Clase 10. JAVASCRIPT

Storage & JSON

Esta clase va a ser

- grabada

CLASE N°9

Glosario

Evento: es la manera que tenemos en Javascript de controlar las acciones de los usuarios, y definir un comportamiento de la página o aplicación cuando se produzcan. Hay distintos tipos de eventos:

- ✓ Eventos de mouse
- ✓ Eventos de teclado
- ✓ Evento change
- ✓ Evento submit

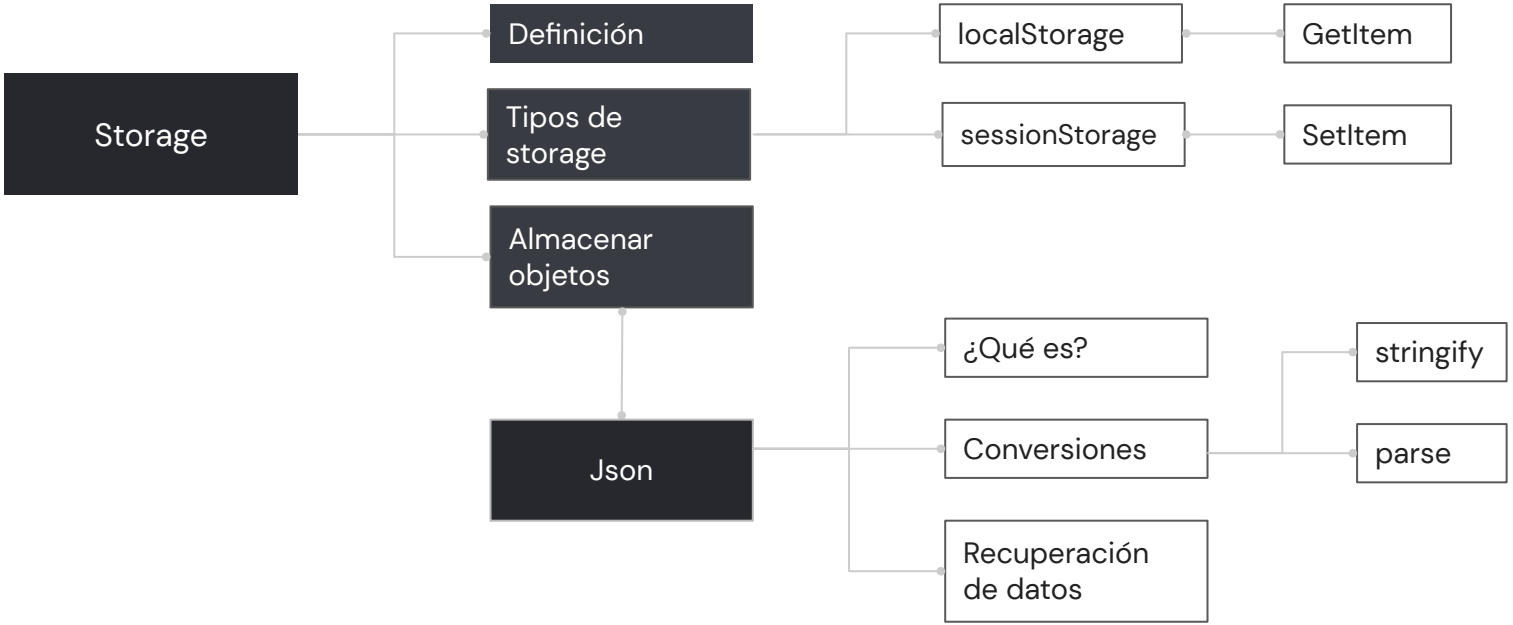
Objetivos de la clase

- **Definir Storage**
- **Identificar y diferenciar localStorage y sessionStorage.**
- **Definir JSON** y entender su alcance y utilidad en cada situación.

Temario

09	10	11
Eventos	**Storage & Json**	**Workshop I**
✓ Eventos en JS ✓ Cómo definirlos ✓ Eventos más comunes	✓ Storage ✓ JSON	✓ Repaso general ✓ Recomendaciones para el proyecto

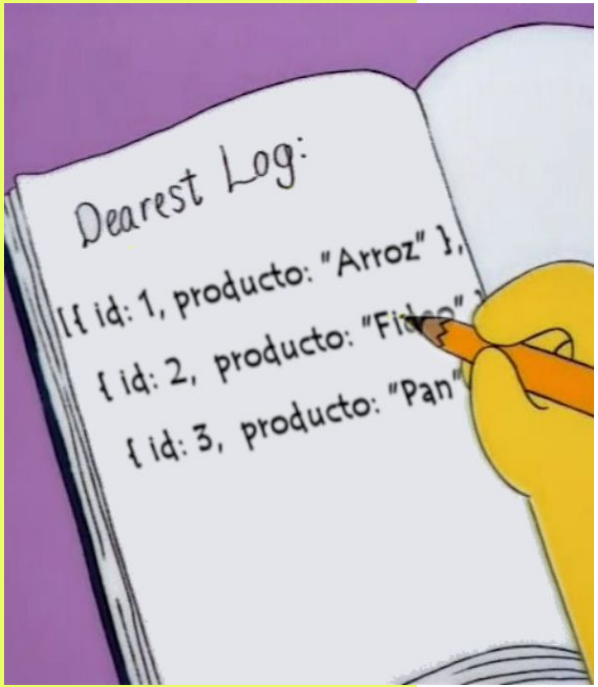
MAPA DE CONCEPTOS



¡Vamos a la clase!

CODERHOUSE

Storage



Storage o almacenamiento

El objeto **Storage** (API de almacenamiento web) permite almacenar datos de manera local en el navegador sin necesidad de realizar ninguna conexión con el servidor. De esta manera, cada cliente puede **preservar información** de la aplicación.
El navegador nos ofrece dos tipos de storage: localStorage y sessionStorage.

CODERHOUSE

Localstorage: Setitem

Los datos almacenados en localStorage (variable global preexistente) se almacenan en el navegador de forma indefinida (o hasta que se borren los datos de navegación del browser):

La información persiste reinicio de navegador y hasta del sistema operativo.

Localstorage: Setitem

Para almacenar información se utiliza setItem:

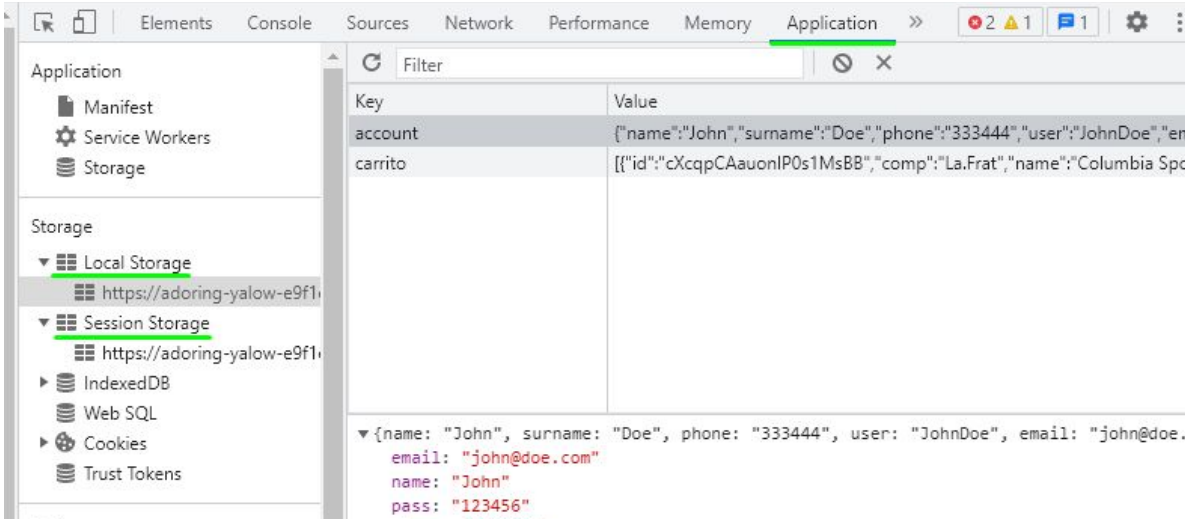
```
// Método ->  localStorage.setItem(clave, valor)
// clave = nombre para identificar el elemento
// valor = valor/contenido del elemento
localStorage.setItem('bienvenida', '¡Hola Coder!');
localStorage.setItem('esValido', true);
localStorage.setItem('unNumero', 20);
```

Clave-valor

La información almacenada en el Storage se guarda en la forma de **clave-valor**.
Similar al tratamiento de objetos, definimos **claves** en el storage donde almacenamos **valores.**

Clave-valor

Podemos ver el Storage en el navegador a través de la pestaña de **application**:



CODERHOUSE

Localstorage: getitem

Podemos acceder a la información almacenada en localStorage utilizando **getItem**. Las claves y valores de Storage se guardan en formato de **cadena de caracteres (DOMString).**

```
let mensaje =  localStorage.getItem('bienvenida');
let bandera =  localStorage.getItem('esValido');
let numero  =  localStorage.getItem('unNumero');

console.log(mensaje); // '¡Hola Coder!'
console.log(bandera); // 'true'
console.log(numero);  // '20'
```

Localstorage: Setitem

La información almacenada en sessionStorage (variable global preexistente) se almacena en el navegador hasta que el usuario cierra la ventana.

Solo existe dentro de la pestaña actual del navegador. Otra pestaña con la misma página tendrá otro sessionStorage distinto, pero se comparte entre iframes en la pestaña (asumiendo que tengan el mismo origen).

Sessionstorage: getitem

El tratamiento es similar al **localStorage**:

```
// Método ->  sessionStorage.setItem(clave, valor)
// clave = nombre del elemento
// valor = Contenido del elemento
sessionStorage.setItem('seleccionados', [1,2,3]);
sessionStorage.setItem('esValido', false);
sessionStorage.setItem('email', 'info@email.com');
```

Sessionstorage: getitem

Podemos acceder a la información almacenada en sessionStorage utilizando **getItem**. Las claves y valores de Storage se guardan siempre en formato de **cadena de caracteres.**

```
let lista   =  sessionStorage.getItem('seleccionados').split(",");
let bandera = (sessionStorage.getItem('esValido') == 'true');
let email   =  sessionStorage.getItem('email');

console.log(typeof lista);   //object ["1","2","3"];
console.log(typeof bandera); //boolean;
console.log(typeof email);   //string;
```

Recorriendo el storage

Es posible obtener todos los valores almacenados en localStorage o sessionStorage con un **bucle**.

Pero no podemos usar **for...of** porque no son objetos iterables, ni **for...in** porque obtenemos otras propiedades del objeto que no son valores almacenados.

Recorriendo el storage

El bucle a emplear es **for** con el método **key**:

```
//Ciclo para recorrer las claves almacenadas en el objeto localStorage
for (let i = 0; i < localStorage.length; i++) {
    let clave = localStorage.key(i);
    console.log("Clave: "+ clave);
    console.log("Valor: "+ localStorage.getItem(clave));
}
```

Eliminar datos del storage

Podemos eliminar la información almacenada en sessionStorage o localStorage usando el **método removeItem** o **clear**:

```
localStorage.setItem('bienvenida', '¡Hola Code!');
sessionStorage.setItem('esValido', true);

localStorage.removeItem('bienvenida');
sessionStorage.removeItem('esValido');
localStorage.clear()    //elimina toda la información
sessionStorage.clear() //elimina toda la información
```

Break

¡10 minutos y volvemos!

JSON

Almacenar objetos en storage

Si queremos almacenar la información de un objeto en un storage, hay que tener en cuenta que tanto la clave como el valor se almacenan en **strings**.
Ante cualquier otro tipo a guardar, como un número o un objeto, se convierte a **cadena de texto automáticamente.**

Almacenar objetos en storage

Entonces, al buscar almacenar un objeto sin una transformación previa, guardamos **[object Object]**, la conversión por defecto de objeto a string. Para guardar la información correctamente hay que transformar el objeto a **JSON**.

```
const producto1 = { id: 2, producto: "Arroz" };
localStorage.setItem("producto1", producto1); // Se guarda [object Object]
```

Acceso tipo objeto

Dado que **localStorage** y **sessionStorage** son objetos globales, es posible crear y acceder a las claves como si fueran propiedades. Pero esto no es recomendable, porque hay eventos asociados a la modificación del storage cuando se emplea **getItem** o **setItem**.

```
//Guarda una clave
localStorage.numeroPrueba = 5;

//Leer una clave
alert( localStorage.numeroPrueba ); // 5

let clave = 'toString';     //toString método reservado
localStorage[clave] = "6"; //No se guarda este dato
```

¿Qué es JSON?

JavaScript Object Notation (JSON) es un **formato basado en texto plano**, para representar datos estructurados con la sintaxis de objetos de JavaScript. Es comúnmente utilizado para enviar y almacenar datos en aplicaciones web.

¿Qué es JSON?

Aunque es muy parecido (casi similar) a la sintaxis de JavaScript, puede ser utilizado independientemente de JavaScript, y muchos entornos de programación poseen la capacidad de leer (convertir; parsear) y generar JSON.
JSON es un string con un formato específico.

Conversión de objetos y almacenamiento

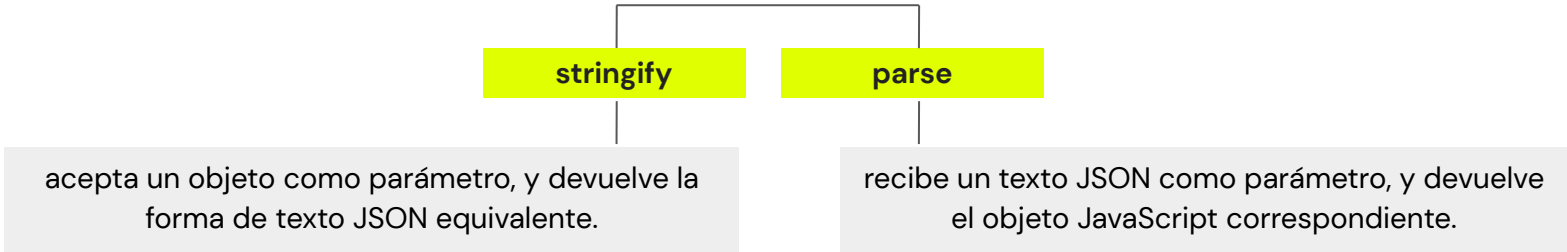
Conversiones de/hacia JSON

Cuando sea necesario enviar un objeto Javascript al servidor o almacenarlo en storage, será necesario convertirlo a un JSON (una cadena) antes de ser enviado.

Conversiones de/hacia JSON

Cuando sea necesario enviar un objeto Javascript al servidor o almacenarlo en storage, será necesario convertirlo a un JSON (una cadena) antes de ser enviado.

Para eso usamos los siguientes métodos:



Stringify

Con **JSON.stringify** podemos transformar un objeto JavaScript a un string en formato JSON.

```
const producto1 = { id: 2, producto: "Arroz" };
const enJSON    = JSON.stringify(producto1);

console.log(enJSON); // {"id":2,"producto":"Arroz"}
console.log(typeof producto1); // object
console.log(typeof enJSON);    // string

localStorage.setItem("producto1", enJSON);
// Se guarda {"id":2,"producto":"Arroz"}
```

Parse

Con **JSON.parse** podemos transformar string en formato JSON a objeto JavaScript.

```
const enJSON    = '{"id":2,"producto":"Arroz"} ';
const producto1 = JSON.parse(enJSON);

console.log(typeof enJSON);      // string
console.log(typeof producto1);   // object
console.log(producto1.producto); // Arroz

const producto2 = JSON.parse(localStorage.getItem("producto1"));
console.log(producto2.id);   // 2
```

Ejemplo aplicado: almacenar array de objetos

```
const productos = [{ id: 1,  producto: "Arroz", precio: 125 },
                   {  id: 2,  producto: "Fideo", precio: 70 },
                   {  id: 3,  producto: "Pan"  , precio: 50},
                   {  id: 4,  producto: "Flan" , precio: 100}];

const guardarLocal = (clave, valor) => { localStorage.setItem(clave, valor) };
//Almacenar producto por producto
for (const producto of productos) {
    guardarLocal(producto.id, JSON.stringify(producto));
}
// o almacenar array completo
guardarLocal("listaProductos", JSON.stringify(productos));
```

CODERHOUSE

Ejemplo aplicado: almacenar array de objetos

```
class Producto {
    constructor(obj) {
        this.nombre  = obj.producto.toUpperCase();
        this.precio  = parseFloat(obj.precio);
    }
    sumaIva() {
        this.precio = this.precio * 1.21;
    }
}
//Obtenemos el listado de productos almacenado
const almacenados = JSON.parse(localStorage.getItem("listaProductos"));
const productos = [];
//Iteramos almacenados con for...of para transformar todos sus objetos a tipo producto.
for (const objeto of almacenados)
    productos.push(new Producto(objeto));
//Ahora tenemos objetos productos y podemos usar sus métodos
for (const producto of productos)
    producto.sumaIva();
```

Recuperar datos

Muchas veces usamos el Storage para recuperar datos relacionados a la última navegación del usuario. Por ejemplo, su última sesión de login o el último estado de su carrito de compras.
Para esto, pensamos en inicializar las variables de la app consultando el Storage en el momento de inicio.

Ejemplo aplicado: recuperar estados previos

```
let usuario;
let usuarioEnLS = JSON.stringify(localStorage.getItem('usuario'))


// Si había algo almacenado, lo recupero. Si no le pido un ingreso
if (usuarioEnLS) {
    usuario = usuarioEnLS
} else {
    usuario = prompt('Ingrese su nombre de usuario')
}
```

Ejemplo aplicado: almacenar array de objetos

```
let carrito = []
let carritoEnLS = JSON.stringify(localStorage.getItem('carrito'))

// Inicializo mi app con carrito como array vacío o con el registro que haya quedado en LS
if (carritoEnLS ) {
    carrito = carritoEnLS
}

// Función que renderizaría el carrito
renderCarrito( carrito )
```

CODERHOUSE

JSON: otros puntos a tener en cuenta

Las datos en formato JSON se pueden almacenar en archivos externos **.json**.
Exemplo: datos.json

JSON es sólo un formato de datos - contiene sólo **propiedades**, no métodos.

Una coma o dos puntos mal ubicados pueden producir que un archivo JSON no funcione. Se debe ser cuidadoso para validar cualquier dato que se quiera utilizar.
https://jsonformatter.curiousconcept.com/

A diferencia del código JavaScript en que las propiedades del objeto pueden no estar entre comillas, en JSON sólo las cadenas entre comillas pueden ser utilizadas como propiedades.

¡Hands on!

CODERHOUSE

Ejercitar JSON y Storage

¡Llevemos lo visto hasta el momento a la acción!
Les proponemos que de manera individual realicen la siguiente actividad.

Duración: **25/30 minutos**

CODERHOUSE

ACTIVIDAD

Ejercitar JSON y Storage

Realiza un algoritmo que almacene información en Storage y guarde un array de objetos en formato JSON.

Deberá cumplir los siguientes requisitos:

- ✓ Almacenar en Storage información ingresada por el usuario. Puede ser un texto, números, o combinación. Luego mostrarla mediante alert o console.
- ✓ Declarar un array de objetos (literales, con funciòn constructora o con clases) y almacenar el array en formato JSON en el storage.
- ✓

Cuentas con 25 minutos para hacer la actividad.

¿Preguntas?

Para pensar

¿Te gustaría comprobar tus conocimientos de la clase?

Te compartimos a través del chat de Zoom
el enlace a un breve quiz de tarea.

Para el profesor:
Acceder a la carpeta “Quizzes” de la camada.
Ingresar al formulario de la clase.
Pulsar el botón “Invitar”.
Copiar el enlace.
Compartir el enlace a los alumnos a través del chat.

CODERHOUSE

MATERIAL AMPLIADO

Recursos multimedia

LocalStorage, sessionStorage

- ✓ Javascript.info

JSON

- ✓ GitBooks. El formato JSON
- ✓ JSON Formatter
- ✓ Generador JSON

Documentación

- ✓ Documentación localStorage
- ✓ Documentación sessionStorage
- ✓ Documentación JSON

Disponible en nuestro repositorio.

Resumen
de la clase hoy

- ✓ Objeto Storage: localStorage y sessionStorage.
- ✓ JSON: concepto y uso en JS.

Opina y valora esta clase

Tu Proyecto final

#CoderTip: Ingresa a la carpeta de Google Drive de esta clase y **revisa el documento de Proyectos finales.** Allí podrás encontrar referencias que te serán útiles al momento de crear tu trabajo original.

Muchas gracias.

#DemocratizandoLaEducación