

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 un conjunto de métodos integrados que nos permiten realizar diversas tareas utilizando los datos o valores a los que pertenecen. Estos métodos nos brindan funcionalidades adicionales y nos permiten manipular y trabajar con los datos de manera eficiente.

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 comunes y cómo podemos utilizarlos para crear interacciones con el DOM.

¡Vamos con todo!



Tabla de contenidos

Agregando elementos a un arreglo	4
El método push	4
El método unshift	4
El método splice	5
Actividad 1: Agregando elementos con push	6
Eliminando elementos de un arreglo	6
El método shift	6
El método pop	6
El método splice	6
Actividad 2: Agregando y borrando elementos	7
Membrecía	7
Contar elementos	7
Buscando el índice de un elemento	8
El método findIndex	8
El método findIndex en un arreglo con objetos	8
Actividad 3: Encontrando y eliminando una película	10
Iterando arreglos	10
El método foreach	10
Transformando arreglos	11
¿Cuándo usar .forEach y cuándo .map?	11
Filtrando elementos de un arreglo	12
Actividad 4: Filtrar y contar	13
Unir arreglos	13
Arreglos y DOM	17
Creando una interfaz para insertar elementos	17
Actividad 5: Creando una interfaz para ingresar tareas.	19
Actividad 6: Actualizando la cuenta de tareas	20
Pensando en objetos	20
Modificando el programa de invitados para que funcione con objetos	20
Actividad 7: Modifica el programa de tareas	21
Eliminando elementos desde la interfaz	21
Paso a paso	21
Actividad 8: Agrega la función de borrado al programa de tareas	23
Reutilizando código con funciones	23
Filtrando elementos de una lista	25
Actividad 9: Creando un buscador de tareas	27
Preguntas para preparar una entrevista laboral	28
Solucionario Actividades	29



¡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 corresponderá al nuevo dato almacenado al final del arreglo.

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

Podemos ver que el string 'hola' fue agregado al final.



Importante: Podemos ver que `arreglo1` fue modificado aunque fue declarado con `const`. Esto se debe a que cuando el valor de la variable es un objeto o arreglo, los elementos o propiedades de estos pueden ser modificados, pero la variable en sí misma no puede ser reasignada.

```
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) /*['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 añadir elementos utilizaremos la sintaxis `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) /* [1,2,'hola',3,4,5] */
```

Para borrar un elemento en cualquier posición utilizaremos 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) /*[1,2,4,5] se eliminó elemento en posición 2 */
```

Finalmente, para reemplazar un elemento utilizaremos la sintaxis `arreglo.splice(index, 1, valor)`, en donde 1 nos indica que ese valor se eliminará y luego se insertará el valor indicado.

```
const arreglo7 = [1,2,3,4,5]
arreglo7.splice(2, 1, 'hola')
console.log(arreglo7) /*[1,2,'hola',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

Como ya mencionamos anteriormente, con splice podemos borrar un elemento en cualquier posición si lo utilizamos con los siguientes argumentos `arreglo.splice(index, 1)`.



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

Hablamos de membrecía cuando queremos determinar si un elemento pertenece a un arreglo o no. Para verificar esto ocuparemos el método `.includes()`

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

Usualmente se ocupan en conjunto con un `if` para realizar acciones basadas en la membrecía del elemento.

```
if(usuarios.includes("Erick")){  
  console.log("Hola Erick!!");  
}
```

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`

En algunos casos, es posible que no conozcamos el índice del elemento que queremos eliminar o modificar en un arreglo. Por ejemplo, si deseamos eliminar a un usuario por su nombre (u otro atributo), necesitamos buscar en el arreglo su índice correspondiente 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 apellidos:

```
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" )  
console.log(indiceDePerry); // 3
```

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" }  
];
```


Si quisiéramos eliminar al actor de id 541, entonces podríamos 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.



Actividad 3: Encontrando y eliminando una película

Realicemos el siguiente ejercicio en donde debemos 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.

```
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 */
```

El anterior ejemplo se lee como: por cada elemento del arreglo de estaciones, se realizará un `console.log` de ese elemento.

`.forEach` recibe una función como argumento y aplica esa función a cada elemento del arreglo. Esa función puede ser algo sencillo como un `console.log` o volver a modificar el DOM.

Si creamos una página web nueva llamada `foreach.html` con el siguiente script, 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 (cadenas de texto definidas entre comillas invertidas), además de interpolar variables, podemos hacer cálculos o ejecutar código js, como por ejemplo ``<p> ${2* x} </p>``.

Transformando arreglos

El método `.map` nos permite generar un nuevo arreglo a partir de la aplicación de una función a cada elemento del arreglo inicial.

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

El método `.map` no modifica el arreglo original, sino que genera uno nuevo a partir de este.

¿Cuándo usar `.forEach` y cuándo `.map`?

El método `.forEach` resulta útil cuando queremos realizar una acción inmediata con cada elemento iterado, como mostrarlo en pantalla o agregarlo al HTML. En cambio, el método `.map` se utiliza cuando queremos transformar los datos para seguir trabajando con ellos.

Filtrando elementos de un arreglo

Con el método `.filter` podemos filtrar los elementos 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)
```

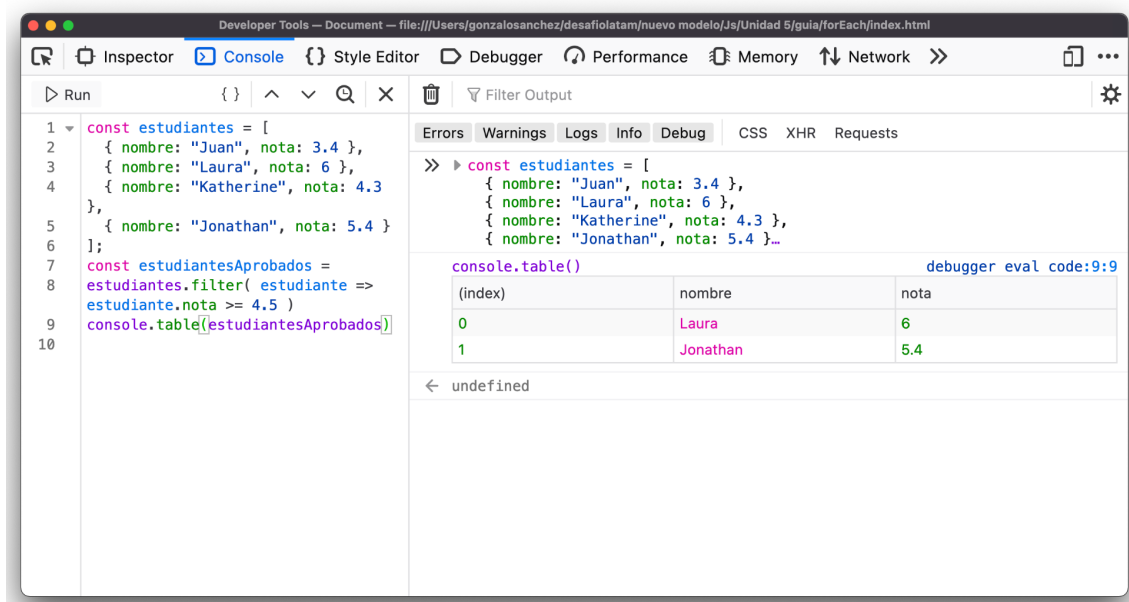


Imagen 2. Filter con un arreglo de objetos.

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. Con el método `filter`, filtrar el arreglo de trabajadores, creando un nuevo arreglo llamado `garzones` que contenga sólo los trabajadores con cargo "garzón".
2. Utiliza el arreglo resultante del requerimiento anterior y la propiedad `length` para mostrar con `console.log` cuántos garzones hay.

Unir arreglos

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

```
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 con la unión de los elementos, los cuales fueron almacenados 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` ordena predeterminadamente de forma ascendente, pero también puede recibir una función que permite comparar 2 elementos del arreglo y, en base a ese resultado, ordenar los elementos. Por ejemplo:

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

En el ejemplo anterior, se comparan los elementos `x` e `y` restando `y` menos `x`. Si el resultado es menor a 0, significa que `x` es mayor. Si el resultado es 0, ambos elementos son iguales. Y si el resultado es mayor a 0 significa que `y` es mayor. En este caso particular, la primera comparación sería 1 menos 4, lo que da como resultado -3. Esto significa que `x`, es decir, 4, es mayor, por lo se coloca primero que 1 en el ordenamiento resultante.

En otras palabras, utilizando una función de comparación personalizada en el método `sort` nos permite controlar el criterio de ordenamiento según nuestras necesidades.

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)
```

```
> 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)
```

(index)	nombre	nota
0	'Juan'	3.4
1	'Katherine'	4.3
2	'Jonathan'	5.4
3	'Laura'	6

► Array(4)

Imagen 3. Aplicación método sort en un arreglo de objetos.

Fuente: Desafío Latam 

Resumen de los métodos

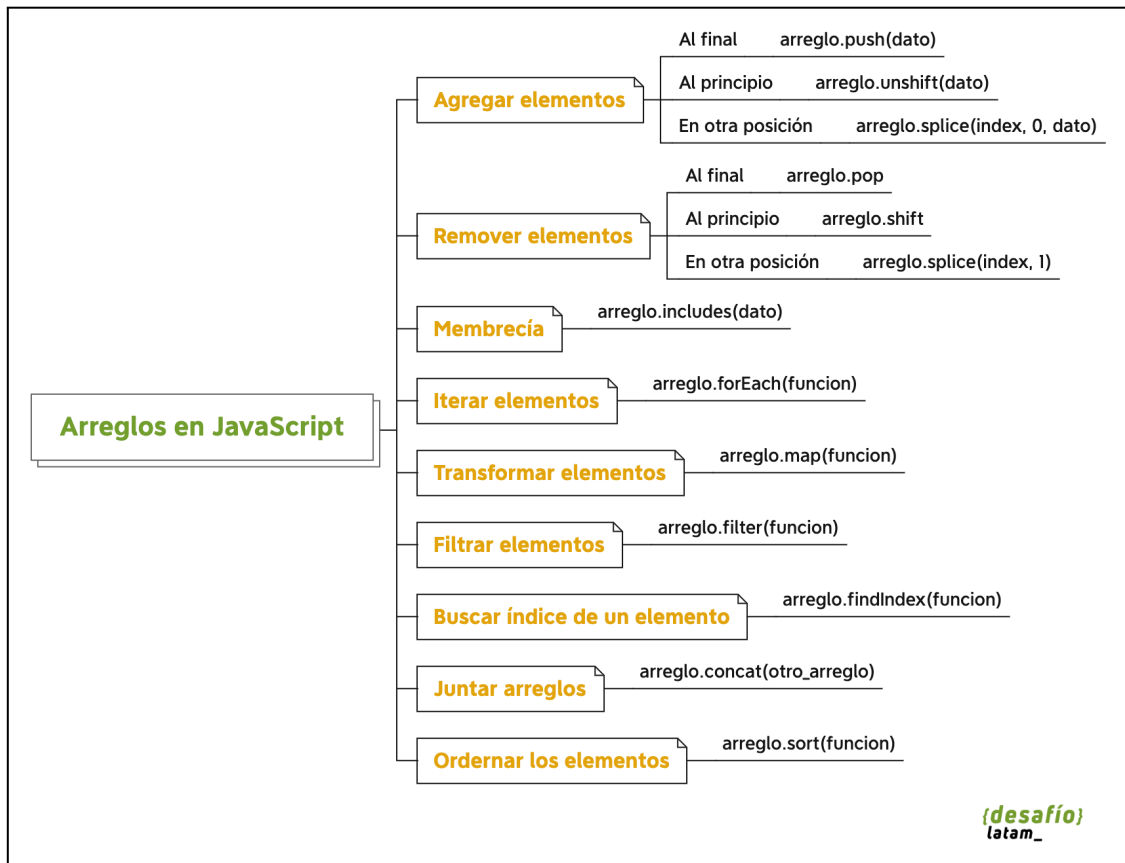


Imagen 4. Resumen de métodos.

Fuente: Desafío Latam

Arreglos y DOM

Creando una interfaz para insertar elementos

Ahora crearemos apps donde agregaremos y borrarremos 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:

```
<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>
```

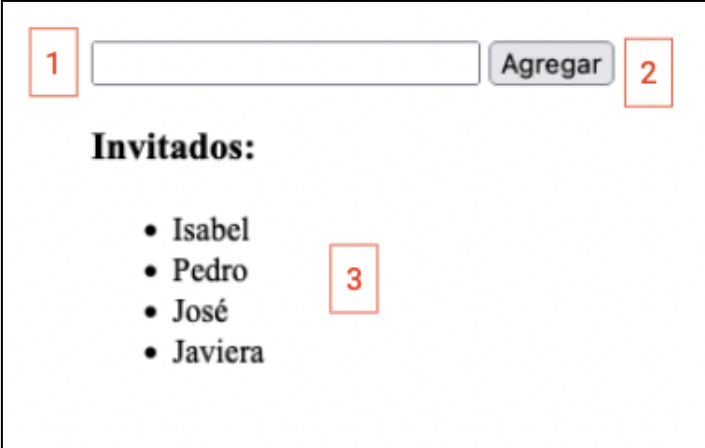


Imagen 5 Formulario de invitados
Fuente: Desafío Latam

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 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:

Y ahora nuestra lista ahora puede agregar nuevos invitados:

Invitados

- Sam
- Spencer
- Nataly
- Juanes

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 ` `
- 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 necesitamos almacenar múltiples elementos, como en el caso de una lista de invitados donde queremos guardar la fecha de ingreso, el nombre de un compañero, un identificador u otro tipo de información asociada, resulta más conveniente organizar dichos elementos en objetos.

Además, al pensar en objetos, tenemos la posibilidad de añadir un identificador que simplificará el proceso de eliminación o modificación de los elementos de la lista. Cabe mencionar que existen otras formas de lograrlo, pero en esta guía utilizaremos esta opción.

Modificando el programa de invitados para que funcione con objetos

Al guardar un invitado lo vamos a guardar junto con un id que nos ayudará más adelante a eliminar el dato. Este identificador se creará automáticamente utilizando una fecha única asociada al momento exacto del ingreso. Aunque existen otras formas de generar un identificador, es común que se obtenga a través de una API o una base de datos al momento de insertar los datos, pero por ahora utilizaremos la fecha para generar un valor único.

```
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 += `- ${invitado.nombre}</li>`;
}
listaDeInvitados.innerHTML = html;
})

```

Al momento de insertar los datos los insertamos dentro de un objeto `{id: Date.now(), nombre: nuevoInvitado}` y luego, para mostrarlos en 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` de `${invitado.nombre}` para ver qué sucede.



Actividad 7: Modifica el programa de tareas

1. Modifica el programa de tareas creado en la actividad anterior 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, lo saque del arreglo y actualice la lista.
3. El botón borrar debe llamar a la función.

Paso a paso

1. El botón lo agregaremos dentro del código que actualiza el HTML.

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

```

2. Creamos la función borrar

Recordemos que nuestros invitados fueron guardados en un objeto con la siguiente estructura `{id: 1231231312381283, nombre: "Juan"}` con el objetivo de tener un identificador que nos facilitara la eliminación o modificación de los datos. Nuestra función va a recibir este id y a partir de él, buscará el índice correspondiente en el arreglo y lo removerá utilizando splice. Luego, 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 invitados.

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

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

El script del programa para agregar invitados quedará finalmente así:

```
const listaDeInvitados = document.querySelector("#invitados")
const invitadoInput = document.querySelector("#nuevoInvitado")
const btnAgregar = document.querySelector("#agregarInvitado")
const invitados = []
btnAgregar.addEventListener("click", () => {
  const nuevoInvitado = invitadoInput.value
  invitados.push({id: Date.now(), nombre: nuevoInvitado})
  invitadoInput.value = ""
})
```

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

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

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



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

- Modifica el programa de tareas creado en la actividad anterior y agrega la función de borrado.

Reutilizando código con funciones

En nuestro código , cada vez que la lista de invitados cambia, ya sea porque se borran o se agregan elementos, esta debe ser actualizada repitiendo el siguiente código:

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

Para evitar repetirlo podemos convertirlo en 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> eliminar </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})"> eliminar </button> </li>`;  
  }  
  listaDeInvitados.innerHTML = html;  
}  
  
btnAgregar.addEventListener("click", () => {  
  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, agrega el siguiente marcado en el body de la página web.

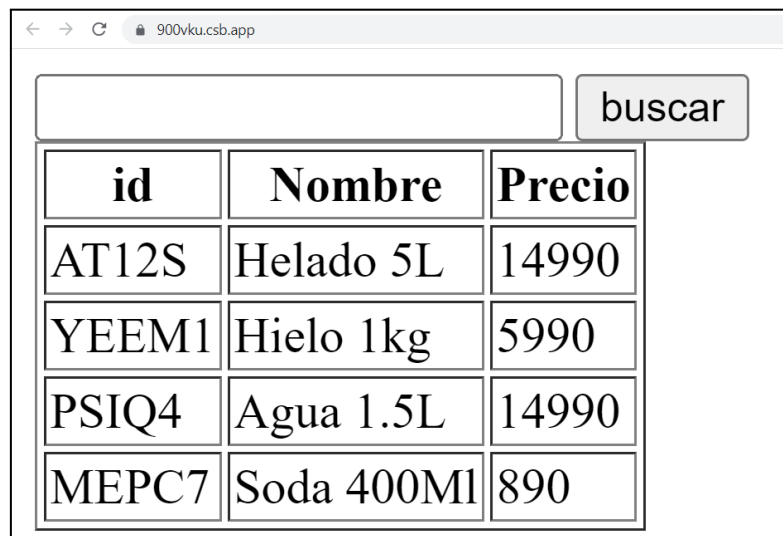
```
<input> <button>buscar</button>
<table border="solid">
  <thead>
    <th>id</th>
    <th>Nombre</th>
    <th>Precio</th>
  </thead>
  <tbody></tbody>
</table>
<script>
</script>
```

Luego, agrega 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);
```



The screenshot shows a web browser window with the address bar displaying "900vku.csb.app". Below the address bar is a search interface consisting of a text input field and a button labeled "buscar". Below the search bar is a table with three columns: "id", "Nombre", and "Precio". The table contains five rows of product data.

id	Nombre	Precio
AT12S	Helado 5L	14990
YEEM1	Hielo 1kg	5990
PSIQ4	Agua 1.5L	14990
MEPC7	Soda 400Ml	890

Imagen 6. Tabla de productos
Fuente: Desafío Latam

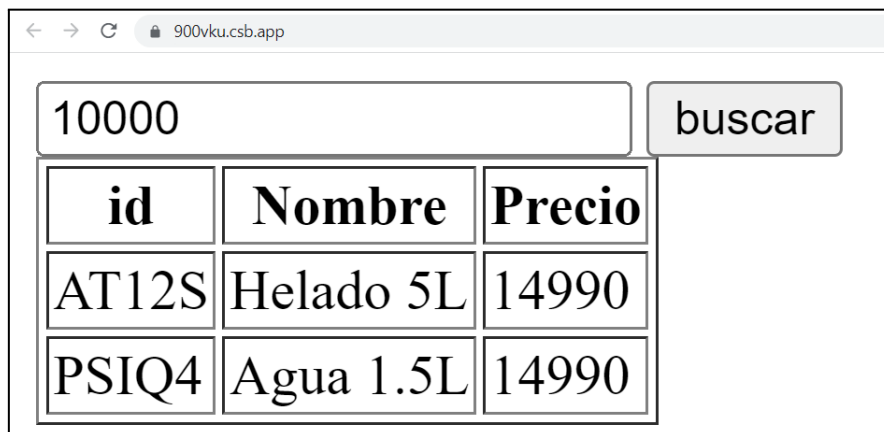
Ahora, necesitamos desarrollar la lógica necesaria para que el usuario escriba un precio en el input y al apretar click en el botón “buscar”, se filtren los productos para mostrar sólo aquellos que sean igual o mayores al precio escrito.

Para esto necesitamos realizar 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 que se muestran sólo aquellos productos igual o mayores a ese valor.



The screenshot shows a web browser window with the address bar displaying "900vku.csb.app". Below the address bar, there is a search input field containing the text "10000" and a button labeled "buscar". Below the search field, there is a table with three columns: "id", "Nombre", and "Precio". The table contains two rows of data:

id	Nombre	Precio
AT12S	Helado 5L	14990
PSIQ4	Agua 1.5L	14990

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".

El objetivo es lograr que el buscador sea de búsqueda parcial, es decir, si alguien escribe `sup` puede aparecer como resultado `supermercado` o `superior` o `marsupial`. En otras palabras la palabra ingresada en el buscador deberá estar contenida en alguna de las palabras de la lista. Para esto 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 una función que reutilice código como vimos anteriormente.



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`?

Solucionario Actividades

Solución Actividad 1

```
let superHeroes= [  
  "Ironman",  
  "Superman",  
  "Hawkeye"  
]  
  
  superHeroes.push('Spiderman')  
  superHeroes.unshift('Batman')  
  superHeroes.splice(3, 0, 'Aquaman')  
  console.log(superHeroes)
```

Solución Actividad 2

```
const usuarios = ["Erick", "Cristian", "Max", "Claudia"];  
  usuarios.pop()  
  usuarios.unshift('María José')  
  usuarios.splice(2, 1)  
  console.log(usuarios)
```

Solución Actividad 3

```
const peliculas = [  
  {id: 1, nombre: "Thor"},  
  {id: 2, nombre: "Ant-Man"},  
  {id: 3, nombre: "Terminator"},  
  {id: 4, nombre: "Ip Man"},  
  {id: 5, nombre: "Rocky"},  
]  
  
const indiceTerminator = peliculas.findIndex( pelicula => pelicula.id === 3)  
  
peliculas.splice(indiceTerminator, 1)  
  
console.log(peliculas);
```

Solución Actividad 4

```
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"},  
]  
  
const garzones = trabajadores.filter(trabajador => trabajador.cargo ===  
  "garzón");  
  
console.log(garzones.length); //4
```

Solución Actividad 5

```
<!DOCTYPE html>  
<html lang="es">  
  <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>Mi lista de tareas</title>  
  </head>  
  <body>  
    <input id="nuevaTarea">  
    <button id="agregarTarea">Agregar</button>  
    <h3>Tareas: </h3>  
    <ul id="tareas"></ul>  
    <script>  
      const listaDeTareas = document.querySelector("#tareas")  
      const tareaInput = document.querySelector("#nuevaTarea")  
      const btnAgregar = document.querySelector("#agregarTarea")  
      const tareas = []  
      btnAgregar.addEventListener("click", () => {  
        const tarea = tareaInput.value  
        tareas.push(tarea)  
        tareaInput.value = ""  
        let html = ""  
        for (let tarea of tareas) {
```

```
    html += `<li>${tarea}</li>`;
  }
  listaDeTareas.innerHTML = html;
})
</script>
</body>
</html>
```

Solución Actividad 6

```
<!DOCTYPE html>
<html lang="es">
  <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>Mi lista de tareas</title>
  </head>
  <body>
    <input id="nuevaTarea">
    <button id="agregarTarea">Agregar</button>
    <h3>Tareas: </h3>
    <span id="cuenta-tareas"> </span>
    <ul id="tareas"></ul>
    <script>
      const listaDeTareas = document.querySelector("#tareas")
      const tareaInput = document.querySelector("#nuevaTarea")
      const btnAgregar = document.querySelector("#agregarTarea")
      const cuentaTareas = document.querySelector("#cuenta-tareas");
      const tareas = []
      btnAgregar.addEventListener("click", () => {
        const tarea = tareaInput.value
        tareas.push(tarea)
        tareaInput.value = ""
        let html = ""
        for (let tarea of tareas) {
          html += `<li>${tarea}</li>`;
        }
        listaDeTareas.innerHTML = html;
        cuentaTareas.textContent = `Total de tareas: ${tareas.length}`;
      })
    </script>
  </body>
</html>
```

Solución Actividad 7

```
<!DOCTYPE html>
<html lang="es">
  <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>Mi lista de tareas</title>
  </head>
  <body>
    <input id="nuevaTarea">
    <button id="agregarTarea">Agregar</button>
    <h3>Tareas: </h3>
    <span id="cuenta-tareas"> </span>
    <ul id="tareas"></ul>
    <script>
      const listaDeTareas = document.querySelector("#tareas")
      const tareaInput = document.querySelector("#nuevaTarea")
      const btnAgregar = document.querySelector("#agregarTarea")
      const cuentaTareas = document.querySelector("#cuenta-tareas");
      const tareas = []
      btnAgregar.addEventListener("click", () => {
        const tarea = tareaInput.value
        tareas.push({id: Date.now(), tarea: tarea})
        tareaInput.value = ""
        let html = ""
        for (let tarea of tareas) {
          html += `<li>${tarea.tarea}</li>`;
        }
        listaDeTareas.innerHTML = html;
        cuentaTareas.textContent = `Total de tareas: ${tareas.length}`;
      })
    </script>
  </body>
</html>
```

Solución Actividad 8

```
<!DOCTYPE html>
<html lang="es">
  <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>Mi lista de tareas</title>
```



```
</head>
<body>
  <input id="nuevaTarea">
  <button id="agregarTarea">Agregar</button>
  <h3>Tareas: </h3>
  <span id="cuenta-tareas"> </span>
  <ul id="tareas"></ul>
  <script>
    const listaDeTareas = document.querySelector("#tareas")
    const tareaInput = document.querySelector("#nuevaTarea")
    const btnAgregar = document.querySelector("#agregarTarea")
    const cuentaTareas = document.querySelector("#cuenta-tareas");
    const tareas = []
    btnAgregar.addEventListener("click", () => {
      const tarea = tareaInput.value
      tareas.push({id: Date.now(), tarea: tarea})
      tareaInput.value = ""
      let html = ""
      for (let tarea of tareas) {
        html += `<li>${tarea.tarea} <button
onclick="borrar(${tarea.id})"> eliminar </button></li>`;
      }
      listaDeTareas.innerHTML = html;
      cuentaTareas.textContent = `Total de tareas: ${tareas.length}`;
    });
    function borrar(id){
      const index = tareas.findIndex((ele) => ele.id == id)
      tareas.splice(index, 1)

      let html = ""
      for (tarea of tareas) {
        html += `<li>${tarea.tarea} <button
onclick="borrar(${tarea.id})"> eliminar </button></li>`;
      }
      listaDeTareas.innerHTML = html;
      cuentaTareas.textContent = `Total de tareas: ${tareas.length}`;
    }
  </script>
</body>
</html>
```

Solución Actividad 9

```
<!DOCTYPE html>
<html lang="es">
  <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>Mi lista de tareas</title>
  </head>
  <body>
    <input id="nuevaTarea">
    <button id="agregarTarea">Agregar</button><br>
    <input id = "buscarTarea">
    <button id="btnBuscarTarea">Buscar</button>
    <h3>Tareas: </h3>
    <span id="cuenta-tareas"> </span>
    <ul id="tareas"></ul>
    <script>
      const listaDeTareas = document.querySelector("#tareas")
      const tareaInput = document.querySelector("#nuevaTarea")
      const buscadorInput = document.querySelector("#buscarTarea")
      const btnAgregar = document.querySelector("#agregarTarea")
      const btnBuscar = document.querySelector("#btnBuscarTarea")
      const cuentaTareas = document.querySelector("#cuenta-tareas");
      const tareas = []

      btnAgregar.addEventListener("click", () => {
        const tarea = tareaInput.value
        tareas.push({id: Date.now(), tarea: tarea})
        tareaInput.value = ""
        renderList(tareas)
      });

      function renderList(tareas){
        let html = ""
        for (let tarea of tareas) {
          html += `<li>${tarea.tarea} <button
onclick="borrar(${tarea.id})"> eliminar </button></li>`;
        }
        listaDeTareas.innerHTML = html;
        cuentaTareas.textContent = `Total de tareas: ${tareas.length}`;
      }
    </script>
  </body>
</html>
```

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

btnBuscar.addEventListener("click", () => {
  const tareaBuscada = buscadorInput.value;
  const tareasFiltradas = tareas.filter(
    (tarea) => tarea.tarea.includes(tareaBuscada)
  );
  renderList(tareasFiltradas);
})

</script>
</body>
</html>
```