**Introducción a Local Storage**

**👋 Introducción**

¡Hola! 👋. ¡Bienvenido a un nuevo paso que te llevará a descubrir la magia del almacenamiento local en JavaScript! Este segmento te presentará una herramienta poderosa que te permite persistir datos en el navegador del usuario: .  **localStorage**

Comenzaremos explorando qué es localStorage y cómo se diferencia de otras formas de almacenamiento de datos.

Aprenderás cómo almacenar y recuperar datos simples, como cadenas de texto y números. Verás cómo este método simple puede hacer una gran diferencia en la experiencia del usuario.

¡Prepárate para descubrir el potencial ilimitado de localStorage y cómo puede transformar la manera en que gestionas y almacenas datos en tus aplicaciones web!

**📖 Material Descargable**

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

[**👉**](https://drive.google.com/file/d/1UgcZsyYTIsmfN8WR9KxM07ko_ehi_XiO/view?usp=drive_link)

# Guardando información en LocalStorage

En este paso aprenderemos a guardar productos en el localStorage.

### ✏️ Actividad:  Modificar product.html

Necesitaremos modificar el html del archivo product.html, agregándole un id a todos los atributos del producto que necesitemos para el carrito.

1. **id al button 'Agregar al carrito'**:

En la línea 155 debería haber un button 'Agregar al carrito'. Agregar un id al mismo.

<div class="bottom">

    <button class="btn-outline" id="btn-add-to-cart">Añadir al Carrito</button> <!-- Button -->

</div>

1. **id al input**

En la línea 151 debería haber un input de tipo number. Agregar un id al mismo.

<div class="top">

    <input id="input-quantity" type="number" value="1" min="1"/>  <!-- Input -->

    <button class="btn-primary">Comprar</button>

</div>

1. **id al precio**

En la línea 124 debería haber un h2 con el precio del producto. Agregar un id al mismo.

<h2 class="checkout-total-price" id="price">$152.400</h2>

1. **id a la descripción**

En la línea 95 debería haber un p con la descripción del producto. Agregar un id al mismo.

<p id="description">...</p>

1. **id al título**

En la línea 74 debería haber un h1 con el título del producto. Agregar un id al mismo.

<h1 class="title" id="title">

    Apple iPhone XR,64 GB, Black - Fully Unlocked (Refurbished)

</h1>

### ✏️ Actividad: Recuperar los elementos

En el archivo product.js, en la función ready, después de iterar los thumbnails, recuperaremos todos los atributos previamente mencionados.

const btnAddToCart = document.querySelector("#btn-add-to-cart");

const inputQuantity = document.querySelector("#input-quantity");

const priceEl = document.querySelector("#price");

const descriptionEl = document.querySelector("#description");

const titleEl = document.querySelector("#title");

### ✏️ Actividad: Agregar click listener

Agregaremos un listener al botón 'Agregar al carrito'.

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

});

### ✏️ Actividad: Transformación de datos (Parseo)

Necesitamos recuperar los valores de cada elemento. El único que nos debería traer problemas es el price, ya que a la hora de recuperar su valor nos va a traer el texto $152.400. Esto es incorrecto ya que nosotros queremos que el precio sea 152.400 en formato número.

Para ello, además de recuperar los datos, vamos a tener que parsear los mismos.

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

    const description = descriptionEl.textContent;

    const title = titleEl.textContent;

    const quantity = Number(inputQuantity.value);

    const price = parseCurrency(priceEl.textContent);

    const image = selected.src; // Previamente seleccionada

    console.log({

        description,

        title,

        image,

        quantity,

        price,

        image

    });

});

function parseCurrency(text) {

    // Eliminar el símbolo del dólar y los puntos

    const numberString = text.replace(/\$|\.+/g, "");

    // Convertir a número

    return Number(numberString);

}

La función parseCurrency convierte el texto mencionado anteriormente, en un número de Javascript. text.replace(/\$|\.+/g, ""): Utiliza expresiones regulares para eliminar todo lo innecesario. NO centrarse en esto ya que no es el punto clave de este paso. De todas formas, al final del mismo, habrá más información acerca de las expresiones regulares :)

### ✏️ Actividad: LocalStorage

Una vez recuperados los valores, debemos agregar dicho producto al carrito.

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

    const description = descriptionEl.textContent;

    const title = titleEl.textContent;

    const quantity = Number(inputQuantity.value);

    const price = parseCurrency(priceEl.textContent);

    const image = selected.src; // Previamente seleccionada

    // Construcción del objeto

    const product = {

      quantity,

      description,

      title,

      price,

      image,

    };

    const cart = JSON.parse(localStorage.getItem("cart") ?? "[]");

    localStorage.setItem("cart", JSON.stringify([...cart, product]));

});

1. **Recuperación del Carrito desde el Local Storage:**

const cart = JSON.parse(localStorage.getItem("cart") ?? "[]");

Aquí se intenta recuperar un carrito de compras, existente o no, del Local Storage del navegador usando localStorage.getItem("cart"). Si no existe ningún carrito (es decir, getItem retorna null), se usa el operador nullish coalescing (??) para proporcionar un valor predeterminado, que en este caso es un arreglo vacío ("[]"). El resultado, que es una cadena de texto, se convierte de nuevo a un objeto o arreglo JavaScript mediante JSON.parse.

1. **Agregar el Producto al Carrito y Guardarlo en el Local Storage:**

localStorage.setItem("cart", JSON.stringify([...cart, product]));

Aquí, se agrega el nuevo product al arreglo cart (que podría estar vacío o contener otros productos) y luego se actualiza el carrito en el localStorage. Esto se hace mediante el uso del operador de propagación (...) (spread operator) para incluir todos los elementos existentes en cart, seguido del nuevo product. Luego, este arreglo actualizado se convierte en una cadena de texto JSON mediante JSON.stringify y se guarda en el localStorage bajo la clave "cart" usando localStorage.setItem.

Su LocalStorage debería verse así:

Nótese que si agregas varias veces el producto se va a ir repitiendo. Esto lo arreglaremos en el próximo paso.

### 🧰Material Complementario

Si deseas profundizar más en lo aprendido en este paso, te recomendamos que visites el siguiente recurso:

* [**Expresiones Regulares**](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_expressions)

# Ajustando detalles en localStorage

En este paso terminaremos de ajustar todos los detalles del localStorage, como así también usar el carrito que se encuentra en el mismo, para la página cart.html

### ✏️ Actividad: Resolver Bug Duplicado

En el paso anterior, mencionamos que si se agregaba varias veces el mismo producto, se duplicaba.

1. **Usar Función map para solucionar el problema**:

Utilizaremos la función map para que no se dupliquen los productos. El listener queda así:

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

    const quantity = Number(inputQuantity.value);

    const description = descriptionEl.textContent;

    const title = titleEl.textContent;

    const price = parseCurrency(priceEl.textContent);

    const image = selected.src;

    const product = {

      quantity,

      description,

      title,

      price,

      image,

    };

    const cart = JSON.parse(localStorage.getItem("cart") ?? "[]");

    const titles = cart.map((item) => item.title);

    const isInCart = titles.includes(title);

    if (isInCart) {

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

        if (item.title === title) {

          return {

            ...item,

            quantity: item.quantity + quantity,

          };

        } else {

          return item;

        }

      });

      localStorage.setItem("cart", JSON.stringify(newCart));

    } else {

      localStorage.setItem("cart", JSON.stringify([...cart, product]));

    }

  });

1. **const titles = cart.map((item) => item.title)`**:

Esta línea crea un arreglo con los títulos de los productos en el carrito. Utiliza el método .map() para transformar cada elemento del carrito en su título, quedando titles como un arreglo de strings (con los títulos de cada producto).

1. **const isInCart = titles.includes(title)**:

Aquí se verifica si el producto que se quiere añadir ya está en el carrito. titles.includes(title) devuelve true si el título del producto está en el arreglo de títulos (titles).

1. **Bloque if (isInCart) {...} else {...}**:

Este bloque condicional maneja dos casos: cuando el producto ya está en el carrito y cuando no lo está.

1. Dentro del if:

* Si el producto ya está en el carrito (isInCart es true), se crea un nuevo carrito (newCart) donde se actualiza la cantidad del producto de la siguiente manera (adentro del map): Para cada elemento en el carrito, si el título del elemento coincide con el título del producto que se está añadiendo, se devuelve una copia del elemento con su cantidad incrementada. Si no coincide, se devuelve el elemento tal como está.
* localStorage.setItem("cart", JSON.stringify(newCart)): Esta línea actualiza el carrito en el localStorage con el newCart modificado. JSON.stringify(newCart) convierte el objeto del carrito en una cadena de texto para almacenarlo.

1. Dentro del else:

Si el producto no está en el carrito (isInCart es false), se añade el nuevo producto al final del carrito existente y se actualiza el localStorage. [...cart, product] crea un nuevo arreglo con todos los elementos del carrito actual, más el nuevo producto.

De esta manera podemos solucionar la repetición del producto en el carrito.

Nótese que solo podemos agregar un producto ya que la página product.html es estática.

Desafío (difícil): Agregar un botón 'Agregar al carrito' en cada producto en la página index.html (y desarrollar el respectivo JavaScript) para poder agregarlo al carrito.

### ✏️ Actividad:  Usar LocalStorage para el carrito

En el archivo cart.js, debemos modificar la variable cart para que en vez de que esté hardcodeada, es decir, con los productos seteados a mano, use el localStorage.

const cart = JSON.parse(localStorage.getItem("cart") ?? "[]");

Esta línea recupera los datos del carrito del localStorage del navegador. localStorage.getItem("cart") obtiene el carrito como una cadena de texto. Si no hay ningún carrito almacenado (null), se usa el operador de Nullish Coalescing Operator (??) para usar un arreglo vacío ("[]") como valor por defecto. JSON.parse convierte la cadena de texto en un objeto JavaScript (en este caso, un arreglo).

Ahora deberían poder agregar productos al carrito y a la hora de refrescar la página, no perder los mismos.