VARIABLES – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1. Declare a variable `name` and assign your full name. Print it.
2. Declare a constant `PI = 3.14` and try to reassign it. Observe what happens.
3. Use `let` and `var` to declare variables in a block and check scope outside the block.
4. Swap two numbers stored in variables using a temporary variable.
5. Create two variables `x` and `y` and swap their values without a temporary variable.
6. Declare multiple variables in one line and assign values to them.
7. Print the type of a variable using `typeof`.
8. Initialize a variable without value, print it, then assign a value and print again.
9. Declare variables for first name and last name, then combine them using template literals.
10. Demonstrate global vs local variable by declaring inside a function.
11. Increment and decrement a numeric variable and print results.
12. Declare a string variable and concatenate it with a number. Observe type coercion.
13. Declare a boolean variable and toggle its value.
14. Use `var` inside a function and try accessing it outside the function.
15. Declare a variable inside an `if` block using `let` and try to access it outside the block.

Intermediate Level (16–35)

Focus: closures, shadowing, block scope, dynamic variables, references.

16. Create a function that counts how many times it has been called using a closure.
17. Create a counter object with `increment`, `decrement`, and `reset` methods using closures.
18. Demonstrate variable shadowing in nested functions.
19. Write a function that returns a multiplier function (higher-order function).
20. Create a module exposing private variables with getters/setters.
21. Write a function that can only be called once; subsequent calls return `undefined`.
22. Demonstrate the difference between `let` and `var` in loops with `setTimeout`.
23. Swap two object properties without using a temporary variable.
24. Merge two objects without overwriting existing keys.
25. Track changes to a variable using closures and print history.
26. Generate unique IDs using a private counter variable.
27. Implement a function that accepts variable names as strings and initializes them dynamically.
28. Write a function that automatically converts numeric strings to numbers in parameters.
29. Demonstrate temporal dead zone (TDZ) using `let` and `const`.
30. Track how many times a property of an object has been accessed.
31. Write a function that logs the value of `this` in different contexts.
32. Auto-increment a variable every second using closures.
33. Count undefined variables in an object.
34. Swap two nested object properties safely.
35. Deep copy a variable safely without using JSON methods.

3️⃣ Advanced / Higher Level (36–50)

Focus: advanced closures, chainable APIs, dynamic behavior, memoization, interview-level logic.

36. Create a function that remembers multiple previous results (memoization).
37. Implement a `once` wrapper function that ensures a function runs only once.
38. Implement a function that toggles a boolean variable but limits the number of toggles.
39. Track historical states of variables and restore any previous state.
40. Freeze an object variable but allow arithmetic operations without modifying the original.
41. Create a function that returns another function to calculate powers dynamically.
42. Implement a chainable API to modify variables step by step (`obj.setX().incrementY().resetZ()`).
43. Create a function that counts the call stack depth dynamically.
44. Track active variables in memory (simulate memory usage counter).
45. Generate a random password stored in a private variable accessible only via a getter.
46. Use closures to track and log variable changes across multiple functions.
47. Implement a function that merges multiple counters into a single object using closures.
48. Demonstrate closure capturing wrong loop values with `var` and fix using `let`.
49. Dynamically generate getters and setters for an object using a function.
50. Create a function that validates variable types dynamically and throws errors for invalid types.

DATA TYPES – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: identifying, converting, and using primitive data types (`number`, `string`, `boolean`, `null`, `undefined`, `symbol`, `bigint`).

1. Declare variables of type `string`, `number`, `boolean`, `null`, and `undefined` and print their types.
2. Check the type of a variable using `typeof`.
3. Convert a string `"123"` into a number and verify its type.
4. Convert a number `456` into a string and concatenate with `" is a number"`.
5. Demonstrate implicit type coercion in addition of a number and string.
6. Check if a variable is `null` or `undefined`.
7. Create a `symbol` and demonstrate that two symbols with the same description are different.
8. Declare a `BigInt` variable and add two BigInts.
9. Create a boolean variable and toggle it using `!`.
10. Demonstrate the difference between `==` and `===` with numbers and strings.
11. Create a string with quotes inside using template literals.
12. Parse a string `"12.34"` into a float using `parseFloat`.
13. Parse a string `"56"` into an integer using `parseInt`.

14. Check if a number is `NaN` and handle it safely.
15. Demonstrate how JavaScript treats `null` and `undefined` in arithmetic operations.

2️⃣ Intermediate Level (16–35)

Focus: type conversions, reference vs primitive, dynamic typing, and type checks.

16. Write a function that converts all string numbers in an array to actual numbers.
17. Implement a function that converts any value to boolean and returns it.
18. Check if a variable is an array using `Array.isArray()`.
19. Convert an array `[1,2,3]` to a string and back to an array.
20. Write a function that sums numbers in an array but ignores non-numeric types.
21. Check if a variable is an object but not `null` or an array.
22. Write a function to safely add two variables that could be numbers or numeric strings.
23. Merge two objects and handle overlapping keys by concatenating values.
24. Write a function that converts a boolean to number and vice versa.
25. Create a function to check if a value is `NaN` or not.
26. Convert a mixed-type array `[1, "2", true, null]` into an array of numbers using rules: `true→1`, `false→0`, `null→0`.
27. Write a function that deeply clones an object with nested properties.
28. Demonstrate how primitive types are copied by value and objects are copied by reference.
29. Create a function that dynamically changes the type of a variable based on input.
30. Convert a number to binary, octal, and hexadecimal strings.

31. Implement a function that validates if a string can be safely converted to a number.
32. Create a function that returns the type of a variable in human-readable form (`Number`, `String`, `Boolean`, `Array`, `Object`).
33. Write a function that converts any type to a string safely.
34. Implement a function to check if a variable is iterable (like array, string, map, set).
35. Create a function to sum numeric values in an object's properties, ignoring non-numeric types.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex type handling, dynamic typing, symbols, BigInt, and interview-level logic.

36. Implement a function that merges two arrays of mixed types into one array of numbers only.
37. Create a function that takes any input and returns a standardized type (`number`, `string`, `boolean`) for further computation.
38. Write a function that counts how many variables of each type exist in an object.
39. Demonstrate a closure that stores variables of different types and performs type-safe operations.
40. Create a function that converts all `BigInt` numbers in an array to normal numbers safely.
41. Implement a function that checks equality of two variables with dynamic type coercion rules.
42. Create a function that converts a nested object into a flat array of values, keeping their type.
43. Write a function that uses `Symbol` as object keys and iterates over all keys safely.

44. Implement a function that dynamically converts all `null` and `undefined` values in an object to zero.

45. Create a function that performs arithmetic on mixed types (numbers, strings, booleans) safely.

46. Write a function that converts a string representation of a boolean (`"true"`, `"false"`) to actual boolean.

47. Implement a function that takes any input and returns a number: `string→parseFloat`, `boolean→1/0`, `null→0`, others→throw error.

48. Write a function that converts an array of numeric strings into an array of BigInt.

49. Implement a function that merges multiple objects with properties of different types, handling conflicts intelligently.

50. Create a function that validates a variable's type against a schema object (e.g., `{name: "string", age: "number"}`) and returns an error for mismatch.

TYPE CONVERSION – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: primitive type conversions (`string ↔ number ↔ boolean`).

1. Convert the string `"123"` to a number using `Number()`.

2. Convert the number `456` to a string using `String()`.

3. Convert boolean `true` to number.

4. Convert number `0` to boolean.

5. Use unary `+` to convert a string `"78"` into a number.
6. Use template literals to convert a number into a string.
7. Convert `null` to a number and explain the result.
8. Convert `undefined` to a number and explain the result.
9. Convert `"false"` string to a boolean using double negation.
10. Use `parseInt` to convert `"123abc"` to number.
11. Use `parseFloat` to convert `"12.34px"` to number.
12. Demonstrate implicit type coercion in `5 + "5"`.
13. Demonstrate implicit type coercion in `"10" - 5`.
14. Convert number `100` to binary, octal, and hexadecimal string.
15. Convert an array `[1,2,3]` to a string using `toString()` and `join()`.

2️⃣ Intermediate Level (16–35)

Focus: advanced conversions, edge cases, reference vs primitive.

16. Write a function that converts all string numbers in an array to actual numbers.
17. Write a function that converts all values in an object to numbers where possible.
18. Convert a string `"true"` or `"false"` into a boolean.
19. Write a function that converts an array of mixed types `[1,"2",true,null]` to numbers.
20. Convert a nested array `[[1,2],[3,4]]` to a flat array of strings.
21. Write a function that converts `BigInt` to number safely, handling overflow.

22. Implement a function that converts a string into a number and returns 0 if it fails.

23. Convert a Set of numbers into an array of strings.

24. Convert a Map into an object with string keys and values.

25. Write a function that converts a numeric string with commas `"1,234"` to a number.

26. Convert a boolean array `[true,false,true]` into an array of 1s and 0s.

27. Write a function that converts a Date object into a timestamp number.

28. Convert a number into a formatted string with 2 decimal places using `toFixed()`.

29. Write a function to convert any iterable (string, array, set) into an array of numbers.

30. Convert a string `"123.456abc"` to a float using `parseFloat` and handle invalid characters.

31. Convert `null` to string, number, and boolean and print results.

32. Convert `undefined` to string, number, and boolean and print results.

33. Convert a string `"0"` to boolean using different methods and compare results.

34. Write a function to safely convert string input to number using `Number()`, `parseInt`, and `parseFloat`, and return the most accurate value.

35. Convert an array of booleans `[true,false,true]` into a single integer treating true as 1 and false as 0 (e.g., `[true,false,true] → 101`).

3️⃣ Advanced / Higher Level (36–50)

Focus: complex conversion logic, dynamic types, and interview-level challenges.

36. Write a function that converts an object of key-value pairs into an array of numbers where possible.

37. Implement a function that converts an array of strings into BigInts safely.

38. Convert a mixed-type array `[1,"2",true,null,"abc"]` into numbers, ignoring invalid conversions.

39. Write a function to dynamically convert a variable to a target type (`string`, `number`, `boolean`).

40. Convert a string `"1e3"` into a number and explain why it works.

41. Write a function that converts all object properties to string recursively.

42. Implement a function that converts a Map to an array of numbers based on the values.

43. Convert a string `"123.456.789"` to a number safely (handle invalid format).

44. Implement a function that converts a nested array of strings into numbers recursively.

45. Write a function that converts any input value to boolean following JS coercion rules but logs the conversion steps.

46. Convert a floating-point number `123.4567` into an integer without rounding.

47. Write a function that converts a string `"true,false,true"` into an array of booleans.

48. Implement a function that converts a date string `"2025-11-13"` into a timestamp number.

49. Convert a string `"0x1A"` into a number using `parseInt` with radix.

50. Write a function that converts any iterable (array, string, set, map) into an array of numbers, ignoring invalid entries, and returns the sum of numbers.

ARRAYS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: array creation, access, length, simple iteration.

1. Create an array of 5 numbers and print all elements.
2. Create an array of strings and print the first and last elements.
3. Find the length of an array.
4. Add an element at the end of an array using `push()`.
5. Remove the last element using `pop()`.
6. Add an element at the beginning using `unshift()`.
7. Remove the first element using `shift()`.
8. Access the middle element of an array.
9. Replace an element at a specific index.
10. Concatenate two arrays using `concat()`.
11. Check if a value exists in an array using `includes()`.
12. Find the index of a specific element using `indexOf()`.
13. Reverse an array using `reverse()`.
14. Sort an array of numbers using `sort()`.
15. Convert an array into a string using `join()`.

2️⃣ Intermediate Level (16–35)

Focus: iteration, filtering, mapping, searching, and array manipulation.

16. Filter all even numbers from an array.
17. Map an array of numbers to their squares.

18. Reduce an array to find the sum of all elements.
19. Find the maximum and minimum number in an array without using `Math.max()` or `Math.min()`.
20. Remove duplicates from an array.
21. Flatten a 2D array into a 1D array.
22. Find all elements greater than a given number.
23. Count the occurrence of each element in an array.
24. Merge two arrays and remove duplicates.
25. Check if an array is sorted in ascending order.
26. Rotate an array to the right by `k` positions.
27. Find the second largest number in an array.
28. Split an array into two halves.
29. Find the first element that satisfies a condition using `find()`.
30. Filter all string elements from a mixed array.
31. Use `every()` to check if all elements satisfy a condition.
32. Use `some()` to check if at least one element satisfies a condition.
33. Remove all falsy values (`0, "", null, undefined, false, NaN`) from an array.
34. Convert an array of strings to uppercase using `map()`.
35. Merge an array of arrays into a single array using `reduce()`.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex array manipulations, nested arrays, higher-order functions, and interview-level challenges.

36. Implement a function to rotate an array left by `k` positions.
37. Implement a function to find the longest increasing subsequence in an array.
38. Group array elements by a specific property (e.g., an array of objects grouped by `age`).
39. Implement a function to find the pair of numbers with the maximum sum.
40. Find all unique pairs in an array that sum to a specific value.
41. Implement a function to move all zeros to the end of an array.
42. Implement a function to flatten a nested array of arbitrary depth.
43. Write a function to remove elements that appear more than `n` times.
44. Implement a function to find the intersection of two arrays.
45. Implement a function to find the difference between two arrays.
46. Implement a function to find the missing number in a sequential array.
47. Implement a function to find duplicates in an array.
48. Write a function to chunk an array into smaller arrays of size `n`.
49. Implement a function to rotate a matrix (2D array) clockwise.
50. Implement a function to find all permutations of elements in an array.

OBJECTS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: object creation, property access, modification, and basic iteration.

1. Create an object representing a person with `name` and `age`.
2. Access and print the `name` property using dot notation.
3. Access and print the `age` property using bracket notation.
4. Add a new property `city` to the object.
5. Modify the `age` property.
6. Delete the `city` property from the object.
7. Check if a property `name` exists using `in` operator.
8. Loop through all properties of an object using `for…in`.
9. Get all keys of an object using `Object.keys()`.
10. Get all values of an object using `Object.values()`.
11. Get all entries (key-value pairs) using `Object.entries()`.
12. Merge two objects using `Object.assign()`.
13. Use `hasOwnProperty()` to check if an object has a specific property.
14. Create a nested object and access a nested property.
15. Create an object using the `new Object()` constructor.

2️⃣ Intermediate Level (16–35)

Focus: object manipulation, dynamic keys, computed properties, and object methods.

16. Create a function that accepts an object and prints its keys and values dynamically.

17. Merge multiple objects into a single object.
18. Dynamically add properties to an object using variables as keys.
19. Create a function to count the number of properties in an object.
20. Convert an object to an array of keys and values and back.
21. Write a function that inverts keys and values of an object.
22. Implement a function to check if two objects are shallowly equal.
23. Create an object with a method that returns the object's full name from `firstName` and `lastName`.
24. Use computed property names to create dynamic keys.
25. Create a function that removes a property from an object dynamically.
26. Write a function to merge two objects and sum the values of numeric keys if they overlap.
27. Loop through an object and print only numeric properties.
28. Write a function that returns the property with the maximum numeric value.
29. Create a function that copies an object deeply (nested objects included).
30. Implement a function that freezes an object and prevents modification.
31. Write a function to create a read-only property in an object.
32. Implement a function that finds all keys in an object whose values match a specific value.
33. Write a function to filter object properties by value type (e.g., keep only strings).
34. Create a function that safely accesses nested object properties using optional chaining.
35. Create an object with getter and setter methods for a property.

3️⃣ Advanced / Higher Level (36–50)

Focus: advanced manipulation, nested objects, dynamic behavior, interview-level problems.

36. Write a function to deeply merge two objects, combining nested objects instead of overwriting.
37. Implement a function to flatten a nested object into a single-level object with dot-separated keys.
38. Write a function that counts all properties (nested included) in an object.
39. Implement a function that converts an object into a query string.
40. Create a function that removes all properties with falsy values from an object.
41. Write a function that finds all paths in a nested object that lead to a specific value.
42. Implement a function to clone an object but replace all string values with their uppercase equivalents.
43. Write a function that swaps keys and values recursively for nested objects.
44. Create a function that filters an array of objects by a property value.
45. Implement a function that calculates the sum of all numeric properties in an object, including nested ones.
46. Write a function that merges an array of objects by a key, grouping values into arrays.
47. Implement a function to detect circular references in a nested object.
48. Create a function to deep freeze an object (including nested objects).
49. Write a function that updates a nested property dynamically using a path string (e.g., `"address.city"`).
50. Implement a function that converts a nested object into a tree structure suitable for rendering a menu.

STRINGS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: string creation, access, length, and basic operations.

1. Declare a string variable and print its length.
2. Access the first and last characters of a string.
3. Concatenate two strings using `+` operator.
4. Concatenate two strings using template literals.
5. Convert a string to uppercase.
6. Convert a string to lowercase.
7. Trim whitespace from the start and end of a string.
8. Extract a substring using `slice()`.
9. Extract a substring using `substring()`.
10. Replace a word in a string using `replace()`.
11. Replace all occurrences of a word using `replaceAll()`.
12. Check if a string includes a substring using `includes()`.
13. Check if a string starts with a specific substring using `startsWith()`.
14. Check if a string ends with a specific substring using `endsWith()`.
15. Split a string into an array of words using `split()`.

2️⃣ Intermediate Level (16–35)

Focus: string manipulation, search, formatting, and dynamic operations.

16. Reverse a string without using built-in reverse methods.
17. Count the number of vowels in a string.
18. Count the number of words in a string.
19. Find the first occurrence of a character using `indexOf()`.
20. Find the last occurrence of a character using `lastIndexOf()`.
21. Capitalize the first letter of each word in a string.
22. Remove all vowels from a string.
23. Check if a string is a palindrome.
24. Convert a hyphen-separated string into camelCase.
25. Convert a snake_case string into camelCase.
26. Find all occurrences of a substring in a string.
27. Replace all spaces in a string with hyphens.
28. Count how many times a specific word occurs in a string.
29. Extract all numbers from a string and return them as an array of numbers.
30. Remove extra spaces between words (normalize spacing).
31. Mask part of a string with asterisks (e.g., mask email or phone).
32. Convert a string into an array of characters and perform some operation (e.g., double vowels).
33. Repeat a string `n` times using `repeat()`.
34. Create a function that compresses repeated characters (e.g., `"aaabbc"` → `"a3b2c1"`).

35. Remove all non-alphanumeric characters from a string.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex string manipulations, regex, and interview-level challenges.

36. Write a function to check if two strings are anagrams.
37. Find the longest substring without repeating characters.
38. Implement your own `trim()` function without using built-in methods.
39. Convert a string representing a number with commas into an actual number (`"1,234,567"` → `1234567`).
40. Implement a function that finds the most frequent character in a string.
41. Replace all numbers in a string with their word equivalents (`"I have 2 apples"` → `"I have two apples"`).
42. Extract all email addresses from a string using regex.
43. Implement a function to reverse the order of words in a sentence.
44. Implement a function to capitalize alternate words in a string.
45. Detect and return all palindromic words in a string.
46. Convert a string to title case (capitalize first letter of each word).
47. Write a function to remove all duplicate characters from a string.
48. Implement a function to check if a string contains only unique characters.
49. Convert a string of digits into its corresponding English words (`"123"` → `"one two three"`).
50. Implement a function to find all substrings of length `k` that are palindromes.

STACK & HEAP IN MEMORY – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: primitive vs reference types, memory basics, stack vs heap understanding.

1. Create a number variable and a string variable. Explain where they are stored (stack or heap).
2. Create an object and explain its storage in memory.
3. Assign one primitive variable to another and modify the first. Observe the second.
4. Assign one object to another and modify the first. Observe the second.
5. Demonstrate how passing a primitive to a function does not change the original value.
6. Demonstrate how passing an object to a function can modify the original object.
7. Compare two different objects with the same content using `==` and `===`.
8. Explain the effect of changing a property in a nested object on other references.
9. Create a primitive variable and log its memory address (conceptually, explain stack storage).
10. Create an object and log its memory address (conceptually, explain heap storage).
11. Create an array and assign it to another variable. Modify one and check the other.
12. Explain the difference in memory allocation between strings and objects.
13. Demonstrate stack behavior using function calls (nested function calls).

14. Demonstrate that primitive variables are copied by value when assigned.
15. Demonstrate that objects are copied by reference when assigned.

2️⃣ Intermediate Level (16–35)

Focus: closures, memory leaks, reference handling, and stack/heap behavior.

16. Write a function that returns a closure storing a counter value. Explain where the counter is stored.
17. Create two objects and swap their references without copying values.
18. Create a function that modifies a nested object property and explain memory effects.
19. Create a deep copy of an object and modify the copy. Show the original remains unchanged.
20. Create a shallow copy of an object and modify the copy. Show how the original changes.
21. Implement a function that uses recursion to sum numbers in a nested object or array.
22. Explain what happens to variables when a function scope ends.
23. Demonstrate memory behavior when large arrays are copied by reference vs by value.
24. Implement a function that creates a circular reference in an object.
25. Explain the memory implications of circular references in JavaScript.
26. Create a function that generates a closure in a loop and observe captured variables.
27. Explain the difference in memory between string concatenation using `+` vs template literals.
28. Demonstrate how garbage collection removes unreferenced objects.
29. Implement a memoization function and explain memory usage.
30. Explain what happens in memory when `let` and `const` variables go out of scope.

31. Show how passing an object to multiple functions affects memory.
32. Implement a function that stores data in a global object and explain heap impact.
33. Explain memory difference between primitive arrays and array of objects.
34. Show how modifying a property of an object in a nested array affects other references.
35. Demonstrate stack overflow by writing a deeply recursive function.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex memory management, recursion, closures, heap optimization, and interview-level problems.

36. Implement a recursive function to reverse a linked list stored as nested objects.
37. Write a function that creates multiple closures holding references to large arrays and discuss memory optimization.
38. Implement a function that avoids memory leaks by properly nullifying references.
39. Demonstrate how a circular linked list is stored in heap memory.
40. Create a function that dynamically generates objects and cleans up unused ones to avoid memory leaks.
41. Explain the difference in memory between using spread operator vs direct assignment for objects.
42. Write a function to merge multiple large objects without duplicating memory unnecessarily.
43. Demonstrate stack and heap usage in recursive factorial computation.
44. Implement a function to deep clone an object with nested arrays efficiently.
45. Create a function that simulates a stack using arrays and explain memory use.
46. Create a function that simulates a heap using objects with dynamic allocation.

47. Implement a function that tracks memory usage of created objects (simulate).
48. Write a function to safely handle very large arrays without crashing the stack.
49. Demonstrate closures retaining references after the outer function scope ends.
50. Implement a function to clean up a large object graph to allow garbage collection.

NUMBERS & MATH – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: number operations, basic math functions, type conversion.

1. Declare a number variable and print its value.
2. Increment and decrement a number using `++` and `--`.
3. Add, subtract, multiply, and divide two numbers.
4. Find the remainder using the modulo operator `%`.
5. Convert a numeric string `"123"` to a number and perform addition.
6. Round a number `4.7` to the nearest integer using `Math.round()`.
7. Round a number `4.7` down using `Math.floor()`.
8. Round a number `4.3` up using `Math.ceil()`.
9. Find the absolute value of a number using `Math.abs()`.

10. Find the maximum and minimum of three numbers using `Math.max()` and `Math.min()`.
11. Generate a random number between 0 and 1 using `Math.random()`.
12. Generate a random integer between 1 and 100.
13. Convert a number to a string using `toString()`.
14. Find the square root of a number using `Math.sqrt()`.
15. Raise a number to a power using `Math.pow()` or `**` operator.

2️⃣ Intermediate Level (16–35)

Focus: array math, loops, number manipulation, and practical use of `Math`.

16. Find the sum of all numbers from 1 to 100 using a loop.
17. Find the factorial of a number using a loop.
18. Find all prime numbers up to `n`.
19. Check if a number is prime.
20. Find the greatest common divisor (GCD) of two numbers.
21. Find the least common multiple (LCM) of two numbers.
22. Sum all numbers in an array.
23. Find the maximum and minimum in an array without using `Math.max()`/`Math.min()`.
24. Calculate the average of numbers in an array.
25. Round all numbers in an array to the nearest integer.
26. Count the number of even and odd numbers in an array.

27. Write a function to check if a number is a perfect square.
28. Write a function to check if a number is a perfect cube.
29. Generate a random number within a range `[min, max]`.
30. Convert a temperature from Celsius to Fahrenheit.
31. Convert a temperature from Fahrenheit to Celsius.
32. Check if a number is an Armstrong number.
33. Calculate the sum of digits of a number.
34. Reverse a number (e.g., `1234 → 4321`).
35. Check if a number is a palindrome (e.g., `121 → true`).

3️⃣ Advanced / Higher Level (36–50)

Focus: advanced number algorithms, optimization, and interview-level challenges.

36. Implement Euclid's algorithm for GCD recursively.
37. Find all divisors of a number.
38. Find the nth Fibonacci number using recursion.
39. Find the nth Fibonacci number using iteration.
40. Generate a list of Fibonacci numbers up to `n`.
41. Find the sum of all prime numbers up to `n`.
42. Find the largest prime factor of a number.
43. Implement the Sieve of Eratosthenes to find all primes up to `n`.
44. Check if two numbers are co-prime.

45. Calculate the factorial of a number using recursion.
46. Implement a function to approximate `π` using the Leibniz series.
47. Solve a quadratic equation and return real roots (if any).
48. Round a floating-point number to `n` decimal places.
49. Implement a function to calculate the sum of the first `n` natural numbers without a loop (formula-based).
50. Generate a random password consisting of numbers and math symbols of a given length.

DATE & TIME – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: creating Date objects, accessing date components, and simple operations.

1. Create a Date object representing the current date and time.
2. Create a Date object for your birthday.
3. Print the year of a Date object using `getFullYear()`.
4. Print the month (0–11) using `getMonth()`.
5. Print the day of the month using `getDate()`.
6. Print the day of the week using `getDay()`.

7. Print hours, minutes, and seconds using `getHours()`, `getMinutes()`, `getSeconds()`.
8. Set the year of a Date object using `setFullYear()`.
9. Set the month of a Date object using `setMonth()`.
10. Set the date of a Date object using `setDate()`.
11. Convert a Date object to a string using `toDateString()`.
12. Convert a Date object to a time string using `toTimeString()`.
13. Convert a Date object to ISO format using `toISOString()`.
14. Get the timestamp (milliseconds since 1970) using `getTime()`.
15. Create a Date object from a timestamp using `new Date(timestamp)`.

2️⃣ Intermediate Level (16–35)

Focus: date manipulation, calculations, formatting, and comparisons.

16. Calculate the number of days between two dates.
17. Calculate the difference in hours between two Date objects.
18. Add 7 days to a given date.
19. Subtract 3 months from a given date.
20. Compare two dates and return which one is earlier.
21. Check if a given date is a weekend.
22. Check if a given date is in the current month.
23. Format a Date object to `YYYY-MM-DD` format.

24. Format a Date object to `DD/MM/YYYY` format.
25. Format a Date object to `MM-DD-YYYY` format.
26. Get the number of milliseconds, seconds, minutes, and hours in a Date object.
27. Find the last day of a given month.
28. Find the first day of a given month.
29. Determine if a given year is a leap year.
30. Calculate the age based on the birth date.
31. Get the week number of a given date.
32. Round a date to the nearest hour.
33. Convert a UTC date to local time.
34. Convert a local date to UTC.
35. Find the number of weekdays between two dates.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex date logic, time zones, scheduling, and interview-level challenges.

36. Implement a countdown timer to a specific date.
37. Calculate the number of business days between two dates.
38. Determine the next Friday from today’s date.
39. Calculate the difference between two dates in weeks.
40. Add/subtract arbitrary months, accounting for varying month lengths.
41. Calculate the number of days remaining in the current year.

42. Calculate the number of seconds since the start of the year.
43. Implement a function to format a date in `MMM DD, YYYY` format (e.g., `Nov 13, 2025`).
44. Find all Fridays that fall on the 13th of any month in a given year.
45. Convert a timestamp to a human-readable relative time (`2 days ago`, `3 hours ago`).
46. Implement a function that returns the start and end dates of the current week.
47. Implement a function that returns the start and end dates of the current month.
48. Implement a function that calculates the next N occurrences of a specific weekday.
49. Parse a date string in multiple formats and create a Date object.
50. Implement a scheduler function that triggers an event at a specified date and time.

FUNCTIONS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: function declaration, invocation, parameters, and return values.

1. Write a function that prints `"Hello World"` to the console.
2. Write a function that takes two numbers and returns their sum.
3. Write a function that takes a name and returns a greeting message.
4. Write a function that returns the square of a number.
5. Write a function that returns the largest of two numbers.

6. Write a function that swaps two numbers using a temporary variable.
7. Write a function that swaps two numbers without using a temporary variable.
8. Write a function that checks if a number is even.
9. Write a function that checks if a number is odd.
10. Write a function that calculates the factorial of a number.
11. Write a function that returns the nth Fibonacci number.
12. Write a function with default parameters and demonstrate calling it with/without arguments.
13. Write a function that converts Celsius to Fahrenheit.
14. Write a function that converts Fahrenheit to Celsius.
15. Write a function that concatenates two strings.

2️⃣ Intermediate Level (16–35)

Focus: higher-order functions, recursion, multiple parameters, and closure.

16. Write a function that takes an array and returns the sum of its elements.
17. Write a function that takes an array and returns a new array with elements squared.
18. Write a recursive function to find the factorial of a number.
19. Write a recursive function to calculate Fibonacci numbers.
20. Write a function that takes another function as a parameter and executes it (callback).
21. Write a function that returns another function (closure) that adds a fixed number.
22. Write a function to check if a string is a palindrome.
23. Write a function to reverse a string.

24. Write a function that filters an array of numbers greater than a given value.
25. Write a function that finds the maximum number in an array.
26. Write a function that finds the minimum number in an array.
27. Write a function that removes duplicates from an array.
28. Write a function that merges two arrays and removes duplicates.
29. Write a function that counts the number of vowels in a string.
30. Write a function that counts occurrences of each character in a string.
31. Write a function that validates if an email string is valid using regex.
32. Write a function that calculates the power of a number using recursion.
33. Write a function that generates an array of `n` random numbers.
34. Write a function that takes any number of arguments and returns their sum.
35. Write a function that returns the type of a variable passed to it.

3️⃣ Advanced / Higher Level (36–50)

Focus: arrow functions, closures, IIFE, recursion, currying, and interview-level challenges.

36. Convert a normal function to an arrow function and explain differences.
37. Write a function that returns a closure to store a private counter with increment and decrement methods.
38. Write an Immediately Invoked Function Expression (IIFE) that prints `"IIFE executed"`.
39. Implement a function that curries a function taking multiple parameters.
40. Write a recursive function to flatten a nested array.

41. Write a function that memoizes another function to optimize performance.

42. Write a function that generates Fibonacci numbers up to `n` using closure.

43. Implement a function that finds all prime numbers up to `n` using recursion.

44. Write a function that returns another function for dynamic multiplication (e.g., `multiplyBy(5)(3) → 15`).

45. Implement a function that throttles another function (execute at most once in `n` ms).

46. Implement a function that debounces another function (execute after `n` ms of inactivity).

47. Write a function that deeply clones an object using recursion.

48. Implement a function that calculates the sum of numbers in a nested array using recursion.

49. Write a function that limits the number of times another function can be called.

50. Implement a function that dynamically creates getter and setter functions for an object property.

ARROW FUNCTIONS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: arrow function syntax, basic operations, and return values.

1. Convert a regular function that adds two numbers into an arrow function.

2. Convert a function that multiplies two numbers into an arrow function.

3. Convert a function that returns a string message into an arrow function.
4. Create an arrow function that returns the square of a number.
5. Create an arrow function that checks if a number is even.
6. Create an arrow function that checks if a number is odd.
7. Write an arrow function that returns the length of a string.
8. Write an arrow function that returns the first character of a string.
9. Write an arrow function that concatenates two strings.
10. Write an arrow function that calculates the factorial of a number using recursion.
11. Write an arrow function that returns the maximum of two numbers.
12. Write an arrow function that converts Celsius to Fahrenheit.
13. Write an arrow function with default parameters.
14. Write a concise arrow function (single-line) to add two numbers.
15. Write an arrow function that takes no parameters and returns a fixed value.

2️⃣ Intermediate Level (16–35)

Focus: arrays, callbacks, higher-order functions, and closures with arrow functions.

16. Use an arrow function in `map()` to square each number in an array.
17. Use an arrow function in `filter()` to get only even numbers from an array.
18. Use an arrow function in `reduce()` to calculate the sum of an array.
19. Write an arrow function that returns another arrow function (closure).
20. Implement an arrow function that reverses a string.

21. Use an arrow function in `forEach()` to print array elements.
22. Use an arrow function in `some()` to check if any number is greater than 10.
23. Use an arrow function in `every()` to check if all numbers are positive.
24. Implement an arrow function that finds the longest word in an array.
25. Implement an arrow function that removes duplicates from an array.
26. Use an arrow function in `sort()` to sort an array of numbers in descending order.
27. Implement an arrow function to capitalize the first letter of each word in a string.
28. Write an arrow function that merges two arrays and removes duplicates.
29. Use an arrow function in `find()` to return the first number divisible by 3.
30. Write an arrow function that calculates the factorial using a loop.
31. Write an arrow function to generate an array of first `n` Fibonacci numbers.
32. Write an arrow function that converts an array of strings to uppercase.
33. Use an arrow function to implement a simple debounce function.
34. Use an arrow function to implement a simple throttle function.
35. Implement an arrow function that returns the type of a given value.

3️⃣ Advanced / Higher Level (36–50)

Focus: advanced arrow function patterns, recursion, currying, memoization, and interview-level challenges.

36. Implement a recursive arrow function to flatten a nested array.
37. Write an arrow function that memoizes another function.

38. Write an arrow function that curries a multiplication function.
39. Implement a closure using arrow functions to store a private counter.
40. Write an arrow function to generate a random integer between min and max.
41. Implement an arrow function that finds all prime numbers up to `n`.
42. Write an arrow function to check if a string is a palindrome.
43. Implement an arrow function to calculate the sum of numbers in a nested array.
44. Write an arrow function that returns another function for dynamic addition (e.g., `add(5)(3) → 8`).
45. Write an arrow function to find the nth Fibonacci number using recursion.
46. Implement an arrow function that deeply clones an object.
47. Write an arrow function that calculates the sum of digits of a number recursively.
48. Implement an arrow function to throttle another function (execute at most once every `n` ms).
49. Implement an arrow function to debounce another function (execute after `n` ms of inactivity).
50. Write an arrow function that dynamically creates getter and setter functions for an object property.

CONTROL FLOW – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: `if`, `else if`, `else` statements and basic conditions.

1. Write a program to check if a number is positive, negative, or zero.
2. Write a program to check if a number is even or odd.
3. Write a program to find the largest of two numbers.
4. Write a program to find the largest of three numbers.
5. Write a program to check if a person is eligible to vote (age ≥ 18).
6. Write a program to check if a character is a vowel or consonant.
7. Write a program to check if a number is divisible by 5 and 11.
8. Write a program to check if a number is divisible by either 5 or 11.
9. Write a program to check if a year is a leap year.
10. Write a program to check if a number is positive and even.
11. Write a program to check if a number is positive or divisible by 3.
12. Write a program to determine if a person is a teenager (age between 13–19).
13. Write a program to check if a number is a single-digit, double-digit, or more.
14. Write a program to check if a character is uppercase, lowercase, or not an alphabet.
15. Write a program to classify a number as small (<10), medium (10–100), or large (>100).

2️⃣ Intermediate Level (16–35)

Focus: multiple conditions, logical operators, and `switch` statements.

16. Write a program to calculate the grade based on marks using `if/else`.
17. Write a program to check if three sides can form a triangle.

18. Write a program to check the type of triangle (equilateral, isosceles, scalene).
19. Write a program to check if a number is divisible by 2, 3, and 5.
20. Write a program to find the smallest among three numbers.
21. Write a program to print the day of the week using `switch`.
22. Write a program to print the month name based on the month number using `switch`.
23. Write a program to determine if a year is a leap year using nested `if`.
24. Write a program to determine eligibility for multiple conditions: age, citizenship, and registration.
25. Write a program to check if a character is a vowel using `switch`.
26. Write a program to implement a simple calculator using `switch` (+, -, *, /).
27. Write a program to classify angles (acute, right, obtuse, straight) using `if/else`.
28. Write a program to check if a number is a multiple of 3 or 7 but not both.
29. Write a program to categorize a person based on age groups (child, teen, adult, senior).
30. Write a program to check if a number is divisible by any of 2, 3, or 5.
31. Write a program to assign a letter grade (A, B, C, D, F) using `switch`.
32. Write a program to print the number of days in a month using `switch`.
33. Write a program to find the maximum of four numbers using nested `if/else`.
34. Write a program to check if a number is divisible by 4 and 6 but not by 9.
35. Write a program to classify a triangle based on angles using `if/else`.

---

3 Advanced / Higher Level (36–50)

Focus: complex conditions, nested control flow, decision-making, and interview-level challenges.

36. Write a program to implement a simple menu system using `switch` with multiple options.
37. Write a program to determine the number of digits in a number using `if/else`.
38. Write a program to find the second largest number among four numbers using nested `if`.
39. Write a program to check if three numbers can form a Pythagorean triplet.
40. Write a program to calculate the total cost with different discounts based on price ranges.
41. Write a program to classify triangles based on both sides and angles.
42. Write a program to determine the quadrant of a point `(x, y)` in a 2D plane.
43. Write a program to check if a year is a leap year and century leap year correctly.
44. Write a program to implement a basic role-based access control using `switch`.
45. Write a program to determine eligibility for a loan based on multiple conditions.
46. Write a program to find the largest even number among four numbers.
47. Write a program to determine if a number is prime using nested `if`.
48. Write a program to calculate tax based on different slabs using `if/else`.
49. Write a program to implement a basic grading system with plus/minus grades using `switch`.
50. Write a program to simulate a vending machine selection using `switch` and nested `if/else`.

LOOPING – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: `for`, `while`, `do…while` loops and simple numeric iterations.

1. Print numbers from 1 to 10 using a `for` loop.
2. Print numbers from 1 to 10 using a `while` loop.
3. Print numbers from 1 to 10 using a `do…while` loop.
4. Print numbers from 10 to 1 in descending order using a `for` loop.
5. Print even numbers between 1 and 20.
6. Print odd numbers between 1 and 20.
7. Print the first `n` natural numbers.
8. Print the first `n` odd numbers.
9. Print the first `n` even numbers.
10. Print the multiplication table of 5.
11. Print numbers divisible by 3 between 1 and 50.
12. Print numbers divisible by 5 but not by 2 between 1 and 50.
13. Sum of numbers from 1 to 100 using a loop.
14. Calculate the factorial of a number using a loop.
15. Print the Fibonacci sequence up to `n` numbers using a loop.

2️⃣ Intermediate Level (16–35)

Focus: nested loops, arrays, strings, and pattern problems.

16. Print a triangle pattern using `*` with `n` rows.
17. Print a pyramid pattern of numbers.
18. Print a reverse pyramid pattern.
19. Print all elements of an array using a `for` loop.
20. Print all elements of an array using `for…of` loop.
21. Print all elements of an array using `forEach`.
22. Find the maximum number in an array using a loop.
23. Find the minimum number in an array using a loop.
24. Calculate the sum of all elements in an array.
25. Reverse an array using a loop (without `reverse()` method).
26. Print a right-angled triangle of numbers.
27. Print a diamond pattern using nested loops.
28. Print all prime numbers between 1 and `n` using loops.
29. Find the sum of all even numbers in an array.
30. Find the sum of all odd numbers in an array.
31. Count the frequency of each element in an array.
32. Print a multiplication table of `n` using nested loops.
33. Print the ASCII values of all characters in a string using a loop.
34. Reverse a string using a loop.

35. Print all substrings of a string using nested loops.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex loops, multi-dimensional arrays, higher-order loops, and interview-level challenges.

36. Print Pascal’s Triangle up to `n` rows.
37. Print a hollow square pattern using `*`.
38. Rotate elements of an array by `k` positions using loops.
39. Find the second largest number in an array using loops.
40. Find the second smallest number in an array using loops.
41. Implement bubble sort using loops.
42. Implement selection sort using loops.
43. Implement insertion sort using loops.
44. Print all prime factors of a number using loops.
45. Find all pairs in an array that sum up to a target value.
46. Print all elements of a 2D array using nested loops.
47. Find the sum of the main diagonal in a 2D array.
48. Find the sum of the secondary diagonal in a 2D array.
49. Transpose a 2D array using nested loops.
50. Implement a spiral traversal of a 2D array using nested loops.

HIGHER-ORDER LOOPS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: understanding `map`, `filter`, `reduce`, `forEach` with simple operations.

1. Use `map()` to double each number in an array.
2. Use `map()` to convert all strings in an array to uppercase.
3. Use `filter()` to get only even numbers from an array.
4. Use `filter()` to get numbers greater than 10 from an array.
5. Use `reduce()` to calculate the sum of an array of numbers.
6. Use `reduce()` to calculate the product of an array of numbers.
7. Use `forEach()` to print each element of an array.
8. Use `forEach()` to print the index and value of each array element.
9. Use `map()` to add 5 to each element of an array.
10. Use `filter()` to get all strings that start with a specific letter.
11. Use `reduce()` to find the maximum number in an array.
12. Use `reduce()` to find the minimum number in an array.
13. Use `forEach()` to create a new array of squares of each number (manual push).
14. Use `map()` to convert an array of numbers to strings.

15. Use `filter()` to remove all falsy values from an array.

2️⃣ Intermediate Level (16–35)

Focus: array manipulations, nested data, and combining higher-order functions.

16. Use `map()` and `filter()` together to get squares of even numbers.
17. Use `reduce()` to count the total number of elements in a nested array.
18. Use `reduce()` to flatten a 2D array into a 1D array.
19. Use `map()` to extract a property from an array of objects.
20. Use `filter()` to get objects with a property value greater than a threshold.
21. Use `reduce()` to sum up a specific property in an array of objects.
22. Use `map()` to prepend a string to each element of an array.
23. Use `filter()` to remove duplicate elements from an array.
24. Use `reduce()` to group objects by a property.
25. Use `forEach()` to update elements of an array in-place.
26. Use `map()` to convert an array of temperatures from Celsius to Fahrenheit.
27. Use `filter()` to get numbers divisible by 3 or 5.
28. Use `reduce()` to find the longest string in an array.
29. Use `map()` to create an array of boolean values based on a condition.
30. Use `filter()` to find words longer than `n` characters.
31. Use `reduce()` to create a frequency counter for array elements.

32. Use `map()` to reverse each string in an array of strings.
33. Use `filter()` to keep only positive numbers from an array.
34. Use `reduce()` to calculate the product of all odd numbers in an array.
35. Use `forEach()` to log elements conditionally (e.g., log only numbers > 10).

3️⃣ Advanced / Higher Level (36–50)

Focus: complex logic, chaining, nested arrays/objects, and interview-level challenges.

36. Use `map()` and `reduce()` to calculate the total price of products in a cart.
37. Use `filter()` and `reduce()` to find the sum of all even numbers in an array.
38. Use `map()` to extract nested object properties into a new array.
39. Use `reduce()` to flatten a deeply nested array (multi-level).
40. Use `map()` and `filter()` to implement a “search” feature in an array of objects.
41. Use `reduce()` to create an object where keys are array values and values are their counts.
42. Chain `filter()` → `map()` → `reduce()` to process numbers in an array (e.g., sum of squares of positive numbers).
43. Use `map()` to add an index property to each object in an array.
44. Use `reduce()` to group strings by their first letter.
45. Use `filter()` and `map()` to normalize and clean an array of strings.
46. Use `reduce()` to merge an array of objects into a single object.
47. Use `map()` to convert an array of date strings to `Date` objects.

48. Use `filter()` to find objects with multiple conditions (e.g., age > 18 and active = true).

49. Implement a “top N elements” finder using `sort()` and `slice()` with `map()`/`filter()`.

50. Chain `map()`, `filter()`, `reduce()`, and `forEach()` to generate a report summary from an array of objects (e.g., total sales, average rating, count of products above threshold).

`forEach` – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: understanding `forEach` syntax, iteration over arrays, and simple operations.

1. Use `forEach()` to print each element of an array.

2. Use `forEach()` to print each element with its index.

3. Use `forEach()` to print elements of an array in reverse order.

4. Use `forEach()` to print only even numbers from an array.

5. Use `forEach()` to print only odd numbers from an array.

6. Use `forEach()` to sum all numbers in an array.

7. Use `forEach()` to multiply each number in an array by 2 (without modifying the original array).

8. Use `forEach()` to print all strings in uppercase.

9. Use `forEach()` to print the length of each string in an array.
10. Use `forEach()` to find the largest number in an array.
11. Use `forEach()` to find the smallest number in an array.
12. Use `forEach()` to count the number of positive numbers.
13. Use `forEach()` to count the number of negative numbers.
14. Use `forEach()` to print numbers divisible by 5.
15. Use `forEach()` to print a custom message for each element (e.g., `"Element: X"`).

2️⃣ Intermediate Level (16–35)

Focus: arrays of objects, nested arrays, transformations, and conditional logic.

16. Use `forEach()` to print the names of all students in an array of objects.
17. Use `forEach()` to print the ages of all people in an array of objects.
18. Use `forEach()` to calculate the total score of students from an array of objects.
19. Use `forEach()` to find students with marks above 75.
20. Use `forEach()` to find students below 18 years of age.
21. Use `forEach()` to modify all numbers in an array in-place by multiplying by 3.
22. Use `forEach()` to reverse all strings in an array in-place.
23. Use `forEach()` to print all elements of a nested array.
24. Use `forEach()` to flatten a 2D array into a 1D array.
25. Use `forEach()` to create a new array with squares of each number.

26. Use `forEach()` to create a new array with only uppercase strings.
27. Use `forEach()` to concatenate all strings in an array into a single string.
28. Use `forEach()` to count the frequency of each element in an array.
29. Use `forEach()` to remove falsy values from an array in-place.
30. Use `forEach()` to create an object where keys are array elements and values are their indices.
31. Use `forEach()` to find all numbers divisible by both 2 and 3.
32. Use `forEach()` to calculate the sum of all even numbers.
33. Use `forEach()` to calculate the sum of all odd numbers.
34. Use `forEach()` to print elements that are prime numbers.
35. Use `forEach()` to check if all numbers in an array are positive.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex arrays, chaining with other methods, nested objects, and interview-level challenges.

36. Use `forEach()` to update all objects in an array to include a new property.
37. Use `forEach()` to normalize all string values in an array of objects.
38. Use `forEach()` to sum a property from an array of objects (e.g., total salary).
39. Use `forEach()` to find the maximum value of a property in an array of objects.
40. Use `forEach()` to find the minimum value of a property in an array of objects.
41. Use `forEach()` to filter and print objects based on multiple property conditions.

42. Use `forEach()` to implement a simple leaderboard ranking system.
43. Use `forEach()` to flatten a deeply nested array of arrays.
44. Use `forEach()` to create a map of object IDs to object names.
45. Use `forEach()` to implement a custom frequency counter for words in a paragraph.
46. Use `forEach()` to remove duplicates from an array in-place.
47. Use `forEach()` to implement a simple calculator that logs results of operations on two arrays.
48. Use `forEach()` to track cumulative sums at each step in an array.
49. Use `forEach()` with `try/catch` to safely process an array with potential errors.
50. Use `forEach()` to generate a report from an array of objects (e.g., total sales, average rating, number of products above threshold).

DOM – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: selecting, reading, and manipulating basic DOM elements.

1. Select an element by `id` and change its text content.
2. Select elements by `class` and change their text content.
3. Select elements by `tag` and change their background color.
4. Get the value of an input field.

5. Set the value of an input field.
6. Get the inner HTML of a div.
7. Set the inner HTML of a div.
8. Change the style (color, font-size) of an element using `style` property.
9. Hide an element by setting `display: none`.
10. Show a hidden element by setting `display: block`.
11. Add a new paragraph element inside a div.
12. Remove a paragraph element from the DOM.
13. Add a class to an element using `classList.add()`.
14. Remove a class from an element using `classList.remove()`.
15. Toggle a class on an element using `classList.toggle()`.

2️⃣ Intermediate Level (16–35)

Focus: DOM traversal, dynamic manipulation, attributes, and event handling.

16. Select the first child of a parent element.
17. Select the last child of a parent element.
18. Select all children of a parent element.
19. Select the parent of a given element.
20. Select the next sibling of an element.
21. Select the previous sibling of an element.
22. Change the `src` attribute of an image.

23. Change the `href` attribute of a link.
24. Get all elements with a specific data attribute.
25. Set a custom data attribute for an element.
26. Create a new list item `<li>` and append it to a `<ul>`.
27. Remove a list item from a `<ul>`.
28. Clone an element and append it to the DOM.
29. Add an event listener to a button for `click` that changes text content.
30. Add an event listener to an input for `change` that logs the new value.
31. Remove an event listener from an element.
32. Add multiple event listeners to elements using a loop.
33. Prevent the default action of a form submission using `preventDefault()`.
34. Stop event propagation using `stopPropagation()`.
35. Dynamically change the style of all paragraphs on button click.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex DOM manipulation, dynamic content, forms, tables, and real-world tasks.

36. Implement a dynamic to-do list where items can be added and removed.
37. Implement a live search filter for a list of items.
38. Create a table dynamically from an array of objects.
39. Update a table row dynamically based on user input.
40. Delete a table row dynamically.

41. Implement a dropdown menu that shows/hides submenus.
42. Create an image gallery that updates the main image when thumbnails are clicked.
43. Implement a modal popup that opens and closes dynamically.
44. Change the text color of list items alternatingly (odd/even).
45. Count the number of visible elements on a page dynamically.
46. Implement a rating system with clickable stars.
47. Implement tabbed content where clicking a tab shows related content.
48. Dynamically sort table rows based on column values.
49. Implement infinite scrolling: append more items as the user scrolls down.
50. Implement drag-and-drop functionality for elements.

EVENTS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: attaching, handling, and understanding basic events.

1. Add a `click` event listener to a button that shows an alert.
2. Add a `mouseover` event to change the background color of a div.
3. Add a `mouseout` event to reset the background color.
4. Add a `dblclick` event to a paragraph that changes its text.
5. Add a `focus` event to an input that highlights the border.

6. Add a `blur` event to an input that resets the border.
7. Add a `keydown` event to log the pressed key.
8. Add a `keyup` event to display the typed character in a div.
9. Add a `keypress` event to count the number of keys pressed.
10. Add a `submit` event to a form and prevent the default submission.
11. Add a `change` event to a dropdown and log the selected value.
12. Add a `resize` event to the window that logs the new dimensions.
13. Add a `scroll` event to detect scrolling on a div.
14. Add a `contextmenu` event to prevent right-click on an element.
15. Add a `load` event to execute a function after the page is fully loaded.

2️⃣ Intermediate Level (16–35)

Focus: event delegation, dynamic elements, event objects, and conditional handling.

16. Add a `click` event to multiple buttons using a loop.
17. Use event delegation to handle clicks on list items in a `<ul>`.
18. Add a `mouseover` event to highlight table rows.
19. Use the event object to get the target element of an event.
20. Use the event object to get the mouse coordinates on `mousemove`.
21. Add a `keydown` event to detect when the Enter key is pressed.
22. Add a `keydown` event to detect combination keys (e.g., Ctrl + S).

23. Prevent multiple form submissions using a `submit` event.
24. Add a `click` event that toggles the visibility of a section.
25. Use `stopPropagation()` to prevent an event from bubbling up.
26. Use `stopImmediatePropagation()` to prevent multiple listeners on the same element.
27. Add a `mouseenter` event to change styles without triggering child events.
28. Add a `mouseleave` event to revert styles.
29. Add a `wheel` event to detect scroll direction.
30. Add a `dragstart` event to log the dragged element.
31. Add a `dragover` event to allow dropping.
32. Add a `drop` event to append a dragged element to a target container.
33. Dynamically add buttons and attach a `click` event to each new button.
34. Use `once: true` option to execute an event listener only once.
35. Add a `touchstart` event for mobile devices and log the touch coordinates.

3️⃣ Advanced / Higher Level (36–50)

Focus: complex event handling, custom events, debouncing, throttling, and interview-level challenges.

36. Implement a debounce function for an `input` event (search bar).
37. Implement a throttle function for `scroll` events.
38. Create and dispatch a custom event when a condition is met.

39. Listen for a custom event on a parent element.
40. Create a key sequence detector (e.g., Konami code) using `keydown`.
41. Implement drag-and-drop sorting of list items.
42. Implement resizing of a div using `mousedown`, `mousemove`, and `mouseup` events.
43. Implement a tooltip that shows on `mouseover` and hides on `mouseout`.
44. Implement a modal popup with open/close events and overlay click handling.
45. Track mouse movement over a canvas and draw lines dynamically.
46. Implement a double-click toggle functionality on an element.
47. Implement event delegation for dynamically added table rows.
48. Use events to dynamically update a live character count in a text area.
49. Implement a swipe detection on a touch device using `touchstart` and `touchend`.
50. Implement a drag-and-drop image upload area with preview using events.

ASYNCHRONOUS FUNCTIONS – JavaScript Practice Questions (Basic → Intermediate → Advanced)

1️⃣ Basic Level (1–15)

Focus: understanding `Promise` creation, `then`, `catch`, and basic `async/await`.

1. Create a simple Promise that resolves with `"Hello World"` and log the result using `.then()`.
2. Create a Promise that rejects with an error and handle it with `.catch()`.
3. Create a Promise that resolves after 2 seconds using `setTimeout`.
4. Use `Promise.resolve()` to immediately resolve a value.
5. Use `Promise.reject()` to immediately reject with an error.
6. Convert a callback-based function (e.g., `setTimeout`) into a Promise.
7. Write an `async` function that returns a string and log it using `await`.
8. Write an `async` function that throws an error and handle it using `try/catch`.
9. Chain multiple `.then()` calls on a Promise to perform sequential operations.
10. Handle errors in a Promise chain using `.catch()`.
11. Use `finally()` to execute code after a Promise resolves or rejects.
12. Convert a simple synchronous function to an `async` function and test it.
13. Use `await` to pause execution until a Promise resolves.
14. Create a Promise that resolves or rejects based on a random number condition.
15. Write an `async` function to fetch a simple resource using `fetch` (mock URL).

2️⃣ Intermediate Level (16–35)

Focus: chaining Promises, multiple async calls, error handling, and parallel execution.

16. Chain multiple Promises to calculate a series of operations sequentially.
17. Use `Promise.all()` to run multiple Promises in parallel and log results.

18. Use `Promise.allSettled()` to get the status of multiple Promises.
19. Use `Promise.race()` to get the result of the first settled Promise.
20. Write an `async` function to fetch data from multiple APIs sequentially using `await`.
21. Write an `async` function to fetch data from multiple APIs in parallel using `Promise.all()`.
22. Write a function that retries a Promise 3 times if it fails.
23. Convert a nested callback function (callback hell) into Promises.
24. Use `.then()` and `.catch()` to handle success and error separately in a Promise chain.
25. Implement a delay function using a Promise and `setTimeout`.
26. Use `async/await` to read multiple JSON files sequentially.
27. Use `async/await` with `try/catch` to handle errors from multiple API calls.
28. Write a Promise that resolves after a random delay and use it in `Promise.all()`.
29. Chain Promises to fetch a user, then fetch posts of that user, then comments of the first post.
30. Write a function to convert a Node.js callback-style function (error-first) into a Promise.
31. Use `async/await` to fetch data, manipulate it, and log the result.
32. Write an `async` function that waits for 2 different Promises to complete and returns the sum.
33. Implement a Promise that resolves only if a number is even, otherwise rejects.
34. Use `Promise.all()` with mixed resolved and rejected Promises and handle results.
35. Implement a function that wraps a fetch API call with timeout handling using Promise.race().

3️⃣ Advanced / Higher Level (36–50)

Focus: complex async patterns, real-world use cases, and interview-level challenges.

36. Implement a function that executes multiple async operations sequentially with delays.
37. Implement a function that fetches multiple URLs concurrently and returns the fastest response.
38. Use `async/await` to handle paginated API results.
39. Write a function that retries a fetch request with exponential backoff.
40. Implement a queue system for async tasks using Promises.
41. Create a wrapper function to handle async errors globally using `try/catch`.
42. Implement a cache system that returns cached Promise results if available.
43. Use `Promise.any()` to get the first successful result among multiple async calls.
44. Implement a timeout wrapper for any async function using Promise.race().
45. Use `async/await` to fetch data, filter results, and return only specific items.
46. Implement a concurrent limit for async requests (e.g., only 3 at a time).
47. Write a function to handle dependent async tasks in order using `async/await`.
48. Implement a polling function that repeatedly calls an API until a condition is met.
49. Create a function that executes multiple async tasks in batches and collects results.
50. Implement a function that wraps a callback API into a Promise-based API and demonstrates error handling.

Implement Stack Data Structure in JavaScript (15 Questions)

1. Implement a stack using an array with `push` and `pop`.
2. Implement `peek` to see the top element without removing it.
3. Implement `isEmpty` to check if the stack is empty.
4. Implement `size` to get the number of elements.
5. Reverse a string using a stack.
6. Check for balanced parentheses using a stack.
7. Implement `min()` to return the minimum element in the stack in O(1).
8. Implement `max()` to return the maximum element in the stack in O(1).
9. Implement a stack using a linked list.
10. Implement a stack that supports `getMiddle()` in O(1).
11. Sort a stack using another stack.
12. Implement two stacks in a single array.
13. Evaluate a postfix expression using a stack.
14. Convert an infix expression to postfix using a stack.
15. Implement undo functionality using a stack.

Queue Data Structure in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: implementing a simple queue using an array.

1. Implement a basic queue using an array with `enqueue` and `dequeue` methods.
2. Implement `peek()` to see the front element without removing it.
3. Implement `isEmpty()` to check if the queue is empty.
4. Implement `size()` to return the number of elements in the queue.
5. Print all elements of a queue from front to rear.

2️⃣ Intermediate Level (6–10)

Focus: queue operations and variations.

6. Implement a circular queue using an array.
7. Implement a priority queue where elements are dequeued based on priority.
8. Implement a queue using **two stacks**.
9. Reverse a queue using a stack.
10. Implement a double-ended queue (deque) with enqueue and dequeue from both ends.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic problems and interview challenges using queue.

11. Implement a sliding window maximum problem using a queue.
12. Implement a queue that tracks the maximum element in O(1) time.
13. Implement a queue to generate binary numbers from 1 to n.
14. Implement a round-robin scheduler using a queue.
15. Use a queue to implement BFS (Breadth-First Search) traversal in a graph.

List Data Structure in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: implementing and manipulating lists (arrays) in JS.

1. Implement a list using an array with `add(element)` and `remove(element)` methods.
2. Implement `size()` to return the number of elements in the list.
3. Implement `isEmpty()` to check if the list is empty.
4. Implement `contains(element)` to check if an element exists in the list.
5. Print all elements of the list.

2️⃣ Intermediate Level (6–10)

Focus: more advanced list operations and manipulations.

6. Insert an element at a specific index in the list.
7. Remove an element at a specific index in the list.
8. Find the index of a given element in the list.
9. Reverse the list without using the built-in `reverse()` method.
10. Merge two lists into one sorted list.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic problems and interview-level challenges using lists.

11. Remove duplicate elements from a list.
12. Rotate the list `k` times to the right.
13. Find the largest sum of contiguous subarray in the list (Kadane's algorithm).
14. Implement a function to find all sublists of a given list.
15. Implement a list that supports `getMax()` and `getMin()` in O(1) time.

Linked List Data Structure in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: implementing a singly linked list and basic operations.

1. Implement a **singly linked list** with `Node` and `LinkedList` classes.
2. Implement `insertAtEnd(value)` to add a node at the end.
3. Implement `insertAtStart(value)` to add a node at the beginning.
4. Implement `deleteValue(value)` to remove a node by value.
5. Implement `printList()` to display all elements of the linked list.

2️⃣ Intermediate Level (6–10)

Focus: intermediate operations and manipulations.

6. Implement `insertAtIndex(index, value)` to insert a node at a specific position.
7. Implement `deleteAtIndex(index)` to delete a node at a specific position.
8. Implement `find(value)` to search for a node by value.
9. Implement `getLength()` to return the number of nodes in the list.
10. Reverse a linked list iteratively.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges using linked lists.

11. Reverse a linked list recursively.
12. Detect a loop in a linked list (Floyd's Cycle-Finding Algorithm).
13. Find the middle node of a linked list in one pass.
14. Merge two sorted linked lists into one sorted linked list.
15. Remove the Nth node from the end of a linked list.

Stack Using Linked List in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: implementing a simple stack using a linked list.

1. Implement a `Node` class and a `Stack` class using a linked list.
2. Implement `push(value)` to add an element to the top of the stack.

3. Implement `pop()` to remove and return the top element of the stack.
4. Implement `peek()` to return the top element without removing it.
5. Implement `isEmpty()` to check if the stack is empty.

2️⃣ Intermediate Level (6–10)

Focus: stack operations and additional functionality.

6. Implement `size()` to return the number of elements in the stack.
7. Implement `printStack()` to display all elements from top to bottom.
8. Implement a stack that keeps track of the **minimum element**.
9. Implement a stack that keeps track of the **maximum element**.
10. Reverse a string using a stack implemented with a linked list.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic problems and interview-level challenges using stack.

11. Evaluate a postfix expression using a stack implemented with a linked list.
12. Convert an infix expression to postfix using a stack.
13. Check for **balanced parentheses** in an expression using a stack.
14. Implement **two stacks in a single linked list**.
15. Sort a stack using only stack operations (no additional arrays).

Queue Using Linked List in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: implementing a simple queue using a linked list.

1. Implement a `Node` class and a `Queue` class using a linked list.
2. Implement `enqueue(value)` to add an element at the rear of the queue.
3. Implement `dequeue()` to remove and return the front element of the queue.
4. Implement `peek()` to return the front element without removing it.
5. Implement `isEmpty()` to check if the queue is empty.

2️⃣ Intermediate Level (6–10)

Focus: queue operations and additional functionality.

6. Implement `size()` to return the number of elements in the queue.
7. Implement `printQueue()` to display all elements from front to rear.
8. Implement a **priority queue** using a linked list.
9. Implement a **circular queue** using a linked list.
10. Reverse the queue using a stack implemented with a linked list.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic problems and interview-level challenges using queue.

11. Implement **BFS (Breadth-First Search)** traversal of a graph using a queue.
12. Implement **first non-repeating character** in a stream using a queue.
13. Implement **sliding window maximum** using a queue.
14. Implement **LRU Cache** using a queue (with linked list + hash map).
15. Implement **round-robin scheduling** using a queue for processes.

Doubly Linked List Implementation in JavaScript – 15 Questions

---

1️⃣ Basic Level (1–5)

Focus: creating a basic doubly linked list and implementing core operations.

1. Implement a `Node` class for DLL with `value`, `next`, and `prev` pointers.
2. Implement a `DoublyLinkedList` class with `insertAtEnd(value)` method.
3. Implement `insertAtStart(value)` to add a node at the beginning.
4. Implement `printForward()` to display elements from head to tail.
5. Implement `printBackward()` to display elements from tail to head.

2️⃣ Intermediate Level (6–10)

Focus: insertion, deletion, and utility methods.

6. Implement `insertAtIndex(index, value)` to insert at a specific position.
7. Implement `deleteAtIndex(index)` to delete a node at a specific position.
8. Implement `deleteValue(value)` to remove the first node with a given value.
9. Implement `find(value)` to search for a node with a specific value.
10. Implement `getLength()` to return the number of nodes in the list.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level operations.

11. Reverse the doubly linked list iteratively.
12. Reverse the doubly linked list recursively.
13. Remove all duplicate values from the DLL.
14. Merge two sorted doubly linked lists into one sorted DLL.
15. Implement a **stack** or **queue** using a doubly linked list.

Deque Data Structure Implementation in JavaScript – 15 Questions

---

1️⃣ Basic Level (1–5)

Focus: implementing a simple deque using an array.

1. Implement a `Deque` class with `addFront(value)` and `addRear(value)` methods.
2. Implement `removeFront()` to remove an element from the front.
3. Implement `removeRear()` to remove an element from the rear.
4. Implement `peekFront()` to return the front element without removing it.
5. Implement `peekRear()` to return the rear element without removing it.

2️⃣ Intermediate Level (6–10)

Focus: deque operations and utility methods.

6. Implement `isEmpty()` to check if the deque is empty.
7. Implement `size()` to return the number of elements in the deque.
8. Implement `printDeque()` to display all elements from front to rear.
9. Implement a deque using a **doubly linked list**.
10. Reverse the deque using stack operations.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level applications.

11. Implement a **sliding window maximum** using a deque.

12. Implement **palindrome checker** using a deque.

13. Implement a deque-based **LRU cache**.

14. Use a deque to implement a **job scheduling system** (priority and order-based).

15. Implement a **circular deque** where adding/removing elements wraps around.

Deque Using Doubly Linked List in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: creating a deque using doubly linked list and basic operations.

1. Implement a `Node` class with `value`, `next`, and `prev` pointers for the deque.

2. Implement a `Deque` class using a doubly linked list.

3. Implement `addFront(value)` to insert at the front of the deque.

4. Implement `addRear(value)` to insert at the rear of the deque.

5. Implement `printDequeForward()` to display elements from front to rear.

2️⃣ Intermediate Level (6–10)

Focus: removal, access, and utility operations.

6. Implement `removeFront()` to remove an element from the front.

7. Implement `removeRear()` to remove an element from the rear.

8. Implement `peekFront()` to return the front element without removing it.

9. Implement `peekRear()` to return the rear element without removing it.

10. Implement `isEmpty()` and `size()` methods for the deque.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level applications.

11. Implement **reverseDeque()** to reverse the order of elements.

12. Implement a **palindrome checker** using a deque with DLL.

13. Implement a **sliding window maximum** using a deque with DLL.

14. Implement **LRU Cache** using deque with doubly linked list.

15. Implement a **circular deque** using a doubly linked list where adding/removing elements wraps around.

Circular Linked List Implementation in JavaScript – 15 Questions

---

1️⃣ Basic Level (1–5)

Focus: creating a basic circular linked list and core operations.

1. Implement a `Node` class with `value` and `next` pointers for a circular linked list.

2. Implement a `CircularLinkedList` class with a `head` pointer.

3. Implement `insertAtEnd(value)` to add a node at the end of the CLL.

4. Implement `insertAtStart(value)` to add a node at the beginning of the CLL.

5. Implement `printList()` to display all nodes, ensuring it loops back to the head.

2️⃣ Intermediate Level (6–10)

Focus: insertion, deletion, and traversal operations.

6. Implement `insertAtIndex(index, value)` to insert a node at a specific position.

7. Implement `deleteAtIndex(index)` to delete a node at a specific position.

8. Implement `deleteValue(value)` to remove the first node with the given value.

9. Implement `find(value)` to search for a node with a specific value.

10. Implement `getLength()` to return the number of nodes in the circular linked list.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level applications.

11. Detect a loop in a circular linked list (confirm circular nature).

12. Split a circular linked list into two equal halves.
13. Implement a **Josephus Problem** using a circular linked list.
14. Reverse a circular linked list iteratively.
15. Merge two circular linked lists into one circular linked list.

Circular Doubly Linked List Implementation in JavaScript – 15 Questions

---

1️⃣ Basic Level (1–5)

Focus: creating a basic circular doubly linked list and core operations.

1. Implement a `Node` class with `value`, `next`, and `prev` pointers.
2. Implement a `CircularDoublyLinkedList` class with a `head` pointer.
3. Implement `insertAtEnd(value)` to add a node at the end of the CDLL.
4. Implement `insertAtStart(value)` to add a node at the beginning.
5. Implement `printForward()` to display all nodes from head to tail (loop back to head).

2️⃣Intermediate Level (6–10)

Focus: insertion, deletion, and traversal operations.

6. Implement `printBackward()` to display all nodes from tail to head.
7. Implement `insertAtIndex(index, value)` to insert a node at a specific position.
8. Implement `deleteAtIndex(index)` to delete a node at a specific position.
9. Implement `deleteValue(value)` to remove the first node with the given value.
10. Implement `getLength()` to return the number of nodes in the CDLL.

3️⃣Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level applications.

11. Reverse a circular doubly linked list iteratively.
12. Reverse a circular doubly linked list recursively.

Priority Queue Implementation in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: implementing a simple priority queue and core operations.

1. Implement a `Node` class with `value` and `priority`.
2. Implement a `PriorityQueue` class using an array.
3. Implement `enqueue(value, priority)` to add elements based on priority.
4. Implement `dequeue()` to remove and return the element with the highest priority.
5. Implement `peek()` to view the element with the highest priority without removing it.

2️⃣ Intermediate Level (6–10)

Focus: different implementations and utility methods.

6. Implement `isEmpty()` to check if the queue is empty.

7. Implement `size()` to return the number of elements in the priority queue.

8. Implement `printQueue()` to display all elements sorted by priority.

9. Implement a **priority queue using a linked list**.

10. Implement a **priority queue using a binary heap** (min-heap or max-heap).

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level applications.

13. Implement a **dynamic priority change** method to update the priority of an existing element.

14. Implement **merge two priority queues** into one.

15. Implement a **task scheduler** using a priority queue.

16. Implement **Dijkstra’s algorithm** using a priority queue.

17. Implement an **event-driven simulation** using a priority queue.

18. Split a circular doubly linked list into two equal halves.

19. Merge two circular doubly linked lists into one circular doubly linked list.

20. Implement a **deque** using a circular doubly linked list.

Tree Data Structure in JavaScript – 15 Questions

1️⃣ Basic Level (1–5)

Focus: creating a tree and implementing core operations.

1. Implement a `Node` class with `value`, `left`, and `right` pointers.
2. Implement a `BinaryTree` class with a `root` node.
3. Implement `insert(value)` to add a node to a binary tree (not necessarily BST).
4. Implement `printPreOrder()` to traverse the tree in pre-order (root → left → right).
5. Implement `printInOrder()` to traverse the tree in in-order (left → root → right).

2️⃣ Intermediate Level (6–10)

Focus: traversal methods and basic operations.

6. Implement `printPostOrder()` to traverse the tree in post-order (left → right → root).
7. Implement `levelOrderTraversal()` (BFS) using a queue.
8. Find the **height** of a binary tree.
9. Count the **total number of nodes** in a binary tree.
10. Count the **number of leaf nodes** in a binary tree.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level operations.

11. Check if a binary tree is **balanced**.
12. Find the **maximum value** in a binary tree.
13. Find the **minimum value** in a binary tree.
14. Find the **diameter** of a binary tree (longest path between two nodes).
15. Implement **lowest common ancestor (LCA)** for two nodes in a binary tree.

Heap Data Structure in JavaScript – 15 Questions

1️⃣Basic Level (1–5)

Focus: implementing a simple heap and basic operations.

Implement a MinHeap class using an array.

Implement insert(value) to add an element to the heap while maintaining heap property.

Implement extractMin() to remove and return the minimum element from a min-heap.

Implement peek() to return the minimum element without removing it.

Implement isEmpty() to check if the heap is empty.

2️⃣Intermediate Level (6–10)

Focus: max-heap, heap operations, and utility methods.

Implement a MaxHeap class.

Implement insert(value) for a max-heap.

Implement extractMax() to remove and return the maximum element from a max-heap.

Implement heapify(array) to build a heap from an unsorted array.

Implement size() to return the number of elements in the heap.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic problems and interview-level applications.

Implement heap sort using min-heap or max-heap.

Find the kth largest element in an array using a heap.

Find the kth smallest element in an array using a heap.

Merge k sorted arrays using a min-heap.

Implement a priority queue using a heap (enqueue/dequeue based on priority).

Graph Data Structure in JavaScript – 15 Questions

1️⃣Basic Level (1–5)

Focus: implementing a graph and basic operations.

1. Implement a `Graph` class using an **adjacency list**.

2. Implement a `Graph` class using an **adjacency matrix**.

3. Implement `addVertex(vertex)` to add a vertex to the graph.

4. Implement `addEdge(vertex1, vertex2)` to add an edge between two vertices (undirected).

5. Implement `printGraph()` to display all vertices and their connections.

2️⃣Intermediate Level (6–10)

Focus: traversal methods and basic operations.

6. Implement **DFS (Depth-First Search)** traversal (recursive).
7. Implement **DFS traversal** (iterative using a stack).
8. Implement **BFS (Breadth-First Search)** traversal using a queue.
9. Check if there is a **path between two vertices** using BFS or DFS.
10. Detect **cycles in an undirected graph**.

3️⃣ Advanced / Higher Level (11–15)

Focus: algorithmic challenges and interview-level problems.

11. Detect **cycles in a directed graph** using DFS.
12. Find **connected components** in an undirected graph.
13. Implement **topological sort** for a directed acyclic graph (DAG).
14. Implement **Dijkstra's shortest path algorithm**.
15. Implement **Prim's or Kruskal's algorithm** to find the Minimum Spanning Tree (MST).

AVL Tree Implementation in JavaScript – 15 Questions

1️⃣Basic Level (1–5)

Focus: understanding AVL Tree node and basic insertion.

1. Implement a `Node` class with `value`, `left`, `right`, and `height` properties.
2. Implement an `AVLTree` class with a `root` property.
3. Implement `insert(value)` to add a node while maintaining the AVL balance.
4. Implement **right rotation** for balancing the tree.
5. Implement **left rotation** for balancing the tree.

2️⃣Intermediate Level (6–10)

Focus: balancing and traversal operations.

6. Implement `getHeight(node)` to return the height of a node.
7. Implement `getBalanceFactor(node)` to calculate the balance factor.
8. Implement **in-order traversal** of the AVL Tree.

9. Implement **pre-order traversal** of the AVL Tree.
10. Implement **post-order traversal** of the AVL Tree.

3️⃣ Advanced / Higher Level (11–15)

Focus: deletion, search, and interview-level challenges.

11. Implement `delete(value)` to remove a node while maintaining AVL balance.
12. Implement `search(value)` to check if a value exists in the AVL Tree.
13. Find the **minimum value** node in the AVL Tree.
14. Find the **maximum value** node in the AVL Tree.
15. Find the **height of the AVL Tree** and verify its balance property for all nodes.