JavaScript Interview Questions and Answers

---

1. JavaScript Basics

Q1: What is JavaScript? A: High-level, interpreted language for interactive web pages.

Q2: Data types in JS - Primitive: string, number, boolean, null, undefined, symbol, bigint - Non-primitive: object (arrays, functions, dates)

Q3: Difference between var, let, const - var: function-scoped, redeclarable, hoisted - let: block-scoped, not redeclarable, hoisted but not initialized - const: block-scoped, not redeclarable or reassignable

Q4: What is NaN and how to check it? - NaN = Not a Number - Check with isNaN(value) or Number.isNaN(value)

Q5: Difference between == and === - == : value check with type coercion - === : value and type check

---

2. Functions

Q6: Ways to declare functions

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

Q7: Callback function

```
function greet(name, cb){ console.log('Hello '+name); cb(); }
greet('Adnan', ()=>console.log('Callback executed'));
```

Q8: IIFE

```
(function(){ console.log('IIFE'); })();
```

---

3. Objects and Classes

Q9: What is an object? - Collection of key-value pairs

```
const person={name:'Adnan', age:22};
```

Q10: this keyword - Refers to execution context - Object method: this = object - Global: this = window (browser)

Q11: Prototypes - Shared properties/methods for all instances

```
function Person(name){this.name=name;}
Person.prototype.greet=function(){console.log('Hello '+this.name);};
```

Q12: Class vs Constructor function - Class = syntactic sugar over constructor function - Cleaner syntax, easier inheritance

Q13: Inheritance

```
class Human{constructor(name){this.name=name;}}
class Employee extends Human{constructor(name,job){super(name);this.job=job;}}
```

---

4. Scope and Closure

Q14: Scope - Determines variable accessibility - Types: global, function, block

Q15: Closure - Function remembers creation scope

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

---

5. Asynchronous JS

Q16: Event loop - Handles async operations in single-threaded JS

Q17: Sync vs Async - Sync: line by line, blocking - Async: non-blocking, runs in background

Q18: Promises

```
let p=new Promise((resolve,reject)=>setTimeout(()=>resolve('Done'),1000));
p.then(console.log);
```

Q19: Async/Await vs Promises - Async/Await = syntactic sugar over Promises

---

6. Arrays and Methods

Q20: Array methods - forEach, map, filter, reduce, find, some, every, slice, splice, push, pop, shift, unshift

Q21: map vs forEach - map returns new array, forEach returns undefined

Q22: slice vs splice - slice = new array, splice = modifies original

---

7. DOM and Events

Q23: Selecting DOM elements

```
document.getElementById('id');
document.querySelector('.class');
document.querySelectorAll('div');
```

Q24: addEventListener vs onclick - addEventListener: multiple handlers - onclick: one handler only

Q25: Event bubbling vs capturing - Bubbling: child → parent - Capturing: parent → child

---

8. Error Handling

Q26: Try/Catch

```
try{ riskyOperation(); } catch(e){ console.log(e); } finally{
console.log('Always'); }
```

Q27: throw vs return - throw: stops execution, sends error - return: stops function, returns value

---

9. Advanced Concepts

Q28: Hoisting - JS moves declarations to top

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

Q29: null vs undefined - undefined: declared but not assigned - null: intentional absence

Q30: Strict mode - 'use strict'; prevents silent errors

Q31: Modules - Export and import variables/functions between files

Q32: call, apply, bind - call: invokes with this, args separately - apply: args as array - bind: returns new function with bound this

---

10. Misc / Practical

Q33: Deep clone object

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

Q34: Shallow vs Deep copy - Shallow: first level only - Deep: nested objects copied

Q35: Event delegation - Attach listener to parent to handle child events

Q36: let in loops - let = block scope, avoids closure problems

Q37: Debounce vs Throttle - Debounce: execute after event stops - Throttle: limit execution rate

Q38: Practical tips - Explain with code - Be ready to write code on whiteboard - Draw mental diagrams for tricky topics

---

This document can be expanded with **~100 questions**, covering ES6+, closures, async patterns, DOM manipulation, web APIs, event loop, memory management, and real interview coding questions.