JavaScript Interview

(Question and Answer)

1. JavaScript Basics

Q1. Difference between `var`, `let` aur `const`?

`var` function-scoped hota hai aur redeclare bhi ho sakta hai. `let` aur `const` block-scoped hote hain. `let` reassign ho sakta hai but redeclare same scope mei nahi. `const` ek baar value assign karne ke baad change nahi hota.

Q2. Difference between `null` aur `undefined`?

`undefined` ka matlab variable declare hua but usko value assign nahi hui. `null` ek intentional empty value hota hai jo developer manually assign karta hai.

Q3. Difference between == aur ===?

== loose equality hai jo type coercion karta hai (matlab automatically type convert kar ke compare karega).

=== strict equality hai jo type aur value dono check karta hai.

Example:

```
console.log(5 == "5");   // true (string ko number banaya)
console.log(5 === "5");  // false (type alag hai)
```

Q4. Primitive vs Reference data types?

Primitive types (string, number, boolean, null, undefined, symbol, bigint) value ke through store hote hain.

Reference types (object, array, function) memory address ke reference ke through store hote hain.

Q5. Type coercion vs Type conversion?

Type coercion automatic conversion hai jo JS khud kar leta hai.

Type conversion developer manually karta hai jaise `Number("123")`.

Example:

```
console.log("5" + 2); // "52" (number -> string coercion)
console.log(Number("5") + 2); // 7 (manual conversion)
```

2. Control Flow

Q6. Difference between `for...of` aur `for...in`?

`for...in` keys ya indexes pe iterate karta hai.

`for...of` values pe iterate karta hai.

Example:

```
let arr = ["a", "b", "c"];
for (let i in arr) console.log(i);   // 0,1,2
for (let v of arr) console.log(v);   // a,b,c
```

Q7. Difference between `break` aur `continue`?

`break` loop ko turant exit kar deta hai.

`continue` current iteration skip karke next iteration chala jata hai.

Q8. Agar `switch` mei break na lagaye to kya hoga?

Agar break na ho to matching case se le kar next saare cases execute hote hain (fall-through behavior).

Q9. Difference between `while` aur `do...while` loop?

`while` pehle condition check karta hai fir run hota hai.

`do...while` kam se kam ek baar run hota hai chahe condition false hi ho.

Example:

```
let i = 5;
do {
  console.log(i);
  i++;
} while(i < 5);
// output: 5 (kam se kam ek baar run hua)
```

3. Functions

Q10. Difference between Function Declaration aur Function Expression?

Function declaration hoisted hota hai matlab code ke top pe chala jata hai aur usko before define call kar sakte ho.

Function expression hoisted nahi hota, usko define karne ke baad hi use kar sakte ho.

Q11. Arrow functions kya hote hain aur `this` kaise handle karte hain?

Arrow functions short syntax hote hain aur unka apna `this` nahi hota. Wo lexical scope ka `this` lete hain.

Example:

```
let obj = {
  name:"Aman",
  normal: function(){ console.log(this.name); },
  arrow: () => console.log(this.name)
};
obj.normal(); // Aman
obj.arrow();  // undefined (lexical this)
```

Q12. Callback function kya hota hai?

Ek function jo doosre function ko argument ke form mei diya jata hai aur baad mei execute hota hai.

Q13. IIFE (Immediately Invoked Function Expression) kya hai?

Ye ek function hota hai jo turant execute hota hai jaise hi define karte ho. Mostly private scope banane ke liye use hota hai.

Example:

```
(function(){
  console.log("IIFE executed!");
})();
```

Q14. Higher-order function kya hota hai?

Function jo ek function ko input le ya ek function return kare usko higher-order function bolte hain.

Q15. `setTimeout` aur `setInterval` mei difference?

`setTimeout` ek baar run hota hai specific delay ke baad.

`setInterval` har interval ke baad repeat hota hai.

Example:

```
setTimeout(()=>console.log("Runs once after 2s"),2000);
let id=setInterval(()=>console.log("Runs every 1s"),1000);
setTimeout(()=>clearInterval(id),4000); // stop after 4s
```

Q16. Function hoisting kya hai?

Function declarations code ke top pe le jaye jate hain runtime pe. Is wajah se unhe pehle define karne ki zarurat nahi.

Q17. Pure function kya hota hai?

Function jo same input pe hamesha same output deta hai aur koi side-effect nahi karta (jaise global state change).

Q18. Default parameters kya hote hain?

Agar function call mei argument pass na karo to default value use ho jati hai.

Q19. Rest operator aur Spread operator mei kya difference hai?

Rest operator values ko collect karta hai array mei.

Spread operator array/object ke elements ko spread karta hai.

Example:

```
function sum(...nums){ return nums.reduce((a,b)=>a+b,0); }
console.log(sum(1,2,3)); // 6

let arr=[1,2,3];
```

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

Q20. Function currying kya hoti hai?

Function jo ek argument le kar ek naya function return karta hai jab tak saare arguments complete na ho jayein.

4. Arrays & Objects

Q21. Commonly used array methods kaunse hain?

JavaScript mei arrays ke liye bohot powerful built-in methods diye gaye hain jo daily life mei use hote hain:

- `map()` → ek naya array banata hai jisme har element transform hota hai.
- `filter()` → condition true hone wale elements return karta hai.
- `reduce()` → array ko single value mei reduce karta hai (sum, product etc).
- `forEach()` → sirf loop chalata hai, return nahi karta.
- `find()` → first element jo condition satisfy kare.
- `some()` → agar ek bhi element condition satisfy kare to true.
- `every()` → agar saare elements condition satisfy kare to true.
- `sort()` → array ko sort karta hai (by default string sorting hoti hai).

Example:

```
let nums = [1,2,3,4];
let doubled = nums.map(x => x*2);
console.log(doubled); // [2,4,6,8]
```

Q22. Difference between `map` aur `forEach`?

Dono iteration ke liye use hote hain lekin kaafi difference hai:

- `map()` naya array return karta hai jisme har element modified hota hai.
- `forEach()` kuch return nahi karta, sirf iteration karne ke liye hai.

Best practice: agar transformation chahiye to `map()`, sirf looping ke liye `forEach()`.

Q23. `reduce` method ka use kya hai?

`reduce()` ka use tab hota hai jab tumhe array ke saare elements ko combine karke ek single output chahiye. Ye function accumulator aur current value ko le kar ek result build karta hai.

Use cases:

- Sum of array
- Max/Min nikalna
- Object grouping

Example:

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

Q24. Difference between shallow copy aur deep copy?

JavaScript mei objects reference type hote hain.

- **Shallow copy**: sirf ek level tak copy hota hai. Agar object ke andar nested object hai to wo same reference share karega.
- **Deep copy**: pura ka pura object independent ban jata hai, even nested properties bhi alag ho jati hain.

Shallow copy banane ke examples: `Object.assign({}, obj)` ya spread `{...obj}`

Deep copy ke liye: `structuredClone(obj)` (modern JS), ya `JSON.parse(JSON.stringify(obj))`.

Q25. Object destructuring kya hai?

Object destructuring ek short syntax hai jisme tum object ke properties ko directly variables mei extract kar lete ho instead of `obj.prop`.

Example:

```
let user = {name:"Aman", age:22};
let {name, age} = user;
console.log(name, age); // Aman 22
```

Ye feature code readability aur clean syntax ke liye bahut important hai.

Q26. Spread operator ka use kya hai?

Spread (`...`) operator ka use array ya object ke elements ko "expand" karne ke liye hota hai.

Use cases:

- Array copy ya merge karna
- Object copy/merge karna
- Function arguments pass karna

Example:

```
let arr = [1,2,3];
let arr2 = [...arr, 4]; // [1,2,3,4]
```

Q27. Rest operator ka use kya hai?

Rest (`...`) operator reverse concept hai spread ka. Ye multiple values ko ek array mei collect karta hai. Mostly function parameters mei use hota hai.

Example:

```
function sum(...nums){
  return nums.reduce((a,b)=>a+b,0);
}
console.log(sum(1,2,3,4)); // 10
```

Q28. Object.freeze aur Object.seal mei difference?

- `Object.freeze(obj)` → object ko completely lock kar deta hai. Tum usme new properties add, delete ya modify nahi kar sakte.
- `Object.seal(obj)` → tum nayi properties add/delete nahi kar sakte, lekin existing properties modify kar sakte ho.

Q29. Difference between pass-by-value aur pass-by-reference in JS?

- **Primitives (string, number, boolean, null, undefined, symbol, bigint)** → pass by value hote hain. Matlab copy ban kar jata hai.
- **Objects/arrays/functions** → pass by reference hote hain. Matlab memory ka same address share karte hain.

Example:

```
let obj1 = {x:1};
let obj2 = obj1;
obj2.x = 5;
console.log(obj1.x); // 5 (same reference share hua)
```

Q30. Difference between `Object.keys()`, `Object.values()`, `Object.entries()`?

- `Object.keys(obj)` → array of keys return karta hai.
- `Object.values(obj)` → array of values return karta hai.
- `Object.entries(obj)` → array of `[key,value]` pairs return karta hai.

5. DOM Manipulation

Q31. DOM elements select karne ke tarike?

Tum alag alag methods use kar sakte ho:

- `getElementById("id")` → single element by ID
- `getElementsByClassName("class")` → HTMLCollection of elements
- `querySelector("cssSelector")` → first match
- `querySelectorAll("cssSelector")` → NodeList of matches

Example:

```
let btn = document.querySelector("#btn");
btn.addEventListener("click", ()=>alert("Clicked!"));
```

Q32. Difference between `innerText` aur `innerHTML`?

- `innerText` sirf text content return karta hai aur CSS ke hisaab se hidden text ignore karega.
- `innerHTML` pura HTML string return karega including tags.

Q33. Event bubbling aur capturing kya hai?

- **Bubbling**: jab tum ek element pe click karte ho to event pehle us element pe trigger hota hai fir uske parent, fir grandparent tak propagate hota hai. Ye default behavior hai.
- **Capturing**: opposite direction, event pehle parent se start hota hai fir child tak aata hai.

Example:

```
document.getElementById("child")
.addEventListener("click",()=>console.log("Child clicked"));

document.getElementById("parent")
.addEventListener("click",()=>console.log("Parent clicked"));
```

Q34. Event delegation kya hota hai?

Jab tum directly har child element pe listener na lagakar ek parent element pe listener lagate ho aur child ke events ko us parent se handle karte ho, ise event delegation kehte hain. Ye fast aur memory efficient hota hai especially jab new elements dynamically add ho rahe ho.

Q35. Difference between `style` aur `classList`?

- `style` property inline CSS directly set karta hai.
- `classList` CSS classes ko add/remove/toggle karne ke liye use hota hai.

Q36. Difference between `addEventListener` aur inline `onclick`?

- Inline `onclick` ek hi handler allow karta hai aur agar tum baar baar assign karte ho to pehle wala overwrite ho jata hai.
- `addEventListener` ek hi element pe multiple event handlers allow karta hai aur unhe remove bhi kar sakte ho.

Example:

```
let btn = document.getElementById("btn");
btn.onclick = ()=>console.log("1");
btn.onclick = ()=>console.log("2"); // overwrite

btn.addEventListener("click",()=>console.log("1"));
btn.addEventListener("click",()=>console.log("2")); // dono run honge
```

Q37. Difference between `querySelector` aur `querySelectorAll`?

`querySelector` sirf first match return karta hai.

`querySelectorAll` saare matching elements NodeList ke form mei return karta hai.

Q38. Kaise check karoge agar ek element viewport mei visible hai ya nahi?

`getBoundingClientRect()` ka use karke element ke position check kar sakte ho aur window ke size ke saath compare karte ho.

Example:

```
function isVisible(el){
  let rect = el.getBoundingClientRect();
  return rect.top >= 0 && rect.bottom <= window.innerHeight;
}
```

Q39. LocalStorage aur SessionStorage mei difference?

- `localStorage` browser ke local storage mei data permanently store karta hai (browser band karke reopen karne ke baad bhi data rahega).
- `sessionStorage` data ko sirf ek tab ke session tak rakhta hai. Tab close karne ke baad data remove ho jata hai.

Q40. Cookies aur localStorage mei kya difference hai?

- Cookies chhote data ke liye hote hain (max ~4KB) aur har HTTP request ke saath server ko bheje jate hain. Mostly authentication/session ke liye use hote hain.
- LocalStorage sirf client side ke liye hota hai, server pe nahi jata, aur ~5MB tak data store kar sakte ho.

5. DOM Manipulation

Q41. document.getElementById aur querySelector mei kya difference hai?

`getElementById` sirf ek element return karega jo specific `id` se match ho.

`querySelector` CSS selector use karke pehla matching element return karega (id, class, tag sab accept karta hai).

Q42. Event bubbling aur capturing kya hota hai?

Event bubbling: event sabse pehle inner element se trigger hota hai aur parent tak bubble karta hai.

Event capturing: event outer element se start hota hai aur inner element tak jata hai.

Example:

```
document.getElementById("child").addEventListener("click", ()=>{
  console.log("Child clicked");
}, false); // bubbling (default)
```

```
document.getElementById("parent").addEventListener("click", ()=>{
  console.log("Parent clicked");
}, true); // capturing
```

Q43. Event delegation kya hota hai?

Jab hum ek parent element par listener lagate hain aur uske child ke events ko handle karte hain → ise delegation kehte hain.

Ye performance improve karta hai (har child par alag listener lagane ki need nahi).

Example:

```
document.getElementById("list").addEventListener("click", (e)=>{
  if(e.target.tagName === "LI"){
    console.log("Clicked on " + e.target.innerText);
  }
});
```

Q44. Difference between innerText vs innerHTML vs textContent?

`innerText`: sirf visible text return karega (style aur CSS respect karega).

`innerHTML`: HTML tags + text dono deta hai.

`textContent`: raw text deta hai (hidden content bhi include hota hai).

Q45. classList methods kya hain?

`element.classList.add("class")` → class add karega.

`element.classList.remove("class")` → class remove karega.

`element.classList.toggle("class")` → class ko on/off karega.

Example:

```
let box = document.getElementById("box");
box.classList.add("active");
box.classList.toggle("hidden");
```

6. ES6+ Features

Q46. Template literals kya hain?

Backticks (``) use hote hain, jisme variable interpolation (${}) aur multiline string likhne ki facility hoti hai.

Q47. let, const aur var ka difference?

`var`: function-scoped, hoisting karta hai, re-declare allowed.

`let`: block-scoped, re-declare nahi kar sakte.

`const`: block-scoped, constant value (re-assign nahi kar sakte).

Example:

```
let a = 10;
const b = 20;
var c = 30;
```

Q48. Destructuring kya hota hai?

Array ya object ke values ko easily extract karne ke liye use hota hai.

Example:

```
let [x,y] = [10,20];
let {name, age} = {name:"Aman", age:22};
```

Q49. Default parameters kya hote hain?

Function ke arguments ke liye default value set kar sakte ho.

Example:

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

Q50. Promise ka basic syntax kya hota hai?

Promise ek asynchronous operation ko handle karne ka tarika hai.

Example:

```
let promise = new Promise((resolve,reject)=>{
  let success = true;
  if(success) resolve("Success");
  else reject("Failed");
});

promise.then(res=>console.log(res))
.catch(err=>console.log(err));
```

7. Asynchronous JS

Q51. JavaScript single-threaded language hone ka matlab kya hai?

Matlab JS ek hi thread pe execute hoti hai (ek hi call stack hota hai).

Ek time pe ek hi task run hota hai, lekin async code ko handle karne ke liye **Event Loop + Callback Queue** use karta hai.

Q52. Callback function kya hota hai?

Ek function jo dusre function ko argument ke taur pe diya jata hai aur baad me execute hota hai.

Example:

```
function greet(name, callback){
  console.log("Hello " + name);
  callback();
}
greet("Aman", ()=> console.log("Welcome!"));
```

Q53. Callback Hell kya hota hai?

Jab multiple nested callbacks ek ke andar ek likhne padte hain → code unreadable aur messy ho jata hai.

Is problem ko solve karne ke liye Promises aur async/await aaya.

Q54. Promise kya hota hai aur states kaunse hote hain?

Promise ek object hai jo future value represent karta hai.

States:

- **Pending** (initial state)
- **Fulfilled** (resolve hua)
- **Rejected** (error aaya)

Example:

```
let p = new Promise((resolve, reject)=>{
  setTimeout(()=> resolve("Done!"), 1000);
});
p.then(res=>console.log(res));
```

Q55. Promise chaining kya hoti hai?

Multiple `.then()` ek ke baad ek chalana → isse sequential async tasks kar sakte hain.

Example:

```
fetch("https://jsonplaceholder.typicode.com/posts/1")
  .then(res => res.json())
  .then(data => console.log("Post:", data))
  .catch(err => console.error(err));
```

Q56. Difference between Promise.all, Promise.race, Promise.allSettled?

Promise.all: saare promises resolve hone ka wait karega, agar ek fail hua to sab reject ho jate hain.

Promise.race: jo promise pehle settle hoga (resolve/reject), wahi return karega.

Promise.allSettled: saare promises ka result (resolve/reject) deta hai, bina fail huye.

Q57. async/await ka use kya hai?

`async` function always promise return karta hai.

`await` ek promise ka result wait karta hai bina nested `.then()` likhe.

Example:

```
async function getData(){
  try {
```

```
    let res = await
fetch("https://jsonplaceholder.typicode.com/posts/1");
    let data = await res.json();
    console.log(data);
  } catch(err){
    console.log("Error:", err);
  }
}
getData();
```

Q58. Fetch API kya hoti hai?

Fetch ek modern API hai jo server se data lane ya bhejne ke liye use hoti hai.

Ye promise-based hai (XMLHttpRequest ka modern replacement).

Example:

```
fetch("https://jsonplaceholder.typicode.com/users")
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Q59. Error handling async code mei kaise karte hain?

Promises mei `.catch()` use hota hai.

Async/await mei `try...catch` block use hota hai.

Example:

```
async function fetchData(){
  try{
    let res = await fetch("wrong-url");
    let data = await res.json();
    console.log(data);
```

```
  }catch(err){
    console.log("Error:", err.message);
  }
}
fetchData();
```

Q60. setTimeout aur setInterval mei kya difference hai?

`setTimeout`: ek function ko ek specific delay ke baad **sirf ek baar** execute karta hai.

`setInterval`: ek function ko fixed interval ke saath **bar-bar repeat** karta hai.

Example:

```
setTimeout(()=> console.log("Runs once"), 2000);

setInterval(()=> console.log("Runs repeatedly"), 1000);
```

8. OOP in JavaScript

Q61. JavaScript object banane ke 3 tareeqe kaunse hain?

1. Object literal `{}`
2. `new Object()` constructor
3. Constructor function / Class se

Q62. Prototype-based inheritance kya hai?

JavaScript mei har object ka ek hidden property `[[Prototype]]` hota hai jo parent object se link hota hai.

Agar ek property current object mei na mile, JS uske prototype chain mei search karta hai.

Example:

```
let animal = { eats: true };
let rabbit = Object.create(animal);
console.log(rabbit.eats); // true (inherited)
```

Q63. Constructor function kya hota hai?

Normal function jisme `this` ka use hota hai aur new object banane ke liye `new` keyword use hota hai.

Example:

```
function Person(name, age){
  this.name = name;
  this.age = age;
}
let p1 = new Person("Aman", 22);
console.log(p1.name); // Aman
```

Q64. ES6 Classes kya hain?

Classes ek syntactic sugar hain jo constructor functions aur prototype inheritance ko easy banati hain.

Example:

```
class Car {
  constructor(name){
    this.name = name;
  }
  drive(){
    console.log(this.name + " is driving");
  }
}
let c1 = new Car("Tesla");
```

```
c1.drive();
```

Q65. Inheritance class ke through kaise hoti hai?

`extends` keyword se ek class dusri class ki properties/methods inherit kar sakti hai.

Example:

```
class Animal {
  speak(){ console.log("Animal speaks"); }
}
class Dog extends Animal {
  speak(){ console.log("Dog barks"); }
}
let d = new Dog();
d.speak(); // Dog barks
```

Q66. Prototype aur `__proto__` mei difference kya hai?

prototype: ek object jo function ke saath attach hota hai (sirf functions ke liye).

proto: ek reference jo har object mei hota hai jo uske prototype ko point karta hai.

Q67. Encapsulation JS mei kaise achieve hoti hai?

Data ko private/protected banakar access ko control karna.

JS mei ye achieve hota hai closures aur `#private` fields se.

Example:

```
class Account {
  #balance = 0;
  deposit(amount){ this.#balance += amount; }
  getBalance(){ return this.#balance; }
```

```
}
let acc = new Account();
acc.deposit(100);
console.log(acc.getBalance()); // 100
```

Q68. Polymorphism ka matlab JS mei kya hai?

Ek hi function/method alag-alag objects ke liye different behavior dikhaata hai.

Example: method overriding inheritance ke case mei.

Q69. Abstraction JS mei kaise hoti hai?

Sirf essential details expose karna aur implementation hide karna.

JS mei abstraction classes, modules, closures ke through hota hai.

Example:

```
function createCounter(){
  let count = 0;
  return {
    increment(){ count++; return count; },
    decrement(){ count--; return count; }
  }
}
let counter = createCounter();
console.log(counter.increment()); // 1
console.log(counter.decrement()); // 0
```

Q70. Difference between Object.create() aur new keyword?

`Object.create(proto)` ek object banata hai jo directly specified prototype ko link karta hai.

new keyword constructor function run karke ek object banata hai aur usse us function ke prototype se link karta hai.

9. Advanced JS Concepts

Q71. Closure kya hota hai?

Closure ek function + uska lexical scope hota hai.

Matlab ek inner function outer function ke variables ko access kar sakta hai even after outer function execute ho chuka ho.

Q72. Closure ka ek real example do.

Example:

```
function outer(){
  let count = 0;
  return function inner(){
    count++;
    return count;
  }
}
let counter = outer();
console.log(counter()); // 1
console.log(counter()); // 2
```

👉 Yahan `inner()` function ne `count` ko yaad rakha (closure).

Q73. Scope aur Lexical Environment mei difference?

Scope: kis variable ko kahan access kar sakte hain (global, function, block).

Lexical Environment: ek structure jisme function ke variables aur uska parent reference hota hai.

Q74. Hoisting kya hai?

Hoisting ka matlab hai JS compile time pe variable aur function declarations ko top pe le aati hai.

`var` hoist hota hai lekin `let` aur `const` hoist hote hain par unko temporal dead zone ki wajah se access nahi kar sakte.

Example:

```
console.log(a); // undefined
var a = 10;
```

Q75. Execution Context kya hota hai?

Execution Context ek environment hai jisme JS code execute hota hai.

2 types:

1. Global Execution Context
2. Function Execution Context

Q76. Call Stack ka role kya hai?

Call Stack ek LIFO (Last In First Out) structure hai jisme function execution context push/pop hote hain.

Jab ek function call hota hai to stack mei push hota hai aur jab complete hota hai to pop hota hai.

Q77. "this" keyword normal function aur arrow function mei kya difference hai?

Normal function mei `this` runtime pe decide hota hai (jis object se call ho raha hai).

Arrow function mei `this` lexical hota hai (parent scope se inherit karta hai).

Example:

```
let obj = {
  name: "Aman",
  normal: function(){ console.log(this.name); },
  arrow: () => { console.log(this.name); }
}
obj.normal(); // Aman
obj.arrow();  // undefined (parent scope -> global)
```

Q78. Bind, Call aur Apply mei difference?

call – function ko turant call karta hai aur arguments alag pass hote hain.

apply – function ko turant call karta hai aur arguments array ke form mei pass hote hain.

bind – ek new function return karta hai jisme `this` permanently set hota hai.

Example:

```
function greet(city){
  console.log(this.name + " from " + city);
}
let user = { name: "Aman" };
```

```
greet.call(user, "Karachi");    // Aman from Karachi
greet.apply(user, ["Lahore"]);  // Aman from Lahore
let bound = greet.bind(user);
bound("Islamabad");             // Aman from Islamabad
```

Q79. Higher Order Function (HOF) kya hai?

HOF wo function hai jo ya to ek function ko argument ke tor pe leta hai ya ek function return karta hai.

Example: `map`, `filter`, `reduce`.

Q80. Currying JS mei kya hota hai?

Currying ek technique hai jisme ek function multiple arguments lene ke bajaye ek ek argument lene wale chain of functions return karta hai.

Example:

```
function add(a){
  return function(b){
    return function(c){
      return a + b + c;
    }
  }
}
console.log(add(2)(3)(4)); // 9
```

10. Browser & Web APIs

Q81. LocalStorage aur SessionStorage mei kya difference hai?

Dono key-value storage provide karte hain browser mei.

- **LocalStorage** → data permanently store hota hai (jab tak manually delete na karo).
- **SessionStorage** → data sirf ek session tak hota hai, tab close hone ke baad remove ho jata hai.

Q82. LocalStorage ka ek example do.

Example:

```
// Save data
localStorage.setItem("user", "Aman");

// Get data
console.log(localStorage.getItem("user")); // Aman

// Remove data
localStorage.removeItem("user");
```

Q83. Cookies kya hoti hain aur LocalStorage se kaise different hain?

Cookies small pieces of data hote hain jo browser mei store hote hain aur har request ke sath server ko bheje jaate hain.

Difference:

- Cookies → max 4KB, mostly server communication ke liye.
- LocalStorage → ~5MB, client-side storage ke liye.

Q84. Cookie ka ek example do.

Example:

```
document.cookie = "username=Aman; expires=Fri, 30 Aug 2025 12:00:00
UTC; path=/";
console.log(document.cookie);
```

Q85. Geolocation API kya hai?

Browser ka API jo user ki location (latitude & longitude) provide karta hai (agar user allow kare).

Q86. Geolocation API ka example do.

Example:

```
navigator.geolocation.getCurrentPosition(
  (pos) => console.log(pos.coords.latitude, pos.coords.longitude),
  (err) => console.log(err)
);
```

Q87. History API ka use kya hai?

History API se hum browser ke navigation ko control kar sakte hain (forward, back, pushState).

Example: `history.back()`, `history.forward()`, `history.pushState()`

Q88. History API ka example do.

Example:

```
// Move to previous page
history.back();

// Move forward
history.forward();

// Push new state
```

```
history.pushState({id: 1}, "new page", "/new-url");
```

Q89. Location object ka use kya hai?

`window.location` ek object hai jo current URL ki info deta hai aur usko change karne ki capability deta hai.

Properties: `href`, `hostname`, `pathname`, `protocol`.

Q90. Location object ka example do.

Example:

```
console.log(location.href);      // full URL
console.log(location.hostname);  // domain
console.log(location.pathname);  // path
console.log(location.protocol);  // http or https

// Redirect
location.href = "https://google.com";
```

11. Error Handling

Q91. try / catch / finally ka use kya hai?

Error handling ke liye JavaScript mei `try...catch` use hota hai.

- `try` → jis code mei error aa sakta hai.
- `catch` → error handle karega.
- `finally` → hamesha chalega (error aaye ya na aaye).

Q92. try/catch ka example do.

Example:

```
try {
  let x = y + 5; // y defined nahi hai → error
} catch (err) {
  console.log("Error: " + err.message);
} finally {
  console.log("Finally block always runs");
}
```

Q93. Custom error kaise throw karte hain?

JavaScript mei `throw` keyword use karke apna custom error throw kar sakte ho.

Q94. Custom error ka example do.

Example:

```
function checkAge(age) {
  if (age < 18) {
    throw new Error("Underage not allowed");
  }
  return "Allowed";
}

try {
  console.log(checkAge(16));
} catch (e) {
  console.log(e.message);
}
```

12. Event Loop & Concurrency

Q95. Event Loop kya hai?

JavaScript single-threaded hai. Event Loop ek mechanism hai jo Call Stack aur Callback Queue ko manage karta hai taake async code properly execute ho.

Q96. Microtask aur Macrotask mei difference kya hai?

- **Macrotask**: setTimeout, setInterval.
- **Microtask**: Promises, queueMicrotask.

Microtask hamesha pehle run hota hai.

Example:

```
console.log("Start");

setTimeout(()=>console.log("Macrotask"), 0);

Promise.resolve().then(()=>console.log("Microtask"));

console.log("End");
// Output: Start → End → Microtask → Macrotask
```

13. Memory Management

Q97. Garbage Collection kya hai?

JavaScript automatic memory management karta hai.

Jab koi object referenceable nahi hota toh garbage collector usse memory se free kar deta hai.

Q98. Memory leaks kaise hote hain JS mei?

Jab memory release nahi hoti due to unwanted references.

Examples:

- Global variables
- Forgotten timers / intervals
- Event listeners na remove karna

14. Functional Programming

Q99. Currying kya hota hai?

Currying ek technique hai jisme ek function multiple arguments lene ke bajaye ek-ek karke function return karta hai.

Example:

```
function sum(a) {
  return function(b) {
    return function(c) {
      return a + b + c;
    }
  }
}

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

Q100. Function composition kya hai?

Function composition ka matlab hai multiple functions ko combine karke ek naya function banana.

Example:

```
const add5 = x => x + 5;
const double = x => x * 2;

const compose = (f, g) => x => f(g(x));

const newFunc = compose(add5, double);

console.log(newFunc(10)); // (10*2)+5 = 25
```

14. Functional Programming

Q101. Functional Programming (FP) kya hai?

FP ek programming paradigm hai jo functions ko "first-class citizens" manta hai.

- Pure functions use hote hain.
- Data immutable hota hai.
- Side effects avoid kiye jate hain.
- Higher-order functions ka use hota hai.

Q102. Pure Function kya hota hai?

Aisa function jo:

1. Same input → hamesha same output deta hai.
2. Koi side effect nahi karta (jaise global variable modify karna, API call, DOM change).

Example:

```
function add(a, b) {
  return a + b; // Pure function
}

let x = 10;
```

```
function impure(y) {
  return x + y; // Impure (depends on external variable)
}
```

Q103. Pure functions ke benefits kya hain?

- Test karna easy hota hai.
- Predictable output.
- Debugging easy hoti hai.
- Concurrency ke liye safe hote hain.

Q104. Immutability kya hai?

Immutability ka matlab hai data ko modify na karna, balki naye data structures create karna.

Example:

```
let arr = [1, 2, 3];

// Immutable approach
let newArr = [...arr, 4];
console.log(arr);    // [1,2,3]
console.log(newArr); // [1,2,3,4]
```

Q105. Mutable vs Immutable objects mei difference?

- **Mutable**: directly change ho jata hai (example: `arr.push(4)`).
- **Immutable**: ek naya object ya array banate ho (example: `[...arr, 4]`).

Q106. Currying kya hota hai aur use kyu karte hain?

Currying ek function ko multiple arguments lene se tod kar ek-ek function return karne ka tareeqa hai.

Useful jab ek function reusable banaana ho partial arguments ke saath.

Example:

```
function multiply(a) {
  return function(b) {
    return function(c) {
      return a * b * c;
    }
  }
}

console.log(multiply(2)(3)(4)); // 24
```

Q107. Currying ka real-life use case?

Example: configuration ya customization.

```
function greet(greeting) {
  return function(name) {
    return `${greeting}, ${name}`;
  }
}

const sayHello = greet("Hello");
console.log(sayHello("Aman")); // Hello, Aman
```

Q108. Function Composition kya hai?

Function composition ka matlab hai multiple functions ko combine karke ek function banana jo sequentially execute ho.

Example:

```
const double = x => x * 2;
const square = x => x * x;

const compose = (f, g) => x => f(g(x));

const result = compose(square, double)(5);
console.log(result); // square(double(5)) = 100
```

Q109. Composition aur Chaining mei difference?

- **Composition**: Functions ko ek naya function banane ke liye combine karna.
- **Chaining**: Ek object par multiple methods ko sequentially call karna (example: `arr.map().filter().reduce()`).

Q110. Higher-order functions FP mei kyu important hote hain?

Higher-order function woh hota hai jo ek function ko **argument** ke taur par leta hai ya ek **function return** karta hai.

FP mei yeh powerful hai kyunki reusability aur modularity badh jaati hai.

Example:

```
function higherOrder(fn, value) {
  return fn(value);
}

console.log(higherOrder(x => x * 2, 5)); // 10
```