<codoacodo/>

Agencia de
Aprendizaje
a lo largo
de la vida

Desarrollo Fullstack

BA

Les damos la bienvenida

Vamos a comenzar a grabar la clase

Agencia de Aprendizaje a lo largo de la vida

Buenos Aires Ciudad Educación

Clase 15

Clase 16

Clase 17

Javascript

- Objetos
- Objetos Literales
- Objetos Funcionales
- Objetos Clases
- Bucle for-in

Javascript Arrays

- **Arrays**
- **Métodos de los Arrays**
- **Storage**
- **JSON**

Javascript Para la Web

- Dom
- Manejo de eventos

JAVASCRIPT

Arrays

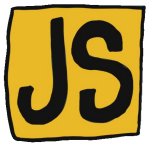
¿Qué es un Array?

- En JavaScript, un array **es una estructura de datos** que se utiliza para **almacenar y organizar elementos**.
- Los elementos en un array pueden ser de **cualquier tipo de datos**, como números, cadenas, objetos, funciones, etc.
- Un array en JavaScript **puede contener elementos de diferentes tipos** en la misma matriz.

Estructura del Array: Declaración.

- Se declara un array utilizando la palabra clave var, let, o const, seguida de un nombre de variable y asignándole un conjunto de elementos entre corchetes, separados por coma [ ].
- Los arreglos poseen un formato de lista.

```
//Ejemplo
var miArray = [“sandía”, 2, “auto”, true, “mosquito”];
```



Arrays: índice y posición.

- Para hacer **referencia** a cada elemento de un array es necesario **indicar la posición** que ocupa en la estructura de almacenamiento.
- Esta **posición** se denomina **índice**.
- **El primer elemento** de un array se almacena en la **posición cero** y el último elemento en la posición n-1, donde **n es el tamaño** del array.

Tipo y longitud de un Array

- En JavaScript, **los arrays son una forma especial de objetos (es un object)** que pueden **contener elementos de diversos tipos de datos,** y su **longitud puede cambiar dinámicamente** a medida que se agregan o eliminan elementos.
- **Propiedad length:** Puedes obtener la longitud de un array utilizando la propiedad length. **La longitud representa la cantidad de elementos en el array.**
- A deferencia de otros lenguajes en JS, **NO se debe declarar la cantidad de elementos a almacenar** en el array antes de ser creado.

Referencias en arrays

- **"Referencias"** se refiere a la **forma en que se accede y se manipula elementos** dentro de un array mediante sus **índices**.
- Al utilizar una referencia, se está haciendo referencia a la **ubicación específica de un elemento en el array**.

Referencias en arrays

- **Notación de corchetes:** <nombre_array>[indice] . Esto devuelve el valor de lo que se encuentra en ese índice, por ello acepta solo valores positivos, por lo que podrá ser almacenado en otra variable o bien impreso por consola. Se debe llamar al método **<array>.length-1 para los últimos elementos.**

- **El método .at():** recibe por argumento un valor numérico entero y **devuelve el elemento en esa posición**, **permitiendo valores positivos y negativos**. Los valores negativos contarán desde el último elemento del array. Se puede llamar al último elemento con **<array>.at(-1).** ***Método incluido desde ECMA 2022***

Referencias en arrays

```
//Ejemplo
var frutas = ["manzana", "banana", "naranja"];
//Acceso al elemento 2, indice 1
console.log(frutas[1]); // imprime "banana"
```

Desafío I

```
/**
 Unidad 16: Arrays y métodos
 DESAFIO I
 Crear un array de 5 elementos
 1- Imprimir por consola el tipo del array
 2- Imprimir por consola el tamaño del array
 3- Imprimir por consola elementos varios del array
 4- Imprimir por consola el primer elemento del array con notacion por corchetes
 5- Imprimir por consola el último elemento del array con notacion por corchetes
 6- Utilizar el método .at() para imprimir por consola el primero y último elemento del array
 */
```

replit

https://replit.com/@Ferlucena/Arrays-y-Metodos#js/array.js

Arrays Methods

Métodos de los arrays

Métodos de los arrays

- Son **funciones nativas** que poseen los arrays y que nos **permiten trabajar** de forma sencilla con los **valores almacenados**, poniendo a disposición herramientas de:
  - Adición.
  - Eliminación.
  - Filtrado.
  - Ordenado.

Métodos para agregar o eliminar elementos

Función	Descripción	Parámetros
`push()`	Agrega uno o más elementos al final de un array y devuelve la nueva longitud del array.	Acepta uno o más parámetros, que son los elementos a agregar al final del array.
`pop()`	Elimina el último elemento de un array y lo devuelve.	No pide parámetros
`unshift()`	Elimina el primer elemento de un array y lo devuelve.	Acepta uno o más parámetros, que son los elementos a agregar al principio del array.
`shift()`	Agrega uno o más elementos al principio de un array y devuelve la nueva longitud del array.	No pide parámetros

Desafío II

```
/**
UNIDAD 16
DESAFIO II - MÉTODOS EN ARRAYS
1- Generar un array de 4 elementos con días laborables.
  [ "martes","martes", "jueves", "jueves"]
  El objetivo es completarlo, corregirlo y llevarlo a ["lunes", "martes", "miercoles", "jueves", "viernes"]
2- Método push() - Agregar un elemento al final del array, devuelve una longitud.
  Agregar días viernes y sábado
3- Método pop() - Eliminar el último elemento del array, devuelve el dato eliminado.
  Eliminar el día sábado.
4- Método shift() - Eliminar el primer elemento del array, devuelve el dato eliminado.
5- Método unshift() - Agregar un elemento al inicio del array, devuelve una longitud.
*/
```

replit

https://replit.com/@Ferlucena/Arrays-y-Metodos#js/metodosArrays.js

Métodos para concatenar, corta y ordenar

Función	Descripción	Sintaxis
`concat()`	Combina 2 o más arrays	`array1.concat(array2)`
`join()`	Crea una cadena de texto a partir de todos los valores de **un solo array**, como argumento se pasa el separador.	`array1.join("-")`
`split()`	Crea un array a partir de una cadena de texto, como argumento se pasa la condición que separa a cada elemento, por ejemplo si están separados por coma.	`array1.split(',')`
`slice()`	Devuelve una porción de un array en un rango definido. Se pasa por argumento el índice de la posición inicial y final de los elementos a cortar. Incluye el primer elemento pero NO el último.	`array1.slice(indice_desde,indice_hasta)`

`sort()`	Ordena **alfabéticamente** los elementos de un array. No funciona con números, ya que aplicado de manera directa presenta un comportamiento erratico	`array1.sort()`
		`//Para números recurrir a la flecha`
		`array1.sort((a,b)=>a-b)`

Desafío III

```
/*
DESAFIO 3
1- Generamos un array con los días no laborables de la semana.
2- Generamos un nuevo array con todos los días de la semana utilizando .concat().
3- Pasamos el array diasDeLaSemana a cadena de texto con .join() y lo guardamos en una
variable.
4- Volvemos a generar un array con todos los días de la semana utilizando .split() y lo
guardamos en una variable.
5- Separamos los días no laborales con .slice() y los guardamos en una variable.
6- Ordenamos alfabeticamente los días de la semana con sort().

*/
```

replit

https://replit.com/@Ferlucena/Arrays-y-Metodos#js/concatSepararOrdenar.js

Arrays Functions

Funciones de los arrays

Funciones de array de orden superior

- Són **métodos de arrays** que originalmente **reciben una función de callback por parámetro** para obtener cierto **resultado.**
- Estos métodos son comúnmente conocidos como métodos **de orden superior o funciones de array de orden superior.**

Iterador forEach

- **forEach():** este método no retorna ningún valor, sino que solo se limita a ejecutar el callback que le pasemos por cada elemento del array.
- forEach() posee un solo parámetro, pero la función que se le pasa por parémetro se puede plantear con uno o dos parámetros

Sintaxis:

- `array.forEach((elemento,index) => {cuerpo_funcion});`
- `array.forEach(index => {cuerpo_funcion});`

Validadores: every( ), some( )

- **every():** verifica si **todos los elementos en el arreglo cumplen una condición dada por la función.** Retorna un valor booleano, **true si todos** los elementos **cumplen con la condición** y **false si al menos uno no cumple**.

Sintaxis:

```
array.every(function(elemento) => {<operación>;return <valor>});

array.every(elemento => <operación>);// Con flecha
```

Validadores: every( ), some( )

- **some():** se utiliza para verificar **si al menos un elemento** de un array **cumple con una condición dada por una función**. Devuelve true si al menos un elemento cumple la condición y false si ninguno la cumple.

Sintaxis:

```
array.some(function(elemento) => {<operación>;return <valor>});

array.some(elemento => <operación>);// Con flecha
```

Reductor: reduce( )

- **reduce():** se **ejecuta por cada elemento** del array y **va acumulando** en una variable, el valor anterior más valor actual de esa iteración.
- Su principio **es como el de una variable acumuladora**.
- **reduce(funcionCallBack,valor_inicial):** la función reduce **acepta dos parámentros**, el primero de ellos se trata de **la función callback** que puede ser **expresada como flecha**, el **segundo parámetro** separado por una coma es un **valor numérico** que refiere al **valor inicial del acumulador**.

Sintaxis:

```
array.some((acumulador,elemento) => <operación>,valor_inicial);
```

Transformador: map( )

- **map():** realiza una transformación en cada elemento del array y **devuelve un nuevo array con los resultados de esas transformaciones.**
- La función map() es especialmente útil cuando se necesita **transformar los elementos de un array** de una manera específica, **sin modificar el array original.**

Sintaxis:

```
// Función flecha de una sola expresión

arrayOriginal.map((elemento, indice, array) => nuevoValor);

// indice, array son opcionales.

arrayOriginal.map((elemento) => nuevoValor);
```

Filtro: filter( )

- **filter():** es otro método de los arrays que **se utiliza para crear un nuevo array** con todos los **elementos que pasan la prueba implementada** por una función proporcionada.

Sintaxis:

```
// Función flecha de una sola expresión

arrayOriginal.filter((elemento, indice, array) => condicion);

// indice, array son opcionales.

arrayOriginal.filter((elemento) => condicion);
```

Buscador: find( )

- **filter():** se utiliza para encontrar el primer elemento que cumple con una condición proporcionada por una función.
- **Una vez que se encuentra** el primer elemento que satisface la condición, find() **devuelve ese elemento**.
- **Si no se encuentra** ningún elemento que cumpla con la condición, find() **devuelve undefined**.

Sintaxis:

```
// Función flecha de una sola expresión

arrayOriginal.find((elemento, indice, array) => condicion);

// indice, array son opcionales.

arrayOriginal.find((elemento) => condicion);
```

Desafío IV

```
/**
DESAFIO 4: Funciones de orden superior en arrays
1- Crear un array de frutas y mediante un bucle forEach, imprimir en consola el nombre de cada fruta y
su indice.
2- Validamos mediante la funcion every() si todos los elementos del array están en minúsculas.
3- Validamos mediante la funcion some() si alguno de los elementos del array contiene la letra "p".
4- Creamos otro array con la función filter, con los elementos del array fruta que contengan letra e.
5- Buscamos mango y reemplazamos por mandarina mediante map() en un nuevo arrray.
6- Contamos la cantidad de letras de nuestro array de frutas mediante reduce() y lo guardamos en una
variable suma.
7- Buscamos alguna fruta con la letra "e" mediante find() y lo guardamos en una variable encontrada.
*/
```

replit

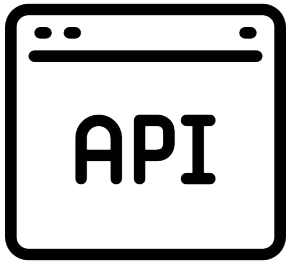
https://replit.com/@Ferlucena/Arrays-y-Metodos#js/funcionesArraySuperior.js

WebStorage

Persistencia de datos en el navegador

Almacenamiento web

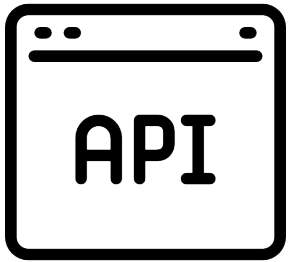
- El almacenamiento web, o Web Storage, es una API de JavaScript que permite a las aplicaciones web almacenar datos de forma persistente en el navegador del usuario.
- Esto significa que los datos almacenados permanecen disponibles incluso después de que el usuario cierre y vuelva a abrir la página web.



Almacenamiento web: tipos

- **localStorage:** Este tipo de almacenamiento permite a las aplicaciones web almacenar datos de forma persistente en el navegador del usuario.
  Los datos almacenados en localStorage permanecen incluso después de que el navegador se cierre y se vuelva a abrir.
  Esto los hace útiles para almacenar configuraciones de usuario, preferencias, datos de sesión, etc. Sin embargo, es importante tener en cuenta que el almacenamiento en localStorage está limitado por el tamaño (generalmente alrededor de 5 MB) y solo puede almacenar datos en formato de cadena.

- **sessionStorage:** A diferencia de localStorage, los datos almacenados en sessionStorage solo están disponibles durante la sesión de navegación actual.
  Una vez que el usuario cierra la pestaña o el navegador, los datos almacenados en sessionStorage se borran.
  Esto hace que sessionStorage sea útil para almacenar datos temporales que solo son relevantes durante la sesión actual del usuario.



Desafío V

```
/**DESAFIO 5: WEBSTORAGE
 * 1- Crea una función que guarde los datos en WebStorage
 * 2- Crea una función que cargue los datos de WebStorage
 */
```

replit

https://replit.com/@Ferlucena/WebStorages?v=1

Objetos Literales

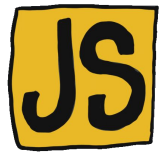
Declaraciones y Formas

Objetos literales

La declaración de objetos literales en JavaScript **es la forma más sencilla y común de crearlos.**

Consiste en **definir un objeto** directamente utilizando la **notación de llaves {}**.

En un objeto literal, se pueden incluir **pares clave-valor** separados por comas, donde las **claves** son **strings (o símbolos desde ECMAScript 6) y los valores pueden ser de cualquier tipo, incluyendo funciones.**



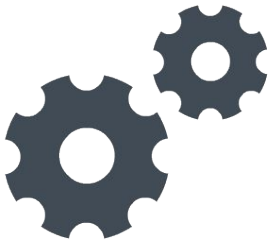
Sintaxis de la declaración de un objeto literal

```
Sintaxis de la declaración.

<const_let_var> <nombre> = {
  <clave1>:<valor1>,
  <clave2>:<valor2>,
  <clave3>:function(){
    //Cuerpo de la función
    }
}
```

- `<const_let_var>:` declaración de la variable.
- **`nombre:`** es el nombre del objeto sanake_case o camelCase.
- **`<clave1>:<valor1>,`** propiedades y valores de cada una.

Métodos

- Se conoce con este nombre a las **funciones que declaremos dentro de un objeto**. Es decir, son funciones que existen solo si existe un objeto que las contenga.
- Se las declara dentro de una propiedad.
- El uso de la palabra reservada “this”, hace referencia al contexto léxico es decir a que la o las propiedades que utiliza es o son parte del mismo objeto y no un valor externo.



Desafío VI

objetosLiterales.js

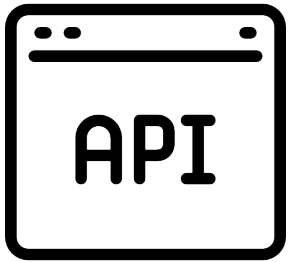
```js
/* Crear un objeto persona que se llame Juan tenga 30 años y pueda dar un saludo
* A partir de este prototipo crear otro objeto persona que se llame María con edad 35 */

//Creación del prototipo
const persona = {
  nombre: "Juan",
  edad: 30,
  saludar: function() {
    console.log(`Hola, mi nombre es ${this.nombre}.`);
  }
};

// Probando el objeto persona
console.log("Mensaje desde persona...");
persona.saludar(); // Output: "Hola, mi nombre es Juan."
console.log("Mi edad es: " + persona.edad); // Output: "Mi edad es 30"

```

JSON

Notación de objetos en JS de intercambio de datos

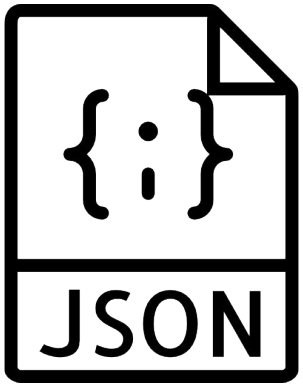
JSON

- JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos que es fácil de leer y escribir para humanos y fácil de analizar y generar para las máquinas.



JSON: sintaxis

- La sintaxis de JSON es muy similar a la de los objetos literales en JavaScript.
- Ambas tienen pares de "clave: valor", donde las claves están entre comillas en JSON pero no necesariamente en objetos literales de JavaScript.
- JSON también permite el uso de arrays, strings, números, booleanos y null. Esta similitud hace que sea fácil para los desarrolladores de JavaScript trabajar con JSON, ya que la sintaxis es familiar.



Sintaxis de la declaración de un objeto JSON

```
Sintaxis de la declaración.

{
  <clave1>:<valor1>,
  <clave2>:<valor2>,
  <clave3>:function(){
    //Cuerpo de la función
    }
}
```

- **`<clave1>:<valor1>,`** propiedades y valores de cada una.
- NOTA: JSON es simplemente un formato de intercambio de datos, no un sistema de archivos. La forma en que se referencia y se utiliza un objeto JSON depende del contexto en el que se esté utilizando.
- A diferencia de un objeto **no necesita declararse dentro de una variable.**

Repo de clases

https://replit.com/@Ferlucena/Arrays-y-Metodos#index.html

Herramientas que utilizamos en clases

VSCode+plugins

No te olvides de dar el presente

Recordá:

- Revisar la Cartelera de Novedades.
- Hacer tus consultas en el Foro.
- Realizá los ejercicios obligatorios.

Todo en el Aula Virtual.