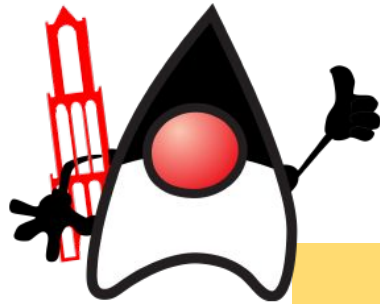




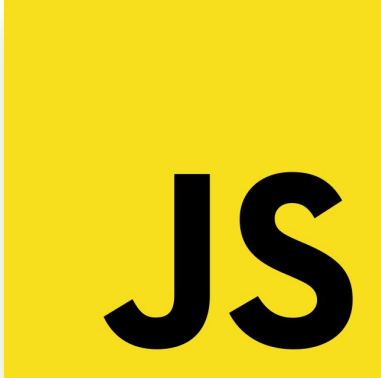
# Curso FullStack Python

Codo a Codo 4.0



# Javascript

## *Parte 4*

A yellow square with a black shadow, containing the letters 'JS' in a bold, black, sans-serif font.

**JS**

# String

En programación, cuando hablamos de una variable que posee información de texto, decimos que su tipo de dato es String. En Javascript, es muy sencillo crear una variable de texto, hay dos formas de hacerlo:

Constructor	Descripción
<b>new String(s)</b>	Crea un objeto de texto a partir del texto <b>s</b> pasado por parámetro.
<b>'s'</b>	Simplemente, el texto entre comillas. Notación preferida.

Los String son tipos de datos primitivos, y como tal, es más sencillo utilizar los literales que la notación con new. Para englobar los textos, se pueden utilizar comillas simples ', comillas dobles " o backticks ` (o comilla invertida o francesa). **Podemos pensar el String como un “array de caracteres”.**

```
// Literales
const texto1 = "¡Hola a todos!";
const texto2 = "Otro mensaje de texto";

// Objeto
const texto1 = new String("¡Hola a todos!");
const texto2 = new String("Otro mensaje de texto");
```

JS

JavaScript llama  
String a una  
cadena de  
caracteres o a un  
solo caracter

# String | Propiedades y métodos

## Propiedades

Propiedad	Descripción
<b>.length</b>	Devuelve el número de caracteres de la variable de tipo string en cuestión

## Métodos

Método	Descripción
<b>.charAt(pos)</b>	Devuelve el carácter en la posición pos de la variable.
<b>.concat(str1, str2...)</b>	Devuelve el texto de la variable unido a str1, a str2...
<b>.indexOf(str)</b>	Devuelve la primera posición del texto str.
<b>.indexOf(str, from)</b>	Idem al anterior, partiendo desde la posición from.
<b>.lastIndexOf(str, from)</b>	Idem al anterior, pero devuelve la última posición.

*Para estos métodos ver ejemplo strings (.html y .js)*

# String | Métodos | charAt y concat

```
var cad= "hola como estas"; JS
```

h	o	l	a		c	o	m	o		e	s	t	a	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Cada carácter está representado por una posición

## .charAt(pos)

```
document.write("CHARAT <br>");  
document.write(cad.charAt(0)); // devuelve "h"  
var pos1= cad[1]; //devuelve o  
var pos2= cad[20]; //indefinido  
document.write(pos1); //devuelve o  
document.write(pos2); //indefinido
```

JS

CHARAT  
h  
o  
undefined

**Charat** permitirá mostrar un carácter determinado de acuerdo a una posición. Podemos guardarlo en una variable y luego mostrarlo en el documento HTML, en la consola o luego utilizarlo en otra tarea.

## .concat(str1, str2...)

```
document.write(cad.concat(". ¿todo bien?", " más texto")); JS
```

**Concat** agregará a la cadena anterior más texto, que pueden estar separados por comas.

CONCAT  
hola como estas. ¿todo bien? más texto

# String | Métodos | indexOf y lastIndexOf

```
var cad= "hola como estas"; JS
```

h	o	l	a		c	o	m	o		e	s	t	a	s
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Cada carácter está representado por una posición

## .indexOf(str), .indexOf(str, from) y .lastIndexOf(str, from)

```
document.write(cad.indexOf("a"));  
document.write(cad.indexOf("a",4));  
document.write(cad.lastIndexOf("o"));  
document.write(cad.lastIndexOf("o", 7)); JS
```

Por su parte, **lastIndexOf** devolverá la última posición de la letra o (la última posición de "o" en **como**), pero al considerar desde la posición 7 la última aparición será la 6, ya que considera de derecha a izquierda.

INDEXOF
3
13
LASTINDEXOF
8
6

**IndexOf** devuelve 3 porque la letra "a" se encuentra en esa posición, pero al considerar desde la posición 4, representada por un espacio, la posición de "a" será la 13.



# String | Más métodos



Método	Descripción
<b>.repeat(n)</b>	Devuelve el texto de la variable repetido n veces.
<b>.toLowerCase()</b>	Devuelve el texto de la variable en minúsculas.
<b>.toUpperCase()</b>	Devuelve el texto de la variable en mayúsculas.
<b>.trim()</b>	Devuelve el texto sin espacios a la izquierda y derecha.
<b>.replace(str, newstr)</b>	Reemplaza la primera aparición del texto str por newstr.
<b>.substr(ini, len)</b>	Devuelve el subtexto desde la posición <b>ini</b> hasta <b>ini+len</b> .
<b>.substring(ini, end)</b>	Devuelve el subtexto desde la posición ini hasta end.

*Para estos métodos ver ejemplo strings-metodos (.html y .js)*

# String | Más métodos | repeat, toLowerCase y toUpperCase

```
var cad= "Aprendiendo JavaScript "; JS
```

## .repeat(n)

```
document.write(cad.repeat(4)); JS
```

*Repeat* repetirá una **n** cantidad de veces la cadena de texto

REPEAT

Aprendiendo JavaScript Aprendiendo JavaScript Aprendiendo JavaScript Aprendiendo JavaScript

## .toLowerCase() y .toUpperCase()

```
document.write(cad.toLowerCase()); JS  
document.write(cad.toUpperCase());
```

TOLOWERCASE  
aprendiendo javascript

TOUPPERCASE  
APRENDIENDO JAVASCRIPT

*Estos métodos convierten a mayúsculas (**toUpperCase**) y minúsculas (**toLowerCase**) una cadena de texto*



# String | Más métodos | trim y replace

```
var cad2= "      Texto de ejemplo" JS
```

.trim()

```
alert(cad2.trim()); JS
```

Esta página dice  
Texto de ejemplo

Aceptar

*En este caso se eliminan los espacios sobrantes de la cadena de texto y se muestran en un alert.*

.replace(str, newstr)

```
document.write(cad.replace("JavaScript", "Python")); JS
```

*Dentro de una cadena de caracteres, **replace** sustituye una subcadena por otra*

Aprendiendo JavaScript



REPLACE  
Aprendiendo Python

# String | Más métodos | substr y substring

```
var cad= "Aprendiendo JavaScript "; JS
```

**.substr(ini, len)**

```
document.write(cad.substr(12, 4)); JS
```

SUBSTR  
Java

En este caso, extraemos desde la posición 12 los 4 caracteres (largo) que conforman la palabra **Java** de la palabra JavaScript.

**.substring(ini, end)**

```
document.write(cad.substring(1, 4)); JS
```

SUBSTRING  
pre

Aquí extraeremos desde el caracter 1 hasta el 4 (no inclusive)

012345  
"Aprendiendo Jav

# Array (arreglo o vector)

Un array es una **colección o agrupación de elementos** en una misma variable, cada uno de ellos ubicado por la posición que ocupa en el array. En Javascript, se pueden definir de varias formas:

Constructor	Descripción
<code>new Array(len)</code>	Crea un array de len elementos .
<code>new Array(e1, e2...)</code>	Crea un array con ninguno o varios elementos.
<code>[e1, e2...]</code>	Simplemente, los elementos dentro de corchetes: []. Notación preferida.

```
// Forma tradicional
const array = new Array("a", "b", "c");
```

JS

```
// Mediante literales (preferida)
const array = ["a", "b", "c"]; // Array con 3 elementos
const empty = []; // Array vacío (0 elementos)
const mixto = ["a", 5, true]; // Array mixto (string, number, boolean)
```

# Array | Acceso a elementos

Propiedad	Descripción
<b>.length</b>	Devuelve el número de elementos del array.
<b>[pos]</b>	Operador que devuelve el elemento número pos del array.

```
const array = ["a", "b", "c"];  
  
console.log(array[0]); // 'a'  
console.log(array[2]); // 'c'  
console.log(array[5]); // undefined  
console.log(array.length); // 3
```

JS

a
c
undefined
3

*Debemos pensar al vector como los vagones de un tren, donde cada vagón va a tener un contenido y un orden. El índice es el orden y el contenido transportado por el vagón del tren es el dato.*

Las posiciones de un array se numeran a partir de **0 (cero)**. Cuando usamos **array[0]** estamos haciendo referencia a la posición 0 del array cuyo contenido, en este caso, es la letra "a".

En el caso de **array[5]** estamos haciendo referencia a una posición que no existe porque el array tiene 3 posiciones, mientras que **length** es una propiedad del array que devuelve la cantidad de elementos que tiene el array.

# Array | Ejemplos (crear, acceder y mostrar elementos)

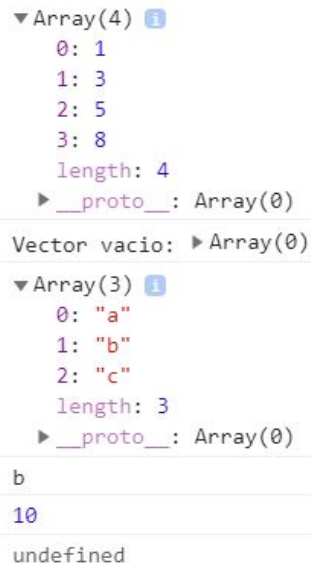
```
const vector= [1,3,5,8]; //0, 1, 2, 3: cantidad de elementos - 1
const vectorVacio= []; //Vector vacío
const vectorDos = new Array("a", "b", "c");
const vectorTres = new Array (1, 5, 10, 15, 20);
```

JS

```
console.log(vector);
document.write(vector);
console.log("Vector vacio:", vectorVacio);
console.log(vectorDos);
console.log(vectorDos[1]);
console.log(vectorTres[2]);
console.log(vectorTres[6]);
```

JS

1,3,5,8



**Tip:** Para acceder al último elemento del array hacemos:

`let ultimo = nombreArray[nombreArray.length - 1]`

donde agregamos - 1 porque las posiciones empiezan desde 0.

Ver ejemplo  
arrays (.html  
y .js)

## Array | Ejemplos (crear, acceder y mostrar elementos)

```
console.log("Elementos del vector 2:");  
for(i = 0; i < vectorDos.length; i++){  
    console.log(vectorDos[i]);  
}
```

JS

Elementos del vector 2:

a

b

c

Utilizando un **for** mostramos el vectorDos. Para ello empleamos **vectorDos.length** que nos da el largo del vector y el **< (menor que)** para recorrer la cantidad de posiciones - 1 (dado que las posiciones en el vector comienzan por 0, sino la última daría indefinido).

```
document.write("Elementos del vector 3: <br>");  
for(i = 0; i < vectorTres.length; i++){  
    document.write(vectorTres[i] + ", ");  
}
```

JS

Elementos del vector 3:  
1, 5, 10, 15, 20,

Utilizando un **for** mostramos el vectorTres de la misma manera que el vectorDos. En este caso mostramos en el body cada elemento del vector, separados por una coma.

[Ver ejemplo arrays \(.html y .js\)](#)

# Array | Métodos (funciones)

Método	Descripción
<b>.push(obj1, obj2...)</b>	Añade uno o varios elementos al final del array. Devuelve tamaño del array.
<b>.pop()</b>	Elimina y devuelve el último elemento del array.
<b>.unshift(obj1, obj2...)</b>	Añade uno o varios elementos al inicio del array. Devuelve tamaño del array.
<b>.shift()</b>	Elimina y devuelve el primer elemento del array.
<b>.concat(obj1, obj2...)</b>	Concatena los elementos (o elementos de los arrays) pasados por parámetro.
<b>.indexOf(obj, from)</b>	Devuelve la posición de la primera aparición de obj desde from.
<b>.lastIndexOf(obj, from)</b>	Devuelve la posición de la última aparición de obj desde from.

**Más información:** <https://lenguajejs.com/javascript/fundamentos/arrays/>

*Para estos métodos ver ejemplo  
arrays-metodos (.html y .js)*

# Array | Métodos | Push, Pop, Unshift y Shift

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango"]; JS
```

## .push(obj1, obj2...)

```
frutas.push("Kiwi", "Pera"); JS
```

PUSH

Banana,Naranja,Manzana,Mango

Banana,Naranja,Manzana,Mango,Kiwi,Pera

## .pop()

```
frutas.pop(); //No tiene argumentos JS
```

POP

Banana,Naranja,Manzana,Mango,Kiwi, ~~Pera~~

Banana,Naranja,Manzana,Mango,Kiwi

```
var colores = ["Rojo", "Amarillo", "Verde", "Celeste"]; JS
```

## .unshift(obj1, obj2...)

```
colores.unshift("Azul", "Naranja"); JS
```

UNSHIFT

Rojo,Amarillo,Verde,Celeste

Azul,Naranja,Rojo,Amarillo,Verde,Celeste

## .shift()

```
colores.shift(); //No tiene argumentos JS
```

SHIFT

~~Azul~~,Naranja,Rojo,Amarillo,Verde,Celeste

Naranja,Rojo,Amarillo,Verde,Celeste



# Array | Métodos | Concat, IndexOf y LastIndexOf

```
var varones = ["Juan", "Pablo"];  
var masVarones = ["Luis", "Pedro"];
```

JS

## .concat(obj1, obj2...)

```
var todos = varones.concat(masVarones);
```

JS

```
todos = todos.concat("Julieta", "Natalia");
```

JS

CONCAT  
Juan,Pablo  
Luis,Pedro  
Juan,Pablo,Luis,Pedro  
Juan,Pablo,Luis,Pedro,Julieta,Natalia

*Declaramos dos vectores y los concatenamos en un tercer vector llamado **todos**. Luego, a ese vector le incorporamos dos valores nuevos.*

```
var letras = ["A", "B", "C", "D", "E", "B", "C"];
```

JS

## .indexOf(obj, from)

```
var pos = letras.indexOf("B");  
pos = letras.indexOf("C", 4);
```

JS

B está en la posición: 1  
C está en la posición: 6

*En el segundo caso partimos desde la pos 4*

## .lastIndexOf(obj, from)

```
var pos2 = letras.lastIndexOf("B");  
pos2 = letras.lastIndexOf("B", 4); //de derecha a izquierda
```

JS

B está en la posición: 5  
B está en la posición: 1

*En el segundo caso parte de la pos 4 (desde la derecha)*

## Array | Otros métodos

Método	Descripción	Referencia
<b>splice()</b>	El método splice () agrega / elimina elementos a / de una matriz y devuelve los elementos eliminados.	<a href="https://www.w3schools.com/jsref/jsref_splice.asp">https://www.w3schools.com/jsref/jsref_splice.asp</a>
<b>.slice()</b>	El método slice () devuelve los elementos seleccionados en una matriz, como un nuevo objeto de matriz. Selecciona los elementos que comienzan en el argumento inicial dado y finalizan en el argumento final dado, pero no lo incluyen.	<a href="https://www.w3schools.com/jsref/jsref_slice_array.asp">https://www.w3schools.com/jsref/jsref_slice_array.asp</a>

# Array | Métodos de orden

Método	Descripción
<b>.reverse()</b>	Invierte el orden de elementos del array.
<b>.sort()</b>	Ordena los elementos del array bajo un criterio de ordenación alfabética.
<b>.sort(func)</b>	Ordena los elementos del array bajo un criterio de ordenación func.

## .reverse()

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango", "Kiwi", "Pera"];  
document.write(frutas, "<br>");  
document.write(frutas.reverse());
```

JS

Banana,Naranja,Manzana,Mango,Kiwi,Pera  
Pera,Kiwi,Mango,Manzana,Naranja,Banana

*Para estos métodos ver ejemplo arrays-orden (.html y .js)*

# Array | Métodos de orden

.sort()

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango", "Kiwi", "Pera"];  
document.write(frutas, "<br>");  
document.write(frutas.sort());
```

JS

Banana,Naranja,Manzana,Mango,Kiwi,Pera  
Banana,Kiwi,Mango,Manzana,Naranja,Pera

# Array | Métodos de orden

## Función de comparación

Como hemos visto, la ordenación que realiza `sort()` por defecto es siempre una ordenación alfabética. Sin embargo, podemos pasarle por parámetro lo que se conoce con los nombres de función de ordenación o función de comparación. Dicha función, lo que hace es establecer otro criterio de ordenación, en lugar del que tiene por defecto:

```
// Función de comparación para ordenación natural
const numeros = [1, 8, 2, 32, 9, 7, 4];

const asc = function (a, b) {
    return a - b;
};
const desc = function (a, b) {
    return b - a;
};
document.write(numeros);
document.write(numeros.sort(asc));
document.write(numeros.sort(desc));
```

JS

1,8,2,32,9,7,4

*Desordenado*

1,2,4,7,8,9,32

*Ascendente*

32,9,8,7,4,2,1

*Descendente*

## Función de comparación (continuación)

### Sintaxis

`array.sort(compareFunction)`

### Valores de parámetros

Parámetro	Descripción
<i>compareFunction</i>	<p>Opcional. Una función que define un orden de clasificación alternativo. La función debe devolver un valor negativo, cero o positivo, según los argumentos, como:</p> <pre>function(a, b){return a-b}</pre> <p>Cuando el método <code>sort ()</code> compara dos valores, envía los valores a la función de comparación y ordena los valores de acuerdo con el valor devuelto (negativo, cero, positivo).</p> <p><b>Ejemplo:</b></p> <p>Al comparar 40 y 100, el método <code>sort ()</code> llama a la función de comparación (40,100). La función calcula 40-100 y devuelve -60 (un valor negativo). La función de ordenación clasificará 40 como un valor inferior a 100.</p>



# Array | Más métodos (funciones)

Método	Descripción
<b>.forEach(cb, arg)</b>	Realiza la operación definida en cb por cada elemento del array.
<b>.every(cb, arg)</b>	Comprueba si todos los elementos del array cumplen la condición de cb.
<b>.some(cb, arg)</b>	Comprueba si al menos un elem. del array cumple la condición de cb.
<b>.map(cb, arg)</b>	Construye un array con lo que devuelve cb por cada elemento del array.
<b>.filter(cb, arg)</b>	Construye un array con los elementos que cumplen el filtro de cb.
<b>.findIndex(cb, arg)</b>	Devuelve la posición del elemento que cumple la condición de cb.
<b>.find(cb, arg)</b>	Devuelve el elemento que cumple la condición de cb.
<b>.reduce(cb, arg)</b>	Ejecuta cb con cada elemento (de izq a der), acumulando el resultado.
<b>.reduceRight(cb, arg)</b>	Idem al anterior, pero en orden de derecha a izquierda.

**Más información:** <https://lenguajejs.com/javascript/caracteristicas/array-functions/>

*Para estos métodos ver ejemplo arrays-metodos2 (.html y .js)*

# Array | Más métodos | forEach y every

```
var frutas = ["Banana", "Naranja", "Manzana", "Mango"];
```

JS

## .forEach(cb, arg)

```
frutas.forEach(mostrar);  
function mostrar(elemento, indice) {  
    document.write(indice + ": " + elemento + "<br>");  
    indice++;  
}
```

JS

FOR EACH  
0: Banana  
1: Naranja  
2: Manzana  
3: Mango

Recorremos el vector y por cada (for each) iteración llamamos a la función **mostrar**, que escribirá en el HTML el índice y el elemento guardado en el array, incrementando el índice en 1

## .every(cb, arg)

```
var edades = [32, 33, 16, 40];  
var edades2 = [32, 33, 20, 40];  
document.write(edades.every(compruebaEdad)); //false  
document.write(edades2.every(compruebaEdad)); //true  
function compruebaEdad(edad) {  
    return edad >= 18;  
}
```

JS

EVERY  
false  
true

Para ambos vectores nos fijamos si **todos (every)** los elementos cumplen con la condición establecida en la función.

**Primer vector:** no cumple el número 16.  
**Segundo vector:** cumplen todos.



# Array | Más métodos | some y map

## .some(cb, arg)

```
var nombres = ["Juan", "Mateo", "Camilo", "Lucas"];
var nombres2 = ["Juan", "Ana", "Luisa", "Mateo", "Camilo"];
document.write(nombres.some(compruebaNombre)); //true
document.write(nombres2.some(compruebaNombre)); //false
function compruebaNombre(nombre) {
    return nombre == "Lucas";
}
```

JS

SOME  
true  
false

En el primer array encontramos algún "Lucas", pero en el segundo array no. El método **some** necesita que alguno de los valores coincida con el valor devuelto por la función **compruebaNombre**

## .map(cb, arg)

```
var numeros = [4, 9, 16, 25];
document.write(numeros.map(raizCuadrada));
document.write("<br>");
function raizCuadrada(numero) {
    return Math.sqrt(numero);
}
```

JS

MAP  
2,3,4,5

A partir del array de números, **map** creará un nuevo array aplicando la función **raizCuadrada** y colocando lo que devuelve la función en el nuevo vector.

# Array | Más métodos | filter y findIndex

## .filter(cb, arg)

```
var personas = ["Ana", "Pablo", "Pedro", "Paola", "Horacio"];
document.write(personas.filter(personasComiezaEnP));
function personasComiezaEnP(persona) {
    return persona[0] == "P";
}
```

JS

FILTER  
Pablo, Pedro, Paola

A partir del array de personas, **filter** creará un nuevo array mostrando (filtrando) solamente aquellos que cumplan con la condición de que la primer letra del cada elemento [0] sea "P", es decir, aquellos cuyo nombre comienza con P.

## .findIndex(cb, arg)

```
var edades3 = [30, 19, 10, 28];
document.write(edades3.findIndex(compruebaMenorEdad));
function compruebaMenorEdad(edad) {
    if (edad < 18) {
        return edad;
    }
}
```

JS

FIND INDEX  
2

A partir del array de edades, **findIndex** buscará la posición del valor que cumple con la condición dada en la función (que la edad sea menor a 18 años)

# Array | Más métodos | find y reduce

## .find(cb, arg)

```
var edades4 = [5, 30, 19, 10, 28];
document.write(edades4.find(compruebaMenorEdad));
function compruebaMenorEdad(edad) {
    if (edad < 18) {
        return edad;
    }
}
```

JS

FIND  
5

A partir del array de edades, **find** buscará el valor que cumple con la condición dada en la función (que la edad sea menor a 18 años)

## .reduce(cb, arg) y .reduceRight(cb, arg)

```
var precios = [110, 10, 25, 50, 15];
document.write(precios.reduce(restaPrecios));
document.write(precios.reduceRight(restaPrecios));
function restaPrecios(total, p) {
    return total - p;
}
```

JS

REDUCE y REDUCE RIGHT  
10  
-180

A partir del array de precios, **reduce** toma el primer elemento y acumula el resultado de izquierda a derecha, en este caso una resta desde el primer valor. Por su parte, **reduceRight** hace lo propio, pero de derecha a izquierda.

# Plantilla de cadena de caracteres (template string)

Las **Template Strings** utilizan las comillas invertidas o backticks para delimitar sus contenidos, en vez de las tradicionales comillas simples o dobles de las cadenas de texto normales.

Las principales funcionalidades que aportan las Template Strings son:

- Interpolación de cadenas.
- Posibilidad de incluir (y evaluar) expresiones dentro de cadenas.
- Definición de cadenas de texto en varias líneas sin tener que usar hacks.
- Formatear cadenas de manera avanzada.
- Cadenas etiquetadas.

```
// esto es una Template String
var saludo = `¡Hola Mundo!`;
// esto es una cadena normal con comillas simples
var saludo = '¡Hola Mundo!';
// esto es una cadena normal con comillas dobles
var saludo = "¡Hola Mundo!";
```

JS

Para colocar  
**backticks**  
(comillas hacia  
atrás) utilizamos  
**ALT + 96**

# Plantilla de cadena de caracteres (template string)

## Interpolación de cadenas

Una de las mejores características de las **Template Strings** es la interpolación de cadenas. La interpolación permite utilizar cualquier expresión válida de JavaScript (como por ejemplo la suma de dos variables) dentro de una cadena y obtener como resultado la cadena completa con la expresión evaluada.

Las partes variables de una *Template String* se denominan *placeholders* y utilizan la sintaxis **`${ }`** para diferenciarse del resto de la cadena. Ejemplo:

```
// Sustitución simple de cadenas  
var nombre = "Juan";  
console.log(`¡Hola ${nombre}!`);  
// resultado => "¡Hola Juan!"
```

JS

# Plantilla de cadena de caracteres (template string)

Como dentro de las partes variables de la cadena se puede incluir cualquier expresión válida de JavaScript, en la práctica sirven para mucho más que mostrar el contenido de una variable. En los siguientes ejemplos se muestran cómo interpolar algunas operaciones matemáticas sencillas:

```
var a = 10;
var b = 10;
console.log(`¡JavaScript se publicó hace ${a+b} años!`);
// resultado => ¡JavaScript se publicó hace 20 años!

console.log(`Existen ${2 * (a + b)} frameworks JavaScript y no ${10 * (a + b)}.`);
// resultado => Existen 40 frameworks JavaScript y no 200.
```

JS

# Plantilla de cadena de caracteres (template string)

Dentro de un valor interpolado también se puede utilizar cualquier función:

```
function fn() { return "Este es el resultado de la función"; }  
console.log(`Hola "${fn()}" Mundo`);  
// resultado => Hola "Este es el resultado de la función" Mundo
```

JS

La sintaxis `${}` también funciona con expresiones que invocan métodos y acceden a propiedades:

```
var usuario = { nombre: 'Juan Perez' };  
console.log(`Estás conectado como ${usuario.nombre.toUpperCase()}.`);  
// resultado => "Estás conectado como JUAN PEREZ."  
  
var divisa = 'Euro';  
console.log(`Los precios se indican en ${divisa}. Utiliza nuestro conversor  
para convertir ${divisa} en tu moneda local.`);  
// resultado => Los precios se indican en Euro. Utiliza nuestro conversor pa  
ra convertir Euro en tu moneda local.
```

JS

# Plantilla de cadena de caracteres (template string)

La ventaja de usar *template strings* es el uso de expresiones incrustadas y la posibilidad de interpolación de cadenas de texto con ellas, facilitando la concatenación de valores. Ejemplo:

```
function suma(a,b){  
    return a+b;  
}  
  
var a=Number(prompt("ingrese un numero a:"));  
var b=Number(prompt("ingrese un numero b:"));  
console.log("la suma entre " + a + " y " + b + " es: " + suma(a,b)); // la suma entre 12 y 21 es: 33  
console.log(`la suma entre ${a} y ${b} es: ${suma(a,b)}`);           // la suma entre 12 y 21 es: 33
```

También podremos escribir una cadena en varias líneas, sin la necesidad de usar `\` o `+` para concatenar:

```
var cadena = `Línea número 1 de la cadena  
Línea número 2 de la cadena`;  
console.log(cadena);
```

JS

```
Línea número 1 de la cadena  
Línea número 2 de la cadena
```

**Fuente (para ampliar):** Clic [aquí](#)

Ver archivos *template-string* y *template-string2* (.html y .js)





## Ejemplo de uso de template string:

- Agregar JavaScript a un sitio, y con **template string** modificar el header y footer del HTML por Javascript.

[Ver archivos de la carpeta ej-template-string](#)

## Ejercicios

- Del archivo **“Actividad Práctica - JavaScript Unidad 2”** están en condiciones de hacer los ejercicios: 19 al 29. Los ejercicios **NO** son obligatorios.