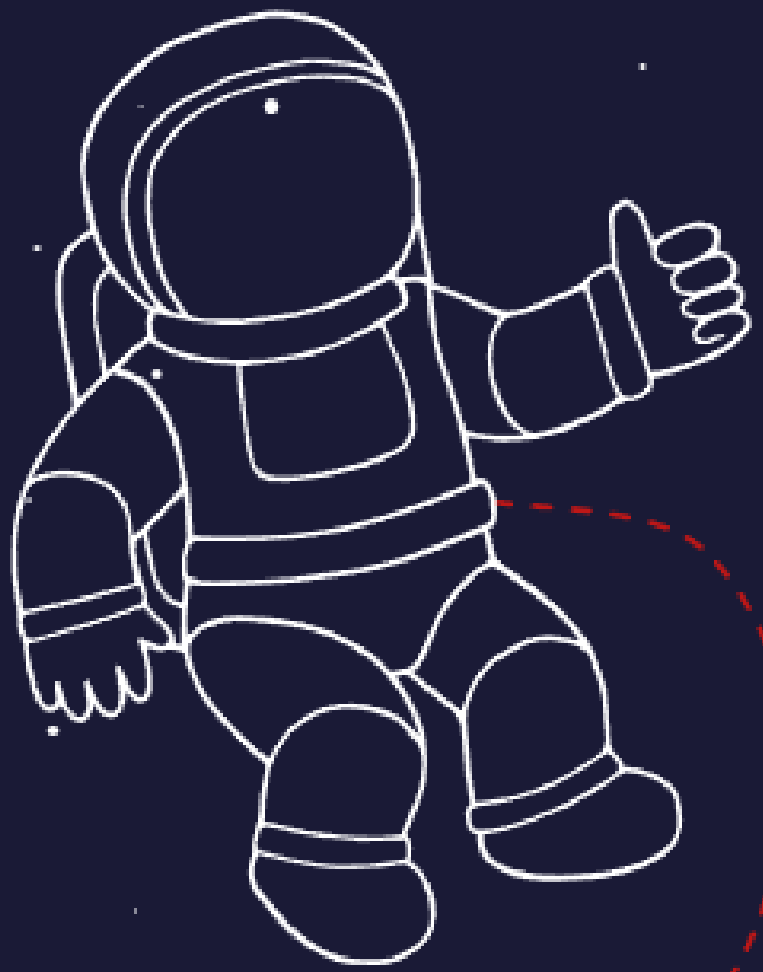


Programa académico CAMPUS

Trainer

Ing. Carlos H. Rueda C.

JAVASCRIPT

DOCUMENT OBJECT MODEL

DOM

DOM

El DOM (Document Object Model) es una interfaz de programación para documentos web. Representa la página para que los programas puedan cambiar la estructura, el estilo y el contenido del documento. El DOM representa el documento como nodos y objetos

DOM

JavaScript se usa a menudo para manipular el DOM. Usando JavaScript, puede ***crear páginas HTML dinámicas*** donde los elementos reaccionan a las acciones del usuario sin requerir una recarga de página.

DOM

También puede crear páginas HTML dinámicas que responden a las acciones del usuario sin requerir una recarga de página.

DOM - PADRES E HIJOS

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>

</body>
</html>
```

DOM - PADRES E HIJOS

```
<body>
    <h1>JS-DOM</h1>
    <div>
        <h2>Bienvenidos</h2>
        <span>Aprendamos DOM con JavaScript</span>
    </div>
</body>
```

DOM - ARBOL

DOM - NODOS Y ELEMENTOS

SELECCIONAR ELEMENTOS EN EL DOCUMENTO

Para seleccionar un elemento:

- **getElementById()**
- **getElementsByTagName()**
- **querySelector()**
- **querySelectorAll()**

MÉTODO getElementById()

En HTML, los `ids` se utilizan como identificadores únicos para los elementos HTML.

EJEMPLO getElementById()

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = "Hello World!";
</script>
```

getElementById().textContent

Si deseamos leer solamente el contenido del párrafo, entonces podemos usar la propiedad `textContent`.

```
const paragraph1 = document.getElementById("para1");
console.log(paragraph1.textContent);
```

METODO getElementsByTagName()

Busca el elemento por el nombre de la etiqueta, devuelve una lista de dichos elementos.

EJEMPLO getElementsByTagName()

```
<h2>JavaScript HTML DOM</h2>
<p>Encontrando un elemento por el nombre.</p>
<p>Este ejemplo demuestra el uso de
<b>getElementsByTagName</b>.</p>
<p id="demo"></p>

<script>
const element = document.getElementsByTagName("p");
document.getElementById("demo").innerHTML =
    'Primer Párrafo (index 0): ' + element[0].innerHTML;
</script>
```

EJEMPLO querySelector()

Si lo usa con una etiqueta HTML, devuelve la primera.

METODO querySelector()

```
<h1>Equipos famosos</h1>
<ul class="list">
    <li>Junior</li>
    <li>Bucaramanga</li>
    <li>Huila</li>
    <li>Cúcuta</li>
</ul>
```

EJEMPLO querySelector()

Si quisiera apuntar a `class="list"` para imprimir la lista no ordenada en la consola, entonces usaría .list dentro del querySelector().

METODO querySelector()

```
<script>
const list = document.querySelector(".list");
console.log(list);
</script>
```

METODO querySelectorAll()

devuelve una lista de todos los elementos que coinciden con un selector especificado.
Este método se utiliza para seleccionar todos los elementos que tienen el nombre de etiqueta especificado en el documento HTML.

METODO querySelectorAll()

Por ejemplo, si desea seleccionar todos los elementos `<p>` en el documento HTML, puede usar `querySelectorAll('p')`.

EJEMPLO querySelectorALL()

```
const lstElementos = document.querySelectorAll("ul > li");

lstElementos.forEach(elem => {
  console.log(elem);
  });
```

RECORDANDO Buscar elementos HTML

```
<p class="parrafo" id="mi-id">Hola este es mi párrafo</p>
<p class="parrafo" id="mi-id-dos">Hola este es mi párrafo</p>

<script>
  console.log(document.head);
  console.log(document.body);
  console.log(document.getElementById("mi-id"));
  console.log(document.querySelector("#mi-id"));
  console.log(document.querySelector(".mi-id"));
  console.log(document.querySelector("p"));
  console.log(document.querySelectorAll("p"));
</script>
```

MODIFICACIÓN de Elementos

```
<p class="parrafo" id="mi-id">Hola este es mi párrafo</p>

<script>
  const parrafo = document.querySelector(".parrafo");
  parrafo.textContent = "Parrafo dinámico";
</script>
```

MODIFICACIÓN de Elementos

```
// Nos devuelve la clase del párrafo
console.log(parrafo.className);

// Agregamos una clase adicional
parrafo.classList.add("clase-adicional");
console.log(parrafo);
```

CREACIÓN de Elementos

```
<ul id="lista-dinamica"></ul>
<script>
  // elemento donde vamos a incorporar los <li>
  const lista = document.getElementById("lista-dinamica");

  // Creamos el <li>
  const li = document.createElement("li");

  // Agregamos texto al <li>
  li.textContent = "Mi <li> dinámico";

  // Finalmente incorporamos al <ul>
  lista.appendChild(li);
</script>
```

FRAGMENT de Elementos

Se utiliza como una versión ligera de Document que almacena un segmento de una estructura de documento compuesta de nodos como un documento estándar.

FRAGMENT de Elementos

Por ende en un fragment vamos a guardar todo un template o nodos HTML que luego pintaremos en nuestro DOM, así evitamos en mayor parte el Reflow.

FRAGMENT de Elementos

```
<script>
  const lista = document.getElementById("lista-dinamica");
  const arrayItem = ["item 1", "item 2", "item 3"];
  const fragment = document.createDocumentFragment();
  arrayItem.forEach((item) => {
    const li = document.createElement("li");
    li.textContent = item;
    fragment.appendChild(li);
  });
  lista.appendChild(fragment);
</script>
```

TEMPLATE HTML

Se necesita incorporar de forma dinámica este elemento:

```
<li class="list" id="lista-dinamica">
  <b>Nombre: </b> <span class="text-danger">descripción...</span>
</li>
```

TEMPLATE HTML

Usando **Fragment** y **createElement** se escribiria el siguiente código:

```
<ul id="lista-dinamica"></ul>

<script>
  const lista = document.getElementById("lista-dinamica");
  const arrayItem = ["item 1", "item 2", "item 3"];
  const fragment = document.createDocumentFragment();
```

TEMPLATE HTML

```
arrayItem.forEach((item) => {
  // creamos li
  const li = document.createElement("li");
  // agregamos clase a li
  li.classList.add("list");
  // creamos b
  const b = document.createElement("b");
  // agregamos texto a b
  b.textContent = "Nombre: ";
```

TEMPLATE HTML

```
// creamos span
const span = document.createElement("span");
// agregamos clase a span
span.classList.add("text-danger");
// agremos texto a span
span.textContent = item;
```

TEMPLATE HTML

```
    // agregamos nodo hijos a li
    li.appendChild(b);
    li.appendChild(span);
    // agregamos li al fragmente
    fragment.appendChild(li);
  });

  lista.appendChild(fragment);
</script>
```

TEMPLATE HTML

Ahora si la estructura fuera muy compleja 😢

¿Cuál es la solución?

El elemento HTML `<template>` es un mecanismo para mantener el contenido HTML del lado del cliente que no se renderiza cuando se carga una página, pero que posteriormente puede ser instanciado durante el tiempo de ejecución empleando JavaScript.

¿Cuál es la solución?

La mejor solución sería utilizar `template` y `fragment`

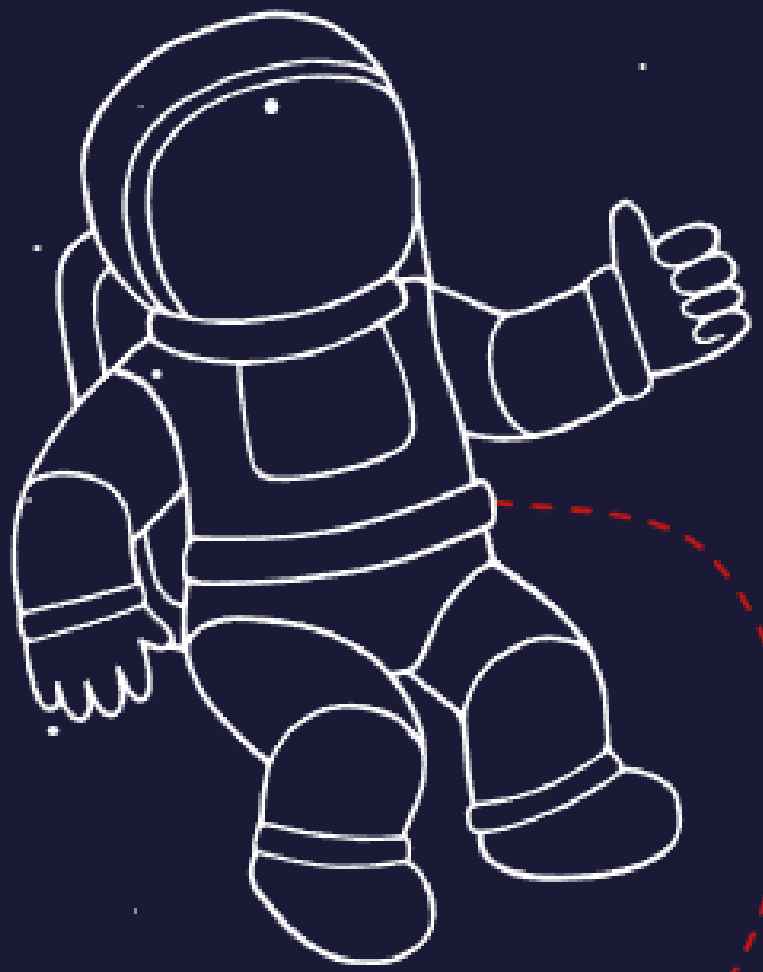
```
<ul id="lista-dinamica"></ul>
<template>
  <li class="list">
    <b>nombre: </b> <span class="text-danger">descripción...</span>
  </li>
</template>
```

```
<script>
  const lista = document.getElementById("lista-dinamica");
  const arrayItem = ["item 1", "item 2", "item 3"];

  const fragment = document.createDocumentFragment();
  const template = document.querySelector("template").content;
```

```
  arrayItem.forEach((item) => {
    template.querySelector("span").textContent = item;
    const clone = template.cloneNode(true);
    // const clone = document.importNode(template, true);
    fragment.appendChild(clone);
  });

  lista.appendChild(fragment);
</script>
```

TEMPLATE HTML

Ahora todos felices 😃



Programa académico CAMPUS

Trainer
Ing. Carlos H. Rueda C.