10. Functions in JavaScript

A **function** is a **block of reusable code** that performs a specific task.
It helps make code **modular, readable, and reusable.**

A. Function Declaration

Syntax:

```
function functionName(parameters) {
  // code to execute
}
```

Example:

```
function greet() {
  console.log("Hello, JavaScript!");
}

greet(); // Function call
```

Output:

```
Hello, JavaScript!
```

B. Function with Parameters and Arguments

Parameters act like variables that accept values when a function is called.

Example:

```
function add(a, b) {
  console.log("Sum:", a + b);
}

add(5, 3);   // Arguments
```

Output:

```
Sum: 8
```

C. Function with Return Value

Functions can **return a result** using the `return` keyword.

Example:

```
function multiply(x, y) {
  return x * y;
}

let result = multiply(4, 5);
console.log("Result:", result);
```

Output:

```
Result: 20
```

D. Function Expression

A function can be stored in a **variable** — called a **function expression**.

Example:

```
const subtract = function(a, b) {
  return a - b;
};

console.log(subtract(10, 4));
```

Output:

```
6
```

E. Arrow Functions (ES6)

A shorter way to write functions using the => arrow symbol.

Syntax:

```
const functionName = (parameters) => {
  // code
};
```

Example 1:

```
const greet = () => {
  console.log("Welcome!");
};

greet();
```

Output:

```
Welcome!
```

Example 2: (One-line return)

```
const square = num => num * num;

console.log(square(6));
```

Output:

```
36
```

F. Default Parameters

You can assign **default values** to parameters.

Example:

```
function greet(name = "Guest") {
  console.log("Hello,", name);
}

greet("Neeraj");
greet(); // uses default
```

Output:

```
Hello, Neeraj
Hello, Guest
```

Arrays in JavaScript

An **array** is a **collection of values** stored in a single variable.
Each value is stored at a numeric **index**, starting from **0**.

A. Array Creation

1. Using square brackets `[ ]`:

```
let fruits = ["Apple", "Banana", "Cherry"];
console.log(fruits);
```

Output:

```
["Apple", "Banana", "Cherry"]
```

2. Using the Array constructor:

```
let numbers = new Array(10, 20, 30);
console.log(numbers);
```

Output:

```
[10, 20, 30]
```

B. Array Indexing

- Index starts from `0`
- You can access or change elements using **indexes**

Example:

```
let colors = ["Red", "Green", "Blue"];
console.log(colors[0]); // Red
console.log(colors[2]); // Blue
```

```
colors[1] = "Yellow";  // Update element
console.log(colors);
```

Output:

```
Red
Blue
["Red", "Yellow", "Blue"]
```

Array Methods (Basic)

A. push()

Adds an element **at the end** of the array.

```
let fruits = ["Apple", "Banana"];
fruits.push("Cherry");
console.log(fruits);
```

Output:

```
["Apple", "Banana", "Cherry"]
```

B. pop()

Removes the **last element** from the array.

```
let fruits = ["Apple", "Banana", "Cherry"];
fruits.pop();
console.log(fruits);
```

Output:

```
["Apple", "Banana"]
```

C. shift()

Removes the **first element** from the array.

```
let fruits = ["Apple", "Banana", "Cherry"];
fruits.shift();
console.log(fruits);
```

Output:

```
["Banana", "Cherry"]
```

D. unshift()

Adds an element **at the beginning** of the array.

```
let fruits = ["Banana", "Cherry"];
fruits.unshift("Apple");
```

```
console.log(fruits);
```

Output:

```
["Apple", "Banana", "Cherry"]
```

Array Methods (map, filter, reduce, find, forEach)

A. map()

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

```
let numbers = [1, 2, 3, 4];
let squares = numbers.map(num => num * num);
console.log(squares);
```

Output:

```
[1, 4, 9, 16]
```

B. filter()

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

```
let numbers = [10, 20, 30, 40];
let result = numbers.filter(num => num > 20);
console.log(result);
```

Output:

```
[30, 40]
```

C. reduce()

Reduces all elements to a **single value** (like sum, average, etc.)

```
let numbers = [10, 20, 30];
let sum = numbers.reduce((acc, num) => acc + num, 0);
console.log(sum);
```

Output:

```
60
```

D. find()

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

```
let numbers = [5, 12, 8, 130, 44];
let found = numbers.find(num => num > 10);
console.log(found);
```

Output:

```
12
```

E. forEach()

Runs a function **for each element** (does not return a new array).

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

fruits.forEach(fruit => {
  console.log("Fruit:", fruit);
});
```

Output:

```
Fruit: Apple
Fruit: Banana
Fruit: Cherry
```

Array Methods (some, every, sort, reverse, concat, slice, splice)

A. some()

Checks if **any** element satisfies a condition → returns `true` or `false`.

```
let ages = [10, 15, 20, 25];
let hasAdult = ages.some(age => age >= 18);
console.log(hasAdult);
```

Output:

```
true
```

B. every()

Checks if **all** elements satisfy a condition → returns `true` or `false`.

```
let ages = [18, 21, 25];
let allAdults = ages.every(age => age >= 18);
console.log(allAdults);
```

Output:

```
true
```

C. sort()

Sorts the array (by default in **alphabetical order**).

```
let fruits = ["Banana", "Apple", "Cherry"];
fruits.sort();
console.log(fruits);
```

Output:

```
["Apple", "Banana", "Cherry"]
```

For numbers:

```
let numbers = [40, 100, 1, 5, 25];
numbers.sort((a, b) => a - b);  // Ascending
console.log(numbers);
```

Output:

```
[1, 5, 25, 40, 100]
```

D. reverse()

Reverses the array order.

```
let numbers = [1, 2, 3, 4];
numbers.reverse();
console.log(numbers);
```

Output:

```
[4, 3, 2, 1]
```

E. concat()

Merges two or more arrays into one.

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

Output:

```
[1, 2, 3, 4]
```

F. slice()

Returns a **portion of the array** (without changing original).

```
let fruits = ["Apple", "Banana", "Cherry", "Mango"];
let sliced = fruits.slice(1, 3);
console.log(sliced);
```

Output:

```
["Banana", "Cherry"]
```

G. splice()

Adds or removes elements from the array (changes original).

Syntax:

```
array.splice(startIndex, deleteCount, newItem1, newItem2, ...)
```

```
let fruits = ["Apple", "Banana", "Cherry"];
fruits.splice(1, 1, "Mango", "Orange");
console.log(fruits);
```

Output:

```
["Apple", "Mango", "Orange", "Cherry"]
```

Summary Table

Method	Description	Changes Original?
push()	Add at end	Yes
pop()	Remove last	Yes
shift()	Remove first	Yes
unshift()	Add at start	Yes
map()	Creates new transformed array	No
filter()	Creates new filtered array	No
reduce()	Reduces array to one value	No
find()	Finds first matching element	No
forEach()	Runs function for each element	No
some()	Checks if any element matches	No
every()	Checks if all match	No
sort()	Sorts array	Yes
reverse()	Reverses array	Yes
concat()	Merges arrays	No
slice()	Extracts part of array	No
splice()	Add/Remove elements	Yes

Multidimensional Arrays

A **multidimensional array** means an **array inside another array**.
It is often used to store data in **rows and columns** (like a table or matrix).

Example 1: Creating a 2D Array

```
let matrix = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

Example 2: Accessing Elements

```
console.log(matrix[0][0]); // 1 (first row, first column)
console.log(matrix[1][2]); // 6 (second row, third column)
```

Example 3: Looping through a 2D Array

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

Output:

```
1 2 3 4 5 6 7 8 9
```

Strings in JavaScript

A **string** is a sequence of characters enclosed in **single quotes (‘ ’)**, **double quotes (“ ”)**, or **backticks ( )**.

Example:

```
let name = "Neeraj";
let greeting = 'Hello';
let message = `Welcome, ${name}!`;
```

String Length

The `.length` property returns the number of characters in a string.

```
let text = "JavaScript";
console.log(text.length); // 10
```

Template Literals

- Introduced in ES6.
- Enclosed in **backticks ( )**.
- Allow **multi-line strings** and **variable interpolation** using `${}`.

Example:

```
let firstName = "Neeraj";
let age = 23;

let info = `My name is ${firstName}.
I am ${age} years old.`;

console.log(info);
```

Output:

```
My name is Neeraj.
I am 23 years old.
```

String Methods (toUpperCase, toLowerCase, trim)

JavaScript provides several **built-in methods** to work with strings.

1. toUpperCase()

Converts all letters of a string to **uppercase**.

```
let name = "neeraj";
console.log(name.toUpperCase());
// Output: "NEERAJ"
```

2. toLowerCase()

Converts all letters of a string to **lowercase**.

```
let country = "INDIA";
console.log(country.toLowerCase());
// Output: "india"
```

3. trim()

Removes **extra spaces** from the **start and end** of a string.

```
let text = "   Hello JavaScript!   ";
console.log(text.trim());
// Output: "Hello JavaScript!"
```

Other Important String Methods

1. split()

Splits a string into an **array** based on a separator.

```
let fruits = "apple,banana,mango";
let arr = fruits.split(",");
console.log(arr);
// Output: ["apple", "banana", "mango"]
```

2. replace()

Replaces a part of a string with another value.

```
let text = "I like Java.";
let newText = text.replace("Java", "JavaScript");
console.log(newText);
// Output: "I like JavaScript."
```

3. substring(start, end)

Extracts a portion of the string between **start** and **end** indexes.

```
let word = "JavaScript";
console.log(word.substring(0, 4));
// Output: "Java"
```

4. includes()

Checks if a string **contains** a given substring.
Returns `true` or `false`.

```
let sentence = "Learning JavaScript is fun!";
console.log(sentence.includes("JavaScript"));
// Output: true
```

5. startsWith()

Checks if a string **starts with** a specific substring.

```
let text = "Hello World!";
console.log(text.startsWith("Hello"));
// Output: true
```

6. endsWith()

Checks if a string **ends with** a specific substring.

```
let text = "Hello World!";
console.log(text.endsWith("World!"));
// Output: true
```

Objects in JavaScript

What is an Object?

An **object** is a collection of **key-value pairs**.
Each key is called a **property**, and the value can be **any data type** (number, string, array, function, etc.).

1. Object Literal Syntax

```
let person = {
  name: "Neeraj",
  age: 23,
  city: "Lucknow"
};
```

Accessing Object Properties

```
console.log(person.name);     // Dot notation → "Neeraj"
console.log(person["city"]);  // Bracket notation → "Lucknow"
```

2. Adding & Deleting Properties

```
person.country = "India";   // Add new property
delete person.age;          // Delete a property

console.log(person);
```

3. Object Methods

A **method** is a function defined inside an object.

```
let student = {
  name: "Amit",
  marks: 90,
  greet: function() {
    console.log("Hello, my name is " + this.name);
  }
};

student.greet();  // Output: Hello, my name is Amit
```

4. The `this` Keyword

- `this` refers to the **current object**.
- It is used inside methods to access object properties.

```
let car = {
  brand: "Toyota",
  model: "Fortuner",
  display: function() {
    console.log(`Car: ${this.brand} ${this.model}`);
  }
};

car.display();  // Output: Car: Toyota Fortuner
```

Object Utilities

JavaScript provides built-in utility methods for working with objects.

1. Object.keys(obj)

Returns an **array of property names (keys)**.

```
let person = { name: "Neeraj", age: 23, city: "Lucknow" };
console.log(Object.keys(person));
// Output: ["name", "age", "city"]
```

2. Object.values(obj)

Returns an **array of property values**.

```
console.log(Object.values(person));
// Output: ["Neeraj", 23, "Lucknow"]
```

3. Object.entries(obj)

Returns an **array of [key, value] pairs**.

```
console.log(Object.entries(person));
// Output: [["name", "Neeraj"], ["age", 23], ["city", "Lucknow"]]
```

Looping through Object using Object.entries()

```
for (let [key, value] of Object.entries(person)) {
  console.log(`${key}: ${value}`);
}
```

Output:

```
name: Neeraj
age: 23
city: Lucknow
```

Object Destructuring

Destructuring allows you to **extract properties** from an object into **variables**.

Example 1: Basic Destructuring

```
let user = { name: "Neeraj", age: 23, city: "Lucknow" };

let { name, age } = user;

console.log(name); // "Neeraj"
console.log(age);  // 23
```

Example 2: Renaming Variables

```
let { name: fullName, city: hometown } = user;
console.log(fullName);  // "Neeraj"
console.log(hometown);  // "Lucknow"
```

Example 3: Default Values

If a property does not exist, you can assign a default value.

```
let { name, country = "India" } = user;
console.log(country);  // "India"
```