Q1. What is JavaScript?
JavaScript is a high-level, interpreted programming language used to make web pages interactive. It runs on browsers and supports both client-side and server-side development (Node.js).

Q2. What are the different data types in JavaScript?

- **Primitive**: String, Number, Boolean, Undefined, Null, Symbol, BigInt.
- **Non-Primitive**: Objects, Arrays, Functions.

Q3. Explain the difference between var, let, and const.

- **var** → Function-scoped, hoisted, can be redeclared.
- **let** → Block-scoped, not hoisted, can be reassigned.
- **const** → Block-scoped, must be initialized, cannot be reassigned.

Q4. What are the advantages and disadvantages of using JavaScript?

- **Advantages**: Fast execution, easy to learn, supported by all browsers, used for both frontend & backend.
- **Disadvantages**: Can be misused for malicious code, loosely typed (errors may go unnoticed), browser compatibility issues.

Q5. What is hoisting in JavaScript?
Hoisting means variables and function declarations are moved to the top of their scope during execution. Example:

```
console.log(a); // undefined

var a = 10;
```

Q6. What is the this keyword in JavaScript?
this refers to the object that is currently executing the function. Its value depends on how the function is called (object method, event handler, global, arrow function).

Q7. What are JavaScript data types?
JavaScript supports **7 primitive types** (String, Number, Boolean, Undefined, Null, Symbol, BigInt) and **reference types** like Objects, Arrays, Functions.

Q8. Explain == vs ===.

- == → Loose equality, checks values after type conversion.
- === → Strict equality, checks both value and type.

Example:

```
"5" == 5   // true

"5" === 5  // false
```

Q9. What is a closure?
A closure is when an inner function remembers and accesses variables from its outer function, even after the outer function has finished execution.

Q10. Explain the difference between function declarations and function expressions.

- **Function Declaration** → Hoisted, can be used before definition.
- **Function Expression** → Not hoisted, stored in a variable.

```
function add(){ }     // declaration

const sub = function(){ }  // expression
```

Q11. What are arrow functions?
Arrow functions are a shorter syntax for writing functions. They don't have their own this and are useful in callbacks.

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

Q12. What is the this keyword in JavaScript?
Same as Q6 → this refers to the current execution context (global object, function caller, or object instance).

Q13. Explain the concept of scope in JavaScript.
Scope defines where variables are accessible:

- **Global Scope**: Available everywhere.
- **Function Scope**: Variables inside functions.
- **Block Scope**: Variables declared with let or const inside {}.

Q14. What are the different ways to iterate over an array in JavaScript?

- for loop
- for...of
- forEach()
- map()
- while loop

Q15. Explain the concept of prototypal inheritance.
Objects in JavaScript can inherit properties and methods from another object via the **prototype chain**. Every object has a hidden [[Prototype]] that links to another object.

Q16. What is the difference between null and undefined?

- **null** → Intentional absence of value.
- **undefined** → Variable declared but not assigned.

Q17. How do you create an object in JavaScript?

```
const obj1 = {name: "John"};      // Object literal

const obj2 = new Object();        // Object constructor

function Person(name){ this.name = name; } // Constructor function
```

Q18. How do you clone an object in JavaScript?

- Using spread:

```
let clone = {...obj};
```

- Using Object.assign():

```
let clone = Object.assign({}, obj);
```

Q19. What is an event loop in JavaScript?
The event loop allows JavaScript to handle **asynchronous tasks**. It constantly checks the call stack and callback queue to execute pending tasks.

Q20. Explain the difference between synchronous and asynchronous code.

- **Synchronous** → Code runs line by line, blocking execution.
- **Asynchronous** → Code can run in background (setTimeout, fetch, promises).

Q21. What are Promises in JavaScript?
A Promise represents a value that may be available now, later, or never. States: **pending → fulfilled → rejected**.

Q22. How do you handle errors in Promises?
Using .catch() or inside async/await with try...catch.

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

Q23. What is async/await in JavaScript?
async/await is syntax sugar over Promises, making asynchronous code look synchronous.

```
async function getData(){

  try { let res = await fetch(url); } catch(err){ console.log(err); }

}
```

Q24. What is the DOM?
The **Document Object Model** is a tree structure that represents HTML elements. JavaScript can manipulate it (add, remove, edit).

Q25. How do you select an element by ID in JavaScript?

```
document.getElementById("myId");
```

Q26. How do you select elements by class name in JavaScript?

document.getElementsByClassName("myClass");

Q27. How do you select elements using a CSS selector in JavaScript?

document.querySelector(".myClass"); // first match

document.querySelectorAll("p"); // all matches

Q28. How do you create an element in JavaScript?
You can use document.createElement().

let div = document.createElement("div");

Q29. How do you add an element to the DOM?
You can use methods like appendChild() or append().

document.body.appendChild(div);

Q30. How do you remove an element from the DOM?
Use removeChild() or remove().

element.remove();

Q31. How do you change the content of an HTML element?

- innerHTML → Sets HTML content.
- textContent → Sets plain text.

document.getElementById("demo").innerHTML = "Hello!";

Q32. How do you change the style of an HTML element?
Use the style property.

element.style.color = "red";

element.style.fontSize = "20px";

Q33. What is event delegation?
A technique where you attach a single event listener to a parent element to handle events of child elements. Improves performance.

Q34. What is a higher-order function in JavaScript?
A function that either takes another function as an argument or returns a function. Example: map, filter, reduce.

Q35. Explain the concept of callback functions.
A callback is a function passed as an argument to another function, executed after the first

function completes.

Q36. What is the bind method in JavaScript?
bind() creates a new function with a fixed this value and arguments.

let bound = func.bind(obj);

Q37. What is the difference between call and apply?

- call() → Calls a function with arguments separated by commas.
- apply() → Calls a function with arguments as an array.

Q38. What is memoization?
Memoization is caching function results so repeated calls with the same arguments return the cached result instead of recomputing.

Q39. What is the event loop and how does it work?
The event loop manages asynchronous operations in JavaScript. It keeps checking the **call stack** and moves tasks from the **callback queue/microtask queue** when the stack is empty.

Q40. What are modules in JavaScript?
Modules are reusable pieces of code exported from one file and imported into another. ES6 introduced import and export.

Q41. What is the difference between ES5 and ES6?

- ES5: var, function constructors, callbacks.
- ES6: let, const, arrow functions, classes, promises, modules.

Q42. What are promises and why are they used?
Promises represent eventual completion or failure of an async operation. They prevent callback hell and make code easier to manage.

Q43. What is async/await and how does it work?
async/await is built on top of promises. async defines an asynchronous function, and await pauses execution until a promise resolves.

Q44. What are design patterns and why are they important?
Design patterns are proven solutions to common coding problems. They improve code reusability, maintainability, and scalability.

Q45. What is the Module pattern?
The Module pattern organizes code into self-contained units using closures. It helps encapsulate private data and exposes only required functionality.

Q46. What is the Observer pattern?
In Observer pattern, an object (subject) maintains a list of dependents (observers) and notifies them automatically of state changes.

Q47. What is the Factory pattern?
Factory pattern provides a way to create objects without specifying the exact class. Instead, a factory method decides which object to create.

Q48. What is the difference between deep copy and shallow copy?

- **Shallow copy** → Copies only the first level, nested objects share references.
- **Deep copy** → Creates a completely independent clone of all nested objects.

Q49. What are web workers and why are they used?
Web workers allow JavaScript to run background tasks without blocking the main thread, improving performance.

Q50. What is CORS and how does it work?
CORS (Cross-Origin Resource Sharing) allows web apps to access resources from another domain securely by setting HTTP headers like Access-Control-Allow-Origin.

Q51. What is the difference between synchronous and asynchronous code execution?

- **Synchronous** → Executes line by line, blocking further execution.
- **Asynchronous** → Non-blocking, allows other tasks while waiting (e.g., setTimeout, fetch).

Q52. How can you create an immediately invoked function expression (IIFE)?

```
(function(){

  console.log("IIFE runs immediately");

})();
```

Q53. Explain event delegation in JavaScript.
Event delegation is attaching one event listener to a parent element instead of multiple child elements, taking advantage of event bubbling.

Q54. What is the difference between null and undefined?

- **null** → Empty or intentional absence of value.
- **undefined** → Variable declared but not initialized.

Q55. What are arrow functions, and how do they differ from regular functions?
Arrow functions are shorter in syntax and do not have their own this, arguments, or prototype. Useful in callbacks.

Q56. How do you create a promise in JavaScript?

```
let promise = new Promise((resolve, reject) => {

  setTimeout(() => resolve("Done!"), 1000);
```

});

Q57. What is the purpose of the async and await keywords?
They simplify working with promises:

- async makes a function return a promise.
- await pauses execution until a promise resolves.

Q58. What is a higher-order function?
A function that accepts other functions as arguments or returns a function. Example: map, filter.

Q59. Explain the concept of hoisting in JavaScript.
Hoisting is when variable and function declarations are moved to the top of their scope before execution.

console.log(a); // undefined

var a = 5;

Q60. What is the difference between slice() and splice() methods in JavaScript?

- slice(start, end) → Returns a shallow copy of a portion of an array, does **not modify** the original.
- splice(start, deleteCount, items...) → Adds/removes elements **in place** and modifies the original array.

Q61. How can you prevent an object from being modified?
Use Object.freeze(obj). It prevents adding, deleting, or changing properties.

Q62. What is the prototype property in JavaScript?
Every function in JavaScript has a prototype object, which is used for inheritance. Objects created from constructors inherit methods from the constructor's prototype.

Q63. How does the bind() method work?
bind() creates a new function with a fixed this value and optional preset arguments. It does not execute immediately.

Q64. What is the difference between map() and forEach()?

- map() → Returns a new array after applying a function.
- forEach() → Iterates over elements but **does not return** a new array.

Q65. Explain the concept of promises chaining.
Promise chaining means handling multiple asynchronous operations sequentially by returning promises inside .then().

Q66. How do you handle errors in promises?
Use .catch() after .then(), or use try...catch inside async/await.

Q67. What is the purpose of the typeof operator?
typeof checks the data type of a variable. Example:

typeof "hello" // "string"

Q68. How can you check if a variable is an array in JavaScript?
Use Array.isArray(variable).

Q69. What is the arguments object in JavaScript?
It's an array-like object available inside functions that contains all passed arguments.

Q70. Explain the concept of function currying.
Currying transforms a function with multiple arguments into a series of functions, each taking one argument.

function add(a){ return b => a + b; }

Q71. What is the difference between synchronous and asynchronous code?

- **Synchronous** → Executes line by line, blocking.
- **Asynchronous** → Does not block, allows parallel execution with callbacks, promises, or async/await.

Q72. How do you handle asynchronous code in JavaScript?
Using callbacks, promises (.then/.catch), or async/await.

Q73. What is a Symbol in JavaScript?
Symbol is a primitive data type introduced in ES6, used to create unique identifiers for object properties.

Q74. Explain the concept of debouncing in JavaScript.
Debouncing delays execution of a function until a certain time has passed since the last call (useful in search boxes).

Q75. What is the event loop in JavaScript?
The event loop manages asynchronous operations by moving tasks from the callback/microtask queue to the call stack when it is empty.

Q76. How does JavaScript handle asynchronous operations?
Through the **event loop**, using APIs like setTimeout, Promises, fetch, and async/await.

Q77. What is the spread operator in JavaScript?
... is used to expand arrays/objects into individual elements.

let arr = [1,2,3];

let copy = [...arr];

Q78. Explain the concept of rest parameters.
Rest parameters (...args) allow functions to accept an indefinite number of arguments as an array.

Q79. How do you create a deep copy of an object?
Options:

- structuredClone(obj) (modern).
- JSON.parse(JSON.stringify(obj)).
- Recursion or libraries like Lodash.

Q80. What is the Proxy object in JavaScript?
Proxy allows you to intercept and redefine fundamental operations (like get, set) on objects.

Q81. Explain the difference between call() and apply().

- call(thisArg, arg1, arg2, ...) → Pass arguments individually.
- apply(thisArg, [args]) → Pass arguments as an array.

Q82. What is event bubbling and event capturing?

- **Bubbling**: Event starts from the target element and moves up to ancestors.
- **Capturing**: Event starts from ancestors and goes down to the target element.

Q83. How can you stop event propagation?
Use event.stopPropagation().

Q84. What is the purpose of the defaultPrevented property?
It checks whether event.preventDefault() was called on the event (e.g., preventing form submission).

Q85. How do you create a custom event in JavaScript?

```
let event = new CustomEvent("myEvent", { detail: { msg: "Hello" }});

element.dispatchEvent(event);
```

Q86. What are service workers?
Scripts that run in the background of a browser, enabling features like offline caching, push notifications, and background sync.

Q87. How do you detect if cookies are enabled in the browser?

```
navigator.cookieEnabled // true or false
```

Q88. Explain the fetch API.
fetch() is a modern API for making network requests, returning a promise.

fetch("url").then(res => res.json());

Q89. What is the difference between localStorage and sessionStorage?

- **localStorage** → Data persists even after the browser closes.
- **sessionStorage** → Data persists only until the browser/tab is closed.

Q90. How do you set a timeout in JavaScript?
Use setTimeout().

setTimeout(() => console.log("Hi"), 2000);

Q91. What is the reduce() method in JavaScript?
reduce() applies a function to an accumulator and each element, reducing the array to a single value.

let sum = [1,2,3].reduce((acc, val) => acc + val, 0);

Q92. How do you check if a variable is a function?

typeof variable === "function"

Q93. What is the instanceof operator?
Checks if an object is an instance of a constructor.

obj instanceof Constructor

Q94. How do you compare two objects in JavaScript?
Direct == or === only compares references. For deep comparison, check keys/values recursively or use JSON.stringify(obj1) === JSON.stringify(obj2) (not perfect but works in many cases).

Q95. What is a WeakMap?
A WeakMap stores key-value pairs where keys are **objects only**, and references are weak (eligible for garbage collection).

Q96. How can you convert an array-like object to an array?

Array.from(arguments)

// OR

[...arguments]

✅ **Find max number from array list:**

let arr = [5, 12, 8, 130, 44];

let max = Math.max(...arr); // 130

Q97. Explain the Promise.all() method.

Takes an array of promises → returns a single promise resolved when **all** are fulfilled, or rejected if any fail.

Q98. How can you debounce a function in JavaScript?

```
function debounce(fn, delay){
  let timer;
  return function(...args){
    clearTimeout(timer);
    timer = setTimeout(() => fn.apply(this, args), delay);
  };
}
```

Q99. Difference between forEach and throttle + Unique elements task

- forEach: Iterates through array, cannot break/return new array.
- **Throttle:** Ensures a function executes at most once every given interval.

```
function throttle(fn, delay){
  let last = 0;
  return (...args) => {
    let now = Date.now();
    if(now - last >= delay){
      fn.apply(this, args);
      last = now;
    }
  };
}
```

✅ **Find unique elements and arrange:**

```
let arr = [1,2,2,3,4,4,5];
let unique = [...new Set(arr)];
```

// [1,2,3,4,5]

Q100. What is the Reflect object in JavaScript?
Reflect is a built-in object that provides methods for object manipulation (like Reflect.get, Reflect.set) similar to Object methods, but with better error handling.

Q101. Spread operator to copy an array

let arr = [1, 2, 3];

let copy = [...arr];

✅ **Find max number:**

let max = Math.max(...arr);

Q102. What is a tagged template literal?
A special form of template literal where a function processes the template string parts and substitutions.

Q103. Explain the concept of the temporal dead zone.
Variables declared with let/const cannot be accessed before their declaration, even though they are hoisted. That region is the TDZ.

Q104. Difference between Object.seal() and Object.freeze()

- Object.seal() → Prevents adding/removing properties, but existing values can be changed.
- Object.freeze() → Prevents adding, removing, or modifying properties.

Q105. How can you shallow copy an object?

let copy = Object.assign({}, obj);

let copy2 = {...obj};

Q106. Purpose of Promise.race()
Returns a promise that resolves/rejects as soon as the **first promise settles** (success or failure).

Q107. Convert string to number in JavaScript

Number("123")

parseInt("123")

+"123"

Q108. Difference between call() and apply()?

- call(thisArg, arg1, arg2, ...) → args separately.

- apply(thisArg, [args]) → args in an array.

Q109. Remove duplicates + Merge two sorted arrays (LeetCode-style)

✅ Remove duplicates:

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

let unique = [...new Set(arr)];
```

✅ Merge two sorted arrays (nums1, nums2):

```
function merge(nums1, m, nums2, n){

 let result = nums1.slice(0, m).concat(nums2.slice(0, n));

 return result.sort((a,b) => a - b);

}
```

Q110. Difference between window.onload and document.ready

- window.onload → Fires after page + all resources (images, scripts) are fully loaded.
- document.ready (DOMContentLoaded) → Fires as soon as DOM is loaded (faster).

Q111. What are JavaScript generators?
Functions declared with function* that can pause (yield) and resume execution. Used for lazy evaluation, async flow control.

Q112. How do you handle deep comparisons in JavaScript?
Use recursive function to compare keys & values deeply, or libraries like Lodash's _.isEqual().

Q113. What is a template literal?
Strings wrapped in backticks (`) allowing interpolation and multi-line strings.

```
let name = "Subham";

`Hello, ${name}!`
```

Q114. How do you create a multi-line string in JavaScript?

```
let str = `This is

a multi-line

string.`;
```

Q115. What is an ES6 module?
A JavaScript file that uses export/import syntax to share code across files.

Q116. What is a WeakSet?

A WeakSet stores objects only (no primitives). References are weak, so objects can be garbage collected if not referenced elsewhere.

Q117. Explain the concept of memoization.

Memoization is an optimization technique where function results are cached so repeated calls with the same input return instantly.

```
function memoizedAdd() {
  let cache = {};
  return function(x, y) {
    let key = `${x},${y}`;
    if(cache[key]) return cache[key];
    cache[key] = x + y;
    return cache[key];
  }
}
let add = memoizedAdd();
console.log(add(2,3)); // 5
console.log(add(2,3)); // Cached: 5
```

Q118. How do you detect the user's browser in JavaScript?

```
console.log(navigator.userAgent);
```

It returns details like Chrome, Firefox, Safari, etc.

Q119. What is the purpose of requestAnimationFrame?

requestAnimationFrame() schedules animations to run before the next repaint, making them smoother and more efficient than setInterval.

Q120. How do you implement inheritance in JavaScript?

Using class and extends:

```
class Animal {
  speak(){ console.log("Animal sound"); }
}
```

```
class Dog extends Animal {
  speak(){ console.log("Bark!"); }
}
new Dog().speak();
```

Q121. What is a Proxy in JavaScript?

Proxy lets you intercept and redefine fundamental operations (like get, set) on objects.

```
let target = { name: "Subham" };
let proxy = new Proxy(target, {
  get: (obj, prop) => prop in obj ? obj[prop] : "Not Found"
});
console.log(proxy.name); // Subham
console.log(proxy.age);  // Not Found
```

Q122. How do you debounce a function in JavaScript?

```
function debounce(fn, delay) {
  let timer;
  return function(...args) {
    clearTimeout(timer);
    timer = setTimeout(() => fn.apply(this, args), delay);
  }
}
```

Q123. Explain the concept of currying.

Currying transforms a function with multiple arguments into a sequence of functions each taking one argument.

```
function curryAdd(a) {
  return function(b) {
    return function(c) {
      return a + b + c;
```

```
  }
 }
}
console.log(curryAdd(2)(3)(4)); // 9
```

Q124. Difference between bind and call?

- bind: Returns a new function with bound this. Does not invoke immediately.
- call: Immediately invokes the function with specified this and arguments.

Q125. How can you create a custom event in JavaScript?

```
let event = new CustomEvent("greet", { detail: { name: "Subham" }});
document.addEventListener("greet", e => console.log("Hello", e.detail.name));
document.dispatchEvent(event);
```

Q126. Difference between undefined and not defined?

- **undefined**: Declared but no value assigned.
- **not defined**: Variable does not exist at all.

Q127. How do you create a private variable in JavaScript?

Using closures or # private fields in classes:

```
class Person {
 #secret = "hidden";
 reveal() { return this.#secret; }
}
console.log(new Person().reveal()); // hidden
```

Q128. Purpose of Object.create()?

Creates a new object with the specified prototype.

```
let proto = { greet(){ console.log("Hello"); } };
let obj = Object.create(proto);
obj.greet();
```

Q129. JS function to add two numbers:

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

Q130. JS function to check even or odd:

```
function checkEvenOdd(num){
  return num % 2 === 0 ? "Even" : "Odd";
}
```

Q131. JS function to print numbers 1–10:

```
function printNumbers(){
  for(let i=1;i<=10;i++) console.log(i);
}
```

Q132. JS function to find largest number in an array:

```
function findLargest(arr){
  return Math.max(...arr);
}
```

Q133. Validate form input (non-empty before submit):

```
<input type="text" id="name">
<button onclick="validate()">Submit</button>
<script>
function validate(){
  let val = document.getElementById("name").value;
  if(val.trim() === ""){
    alert("Field cannot be empty");
    return false;
  }
  alert("Submitted!");
}
</script>
```

Q134. Handle errors using try/catch:

```
try {
  let result = riskyFunction();
} catch (err) {
  console.error("Error occurred:", err.message);
}
```