

# JavaScript desde cero

Módulo 5

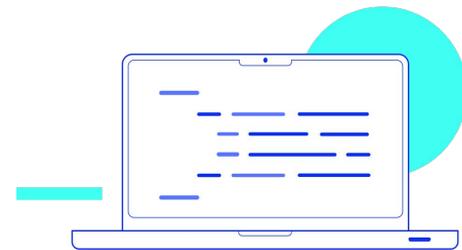
# Template String + Literals

## Introducción al uso de *template String + Literals*

Hasta el momento, todo armado de código en el que tenemos que intercalar contenido estático con contenido dinámico (generado en JS), requería que se implemente el manejo del operador `+` para concatenar “textos” + variables.

Esta tarea no es compleja cuando se trabaja con bloques de textos simples, pero al trabajar con estructuras complejas como una *card* HTML - donde se debe escribir un título, referenciar la ruta de su imagen, agregar una descripción, y seguramente manipular el atributo **ID** de un botón-, la concatenación de todo este bloque de texto HTML con JS, se complejiza.

Allí es donde el ***template String + Literals*** ayuda a **simplificar** muy bien esta tarea. Veamos cómo aplicarlo.



## Arrancamos con un ejemplo simple:

Tenemos un *tag* HTML `<u1>`, que debemos llenar con elementos que provienen desde JS, almacenados en un *array*.

Nos enlazamos al *tag* `<u1>` desde JS, e iteramos el *array* `países` para crear los *tags* `<li>` dinámicos.

```
...
<script defer src="js/main.js"></scrip>
</head>
<body>
  <u1>
    <!-- los tags <li> serán dinámicos -->
  </u1>
</body>
```

```
main.js
const países = ["Argentina", "Uruguay", "Chile", "Venezuela", "Bolivia"]
```



Nos enlazamos con el elemento `<ul>` y en primera instancia, vaciamos su contenido.

Luego, utilizando la cláusula **for...of** iteramos el *array* `países`, e intercalamos HTML y la variable JS que contiene el nombre de cada elemento del *array*.

### Ahora los realizamos, pero integrando *Template String + Literals*.

En este escenario, tal vez no se note la diferencia, pero en bloques HTML más complejos, sí sacaremos grandes ventajas con esta funcionalidad.

```
main.js
const lista = document.querySelector("ul")
lista.innerHTML = "" //vaciamos el tag <ul>

for (pais of países) {
  lista.innerHTML += "<li>" + pais + "</li>"
}
```

```
main.js
const lista = document.querySelector("ul")
lista.innerHTML = "" //vaciamos el tag <ul>

for (pais of países) {
  lista.innerHTML += `<li>${pais}</li>`
}
```

Analicemos cada parte que conforma esta estructura moderna para interactuar entre HTML y JS:

```
main.js
for (pais of paises) {
  lista.innerHTML += `- ${pais}</li>`
}

```

**Template Literals:** se denomina a todo elemento que corresponde a JS, y que aporta valores dinámicos, generados desde este lenguaje: variables, constantes, propiedades de un objeto literal, retorno de funciones o métodos, y otros.

**Template String:** se denomina así a todo el contenido HTML encerrado entre los caracteres *backtick* (```). Agrupa contenido HTML, textos, atributos y valores, y otros.

**Carácter *backtick*** ```, también conocido como *acento grave* o *acento europeo*.

# Uso de template String + Literals en estructuras complejas

Ahora nos encontramos con **una card HTML mucho más compleja**, que **representa la información de un producto electrónico**. Además, cuenta con un botón que permite agregar el producto al carrito de compras.

A su vez, la *card* HTML utiliza comillas dobles para encerrar las clases CSS de sus elementos, y texto descriptivo que posee comillas simples.

```
<div class="card card-product">
  <h1>Teclado Bluetooth Logi</h1>
  
  <p>El teclado bluetooth Logi es un portátil y de amplia duración.
    Su batería soporta 'hasta 6 meses de energía continua' con un
    uso estimado de 8 horas diarias.
  </p>
  <button class=button button-cart" id="12345">AGREGAR</button>
</div>
```

Ante este escenario, la antigua forma de concatenar contenido JS con HTML, se torna más que compleja. Allí es donde *Template String + Literals* aporta su valor agregado.

Aquí tenemos un *array* de objetos con mucha información en cada uno de ellos.

Si pensamos en cómo se puede **iterar para armar *cards* HTML dinámicas**, lo más apropiado será **crear una función JS con retorno y, en su interior, definir la *card* HTML en formato *Template String***.

Esta misma función JS, recibe como parámetro un objeto literal del *array*, y **utiliza *Template Literals* para intercalar en el HTML, los datos dinámicos** que provienen de JS.

```
// array de objetos literales con muchos productos
const productosElectronicos = [{...}, {...}, {...}, {...}]

// representación aislada de un producto de
// los que conforman el array de más arriba.
const producto = {
  id: 12345,
  titulo: "Teclado Bluetooth Logi",
  descripcion: "El teclado bluetooth Logi
                es un portátil y de amplia
                duración. Su batería soporta
                'hasta 6 meses de energía
                continua' con un uso
                estimado de 8 hs diarias",
  imagen: "images/teclado-bt-logi.jpg"
}
```

1. Nos enlazamos al **<div> contenedor** donde se cargarán las *cards* HTML.
2. **Iteramos el *array*** de productos y, cada producto, se pasa como parámetro a la función mencionada.

```
const productosElectronicos = [{...}, {...}, {...}, {...}]
const divContenedor = document.querySelector("div.container")

divContenedor.innerHTML = "" // vaciamos el contenedor HTML

for (producto of productosElectronicos) {
  divContenedor.innerHTML += retornarHTMLDeProducto(producto)
}
```

3. La función JS utiliza el objeto literal que llega como parámetro y, **mediante *Template Literals***, encaja cada propiedad de este **dentro del HTML**, retornando un HTML listo para representar en pantalla.

```
function retornarHTMLDeProducto(producto) {

  return `<div class="card card-product">
    <h1>${producto.titulo}</h1>
    
    <p>${producto.descripcion}</p>
    <button class=button button-cart
      id="${producto.id}">
      AGREGAR
    </button>
  </div>`
}
```

```
function retornarHTMLDeProducto(producto) {  
  return `<div class="card card-product">  
    <h1>${producto.titulo}</h1>  
      
    <p>${producto.descripcion}</p>  
    <button class="button button-cart"  
      id="${producto.id}">  
      AGREGAR  
    </button>  
  </div>`  
}
```

**producto.titulo** se usa para representar, tanto el nombre del producto como también el atributo alt de la imagen.

Se aprovecha el código del producto, **producto.id** para generar el atributo id de `<button>`.

Esto será útil para detectar con Eventos JS, qué botón presionó el usuario y saber cuál es producto de su interés.

## Ventajas obtenidas con *Template String + Literals*:

- Reutilización de propiedades/valores.
- **Simplificación** de código complejo, evitando intercalar + / “ / ‘, y datos JS dinámicos.
- Generación de **atributos HTML id dinámicos**.



## Consideraciones

Es importante comprender que, **en todo desarrollo de aplicaciones Web, los datos** que representan, por ejemplo, a productos en un *E-commerce*, **proviene de una aplicación backend + base de datos.**

**Esos datos siempre serán dinámicos y manejados por JS**, por lo tanto, el HTML que los muestre debe pensarse como HTML dinámico. Y la herramienta que se necesita es *Template String + Literals*.

Además, **aplicar *Template String + Literals* ayudará a comprender el fundamento de componentes** utilizado en la **librería React**, o en los *frameworks JS* como ser: **Angular, Vue, Svelte y Astro**, entre otros.



¡Sigamos  
trabajando!