**Arrays filter, arrays map**

**👋 Introducción**

¡Hola! 👋. Bienvenido a un nuevo paso en el cual veremos funcionalidades de arrays en JavaScript! Nos centraremos en dos herramientas fundamentales e increíblemente versátiles: los métodos filter y map. Estos métodos son esenciales en el arsenal de cualquier desarrollador, permitiéndote no solo transformar datos de manera elegante, sino también crear flujos de trabajo eficientes y expresivos.

Lo que abordaremos:

* Comenzaremos explorando el método filter, una herramienta esencial para la creación de subconjuntos de datos. Aprenderás a seleccionar elementos específicos basándote en criterios personalizados, abriendo la puerta a la manipulación precisa de tus arrays.
* Funciones de Retorno Creativas para filter: Nos sumergiremos en la creación de funciones de retorno personalizadas que ampliarán tus capacidades de filtrado.
* Desbloqueando el Potencial de map: Avanzaremos hacia el método map, un aliado poderoso en la transformación de datos. Descubrirás cómo crear nuevos arrays modificando cada elemento del original, permitiéndote realizar cambios masivos y estructurales con elegancia y eficiencia.
* Combinando filter y map para Soluciones Complejas: Verás cómo estos dos métodos pueden trabajar en armonía para resolver problemas complejos.
* Optimización y Mejores Prácticas: Exploraremos técnicas de optimización para asegurar que tu código sea eficiente y fácil de mantener.

¡Manos a la obra!

**📖 Material Descargable**

Aquí es donde encontrarás el material descargable que complementará los ejercicios prácticos que vamos a ver en los siguientes pasos.

👉[**Teoría  Arrays - filter y map**](https://drive.google.com/file/d/1fsYEWbXw1QLR7a1Q9ZsD8rdaJ1rlLEc7/view?usp=drive_link)

**⏭️ En el paso siguiente** pondrás en práctica estos conceptos

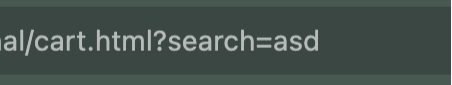
**Métodos de Array - filter**

En esta sección, aprenderemos a usar el método filter de los arrays.

**✏️ Actividad: Verificar comportamiento por default del form, en el header**



Si escribes algo en el form y le das al enter, notarás que el comportamiento por defecto es intentar mandar esos datos a algún lado.



Además notarás que la página se recarga, perdiendo los datos del mismo.

Para ello necesitamos, mediante JavaScript, "anular" el comportamiento por defecto.

**✏️ Actividad: Modificación del form en el header del archivo cart.html**

Necesitaremos agregarle un atributo id al <form> para recuperarlo desde JavaScript y un atributo name , con el valor "search', al input.

El form quedaría así:

<form id="filter-form"><input type="text" placeholder="Search" name="search"/></form>

**✏️ Actividad: Recuperar el form desde JavaScript**

Dado que en cart.html ya tenemos el id del form, simplemente con un querySelector, podríamos recuperarlo.

El código quedaría así:

const form = document.querySelector("#filter-form"); // `#` es el selector CSS para los `id`.

console.log(form); // Chequeamos en la consola que no sea `null` o `undefined`. Si es el caso, verifica el `id` tanto en este archivo com en el `cart.html`.

**✏️ Actividad: Agregar listener**

Necesitaremos agregar un listener el cual detecte el enter del usuario, para poder filtrar en base al contenido del input asociado.

El código sería algo como esto:

form.addEventListener('submit', event => {

  event.preventDefault()

})

Este evento se llama submit. Se "ejecuta" cuando el usuario hace un enter, hace un click en un **<button></button>** de tipo submit o bien si el **<button></button>** es el último button del form.

Nótese que la primer línea de la función es: event.preventDefault(). Esto es justamente para anular el comportamiento por defecto de los formularios.

Ahora no debería recargarse la página al hacer enter.

**✏️ Actividad:  Agregar funcionalidad con filter**

Es hora de aplicar el método **filter**.

Primero modificaremos la función **renderTotalPrice**, para que reciba el carrito como parámetro.

Algo como esto:

function renderTotalPrice(cart) {

   // Código existente...

}

Después modificaremos un poco la función populate para que reciba como parámetro el array con los productos y elimine el contenedor del carrito previo (para evitar duplicados).

Algo como esto:

function populate(cart) {

    document.querySelector(".products")?.remove();

    // Código existente...

    // En cada llamada de la función `renderTotalPrice` habrá que mandarle el `cart` como argumento.

}

**document.querySelector(".products")?.remove();**: Selecciona el elemento con la clase products y ejecuta la función remove(), removiéndolo del DOM.

La función completa quedaría algo como esto:

function populate(cart) {

  const cartContainer = document.querySelector(".cart-container");

  document.querySelector(".products")?.remove();

  if (!cartContainer) return console.log("Error");

  const productContainer = document.createElement("div");

  productContainer.classList.add("card", "products");

  cart.forEach((product) => {

    const productElement = document.createElement("article");

    productElement.classList.add("product-cart");

    const productImage = document.createElement("img");

    productImage.src = product.image;

    productImage.alt = product.title;

    const productMiddle = document.createElement("div");

    productMiddle.classList.add("middle");

    const productInfo = document.createElement("div");

    productInfo.classList.add("info");

    const productTitle = document.createElement("strong");

    productTitle.classList.add("title");

    productTitle.textContent = product.title;

    const productDescription = document.createElement("span");

    productDescription.classList.add("description");

    productDescription.textContent = product.description;

    const productInput = document.createElement("input");

    productInput.type = "number";

    productInput.value = product.quantity;

    productInput.min = 1;

    productInput.addEventListener("change", (event) => {

      const { value } = event.target;

      const index = cart.findIndex(({ title }) => product.title === title);

      if (index < 0) return console.log("Error");

      cart[index].quantity = value;

      renderTotalPrice(cart);

    });

    const productPrice = document.createElement("strong");

    productPrice.classList.add("price");

    productPrice.textContent = `AR$ ${formatPrice(product.price)}`;

    productInfo.appendChild(productTitle);

    productInfo.appendChild(productDescription);

    productMiddle.appendChild(productInfo);

    productMiddle.appendChild(productInput);

    productElement.appendChild(productImage);

    productElement.appendChild(productMiddle);

    productElement.appendChild(productPrice);

    productContainer.appendChild(productElement);

  });

  cartContainer.insertAdjacentElement("afterbegin", productContainer);

  renderTotalPrice(cart);

}

Después, modificaremos el **listener**del **document**.

Algo como esto:

document.addEventListener("DOMContentLoaded", () => populate(cart));

De esta manera, al cargar el DOM, debería popular el cart.

Ahora mismo deberías tener lo mismo que antes, excepto que ahora tenemos control de qué array de productos podemos popular.

**✏️ Actividad:  Función filter**

Finalmente, añadiremos la funcionalidad del filter en el listener del form:

form.addEventListener("submit", (event) => {

  event.preventDefault();

  const userInput = event.target.search.value;

  const filteredCart = cart.filter((item) =>

    item.title.toLowerCase().includes(userInput.toLowerCase())

  );

  populate(filteredCart);

});

**event.target**: Hace referencia al elemento form.

**event.target.search**: Como el input asociado al form tiene un atributo name con el valor search, podemos acceder a este mediante event.target.search. Si el valor del atributo name hubiera sido hola

<form id="filter-form">

    <input type="text" placeholder="Search" name="hola" />

    <!-- El campo name tiene como valor "hola" -->

</form>

habríamos accedido event.target.hola.

**event.target.search.value**: Obtiene el valor que escribe el usuario.

**item.title.toLowerCase().includes(value.toLowerCase())**:

* Por cada ítem del carrito, se fija si el valor que escribe el usuario en lowercase, está incluido en el título del ítem en sí.

**const filteredCart**:

* Almacena el array filtrado.

**populate(filteredCart)**:

* Popula el carrito de compras con el nuevo carrito.

Si escribes algo y le das al enter, debería filtrar los productos del carrito que contengan eso que has escrito.

**🧰Material Complementario**

Si deseas profundizar más en lo aprendido en este paso, te recomendamos que visites los siguientes recursos:

* [**'submit' Event**](https://developer.mozilla.org/en-US/docs/Web/API/HTMLFormElement/submit_event)

**Arrays - map**

En esta sección aprenderemos a usar la función map de los Arrays, aplicando un descuento del 10% a cada producto.

**✏️  Actividad:  Agregar button a cart.html**

Como primer paso, agregaremos un **<button></button>**a cart.html en la sección del cart-container.

Los invitamos a que traten de crear el **<button></button>** ustedes mismos.

Debería quedar así:

**📢 Pista**: Existe una clase CSS llamada btn-primary.

**✏️  Actividad:   Recupera el button desde JavaScript**

Necesitaremos recuperar el button desde JavaScript, para ello una opción, podría ser agregarle un id al mismo.

Finalmente desde JavaScript, lo recuperamos:

const btnDiscount = document.querySelector("#btn-discount")

**✏️  Actividad:  Agregar listener para el click**

Vamos a necesitar un listener para que se ejecute el descuento de un 10%.

const btnDiscount = document.querySelector("#btn-discount");

btnDiscount.addEventListener("click", () => {

  console.log("Click!");

});

**✏️  Actividad:   Lógica de descuento**

Finalmente, aplicaremos la función map en el listener

btnDiscount.addEventListener("click", () => {

  const cartWithDiscount = cart.map((item) => {

    return {

      ...item,

      price: item.price \* 0.9,

    };

  });

  populate(cartWithDiscount);

});

* { ...item }: Esto crea un nuevo objeto, el cual es una copia del item actual. Se utiliza el operador de propagación (...), más conocido como spread operator para copiar todas las propiedades y valores del objeto item.
* price: item.price \* 0.9: Aquí se sobrescribe el valor de la propiedad price del nuevo objeto. Se toma el precio original del artículo item.price y se multiplica por 0.9, lo que efectivamente aplica un descuento del 10% al precio para cada producto.

**Pregunta: ¿Por qué si se hace click varias veces en el botón de descuento, sólo se aplica una vez?**

**✏️  Actividad:  Descuento condicional**

Ahora intenta aplicar el descuento solo a los productos que tienen un precio mayor a AR $500.000. (mediana dificultad)

**⏭️ En el paso siguiente** verás cómo almacenar datos de manera local, en el navegador del usuario.