Javascript Interview Questions

Q1. What is JavaScript?

Answer: JavaScript is a programming language primarily used to create interactive web pages and web applications. It can run in the browser or on the server (Node.js).
Example:

```
console.log("Hello, JavaScript!");
```

Real-Time Usage: Adding dynamic content like sliders, modals, and interactive forms on web pages.

Q2. What are the features of JavaScript?

Answer: Key features include:

- Client-side scripting language
- Interpreted and dynamically typed
- Supports object-oriented programming
- Rich built-in functions and libraries

Example:

```
let name = "Mounika"; // dynamically typed
console.log(`Hello, ${name}`);
```

Real-Time Usage: Dynamic web pages, user input validation, animations.

Q3. What are the different data types in JavaScript?

Answer:

- **Primitive:** string, number, boolean, null, undefined, symbol
- **Complex:** object

Example:

```
let name = "Alice"; // string
let age = 25;       // number
let isStudent = true; // boolean
```

```
let obj = { city: "NY" }; // object
```

Real-Time Usage: Storing and manipulating user information, configuration data, and API responses.

Q4. What is the difference between let, var, and const?

Answer:

- **var:** Function-scoped, can be redeclared & reassigned
- **let:** Block-scoped, can be reassigned but not redeclared
- **const:** Block-scoped, cannot be redeclared or reassigned

Example:

```
var x = 10;
let y = 20;
const z = 30;
```

Real-Time Usage: Use `const` for constants, `let` for variables that change, `var` is rarely used in modern JS.

Q5. What is the difference between == and === in JavaScript?

Answer:

- == checks equality after type conversion (loose equality)
- === checks equality without type conversion (strict equality)

Example:

```
console.log(5 == "5");  // true
console.log(5 === "5"); // false
```

Real-Time Usage: Always use === to avoid unexpected type coercion.

Q6. What is an immediately-invoked function expression (IIFE)?

Answer: A function executed immediately after it's defined, creating a new scope.

Example:

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

Real-Time Usage: Prevent polluting global namespace in modular scripts.

Q7. What is a closure in JavaScript?

Answer: A closure is a combination of a function and its lexical environment, allowing access to outer variables even after the outer function has executed.

Example:

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

Real-Time Usage: Creating private variables, function factories, and module patterns.

Q8. What is the purpose of the this keyword in JavaScript?

Answer: Refers to the object executing the current code. Value depends on function invocation.

Example:

```
const obj = {
  name: "Alice",
  greet() {
    console.log(this.name);
  }
};
obj.greet(); // Alice
```

Real-Time Usage: Managing object methods and context in classes, event handlers.

Q9. What is event delegation in JavaScript?

Answer: Attaching a single event listener to a parent element to handle events from child elements.

Example:

```
document.getElementById("parent").addEventListener("click", (e) => {
  if(e.target && e.target.nodeName == "BUTTON") {
    console.log("Button clicked:", e.target.textContent);
  }
});
```

Real-Time Usage: Efficiently managing dynamic elements like lists or tables.

Q10. Explain prototypal inheritance in JavaScript

Answer: Objects inherit properties and methods from another object (prototype). JS searches the prototype chain if property is not found.

Example:

```
const parent = { greet: () => console.log("Hello!") };
const child = Object.create(parent);
child.greet(); // Hello!
```

Real-Time Usage: Reusing methods across objects, reducing memory usage.

Q11. What is the difference between null and undefined?

Answer:

- `null` represents the intentional absence of any value.
- `undefined` means a variable has been declared but not initialized.

Example:

```
let a;           // undefined
let b = null;    // null
console.log(a);  // undefined
console.log(b);  // null
```

Real-Time Usage: Use `null` to explicitly clear a variable; `undefined` usually signals missing initialization.

Q12. What is hoisting in JavaScript?

Answer: Variable and function declarations are moved to the top of their scope during compilation. Only declarations are hoisted, not initializations.

Example:

```
console.log(x); // undefined
var x = 5;

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

Real-Time Usage: Helps understand variable accessibility and function calls before declaration.

Q13. What are the different ways to handle asynchronous operations in JavaScript?

Answer:

- **Callback functions**
- **Promises**
- **Async/await**

Example (Promise):

```
fetch("https://api.example.com/data")
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Real-Time Usage: Fetching data from APIs, timers, or server requests.

Q14. What is a callback function?

Answer: A function passed as an argument to another function, executed after an event or task completes.

Example:

```
function greet(name, callback) {
  console.log("Hello, " + name);
```

```
  callback();
}
greet("Alice", () => console.log("Callback executed!"));
```

Real-Time Usage: Handling asynchronous operations like event listeners or API calls.

Q15. What are promises in JavaScript?

Answer: Objects representing eventual completion or failure of an asynchronous operation. They support chaining with `.then()` and `.catch()`.

Example:

```
let promise = new Promise((resolve, reject) => {
  let success = true;
  success ? resolve("Done!") : reject("Failed!");
});
promise.then(msg => console.log(msg)).catch(err => console.error(err));
```

Real-Time Usage: Managing async flows cleanly without callback hell.

Q16. What is the purpose of the async keyword in JavaScript?

Answer: Defines an asynchronous function that returns a Promise. Allows `await` inside to pause execution until a Promise resolves.

Example:

```
async function fetchData() {
  let response = await fetch("https://api.example.com/data");
  let data = await response.json();
  console.log(data);
}
fetchData();
```

Real-Time Usage: Simplifies API requests, timers, and sequential async tasks.

Q17. What is the difference between a deep copy and a shallow copy?

Answer:

- **Shallow copy:** Copies object references, nested objects remain shared.

- **Deep copy:** Recursively copies all properties, creating an independent clone.

Example:

```
let obj = {a:1, b:{c:2}};
let shallow = {...obj}; // shallow copy
let deep = JSON.parse(JSON.stringify(obj)); // deep copy
```

Real-Time Usage: Avoid unintentional mutations in nested objects.

Q18. What is event bubbling in JavaScript?

Answer: Event triggered on an element propagates upward to its parent elements until reaching the root.

Example:

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

Real-Time Usage: Managing nested DOM events efficiently.

Q19. What is event capturing in JavaScript?

Answer: Event is first captured by the outermost element and propagates inward (top-down).

Example:

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

Real-Time Usage: Useful when you need to intercept events before they reach inner elements.

Q20. What is the purpose of the bind() method in JavaScript?

Answer: Creates a new function with a specified `this` value and optional initial arguments.

Example:

```
const person = { name: "Alice" };
function greet(greeting) {
  console.log(`${greeting}, ${this.name}`);
}
const greetAlice = greet.bind(person, "Hello");
greetAlice(); // Hello, Alice
```

Real-Time Usage: Ensures correct `this` context in callbacks or event handlers.

Q21. What is the difference between call() and apply() methods?

Answer: Both invoke a function with a specified `this` value.

- **call():** Arguments are passed individually
- **apply():** Arguments are passed as an array

Example:

```
function greet(greeting, punctuation) {
  console.log(`${greeting}, ${this.name}${punctuation}`);
}
const person = { name: "Alice" };
greet.call(person, "Hello", "!"); // Hello, Alice!
greet.apply(person, ["Hi", "!!"]); // Hi, Alice!!
```

Real-Time Usage: Borrow methods from other objects or arrays.

Q22. How do you compare two objects for equality in JavaScript?

Answer: Objects are equal only if they reference the same object. For deep equality, manually compare properties.

Example:

```
const obj1 = { a: 1 };
const obj2 = { a: 1 };
console.log(obj1 === obj2); // false
```

Real-Time Usage: Useful when checking state changes in frontend frameworks like React.

Q23. What is the purpose of the reduce() method in JavaScript?

Answer: Reduces an array to a single value by applying a reducer function.

Example:

```
const numbers = [1, 2, 3, 4];
const sum = numbers.reduce((acc, curr) => acc + curr, 0);
console.log(sum); // 10
```

Real-Time Usage: Calculating totals, averages, or merging arrays.

Q24. How do you clone an object in JavaScript?

Answer: Use `Object.assign()`, spread operator `...`, or `JSON.parse(JSON.stringify())`.

Example:

```
const obj = {a:1, b:2};
const clone1 = {...obj};
const clone2 = Object.assign({}, obj);
```

Real-Time Usage: Avoid mutating the original object when updating state in React.

Q25. Explain the concept of event loop in JavaScript.

Answer: The event loop handles asynchronous callbacks, ensuring JS executes code non-blockingly.

Example:

```
console.log("Start");
setTimeout(() => console.log("Timeout"), 0);
console.log("End");
// Output: Start, End, Timeout
```

Real-Time Usage: Understanding async operations, timers, and UI responsiveness.

Q26. What are arrow functions in JavaScript?

Answer: Concise function syntax that lexically binds `this`.

Example:

```
const add = (a, b) => a + b;
console.log(add(2,3)); // 5
```

Real-Time Usage: Shorter syntax for callbacks, array methods, and functional programming.

Q27. What is the purpose of the map() method in JavaScript?

Answer: Creates a new array by transforming each element of the original array.

Example:

```
const numbers = [1,2,3];
const doubled = numbers.map(n => n * 2);
console.log(doubled); // [2,4,6]
```

Real-Time Usage: Transforming API data before rendering UI components.

Q28. What is the purpose of the filter() method in JavaScript?

Answer: Creates a new array containing elements that pass a condition.

Example:

```
const numbers = [1,2,3,4];
const even = numbers.filter(n => n % 2 === 0);
console.log(even); // [2,4]
```

Real-Time Usage: Filtering user lists, search results, or datasets dynamically.

Q29. What is the purpose of the forEach() method in JavaScript?

Answer: Executes a function on each array element. Does not return a new array.

Example:

```
const numbers = [1,2,3];
numbers.forEach(n => console.log(n * 2));
// Output: 2,4,6
```

Real-Time Usage: Iterating over arrays to perform side effects like DOM updates.

Q30. What is the difference between null and undefined?

Answer:

- **null:** intentional absence of value
- **undefined:** declared but uninitialized variable

Example:

```
let a; // undefined
let b = null;
console.log(a, b); // undefined null
```

Real-Time Usage: Clear distinction when checking API responses or state variables.

Q31. What is the difference between function declarations and function expressions?

Answer:

- **Function declaration:** Hoisted, can be called before declaration
- **Function expression:** Not hoisted, must be defined before use

Example:

```
greet(); // Works
function greet(){ console.log("Hi"); }

const greet2 = function(){ console.log("Hello"); }
greet2(); // Works only after definition
```

Real-Time Usage: Choosing expressions for callbacks and declarations for reusable functions.

Q32. How do you handle errors in JavaScript?

Answer: Use `try...catch` to catch and handle exceptions.

Example:

```
try {
  let result = riskyOperation();
} catch(err) {
  console.error("Error caught:", err);
}
```

Real-Time Usage: Prevent app crashes from unexpected inputs or API errors.

Q33. What is the purpose of the setTimeout() function?

Answer: Executes a function after a specified delay.

Example:

```
setTimeout(() => console.log("Executed after 2 seconds"), 2000);
```

Real-Time Usage: Creating delays, animations, or polling APIs.

Q34. What is a JavaScript event?

Answer: An action or occurrence that happens in the browser, like clicks or key presses.

Example:

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

Real-Time Usage: Interactive UI elements like buttons, forms, or modals.

Q35. What is the purpose of the addEventListener() method?

Answer: Attaches an event handler to an element.

Example:

```
document.getElementById("btn").addEventListener("mouseover", () =>
console.log("Hovered"));
```

Real-Time Usage: Handling user interactions without inline HTML events.

Q36. What is the difference between event.preventDefault() and event.stopPropagation()?

Answer:

- `preventDefault()`: Stops default behavior (e.g., form submit)
- `stopPropagation()`: Stops event from bubbling or capturing

Example:

```
document.getElementById("link").addEventListener("click", (e) =>
e.preventDefault());
```

Real-Time Usage: Form validation, custom links, nested event handling.

Q37. What are the different types of error in JavaScript?

Answer:

- `Error` (base type)
- `SyntaxError`
- `TypeError`
- `ReferenceError`
- `RangeError`
- `EvalError`

Example:

```
let a = undefinedVar; // ReferenceError
```

Real-Time Usage: Debugging and handling exceptions effectively.

Q38. What is the purpose of the Object.keys() method?

Answer: Returns an array of enumerable property names of an object.

Example:

```
const obj = { a:1, b:2 };
console.log(Object.keys(obj)); // ["a","b"]
```

Real-Time Usage: Iterating object properties dynamically.

Q39. What is the purpose of the Object.values() method?

Answer: Returns an array of enumerable property values of an object.

Example:

```
console.log(Object.values({ a:1, b:2 })); // [1,2]
```

Real-Time Usage: Extracting values for UI display or calculations.

Q40. What is the purpose of the Object.entries() method?

Answer: Returns an array of `[key, value]` pairs from an object.

Example:

```
console.log(Object.entries({a:1, b:2})); // [["a",1], ["b",2]]
```

Real-Time Usage: Converting objects for iteration or transforming to maps.

Q41. What is the purpose of the isNaN() function?

Answer: Checks if a value is `NaN` (Not-a-Number). Returns `true` if NaN, otherwise `false`.

Example:

```
console.log(isNaN(123)); // false
console.log(isNaN("hello")); // true
```

Real-Time Usage: Validate numeric inputs in forms or calculations.

Q42. What is event delegation in JavaScript?

Answer: Attach a single event listener to a parent element to handle events from its child elements.

Example:

```
document.getElementById("parent").addEventListener("click", (e) => {
  if(e.target && e.target.tagName === "BUTTON") console.log("Button
clicked");
});
```

Real-Time Usage: Improve performance for lists or dynamically generated elements.

Q43. What is the purpose of the trim() method in JavaScript?

Answer: Removes whitespace from the beginning and end of a string.

Example:

```
const str = "   Hello World   ";
console.log(str.trim()); // "Hello World"
```

Real-Time Usage: Cleaning user input before form submission.

Q44. What is the difference between a function declaration and a function expression?

Answer:

- **Declaration:** Hoisted, can be called before definition
- **Expression:** Not hoisted, assigned to variable

Example:

```
function greet(){ console.log("Hi"); } // declaration
const greet2 = function(){ console.log("Hello"); } // expression
```

Real-Time Usage: Choose based on scoping needs and callbacks.

Q45. What is the difference between == and === in JavaScript?

Answer:

- ==: Loose equality, converts types before comparison
- ===: Strict equality, compares value and type

Example:

```
console.log(5 == "5");  // true
console.log(5 === "5"); // false
```

Real-Time Usage: Avoid bugs when comparing values of different types.

Q46. What are the different types of loops in JavaScript?

Answer:

- `for` loop
- `while` loop
- `do...while` loop
- `for...in` loop (object properties)
- `for...of` loop (iterables like arrays, strings)

Example:

```
for(let i=0; i<3; i++) console.log(i);
```

Real-Time Usage: Iterating arrays, objects, or custom conditions.

Q47. What is the purpose of the continue statement in JavaScript?

Answer: Skips the current iteration of a loop and continues with the next iteration.

Example:

```
for(let i=1; i<=5; i++){
  if(i === 3) continue;
  console.log(i); // 1,2,4,5
}
```

Real-Time Usage: Skip certain items during loops without breaking the entire loop.

Q48. What is the purpose of the break statement in JavaScript?

Answer: Terminates the execution of a loop or switch statement immediately.

Example:

```
for(let i=1; i<=5; i++){
  if(i === 3) break;
  console.log(i); // 1,2
}
```

Real-Time Usage: Stop iteration once a condition is met to save computation.

Q49. What is the purpose of the try...catch statement in JavaScript?

Answer: Catches and handles exceptions in code.

Example:

```
try {
  throw new Error("Something went wrong");
} catch(err) {
  console.log(err.message);
}
```

Real-Time Usage: Handle unexpected errors in API calls or DOM operations gracefully.

Q50. What is the purpose of the finally block in a try...catch statement?

Answer: Executes code regardless of whether an exception was thrown or caught.

Example:

```
try {
  console.log("Try block");
} catch(err) {
  console.log("Catch block");
} finally {
  console.log("Finally block"); // Always executes
}
```

Real-Time Usage: Closing resources or cleanup tasks in asynchronous code.

Q51. What is a generator function in JavaScript?

Answer: A function that can be paused and resumed using the `yield` keyword.

Example:

```
function* gen() {
  yield 1;
  yield 2;
  yield 3;
}
const g = gen();
console.log(g.next().value); // 1
console.log(g.next().value); // 2
```

Real-Time Usage: Lazy evaluation or handling large data sequences efficiently.

Q52. What is the purpose of the yield keyword in a generator function?

Answer: Pauses execution and yields a value to the caller.

Example:

```
function* genNumbers() {
  yield 10;
  yield 20;
}
const g = genNumbers();
console.log(g.next().value); // 10
console.log(g.next().value); // 20
```

Real-Time Usage: Iterating through large datasets or streaming data.

Q53. What is the difference between a spread operator (...) and the Object.assign() method?

Answer:

- `...` spreads elements/properties into a new array/object (shallow copy)
- `Object.assign()` copies enumerable properties from one or more source objects to a target object

Example:

```
const obj1 = {a:1};
const obj2 = {...obj1}; // Spread operator
const obj3 = Object.assign({}, obj1); // Object.assign
```

Real-Time Usage: Copying objects/arrays without mutating originals.

Q54. What is the purpose of the fetch() function in JavaScript?

Answer: Makes asynchronous HTTP requests and returns a Promise.

Example:

```
fetch("https://api.example.com/data")
```

```
  .then(res => res.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

Real-Time Usage: Calling APIs in web applications.

Q55. What is the purpose of the localStorage object in JavaScript?

Answer: Stores key-value pairs in the browser that persist after the browser is closed.

Example:

```
localStorage.setItem("name", "Alice");
console.log(localStorage.getItem("name")); // Alice
```

Real-Time Usage: Storing user preferences or session data.

Q56. What is the purpose of the sessionStorage object in JavaScript?

Answer: Stores key-value pairs for the duration of a session. Cleared when the browser window is closed.

Example:

```
sessionStorage.setItem("tempData", "123");
console.log(sessionStorage.getItem("tempData")); // 123
```

Real-Time Usage: Temporary data storage during user sessions.

Q57. What is a closure in JavaScript?

Answer: Function with access to variables from its outer scope even after the outer function has finished.

Example:

```
function outer() {
  let count = 0;
  return function() {
    count++;
    return count;
  };
```

```
}
const counter = outer();
console.log(counter()); // 1
console.log(counter()); // 2
```

Real-Time Usage: Private variables, data encapsulation, and function factories.

Q58. What is the purpose of the this keyword in JavaScript?

Answer: Refers to the object that is currently executing the code.

Example:

```
const person = { name: "Alice", greet() { console.log(this.name); } };
person.greet(); // Alice
```

Real-Time Usage: Referring to the current object context in methods or event handlers.

Q59. What are modules in JavaScript?

Answer: Reusable pieces of code that encapsulate functionality and expose only what's needed.

Example:

```
// math.js
export function add(a, b){ return a + b; }

// main.js
import { add } from './math.js';
console.log(add(2,3)); // 5
```

Real-Time Usage: Organizing code for large-scale applications.

Q60. What is the purpose of the export keyword in JavaScript modules?

Answer: Exports variables, functions, or objects from a module for use in other modules.

Example:

```
export const name = "Alice";
export function greet(){ console.log("Hello"); }
```

Real-Time Usage: Sharing code between files or components.

Q61. What is the purpose of the import keyword in JavaScript modules?

Answer: Imports variables, functions, or objects from other modules so you can use them in the current file.

Example:

```
import { add } from './math.js';
console.log(add(5, 3)); // 8
```

Real-Time Usage: Using shared code across multiple files or projects.

Q62. What is the difference between null and undefined?

Answer:

- `null`: intentional absence of any value.
- `undefined`: variable is declared but not assigned.

Example:

```
let a;
console.log(a); // undefined
let b = null;
console.log(b); // null
```

Real-Time Usage: Checking uninitialized variables vs intentionally empty values.

Q63. What is the difference between a shallow copy and a deep copy?

Answer:

- **Shallow copy:** Copies references; nested objects are shared.
- **Deep copy:** Recursively copies all properties, creating independent objects.

Example:

```
const obj = {a:1, b:{c:2}};
const shallow = {...obj};
```

```
const deep = JSON.parse(JSON.stringify(obj));
```

Real-Time Usage: Avoid modifying original objects when copying complex data.

Q64. What is the purpose of the async and await keywords in JavaScript?

Answer:

- `async`: Declares an asynchronous function that returns a Promise.
- `await`: Pauses function execution until the Promise resolves.

Example:

```
async function fetchData(){
  const res = await fetch("https://api.example.com");
  const data = await res.json();
  console.log(data);
}
fetchData();
```

Real-Time Usage: Writing readable asynchronous code without nested callbacks.

Q65. What are the different ways to handle asynchronous operations in JavaScript?

Answer:

- **Callbacks:** Functions called after async operations.
- **Promises:** Represent eventual completion/failure of async operations.
- **Async/await:** Cleaner syntax using Promises.

Example (Promise):

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

Real-Time Usage: Handling API requests, timers, or event-driven code.

Q66. What is the purpose of the Promise object in JavaScript?

Answer: Represents eventual completion/failure of async operations, enabling chaining.

Example:

```
const promise = new Promise((resolve, reject) => {
  setTimeout(() => resolve("Done"), 1000);
});
promise.then(res => console.log(res)); // Done
```

Real-Time Usage: Manage async workflows like API calls or file reads.

Q67. What are the states of a Promise in JavaScript?

Answer:

1. **Pending:** Initial state, not fulfilled/rejected.
2. **Fulfilled:** Operation completed successfully.
3. **Rejected:** Operation failed with an error.

Example:

```
const p = new Promise((res, rej) => res("Success"));
console.log(p); // Pending -> Fulfilled
```

Real-Time Usage: Track async operation outcomes for error handling or flow control.

Q68. How do you handle errors in asynchronous operations using promises?

Answer: Using the `.catch()` method on a Promise.

Example:

```
fetch("https://api.invalid.com")
  .then(res => res.json())
  .catch(err => console.error("Error:", err));
```

Real-Time Usage: Prevent unhandled errors during API calls or async tasks.

Q69. What is the difference between the window object and the document object in JavaScript?

Answer:

- **window:** Represents the browser window; global object for JS runtime.
- **document:** Represents the HTML document loaded in the window.

Example:

```
console.log(window.innerWidth); // Width of browser
console.log(document.title); // Page title
```

Real-Time Usage: Access DOM elements and window properties.

Q70. What is event bubbling in JavaScript?

Answer: Event propagates from the target element up to parent elements.

Example:

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

Clicking child triggers both: `"Child clicked"`, `"Parent clicked"`

Real-Time Usage: Use parent handlers to manage multiple child events efficiently.

Q71. What is event capturing in JavaScript?

Answer: Event propagates from the outermost element down to the target element.

Example:

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

Real-Time Usage: Capture events before they reach child elements for validation or interception.

Q72. What is the difference between event bubbling and event capturing?

Answer:

- **Bubbling:** Target → parent → root (bottom-up).
- **Capturing:** Root → parent → target (top-down).

Real-Time Usage: Choose propagation phase based on event handling strategy.

Q73. What is the purpose of the event.preventDefault() method?

Answer: Prevents default browser behavior of an event.

Example:

```
document.querySelector("form").addEventListener("submit", e =>
e.preventDefault());
```

Real-Time Usage: Prevent page reloads, default link clicks, or other browser actions.

Q74. What is event delegation in JavaScript?

Answer: A single listener handles events for multiple child elements.

Example:

```
document.getElementById("list").addEventListener("click", e => {
  if(e.target.tagName === "LI") console.log("Item clicked:",
e.target.textContent);
});
```

Real-Time Usage: Reduce memory usage when dealing with dynamic elements.

Q75. What is the purpose of the this keyword in JavaScript?

Answer: Refers to the object executing the current function.

Example:

```
const obj = {name:"Alice", greet(){ console.log(this.name); }};
obj.greet(); // Alice
```

Real-Time Usage: Access object properties in methods or event callbacks.

Q76. What are closures in JavaScript?

Answer: Functions with access to outer function variables even after the outer function has executed.

Example:

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

Real-Time Usage: Data privacy, state management in functions, or functional programming.

Q77. What is hoisting in JavaScript?

Answer: Variable and function declarations are moved to the top of their scope during compilation.

Example:

```
console.log(a); // undefined
var a = 5;
```

Real-Time Usage: Understand variable behavior and avoid bugs with early access.

Q78. What is the purpose of the typeof operator in JavaScript?

Answer: Returns a string indicating the type of a variable or value.

Example:

```
console.log(typeof 123); // number
console.log(typeof "Hello"); // string
```

Real-Time Usage: Type checking before performing operations.

Q79. What is a callback function in JavaScript?

Answer: A function passed as an argument to another function and executed later.

Example:

```
function greet(name, callback){ callback(name); }
greet("Alice", (n) => console.log("Hello " + n)); // Hello Alice
```

Real-Time Usage: Handle async tasks or events like API responses or DOM actions.

Q80. What is the difference between a named function and an anonymous function?

Answer:

- **Named:** Has a name; can reference itself.
- **Anonymous:** No name; often used as a callback or assigned to a variable.

Example:

```
function namedFunc(){ console.log("Hi"); }
const anonFunc = function(){ console.log("Hello"); };
```

Real-Time Usage: Use named for recursion/debugging, anonymous for inline callbacks.

Perfect! Here's the final batch of **JavaScript Interview Questions (Q81–Q100)** with **answers, examples, and real-time usage**:

Q81. What is the purpose of the bind() method in JavaScript?

Answer: Creates a new function with a specific `this` value and optionally preset arguments.

Example:

```
const obj = {name: "Alice"};
function greet(greeting){ console.log(greeting + " " + this.name); }
const boundGreet = greet.bind(obj);
boundGreet("Hello"); // Hello Alice
```

Real-Time Usage: Ensure correct `this` context when passing methods as callbacks.

Q82. What is the purpose of the call() method in JavaScript?

Answer: Invokes a function with a specified `this` value and arguments individually.

Example:

```
function greet(age){ console.log(`${this.name} is ${age}`); }
const person = {name: "Bob"};
greet.call(person, 25); // Bob is 25
```

Real-Time Usage: Borrow methods from other objects dynamically.

Q83. What is the purpose of the apply() method in JavaScript?

Answer: Invokes a function with a specified `this` value and arguments as an array.

Example:

```
greet.apply(person, [30]); // Bob is 30
```

Real-Time Usage: Similar to `call()`, but useful when arguments are in arrays.

Q84. What is the purpose of the map() method in JavaScript?

Answer: Transforms each element of an array and returns a new array.

Example:

```
const nums = [1,2,3];
const squares = nums.map(x => x*x);
console.log(squares); // [1,4,9]
```

Real-Time Usage: Transform data sets for UI rendering or calculations.

Q85. What is the purpose of the filter() method in JavaScript?

Answer: Returns a new array containing elements that pass a condition.

Example:

```
const nums = [1,2,3,4];
const evens = nums.filter(x => x%2===0);
console.log(evens); // [2,4]
```

Real-Time Usage: Filter data arrays based on conditions (e.g., search results).

Q86. What is the purpose of the reduce() method in JavaScript?

Answer: Reduces an array to a single value using an accumulator function.

Example:

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

Real-Time Usage: Aggregate data such as sums, averages, or concatenations.

Q87. What is the purpose of the forEach() method in JavaScript?

Answer: Executes a function for each array element without returning a new array.

Example:

```
nums.forEach(num => console.log(num*2)); // 2 4 6 8
```

Real-Time Usage: Iterate arrays for side-effects like logging or DOM updates.

Q88. What are arrow functions in JavaScript?

Answer: Concise syntax for functions; automatically binds `this` lexically.

Example:

```
const add = (a,b) => a+b;
console.log(add(2,3)); // 5
```

Real-Time Usage: Shorter, cleaner syntax for callbacks and functional programming.

Q89. What is destructuring assignment in JavaScript?

Answer: Extracts values from arrays or objects into variables.

Example:

```
const [x, y] = [1, 2];
const {name, age} = {name:"Alice", age:25};
```

Real-Time Usage: Simplifies accessing data from API responses or arrays.

Q90. What are template literals in JavaScript?

Answer: Strings with embedded expressions using backticks (`) and ${}`.

Example:

```
const name = "Alice";
console.log(`Hello ${name}`); // Hello Alice
```

Real-Time Usage: Dynamically build strings, HTML templates, or log messages.

Q91. What are JavaScript promises?

Answer: Objects representing future completion or failure of asynchronous operations.

Example:

```
const p = new Promise((res, rej) => res("Done"));
p.then(val => console.log(val)); // Done
```

Real-Time Usage: Handle async operations like API calls, file reading, or timers.

Q92. What are the states of a JavaScript promise?

Answer:

1. Pending
2. Fulfilled
3. Rejected

Example:

```
const p = new Promise((res, rej) => setTimeout(() => res("Success"), 1000));
```

Real-Time Usage: Track async task completion for error handling and chaining.

Q93. What are JavaScript proxies?

Answer: Objects that intercept fundamental operations (get, set, function calls).

Example:

```
const handler = { get: (obj, prop) => prop in obj ? obj[prop] : 0 };
const proxy = new Proxy({a:10}, handler);
console.log(proxy.a); // 10
console.log(proxy.b); // 0
```

Real-Time Usage: Validate, log, or modify object behavior dynamically.

Q94. What is the Event Loop in JavaScript?

Answer: Mechanism that handles async operations, ensuring single-threaded execution.

Example:

```
console.log("Start");
setTimeout(()=>console.log("Timeout"),0);
console.log("End");
// Output: Start End Timeout
```

Real-Time Usage: Understand non-blocking operations and async behavior.

Q95. What is the purpose of localStorage in JavaScript?

Answer: Stores key-value pairs in the browser that persist across sessions.

Example:

```
localStorage.setItem("name","Alice");
console.log(localStorage.getItem("name")); // Alice
```

Real-Time Usage: Save user preferences, settings, or offline data.

Q96. What is the purpose of sessionStorage in JavaScript?

Answer: Stores key-value pairs for a single session; cleared when the tab closes.

Example:

```
sessionStorage.setItem("token","abc123");
```

Real-Time Usage: Session-specific data like authentication tokens.

Q97. What are JavaScript modules?

Answer: Reusable pieces of code with private scope, exposing only what's necessary.

Example:

```
// math.js
export function add(a,b){ return a+b; }
// main.js
import { add } from './math.js';
```

Real-Time Usage: Code organization and reusability in large projects.

Q98. What is a generator function in JavaScript?

Answer: Functions that can pause and resume execution using `yield`.

Example:

```
function* gen(){ yield 1; yield 2; }
const g = gen();
console.log(g.next().value); // 1
console.log(g.next().value); // 2
```

Real-Time Usage: Lazy data processing, iterators, or asynchronous flows.

Q99. What is the purpose of the yield keyword in a generator function?

Answer: Pauses generator execution and returns a value.

Example:

```
function* counter(){ let i=0; while(i<3) yield i++; }
for(let val of counter()) console.log(val); // 0 1 2
```

Real-Time Usage: Control flow in sequences and async iterables.

Q100. How do you clone an object in JavaScript?

Answer: Create a copy of an object using spread, Object.assign, or deep copy methods.

Example:

```
const obj = {a:1, b:{c:2}};
const shallow = {...obj};
const deep = JSON.parse(JSON.stringify(obj));
```

Real-Time Usage: Avoid modifying original objects when working with nested data.