Objects and Arrays in JavaScript

In JavaScript, **Objects** and **Arrays** are fundamental data structures used to store and organize data efficiently.

□ 1. Objects

An **object** is a collection of **key–value pairs** (properties).
Each key (also called a property name) is a string, and its value can be anything — a number, string, array, another object, or even a function.

Syntax:

```
let objectName = {
  key1: value1,
  key2: value2,
  key3: value3
};
```

Example:

```
let person = {
  name: "Raj",
  age: 24,
  isStudent: true
};
```

Accessing Object Properties

```
console.log(person.name);       // Dot notation → Raj
console.log(person["age"]);     // Bracket notation → 24
```

Adding / Modifying Properties

```
person.city = "Kathmandu";      // Add new property
person.age = 25;                // Modify existing property
```

Deleting Properties

```
delete person.isStudent;
```

⚙ Methods in Objects

Objects can also contain **functions** called **methods**.

```
let car = {
  brand: "Toyota",
  model: "Corolla",
  start: function() {
    console.log("Car started");
  }
};

car.start();   // Output: Car started
console.log(car.brand); // access property Output:Toyota
```

Nested Objects

```
const person = {
  name: "Raj",
  age: 24,
  address: {
    city: "Kathmandu",
    country: "Nepal"
  }
};

console.log(person.name);           // Raj
console.log(person.address.country); // Nepal

// Looping through object properties with nested object
for (let key in person) {
  if (typeof person[key] === 'object') {
    for (let nestedKey in person[key]) {
      console.log(`${nestedKey}: ${person[key][nestedKey]}`);
    }
  } else {
    console.log(`${key}: ${person[key]}`);
  }
}
```

---

2. Array

An **array** is a special type of object used to store **ordered collections of data** (elements).

Syntax:

```
let arrayName = [value1, value2, value3];
```

Example:

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

Accessing Array Elements:

```
console.log(fruits[0]);  // Output: Apple
console.log(fruits[2]);  // Output: Mango
```

Modifying Elements:

```
fruits[1] = "Orange";
console.log(fruits);     // ["Apple", "Orange", "Mango"]
```

Adding and Removing Items

```
fruits.push("grapes"); // Add to end
fruits.pop();          // Remove last
fruits.unshift("pear"); // Add to beginning
fruits.shift();         // Remove first
```

🔁 Looping Through Arrays

```
let colors = ["Red", "Green", "Blue"];

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

Or using **forEach()**:

```
colors.forEach(function(color) {
  console.log(color);
});
```

Or using arrow function

```
colors.forEach((color) => console.log(color));
```

Array of Objects Example

Objects and arrays are often combined for real-world data structures:

```
let students = [
  { name: "Raj", age: 24 },
  { name: "Sita", age: 22 },
  { name: "Aman", age: 23 }
];

console.log(students[1].name);  // Output: Sita
```

Array Methods in JavaScript

1. Adding / Removing Elements

Method	Description	Example
`push()`	Add item **at end**	`arr.push(5)`
`pop()`	Remove last item	`arr.pop()`
`unshift()`	Add item **at start**	`arr.unshift(0)`
`shift()`	Remove first item	`arr.shift()`
`splice(start, deleteCount, ...items)`	Add/remove items at any position	`arr.splice(2, 1, "newItem")`
`slice(start, end)`	Returns a **copy** of part of array (doesn't modify original)	`arr.slice(1, 3)`

Example:

```
let fruits = ["apple", "banana", "mango"];

fruits.push("orange");  // ["apple","banana","mango","orange"]
fruits.pop();           // ["apple","banana","mango"]
fruits.unshift("grape"); // ["grape","apple","banana","mango"]
fruits.shift();         // ["apple","banana","mango"]

fruits.splice(1, 1, "kiwi"); // replaces "banana" →
["apple","kiwi","mango"]

let sliced = fruits.slice(0, 2); // ["apple","kiwi"]
```

---

2. Searching / Checking Elements

Method	Description	Example
`includes(value)`	Checks if value exists → `true/false`	`arr.includes(3)`
`indexOf(value)`	Returns first index or -1	`arr.indexOf(2)`
`lastIndexOf(value)`	Returns last index of value	`arr.lastIndexOf(2)`
`find(callback)`	Returns **first matching element**	`arr.find(x => x > 10)`
`findIndex(callback)`	Returns **index** of first match	`arr.findIndex(x => x > 10)`

Example:

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

console.log(numbers.includes(20));   // true
console.log(numbers.indexOf(30));    // 2
console.log(numbers.find(x => x > 25));  // 30
console.log(numbers.findIndex(x => x > 25)); // 2
```

---

3. Iterating / Looping Methods

Method	Description	Example
`forEach(callback)`	Runs a function for each element	`arr.forEach(x => console.log(x))`
`map(callback)`	Returns **new array** after transforming each item	`arr.map(x => x * 2)`
`filter(callback)`	Returns **new array** with elements that pass test	`arr.filter(x => x > 5)`
`reduce(callback, initial)`	Combines values into one result	`arr.reduce((a,b)=>a+b)`
`some(callback)`	Checks if **any** element matches → `true/false`	`arr.some(x => x > 10)`
`every(callback)`	Checks if **all** elements match → `true/false`	`arr.every(x => x > 0)`

Example:

```
const nums = [2, 4, 6, 8];

nums.forEach(n => console.log(n));        // Just prints
const double = nums.map(n => n * 2);          // [4,8,12,16]
const even = nums.filter(n => n % 2 === 0);   // [2,4,6,8]
const sum = nums.reduce((a,b) => a + b, 0);   // 20
console.log(sum);
```

---

4. Sorting & Reversing

Method	Description	Example
`sort()`	Sorts array **alphabetically** (by string by default)	`arr.sort()`
`reverse()`	Reverses the array order	`arr.reverse()`

⚠ Note: For **numbers**, use a compare function:

```
const nums = [10, 2, 30];
nums.sort((a, b) => a - b); // Ascending
nums.sort((a, b) => b - a); // Descending
```

---

5. Combining / Converting

Method	Description	Example
`concat()`	Combines arrays	`arr1.concat(arr2)`
`join(separator)`	Converts array → string	`arr.join(", ")`
`flat(depth)`	Flattens nested arrays	`[1,[2,[3]]].flat(2)`

Example:

```
const arr1 = [1, 2];
const arr2 = [3, 4];
console.log(arr1.concat(arr2));   // [1,2,3,4]

const fruits = ["apple", "banana"];
console.log(fruits.join(" & "));  // "apple & banana"

const nested = [1, [2, [3, 4]]];
console.log(nested.flat(2));      // [1,2,3,4]
```

---

6. Other Useful Methods

Method	Description	Example
`Array.isArray()`	Checks if value is array	`Array.isArray(arr)`
`fill(value, start, end)`	Fills part of array with same value	`arr.fill(0, 1, 3)`
`toString()`	Converts array → string	`[1,2,3].toString()`
`at(index)`	Access element using positive/negative index	`arr.at(-1)` → last element
`from()`	Creates array from iterable (like string)	`Array.from("Raj")` → `['R','a','j']`
`keys(), values(), entries()`	Iterators for keys/values/pairs	`for (let [i,v] of arr.entries()) {...}`

🧠 Chain Methods

You can **chain multiple methods** for clean code:

```
const numbers = [1, 2, 3, 4, 5, 6];

const result = numbers
  .filter(n => n % 2 === 0)   // even numbers → [2,4,6]
  .map(n => n * 10)           // multiply each by 10 → [20,40,60]
  .reduce((a,b) => a + b, 0); // sum → 120

console.log(result); // 120
```

👉 Mini Assignment (Practice)

1. Create an object `book` with properties: `title`, `author`, and `year`.
   - Print: `"The book <title> was written by <author> in <year>."`
2. Create an array `countries` with 4 country names.
   - Print the **first** and **last** country.
   - Add a new country at the end and print the updated array.
3. Create an array of objects `students` with 3 students (each having `name` and `age`).
   - Print the name of the **second student**.

4. Create an array `nums = [3, 7, 10, 15, 20]`

   - Add `25` at the end
   - Remove first item
   - Print remaining array
   - Use `filter()` to print numbers greater than 10.
   - Use `map()` to multiply each number by 2.
   - Use `reduce()` to find the total sum.
   - Use `slice()` to copy only the middle 3 numbers.
   - Use `splice()` to remove the 2nd and 3rd elements.

5. Sort `[5, 12, 3, 8, 1]` in **ascending** and **descending** order.
6. Combine two arrays `["a","b"]` and `["c","d"]` → one array.