**Aplicación de Tecnologías Web**

Tema n.° 6

Estructuras de datos en JavaScript

**Índice**

**Pág.**

[Introducción 3](#_Toc161316633)

[1.6. Arrays 4](#_Toc161316634)

[1.6.1. Arrays Multidimensionales 6](#_Toc161316635)

[1.7. Conjuntos 7](#_Toc161316636)

[1.8. Mapas 9](#_Toc161316637)

[Recursos complementarios 11](#_Toc161316638)

[Referencias 11](#_Toc161316639)

[Autoevaluación 12](#_Toc161316640)

# Introducción

En el campo de la programación, la elección y manipulación eficiente de estructuras de datos son cruciales para el desarrollo de aplicaciones robustas y eficaces. En esta sección, exploraremos algunas de las estructuras de datos más fundamentales y ampliamente utilizadas en JavaScript: los vectores o arrays, los conjuntos y los mapas.

Los arrays, también conocidos como vectores, son estructuras de datos que permiten almacenar elementos de manera secuencial. En JavaScript, los arrays pueden contener cualquier tipo de dato, desde números y cadenas hasta objetos complejos. Son ideales para organizar y acceder a datos de forma eficiente, ya que cada elemento se accede mediante un índice. Exploraremos cómo declarar, acceder, modificar y manipular arrays, así como las numerosas funciones integradas que facilitan su gestión.

Los conjuntos, representados por la estructura de datos Set en JavaScript, ofrecen una colección de valores únicos. A diferencia de los arrays, los sets no permiten duplicados, proporcionando una herramienta efectiva para garantizar la unicidad de elementos. Abordaremos la creación de sets, la adición y eliminación de elementos, así como las operaciones comunes entre sets, como la unión e intersección.

Los mapas son estructuras de datos que almacenan asociaciones clave-valor, proporcionando una manera eficiente de buscar y recuperar información. En JavaScript, los mapas son versátiles y permiten cualquier tipo de dato tanto para las claves como para los valores. Exploraremos cómo declarar, modificar y acceder a elementos en un mapa, así como las funciones integradas que facilitan la gestión de estas asociaciones.

# Arrays

Lo primero que debemos conocer sobre estos tres tipos de estructuras es que son objetos. En JavaScript existen diferentes tipos de datos básicos. Pues bien, todo aquello que no es un dato básico es un objeto. La diferencia principal entre los objetos y los datos básicos es la forma de almacenarlos y manejarlos.

Los datos básicos se manejan como valores independientes (varias variables tienen una copia personal de un mismo dato), pero los objetos se manejan por referencia de manera que distintas variables pueden apuntar a un mismo pedazo de información.

**Un vector (en adelante, array)** es un conjunto de valores indexado. Esto quiere decir que cada elemento del array tiene una posición concreta, definida por un índice y que, salvo que se produzcan modificaciones, se mantendrá durante toda su vida útil.

Podemos crear un array de diversas maneras, dependiendo de si queremos inicializarlo vacío o con datos.

**Figura 1**

Creación de arrays.



Todos los elementos de un array están indexados y podemos utilizar el índice asociado a cada elemento para obtenerlo. Por ejemplo, en el array creado antes, podemos obtener el valor del primer elemento de este, accediendo al elemento cero de la estructura (recordemos que el cero se refiere al primer elemento):

**Figura 2**

Acceso a elementos de un array



Utilizando la misma notación es posible añadir un nuevo elemento al array, haciendo uso de nuevo del operador de asignación (=):

**Figura 3**

Operador de asignación.



Los *arrays* pueden contener cualquier tipo de datos en su interior, incluso otros *arrays*, lo que daría lugar a ***arrays* multidimensionales o matrices.** Sin embargo, aquellas variables declaradas con const no podrían ver modificado su valor. Entonces, ¿por qué hemos podido añadir un nuevo elemento al *array*, modificándolo?

**Figura 4**

Arrays y referencias



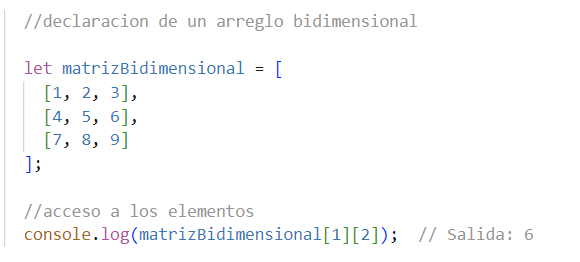
A pesar de declarar un objeto como constante, es posible modificar sus contenidos. El contenido de un array declarado como const sigue siendo mutable.

## Arrays Multidimensionales

Un array multidimensional en JavaScript es una estructura que contiene otros arrays. Esto permite organizar datos de manera más compleja y estructurada.

**Figura 5**

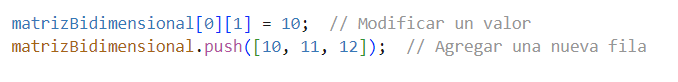
Declaración de un arreglo bidimensional



Podemos realizar operaciones como añadir o eliminar filas y columnas, así como modificar valores en una matriz mediante las siguientes operaciones.

**Figura 6**

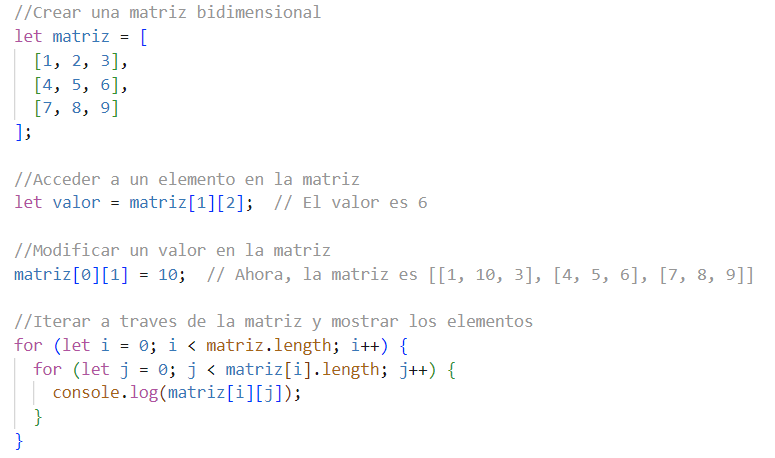
Actualización de los elementos de un array.



La iteración de este tipo de arreglos se la realiza mediante la estructura for.

**Figura 7**

Recorrido de un array



# Conjuntos

Los conjuntos fueron introducidos en la versión 6 de JavaScript. Un conjunto o set es una estructura que representa una colección de valores únicos (no se pueden repetir).

Los sets permiten manejar conjuntos de datos y asegurarnos de manera automática que no contiene duplicados. Para crear un set vacío recurrimos a una función constructora y al operador new:

**Figura 8**

Declaración de conjunto.



Para añadir un elemento a un set utilizamos el método. **add(elemento).**

Figura 9

Añadir elementos a un conjunto.



Este método devuelve nuevamente el set con el nuevo elemento añadido, por lo que es posible encadenar nuevos.add(). Ejemplo:

**Figura 10**

Añadir elementos a un conjunto



A pesar de haber declarado la variable conjunta como constante, podemos modificar el set. Esto es así porque los **sets son también objetos** y estos sí son modificables a pesar de que hayan sido introducidos mediante la cláusula const. ¿Qué pasa si intentamos añadir un elemento que ya existe en el set? Si intentamos añadir un elemento que ya estaba presente en el set, **la orden es ignorada.** Podemos añadir varios elementos de una vez a un set con un *array*:

**Figura 11**

Creación de un conjunto a partir de un array.



Si dicho array contuviese elementos ya repetidos, estos son automáticamente filtrados para eliminar duplicidades. Ejemplo:

**Figura 12**

Los conjuntos no contienen duplicados.



# Mapas

Los mapas no son más que listas de parejas clave-valor, donde la clave y el valor pueden ser cualquier tipo de dato. En otros lenguajes de programación, son conocidos también como mapas hash, tablas o diccionarios. En JavaScript, los mapas también son objetos.

Las claves de un mapa deben ser únicas (algo que ocurre también en los sets). No cuentan con una forma literal para su definición, por lo que para crearlos es necesario usar una función constructora.

**Figura 13**

Construcción de un mapa.



Al igual que ocurría con los sets, los mapas tienen distintos métodos que nos permiten trabajar con ellos. Los más importantes son:

* Método set (clave, valor), que permite añadir una nueva pareja clave-valor al mapa.
* Método get (clave), a partir de la clave, nos devuelve su valor asociado, si existe.
* Método has (clave), que devolverá true o false en función de si el mapa contiene alguna pareja con la clave indicada.
* Método delete (clave), el cual permitirá borrar una entrada del mapa que tenga la clave especificada.
* Método clear (), que permitirá vaciar completamente el mapa.

Las siguientes líneas de código ilustran el funcionamiento de estos métodos:

**Figura 14**

Métodos sobre mapas.



# Recursos complementarios

* Documentación de JavaScript

<https://developer.mozilla.org/es/docs/Web/JavaScript>

# Referencias

Brown, A. R., & Wilson, S. (2013). *JavaScript Testing with Jasmine: JavaScript Behavior-Driven Development.* O'Reilly Media.

Crockford, D. (2008)*. JavaScript: The Good Parts*. O'Reilly Media.

Duckett, J. (2011). *JavaScript & jQuery*. John Wiley & Sons Books.

Flanagan, D. (2011). *JavaScript: The Definitive Guide: Activate Your Web Pages (6th ed.).* O'Reilly Media.

Freeman, A., & Robson, S. (2012). *Head First JavaScript Programming: A Brain-Friendly Guide.* O'Reilly Media.

McPeak, J., Kuksenok, K., & Robinson, C. (2012). *Learning Node: Moving to the Server-Side*. O'Reilly Media.

Osmani, A. (2013). *Developing Backbone.js Applications*. O'Reilly Media.

Resig, J., Bibeault, B., & Marconcini, J. (2013). *Secrets of the JavaScript Ninja*. Manning Publications.

# Autoevaluación

1. **¿Cómo se accede al tercer elemento de un array en JavaScript?**

array(3)

array[3]

array.third

array[2]

1. **¿Cuál de las siguientes opciones agrega un elemento al final de un array?**

array.insertLast(elemento)

array.push(elemento)

array.append(elemento)

array.addToEnd(elemento)

1. **¿Cuál es la función del método splice en un array?**

Elimina elementos de un array

Añade elementos a un array

Modifica elementos en un array

Todas las anteriores

1. **¿Qué característica define a un conjunto en JavaScript?**

Puede contener elementos duplicados

Solo permite números como elementos

Contiene únicamente valores únicos

Es una estructura ordenada

1. **¿Cómo se elimina un elemento de un conjunto en JavaScript?**

conjunto.remove(elemento)

conjunto.delete(elemento)

conjunto.removeAt(elemento)

conjunto.erase(elemento)

1. **¿Cuál es la principal ventaja de usar un conjunto en lugar de un array para almacenar datos únicos?**

Mayor flexibilidad

Menor consumo de memoria

Rápida búsqueda de elementos

Facilidad de iteración

1. **¿Cuál es el propósito principal de un mapa en JavaScript?**

Almacenar elementos ordenados

Contener únicamente números

Asociar claves con valores

Realizar operaciones matemáticas

1. **¿Cómo se agrega una nueva asociación clave-valor a un mapa?**

map.add(clave, valor)

map.insert(clave, valor)

map.set(clave, valor)

map.append(clave, valor)

1. **¿Cuál es la diferencia entre map.get(clave) y map.has(clave) en un mapa?**

map.get(clave) devuelve el valor, map.has(clave) verifica si la clave existe

Ambos devuelven el valor asociado a la clave

map.get(clave) verifica si la clave existe, map.has(clave) devuelve el valor

Ambos verifican si la clave existe

1. **¿Cuál es la forma correcta de declarar un array vacío en JavaScript?**

let miArray = {}

let miArray = []

array miArray = new Array()

array miArray = empty