Training Day 23 Report

23 July 2025

Arrays and Objects in JavaScript

1. Introduction

In JavaScript, **arrays** and **objects** are used to store **collections of data**.
They are fundamental structures that make it easy to organize, access, and manipulate information.

- **Array:** A collection of **ordered** values.
- **Object:** A collection of **key-value pairs** representing properties of an entity.

Both are **dynamic** and can store different types of data including numbers, strings, and even other arrays or objects.

Arrays in JavaScript

2. What is an Array?

An array is a special type of variable that can hold **multiple values** in a single variable. Each value is called an **element**, and its position in the array is called an **index** (starting from 0).

Syntax:

```
let arrayName = [element1, element2, element3];
```

Example:

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

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

3. Array Methods

JavaScript provides many built-in methods to manipulate arrays:

1. **push()** – Adds an element at the end.

```
fruits.push("Orange");

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

2. **pop()** – Removes the last element.

```
fruits.pop();

console.log(fruits);  // ["Apple", "Banana", "Mango"]
```

3. **shift()** – Removes the first element.

```
fruits.shift();

console.log(fruits);  // ["Banana", "Mango"]
```

4. **unshift()** – Adds an element at the beginning.

```
fruits.unshift("Strawberry");

console.log(fruits);  // ["Strawberry", "Banana", "Mango"]
```

5. **length** – Returns the number of elements.

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

6. **indexOf()** – Finds the index of an element.

```
console.log(fruits.indexOf("Mango"));  // 2
```

4. Looping Through Arrays

You can use loops to access array elements:

Using for loop:

```
for (let i = 0; i < fruits.length; i++) {

  console.log(fruits[i]);

}
```

Using for...of loop:

```
for (let fruit of fruits) {

  console.log(fruit);

}
```

5. Multidimensional Arrays

Arrays can contain other arrays, creating **2D or multidimensional arrays**.

Example:

```
let matrix = [

  [1, 2, 3],

  [4, 5, 6],

  [7, 8, 9]

];
```

```
console.log(matrix[1][2]); // Output: 6
```

Objects in JavaScript

6. What is an Object?

An object is a collection of **properties**, where each property has a **key** (name) and a **value**.
Objects are used to represent real-world entities or structured data.

Syntax:

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

Example:

```
let person = {
  name: "Shubhdeep",
  age: 20,
  city: "Ludhiana"
};
console.log(person.name); // Output: Shubhdeep
```

7. Accessing Object Properties

Properties can be accessed in two ways:

1. **Dot notation**

```
console.log(person.age);  // 20
```

2. **Bracket notation**

```
console.log(person["city"]);  // Ludhiana
```

8. Modifying Objects

You can **add, update, or delete** properties of an object.

Example:

```
person.country = "India";  // Add new property
person.age = 21;         // Update property
```

```
delete person.city;      // Delete property
console.log(person);
```

Output:

```
{ name: 'Shubhdeep', age: 21, country: 'India' }
```

9. Looping Through Objects

Using for...in loop:

```
for (let key in person) {
  console.log(key + ": " + person[key]);
}
```

Output:

```
name: Shubhdeep
age: 21
country: India
```

10. Nested Objects

Objects can contain **other objects or arrays**, allowing complex data structures.

Example:

```
let student = {
  name: "Riya",
  marks: { math: 90, english: 85 },
  hobbies: ["Reading", "Dancing"]
};
console.log(student.marks.math);  // 90
console.log(student.hobbies[1]);  // Dancing
```

11. Difference Between Arrays and Objects

Feature	**Array**	**Object**
Structure	Ordered collection	Key-value pairs
Indexing	Numeric index	Named keys
Syntax	[value1, value2]	{ key1: value1, key2: value2 }

Feature	**Array**	**Object**
Best for	List of items	Data with properties
Example	[1, 2, 3]	{name: "John", age: 25}

12. Advantages of Using Arrays and Objects

1. **Efficient Data Storage:** Store multiple values in one variable.
2. **Organization:** Keeps data structured and manageable.
3. **Dynamic Access:** Easily access or modify data.
4. **Flexibility:** Can store mixed data types.
5. **Nested Structures:** Can create complex data models.