Diplomatura en

# Programación FullStack Developer

GLOBALACADEMY

## JavaScript
## Datos estructurales y sus métodos

# ¿Qué son los Datos Estructurales?

- Los datos estructurales se refieren a aquellos que pueden almacenar múltiples valores y permiten la organización y gestión de estos datos de manera eficiente.

- Son fundamentales para manejar colecciones de datos y realizar operaciones complejas sobre ellos en programación.

**Importancia en Programación**

- Permiten la representación de información en formatos que son más cercanos a cómo los humanos perciben y organizan la información en la vida real.

- Facilitan la implementación de estructuras de datos más complejas como listas, tablas, grafos, etc.

- Son esenciales para el manejo de datos en aplicaciones, permitiendo realizar operaciones como búsqueda, inserción, eliminación, y más de manera eficiente.

**JavaScript y los Datos Estructurales**

- En JavaScript, los principales tipos de datos estructurales son los arrays y los objetos.
- Estas estructuras son altamente flexibles y se utilizan en casi todos los aspectos de la programación en JavaScript, desde la gestión de estados en aplicaciones web hasta la manipulación de datos en servidores.

## Arrays

Un array es un tipo de objeto que permite almacenar múltiples valores en una sola variable. Los valores se almacenan en una secuencia ordenada y se pueden acceder mediante un índice numérico.

```
let frutas = ["manzana", "banana", "naranja", "kiwi", "uva"]
```

**Indexación Base Cero:** El primer elemento de un array está en el índice 0, lo que es importante recordar al acceder a los elementos del array.

**Homogéneos o Heterogéneos:** Aunque generalmente se usan para almacenar elementos del mismo tipo, los arrays en JavaScript pueden contener elementos de diferentes tipos, incluyendo otros arrays y objetos.

## Acceso Directo por Índice

Es la forma más básica de acceder a un elemento en un array. Los índices de los arrays en JavaScript comienzan en cero.

```
console.log(frutas[0]); // Muestra: manzana
```

## Longitud

La propiedad .length es extremadamente útil y se utiliza frecuentemente para iterar sobre arrays con bucles, verificar si un array está vacío, o simplemente para obtener la cantidad de elementos presentes en el array.

```
console.log(frutas.length); // Muestra: 5
```

**Métodos** incorporados que permiten realizar manipulaciones complejas y gestionar eficientemente los datos que contienen.

push(): Añade uno o más elementos al final del array y devuelve la nueva longitud.

```
frutas.push('mandarina') // ['manzana', 'banana', 'naranja', 'kiwi', 'uva', 'mandarina']
```

pop(): Elimina el último elemento de un array y lo devuelve.

```
frutas.pop() // ['manzana', 'banana', 'naranja', 'kiwi']
```

shift(): Elimina el primer elemento de un array y lo devuelve.

```
frutas.shift() //['banana', 'naranja', 'kiwi', 'uva']
```

splice(): cambia el contenido de un array eliminando, reemplazando o añadiendo elementos.

```
frutas.splice(2, 2);
console.log(frutas); // Muestra: ["manzana", "banana", "uva"]
```

join(): Une todos los elementos de un array en una cadena y la devuelve.

```
frutas.join(' - ') // "banana - naranja - kiwi - uva "
```

slice(): Devuelve una copia superficial de una porción del array en un nuevo array.

```
frutas.slice(1, 3)// ['naranja', 'kiwi']
```

reverse(): Invierte el orden de los elementos de un array.

```
frutas.reverse() // ['uva', 'kiwi', 'naranja', 'banana']
```

concat(): Combina dos o más arrays y devuelve un nuevo array.

```
let verduras = ['papa', 'cebolla']

frutas.concat(verduras) // [ 'uva', 'kiwi', 'naranja', 'banana', 'papa', 'cebolla' ]
```

sort(): Ordena los elementos de un array in situ y devuelve el array.

```
frutas.sort() // [ 'banana', 'kiwi', 'naranja', 'uva' ]
```

# Objetos

Los objetos en JavaScript funcionan esencialmente como contenedores estructurados donde cada elemento de información se almacena como una propiedad definida por un par clave-valor. Este sistema permite una organización clara y eficiente de los datos, facilitando su acceso y manipulación a través de las claves.

```
let libro = {
    titulo: "Rayuela",
    autor: "Julio Cortázar",
    anio: 1963
};
```

Aquí, titulo, autor y anio son claves que ayudan a identificar los valores correspondientes. Los objetos son altamente flexibles, permitiéndote añadir, modificar o eliminar propiedades según sea necesario, lo que los hace fundamentales para el manejo de datos dinámicos en aplicaciones.

## Notación de Puntos

La notación de puntos es el método más sencillo y directo para acceder a las propiedades de un objeto cuando se conoce el nombre fijo de la propiedad. Se emplea un punto seguido del identificador de la propiedad, sin comillas.

```
let persona = {
    nombre: "Juan",
    edad: 28
};
console.log(persona.nombre); // Muestra: Juan
console.log(persona.edad);   // Muestra: 28
```

**Cuándo usarla:**

- **Propiedades con nombres estáticos:** Cuando el nombre de la propiedad no cambia y es un identificador válido en JavaScript (sin espacios, no comienza con un número, y no incluye caracteres especiales).
- **Claridad y simplicidad:** La notación de puntos es más legible y directa, lo cual es útil en contextos donde la claridad del código es prioritaria.
- **Acceso directo a propiedades:** Ideal para acceder a propiedades conocidas de un objeto en operaciones simples y directas.

## Notación de Corchetes

La notación de corchetes se utiliza cuando el nombre de la propiedad es una cadena de caracteres que no cumple con las reglas normales para identificadores, como cuando incluye espacios, caracteres especiales, o comienza con un número. También es necesaria cuando el nombre de la propiedad se determina dinámicamente.

```
let persona = {
    nombre: "Juan",
    "lugar de nacimiento": "Buenos Aires",
    1: "Número uno"
};
console.log(persona["nombre"]);                 // Muestra: Juan
console.log(persona["lugar de nacimiento"]);    // Muestra: Buenos Aires
console.log(persona[1]);                        // Muestra: Número uno
```

**Cuándo usarla:**

- **Propiedades con nombres dinámicos o variables:** Cuando el nombre de la propiedad es una variable o se determina en tiempo de ejecución, la notación de corchetes permite insertar el nombre de la propiedad como una cadena de texto.
- **Nombres de propiedades no convencionales:** Si el nombre de la propiedad incluye espacios, caracteres especiales, o comienza con un número, debes usar la notación de corchetes.
- **Acceso más flexible:** Permite realizar iteraciones sobre las propiedades de un objeto o acceder a propiedades cuyos nombres son resultado de operaciones o concatenaciones.

# Métodos de objetos

```
let persona = {
    nombre: "Carlos",
    edad: 30
};
```

### Object.keys(objeto)

Devuelve un array con las claves de todas las propiedades propias (no heredadas) enumerables de un objeto.

```
console.log(Object.keys(persona)); // ["nombre", "edad"]
```

### Object.values(objeto):

Retorna un array con los valores de todas las propiedades propias enumerables de un objeto.

```
console.log(Object.values(persona)); // ["Carlos", 30]
```

**Object.entries(objeto):**

Retorna un array de arrays, donde cada sub-array es un par clave-valor del objeto.

```
console.log(Object.entries(persona)); // [["nombre", "Carlos"], ["edad", 30]]
```

**Object.assign(objetoDestino, objetoFuente):**

Se utiliza para copiar las propiedades de uno o más objetos fuente a un objeto destino. Si una propiedad existe tanto en el destino como en la fuente, el valor de la fuente sobrescribirá el del destino.

```
let objetoDestino = { color: "rojo", tamanio: "grande" };
let objetoFuente = { color: "azul", forma: "circular" };

Object.assign(objetoDestino, objetoFuente);

console.log(objetoDestino);
// Salida: { color: "azul", tamanio: "grande", forma: "circular" }
```

**Object.freeze(objeto):**

Congela un objeto, lo que significa que no se pueden agregar nuevas propiedades, cambiar las existentes o eliminarlas.

```
console.log(Object.entries(persona)); // [["nombre", "Carlos"], ["edad", 30]]
```

**Object.seal(objeto):**

Sella un objeto, lo que impide agregar nuevas propiedades y marca todas las propiedades existentes como no configurables. Las propiedades existentes aún pueden ser modificadas.

```
let ajustes = { volumen: 70 };
Object.seal(ajustes);
ajustes.volumen = 75; // Esto es permitido
ajustes.modo = "silencio"; // No se añadirá
console.log(ajustes); // { volumen: 75 }
```

**Object.freeze(objeto):**

Congela un objeto, lo que significa que no se pueden agregar nuevas propiedades, cambiar las existentes o eliminarlas.

```
console.log(Object.entries(persona)); // [["nombre", "Carlos"], ["edad", 30]]
```

**Object.seal(objeto):**

Sella un objeto, lo que impide agregar nuevas propiedades y marca todas las propiedades existentes como no configurables. Las propiedades existentes aún pueden ser modificadas.

```
let ajustes = { volumen: 70 };
Object.seal(ajustes);
ajustes.volumen = 75; // Esto es permitido
ajustes.modo = "silencio"; // No se añadirá
console.log(ajustes); // { volumen: 75 }
```

# Combinando Array y Objetos

### Arrays de Objetos

Esta es probablemente la combinación más común y útil. Cada elemento del array es un objeto, lo que permite almacenar y gestionar una lista de registros o entidades, donde cada entidad tiene múltiples atributos o propiedades.

```
let libros = [
    { titulo: "Cien años de soledad", autor: "Gabriel García Márquez", anio: 1967 },
    { titulo: "El Aleph", autor: "Jorge Luis Borges", anio: 1949 },
    { titulo: "La invención de Morel", autor: "Adolfo Bioy Casares", anio: 1940 }
];
```

**Uso:**

- Gestión de datos complejos: Ideal para manejar datos que pertenecen a una categoría con múltiples atributos, como productos en un catálogo, usuarios en un sistema, etc.

- Facilidad de iteración y búsqueda: Permite usar métodos de array para filtrar, ordenar y buscar datos basados en uno o más atributos de los objetos.

## Objetos con Arrays como Valores

Los objetos utilizan arrays como valores de sus propiedades. Esto es útil para representar datos donde cada clave puede tener múltiples valores.

```
let receta = {
    nombre: "Pizza",
    ingredientes: ["masa", "salsa de tomate", "queso"]
};

console.log(receta.ingredientes.join(", ")); // Muestra: masa, salsa de tomate, queso
```

**Uso:**

- Representación de colecciones de datos: Útil para cuando una propiedad puede contener múltiples valores o variantes.
- Manipulación de grupos de datos: Permite aplicar métodos de arrays para gestionar los valores de una manera que sería imposible con un valor único.

**Array de Arrays (Matriz)**

Aunque técnicamente no combina objetos y arrays, esta estructura se usa para representar datos en forma de tabla o matriz.

```
let matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
];
console.log(matriz[1][2]); // Muestra: 6
```

**Uso:**

- Representación de datos tabulares: Ideal para datos que se organizan en filas y columnas, como una hoja de cálculo.
- Operaciones matemáticas y transformaciones: Utilizado en cálculos matemáticos, juegos de tablero, simulaciones, etc.