Arrays

Creación y manipulación de arrays

Un array es una estructura de datos que permite almacenar múltiples valores en una sola variable. Cada elemento del array puede ser de cualquier tipo de dato, como números, cadenas, objetos o incluso otros arrays. Son muy útiles para organizar y manipular grandes cantidades de información de forma eficiente y se declaran utilizando corchetes `[]`.

```javascript
const frutas = ['manzana', 'banana', 'naranja'];
console.log(frutas); // ['manzana', 'banana', 'naranja']
```

Métodos básicos de arrays

Los métodos básicos de arrays son funciones incorporadas en JavaScript que permiten realizar operaciones comunes sobre arrays, como agregar, eliminar o modificar elementos. Entre los métodos más utilizados se encuentran push, pop, shift y unshift, que facilitan la manipulación del contenido de un array de forma eficiente y sencilla. Estos métodos son fundamentales para trabajar con datos almacenados en esta estructura.

- `.push()`: Este método añade un nuevo elemento al final del array y devuelve la nueva longitud del array.
- `.pop()`: Elimina el último elemento del array y lo devuelve.
- `.shift()`: Elimina el primer elemento del array y lo devuelve, desplazando el resto de los elementos hacia la izquierda.
- `.unshift()`: Añade un nuevo elemento al inicio del array y devuelve la nueva longitud del array.

```javascript
const frutas = ['manzana', 'banana'];

frutas.push('naranja');
console.log(frutas); // ['manzana', 'banana', 'naranja']

frutas.pop();
console.log(frutas); // ['manzana', 'banana']

frutas.shift();
console.log(frutas); // ['banana']

frutas.unshift('pera');
console.log(frutas); // ['pera', 'banana']
```

Otros métodos importantes

- `.slice()`: Devuelve una copia superficial de una parte del array, sin modificar el array original. Recibe como parámetros el índice de inicio y, opcionalmente, el de fin.

```javascript
const numeros = [1, 2, 3, 4, 5];
const subArray = numeros.slice(1, 4);
console.log(subArray); // [2, 3, 4]
```

- `.splice()`: Permite agregar, eliminar o reemplazar elementos en el array. Modifica el array original. Recibe el índice de inicio, la cantidad de elementos a eliminar y, opcionalmente, nuevos elementos a agregar.

```javascript
const numeros = [1, 2, 3, 4, 5];
numeros.splice(2, 1, 10);
console.log(numeros); // [1, 2, 10, 4, 5]
```

- `.join()`: Une todos los elementos del array en una cadena, separándolos por un delimitador especificado.

```javascript
const palabras = ['Hola', 'mundo'];
const frase = palabras.join(' ');
```

```
console.log(frase); // "Hola mundo"
```

- `.concat()`: Combina dos o más arrays y devuelve un nuevo array sin modificar los originales.

```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
const combinado = array1.concat(array2);
console.log(combinado); // [1, 2, 3, 4, 5, 6]
```

Acceso a elementos y longitud de un array

Cada elemento de un array tiene un índice que comienza desde 0 para el primer elemento. Puedes acceder a cualquier elemento utilizando su índice. Además, la propiedad `length` devuelve el número total de elementos en el array.

```
const numeros = [10, 20, 30, 40];

console.log(numeros[0]); // 10 (primer elemento)
console.log(numeros.length); // 4 (longitud del array)
```

Arrays multidimensionales

Un array también puede contener otros arrays como elementos, creando así estructuras multidimensionales. Esto es útil para representar tablas o matrices. Para acceder a un elemento, necesitas especificar los índices de cada nivel.

```
const matriz = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

console.log(matriz[0][1]); // 2 (elemento de la primera fila, segunda columna)
console.log(matriz[2][0]); // 7 (elemento de la tercera fila, primera columna)
```

Iteración sobre arrays

Iterar sobre un array permite realizar operaciones en cada uno de sus elementos de forma sistemática. Esto es especialmente útil para procesar, transformar, buscar o imprimir el contenido del array.

Uso del bucle `for`

Como ya vimos en el tema anterior, el bucle `for` es una de las formas más comunes de iterar sobre los elementos de un array. Con este enfoque, puedes controlar el índice de cada iteración y realizar operaciones específicas.

```
const numeros = [1, 2, 3, 4];

for (let i = 0; i < numeros.length; i++) {
  console.log(numeros[i]); // Imprime cada elemento
}
```

Uso de `forEach`

El método `.forEach()` permite ejecutar una función para cada elemento del array. Esta función recibe hasta tres argumentos: el elemento actual, el índice del elemento y el array completo.

```
const frutas = ['manzana', 'banana', 'naranja'];

frutas.forEach(function(fruta, indice) {
  console.log(`Fruta en posición ${indice}: ${fruta}`);
});
```

El uso de `forEach` simplifica la iteración y mejora la legibilidad del código, especialmente cuando solo necesitas procesar cada elemento una vez.

Objetos

En JavaScript, un objeto es una estructura que permite almacenar colecciones de datos y funcionalidades relacionadas. Está compuesto por pares clave-valor, donde las claves son cadenas (propiedades) y los valores pueden ser cualquier tipo de dato, incluyendo funciones. Los objetos son fundamentales para representar entidades y modelar datos complejos.

Definición y creación de objetos:

La **definición y creación de objetos** en JavaScript se realiza usando la sintaxis literal `{}`. y asignándolos a una variable, permitiendo modelar entidades y sus características de forma estructurada.

```
const persona = {
  nombre: 'Juan',
  edad: 30,
  profesion: 'Ingeniero'
};

console.log(persona);
```

Acceso a propiedades

Las propiedades de un objeto se pueden acceder utilizando dos notaciones:

Notación de punto:

```
const persona = {
  nombre: 'Ana',
  edad: 25
};

console.log(persona.nombre); // Ana
```

Notación de corchetes:

```
const persona = {
  nombre: 'Ana',
  edad: 25
};

console.log(persona['edad']); // 25
```

La notación de corchetes es útil cuando la clave se almacena en una variable.

```
const propiedad = 'nombre';
console.log(persona[propiedad]); // Ana
```

Métodos

Al igual que se pueden almacenar diferentes tipos de datos en las propiedades de un objeto, también es posible almacenar funciones. Estas funciones, cuando son asociadas a un objeto, se denominan **métodos**. Los métodos permiten que un objeto realice acciones o ejecute comportamientos relacionados con sus datos.

Declaración de métodos

Puede declararse de las siguientes formas:

```
const objeto = {
  propiedad: 'valor',
  metodo: function() {
    console.log('Este es un método del objeto.');
  }
};
```

O de forma más simplificada con la sintaxis de métodos:

```
const objeto = {
  propiedad: 'valor',
  metodo() {
    console.log('Este es un método del objeto.');
  }
};
```

Uso de métodos

Para ejecutar un método, se utiliza la notación de punto (`objeto.metodo()`) o corchetes (`objeto['metodo']()`).

Ejemplo práctico

```
const usuario = {
  nombre: 'Juan',
  saludar() {
    console.log(`Hola, me llamo ${this.nombre}`);
  },
  actualizarNombre(nuevoNombre) {
    this.nombre = nuevoNombre;
  }
};

// Llamar a un método
usuario.saludar(); // Hola, me llamo Juan

// Actualizar el nombre usando un método
usuario.actualizarNombre('María');
usuario.saludar(); // Hola, me llamo María
```

En este ejemplo:

- `saludar` es un método que accede a la propiedad `nombre` usando `this`.
- `actualizarNombre` es un método que modifica el valor de la propiedad `nombre`.

> **Important**
>
> La palabra reservada this en JavaScript es un contexto especial que hace referencia al objeto desde el cual se está ejecutando una función. Su valor depende de cómo y dónde se invoque la función, y puede variar dependiendo del entorno (global, objeto, función, etc.).

Los métodos son fundamentales para encapsular lógica y comportamientos dentro de un objeto, facilitando la organización y reutilización del código.