Guía de ejercicios 5 - Métodos de los arreglos

 ¡Hola! Te damos la bienvenida a esta nueva guía de ejercicios.

¿En qué consiste esta guía?

En JavaScript la mayoría de los tipos de datos que utilizamos para asignar valores a nuestras variables poseen una serie de métodos de forma intrínseca que podemos ocupar para realizar diferentes tareas a partir del dato o valor al que pertenece.

Los arreglos poseen una gran variedad de métodos, y entre sus diferentes utilidades nos ofrecen la posibilidad de agregar, eliminar y modificar los elementos almacenados. En esta guía estudiaremos algunos de los más utilizados y cómo podemos utilizarlos para crear interacciones con el DOM.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Agregando elementos a un arreglo

El método push

El método push se utiliza para agregar elementos en un arreglo, en donde el argumento que asignemos será este nuevo dato almacenado justo al final del arreglo.

```
const arreglo1 = [1,2,3,4,5]
arreglo1.push('hola')
console.log(arreglo1) /* Array(6) [1,2,3,4,5,'hola'] */
```

Podemos ver que el número 6 fue agregado al final.

Importante: Podemos ver que `arreglo1` fue modificado aunque fue declarado con const y aun así fue modificado, esto se debe a que const no quiere decir que no pueda cambiar, solo quiere decir que no podemos asignarle un nuevo valor.

```
const prueba = [1,2,3,4]
prueba = [5,6,7,8] /* Esto dará error por que le estamos asignando un
nuevo valor */
prueba[0]  = 5 /* Esto funciona porque solo estamos cambiando algo */
```

Cuando un objeto puede cambiar se dice que es **mutable**.

El método unshift

Podemos agregar elementos al principio de un arreglo con `.unshift`

```
const arreglo2 = [1,2,3,4,5]
arreglo2.unshift('hola')
console.log(arreglo2) /* Array(6) ['hola', 1,2,3,4,5] */
```

El método splice

El método splice es muy flexible y sirve para añadir, remover y reemplazar elementos en un arreglo, para esto utilizar `arreglo.splice(index, 0, valor)` donde index es la posición

donde añadiremos el dato, 0 indica que no borraremos dato y valor es lo que agregaremos al arreglo

```
const arreglo3 = [1,2,3,4,5]
arreglo3.splice(2, 0, 'hola')
console.log(arreglo3) /* Array(6) [1,2,'hola',3,4,5] */
```

Actividad 1: Agregando elementos con push

Crea el archivo 'superheroes.js' donde a partir del siguiente arreglo deberás:

```
let superHeroes= [
  "Ironman",
  "Superman",
  "Hawkeye"
]
```

1. Agregar el nombre de un superhéroe al final del arreglo usando el método push
2. Agregar el nombre de un superhéroe al principio del arreglo usando el método unshift
3. Agregar el nombre de un tercer superhéroe en la mitad del arreglo usando el método splice
4. Mostrar el arreglo utilizando `console.log`

Eliminando elementos de un arreglo

Hay 3 formas frecuentes de eliminar un elemento de un arreglo.

El método shift

Con este método podemos borrar el primer elemento de un arreglo.

```
const arreglo4 = [1,2,3,4,5]
arreglo4.shift()
console.log(arreglo4) /* Array(4) [2,3,4,5] */
```

El método pop

Con este método podemos borrar el último elemento de un arreglo.

```
const arreglo5 = [1,2,3,4,5]
arreglo5.pop()
console.log(arreglo5) /* Array(4) [1,2,3,4] */
```

El método splice

Con el método splice también podemos borrar un elemento en cualquier posición si lo utilizamos con los siguientes argumentos `arreglo.splice(index, 1)` donde 1 indica que se borrará solo 1 elemento.

```
const arreglo6 = [1,2,3,4,5]
arreglo6.splice(2, 1)
console.log(arreglo6) /* Array(4) [1,2,4,5] */
```

Actividad 2: Agregando y borrando elementos

Crear el archivo inAndOut.js con el siguiente arreglo.

```
const usuarios = ["Erick", "Cristian", "Max", "Claudia"];
```

1. Utilizando pop sacar el último elemento
2. Agregar a María José al principio
3. Remueve a Cristian utilizando splice

Membrecía

Una membrecía es simplemente saber si un elemento está dentro de un arreglo o no, para esto ocuparemos el método `.includes()`

```
const usuarios = ["Erick", "Cristian", "Max", "Claudia"];
usuarios.includes("Erick") /* true */
```

Usualmente se ocupan en conjunto con un `if`

```
if(usuarios.includes("Erick")){
  /* Hey !!! Erick está en el club */
}
```

Contar elementos

Podemos contar los elementos de un arreglo con `arreglo.length`

```
const usuarios = ["Erick", "Cristian", "Max", "Claudia"];
console.log(usuarios.length) /* 4 */
```

Buscando el índice de un elemento

El método findIndex

Es posible que no sepamos cuál es el índice del elemento que deseemos eliminar o modificar, por ejemplo queremos eliminar a un usuario por su nombre (u otro atributo), entonces dentro de un input el usuario ingresará esa información, lo que tendremos que hacer es buscar en el arreglo a qué índice corresponde y luego con esa información podremos borrarlo. Para buscar el índice de un elemento utilizaremos .findIndex

```
arreglo.findIndex((elemento) => elemento == valor)
```

Por ejemplo, dado el siguiente arreglo de nombres:

```
const apellidos = ["Aniston", "Cox", "Buffay", "Perry", "LeBlanc", "Schwimmer"]
```

Se necesita saber cuál es el índice del apellido Perry, entonces podemos hacer lo siguiente:

```
const indiceDePerry = apellidos.findIndex( apellido => apellido === "Perry" )
```

Y si mostramos esta variable en consola, veremos que será el número 3, que representa el índice del apellido Perry.

Ahora podemos combinar el uso del método findIndex con el método splice, para eliminar un elemento de un arreglo que cumpla una expresión.

Si todavía te cuesta escribir arrow functions puedes escribir la función así:

```
const otraFormaDeBuscar = apellidos.findIndex(function(apellido) { return
apellido === "Perry"} )
```

El método findIndex en un arreglo con objetos

Por ejemplo, dado el siguiente arreglo de objetos:

```
let actores = [
  { id: 431, nombre: "Jhonny Depp" },
  { id: 124, nombre: "Brad Pitt" },
  { id: 541, nombre: "Leonardo DiCaprio" },
  { id: 664, nombre: "Morgan Freeman" }
];
```

Se necesita eliminar el actor de id 541, entonces podemos hacer lo siguiente:

```
const indiceDelActorAEliminar = actores.findIndex( actor => actor.id == 541)
actores.splice(indiceDelActorAEliminar, 1)
```

Y ahora si revisamos el arreglo, podremos ver que Leonardo DiCaprio ya no está dentro del arreglo actores.

Imagen 1. El método splice y findIndex
Fuente: Desafío Latam

Actividad 3: Encontrando y eliminando una película

Realicemos el siguiente ejercicio en donde debamos eliminar la película Terminator del siguiente arreglo

```
const peliculas = [
  {id: 1, nombre: "Thor"},
  {id: 2, nombre: "Ant-Man"},
  {id: 3, nombre: "Terminator"},
  {id: 4, nombre: "Ip Man"},
  {id: 5, nombre: "Rocky"},
]
```

1. Utiliza el findIndex para encontrar el índice de la película Terminator
2. Utiliza el splice y el índice encontrado para eliminar la película del arreglo
3. Muestra por consola el arreglo de películas para confirmar que Terminator fue eliminada.

Iterando arreglos

El método foreach

`.forEach` es otra forma de iterar un arreglo parecida a la instrucción for

```
const estaciones = ["Verano", "Otoño", "Invierno", "Primavera"];
estaciones.forEach(x => console.log(x)) /* con arrow functions */
estaciones.forEach(function(x) {console.log(x)}) /* con función anónima
*/
```

Esto se lee como por cada elemento del arreglo de estaciones haz un `console.log` de ese elemento.

`.forEach` recibe una función como argumento y aplica esa función a cada elemento de la lista, esa función puede ser algo sencillo como un `console.log` o un alert o volver a modificar el DOM.

Si creamos una página web nueva llamada `foreach.html` con el autocompletado de HTML y con el siguiente script al final del body, veremos en pantalla el doble de cada número.

```
body = document.querySelector("body")
const valores = [200, 100, 500, 300, 250]
valores.forEach(x => body.innerHTML += `<p> ${2* x} </p>`)
```

Esto se lee como por cada valor agrega al html un párrafo mostrando el doble del valor.

Dentro de los template literals además de mostrar variables podemos hacer cálculos o ejecutar código js.

Transformando arreglos

El método `.map` nos permite generar un arreglo nuevo a partir de aplicar una función a cada elemento de un arreglo

```
const valores = [200, 100, 500, 300, 250]
const nuevos_valores = valores.map(x => 2* x)
console.log(nuevos_valores) /* [400, 200, 1000, 600, 500]
```

map no modifica el arreglo original.

¿Cuando usar .forEach y cuando .map?

`.forEach` es útil cuando queremos hacer algo de inmediato con cada dato iterado, por ejemplo mostrarlo en pantalla o agregarlo al html. `.map` cuando queremos transformar los datos para seguir trabajando con ellos.

Filtrando elementos de un arreglo

Con el método `.filter` podemos filtrar los elementos dentro de un arreglo a partir de una condición.

```
const valores = [200, 100, 500, 300, 250]
const valores_filtrados = valores.filter(x => x >= 300)
console.log(valores_filtrados) /* [500, 300] */
```

También podemos ocupar filter con un arreglo de objetos. Por ejemplo:

```
const estudiantes = [
  { nombre: "Juan", nota: 3.4 },
  { nombre: "Laura", nota: 6 },
  { nombre: "Katherine", nota: 4.3 },
  { nombre: "Jonathan", nota: 5.4 }
];
const estudiantesAprobados =
estudiantes.filter( estudiante => estudiante.nota >= 4.5 )
```

```
console.table(estudiantesAprobados)
```

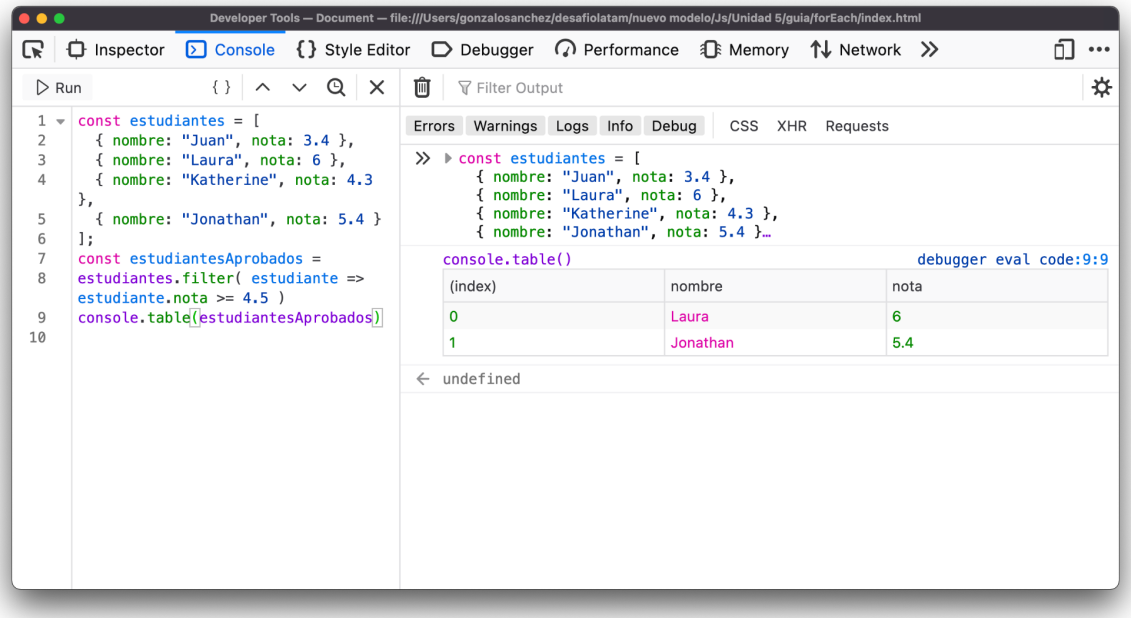

Imagen 2. Filter con un arreglo de objetos.
Fuente: Desafío Latam

Actividad 4: Filtrar y contar

```
let trabajadores = [
   {nombre: "Contanza", cargo: "Chef"},
   {nombre: "Luis", cargo: "garzón"},
   {nombre: "Estefany", cargo: "Administradora"},
   {nombre: "Andrés", cargo: "Recepcionista"},
   {nombre: "Pedro", cargo: "garzón"},
   {nombre: "Felipe", cargo: "Aseo"},
   {nombre: "Maria", cargo: "garzón"},
   {nombre: "Diego", cargo: "garzón"},
]
```

1. Usar el método filter para crear un nuevo arreglo con arreglo de trabajadores creando un nuevo arreglo llamado garzones

2. Utiliza el arreglo resultante del requerimiento anterior, console.log y la propiedad length para mostrar cuántos garzones hay.

Juntar arreglos

Con el método concat podemos unir los elementos de dos arreglos en un arreglo nuevo.

```
const arr1 = [1,2,3]
const arr2 = [4,5,6]
const arr3 = arr1.concat(arr2)
console.log(arr3) /* [1,2,3,4,5,6] */
```

Si hacemos `console.log(arr1)` o `console.log(arr2)` veremos que los arreglos no fueron modificados, sólo se generó un arreglo nuevo producto de la unión y ese lo guardamos en arr3.

También podemos unir arreglos de objetos:

```
const autosCompactos = [
  {marca: 'Toyota', modelo: 'Corolla', combustible: 'Gasolina'},
  {marca: 'Mazda', modelo: '3', combustible: 'Gasolina'},
  {marca: 'Honda', modelo: 'Civic', combustible: 'Gasolina'},
  {marca: 'Bmw', modelo: '116d', combustible: 'Diesel'},
];
const autosDeportivos = [
  {marca: 'Opel', modelo: 'Astra OPC', combustible: 'Gasolina'},
  {marca: 'Renault', modelo: 'Megane RS', combustible: 'Gasolina'},
  {marca: 'Mitsubishi', modelo: 'Lancer Evo', combustible: 'Gasolina'},
];

const autos = autosCompactos.concat(autosDeportivos);
console.log(autos);
```

Ordenar elementos

Podemos ordenar los elementos de un arreglo con el método `.sort`

```
const arr1 = [4,1,2,3]
const ordenado = arr1.sort()
console.log(ordenado) /* 1,2,3,4*/
```

El método sort también puede recibir una función que compare 2 elementos digamos x e y, si la comparación en menor que 0 entonces x es menor que y, si es cero entonces x es igual a y, y si es mayor que cero entonces x es mayor que y.

```
const arr1 = [4,1,2,3]
const ordenado = arr1.sort((x,y) => y - x)
console.log(ordenado) /* 4,3,2,1*/
```

y - x significa que restamos el segundo elemento con el primero, para el primer caso sería el término en el índice cero menos en el índice 1, o sea 1 - 4 = -3, lo que es menor que cero, lo que significa que 4 va primero que 1. Si la función devolviese x - y entonces el resultado sería que los elementos quedarían ordenados de menor a mayor (como funciona por defecto sin necesidad de especificarlo)

Utilizando esto podemos ordenar un arreglo de objetos

```
const estudiantes = [
  { nombre: "Juan", nota: 3.4 },
  { nombre: "Laura", nota: 6 },
  { nombre: "Katherine", nota: 4.3 },
  { nombre: "Jonathan", nota: 5.4 }
];
const estudiantesOrdenado = estudiantes.sort((x,y) => x.nota - y.nota)
console.table(estudiantesOrdenado)
```

Resumen de los métodos

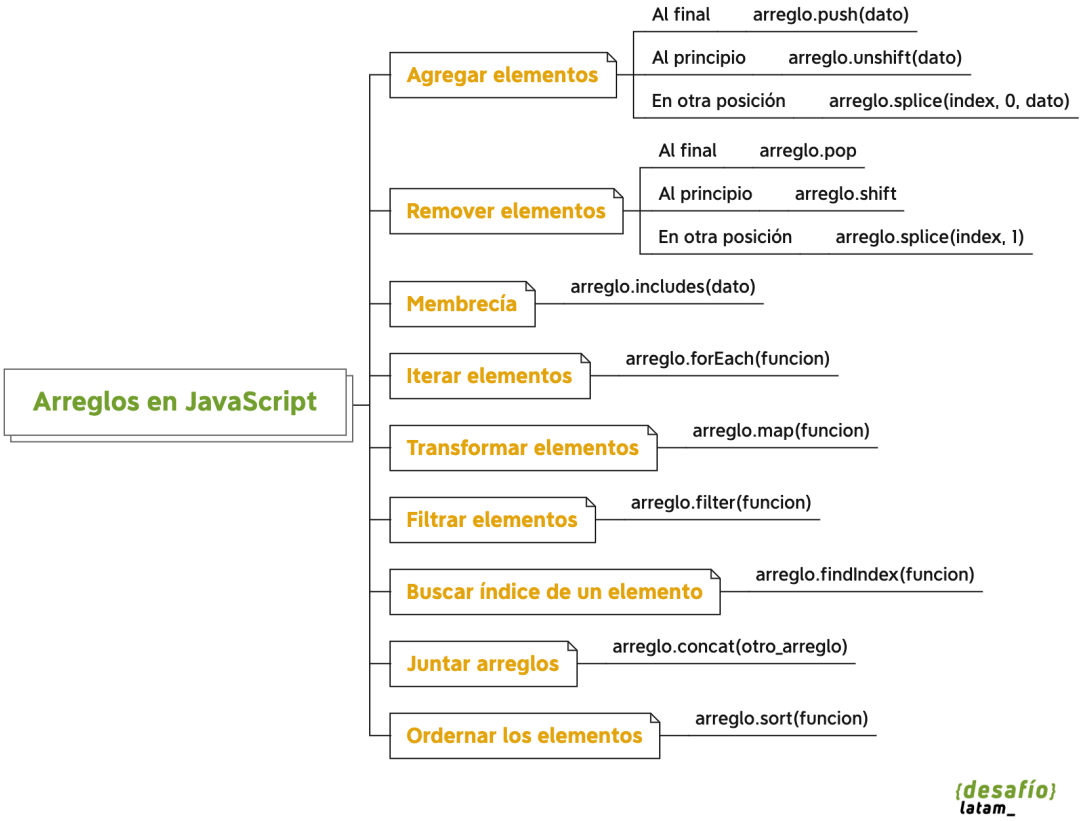

Imagen 3. Resumen de métodos.
Fuente: Desafío Latam

Arreglos y DOM

Creando una interfaz para insertar elementos

Ahora crearemos apps donde agregaremos y borraremos elementos de una página web utilizando como base arreglos.

Para esto, realizaremos un ejercicio paso a paso en donde agregaremos nuevos invitados para una reunión.

1. Ocupemos el siguiente código HTML:

```
<!-- invitados.html -->
```

```
<input id="nuevoInvitado"> <!-- 1 -->
<button id="agregarInvitado">Agregar</button> <!-- 2 -->
<h3>Invitados: </h3>
<ul id="invitados"></ul> <!-- 3 →
<script>
  /* Aquí agregaremos el script más adelante */
</script>
```

Imagen 4 Formulario de invitados
Fuente: Desafío Latam

2. Dentro de la etiqueta script seleccionaremos los elementos que vamos a necesitar.

```
const listaDeInvitados = document.querySelector("#invitados")
const invitadoInput = document.querySelector("#nuevoInvitado")
const btnAgregar = document.querySelector("#agregarInvitado")
```

3. Lo siguiente será será tener crear un arreglo para guardar la información de los invitados y agregar interacción al botón para que al ser presionado, tome el valor del input y lo agregue como un nuevo elemento al arreglo:

```
const invitados = []
btnAgregar.addEventListener("click", () => {
  const nuevoInvitado = invitadoInput.value
  invitados.push(nuevoInvitado)
  invitadoInput.value = "" /* Vaciamos el input */
})
```

4. Agregar el nombre del invitado al arreglo no es suficiente. Luego de agregarlo debemos actualizar la lista de nombre a la página web. Para esto tenemos que:

    - Vaciar el contenido de la lista actual
    - Recorrer el arreglo de invitados
    - En cada iteración, ocupar el innerHTML para agregar el invitado a la lista

5. La función quedaría entonces de la siguiente manera:

```
const invitados = []

 btnAgregar.addEventListener("click", () => {
   /* Agregamos el invitado al arreglo */
   const nombreInvitado = invitadoInput.value
   invitados.push(nombreInvitado)
   invitadoInput.value = ""

   /* Actualizamos la información en el HTML */
   let html = ""
   for (let invitado of invitados) {
     html += `<li>${invitado}</li>`;
   }
   listaDeInvitados.innerHTML = html;
 })
```

6. Y nuestra lista ahora puede agregar nuevos invitados:

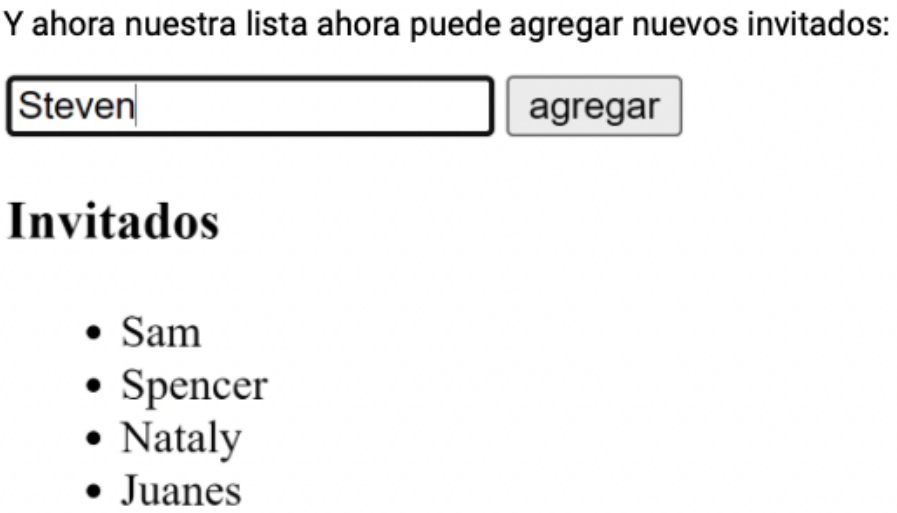

Imagen 5. Lista de invitados
Fuente: Desafío Latam

Actividad 5: Creando una interfaz para ingresar tareas.

Crea el archivo `milistadetaraes.html` donde practicaremos realizando los mismos pasos desde cero, esta vez construyendo una lista de tareas. **A partir de ahora las actividades servirán como base para el desafío.**

- Crear el archivo milistadetaraes.html con la base de HTML.
- Agregar un input para ingresar el nombre de la tarea
- Agregar el botón
- Dejar una sección, div, ul u otro elemento para ingresar las tareas
- Agregar la etiqueta script al final de body
- Obtener los elementos HTML necesarios y guardarlos en variables globales
- Crear un arreglo vacío llamado tareas para guardar las tareas
- Agregar el listener del evento al botón donde se seleccione el texto del input y se guarda dentro del arreglo
- Dentro de la función del listener actualizar el HTML

Actividad 6: Actualizando la cuenta de tareas

- Dentro del html agregar la siguiente etiqueta `<span id="cuenta-tareas"></span>`
- Dentro del script obtener el elemento utilizando querySelector o getElementById y guardarlo en una variable global
- Después de actualizar la lista de tareas en el event listener, actualizar la lista de tareas. Utilizar tareas.length para obtener la cantidad de tareas.

Pensando en objetos

Cuando queremos guardar múltiples elementos, por ejemplo en la lista de invitados queremos guardar la fecha de ingreso a la lista, un compañero, un identificador o cualquier otro elemento, o en la lista de tareas queremos guardar otro atributo asociado como por ejemplo el encargado será más sencillo organizar los elementos en objetos.

Adicionalmente pensar en objetos nos permitirá agregar un identificar que nos hará más sencillo el borrado o modificación de los elementos de la lista, no es la única forma de lograrlo pero es la que utilizaremos en esta guía.

Modificando el programa de invitados para que funcione con objetos

Al guardar un invitado lo guardaremos junto con un id que nos ayudará más adelante a borrar el dato. El identificador lo generaremos automáticamente con una fecha única asociada al momento exacto del ingreso, hay otras formas de generar este identificador pero usualmente estos son entregados por una API o una base de datos al momento de insertar el datos, así que por ahora para generar un valor único utilizaremos la fecha.

```
const invitados = []
btnAgregar.addEventListener("click", () => {
  /* Agregamos el invitado al arreglo */
  const nuevoInvitado = invitadoInput.value
  invitados.push({id: Date.now(), nombre: nuevoInvitado})
  invitadoInput.value = ""

  /* Actualizamos la información en el HTML */
  let html = ""
  for (let invitado of invitados) {
    html += `<li>${invitado.nombre}</li>`;
  }
  listaDeInvitados.innerHTML = html;
})
```

Vemos que la diferencia es pequeña, al momento de insertar los datos los insertamos dentro de un objeto `{id: Date.now(), nombre: nuevoInvitado}` y cuando mostramos la pantalla lo hacemos como: `${invitado.nombre}` en lugar de `${invitado}` puesto que invitado es un objeto y lo que queremos mostrar de ese objeto es el nombre.

Puedes probar quitando el `.nombre` para ver qué sucede.

Actividad 7: Modifica el programa de tareas para que funcione con un arreglo de objetos.

Eliminando elementos desde la interfaz

Volvamos al programa invitados. Para eliminar invitados de la lista tendremos que:

1. Agregar un botón con el texto eliminar en cada invitado
2. Crear una función que busque el elemento en el arreglo, sacarlo del arreglo y actualizar la lista
3. El botón borrar debe llamar a la función

Paso a paso

1. Agregar el botón es sencillo, para esto dentro del código donde actualizamos el HTML agregamos el botón.

```
/* Actualizamos la información en el HTML */
let html = ""
for (let invitado of invitados) {
  html += `<li>${invitado.nombre} <button> x </button> </li>`;
}
listaDeInvitados.innerHTML = html;
```

2. Creamos la función borrar

Recordemos que nuestros invitados están guardados en un arreglo con la siguiente estructura `{id: 1231231312381283, nombre: "Juan"}` y la razón por la que le agregamos un identificador fue para hacer fácil el borrado. Ahora es donde lo ocuparemos. Nuestra función va a recibir este id y a partir de él va a buscar el índice en el arreglo y lo va a remover utilizando splice. Luego simplemente volveremos a actualizar la lista.

Entonces lo que haremos será:

- Encontrar el índice en el arreglo
- Eliminarlo con el método splice

- Actualizar la lista de condimentos

```
function borrar(id){
  const index = invitados.findIndex((ele) => ele.id == id) /* 2.1 */
  invitados.splice(index, 1) /* 2.2 */

  /* Actualizamos la información en el HTML 3.3 */
  let html = ""
  for (invitado of invitados) {
    html += `<li>${invitado.nombre} <button> x </button> </li>`;
  }
  listaDeInvitados.innerHTML = html;
}
```

3. Conectando el botón de borrar con la función de borrado. Para conectar el botón con la función utilizaremos onclick y le pasaremos el id

```
html += `<li>${invitado.nombre} <button
onclick="borrar(${invitado.id})"> x </button> </li>`;
}
```

Actividad 8: Agrega la función de borrado al programa de tareas

Reutilizando código con funciones

El siguiente código se utiliza tanto al momento de agregar datos como al borrarlos, cada vez que cambia la lista tenemos de alguna forma debemos actualizar los elementos repitiendo este código.

```
  let html = ""
  for (invitado of invitados) {
    html += `<li>${invitado.nombre} <button> x </button> </li>`;
  }
  listaDeInvitados.innerHTML = html;
```

Para evitar repetirlo lo convertiremos en una función. Usualmente a las funciones que actualizan el DOM se denominan renders

```
function renderInvitados() {
  let html = ""
  for (invitado of invitados) {
    html += `<li>${invitado.nombre} <button> x </button> </li>`;
  }
  listaDeInvitados.innerHTML = html;
}
```

El paso final es cambiar el código llamando a la función. El código completo quedaría así:

```
   const listaDeInvitados = document.querySelector("#invitados")
   const invitadoInput = document.querySelector("#nuevoInvitado")
   const btnAgregar = document.querySelector("#agregarInvitado")
   const invitados = []

   /* Actualizamos la información en el HTML */
   function renderInvitados(){
     let html = ""
     for (let invitado of invitados) {
       html += `<li>${invitado.nombre} <button
onclick="borrar(${invitado.id})"> x </button> </li>`;
     }
     listaDeInvitados.innerHTML = html;
   }

   btnAgregar.addEventListener("click", () => {
     /* Agregamos el invitado al arreglo */
     const nuevoInvitado = {id: Date.now(), nombre: invitadoInput.value}
     invitados.push(nuevoInvitado)
     invitadoInput.value = ""
     renderInvitados()
   })

   function borrar(id){
     const index = invitados.findIndex((ele) => ele.id == id)
     invitados.splice(index, 1)
     renderInvitados()
   }
```

Filtrando elementos de una lista

Para esta sección vamos a crear un nuevo archivo llamado productos.html, en el archivo autocompleta el HTML y luego agrega el siguiente marcado en el body de la página web.

```
<input> <button>buscar</button>
<table border="solid">
  <thead>
    <th>id</th>
    <th>Nombre</th>
    <th>Precio</th>
  </thead>
  <tbody></tbody>
</table>
<script>
</script>
```

Y el siguiente script:

```
const tbody = document.querySelector("tbody");
const btn = document.querySelector("button");
const input = document.querySelector("input");

const productos = [
  { id: "AT12S", nombre: "Helado 5L", precio: 14990 },
  { id: "YEEM1", nombre: "Hielo 1kg", precio: 5990 },
  { id: "PSIQ4", nombre: "Agua 1.5L", precio: 14990 },
  { id: "MEPC7", nombre: "Soda 400Ml", precio: 890 }
];

function renderRows(productos) {
  tbody.innerHTML = "";
  productos.forEach((producto) => {
    tbody.innerHTML += `
            <tr>
              <td>${producto.id}</td>
              <td>${producto.nombre}</td>
              <td>${producto.precio}</td>
            </tr>`;
```

```
  });
}
renderRows(productos);
```

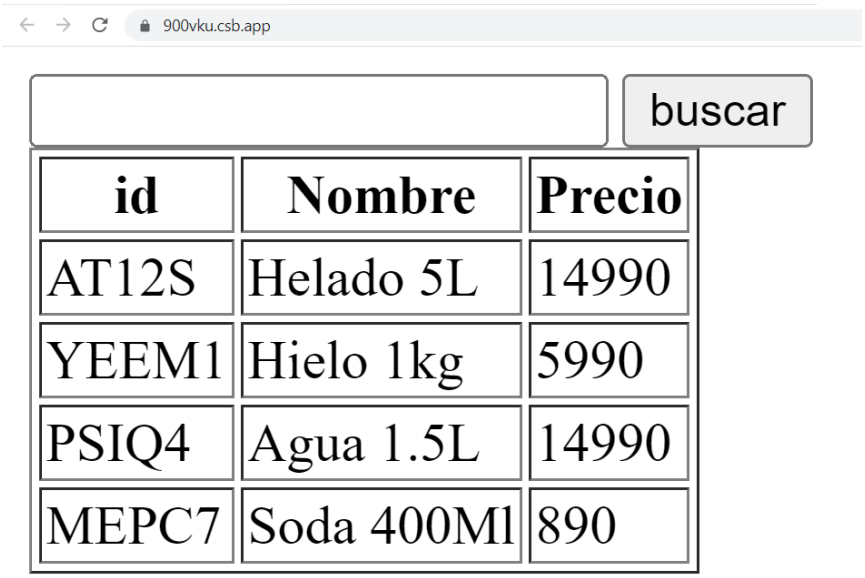

Imagen 6. Tabla de productos
Fuente: Desafío Latam

Creemos la lógica necesaria para que el usuario escriba un precio en el input y al apretar click en el botón "buscar", se filtren los productos para mostrar solo aquellos que sean igual o mayores al precio escrito.

Para esto agreguemos las siguientes tareas:

1. Agregar el evento click en el botón de búsqueda
2. Guardar en una variable el *value* del *input* y usarlo como argumento de la función `filtrarProductos`
3. Usar el método *filter* para filtrar el arreglo *productos* usando el precio escrito en el input
4. Ejecutar la función `renderRows` pasando como argumento el arreglo filtrado producido

```
btn.addEventListener("click", () => {
  const precio = input.value;
  const productosFiltrados = productos.filter(
```

```
    (producto) => producto.precio >= precio
  );
  renderRows(productosFiltrados);
});
```

Ahora si escribimos por ejemplo 10000 en el input y presionamos click en el botón “buscar”, veremos como solo se muestran aquellos productos igual o mayores a ese valor.

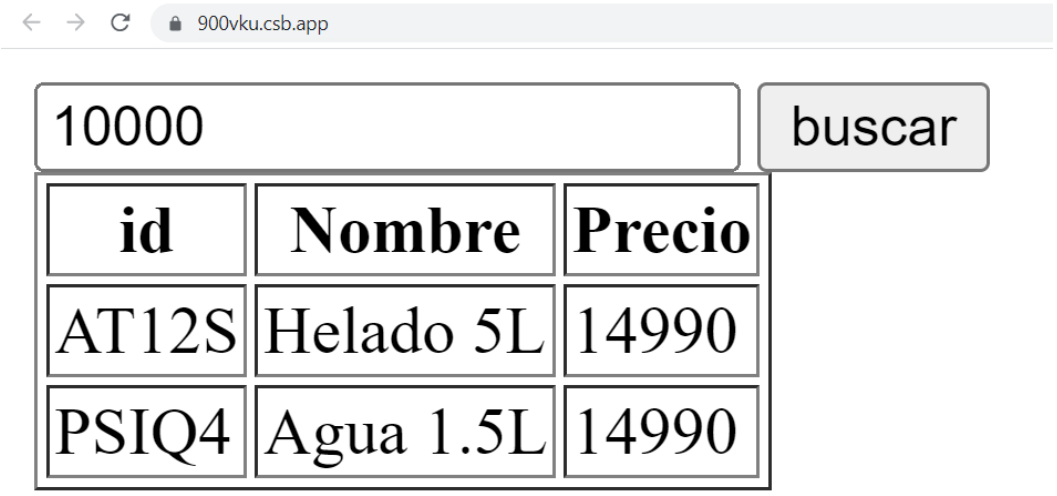

Imagen 7. Tabla de productos filtrada
Fuente: Desafío Latam

Actividad 9: Creando un buscador de tareas

Para la siguiente actividad crea una copia del programa milistadetaraes.html y renombra a busqueda.html puesto que esta actividad no te servirá exactamente para el desafío.

En la copia del programa de tareas crea un buscador de tareas por nombre. Para hacer la búsqueda tendrás que agregar un input y un botón "buscar"

Al buscar no compararemos contra el texto exacto, lo que haremos será una búsqueda parcial, o sea si alguien escribe sup puede aparecer como resultado supermercado o superior o marsupial, o sea el string final deberá contener en alguna parte la búsqueda. para eso ocuparemos un método de los strings llamado includes.

```
"Comprar los regalos".includes("Comprar") /* true */
```

Tips: Utiliza el método .filter del arreglo y compara utilizando includes. Además, recuerda actualizar la página con la función creada en la actividad anterior

Preguntas para preparar una entrevista laboral

- ¿Cuál es la diferencia entre el método `.map` y `.forEach`?
- ¿Para qué sirve pasarle una función al método `.sort`?
- ¿Cómo podemos saber si un elemento está dentro de un arreglo?
- ¿Qué significa que un tipo de dato sea mutable?
- ¿Por qué al hacer `const a = [1,2,3,4,5]` podemos cambiar valores dentro de `a`?