Arrays

Arrays in JavaScript

Arrays in JavaScript are used to store **collections of elements**. They can contain a mix of data types including numbers, strings, booleans, or even objects. Unlike some languages, JavaScript arrays are **dynamic**—you can add or remove elements at any time. They are also **indexed**, meaning each item in the array has a position, starting from index 0.

For example:

```
let arr = [1, 2, 3];
console.log(arr[0]);  // Output: 1
```

Declaring Arrays

Arrays can be created using either the Array constructor or by using square brackets:

```
let arr1 = new Array();
let arr2 = [];
let arr3 = [1, 2, 3];
```

Array Length

The .length property returns the total number of elements in an array:

```
let arr = [1, 2, 3];
console.log(arr.length);  // Output: 3
```

Accessing Array Elements

You can access elements by their index using square brackets:

```
let fruits = ["apple", "banana", "cherry"];
console.log(fruits[0]);   // Output: apple
```

JavaScript also supports **negative indexing** using the .at() method, which allows access from the end of the array:

```
console.log(fruits.at(-1));  // Output: cherry
console.log(fruits.at(-2));  // Output: banana
```

Adding and Removing Elements

JavaScript provides several methods for modifying arrays:

- push(value) adds an element to the **end**.
- pop() removes the **last** element and returns it.
- unshift(value) adds an element to the **beginning**.
- shift() removes the **first** element and returns it.

Example:

```
let arr = [1, 2, 3];

arr.push(4);     // [1, 2, 3, 4]
arr.pop();       // [1, 2, 3]
arr.unshift(0);  // [0, 1, 2, 3]
arr.shift();     // [1, 2, 3]
```

Looping Over Arrays

Arrays can be looped over using traditional loops like for, while, and do...while. However, there are some array-specific styles that are preferred.

Using a for loop:

```
let arr = ["a", "b", "c"];

for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

Using a for...of loop:

This is a modern and cleaner way to loop through arrays:

```
for (let item of arr) {
  console.log(item);
}
```

Using forEach()

The forEach() method runs a function for each element in the array:

```
let arr = ["apple", "banana", "cherry"];

arr.forEach(item => console.log(item));
```

Output:

apple

banana

cherry

Nested Arrays (Multi-dimensional Arrays)

You can store arrays within arrays — called nested or multi-dimensional arrays.

```
let matrix = [
  [1, 2],
  [3, 4],
  [5, 6]
];

console.log(matrix[1][1]); // Output: 4
```

In the above example, matrix[1][1] accesses the second row, second column value.

Common Array Methods

Here are some powerful array methods that help in working with and transforming data:

1. includes(value)

Checks if the array contains a specific value.

```
let arr = [10, 20, 30];
console.log(arr.includes(20)); // true
```

2. indexOf(value)

Returns the first index of the value if it exists, or -1 if not found.

```
console.log(arr.indexOf(30)); // 2
console.log(arr.indexOf(50)); // -1
```

3. concat()

Merges two or more arrays into a new one.

```
let arr1 = [1, 2];
let arr2 = [3, 4];
let combined = arr1.concat(arr2); // [1, 2, 3, 4]
```

4. slice(start, end)

Returns a shallow copy of a portion of an array.

```
let colors = ["red", "green", "blue", "yellow"];
let part = colors.slice(1, 3); // ["green", "blue"]
```

5. splice(start, deleteCount, ...items)

Adds or removes elements **in place**.

```
let nums = [1, 2, 3, 4];
nums.splice(2, 1, 99); // [1, 2, 99, 4] — removes 1 item at index 2, inserts 99
```