Recuerden poner a grabar la clase

Clase 13

webStorage

Diplomatura UNTREF

Temario

UNTREF
UNIVERSIDAD NACIONAL
DE TRES DE FEBRERO

Web storage:

- Tipos de almacenamientos en el navegador.
- localStorage y sessionStorage
- Metodos:
  - setItem
  - getItem
  - removeItem
  - clear
- Objeto JSON
- Metodo PARSE y STRINGIFY

WebStorage

Web Storage

En JavaScript es una API que permite a las aplicaciones web almacenar datos en el navegador web de un usuario.

Proporciona una forma sencilla de almacenar información localmente en el dispositivo del usuario, lo que puede ser útil para almacenar preferencias del usuario, datos temporales o cualquier tipo de información que deba persistir entre sesiones.

Tipos de almacenamiento

Tipos de almacenamientos en los navegadores

En el contexto de las aplicaciones web, los navegadores web proporcionan diferentes tipos de almacenamiento para que los desarrolladores puedan guardar datos en el lado del cliente.

Estos tipos de almacenamiento varían en términos de duración, capacidad y accesibilidad.

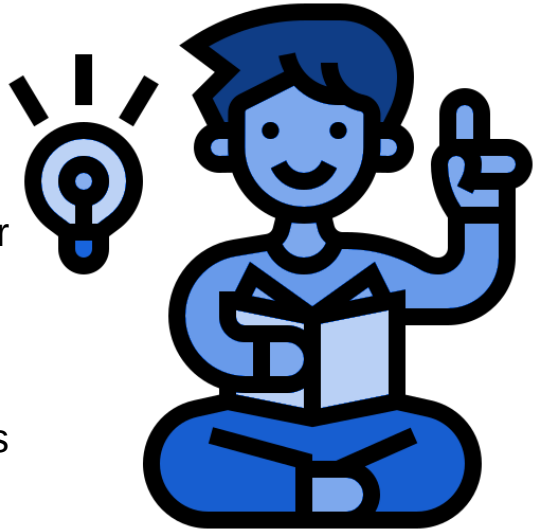
Tipos de almacenamientos en los navegadores

Algunos ejemplos de tipos de almacenamiento son:

Cookies: Las cookies son pequeños fragmentos de datos que los sitios web pueden almacenar en el navegador del usuario. Las cookies se envían con cada solicitud HTTP, lo que las hace adecuadas para almacenar información que debe ser enviada al servidor en cada solicitud, como datos de sesión o preferencias del usuario. Sin embargo, las cookies tienen limitaciones en términos de tamaño (generalmente alrededor de 4 KB) y cantidad (generalmente un sitio web puede almacenar un número limitado de cookies).

Tipos de almacenamientos en los navegadores

localStorage: localStorage es una API que permite a las aplicaciones web almacenar datos en el navegador con un tiempo de vida indefinido. Los datos almacenados en localStorage permanecen incluso después de cerrar el navegador y se mantienen entre sesiones. El espacio de almacenamiento es mayor que el de las cookies, generalmente alrededor de 5-10 MB por dominio.



sessionStorage: sessionStorage es similar a localStorage, pero los datos almacenados en él solo están disponibles durante la duración de una sesión del navegador. Se borran cuando se cierra la ventana o pestaña del navegador. Al igual que localStorage, cada dominio tiene su propio espacio de almacenamiento.

UNTREF
UNIVERSIDAD NACIONAL DE TRES DE FEBRERO

Tipos de almacenamientos en los navegadores

indexedDB: IndexedDB es una base de datos NoSQL en el navegador que permite a las aplicaciones web almacenar grandes cantidades de datos estructurados. Es más potente que las opciones de almacenamiento mencionadas anteriormente, pero también requiere un código más complejo para su manejo.

Cache API: La Cache API permite a las aplicaciones web almacenar recursos (como imágenes, archivos CSS y JavaScript) en la memoria caché del navegador. Esto puede mejorar el rendimiento de la aplicación al permitir la carga rápida de recursos desde la caché en lugar de descargarlos nuevamente del servidor.

localStorage y sessionStorage

LocalStorage

localStorage es una forma de almacenamiento de datos en el navegador web que permite a las aplicaciones web guardar información de forma persistente.

Los datos almacenados en localStorage persisten incluso después de que el usuario cierre el navegador y vuelva a abrirlo en sesiones futuras.

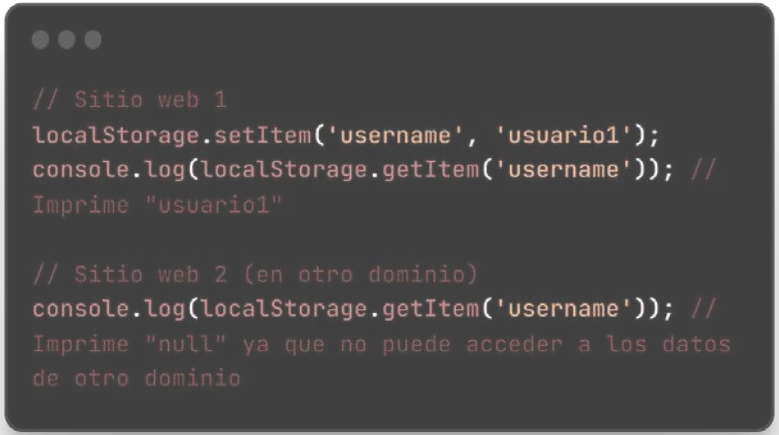
Entre sus características claves tenemos:

- Alcance de dominio
- Capacidad
- Persistencia

localStorage: alcance de dominio

Cada dominio web tiene su propio espacio de almacenamiento **localStorage**, lo que significa que los datos almacenados en **localStorage** por un sitio web no son accesibles por otros sitios web debido a las políticas de seguridad del navegador.

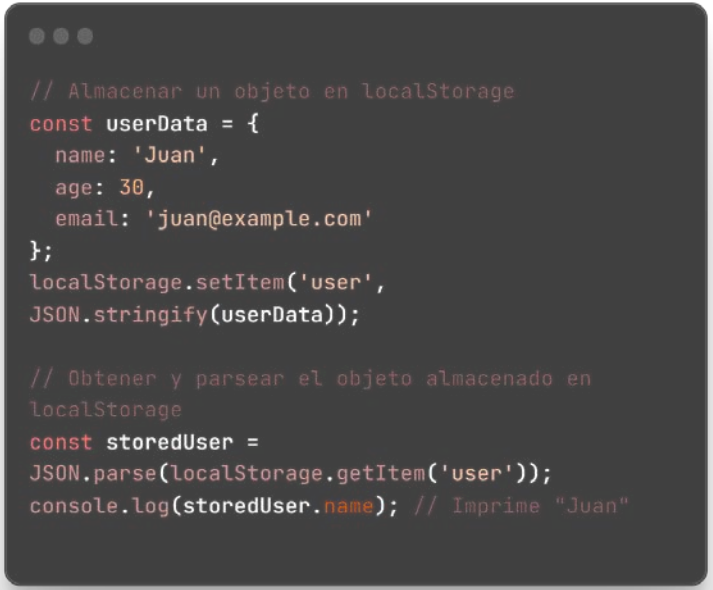
Esto ayuda a garantizar que la información sensible o personal no se comparta inadvertidamente entre diferentes sitios.



localStorage: capacidad

localStorage ofrece más espacio de almacenamiento en comparación con las cookies, lo que lo hace adecuado para guardar cantidades moderadas de datos.

Sin embargo, debido a su límite de tamaño (generalmente alrededor de 5-10 MB por dominio), no se debe abusar de él para almacenar grandes cantidades de datos.



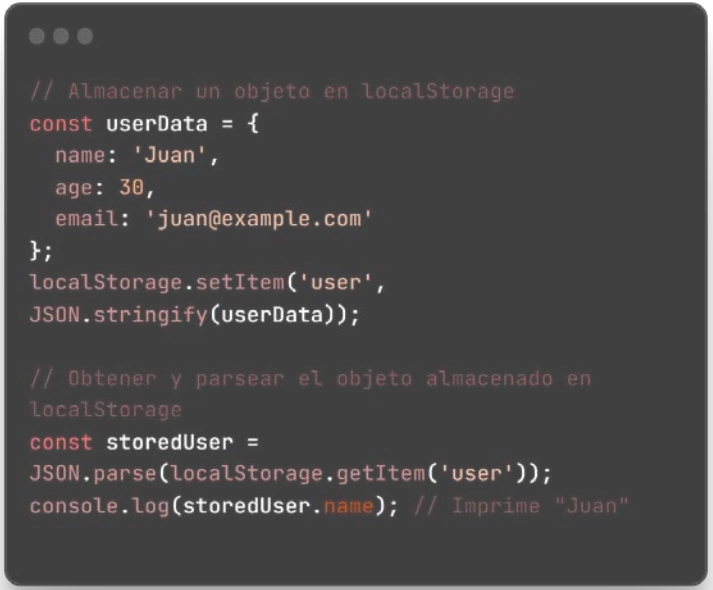
```
// Almacenar un objeto en localStorage
const userData = {
  name: 'Juan',
  age: 30,
  email: 'juan@example.com'
};
localStorage.setItem('user',
JSON.stringify(userData));

// Obtener y parsear el objeto almacenado en
localStorage
const storedUser =
JSON.parse(localStorage.getItem('user'));
console.log(storedUser.name); // Imprime "Juan"
```

localStorage: capacidad

Los datos almacenados en **localStorage** son persistentes y no tienen un tiempo de expiración predeterminado.

Esto significa que los datos permanecen en el navegador a menos que el usuario los elimine explícitamente o el sitio web los borre mediante programación.



```
// Almacenar un objeto en localStorage
const userData = {
  name: 'Juan',
  age: 30,
  email: 'juan@example.com'
};
localStorage.setItem('user',
JSON.stringify(userData));

// Obtener y parsear el objeto almacenado en
localStorage
const storedUser =
JSON.parse(localStorage.getItem('user'));
console.log(storedUser.name); // Imprime "Juan"
```

localStorage

En resumen, **localStorage** es una forma conveniente de almacenar datos en el lado del cliente en un navegador web.

Ofrece persistencia, mayor capacidad en comparación con las cookies y un alcance de dominio que asegura la privacidad y seguridad de los datos.

Sin embargo, es importante tener en cuenta que **localStorage** no es adecuado para almacenar datos altamente sensibles o grandes volúmenes de datos debido a sus limitaciones y a consideraciones de seguridad.

sessionStorage

sessionStorage es una API en los navegadores web que permite a las aplicaciones web almacenar datos en el lado del cliente de manera similar a localStorage, pero con una diferencia fundamental en cuanto a su duración.

A diferencia de localStorage, los datos almacenados en sessionStorage están disponibles solo durante la duración de una sesión del navegador.

Alcanse de sessionStorage

Al igual que con localStorage, los datos almacenados en sessionStorage están limitados a un dominio específico.

Cada dominio tiene su propio espacio de almacenamiento sessionStorage, lo que significa que los datos almacenados por un sitio web no son accesibles por otros sitios web debido a las políticas de seguridad del navegador.

Capacidad de sessionStorage

La capacidad de almacenamiento en sessionStorage es similar a la de localStorage, generalmente alrededor de 5-10 MB por dominio.

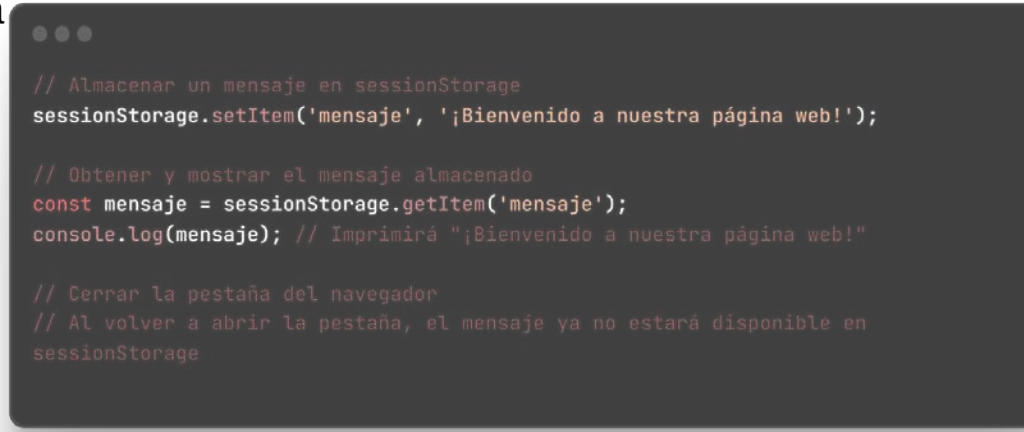
Esto hace que sessionStorage sea adecuado para almacenar cantidades moderadas de datos que son relevantes para la sesión actual del usuario.

Duración de sessionStorage

La característica distintiva de sessionStorage es su duración limitada a la sesión actual del navegador.

Esto significa que los datos almacenados en sessionStorage se mantienen disponibles solo mientras el usuario mantiene abierta la ventana o pestaña del navegador en la que se almacenaron.

Una vez que el usuario cierra la ventana o pestaña, los datos de sessionStorage se borran automáticamente.



Metodos

setItem

setItem

El método **setItem** es una función que forma parte de las API **localStorage** y **sessionStorage**, y se utiliza para **almacenar datos** en el navegador web.

Estas APIs te permiten guardar información localmente en el dispositivo del usuario.

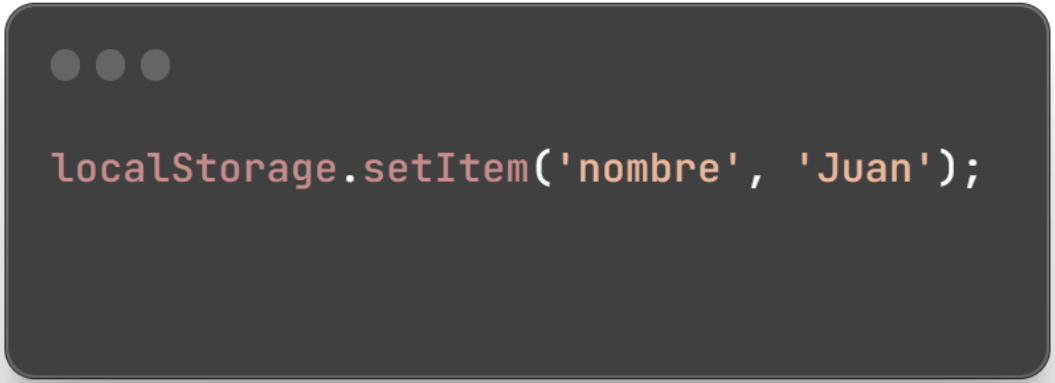
La sintaxis básica es la siguiente:

```
localStorage.setItem(clave, valor);
sessionStorage.setItem(clave, valor);
```

- clave: Un string que actúa como la clave del elemento que deseas almacenar.
- valor: El valor que deseas almacenar. Puede ser un string, un número, un objeto serializado, etc.

setItem Aplicación

Almacenar un nombre en localStorage:



setItem Aplicación

Almacenar un objeto serializado en sessionStorage:

```
const usuario = {
  nombre: 'Ana',
  edad: 25,
  email: 'ana@example.com'
};

sessionStorage.setItem('usuario', JSON.stringify(usuario));
```

setItem Aplicación

Guardar un contador en localStorage:

```
let contador = 0;

function incrementarContador() {
  contador++;
  localStorage.setItem('contador', contador.toString());
}
```

SetItem conclusión

el método setItem es una forma de guardar información en localStorage y sessionStorage. Puedes usarlo para almacenar datos como nombres de usuario, preferencias del usuario, datos temporales o cualquier tipo de información que necesites conservar en el navegador.

getItem

getItem

El método **getItem** es una función que se utiliza en las APIs **localStorage** y **sessionStorage** para **recuperar datos** que previamente hayas almacenado utilizando el método setItem.

Permite obtener el valor asociado a una clave específica en el almacenamiento local del navegador.

clave: Es la cadena que usaste anteriormente como identificador único para el dato que almacenaste.
Al proporcionar esta clave a getItem, obtendrás el valor asociado a esa clave.

getItem Aplicación

Recuperar el nombre almacenado en localStorage:

```
const nombre = localStorage.getItem('nombre');
console.log(nombre); // Imprimirá el valor almacenado, por ejemplo, "Juan"
```

getItem Aplicación

Recuperar y deserializar un objeto almacenado en sessionStorage:

```
const usuarioString = sessionStorage.getItem('usuario');
const usuario = JSON.parse(usuarioString);
console.log(usuario.nombre); // Imprimirá el nombre del usuario, por ejemplo, "Ana"
```

getItem Aplicación

Recuperar el contador almacenado en localStorage:

```
const contadorString = localStorage.getItem('contador');
const contador = parseInt(contadorString);
console.log(contador); // Imprimirá el valor del contador almacenado
```

getItem Conclusión

El **método getItem** es útil para acceder a los datos almacenados previamente en el almacenamiento local del navegador.

Es importante recordar que **el valor recuperado** será una cadena (string) en su forma original.

Si almacenaste objetos u otros tipos de datos más complejos, es posible que necesites realizar conversiones adicionales, como el uso de JSON.parse para los objetos serializados.

getItem es una herramienta esencial para acceder a la información que has almacenado con el método setItem en las APIs de almacenamiento local del navegador.

removeItem

removeItem

El método **removeItem** es una función que se utiliza en las APIs **localStorage** y **sessionStorage** para **eliminar un elemento** específico del almacenamiento local del navegador. Puedes utilizar este método para borrar datos previamente almacenados bajo una clave específica.

clave: La clave (nombre) del elemento que deseas eliminar.
Esto es como decirle al navegador cuál de los datos almacenados deseas quitar.

removeItem Aplicacion

Eliminar un elemento de localStorage:

```
localStorage.setItem('nombre', 'Juan');
console.log(localStorage.getItem('nombre')); // Imprimirá "Juan"

// Ahora, eliminemos 'nombre' de localStorage
localStorage.removeItem('nombre');

console.log(localStorage.getItem('nombre')); // Imprimirá "null" porque 'nombre' ya
no existe
```

removeItem Aplicacion

Eliminar un elemento de sessionStorage:

```
sessionStorage.setItem('tema', 'oscuro');
console.log(sessionStorage.getItem('tema')); // Imprimirá "oscuro"

// Ahora, eliminemos 'tema' de sessionStorage
sessionStorage.removeItem('tema');

console.log(sessionStorage.getItem('tema')); // Imprimirá "null" porque 'tema' ya no
existe
```

removeItem Conclusion

El método **removeItem** es útil cuando deseas borrar datos específicos del almacenamiento local del navegador, ya sea localStorage o sessionStorage. Esto es especialmente útil para limpiar datos que ya no son necesarios o para restablecer preferencias del usuario.

Es importante destacar que removeItem eliminará permanentemente el elemento con la clave proporcionada. Si intentas obtener ese elemento después de usar removeItem, obtendrás null, lo que indica que el elemento ya no existe en el almacenamiento.

removeItem Conclusión

En resumen, **removeItem** es una función útil para eliminar datos específicos del almacenamiento local del navegador, lo que te permite gestionar y limpiar los datos almacenados según sea necesario en tu aplicación web.

clear

clear

El **método clear** es una función que se utiliza para eliminar todos los elementos almacenados en un espacio de almacenamiento específico (ya sea localStorage o sessionStorage) en una sola operación.

No necesita argumentos, simplemente se llama al método en el objeto localStorage o sessionStorage que deseas limpiar.

```
localStorage.clear(); // Elimina todos los elementos en localStorage
sessionStorage.clear(); // Elimina todos los elementos en sessionStorage
```

clear Aplicación

Limpiar localStorage:

```
localStorage.setItem('nombre', 'Juan');
localStorage.setItem('tema', 'claro');

console.log(localStorage.getItem('nombre')); // Imprimirá "Juan"
console.log(localStorage.getItem('tema')); // Imprimirá "claro"

// Ahora, eliminemos todos los elementos en localStorage
localStorage.clear();

console.log(localStorage.getItem('nombre')); // Imprimirá "null" porque 'nombre' ya
no existe
console.log(localStorage.getItem('tema')); // Imprimirá "null" porque 'tema' ya no
existe
```

clear Aplicación

Limpiar sessionStorage:

```
sessionStorage.setItem('tema', 'oscuro');
sessionStorage.setItem('usuario', 'ana');

console.log(sessionStorage.getItem('tema')); // Imprimirá "oscuro"
console.log(sessionStorage.getItem('usuario')); // Imprimirá "ana"

// Ahora, eliminemos todos los elementos en sessionStorage
sessionStorage.clear();

console.log(sessionStorage.getItem('tema')); // Imprimirá "null" porque 'tema' ya no
existe
console.log(sessionStorage.getItem('usuario')); // Imprimirá "null" porque 'usuario'
ya no existe
```

clear Conclusión

El **método clear** es útil cuando deseas **eliminar** todos los datos almacenados en un espacio de almacenamiento local de manera rápida y sencilla.

Puede ser útil en situaciones donde necesitas restablecer la configuración del usuario o limpiar los datos de una sesión anterior.

Es importante tener en cuenta que **clear eliminará todos los elementos**, por lo que **debes usarlo con precaución**, especialmente si tienes datos importantes en el almacenamiento local. Asegúrate de que estás eliminando los datos en el momento adecuado y que no necesitas esos datos en el futuro.

clear Conclusión

En resumen, clear es una función que te permite borrar todos los elementos almacenados en localStorage o sessionStorage de una sola vez, lo que puede ser útil en situaciones donde necesitas eliminar todos los datos almacenados en un espacio de almacenamiento local.

Objeto .JSON

Objeto .JSON

JSON, que significa **"JavaScript Object Notation"** (Notación de Objetos de JavaScript), es un **formato de datos ligero** y muy utilizado para el intercambio de información entre un servidor y un cliente, o entre diferentes partes de una aplicación.

JSON es fácilmente legible tanto para los humanos como para las máquinas, lo que lo hace muy popular en el mundo de la programación y la web.

Objeto .JSON Caracteristicas

Sintaxis Simple: JSON utiliza una sintaxis simple basada en pares clave-valor, similar a la forma en que los objetos literales se definen en JavaScript. Esto lo hace fácil de entender y escribir.

Legibilidad: Los datos en formato JSON son legibles para los humanos, ya que utiliza una estructura jerárquica que se asemeja a los diccionarios y listas.

Objeto .JSON Caracteristicas

Independiente del Lenguaje: JSON es independiente del lenguaje de programación, lo que significa que **se puede utilizar en una amplia variedad de lenguajes** de programación.

Ligero: JSON es un **formato de datos ligero**, lo que significa que no agrega una gran cantidad de sobrecarga de datos al contenido.

Esto es importante cuando **se transfiere información a través de redes,** como en aplicaciones web o servicios web.

Objeto .JSON Caracteristicas

Amplia Adopción: JSON es ampliamente adoptado en la industria de desarrollo de software y es el formato de elección para muchas API web, bases de datos NoSQL y configuraciones de datos en muchas aplicaciones.

Objeto .JSON Estructura

Un **objeto JSON** consiste en una **colección de pares clave-valor.**

Las **claves (keys)** deben estar en comillas dobles y seguidas de dos puntos, y **los valores** pueden ser de varios tipos, incluyendo strings, números, objetos JSON anidados, arreglos, booleanos y null.


```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
}
```

Objeto .JSON Uso

Comunicación entre Cliente y Servidor: En aplicaciones web, es común que los servidores envíen datos al navegador del usuario en formato JSON. Esto permite transmitir información estructurada de manera eficiente.

API Web: Muchas API web devuelven datos en formato JSON. Los desarrolladores pueden solicitar datos a través de solicitudes HTTP y recibir respuestas en formato JSON que luego pueden procesar en sus aplicaciones.

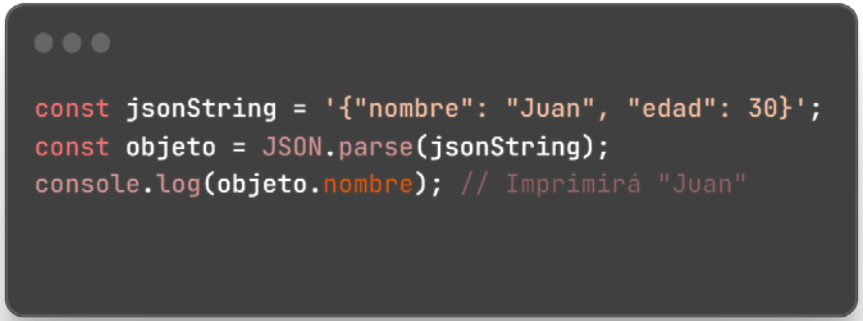
Objeto .JSON Uso

Almacenamiento de Datos: JSON se utiliza a menudo para almacenar configuraciones o datos estructurados en archivos o bases de datos. Esto es común en aplicaciones móviles y de servidor.

Configuración de Aplicaciones: JSON se utiliza para configurar aplicaciones, como especificar opciones de configuración, reglas comerciales y más.

Objeto .JSON Uso

En **JavaScript**, puedes convertir una cadena JSON en un **objeto JavaScript** utilizando el método **JSON.parse()** y viceversa utilizando **JSON.stringify()** para convertir un objeto JavaScript en una cadena JSON.



```
const jsonString = '{"nombre": "Juan", "edad": 30}';
const objeto = JSON.parse(jsonString);
console.log(objeto.nombre); // Imprimirá "Juan"
```

Objeto .JSON Uso

Ejemplo de conversión de un objeto JavaScript
en una cadena JSON:

```
const objeto = { nombre: "Juan", edad: 30 };
const jsonString = JSON.stringify(objeto);
console.log(jsonString); // Imprimirá '{"nombre":"Juan","edad":30}'
```

Metodo PARSE y STRINGIFY

Metodo JSON.parse()

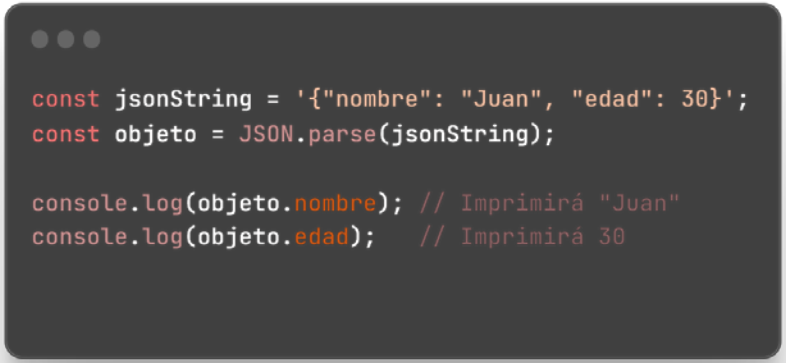
El **método JSON.parse()** se utiliza para convertir una cadena JSON en un objeto JavaScript.

Toma una **cadena JSON** como argumento y devuelve un objeto JavaScript que representa los datos contenidos en esa cadena JSON.

Esto permite que los datos sean utilizados y manipulados en el código JavaScript.

Metodo JSON.parse()

- jsonString es una cadena JSON válida que representa un objeto con claves "nombre" y "edad".
- JSON.parse(jsonString) convierte esta cadena JSON en un objeto JavaScript objeto.
- Puedes acceder a los valores dentro del objeto como lo harías con cualquier otro objeto JavaScript (objeto.nombre, objeto.edad)



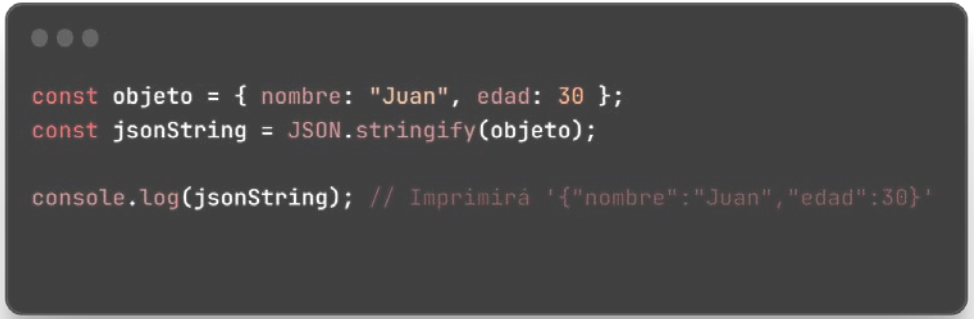
Metodo JSON.stringify()

El **método JSON.stringify()** se utiliza para **convertir un objeto JavaScript** en una cadena JSON.

Toma un **objeto como argumento y devuelve una cadena JSON** que representa ese objeto. Esto es particularmente útil cuando deseas enviar datos estructurados a un servidor o almacenar datos en el almacenamiento local del navegador.

Metodo JSON.stringify()

- objeto es un objeto JavaScript con propiedades "nombre" y "edad".
- **JSON.stringify(objeto)** toma este objeto y lo convierte en una cadena JSON jsonString.



```
const objeto = { nombre: "Juan", edad: 30 };
const jsonString = JSON.stringify(objeto);

console.log(jsonString); // Imprimirá '{"nombre":"Juan","edad":30}'
```

JSON.parse() JSON.stringify() USO

JSON.parse() se utiliza comúnmente cuando **se reciben datos** desde una API web o un servicio web en formato JSON. Permite convertir esos datos en objetos JavaScript que se pueden utilizar en la lógica de la aplicación.

JSON.stringify() es útil cuando **deseas enviar datos estructurados** a un servidor, como enviar un formulario con datos de usuario. También se utiliza para almacenar datos complejos en el almacenamiento local o enviar datos a través de una conexión de red.

JSON.parse() JSON.stringify() Consideraciones

Al utilizar **JSON.parse(),** es importante asegurarse de que la **cadena JSON sea válida**. De lo contrario, se generará un error de análisis.

Al usar **JSON.stringify()**, ten en cuenta que **no todos los objetos JavaScript se pueden convertir en JSON.** Deben consistir en tipos de datos compatibles con JSON, como strings, números, objetos, arreglos, booleanos y null.

¿Dudas o consultas?

¡Muchas Gracias!