ARRAYS IN JAVASCRIPT

An **array** is a **data structure** that stores a collection of elements in a sequential, contiguous memory location. While the core concept is consistent across programming languages, the implementation and features vary slightly depending on the language.

---

1. General Definition

An **array**:

Stores elements of the same type (in statically-typed languages like C or Java).

Has a fixed size (in languages like C, DSA, and Java, unless you use dynamic structures).

Uses an index to access elements, where indexing typically starts at 0.

Is an efficient way to manage large collections of data because elements are stored in contiguous memory, allowing for constant-time access via index.

---

2. Arrays in Different Languages

C

Arrays in C are **fixed in size** and must store elements of the same type.

Declared like:

```
int arr[5] = {1, 2, 3, 4, 5};
```

Access elements by index:

```
printf("%d", arr[0]); // Output: 1
```

No built-in methods for resizing or manipulating arrays. Developers handle resizing manually using pointers and dynamic memory allocation.

Data Structures & Algorithms (DSA)

Arrays are fundamental in **DSA**, serving as the base for many advanced structures (e.g., stacks, queues, heaps).

Operations:

Traversal: Access elements sequentially.

Search: Linear or binary search.

Insertion/Deletion: Costly operations due to shifting of elements.

Examples:

Dynamic arrays like **Vectors** (in C++) or **ArrayLists** (in Java).

Java

Arrays are **fixed in size** once created.

Declared like:

int[] arr = {10, 20, 30};

Access:

System.out.println(arr[1]); // **Output: 20**

Provides utility methods through java.util.Arrays (e.g., sorting, searching).

Java also has dynamic array-like structures like **ArrayList**.

JavaScript

Arrays in JavaScript are **dynamic**, meaning their size can grow or shrink automatically.

Can store **different types of elements** in a single array.

Declared like:

let arr = [10, 'hello', true];

Access:

console.log(arr[1]); // **Output: 'hello'**

Comes with **powerful built-in methods**:

Mutating: push, pop, splice, shift, unshift

Iterative: forEach, map, filter, reduce

Utility: concat, slice, indexOf, sort

Key Differences Across Languages

Feature	**C**	**DSA**	**Java**	**JavaScript**
Size	Fixed	Fixed/Dynamic	Fixed	Dynamic
Type Restriction	Same Type	Same Type	Same Type	Any Type
Indexing	0-based	0-based	0-based	0-based
Flexibility	Limited	Limited	Limited (use ArrayList)	Very Flexible
Built-in Methods	None	Basic	Utility methods	Extensive Built-in Methods

What is an Array in JavaScript?

- An **array** in JavaScript is a **special object** used to store multiple values in a **single variable**.
- Arrays can hold elements of **any data type**: numbers, strings, booleans, objects, or even other arrays.
- They are **dynamic**, meaning you can change their size (add or remove elements) without declaring a fixed size.

➢ **How to Create an Array**

1. **Using Square Brackets (Recommended)**:

   let departments = ['CSE', 'ECE', 'CSM'];

2. **Using the Array Constructor**:

   let numbers = new Array(10, 20, 30);

   Accessing Array Elements

- Each element in an array is accessed using its **index** (starting from 0):

  let colors = ['Red', 'Green', 'Blue'];

  console.log(colors[0]); // **Output: Red**

  console.log(colors[2]); // **Output: Blue**

Features of JavaScript Arrays

1. **Dynamic Size**:
   - You don't need to declare the size of the array. It grows or shrinks as needed.
2. **Mixed Data Types**:
   - An array can contain different types of values.

   let mixedArray = [42, 'hello', true, [1, 2, 3]];

   console.log(mixedArray[3]); // **Output: [1, 2, 3]**
3. **Length Property**:
   - The length property gives the total number of elements in an array.

   let arr = [1, 2, 3];

   console.log(arr.length); // **Output: 3**

JavaScript offers different types of arrays, each serving various purposes. Here's a quick rundown:

1. **Single-Dimensional Arrays**: These are the most common arrays in JavaScript, used for storing a list of values.

   ```
   let fruits = ["Apple", "Banana", "Mango"];

   console.log(fruits);
   ```

2. **Multi-Dimensional Arrays**: Arrays that contain other arrays. Often used to represent matrices or grids.

   ```
   let matrix = [

           [1, 2, 3],

           [4, 5, 6],

           [7, 8, 9]

           ];

   console.log(matrix[1][2]);
   ```

3. **Sparse Arrays**: These arrays have elements with gaps, meaning some indexes do not have any value(undefined indices).

   ```
   let sparseArray = [1, , , 4]; // Missing elements at indexes 1 and 2

   console.log(sparseArray[1]); //undefined
   ```

4. **Associative Arrays**: In JavaScript, these are actually just objects. They allow you to use named keys instead of numeric indexes.

   ```
   let obj = { "firstName": "John", "lastName": "Doe" };
   ```

Array Properties:

1. length:
   - This property returns the number of elements in the array.

```
let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits.length); // Outputs: 3
```

2. constructor:
   - This property returns the function that created the Array object's prototype. For arrays, this is the Array constructor.

```
let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits.constructor === Array); // Outputs: true
```

3. prototype:
   - The prototype property allows you to add new properties and methods to array objects.

```
Array.prototype.first = function() {

return this[0];

};

let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits.first()); // Outputs: Apple
```

4. toString():
   - This method converts an array to a string of (comma-separated) array values.

```
let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits.toString()); // Outputs: Apple,Banana,Mango
```

5. valueOf():
   - This method returns the array itself. It's often used internally by JavaScript.

```
let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits.valueOf() === fruits); // Outputs: true
```

6. join():
    - This method joins all elements of an array into a string. You can specify a separator.

```
let fruits = ["Apple", "Banana", "Mango"];

console.log(fruits.join(" - ")); // Outputs: Apple - Banana - Mango
```

7. concat():
    - This method is used to merge two or more arrays. It returns a new array.

```
let fruits = ["Apple", "Banana"];

let moreFruits = ["Mango", "Pineapple"];

let allFruits = fruits.concat(moreFruits);

console.log(allFruits); // Outputs: ["Apple", "Banana", "Mango", "Pineapple"]
```

8. slice():
    - This method returns a shallow copy of a portion of an array into a new array object.

```
let fruits = ["Apple", "Banana", "Mango"];

let citrus = fruits.slice(1, 2);

console.log(citrus); // Outputs: ["Banana"]
```

Here’s a **tabulated list of JavaScript array methods** with their names and meanings:

Method	Description
push()	Adds one or more elements to the **end** of the array.
pop()	Removes the **last** element from the array and returns it.
unshift()	Adds one or more elements to the **beginning** of the array.
shift()	Removes the **first** element from the array and returns it.
concat()	Combines two or more arrays into a **new array**.
slice()	Returns a **shallow copy** of a portion of the array (selected elements).
splice()	Adds, removes, or replaces elements in an array at a specified index.
indexOf()	Returns the **first index** of a specified element. Returns -1 if the element is not found.
lastIndexOf()	Returns the **last index** of a specified element. Returns -1 if the element is not found.
includes()	Checks if the array contains a specified element and returns true or false.
forEach()	Executes a function for each element in the array (does not return a new array).
map()	Creates a **new array** by applying a function to each element.
filter()	Creates a **new array** with elements that match a specified condition.
reduce()	Reduces the array to a **single value** by applying a function (e.g., sum of all elements).
reduceRight()	Similar to reduce(), but processes the array from **right to left**.
find()	Returns the **first element** that satisfies a condition.
findIndex()	Returns the **index of the first element** that satisfies a condition.
some()	Checks if **any element** satisfies a condition; returns true or false.
every()	Checks if **all elements** satisfy a condition; returns true or false.
sort()	Sorts the array **in place** (can handle strings and numbers, but needs a comparator for numbers).
reverse()	Reverses the order of the elements in the array **in place**.
join()	Joins all elements of the array into a string, separated by a specified delimiter.

split()	(String method, but often used with arrays) Splits a string into an array of substrings.
toString()	Converts an array into a string of comma-separated values.
flat()	Flattens a **nested array** into a single-level array.
flatMap()	Maps each element using a function and flattens the result into a new array.
fill()	Fills all or part of the array with a static value.
copyWithin()	Copies part of the array to another location **within the same array**.
entries()	Returns an **iterator** of key/value pairs for each element.
keys()	Returns an **iterator** of keys (indices) in the array.
values()	Returns an **iterator** of values in the array.
Array.isArray()	Checks if a variable is an array; returns true or false.

Examples:

1. push()

Adds elements to the **end** of an array.

```
let arr = [1, 2, 3];
arr.push(4);
console.log(arr); // Output: [1, 2, 3, 4]
```

---

2. pop()

Removes the **last** element and returns it.

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

---

3. unshift()

Adds elements to the **beginning** of an array.

```
let arr = [1, 2, 3];
arr.unshift(0);
```

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

---

4. shift()

Removes the **first** element and returns it.

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

---

5. concat()

Combines two or more arrays into a **new array**.

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

---

6. slice()

Returns a **portion** of an array as a new array.

```
let arr = [1, 2, 3, 4, 5];
let sliced = arr.slice(1, 4);
console.log(sliced); // Output: [2, 3, 4]
```

---

7. splice()

Adds, removes, or replaces elements at a specified index.

```
let arr = [1, 2, 3, 4];
arr.splice(1, 2, 'a', 'b'); // Removes 2 elements starting from index 1 and adds 'a', 'b'
console.log(arr); // Output: [1, 'a', 'b', 4]
```

---

8. indexOf()

Finds the **first index** of a specified element.

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

---

9. lastIndexOf()

Finds the **last index** of a specified element.

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

---

10. includes()

Checks if the array contains a specified element.

```
let arr = [1, 2, 3];
console.log(arr.includes(2)); // Output: true
console.log(arr.includes(4)); // Output: false
```

---

11. forEach()

Executes a function for each element in the array.

```
let arr = [1, 2, 3];
arr.forEach(num => console.log(num * 2));
            (or)
arr.forEach(function(num) {
 console.log(num * 2);
});
// Output: 2, 4, 6
```

---

12. map()

Creates a **new array** by applying a function to each element.

```
let arr = [1, 2, 3];
let squared = arr.map(num => num ** 2);
console.log(squared); // Output: [1, 4, 9]
```

---

13. filter()

Creates a **new array** with elements that satisfy a condition.

```
let arr = [1, 2, 3, 4];
let even = arr.filter(num => num % 2 === 0);
console.log(even); // Output: [2, 4]
```

---

14. reduce()

Reduces the array to a **single value**.

```
let arr = [1, 2, 3, 4];
let sum = arr.reduce((acc, num) => acc + num, 0);
console.log(sum); // Output: 10
```

---

15. reduceRight()

Similar to reduce(), but processes the array **right to left**.

```
let arr = ['a', 'b', 'c'];
let result = arr.reduceRight((acc, char) => acc + char, '');
console.log(result); // Output: "cba"
```

---

16. find()

Returns the **first element** that satisfies a condition.

```
let arr = [1, 2, 3, 4];
let found = arr.find(num => num > 2);
console.log(found); // Output: 3
```

---

17. findIndex()

Returns the **index of the first element** that satisfies a condition.

```
let arr = [1, 2, 3, 4];
let index = arr.findIndex(num => num > 2);
console.log(index); // Output: 2
```

---

18. some()

Checks if **any element** satisfies a condition.

```
let arr = [1, 2, 3];
console.log(arr.some(num => num > 2)); // Output: true
```

---

19. every()

Checks if **all elements** satisfy a condition.

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

---

20. sort()

Sorts the array **in place**.

```
let arr = [3, 1, 4, 2];
arr.sort();
console.log(arr); // Output: [1, 2, 3, 4]
```

---

21. reverse()

Reverses the array **in place**.

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

---

22. join()

Joins all elements into a string.

```
let arr = ['a', 'b', 'c'];
let joined = arr.join('-');
console.log(joined); // Output: "a-b-c"
```

---

23. flat()

Flattens a **nested array**.

```
let arr = [1, [2, [3, 4]]];
let flatArr = arr.flat(2);
console.log(flatArr); // Output: [1, 2, 3, 4]
```

---

24. flatMap()

Maps and flattens the result into a new array.

```
let arr = [1, 2, 3];
let mapped = arr.flatMap(num => [num, num * 2]);
console.log(mapped); // Output: [1, 2, 2, 4, 3, 6]
```

---

25. fill()

Fills an array with a static value.

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

---

26. copyWithin()

Copies part of the array to another location **within the same array**.

```
let arr = [1, 2, 3, 4];
arr.copyWithin(1, 2);
console.log(arr); // Output: [1, 3, 4, 4]
```

---

27. entries()

Returns an **iterator** of key/value pairs.

```
let arr = ['a', 'b', 'c'];
let iterator = arr.entries();
for (let [index, value] of iterator) {
  console.log(index, value);
}
// Output: 0 'a', 1 'b', 2 'c'
```

---

28. keys()

Returns an **iterator** of indices.

```
let arr = ['a', 'b', 'c'];
let iterator = arr.keys();
for (let key of iterator) {
  console.log(key);
}
// Output: 0, 1, 2
```

---

29. values()

Returns an **iterator** of values.

```
let arr = ['a', 'b', 'c'];
let iterator = arr.values();
for (let value of iterator) {
  console.log(value);
}
// Output: 'a', 'b', 'c'
```

---

30. Array.isArray()

Checks if a variable is an array.

```
console.log(Array.isArray([1, 2, 3])); // Output: true
console.log(Array.isArray('hello'));  // Output: false
```

Property and Method in JavaScript

Property

- A **property** is a **value** associated with an object.
- It is like a variable that is tied to the object.
- Properties **store data** about the object.

Example

```
let person = {
  name: 'Alice',      // 'name' is a property
  age: 25             // 'age' is a property
};

console.log(person.name); // Output: Alice
console.log(person.age);  // Output: 25
```

Here:

- name and age are **properties** of the person object.
- They store values 'Alice' and 25.

Method

- A **method** is a **function** that is associated with an object.
- It defines **behavior** or actions the object can perform.
- Methods **operate** on the object's properties or perform specific tasks.

Example

```
let person = {
  name: 'Alice',
  age: 25,
  greet: function() {   // 'greet' is a method
    console.log(`Hello, my name is ${this.name}`);
  }
};
```

```
person.greet(); // Output: Hello, my name is Alice
```

Here:

- greet is a **method** of the person object.
- It performs the action of printing a greeting, and it uses the name property.

Key Differences

Aspect	Property	Method
Definition	A **value** associated with an object.	A **function** associated with an object.
Purpose	Stores data or state.	Defines behavior or actions.
Data Type	Typically a value (e.g., string, number, boolean).	A function.
Example	person.name or person.age	person.greet()
How to Access	Accessed like a variable.	Accessed and executed like a function.

Combined Example

```
let car = {
  brand: 'Toyota',           // Property
  speed: 120,                // Property
  drive: function() {        // Method
    console.log(`Driving at ${this.speed} km/h`);
  }
};


console.log(car.brand);  // Output: Toyota (Property)
car.drive();             // Output: Driving at 120 km/h (Method)
```

Summary

- **Properties** hold **data** (like variables).
- **Methods** define **behavior** (like functions tied to an object).