carácter contra barra (\) antes de las comillas.

```
var y = "El coche de marca \"Audi \" es blanco."
```

Las cadenas de texto se pueden comportar como objetos y se pueden declarar así. No se recomienda declarar Strings como objeto porque ralentiza la velocidad de ejecución y puede provocar resultados inesperados. ES5 acepta esta última definición en el estándar.

```
var x = "Pep";
var y = new String("Pep"); // Ralentiza ejecución del programa.
```

Además, las cadenas de caracteres o strings se almacenan como array de tal forma que se podría acceder a los caracteres del string como si fuese un Array.

```
var x = "Pep";

var y = x[1];   // Devuelve la letra 'e' como resultado.
```

Métodos en Strings (Cadenas De Texto)

Las cadenas de texto al considerarse objetos de la clase String pueden acceder a las propiedades y métodos que esta dispone. La clase String solo dispone de la propiedad '**length**' que determina el tamaño de la cadena de texto.

- **length:** Es una propiedad que devuelve el tamaño de la cadena de texto. Los strings comienzan por la **posición cero** como si fuesen un array.

```
var str = "Audi Q5";
var pos = str.length;  // Devuelve el número 7.
```

El número de métodos se especifican a continuación:

- **String():** Es el método más representativo de la clase y convierte una variable a string.

```
var edad = 35;
var edadtxt = String(edad); // Convierte edad a texto.
```

- **indexOf():** Devuelve la posición de la primera ocurrencia de un texto especificado en una cadena de texto. La posición del primer carácter es cero. Si no encuentra el texto devuelve -1. El método acepta un segundo parámetro que indica la posición de partida.

```
var str = "El coche Audi es blanco y la marca Audi tiene aros";
var pos = str.indexOf("Audi")  // 'pos' vale 9.
```

- **lastIndexOf():** Devuelve la posición del elemento buscado empezando por la derecha. La posición del primer carácter de la cadena de texto es cero. Si no encuentra el texto devuelve -1. El método acepta un segundo parámetro que indica la posición de partida.

```
var str = "El coche Audi es blanco y la marca Audi tiene aros";
var pos = str.lastindexOf("Audi")  // 'pos' vale 35.
```

- **search():** Busca un valor especificado en una cadena y devuelve la posición inicial. Trabaja igual que el método '*indexOf()*' pero a diferencia de este el método search() acepta muchos más parámetros de configuración.

```
var str = "El coche Audi es blanco";
var pos = str.search("Audi")  // 'pos' vale 9.
```

- **slice():** Extrae una parte de una cadena de texto y devuelve la parte extraída en una nueva cadena. El método tiene dos parámetros (posición de inicio y posición final). Si el valor es negativo se cuenta desde el final de la cadena, pero a la hora de extraer lo realiza igualmente de izquierda a derecha. Empieza a contabilizar desde la posición cero como si fuese un array.

```
var str = "Audi, Mercedes, BMW";
var res = str.slice(6, 13); //Extrae 'Mercedes' y guarda en 'res'
var res = str.slice(6);    // Desde posición 6 hasta el final
var res = str.slice(-13);  // Resultado 'Mercedes, BMW'
var res = str.slice(-13, -6); // Resultado 'Mercedes'
```

- **substring():** Funciona exactamente igual que el método 'slice()' pero la diferencia es que este método no acepta los valores negativos como parámetros. En este caso el segundo parámetro determina cuantos carácteres se deben extraer desde la posición facilitada en parámetro uno.

```
var str = "Audi, Mercedes, BMW";
var res = str.substring(6, 8);  // Extrae 'Mercedes'
```

- **substr():** Realiza una acción casi idéntica a slice(), extraer un texto, pero la diferencia radica en los parámetros que se le pasan. El primer parámetro determina la posición de inicio (incluida), y la segunda el número de caracteres hacia la derecha (y no la posición) que debe extraer.

```
var str = "Audi, Mercedes, BMW";
var res = str.substr(6, 8);  //Extrae 'Mercedes' y guarda en 'res'
```

- **replace():** Realiza la sustitución del primer texto por el segundo. El primer parámetro es el texto que se desea buscar, mientras que el segundo parámetro es el texto por el que se sustituirá. La función devuelve una cadena de texto nueva. El patrón /g sustituye todas las coincidencias en el texto, sino solo realizará la sustitución la primera instancia de la palabra buscada.

  El segundo parámetro de este método también acepta una función que nos permitirá realizar operaciones más complejas.

```
var str = "Audi, Mercedes, BMW";

var res = str.replace(/"Mercedes"/g, "Ferrari");
```

- **replaceAll():** Reemplaza todas las repeticiones del texto buscado por el sustituido en una cadena de texto.

```
var str = "Audi, Mercedes, BMW, Audi, Mercedes, BMW ";

var res = str.replaceAll(/"Mercedes"/g, “Ferrari”);
```

  Resultado: "Audi, Ferrari, BMW, Audi, Ferrari, BMW"

- **toUpperCase():** Convierte toda una cadena de texto a mayúsculas. Para trabajar con textos que no conocemos el formato exactamente es recomendable convertirlo todo a mayúsculas o minúsculas.

```
var txt = "Audi, Mercedes, BMW";
var res = txt.toUpperCase(); //Convierte el texto en mayúsculas
```

- **toLowerCase():** Convierte toda una cadena de texto a minúsculas. Esta función y la anterior pueden ser muy útiles cuando el usuario puede introducir valores en mayúsculas o minúsculas indistintamente.

```
var txt = "Audi, Mercedes, BMW";
var res = txt.toLowerCase(); // Convierte el texto en minúsculas
```

- **concat():** Enlaza dos o más cadenas de texto. Puede ser utilizado para sustituir el carácter más (+) que también está aceptado para concatenar textos. El resultado es una nueva cadena que se debe almacenar en una variable.

```
var text1 = "Hello";
var text2 = "World";
var text3 = text1.concat(" ", text2, "JEJEJE"); //Enlaza 3 textos
```

- **charAt():** Devuelve el carácter de un índice (posición) específico. Las posiciones en un string comienzan por cero.

```
var txt = "Audi, Mercedes, BMW";
var res = txt.charAt(6); //Devuelve la letra "M" según la posición
```

- **charCodeAt():** Devuelve el valor Unicode del carácter situado en un índice específico. El valor Unicode se puede localizar en las tablas ASCII

```
var txt = "Audi, Mercedes, BMW";
var res = txt.charCodeAt(1); // Devuelve el num Unicode 65
```

- **fromcharCodeAt():** Devuelve el carácter de un valor Unicode pasado como parámetro

```
var res = txt.fromcharCodeAt(65); // Devuelve la letra A
```

Es posible acceder a los elementos de un string como si fuese una matriz aunque algunos navegadores no aceptan esta codificación (cada vez menos). El string se trata como si fuese un objeto con lo que ralentiza el programa. Se debe evitar trabajar con cadenas de texto tratadas como matrices ya que los procedimientos no son o eficientes que deberían ser durante su ejecución.

- **split():** Este método permite **convertir una cadena de texto en una matriz**. Hay que especificar qué carácter es el elemento separador para cada posición de la matriz (Array). El valor devuelto se almacena en una variable como array.

```
var txt = "Audi, Mercedes, BMW";
```

```
var res = txt.split(","); // Convierte un string a Array
```

- **repeat():** Este método **repite la cadena tantas veces como se le especifique** en el parámetro. El valor puede ser almacenado en otra variable. Este método se utiliza para simular texto en la web y poder maquetar correctamente la página con contenido cuando estas se ajustan dinámicamente.

```
var txt = "Audi, Mercedes, BMW";

var res = txt.repeat(2);  // Repite dos veces el texto
```

Resultado: "Audi, Mercedes, BMW, Audi, Mercedes, BMW"

- **localeCompare():** El método comprueba **si dos valores son iguales o no**. Con valor cero las dos cadenas son idénticas, con valor -1 o 1 determina que una es mayor que la otra. Este método hace diferenciación entre mayúsculas y minúsculas.

```
var str1 = "ab";

var str2 = "cd";

var n = str1.localeCompare(str2);  // Diferencia Mayusc y Minusc
```

- **startsWith():** (ES2015) Comprueba **si una cadena comienza por el texto especificado**. El resultado devuelve True o False. El valor se puede guardar en una variable. El segundo parámetro es la posición de inicio de la búsqueda.

```
var txt = "Audi, Mercedes, BMW";

var res = txt.starstWith("Audi", 0); // Devuelve True
```

- **endsWith():** (ES2015) Comprueba **si una cadena termina por el texto especificado**. El resultado devuelve True o False. El valor se puede guardar en una variable.

```
var txt = "Audi, Mercedes, BMW";

var res = txt.endsWith("Audi"); // Devuelve False
```

- **includes():** (ES2015) Comprueba **si una cadena contiene el texto especificado**. El resultado devuelve True o False. El valor se puede almacenar en una variable. El segundo parámetro es la posición de inicio de la búsqueda.

```
var txt = "Audi, Mercedes, BMW";

var res = txt.includes("BMW", 10); // Devuelve True
```

- **padStart():** (ES2017) Este método rellena por la izquierda de un texto con tantos caracteres como se le especifique.

```
var txt = "5";

txt.padStart(6, "0");  // Devuelve '000005'
```

- **padEnd():** (ES2017) Este método rellena por la izquierda de un texto con tantos caracteres como se le especifique.

```
var txt = "hola";

txt.padEnd(6, "--");  // Devuelve 'hola--'
```

- **match():** Comprueba si una cadena de texto se encuentra en un texto y devuelve como resultado el texto buscado. Con el **patrón /g** devuelve todas las coincidencias que haya encontrado y las almacena dentro de un array. Y con el **patrón /i** (in case sensitive) no hace distinción entre mayúscula y minúsculas.

```
var str = "Manzana, banana, anciana";

var res = str.match(/ana/gi);
```

Resultado como array: "ana, ana, ana"

- **toString():** Devuelve el valor del String. El valor se puede almacenar en una variable.

```
var str = "Hola mundo!";

var res = str.toString();
```

- **trim():** Devuelve la cadena especificada **sin espacios a ambos lados** (derecha e izquierda).

```
var str = "   Hola mundo!   ";
var res = str.trim();
```

Resultado 'res': "Hola mundo!"

- **trimStart():** (ES2017) Devuelve el texto **sin espacios a la izquierda**.

```
var str = "   Hola mundo!   ";

var res = str.trim(); // Devuelve "Hola mundo!     "
```

- **trimEnd():** (ES2017) Devuelve el texto **sin espacios a la derecha**.

```
var str = "   Hola mundo!   ";

var res = str.trim();   // Devuelve "    Hola mundo!"
```

- **toLocaleLowercase()** y **toLocaleUpperCase()**: Estas dos funciones son un pocos especiales respecto al resto de funciones de texto en JavaScript. Convierten el texto en minúsculas y mayúsculas respectivamente, pero respetando la ubicación y configuración regional del host. La configuración regional **se basa en la configuración del idioma del navegador**.

  Normalmente devolverá el mismo resultado que la función **toLowerCase()** salvo en zonas donde haya conflictos con varios idiomas (Bélgica, Austria, Canadá, Nueva Zelanda) que entonces la función adaptará el resultado a la zona geográfica regional.

Plantillas (backtics) y literales

Las **plantillas** son una forma de trabajar concatenando textos, pero escribiendo la sintaxis de una forma que en el futuro pueda hacer más fácil y accesible el mantenimiento del código.

Para escribir el texto con esta sintaxis el texto resultante deberá estar dentro de los acentos abiertos (*en inglés **'backticks'***) y escribir el nombre de las variables que almacenan los valores de texto con el símbolo dollar $ y comprendido entre las dos llaves. De esta forma es más sencillo realizar cualquier modificación de código porque dispondremos de una plantilla más versátil.

```
var lenguaje = 'JavaScript';

var lenguaje2 = 'HTML';

var mensaje = `Me gusta ${lenguaje} y su integración con ${lenguaje2}`;

console.log(mensaje);
```

Los **literales** son un concepto muy parecido que nos permite mostrar en la página el texto escrito de la misma forma, respetando los saltos de línea y la misma estructura como haya sido creada. También se realiza dentro de los acentos abiertos (*en inglés 'backticks'*) y siguiendo la misma lógica que en el caso anterior.

```
var mensajeMultilinea = `Hola mundo, estoy aprendiendo
        ${lenguaje} y me gusta como se integra con ${lenguaje2}`;

    console.log(mensajeMultilinea);
```

NOTA: La concatenación de textos también se puede realizar utilizando el operador más (+) pero este sistema es más eficiente a la hora de realizar el mantenimiento del código y los algoritmos.

La clase String dispone de una serie de métodos para modificar determinadas propiedades de HTML. Estos **NO son métodos estandarizados** y pueden no funcionar en todos los navegadores. Es recomendable en la medida de lo posible no utilizarlos.

Todos estos métodos pertenecen a ECMASript 1. A continuación, se muestran estos métodos:

```
txt.anchor("ancla"); // Crea un ancla del tipo <a name=”.”> … </a>

var result = str.big(); // Pone letra grande en HTML es <big>

var result = str.blink(); // Texto que parpadea, descatalogado.

var result = str.bold(); // Texto en negrita equivalente a <b>

var result = str.fixed(); // Texto como teletipo <tt>

var result = str.fontcolor("green"); // Color de letra

var result = str.fontsize(7); //Tamaño de letra, solo los 7 tipos HTML

var result = str.italics(); //  Texto en cursiva equivalente <i>

var result = str.link("https://www.google.com"); // Crea enlace <a>…</a>

var result = str.small(); // Texto en letra pequeña <small>

var result = str.strike(); // Muestra el texto tachado <strike>

var result = str.sub();  // Texto como subíndice en HTMl <sub>

var result = str.sub();  // Texto como superíndice en HTML <sup>
```

NUMBERS

JavaScript tiene solo un tipo de número. Los números pueden contener o no decimales. Los decimales **se introducen con punto y no con coma**. Los números en JavaScript difieren de otros lenguajes en no disponer de un tipo de variable para cada tipo de número (entero, decimal, entero largo, etc…).

En Javascript todos los números están en formato binario de doble precisión de 64 bits según la IEEE 754 y comprenden los números desde –(253)-1 y 253-1

Los números en JavaScript se almacenan en variables de 64 bits (coma flotante de doble precisión) donde los bits 0 a 51 almacenan parte entera, los bits del 52 al 62 almacena parte decimal, y el bit 63 almacena el signo del valor.

```
var x = 34.00;     // Numero con decimales (punto).

var x = 123e5;     // Añade ceros a la derecha 12300000.

var y = 123e-5;    // Añade ceros a la izquierda 0.00123.

var x = 0xFF;      // Valor pasado como hexadecimal.
```

Con el método **toString()** es posible cambiar la base de representación del número.

```
var numero = 128;       // Inicialización en base decimal

numero.toString(16);    // Devuelve 80 en hexadecimal

numero.toString(8);     // Devuelve 200 en octal

numero.toString(2);     // Devuelve 10000000 en binario
```

El valor **Infinity** será devuelto por cualquier método que su valor exceda del máximo permitido. También devuelve este valor cuando se realizan divisiones entre cero.

```
var x =  2 / 0;     // x devolverá como resultado 'Infinity'.
```

La palabra reservada **NaN** indica que un valor no es un número, por tanto, si se intenta realizar operaciones entre textos, o números y textos el valor resultante será **NaN**.

```
var x = 100 / "Audi";  // El resultado de la división es NaN.
```

Sin embargo, la excepción se produce si se realiza un cálculo entre un número y un texto que representa un número que entonces si realiza el cálculo.

```
var x = 100 / "10";  // El resultado de la división es 10.
```

La función **isNan()** comprueba si un valor es un número o no. El resultado devuelto es true o false. Si es una cadena de caracteres que representa un número también devuelve true. Este último aspecto es muy importante tenerlo presente ya que un número en formato texto puede trabajar como un número en formato numérico.

Los números se pueden declarar como objetos pero no se recomienda porque ralentizan la función.

```
var x = 500;     // Numero declarado correctamente
var y = new Number(500); // Declarado como objeto, no recomendable.
```

Las variables tipo **Number()** además de valores decimales en Javascript pueden almacenar valores binarios, octales y hexadecimales utilizando la siguiente notación:

- **Binarios**: empezando el número con **0b** seguido de los ceros y unos que lo formen. Por ejemplo: *let n= 0b101*; para representar el 5.
- **Octales**: empezando el número con un **0** seguido de los números entre 0 y 7 que lo formen. Por ejemplo: *let n = 017*; para representar el 15.
- **Hexadecimal**: empezando el número con un **0x** seguido de los números entre 0 y 9 y las letras entre A y F que lo formen. Por ejemplo: *let n = 0xA*; para representar el 10.

Una notación perfectamente válida desde ES2020 es declarar números utilizando un sistema mas visual

```
const number = 100000000
const number = 100_000_000  // Recomendado
```

Propiedades de los números: Solo se puede acceder a estas propiedades desde el objeto **Number**.

Constante	Valor en Javascript	Descripción
Number.POSITIVE_INFINITY	Infinity	Infinito positivo: +∞
Number.NEGATIVE_INFINITY	-Infinity	Infinito negativo: -∞
Number.MAX_VALUE	1.7976931348623157e+308	Valor más grande
Number.MIN_VALUE	5e-324	Valor más pequeño
Number.MAX_SAFE_INTEGER (ES2015)	9007199254740991	Valor seguro más grande
Number.MIN_SAFE_INTEGER (ES2015)	-9007199254740991	Valor seguro más pequeño
Number.EPSILON (ES2015)	2^{-52}	Número muy pequeño: ε
Number.NaN	NaN	Not A Number

La diferencia entre **Number.MAX_VALUE** y **Number.MAX_SAFE_INTEGER** es que, el primero es el **valor máximo** que es posible representar en Javascript. Por otro lado, el segundo es el **valor máximo** para realizar cálculos con seguridad en Javascript.

Si necesitamos realizar operaciones de muy alta precisión existen librerías específicas como '**decimal.js**' o '**bigNumbers.js**' que facilitan esta tarea.

Métodos en Numbers (Números)

Las variables tipo 'number' pueden acceder a una serie de métodos. Se pueden declarar como objetos con 'new' pero es recomendable no hacerlo porque ralentiza el desarrollo de la función.

- **toString():** Devuelve el número como una cadena de texto. NO confundir con la opción toString() ofrece para cambiar la base de un número (decimal, binario, etc..).

```
var x = 123;
x.toString();  // Numero declarado y convertido a texto
```

- **toExponential():** Convierte un número en forma exponencial. Método poco utilizado.

```
var x = 9.656;
x.toExponential(2);     // Devuelve con 2 dígitos,  9.66e+0
x.toExponential(4);     // devuelve con 4 dígitos,  9.6560e+0
x.toExponential(6);     // Devuelve con 6 dígitos,  9.656000e+0
```

- **toFixed():** Devuelve una cadena con el número de decimales especificado en el método.

```
var x = 9.656;
x.toFixed(0);     // Redondea y devuelve  10
x.toFixed(2);     // Devuelve con 2 dígitos,  9.66
x.toFixed(4);     // Devuelve con 4 dígitos,  9.6560
x.toFixed(6);     // Devuelve con 6 dígitos,  9.656000
```

- **toPrecision():** Devuelve la parte entera y la parte decimal del tamaño especificado, siempre empezando desde la izquierda.

```
var num = 13.3714;
var n = num.toPrecision(3);      // Devolverá 13,3
var n = num.toPrecision(5);      // Devolverá 13,371
```

- **valueOf()**: Devuelve el valor que contenga la variable. Es una función común a muchos objetos. Se utiliza para saber el contenido de la variable.

```
var num = 15;
var n = num.valueOf();    // Devuelve el valor 15
```

Existen unos métodos globales, es decir, utilizables por todas las clases de JavaScript, que afectan a las variables tipo *Number* y que convierten las variables a números.

- **Number()**: Convierte las variables de Javascript a números siempre que representen un número. Si el valor convertido no representase un valor numérico devolvería como resultado NaN. Es recomendable realizar el control de este tipo de conversiones antes de continuar con la ejecución del código. El método String() convierte números a texto en formato tipo texto.

```
x = Number(false); // Devuelve 0, 'true' seria 1.
x = Number("10");  // Devuelve 10, al ser número lo convierte
x = Number("10  20"); // Devuelve NaN, no reconoce un numero
// En una fecha devuelve el valor en milisegundos
x = Number(11/09/1976); // Devuelve 0.0006185335132703555
```

- **parseInt()**: Analiza una cadena y devuelve un número entero. Si el número contiene parte decimal esta no se convierte. Si el número no se puede convertir devuelve NaN.

```
X = parseInt("10");          // Devuelve 10
X = parseInt("10.33");       // Devuelve 10
X = parseInt("10 20 30");    // Devuelve 10
X = parseInt("años 10");     // Devuelve NaN (no es un número)
```

- **parseFloat()**: Analiza una cadena y devuelve un número entero o decimal si este tuviera parte decimal. Si hay varios números separados por espacios en blanco devuelve el primer número. Si el número no se puede convertir devuelve NaN.

```
X = parseFloat("10");          // Devuelve 10

X = parseFloat("10.33");       // Devuelve 10.33

X = parseFloat("10 20 30");    // Devuelve 10

X = parseFloat("años 10");     // Devuelve NaN (no es número)
```

ARRAYS (MATRICES)

Un Array en JavaScript se utiliza para almacenar varios valores en una única variable. Esta variable se puede considerar "especial" porque almacena muchos valores. Los valores del Array pueden ser de diferente tipo, numérico, texto, etc. Para acceder a los valores de la matriz se realiza mediante el número de índice.

A continuación, se muestran diferentes formas de crear Arrays. La primera forma es la más óptima porque no ralentiza la ejecución, es una asignación directa en cada posición del array de tamaño fijo. Al crear objetos con **new** provocará que la rutina vaya más lenta.

```
var cars = ["Saab", "Volvo", "BMW"]; // Método literal de array.

var cars = new Array("Saab", "Volvo", "BMW"); // Crea array iniciándolo

var cars = new Array(); // Crea array sin elementos, método 'lento'.

var cars = new Array(5);  // Crea array de 6 elementos.

var cars = ["Saab", 5, true, "hola"];  // Array mixto

var cars = [];   // Crea array sin elementos, es lo más recomendable
```

El primer elemento de un Array se encuentra en la posición cero [0]. Esto hay que tenerlo presente a la hora de hacer los cálculos.

```
var cars = ["Saab", "Volvo", "BMW"];
document.getElementById("parrafo").innerHTML = cars[0]; // Result Saab
```

El operador **typeof** devuelve el valor primitivo (que no tiene ni propiedades ni métodos) del objeto, es decir, determina qué tipo de valor almacena una variable. A veces es necesario conocer el tipo de elemento para saber como tratarlo. Los valores devueltos pueden ser 'number', 'string', 'boolean'. El valor devuelto en el caso de los arrays será 'object'.

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
typeof fruits;       // Devuelve 'object' porque es un array
```

Desde **ECMAScript 5** la clase array tiene el método **isArray()** que devuelve 'true' o 'false' en función de si es un array o no el elemento o variable evaluada.

```
Array.isArray(fruits); // Devuelve Verdadero o falso si es un array
```

Para crear la clase Array es importante tener en cuenta que se escribe la primera letra en mayúscula. Los arrays comienzan siempre en la posición 0. Un error frecuente es no acordarse que la primera posición del array es cero y no un uno.

El objeto Array y sus Propiedades (2):

- La propiedad ***length*** de la clase array es de lectura-escritura, es decir, permite conocer el tamaño de un Array o asignarle un tamaño. Si asignamos un tamaño que es menor se eliminarán valores, y si es mayor habrá espacios vacíos con el valor *'undefined'*.

  La propiedad tiene la particularidad que devuelve el número de elementos del array empezando por 1 y no tomando como referencia el valor cero del array. Dicho de otra forma, informa del numero de elementos que contiene el array.

```
// Declaración de un array de tamaño.
var fruits = ["Banana", "Orange", "Apple", "Mango"];

// Devuelve 4 que el número de elementos.
var n = str.length;

// Asignamos un nuevo tamaño al array
str.length = 3;
```

- La propiedad ***prototype*** permite **añadir nuevos métodos y propiedades al objeto array**. Para ello es necesario declarar previamente la propiedad o el método sobre el objeto. Es una práctica poco frecuente añadir nuevas propiedades en los objetos de Javascript, pero es posible.

  El siguiente ejemplo muestra como añadir una propiedad al objeto, como asignarle un valor, y como recuperar el valor para mostrarlo por pantalla.

Ejemplo 1: Creación de una propiedad nueva para el objeto Array

```
var dominio_comercial = new Array();
// Aquí declaramos una variable de tipo array.

dominio_comercial.prototype.dominio = ".org";
// Declaramos una propiedad nueva al objeto Array y le
asignamos un valor.

document.write(dominio_comercial.dominio);
// Muestra el valor de la nueva propiedad.
// Ya no es necesario escribir 'prototype'.
```

La diferencia entre los valores **null** y **undefined** es que **undefined** indica que la variable fue declarada pero ningún valor le fue asignado, mientras que **null** indica que existe un valor asignado a la variable, pero es nulo.

Arrays multidimensionales

Los **arrays multidimensionales** son un tipo de variables que en su interior almacenan arrays, es decir arrays dentro de arrays. El siguiente ejemplo muestra un array con dos posiciones y cada una de estas posiciones almacena un array en su interior con 3 y 4 números respectivamente.

Los arrays son objetos que se pueden convertir a texto, concretamente JSON, para que toda la información pueda ser enviada a través de Internet. Hay que recordar que los objetos no se pueden enviar en su formato original y deben ser convertidos a un formato exportable para su envio.

```
var categorias = [ "Terror", "Accion", "Comedia" ];

var peliculas = [ "Scream", "Rambo", "Gladiator", "Benhur", "Lucy" ];

var cine = [ categorías, películas ];

console.log( cine[1][3] ); // Resultado: Benhur.
```

Array Multidimensional 'Cine'	0 (categorías)	1 (películas)
0	Terror	Scream
1	Acción	Rambo
2	Comedia	Gladiator
3		Benhur
4		Lucy

Arrays asociativos:

Los **arrays asociativos** son aquellos arrays que almacenan sus valores en posiciones **identificadas por un texto**. Si el valor de la *'clave'* es un número no haría falta escribirlo, podemos decir que en esta situación estamos trabajando con un array normal a través de la indexación a partir de su numero de posición.

En estos dos ejemplos el primer array asociativo es para almacenar un único registro de datos, mientras que el segundo array almacena un conjunto de registros con todos sus valores. Esta notación o sintaxis es denominada JSON.

Para leer un array asociativo se puede realizar de forma mas cómoda con la sentencia **for…in** o **foreach**

```
let miArrayAssoc = { "clave1":"valor 1" , "clave2": 300 , ..etc..}

var peliculas = [ {titulo: "Superman", year: 2015, pais: "EUA"},
                  { titulo: "Batman", year: 2016, pais: "Canada"},
                  { titulo: "Superlopez", year: 2018, pais: "Spain"} ];

      // Recorrer un array asociativo
      for ( let index in peliculas) {
                  console.log(peliculas[index].titulo)
            }
```

Métodos En Arrays (Matrices)

El objeto Array dispone de una serie métodos para trabajar con la matriz.

- **toString():** Convierte una matriz a una cadena de texto. Este método no es exclusivo de la clase array y se utiliza en otros objetos. Por ejemplo, al convertir números a texto es posible determinar la base del valor de salida (binario, octal, o hexadecimal).

```
var car = ["Audi", "BMW", "Mercedes", "Nissan"];
document.getElementById("demo").innerHTML = car.toString();
// Resultado 'Audi, BMW, Mercedes, Nissan'
```

- **join():** Permite concatenar los elementos de un array **a partir de un carácter** especificado. Si no se especifica el carácter separador lo hace automáticamente con una coma.

```
var car = ["Audi", "BMW", "Mercedes", "Nissan"];
document.getElementById("demo").innerHTML = car.join(" - ");
// Resultado 'Audi - BMW - Mercedes - Nissan'
```

- **pop():** Elimina el último elemento del array y lo devuelve como resultado. El método modifica el tamaño del array. Si el array estuviese vacío devolvería el valor de *'undefined'*.

```
var car = ["Audi", "BMW", "Mercedes", "Nissan"];
document.getElementById("demo").innerHTML = car.pop();
// Resultado 'Audi, BMW, Mercedes'
```

- **push():** Añade al final del array un elemento. Se pueden añadir varios elementos a la vez, pero siempre separados por comas. El método devuelve como resultado la nueva longituddel array

```
var car = ["Audi", "BMW", "Mercedes", "Nissan"];
document.getElementById("demo").innerHTML = car.push("Honda");
// Resultado 'Audi, BMW, Mercedes, Nissan, Honda'
```

- **shift():** Elimina el primer elemento del array y modifica el tamaño del mismo. Devuelve como el elemento que ha sido eliminado.

```
var car = ["Audi", "BMW", "Mercedes", "Nissan"];
document.getElementById("demo").innerHTML = car.shift();
// Resultado 'BMW, Mercedes, Nissan'
```

- **unshift():** Añade al principio del array un elemento nuevo. Este método modifica el tamaño del array y devuelve como resultado la nueva longitud.

```
var car = ["Audi", "BMW", "Mercedes", "Nissan"];
document.getElementById("demo").innerHTML = car.unshift("Honda");
// Resultado 'Honda, Audi, BMW, Mercedes, Nissan'
```

- **delete():** Elimina el elemento con el índice seleccionado. La matriz convierte ese espacio en '*undefined*' pero NO lo elimina de la matriz, solo afecta al valor.

```
var car = ["Audi", "BMW", "Mercedes", "Nissan"];
delete[2].car;
// Resultado 'Audi, BMW, undefined, Nissan'
```

- **splice()**: (*Empalmar*)Permite realizar las acciones de borrar, insertar, y borrar-insertar (a la vez). Cuando se insertan valores el array desplaza el resto de elementos hacia la derecha.
  - El primer parámetro determina la posición donde el elemento será añadido.
  - El segundo parámetro determina cuantos valores se eliminarán hacia la derecha.
  - El tercer parámetro determina el valor o los valores que serán añadidos.

```
var car = ["Audi", "BMW", "Nissan"];
car.splice(1,0,"honda");  // Inserta 'Audi, Honda, BMW, Nissan'
car.splice(1,1,"honda");  // Elim-Insert 'Audi, Honda, Nissan'
car.splice(0,2);  // Elimina los dos primeros valores del array
```

- **slice()**: Crea un nuevo array con un subconjunto de elementos pertenecientes a otro array. Los parámetros pasados indican el índice inicial y final del subconjunto, aunque hay otras formas de extraer subconjuntos de un array. El índice inicial puede ser un número negativo con lo que la selección de elementos comenzaría desde el final. El array inicial se mantiene intacto.

```
var numeros = new Array(1,2,3,4,5,6,7,8,9,10);
var nums1 = numeros.slice(1,5);  // Devuelve Array 2,3,4,5
var nums2 = numeros.slice(-4);   // Devuelve Array 7,8,9,10
var nums3 = numeros.slice(4);    // Devuelve Array 5,6,7,8,9,10
```

- **concat()**: Permite enlazar o concatenar arrays. El resultado que devuelve es la suma de los dos arrays y que deberán guardarse en otra variable para trabajar de forma más cómoda.

```
var marcas1 = ["Audi", "BMW", "Nissan"];
var marcas2 = ["Honda", "Toyota", "Mercedes"];
var marcasT = marcas1.concat(marcas2);  // Enlaza los textos
document.getElementById("parrafo").innerHTML = marcasT;
```

- **indexOf()**: Este método devuelve la posición del valor buscado en el array realizando la búsqueda de izquierda a derecha y empezando por la posición 0. Si no encuentra el valor devuelve -1. El método **search()** realiza la misma acción con el mismo resultado.

```
var marcas1 = ["Audi", "BMW", "Nissan"];
marcas1.indexOf("BMW")   // Devuelve el valor 1.
```

- **lastIndexOf()**: Devuelve la posición del elemento del array especificado realizando la búsqueda de derecha a izquierda. Si hubiese mas de una coincidencia en el array devolvería el último o mayor posición. El resultado devuelto es el valor del índice igual que en el caso anterior. Si no encuentra el valor devuelve -1.

```
var marcas1 = ["Audi", "BMW", "Nissan"];
marcas1.lastindexOf("Audi")   // Devuelve el valor 2.
```

- **sort():** Este método **ordena los elementos de un array de forma alfabética**, ascendente A-Z. Ordena el array y devuelve el valor sobre el mismo array.

```
var marcas = ["Nissan", "BMW", "Audi"];
marcas.sort();    // Devuelve 'Audi, BMW, Nissan'
document.getElementById("demo").innerHTML = marcas;
```

- **reverse():** Ordena de forma inversa los elementos del array, descendente Z-A. En este caso mas que una ordenación inversa lo que realiza es "girar" el array a partir de la posición actual, es decir, si no está ordenado lo ordena descendentemente (Z-A) pero si ya estuviese ordenado lo devolvería de forma ascendente.

```
var marcas = ["BMW", "Nissan", "Audi"];
marcas.reverse();    // Devuelve 'Nissan, BMW, Audi'
document.getElementById("demo").innerHTML = marcas;
```

- **every():** Comprueba **si todos los elementos de una matriz pasan una determinada condición**, es decir, todos los valores han de pasar la prueba para que devuelva **true**. La particularidad es que se le pasa como parámetro una función que es la que ejecuta la condición. Es el equivalente la función Y donde se deben cumplir todas las condiciones.

```
var menu = [
    { nombre: 'Ceviche', precio: 20, pais: 'Perú' },
    { nombre: 'Tacos',   precio: 10, pais: 'México' },
    { nombre: 'Pasta',   precio: 50, pais: 'Italia' },
    { nombre: 'Queso',   precio: 15, pais: 'México' }
]

var resultado = menu.every( plato => plato.precio <= 10);
// RESULTADO: False
```

- **some():** Esta función comprueba que **al menos uno de los valores pasados cumple la condición**, y si es asi, devuelve *'true'*. Para que el resultado final de la función sea *'false'* ningún valor supera la condición. Es el equivalente a la función O.

  Ejemplo 1: Disponemos de un array de objetos con tres atributos y sus valores. El método **some()** utiliza en este caso una arrow function como parámetro que contiene para cada pasada el registro (*'plato'*) y a continuación la condición que evalúa (*'plato.precio'*). El resultado del método devuelve *'True'* porque al menos un valor de todos cumple la condición establecida.

```
var menu = [
      { nombre: 'Ceviche',  precio: 20, pais: 'Perú' },
      { nombre: 'Tacos',    precio: 10, pais: 'México' },
      { nombre: 'Pasta',    precio: 50, pais: 'Italia' },
      { nombre: 'Queso',    precio: 15, pais: 'México' } ]

var resultado = menu.some( plato => plato.precio <= 10);
// RESULTADO: True. El parámetro de la arrow function es el nombre
que le dará el usuario a cada registro en cada una de las pasadas.
```

- **filter():** Este método **crea una matriz con todos los valores que superan la condición**. Si es una matriz de objetos devuelve todos los objetos devueltos por el campo especificado.

  Ejemplo 1: Disponemos de un array con 4 valores. El método **'miFuncion()'** escribe dentro del documento HTML con la etiqueta *'innerHTML'* el resultado del filtro. El método **filter()** llama a una función de forma recurrente tantas veces como elementos dispone el array. Dentro de esta función se comprueba si es valor el mayor o igual que 18. El resultado final es un array que contiene todos los valores que han superado el criterio.

```
// Devuelve el resultado 32, 33, 40 dentro de un array.
var edades = [32, 33, 16, 40];

function checkAdult(edad) {
    return edad >= 18;
}
```

```
function miFuncion() {
    document.getElementById("parrafo").innerHTML =
    edades.filter(checkAdult);
}
```

- **map():** Crea una nueva matriz con los resultados obtenidos al llamar a una función que ejecuta una operación sobre cada elemento de la matriz. El parámetro pasado es una función que ejecuta un cálculo.

```
// Resultado es un array nuevo con los valores 2, 3, 4, 5 que
corresponden a la raíz cuadrada de cada número.
var numbers = [4, 9, 16, 25];
function myFunction() {
    document.getElementById("demo").innerHTML =
    numbers.map(Math.sqrt);
}
```

- **split():** Este método permite **convertir una cadena de texto en una matriz**. Hay que especificar que carácter es el elemento separador para cada posición de la matriz (Array).

```
var txt = "Audi, Mercedes, BMW";
var res = txt.split(",");
// Convierte string a Array
```

- **Array.of():** Convierte el contenido que se le pasa en la función **como un array formal**. En el siguiente ejemplo convierte los tres valores que son tres valores en formato texto en un array con 3 elementos.

```
var platillos = Array.of("carne", "tacos", "pasta");
```

- **Array.from():** Convierte el contenido de un array o un string en un array formal, es decir, la conversión habrá dejado el objeto de tal forma que nos permitirá acceder a los métodos y propiedades de la clase array.

  En el siguiente ejemplo disponemos de un código HTML con una caja y tres párrafos en su interior. A continuación, hay un código Javascript que realiza la misma acción y que carga los valores de dos formas diferentes con sus pequeñas diferencias.

  - La primera opción '*Op1*' carga los valores en un array, pero los métodos a los que puede acceder serán los de la clase string porque no se ha convertido 100% a un elemento del tipo array.
  - La segunda opción 'Op2' además de cargar los datos en un array convierte el elemento a un array real permitiéndonos de esta forma acceder a todos los métodos disponibles del objeto array.

```
----------  Código HTML  ---------
<body>
    <div class="platos">
        <p>Carne</p>
        <p>Tacos</p>
        <p>Pasta</p>
    </div>
    <script src="js/app.js"></script>
</body>

----------  Código Javasript  ---------
// Guarda todos los elementos <p> que se encuentran en <div> dentro de la variable 'plats'. En este caso los métodos accesibles desde la variable serán los de la clase string.

    Op1: var plats = document.querySelectorAll('.platos p'));
```

```
// En este caso también guarda todos los elementos <p> que se encuentran dentro de la etiqueta <div>, pero en este caso al hacer la conversión con el método Array.from() permite posteriormente acceder a los métodos del objeto array.

Op2: var plats = Array.from(document.querySelectorAll('.platos p'));
```

- **find():** Este método (ES6) permite **realizar la búsqueda de un elemento dentro de un array**. El mismo método recorre todos los registros que haya dentro del array. El método **devuelve el valor encontrado** en caso que haya coincidencia con algún elemento del array, o *'undefined'* si ningún valor coincide con el valor buscado. Si hubiese más de un valor coincidente solo devolvería el primer valor encontrado y no una colección de valores en forma de array.

  El contenido de la función es una Arrow function que se ejecuta en cada iteración con los diferentes valores que se encuentra en el array.

  Ejemplo 1: Disponemos de un array declarado con 4 valores en su interior. El método **find()** en este caso recibe una arrow function donde se le pasa un parámetro y a continuación el criterio que tiene que evaluar. La variable *'pElegido'* almacena el resultado de la función.

```
// Declaración del array con 4 elementos.
var platos = ["carne", "tacos", "pasta", "tostadas"];

// Recorre array hasta encontrar el primer elemento coincidente.
var pElegido = platos.find( plato => plato == 'Tacos');
```

El parámetro de la arrow function es el nombre que le dará el usuario a cada registro en cada una de las pasadas.

El código de la arrow function realiza la comparación de elemento del array con el valor buscado.

Ejemplo 2: Tenemos un array con una colección de 4 objetos en su interior. La arrow function recibe como parámetro cada uno de los registros para cada una de las pasadas. El cálculo de la arrow function compara si el nombre de un registro coincide con la palabra *'Tacos'*. En este caso si se produce coincidencia el resultado que devuelve es todo el registro de datos (nombre, precio y país).

```
var menu = [
      {nombre: 'Ceviche', precio: 20, pais: 'Perú'},
      {nombre: 'Tacos', precio: 10 , pais: 'México'},
      {nombre: 'Pasta', precio: 50, pais: 'Italia'}
];

var pElegido = menu.find( plato => plato.nombre == 'Tacos');
// RESULTADO: "nombre:'Tacos', precio:10 , pais:'México'".
```

- **findIndex():** Este método (ES6) realiza la búsqueda de un elemento dentro de un array. En este caso el resultado de la operación devuelve la posición (índice) del array una vez ha localizado el valor. Si no hay coincidencia devuelve -1.

```
var platos = ["carne", "tacos", "pasta", "tostadas"];

// Devuelve el número 1 que corresponde a la 2º posición del array
var pElegido = platos.findIndex( plato => plato == 'tacos');
```

El objeto **boolean()** determina si un valor o una comparación es verdadero o falso.

```
Boolean(10 > 9); // Devuelve True, evalúa la condición.

Boolean(5 == "5"); // Devuelve True.

Boolean(5 === "5"); // Devuelve False. Valor y tipo no iguales.

Boolean("BMW" == "Audi"); // Devuelve False
```

La **copia del contenido de un array a otro array** NO se puede realizar con asignación directa como se realizaría con una variable. En este caso como los valores siempre son referencias a memoria si hacemos una asignación directa entre dos arrays se podrá modificar desde el segundo array el contenido primer array.

```
var platosA = ["carne", "tacos", "pasta"];

platosB = platosA; // Asignación directa del primer al segundo array

platosB[0] = "pollo"; // Cambio de valor en el segundo array.

console.log(platosA); // Resultado: ["pollo", "tacos", "pasta"]
```

La forma clásica de solucionar este problema es copiar los valores de un array a otro array, pero esto nos implicaría generar una sentencia for dentro de nuestro código. Este problema se puede solucionar fácilmente siempre que utilicemos el método ***'assign()'*** del objeto Object de Javascript. De esta forma tendremos la misma variable con los mismos valores pero en zonas de memoria diferente.

```
Object.assign(platosB, platosA); // PlatoB es = platoA.
```

Los **arrays multidimensionales** (objetos dentro de arrays o objetos dentro de objetos) no permiten la utilización de este método ya que solo son capaces de leer hasta el primer nivel de datos. Ante esta situación una buena forma para poder copiar elementos de este tipo es pasarlo a objeto JSON (que básicamente es un texto), y copiarlo a una variable para posteriormente devolverlo a objeto.

Desestructuración de arrays

La **desestructuración de arrays** consiste en realizar una **asignación a la inversa** desde un array hacia unas variables utilizando una sintaxis específica. En el siguiente ejemplo se realiza una asignación a una serie de variables a partir del contenido del array.

```
// 1ª Línea: Declaración del array con sus 4 elementos.
// 2ª a 5ª Línea: Declaración de las 4 variables sin contenido.
// 6ª Línea: Asignación a cada variable desde los elementos del array.

    var platos = ["carne", "tacos", "pasta", "tostadas"];

    var plato1 = null;
    var plato2 = null;
    var plato3 = null;
    var plato4 = null;

    var [plato1, plato2, plato3, plato4] = platos;
```

Si en la asignación de las variables con los datos del array utilizamos la palabra reservada **var** no será necesario declarar anteriormente todas las variables.

Sentencia for…in

Otra forma de interactuar con arrays es mediante la sentencia **for … in**. Este sistema es muy parecido al **for** clásico, pero es más abreviado y permite optimizar mejor el código. La gran ventaja de utilizar este tipo de sentencia es que **permite leer los datos de un array asociativo**, donde el índice pasa de ser un número a un texto.

```
var plato["primero"] = "carne";
```

El siguiente código tiene una variable declarada del tipo array con 5 elementos. La sentencia **for … in** recorre el array tantas veces como elementos tiene ya que la variable *'index'* almacena la posición del array para cada uno de los elementos que dispone.

En el siguiente ejemplo la última línea (*'console.log'*) muestra el valor del array a partir de la posición especificada.

```
var platos = ["carne", "tacos", "pasta", "pescado", "fruta"];

for ( let index in platos) {

    console.log(platos[index])
}
```

Sentencia for…of

Este caso es similar al bucle **for… in** aunque se diferencia de este porque al iterar con los elementos del array no devuelve el índice de la posición sino que **devuelve el valor de la posición** donde está iterando.

```
<script>
    let numeros = [1, 3, 5, 7, 9];

    for (let i in numeros) {
        console.log(i); // log -> 0, 1, 2, 3, 4
    }
</script>
```

For …. in

```
<script>
    let numeros = [11, 33, 55, 77, 99];

    for (let i of numeros) {
        console.log(i);     // log -> 11, 33, 55, 77, 99
    }
</script>
```

For …. of

Sentencia Foreach

La sentencia **foreach** está pensada únicamente para recorrer arrays u objetos de forma más cómoda. Puede tener hasta tres parámetros y los nombres de estos no son relevantes, pero si la posición que ocupan cuando se están informando a la función. No es obligatorio pasar todos los parámetros. A diferencia del *'for'* la sentencia *'foreach'* no puede recorrer una cadena de texto. Además *'foreach'* recorre todo el array sin tener que especificar el tamaño del array. Cada parámetro realiza:

1. *elemento*: Hace referencia al valor del array en esa pasada de bucle según la posición.
2. *Índice*: Determina la posición dentro del array, es un número.
3. *arraycompleto*: Contiene todo el array por si se necesitase.

Es frecuente cuando se invoca esta sentencia pasarle una función anónima como parámetro con los tres argumentos que acepta para poder trabajar interiormente con ellos en la función.

El siguiente código muestra diferentes formas de recorrer arrays y como estas formas han ido evolucionando hasta la utilización del método **forEach.** Este método recorre el array pasándole una función con los tres parámetros que acepta que son (valor, índice, array).

```
var lenguajes = ["JS", "CSS", "PHP"];

------------- Método clásico para recorrer arrays ---------------

for (var i=0; i<lenguajes.length; i++) {

    console.log(lenguajes[i]); // Resultado: Cada pasada es un valor
}

-------- Método forEach para recorrer arrays con función anónima --------

lenguajes.forEach( function( elemento, indice, arraycompleto ) {

      console.log(elemento);    // Resultado: JS (1ª pasada)

      console.log(indice);   // Resultado: 0

      console.log(arraycompleto); // Resultado: JS, CSS, PHP
}
```

```
----- Método forEach para recorrer arrays con arrow function ---------

  lenguajes.forEach(( elemento, indice, arraycompleto ) => {

        document.write(elemento);    // Resultado: JS (1ª pasada)

  }); // No es obligatorio pasar los 3 parámetros
```

Diferencias entre for, forEach, for…in, for…of

Un bucle **for** que recorra la longitud del array accederá a todos los valores del array (también los undefined) y no accederá a las propiedades del array.

Un bucle **forEach** recorrerá todas las claves numéricas del array que no contengan un valor *'undefined'* y no accederá a las propiedades del array.

Un bucle **for/in** recorrerá todas las claves numéricas del array que no contengan un valor '*undefined*' y también todas las propiedades del array. Trabaja con el índice.

El bucle **for/of** trabaja con el valor que se encuentra dentro del array.

Ejemplo:

Un bucle **for** imprime los valores por consola:

- 0: rama
- 1: undefined
- 2: fruta

Un bucle **forEach** imprime los valores por consola sin tener en cuenta los valores ***'undefined'***:

- 0: rama
- 2: fruta

Un bucle **for/in** imprime los valores por consola incluyendo los valores de **arrays asociativos**:

- 0: rama
- 2: fruta
- Hoja: perenne
- Edad: 50