* Advanced Javascript (All ES6 Versions)
  + ✔ Variable Scoping

JavaScript mai **variable scoping** ka matlab hota hai **"ek variable kis jagah tak accessible hai"**, yaani **kis area ya block ke andar aap us variable ko use kar sakte ho.**

JavaScript mai **3 types ke scope** hote hain:

**✅ 1. Global Scope**

* Jo variable function ke bahar banaya jata hai.
* Wo har jagah accessible hota hai (poore code mai).

Example:

let name = "Bilal"; // global scope

function greet() {

console.log(name); // accessible here

}

greet();

console.log(name); // accessible here too

**✅ 2. Function Scope**

* Jo variable function ke andar banaya jata hai, wo sirf us function ke andar hi accessible hota hai.

Example:

function greet() {

let message = "Hello"; // function scope

console.log(message);

}

greet();

// console.log(message); ❌ error: message is not defined

**✅ 3. Block Scope (ES6 ke baad)**

* let aur const se banaye gaye variables {} (curly braces) ke andar hi kaam karte hain.

Example:

{

let a = 10;

const b = 20;

console.log(a, b); // ✅ works here

}

// console.log(a); ❌ error

// console.log(b); ❌ error

**Note:** var block scope ko follow nahi karta, sirf function scope follow karta hai.

**🔄 Summary Table:**

| **Keyword** | **Function Scope** | **Block Scope** |
| --- | --- | --- |
| var | ✅ Yes | ❌ No |
| let | ✅ Yes | ✅ Yes |
| const | ✅ Yes | ✅ Yes |

* + ✔ Closure

**🔒 Closure kya hota hai?**

* **Closure** ka matlab hai:
* Jab koi **inner function** apne **outer function** ke variables ko **yaad rakhta hai aur access kar sakta hai**, **chahe outer function already khatam ho chuka ho** — is behavior ko closure kehte hain.

**✅ Example:**

function outerFunction() {

let name = "Bilal";

function innerFunction() {

console.log("Hello " + name); // 'name' is from outer function

}

return innerFunction;

}

const greet = outerFunction(); // outerFunction executes

greet(); // "Hello Bilal"

* Yahan innerFunction() ek **closure** hai kyun ke wo name variable ko **yaad rakhta hai**, jo outerFunction() ke andar tha, **even after** outerFunction complete ho chuki hai.

**🔍 Real-life Example: Counter**

function createCounter() {

let count = 0;

return function () {

count++;

console.log(count);

};

}

const counter = createCounter();

counter(); // 1

counter(); // 2

counter(); // 3

* Yahan count variable **closure ke through yaad rakha gaya hai**. Har dafa counter() call karne par, wo usi count ko update karta hai.

**💡 Summary:**

| **Point** | **Description** |
| --- | --- |
| Closure kya hai? | Inner function jo outer function ke variables ko access karta hai |
| Kyun useful hai? | Data ko private rakhne ke liye (encapsulation) |
| Kab banta hai? | Jab ek function ke andar doosra function define hota hai |

* + ✔ Template literals

**template literals** JavaScript ka ek feature hai jo **string banane ko easy aur powerful** banata hai. Ye **backticks (`)** ka use karta hai instead of normal quotes (' ' ya " ").

**✅ Template Literals kya hota hai?**

Template literals allow:

1. **String interpolation** (variables ko directly string ke andar lagana)
2. **Multi-line strings** (asaani se multi-line likhna)
3. **Expressions** ka use directly string ke andar

**🔤 1. Basic Syntax**

Example:

let name = "Bilal";

let message = `Hello, ${name}!`;

console.log(message); // Hello, Bilal!

${variable} ka matlab hota hai "variable ko string ke andar inject karna".

**🔁 2. Expressions bhi likh sakte ho**

Example:

let a = 5;

let b = 10;

console.log(`The sum is ${a + b}`); // The sum is 15

**📄 3. Multi-line Strings**

Example:

let msg = `This is

a multi-line

message.`;

console.log(msg);

Agar aap ' ' ya " " use karte to \n likhna padta, lekin **template literals mai seedha multi-line likh sakte ho**.

**🔐 Compare with Normal String:**

Example:

let name = "Bilal";

let normal = "Hello, " + name + "!";

let template = `Hello, ${name}!`;

console.log(normal); // Hello, Bilal!

console.log(template); // Hello, Bilal!

**💡 Use Karne Ka Faida:**

| **Feature** | **Normal String** | **Template Literal** |
| --- | --- | --- |
| Variable insert | ❌ Hard | ✅ Easy with ${} |
| Multi-line string | ❌ \n needed | ✅ No \n needed |
| Expression support | ❌ Not direct | ✅ Yes with ${} |

* + ✔ Destructuring

**Destructuring** JavaScript ka ek **short aur smart** tareeqa hai **objects ya arrays ke andar ke values ko nikaal kar variables mai rakhne** ka.

**✅ Destructuring ka Matlab:**

**Destructuring** ka matlab hai **"todna"** — yaani array ya object ko tod kar uske values ko **alag-alag variables mai assign karna**.

**🔹 1. Array Destructuring**

Example:

const arr = [10, 20, 30];

// Normal way

let a = arr[0];

let b = arr[1];

// Destructuring way

let [x, y, z] = arr;

console.log(x); // 10

console.log(y); // 20

console.log(z); // 30

**🔹 2. Object Destructuring**

Example:

const person = {

name: "Bilal",

age: 25,

};

// Normal way

// let name = person.name;

// let age = person.age;

// Destructuring way

let { name, age } = person;

console.log(name); // Bilal

console.log(age); // 25

**🔹 3. Rename During Destructuring**

Example:

const user = {

username: "bilal123",

email: "bilal@example.com",

};

let { username: userName, email: userEmail } = user;

console.log(userName); // bilal123

console.log(userEmail); // bilal@example.com

**🔹 4. Destructuring Function Parameters**

Example:

function greet({ name, age }) {

console.log(`Hello ${name}, you are ${age} years old`);

}

greet({ name: "Bilal", age: 25 });

**🔹 5. Skipping Items in Array**

Example:

const nums = [1, 2, 3, 4];

const [first, , third] = nums;

console.log(first); // 1

console.log(third); // 3

**💡 Summary:**

| **Type** | **Syntax Example** |
| --- | --- |
| Array | let [a, b] = [1, 2]; |
| Object | let {name, age} = {name: "Ali", age: 20} |
| Rename variable | let {name: myName} = person; |
| Function param | function greet({name}) {} |

* + ✔ Default parameters
* **default parameter** JavaScript ka wo feature hai jo aapko **function ke parameter ke liye ek default value set karne** ki facility deta hai — agar user koi value **na de**, to ye default value use hoti hai.

**✅ Default Parameter Kya Hota Hai?**

Jab function call karte waqt koi argument **na diya jaye**, to **function ek default value use karta hai** — is feature ko **default parameter** kehte hain.

**🔹 Syntax:**

function greet(name = "Guest") {

console.log(`Hello, ${name}`);

}

**🔍 Example 1: Value Di Gayi Hai**

greet("Bilal"); // Hello, Bilal

**🔍 Example 2: Value Nahi Di Gayi**

greet(); // Hello, Guest

Kyun? Kyun ke name ki default value "Guest" hai.

**🧮 Example with Numbers**

function add(a = 1, b = 2) {

return a + b;

}

console.log(add(5, 10)); // 15

console.log(add(5)); // 5 + 2 = 7

console.log(add()); // 1 + 2 = 3

**⚠️ Without Default Parameter (Old Way)**

Pehle aise likhte the:

function greet(name) {

name = name || "Guest";

console.log(`Hello, ${name}`);

}

But ab ES6 ke baad, seedha default set kar sakte ho parameter mai hi:

function greet(name = "Guest") {

console.log(`Hello, ${name}`);

}

**💡 Summary:**

| **Point** | **Example** |
| --- | --- |
| Default value for string | function hi(name = "Ali") |
| Default value for number | function sum(a = 0) |
| Default value used when | Argument is undefined |

* + ✔ Rest parameter

**rest parameter** JavaScript ka ek feature hai jo aapko allow karta hai ke aap **ek hi parameter mai multiple (unlimited) arguments ko collect** kar sako **as an array**.

**✅ Rest Parameter Kya Hota Hai?**

**Rest parameter** ... (3 dots) ka use karta hai, aur function ke andar jitne bhi extra arguments diye gaye ho, un sab ko **ek array ke andar collect** kar leta hai.

**🔹 Syntax:**

function funcName(...rest) {

// rest is an array

}

**🔍 Example 1: Simple Rest Parameter**

function showNames(...names) {

console.log(names);

}

showNames("Bilal", "Ali", "Ahmed");

// Output: ["Bilal", "Ali", "Ahmed"]

Yahan ...names ek array ban gaya jisme saare arguments aa gaye.

**🔍 Example 2: Rest + Regular Parameters**

function greet(greeting, ...names) {

console.log(`${greeting} to: ${names.join(", ")}`);

}

greet("Hello", "Bilal", "Ali", "Sara");

// Output: Hello to: Bilal, Ali, Sara

greeting first argument le raha hai, baaqi sab ...names mai chala gaya.

**🔍 Example 3: Sum Function with Rest Parameter**

function sum(...numbers) {

let total = 0;

for (let num of numbers) {

total += num;

}

return total;

}

console.log(sum(1, 2, 3)); // 6

console.log(sum(5, 10, 15, 20)); // 50

Kitne bhi numbers ho, ...numbers un sab ko array bana deta hai.

**❗ Important Note:**

* **Rest parameter hamesha last mai hota hai**

Example:

function wrong(a, ...b, c) {} // ❌ Error

**💡 Summary:**

| **Feature** | **Description** |
| --- | --- |
| Syntax | function(...args) {} |
| Result | args is an array of all arguments |
| Use Case | Jab arguments ka number unknown ho |
| Rule | Hamesha **function ke last** mai likhna hota hai |

* + ✔ Spread Operator

**spread operator (...)** JavaScript ka ek powerful feature hai jo aapko **array ya object ke elements ya properties ko "phailaane"** (spread karne) ki facility deta hai.

Yeh bilkul **rest parameter jaise dikhta hai (...)**, **lekin kaam alag karta hai**.

**✅ Spread Operator Kya Hota Hai?**

**Spread operator** ka kaam hota hai:  
**"Array ya object ke elements/properties ko ek-ek karke kisi naye array, object, ya function mai daalna."**

**🔹 1. Array Spread**

const arr1 = [1, 2, 3];

const arr2 = [4, 5];

const combined = [...arr1, ...arr2];

console.log(combined); // [1, 2, 3, 4, 5]

...arr1 aur ...arr2 ne dono arrays ke elements ko naya array mai phaila diya.

**🔹 2. Copy Array**

const nums = [10, 20, 30];

const copy = [...nums];

console.log(copy); // [10, 20, 30]

Yeh simple aur safe tareeqa hai array ki **copy banane ka**, bina reference share kiye.

**🔹 3. Spread in Function Arguments**

Example:

function sum(a, b, c) {

return a + b + c;

}

const values = [1, 2, 3];

console.log(sum(...values)); // 6

Array ke values ko function ke arguments mai spread kar diya.

**🔹 4. Object Spread**

Example:

const person = { name: "Bilal", age: 25 };

const updated = { ...person, city: "Karachi" };

console.log(updated);

// { name: 'Bilal', age: 25, city: 'Karachi' }

Existing object ke saath naya data add/update karne ka **best** tareeqa.

**🔹 5. Merging Objects**

Example:

const obj1 = { a: 1 };

const obj2 = { b: 2 };

const merged = { ...obj1, ...obj2 };

console.log(merged); // { a: 1, b: 2 }

**🧠 Spread vs Rest — Difference:**

| **Feature** | **Looks Like** | **Works In** | **Purpose** |
| --- | --- | --- | --- |
| Rest parameter | ... | Function parameters | **Collect** arguments as array |
| Spread operator | ... | Arrays/Objects/Calls | **Expand** values from array/object |

**💡 Summary:**

| **Use Case** | **Example** |
| --- | --- |
| Merge arrays | [...a, ...b] |
| Merge objects | {...obj1, ...obj2} |
| Copy arrays/objects | [...arr], {...obj} |
| Function arguments | func(...array) |

* + ✔ Arrow functions

**arrow function** JavaScript ka ek **short aur modern** tareeqa hai function likhne ka. Yeh ES6 (ECMAScript 2015) mai introduce hua tha, aur isse code **simple aur readable** ban jaata hai.

**✅ Arrow Function Kya Hota Hai?**

**Arrow function** ek short syntax hai jo normal function ko likhne ka alternative hai.  
Iska symbol hota hai: => (arrow)

**🔹 Normal Function vs Arrow Function**

Example:

// Normal function

function add(a, b) {

return a + b;

}

// Arrow function

const add = (a, b) => {

return a + b;

};

**🔹 1. Short One-liner Arrow Function**

Agar function sirf **ek line ka ho** to {} aur return bhi hata sakte ho:

Example:

const square = (x) => x \* x;

console.log(square(5)); // 25

Agar **ek hi parameter** ho to () bhi optional hai:

Example:

const greet = name => `Hello, ${name}`;

console.log(greet("Bilal")); // Hello, Bilal

**🔹 2. No Parameter Function**

Example:

const sayHello = () => console.log("Hello!");

sayHello(); // Hello!

**🔹 3. Arrow Function in Arrays / Loops**

Example:

const numbers = [1, 2, 3, 4];

const doubled = numbers.map(num => num \* 2);

console.log(doubled); // [2, 4, 6, 8]

**⚠️ Arrow Function ka Important Note:**

**❌ this ka behavior change ho jaata hai**

Arrow functions **apna this nahi banate**, wo **parent ka this inherit** karte hain.

Example:

const person = {

name: "Bilal",

greet: () => {

console.log(`Hello, ${this.name}`); // ❌ undefined

}

};

person.greet();

👆 Is case mai this.name undefined ho jata hai.

Isliye arrow functions **methods** ke liye nahi use karte, balkay callbacks, array methods, short functions ke liye use karte hain.

**💡 Summary:**

| **Feature** | **Arrow Function** |
| --- | --- |
| Syntax | const fn = () => {} |
| Shorter | ✅ Yes |
| Auto return | ✅ If one line, no {} needed |
| this Binding | ❌ Does not have its own this |
| Best for | Callbacks, array methods |

* + Enhanced object literals
* **Enhanced Object Literals** JavaScript ka ek modern feature hai jo **object banane ka tareeqa asaan aur powerful** bana deta hai. Yeh feature **ES6 (ECMAScript 2015)** mai introduce hua tha.

**✅ Enhanced Object Literals Kya Hota Hai?**

* Jab aap **object banate ho**, to **shorter syntax**, **dynamic keys**, aur **functions ko directly define karne ka tareeqa** milta hai — is feature ko **Enhanced Object Literals** kehte hain.

**🔥 Features of Enhanced Object Literals:**

**1. Property shorthand**

**2. Method shorthand**

**3. Computed property names**

**🔹 1. Property Shorthand**

Agar variable ka naam aur object key ka naam same ho, to aap sirf ek naam likh saktay ho.

Example:

const name = "Bilal";

const age = 25;

// Old way

const user1 = {

name: name,

age: age

};

// Enhanced way

const user2 = {

name,

age

};

console.log(user2); // { name: 'Bilal', age: 25 }

**🔹 2. Method Shorthand**

Function ko object ke andar define karne ka chhota tareeqa.

Example:

// Old way

const user = {

greet: function () {

console.log("Hello!");

}

};

// Enhanced way

const user2 = {

greet() {

console.log("Hello!");

}

};

user2.greet(); // Hello!

**🔹 3. Computed Property Names**

Aap object ke key ko **dynamic (expression se)** bana sakte ho.

Example:

const key = "email";

const user = {

name: "Bilal",

[key]: "bilal@example.com"

};

console.log(user);

// { name: 'Bilal', email: 'bilal@example.com' }

**🧠 Summary Table:**

| **Feature** | **Example** |
| --- | --- |
| Property shorthand | { name, age } |
| Method shorthand | greet() {} |
| Computed property names | [key]: value |

**📦 Full Example:**

const city = "Karachi";

const population = 15000000;

const info = {

city,

population,

showInfo() {

console.log(`${this.city} has ${this.population} people.`);

},

["capital" + "Status"]: false

};

console.log(info);

// { city: 'Karachi', population: 15000000, showInfo: [Function: showInfo], capitalStatus: false }

info.showInfo(); // Karachi has 15000000 people.

* + Iterators & For..of
* **"Iterators"** aur **for...of loop** dono JavaScript mai **collections (like arrays, strings, etc.) ko step-by-step process karne** ke liye use hote hain.

**✅ Iterator Kya Hota Hai?**

* **Iterator** ek aisa object hota hai jo aapko **step by step** kisi collection (array, string, etc.) ke elements par loop chalane deta hai.  
  Har step par next() function se **ek value** milti hai aur pata chalta hai ke loop khatam hua ke nahi.

**🔹 Simple Example:**

Example:

const arr = [10, 20, 30];

// Iterator banaye:

const iterator = arr[Symbol.iterator]();

console.log(iterator.next()); // { value: 10, done: false }

console.log(iterator.next()); // { value: 20, done: false }

console.log(iterator.next()); // { value: 30, done: false }

console.log(iterator.next()); // { value: undefined, done: true }

done: true ka matlab ke elements khatam ho gaye hain.

**✅ for...of Loop Kya Hota Hai?**

for...of ek loop hai jo automatically **iterator use karta hai** aur **array, string, set, map** jese iterable objects ke har item ko directly access karta hai.

**🔹 Example 1: Array**

Example:

const numbers = [1, 2, 3];

for (const num of numbers) {

console.log(num);

}

// Output:

// 1

// 2

// 3

**🔹 Example 2: String**

Example:

const name = "Bilal";

for (const char of name) {

console.log(char);

}

// Output:

// B

// i

// l

// a

// l

**💡 Summary Table:**

| **Concept** | **Description** |
| --- | --- |
| **Iterator** | Object with next() method to loop items |
| **Iterable** | Any item usable with iterator (array, string, map, set) |
| **for...of** | Easy way to loop over any iterable object |
| for...in | ❗ Loops over object **keys**, not values |

**❗ for...of vs for...in (Don't confuse):**

| **Loop Type** | **Used For** | **Loops Over** |
| --- | --- | --- |
| for...of | Arrays, Strings, Sets, etc. | **Values** |
| for...in | Objects | **Keys / indexes** |

Example:

const obj = { a: 1, b: 2 };

for (const key in obj) {

console.log(key); // a, b

console.log(obj[key]); // 1, 2

}

* + Generators

Bai jan, **Generator** JavaScript ka ek advanced feature hai jo aapko **function ko pause aur resume karne** ki power deta hai.

Ye normal function se alag hota hai, aur **lazy execution** karta hai — mtlb ek waqt mai sirf ek value return karta hai, baaki values baad mai **next()** se milti hain.

**✅ Generator Function Kya Hota Hai?**

Generator function ek **special type ka function** hota hai jo function\* likh kar banaya jaata hai, aur usme yield keyword use hota hai.

**🔹 Syntax:**

js

CopyEdit

function\* generatorName() {

yield value1;

yield value2;

...

}

**🔹 Simple Example:**

js

CopyEdit

function\* greet() {

yield "Hello";

yield "How are you?";

yield "Goodbye";

}

const gen = greet();

console.log(gen.next()); // { value: "Hello", done: false }

console.log(gen.next()); // { value: "How are you?", done: false }

console.log(gen.next()); // { value: "Goodbye", done: false }

console.log(gen.next()); // { value: undefined, done: true }

Har next() call se ek yield wali value milti hai.

**🧠 Key Concepts:**

| **Concept** | **Meaning** |
| --- | --- |
| function\* | Generator function declare karne ka syntax |
| yield | Wahan tak function ko **pause** kar do |
| next() | Execution ko resume karo next yield tak |
| done | true ho jaata hai jab values khatam ho jayein |

**🔹 Generator with Loop Example:**

js

CopyEdit

function\* countUpTo(n) {

for (let i = 1; i <= n; i++) {

yield i;

}

}

const counter = countUpTo(3);

console.log(counter.next()); // { value: 1, done: false }

console.log(counter.next()); // { value: 2, done: false }

console.log(counter.next()); // { value: 3, done: false }

console.log(counter.next()); // { value: undefined, done: true }

**✅ Use Cases (Kahan Use Hota Hai?):**

* **Lazy iteration:** Jab aapko ek waqt mai saari values nahi chahiyein.
* **Custom iterators** banane ke liye.
* **Infinite sequences** banane ke liye.
* **Pause & resume execution**.
* **Asynchronous workflows** (advanced: async generators).

**🔥 Bonus: for...of with Generator**

js

CopyEdit

function\* names() {

yield "Ali";

yield "Bilal";

yield "Usman";

}

for (const name of names()) {

console.log(name);

}

// Output:

// Ali

// Bilal

// Usman

**📌 Summary:**

| **Feature** | **Generator Function** |
| --- | --- |
| Syntax | function\* + yield |
| Execution | next() se hota hai |
| Control | Pause & resume hoti hai |
| Best for | Iteration, streaming, laziness |

**OR**

**Generator JavaScript ka ek special type ka function hota hai jo aapko pause (rukna) aur resume (dobara chalu karna) ki ability deta hai. Yeh normal functions se alag hote hain.**

**✅ Generator Function Kya Hota Hai?**

**Ek generator aisa function hota hai jo apna kaam step-by-step karta hai, aur har step par yield keyword ke zariye ek value deta hai, lekin function khatam nahi hota — usse aap baad mai resume kar sakte ho.**

**🔹 Generator Function Syntax:**

**js**

**CopyEdit**

**function\* generatorName() {**

**yield value1;**

**yield value2;**

**...**

**}**

**Note: function ke baad \* lagana zaroori hai.**

**🔄 Example: Basic Generator**

**js**

**CopyEdit**

**function\* numberGen() {**

**yield 1;**

**yield 2;**

**yield 3;**

**}**

**const gen = numberGen();**

**console.log(gen.next()); // { value: 1, done: false }**

**console.log(gen.next()); // { value: 2, done: false }**

**console.log(gen.next()); // { value: 3, done: false }**

**console.log(gen.next()); // { value: undefined, done: true }**

**gen.next() call har yield value ko return karta hai.**

**📘 Important Concepts**

| **Concept** | **Description** |
| --- | --- |
| **function\*** | **Generator function banata hai** |
| **yield** | **Ek value return karta hai, aur function ko pause kar deta hai** |
| **next()** | **Generator function ko resume karta hai** |
| **done: true** | **Jab sab yield complete ho jaaye** |

**🧠 Generator vs Normal Function**

| **Feature** | **Normal Function** | **Generator Function** |
| --- | --- | --- |
| **return** | **Sirf ek baar result** | **yield se multiple values** |
| **Execution** | **One-time, complete** | **Step-by-step, resumable** |
| **Memory** | **Sab kuch ek sath** | **Thoda-thoda, efficient** |

**🔹 Example: Infinite Generator**

**js**

**CopyEdit**

**function\* infiniteCounter() {**

**let i = 1;**

**while (true) {**

**yield i++;**

**}**

**}**

**const counter = infiniteCounter();**

**console.log(counter.next().value); // 1**

**console.log(counter.next().value); // 2**

**console.log(counter.next().value); // 3**

**// and so on...**

**🔹 Generator with for...of Loop**

**js**

**CopyEdit**

**function\* fruits() {**

**yield "Apple";**

**yield "Banana";**

**yield "Mango";**

**}**

**for (const fruit of fruits()) {**

**console.log(fruit);**

**}**

**// Output:**

**// Apple**

**// Banana**

**// Mango**

**✅ Real-life Use Cases of Generators**

* **Lazy loading of data**
* **Paginated API fetch**
* **Custom iterators**
* **Asynchronous workflows (like Redux-Saga)**

**🔧 Summary**

| **Term** | **Meaning** |
| --- | --- |
| **function\*** | **Generator function** |
| **yield** | **Pause and give a value** |
| **next()** | **Resume the function** |
| **for...of** | **Loop through generator output** |

* + Modules

**JavaScript module** ka matlab hota hai:

**Code ko alag alag files mai divide karna** jese parts (modules) mein taake code **clean, reusable, aur manageable** ban jaye.

JavaScript mai **module system** use hota hai jab hum chahte hain ke:

* Ek file ka function/dataset dusri file mai use ho.
* Code cluttered na ho.
* Har part ka kaam alag se ho.

**✅ Module Kya Hota Hai?**

JavaScript module ek **file** hoti hai jisme kuch variables, functions, classes define kiye jate hain, aur unhe **export** karke kisi aur file mai **import** karke use kiya jata hai.

**🔄 Module Use Karne Ka Syntax**

**🔹 1. export – kisi cheez ko dusre file ke liye bahar bhejna**

**🔹 2. import – dusri file se cheez ko andar lena**

**📁 Example: Basic ES6 Module**

**📝 math.js (Module File)**

js

CopyEdit

export const add = (a, b) => a + b;

export const subtract = (a, b) => a - b;

**📄 main.js (File jahan use hoga)**

js

CopyEdit

import { add, subtract } from './math.js';

console.log(add(2, 3)); // 5

console.log(subtract(5, 2)); // 3

**🧾 Notes:**

* File extension .js hona zaroori hai.
* HTML file mai <script type="module" src="main.js"></script> likhna hoga.
* Modules **default** async hote hain.

**🔹 Default Export**

Agar aap sirf **ek default** value/function export karna chahte ho:

**greet.js**

js

CopyEdit

export default function greet(name) {

return `Hello, ${name}`;

}

**main.js**

js

CopyEdit

import greet from './greet.js';

console.log(greet("Bilal")); // Hello, Bilal

**💡 Why Use Modules?**

| **Fayda** | **Wajah** |
| --- | --- |
| Reusability | Ek function ko multiple files mai use kar sakte ho |
| Maintainability | Code structured aur readable hota hai |
| Avoid global variables | Har file ka scope alag hota hai |
| Team collaboration friendly | Har developer apna module bana sakta hai |

**❗ CommonJS vs ES6 Modules**

| **Feature** | **CommonJS (require)** | **ES6 Modules (import/export)** |
| --- | --- | --- |
| Use in Node.js | ✅ | ✅ (recent versions) |
| Use in Browser | ❌ (direct nahi) | ✅ via <script type="module"> |
| Export Syntax | module.exports | export |
| Import Syntax | require() | import |

**✅ Example with CommonJS (Node.js only)**

js

CopyEdit

// math.js

function add(a, b) {

return a + b;

}

module.exports = add;

// main.js

const add = require('./math');

console.log(add(3, 4)); // 7

* + Map

JavaScript mein **Map** ek built-in object hai jo **key-value pairs** ko store karta hai — lekin ye **normal object se zyada powerful** hota hai.

**✅ Map Kya Hota Hai?**

Map ek **collection of key-value pairs** hota hai jahan **keys kisi bhi type ki ho sakti hain** — number, string, object, even function.

**🆚 Object vs Map**

| **Feature** | **Object** | **Map** |
| --- | --- | --- |
| Key types allowed | Sirf string ya symbol | **Kisi bhi type ka key** |
| Key order | Not guaranteed | **Insertion order maintained** |
| Iteration | Complex | **Easy with for...of** |
| Size count | Manual (Object.keys().length) | **map.size hota hai** |

**🔹 Map Create Karna**

js

CopyEdit

const myMap = new Map();

**🔹 Value Set Karna (add key-value pair)**

js

CopyEdit

myMap.set('name', 'Bilal');

myMap.set('age', 25);

myMap.set(true, 'Active');

myMap.set({ id: 1 }, 'Object Key');

**🔹 Value Get Karna**

js

CopyEdit

console.log(myMap.get('name')); // Bilal

console.log(myMap.get(true)); // Active

**🔹 Size Pata Karna**

js

CopyEdit

console.log(myMap.size); // 4

**🔹 Check if Key Exists**

js

CopyEdit

console.log(myMap.has('age')); // true

**🔹 Delete Key**

js

CopyEdit

myMap.delete('age');

**🔹 Clear All Entries**

js

CopyEdit

myMap.clear();

**🔹 Loop Through Map**

js

CopyEdit

const userInfo = new Map();

userInfo.set('name', 'Ali');

userInfo.set('city', 'Lahore');

for (const [key, value] of userInfo) {

console.log(`${key}: ${value}`);

}

// Output:

// name: Ali

// city: Lahore

**✅ Real-Life Example:**

js

CopyEdit

const students = new Map();

students.set(1, 'Ahmed');

students.set(2, 'Sara');

students.set(3, 'Zara');

console.log(students.get(2)); // Sara

**🧠 Jab Map Use Karte Hain:**

* Jab keys **non-string types** ho sakti hain.
* Jab aapko **ordered data chahiye**.
* Jab aapko frequently **add/get/delete** operations perform karne hote hain.
  + Array methods
* **Array methods** JavaScript ke wo built-in functions hote hain jo **array ke data ko process karne**, **manipulate karne** aur **search/filter/map** karne ke liye use hote hain.
* Inse hum array ke andar ki values ke sath aasani se kaam kar sakte hain — bina manually loop lagaye.
* **✅ Commonly Used Array Methods (with Examples)**
* **🔹 1. push() – Array ke end mai value add karta hai**
* js
* CopyEdit
* let arr = [1, 2];
* arr.push(3);
* console.log(arr); // [1, 2, 3]
* **🔹 2. pop() – Last element ko remove karta hai**
* js
* CopyEdit
* arr.pop();
* console.log(arr); // [1, 2]
* **🔹 3. shift() – Pehla element hata deta hai**
* js
* CopyEdit
* arr.shift();
* console.log(arr); // [2]
* **🔹 4. unshift() – Array ke start mai value add karta hai**
* js
* CopyEdit
* arr.unshift(0);
* console.log(arr); // [0, 2]
* **🔹 5. concat() – Do arrays ko mila deta hai**
* js
* CopyEdit
* const a = [1, 2];
* const b = [3, 4];
* const result = a.concat(b);
* console.log(result); // [1, 2, 3, 4]
* **🔹 6. join() – Array ko string mai convert karta hai**
* js
* CopyEdit
* const items = ["apple", "banana"];
* console.log(items.join(", ")); // "apple, banana"
* **🔹 7. slice(start, end) – Part of array return karta hai (original array ko change nahi karta)**
* js
* CopyEdit
* const arr = [10, 20, 30, 40];
* console.log(arr.slice(1, 3)); // [20, 30]
* **🔹 8. splice(index, deleteCount, ...addItems) – Element delete ya insert karta hai (original array ko change karta hai)**
* js
* CopyEdit
* const arr = [1, 2, 3, 4];
* arr.splice(2, 1, 99); // index 2 se 1 item delete kar ke 99 add karo
* console.log(arr); // [1, 2, 99, 4]
* **🔹 9. indexOf(value) – Value ka index return karta hai**
* js
* CopyEdit
* const arr = [5, 10, 15];
* console.log(arr.indexOf(10)); // 1
* **🔹 10. includes(value) – Value hai ya nahi (true/false)**
* js
* CopyEdit
* console.log(arr.includes(15)); // true
* **✅ Advanced Array Methods**
* **🔹 11. map() – Har item pe function apply karke naya array banata hai**
* js
* CopyEdit
* const nums = [1, 2, 3];
* const doubled = nums.map(n => n \* 2);
* console.log(doubled); // [2, 4, 6]
* **🔹 12. filter() – Jo items condition satisfy karein unka naya array return karta hai**
* js
* CopyEdit
* const nums = [1, 2, 3, 4];
* const evens = nums.filter(n => n % 2 === 0);
* console.log(evens); // [2, 4]
* **🔹 13. find() – Pehla matching item return karta hai**
* js
* CopyEdit
* const nums = [3, 6, 9];
* const found = nums.find(n => n > 5);
* console.log(found); // 6
* **🔹 14. forEach() – Har item pe kaam karta hai (lekin return nahi karta)**
* js
* CopyEdit
* const fruits = ["apple", "banana"];
* fruits.forEach(f => console.log(f));
* **🔹 15. reduce() – Saare elements ko ek hi value mai reduce karta hai**
* js
* CopyEdit
* const nums = [1, 2, 3, 4];
* const total = nums.reduce((acc, curr) => acc + curr, 0);
* console.log(total); // 10
* **🔥 Bonus: Array from String**
* js
* CopyEdit
* const str = "HELLO";
* console.log(Array.from(str)); // ['H', 'E', 'L', 'L', 'O']
* **📌 Summary Table:**

| **Method** | **Purpose** |
| --- | --- |
| push() | Add at end |
| pop() | Remove from end |
| shift() | Remove from start |
| unshift() | Add at start |
| map() | Return new array with changes |
| filter() | Return new array with filtered items |
| find() | Return first match |
| forEach() | Loop through each item |
| reduce() | Reduce to single value |

* + Higher-order function

**High Order Function** (yaani HOF) JavaScript ka ek **important concept** hai jo functions ke sath **flexible aur powerful coding** karne deta hai.

**✅ High Order Function Kya Hota Hai?**

**High Order Function** woh function hota hai jo:

1. **Ek function ko argument ke tor pe accept karta hai**, **ya**
2. **Ek function ko return karta hai**.

**🔹 Definition (Simple Words Mai):**

**Function jo dusre function ko le ya return kare = High Order Function**

**🔧 Example 1: Function ko as Argument dena**

js

CopyEdit

function greet(name) {

return "Hello " + name;

}

function processUser(name, callback) {

return callback(name);

}

console.log(processUser("Bilal", greet)); // Hello Bilal

✅ Yahan processUser ek high order function hai kyun ke usne greet function ko argument ke tor pe liya.

**🔧 Example 2: Function ko Return Karna**

js

CopyEdit

function multiplier(factor) {

return function(num) {

return num \* factor;

};

}

const double = multiplier(2);

console.log(double(5)); // 10

✅ multiplier ek **high order function** hai kyun ke usne ek function **return** kiya.

**🎯 Common JavaScript High Order Functions:**

Ye built-in methods bhi high order functions hain:

| **Method** | **Description** |
| --- | --- |
| map() | Har element pe operation, returns array |
| filter() | Condition pass karne wale items return |
| reduce() | All items ko combine kar ke ek value |
| forEach() | Har element pe kaam karta hai |
| sort() | Sorting ke liye function leta hai |

**🔹 Example with map():**

js

CopyEdit

const nums = [1, 2, 3];

const doubled = nums.map(function(n) {

return n \* 2;

});

console.log(doubled); // [2, 4, 6]

✅ map() ne ek function liya as argument — so map() is a high order function.

**📌 Summary:**

| **Feature** | **High Order Function** |
| --- | --- |
| Takes function as input | ✅ |
| Returns function as output | ✅ |
| Example | map(), filter(), custom HOF |

**🔥 Real-Life Example:**

js

CopyEdit

function calculate(a, b, operation) {

return operation(a, b);

}

function add(x, y) {

return x + y;

}

console.log(calculate(5, 3, add)); // 8

✅ calculate() yahan high order function hai.

* + CallBack

**callback function** JavaScript ka ek basic aur important concept hai — especially jab hum asynchronous kaam karte hain (jaise API fetch, setTimeout, events waghera).

**✅ Callback Function Kya Hota Hai?**

**Callback function** wo function hota hai jo **kisi doosre function ko argument ke taur pe diya jata hai**, aur **wo baad mein execute hota hai**.

Matlab:

**Function ke andar function** – aur andar wala function **baad mein call hota hai**, jab kaam ready ho jaye.

**🧠 Simple Definition:**

**Callback** = Function jo **"call back"** hota hai **baad mein**, jab kaam ho chuka ho.

**🔧 Simple Example**

js

CopyEdit

function greetUser(name, callback) {

console.log("Hello " + name);

callback(); // yeh baad mein chalta hai

}

function sayBye() {

console.log("Goodbye!");

}

greetUser("Bilal", sayBye);

**Output:**

nginx

CopyEdit

Hello Bilal

Goodbye!

Yahan sayBye ek **callback function** hai jo greetUser ke andar **argument ke tor pe** gaya hai, aur **baad mein call** hua.

**⏱️ Example with setTimeout (Asynchronous)**

js

CopyEdit

setTimeout(function () {

console.log("3 seconds baad chala!");

}, 3000);

Yahan function () { ... } ek callback hai — jo 3 seconds baad **automatically call hota hai**.

**🔁 Real-life Example: Array Methods ke sath**

js

CopyEdit

const numbers = [1, 2, 3];

numbers.forEach(function (num) {

console.log(num);

});

Yahan function(num) ek callback hai jo har element ke liye call hota hai.

**📌 Summary Table**

| **Feature** | **Description** |
| --- | --- |
| Kya hota hai? | Function jo dusre function mein pass hota hai |
| Kab chalta hai? | Jab main function usse call karta hai |
| Use hota hai | Async tasks, events, array methods |
| Examples | setTimeout(), forEach(), fetch() |

**✅ Use Cases of Callback Functions**

* setTimeout, setInterval
* Array.map, Array.filter, Array.forEach
* API calls (like fetch)
* Event listeners (button.addEventListener)
  + Promises

**Promises** JavaScript ka ek powerful feature hai jo asynchronous kaam (jo baad mein complete hota hai — jaise data fetch karna, file load karna, etc.) ko **asani se handle** karne ke liye use hota hai.

**✅ Promise Kya Hota Hai?**

**Promise** ek JavaScript object hota hai jo future ka result represent karta hai — **jo ya to mil jayega (resolve), ya error aayega (reject)**.

Aise samjho:

**"Aap ne kisi se wada kiya (promise), ya to wo wada pura hoga, ya nahi hoga."**

**🔁 Promise ke 3 States**

| **State** | **Matlab** |
| --- | --- |
| pending | Jab tak result nahi mila |
| fulfilled | Jab kaam sahi se complete ho gaya |
| rejected | Jab kaam fail ho gaya (error aayi) |

**🔧 Simple Example (Banana aur Use Karna)**

js

CopyEdit

const promise = new Promise((resolve, reject) => {

let success = true;

if (success) {

resolve("Kaam hogaya!");

} else {

reject("Kaam fail hogaya!");

}

});

promise

.then((result) => console.log(result)) // jab resolve ho

.catch((error) => console.log(error)); // jab reject ho

**✅ Real-Life Example: setTimeout ke sath**

js

CopyEdit

function delay() {

return new Promise((resolve) => {

setTimeout(() => {

resolve("3 seconds baad kaam hua");

}, 3000);

});

}

delay().then((msg) => console.log(msg));

**Output (3 seconds baad):**

CopyEdit

3 seconds baad kaam hua

**🔁 Compare: Callback vs Promise**

**Callback Hell Example:**

js

CopyEdit

setTimeout(() => {

console.log("1");

setTimeout(() => {

console.log("2");

}, 1000);

}, 1000);

**Promise Style:**

js

CopyEdit

new Promise((res) => {

setTimeout(() => res("1"), 1000);

})

.then((msg) => {

console.log(msg);

return new Promise((res) => setTimeout(() => res("2"), 1000));

})

.then((msg) => console.log(msg));

**📌 Summary Table**

| **Feature** | **Detail** |
| --- | --- |
| Object Type | Built-in JS object |
| Purpose | Async kaam ko track karna |
| States | Pending → Fulfilled / Rejected |
| Methods Used | .then(), .catch(), .finally() |
| Benefit | Callback hell se bachata hai |

**📘 Useful Methods:**

| **Method** | **Description** |
| --- | --- |
| .then() | Jab promise resolve ho jaye |
| .catch() | Jab promise reject ho jaye (error) |
| .finally() | Har halat mein chalega (end main) |

Bhai jan agar aap chaho to mai:

* fetch() ke sath promise example
* API practice task
* ya async/await ka samajh

Sab kuch de sakta hoon. Batao kya chahiye?

* + Exponentiation Operator
* **Exponentiation Operator (\*\*)** JavaScript mein use hota hai **power nikalne** ke liye — matlab:
* a \*\* b ka matlab hai:  
  **a ko b martaba multiply karna** (a raised to the power b)
* **✅ Basic Syntax**
* js
* CopyEdit
* base \*\* exponent
* **🔹 Example:**
* js
* CopyEdit
* console.log(2 \*\* 3); // 8
* // 2 \* 2 \* 2 = 8
* **🔹 Kuch aur Examples:**
* js
* CopyEdit
* console.log(3 \*\* 2); // 9 (3\*3)
* console.log(5 \*\* 4); // 625 (5\*5\*5\*5)
* console.log(10 \*\* 0); // 1 (kisi bhi number ki power 0 = 1)
* **🔹 Negative Exponent:**
* js
* CopyEdit
* console.log(2 \*\* -2); // 0.25
* // 1 / (2\*2) = 1/4 = 0.25
* **🔹 With Variables:**
* js
* CopyEdit
* let base = 4;
* let exponent = 2;
* let result = base \*\* exponent;
* console.log(result); // 16
* **🔸 Ye operator ES2016 (ES7) mein introduce hua tha.**
* **📌 Math.pow vs Exponentiation**

| **Method** | **Example** | **Result** |
| --- | --- | --- |
| Math.pow() | Math.pow(2, 3) | 8 |
| \*\* Operator | 2 \*\* 3 | 8 |

* **Dono ka kaam same hai**, lekin \*\* operator likhne mein asaan aur clean hai.
  + Classes

**Classes** JavaScript mein ek **template** hoti hain objects banane ke liye. Ye aapko **object-oriented programming (OOP)** karne mein madad deti hain — jisme aap **objects aur unke behavior ko define** kar sakte ho.

**✅ Class Kya Hoti Hai?**

Class ek blueprint hoti hai — jisme aap properties (data) aur methods (functions) define karte ho.  
Us blueprint se aap **multiple objects** bana sakte ho.

**🔧 Example:**

js

CopyEdit

class Person {

constructor(name, age) {

this.name = name;

this.age = age;

}

greet() {

console.log(`Hello, my name is ${this.name} and I am ${this.age} years old.`);

}

}

// Object banana (Instance)

const person1 = new Person("Bilal", 25);

person1.greet(); // Output: Hello, my name is Bilal and I am 25 years old.

**🔍 Parts of a Class**

| **Part** | **Description** |
| --- | --- |
| class | Keyword to create a class |
| constructor() | Special function jo object banate waqt chalti hai |
| this | Current object ko refer karta hai |
| Methods | Class ke andar ke functions |

**🧠 Aise Samjho:**

Agar class ek **“car ka design”** hai, to aap us design se **bohot sari cars (objects)** bana sakte ho — har car alag ho sakti hai, lekin structure same hoga.

**🔁 Example: Multiple Objects**

js

CopyEdit

const person2 = new Person("Ahmed", 30);

person2.greet(); // Hello, my name is Ahmed and I am 30 years old.

**🧱 Methods Inside Class**

js

CopyEdit

class Rectangle {

constructor(width, height) {

this.width = width;

this.height = height;

}

area() {

return this.width \* this.height;

}

}

const rect = new Rectangle(5, 4);

console.log(rect.area()); // 20

**🧬 Class Inheritance (Agar chaho to advance mein samjha dunga)**

Child class parent class se inherit kar sakti hai using extends.

js

CopyEdit

class Animal {

speak() {

console.log("Animal speaks");

}

}

class Dog extends Animal {

speak() {

console.log("Dog barks");

}

}

const dog = new Dog();

dog.speak(); // Dog barks

**📌 Summary**

| **Feature** | **Description** |
| --- | --- |
| class | Blueprint/template for object |
| constructor() | Run hota hai object banate waqt |
| this | Refers to current object |
| method() | Function defined inside class |
| extends | Inherit from another class |

Agar aap chaho to:

* Main aapko **assignment** de sakta hoon classes par.
* Ya **constructor ke andar logic** samjha sakta hoon.
* Ya aapko inheritance ka practical use bhi dikha sakta hoon.
  + Ternary Operator

**Ternary Operator** JavaScript ka ek short form **if...else condition** likhne ka tareeqa hai. Ye ek **single line** mein condition check karke result return karta hai.

**✅ Definition:**

**Ternary Operator** ek short-hand hota hai:  
condition ? value\_if\_true : value\_if\_false

**🔹 Syntax:**

js

CopyEdit

condition ? expression1 : expression2

* Agar **condition true** ho → expression1 chalega
* Agar **condition false** ho → expression2 chalega

**🔧 Example 1:**

js

CopyEdit

let age = 20;

let result = age >= 18 ? "Adult" : "Minor";

console.log(result); // Adult

**🔧 Example 2:**

js

CopyEdit

let isLoggedIn = false;

let message = isLoggedIn ? "Welcome!" : "Please log in";

console.log(message); // Please log in

**🔧 Example 3 (inside console directly):**

js

CopyEdit

let num = 5;

console.log(num % 2 === 0 ? "Even" : "Odd"); // Odd

**📌 Ternary Operator vs if...else:**

**Normal if...else:**

js

CopyEdit

if (age >= 18) {

result = "Adult";

} else {

result = "Minor";

}

**Same using Ternary:**

js

CopyEdit

result = age >= 18 ? "Adult" : "Minor";

✅ **Short, clean and readable** — that’s the power of ternary.

**🧠 Nested Ternary (Advanced):**

js

CopyEdit

let score = 75;

let grade = score >= 90

? "A"

: score >= 80

? "B"

: score >= 70

? "C"

: "Fail";

console.log(grade); // C

⚠️ Nested ternary readable hona chahiye warna confuse kar sakta hai.

**🎯 Summary Table**

| **Feature** | **Description** |
| --- | --- |
| Name | Ternary Operator |
| Syntax | condition ? true : false |
| Used for | Short if-else |
| Returns | Value based on condition |

* + Optional chaining

**optional chaining (?.)** JavaScript ka ek feature hai jo aapko safely deeply nested object properties access karne deta hai **bina error diye**, agar beech mein koi property undefined ya null ho.

**✅ Simple Definition:**

Agar koi property **exist nahi karti**, to undefined return karega **instead of throwing an error**.

**🔧 Syntax:**

js

CopyEdit

object?.property

object?.[key]

object?.method?.()

**🔥 Problem Without Optional Chaining:**

js

CopyEdit

let user = {};

console.log(user.address.city); // ❌ Error: Cannot read property 'city' of undefined

**✅ With Optional Chaining:**

js

CopyEdit

let user = {};

console.log(user.address?.city); // ✅ undefined (no error)

Yahan user.address undefined hai, lekin .city access karte waqt error nahi aaya — undefined return hua.

**🔹 Example with Nested Objects:**

js

CopyEdit

let student = {

name: "Bilal",

contact: {

email: "bilal@example.com"

}

};

console.log(student.contact?.email); // ✅ bilal@example.com

console.log(student.address?.city); // ✅ undefined (no crash)

**🔹 Optional Chaining with Functions:**

js

CopyEdit

let user = {

sayHi() {

return "Hello!";

}

};

console.log(user.sayHi?.()); // ✅ Hello!

console.log(user.sayBye?.()); // ✅ undefined (function doesn't exist)

**🔹 With Arrays:**

js

CopyEdit

let data = null;

console.log(data?.[0]); // ✅ undefined

**📌 Summary Table:**

| **Feature** | **Description** |
| --- | --- |
| Operator | ?. (optional chaining) |
| Prevents error | Jab property/method undefined/null ho |
| Returns | undefined instead of error |
| Works on | Objects, arrays, methods |

**⚠️ Important Note:**

* ?. **only checks for undefined or null**.
* Agar value false, 0, '' ho to wo **valid** mana jata hai.

js

CopyEdit

let obj = { value: 0 };

console.log(obj?.value); // ✅ 0

* TypeScript
  + TS Compiler
  + Type Annotations
  + Interfaces
  + Classes
  + Generics
  + Enums
  + Type Inference
  + Union and Intersection Types
  + Type Guards
  + Decorators
* Advance Github
  + Branches
  + PR
  + PR review
  + Merge
  + Rebase
  + Cherry Pick
* GSAP Animations
* Supabase
  + Authentication
    - Email
    - Google
  + Database
  + Storage