1. **What is JavaScript?**
    - JavaScript is a high-level, interpreted programming language primarily used for creating interactive web applications. It follows ECMAScript specifications.
2. **What are the different data types in JavaScript?**
    - Primitive types: String, Number, BigInt, Boolean, Undefined, Null, Symbol.
    - Non-primitive type: Object.
3. **What is the difference between var, let, and const?**
    - var: Function-scoped, hoisted, can be re-declared.
    - let: Block-scoped, not hoisted to usable state (TDZ).
    - const: Block-scoped, cannot be reassigned.
4. **What is hoisting in JavaScript?**
    - Hoisting is JavaScript's default behavior of moving declarations (not initializations) to the top of the scope.
5. **Explain null vs undefined.**
    - null: Represents the intentional absence of any value.
    - undefined: Represents an uninitialized variable or missing property.
6. **What are closures in JavaScript?**
    - A closure is a function that retains access to its outer scope, even after the outer function has executed.

```
function outer() {
  let x = 10;
  return function inner() {
    console.log(x);
  };
}
const closureFunc = outer();
closureFunc(); // 10
```

7. **What is lexical scope in JavaScript?**
    - Lexical scope means variables are accessible based on their location in the source code. Functions have access to variables defined in their outer scope.

```
function outer() {
  let a = 5;

  function inner() {

    console.log(a); // Accessible

  }

  inner();

}
```

8. **What is the Temporal Dead Zone (TDZ)?**
   - The TDZ is the phase where a variable declared with let or const is not accessible before its initialization.

```
console.log(x); // ReferenceError

let x = 10;
```

9. **What is the difference between == and ===?**
   - ==: Compares values, performs type coercion.
   - ===: Compares values and types (strict equality).
10. **What is the this keyword?**
    - this refers to the object that is executing the current function. In the global scope, it refers to the window object in browsers.
11. **What is an IIFE?**
    - An Immediately Invoked Function Expression is a function executed immediately after its definition.

```
(function() {

  console.log('IIFE!');

})();
```

12. **What is event delegation?**
    - A technique where a single event listener is added to a parent element to handle events for its child elements.
13. **What is the difference between call(), apply(), and bind()?**
    - call(): Calls a function with a specified this and arguments.
    - apply(): Same as call(), but arguments are passed as an array.
    - bind(): Returns a new function with this bound.
14. **What are Promises?**
    - Promises represent a value that may be available now, in the future, or never.

```
const promise = new Promise((resolve, reject) => {
  resolve('Success');
});
promise.then(console.log); // Success
```

15. **What is async/await?**
    - Syntactic sugar for Promises, allowing asynchronous code to be written in a synchronous style.
16. **What is the DOM?**
    - The DOM is a programming interface for HTML and XML documents. It represents the page so programs can manipulate it.
17. **How to select elements in the DOM?**
    - document.querySelector, document.getElementById, document.getElementsByClassName.
18. **What is the difference between window and document?**
    - window: Represents the browser's window.
    - document: Represents the DOM.
19. **What are event bubbling and event capturing?**
    - Event bubbling: Events propagate from child to parent.
    - Event capturing: Events propagate from parent to child.
20. **What is preventDefault()?**
    - Prevents the default behavior of an event, e.g., stopping a form submission.
21. **What are arrow functions?**
    - Shorter syntax for functions, no this binding.

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

22. **What are template literals?**
    - String literals with embedded expressions using backticks.

```
const name = 'John';
console.log(`Hello, ${name}!`);
```

23. **What is destructuring?**
    - A way to unpack values from arrays or properties from objects.

```
const { name, age } = { name: 'John', age: 30 };
```

24. **What are default parameters?**

- Function parameters with default values.

```
function greet(name = 'Guest') {
  console.log(`Hello, ${name}`);
}
```

25. **What is the spread operator?**
    - Expands an array or object into individual elements.

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

Error Handling

26. **How does try...catch work?**
    - Used to handle errors in a block of code.

```
try {
  throw new Error('Oops!');
} catch (error) {
  console.log(error.message);
}
```

27. **What is the difference between throw and return?**
    - throw: Used to signal an error.
    - return: Exits a function with a value.
28. **What is the finally block?**
    - Executes code after try and catch, regardless of outcome.
29. **What is the event loop?**
    - A mechanism that handles execution of multiple chunks of code (callbacks, Promises).
30. **What is a callback?**
    - A function passed as an argument to another function.
31. **What are memory leaks?**
    - Unused memory that is not released.
32. **What is garbage collection?**
    - Automatic memory management to free unused memory.

33. What is a higher-order function?

- A **higher-order function** is a function that either:
    1. Takes one or more functions as arguments, or
    2. Returns another function as its result.
- Example:

```
function higherOrderFunction(fn) {
  return function (x) {
    return fn(x) * 2;
  };
}

const double = (x) => x * 2;
const result = higherOrderFunction(double);
console.log(result(5)); // 20
```

34. What is currying?

- **Currying** is a technique of transforming a function with multiple arguments into a sequence of functions, each taking a single argument.
- Example:

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

const addFive = add(5);
console.log(addFive(3)); // 8
```

35. What are generators?

- **Generators** are functions that can pause and resume their execution using the yield keyword. They are defined using the function* syntax.
- Example:

```
function* generatorExample() {
  yield 1;
  yield 2;
  yield 3;
}

const gen = generatorExample();
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
console.log(gen.next().value); // 3
```

36. What is a pure function?

- A **pure function** is a function that:
    1. Always produces the same output for the same input.
    2. Has no side effects (does not modify external state).
- Example:

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

console.log(pureAdd(2, 3)); // 5
```

37. What is a debounce function?

- A **debounce** function ensures that a particular function is executed only after a specified delay, and only if no new events occur during that delay.
- Example:

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

const debouncedLog = debounce(() => console.log('Debounced!'), 1000);
window.addEventListener('resize', debouncedLog);
```

38. What is throttling?

- **Throttling** ensures a function is executed at most once in a specified interval, regardless of how many times the event is triggered.
- Example:

```
function throttle(fn, interval) {
  let lastCall = 0;
  return function (...args) {
    const now = Date.now();
    if (now - lastCall >= interval) {
      lastCall = now;
      fn(...args);
    }
  };
}

const throttledLog = throttle(() => console.log('Throttled!'), 1000);
window.addEventListener('scroll', throttledLog);
```

39. What are modules in JavaScript?

- **Modules** are reusable pieces of code that can be exported and imported between files. They allow better code organization and dependency management.
- Example:

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

```
// main.js

import { add } from './math.js';

console.log(add(2, 3)); // 5
```

40. What is the typeof operator?

- The typeof operator returns the type of a variable or expression.
- Example:

```
console.log(typeof 42); // 'number'

console.log(typeof 'hello'); // 'string'

console.log(typeof undefined); // 'undefined'

console.log(typeof null); // 'object' (legacy bug)

console.log(typeof {}); // 'object'

console.log(typeof function () {}); // 'function'
```

41. How do you remove duplicate values from an array in JavaScript?

Answer: You can remove duplicates using a Set, which automatically removes duplicate values. You can also use the filter() method or reduce() for more customized solutions.

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

const uniqueArr = [...new Set(arr)]; // [1, 2, 3]
```

42. What is the difference between map() and forEach() in JavaScript?

Answer:

- map(): Creates a new array with the results of calling a provided function on every element in the array. It returns a new array.

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

const doubled = arr.map(x => x * 2); // [2, 4, 6]
```

- forEach(): Executes a provided function once for each array element but does not return anything (returns undefined).

```
arr.forEach(x => console.log(x)); // 1, 2, 3
```

43. How do you find the index of an element in an array in JavaScript?

Answer: You can use the indexOf() method to find the index of an element.

```
const arr = [10, 20, 30];

const index = arr.indexOf(20); // 1
```

If the element is not found, indexOf() returns -1.

44. How do you merge two arrays in JavaScript?

Answer: You can use the concat() method or the spread operator (...).

```
const arr1 = [1, 2];

const arr2 = [3, 4];

const merged = arr1.concat(arr2); // [1, 2, 3, 4]
```

Or using the spread operator:

```
const merged = [...arr1, ...arr2]; // [1, 2, 3, 4]
```

45. How do you flatten a nested array in JavaScript?

Answer: You can use the flat() method, which flattens an array up to a specified depth (default is 1).

```
const arr = [1, [2, 3], [4, [5, 6]]];

const flattened = arr.flat(2); // [1, 2, 3, 4, 5, 6]
```

46. How do you check if an array contains a specific value?

Answer: You can use the includes() method to check if an array contains a specific value.

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

const contains = arr.includes(2); // true
```

47. How do you sort an array in JavaScript?

Answer: You can use the sort() method. By default, it sorts elements as strings.

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

arr.sort(); // [1, 2, 3, 4]
```

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

Answer:

- slice(): Returns a shallow copy of a portion of an array without modifying the original array.

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

const sliced = arr.slice(1, 3); // [2, 3]
```

- splice(): Modifies the original array by removing, replacing, or adding elements.

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

arr.splice(1, 2, 5); // [1, 5, 4]
```

49. How do you reverse an array in JavaScript?

Answer: You can use the reverse() method to reverse the order of elements in an array.

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

arr.reverse(); // [3, 2, 1]
```

50. How do you find the maximum or minimum value in an array?

Answer: You can use Math.max() and Math.min() along with the spread operator.

```
const arr = [1, 5, 3, 9];

const max = Math.max(...arr); // 9

const min = Math.min(...arr); // 1
```

51. How do you reverse a string in JavaScript?

Answer: You can reverse a string by converting it to an array, using the reverse() method, and then joining it back to a string.

```
const str = 'hello';

const reversed = str.split('').reverse().join(''); // 'olleh'
```

52. How do you check if a string is a palindrome in JavaScript?

Answer: You can compare the string with its reverse.

```
function isPalindrome(str) {

  return str === str.split('').reverse().join('');
}

console.log(isPalindrome('racecar')); // true
```

53. How do you convert a string to an array in JavaScript?

Answer: You can use the split() method to convert a string into an array of characters.

```
const str = 'hello';
const arr = str.split(''); // ['h', 'e', 'l', 'l', 'o']
```

54. How do you convert an array to a string in JavaScript?

Answer: You can use the join() method to convert an array into a string.

```
const arr = ['h', 'e', 'l', 'l', 'o'];
const str = arr.join(''); // 'hello'
```

55. How do you find the index of a character in a string?

Answer: You can use the indexOf() method to find the index of a character.

```
const str = 'hello';
const index = str.indexOf('e'); // 1
```

If the character is not found, it returns -1.

56. How do you check if a string contains a substring in JavaScript?

Answer: You can use the includes() method to check if a string contains a substring.

```
const str = 'hello world';
const contains = str.includes('world'); // true
```

57. How do you replace all occurrences of a substring in a string?

Answer: You can use the replace() method with a global regular expression.

```
const str = 'hello world world';
const newStr = str.replace(/world/g, 'universe'); // 'hello universe universe'
```

58. How do you trim whitespace from both ends of a string?

Answer: You can use the trim() method to remove whitespace from both ends of a string.

```
const str = '  hello  ';

const trimmed = str.trim(); // 'hello'
```

59. How do you convert a string to lowercase or uppercase?

Answer: You can use the toLowerCase() and toUpperCase() methods.

```
const str = 'Hello World';

const lower = str.toLowerCase(); // 'hello world'

const upper = str.toUpperCase(); // 'HELLO WORLD'
```

60. How do you extract a substring from a string in JavaScript?

Answer: You can use the substring() or slice() methods.

```
const str = 'hello world';

const substr = str.substring(0, 5); // 'hello'
```

Or using slice():

```
const sliceStr = str.slice(0, 5); // 'hello'
```

70. What are Client side and Server side?

Answer: Client-side refers to operations that occur on the user's device (browser or app), such as rendering UI, form validation, and interactive elements using technologies like HTML, CSS, and JavaScript. It is fast but less secure.

Server-side refers to operations that occur on the web server, such as database queries, authentication, and dynamic content generation using languages like PHP, Node.js, and Python. It is more secure but can be slower due to server communication.

In short:

- **Client-side**: User's device, fast, less secure.
- **Server-side**: Web server, secure, slower.

71. What is JSON?

Answer: **JSON (JavaScript Object Notation)** is a lightweight, text-based format for representing structured data in key-value pairs. It is easy to read and write for humans and easy for machines to parse and generate. JSON is commonly used for data exchange between a server and client in web applications.

Example:

```
{
  "name": "John",
  "age": 30,
  "skills": ["JavaScript", "Python"]
}
```

72. what is type coercion in JS?

Answer: Type coercion in JavaScript refers to the automatic conversion of one data type to another when performing operations, like when adding a string to a number. JavaScript implicitly converts the types to make the operation work.

Example:

```
let result = '5' + 10; // '510' (string + number -> number is coerced to string)
let sum = '5' - 2;     // 3 (string is coerced to number)
```

In the first example, the number 10 is coerced into a string, and in the second, the string '5' is coerced into a number.

73. What is DOM? What is the difference between HTML and DOM? what is the difference between innerHTML vs textContent?

Answer:DOM (Document Object Model):

The **DOM** is a programming interface for web documents. It represents the structure of an HTML document as a tree of objects, where each object corresponds to a part of the page (like an element, attribute, or text). It allows programming languages (like JavaScript) to interact with and manipulate the content, structure, and style of web pages dynamically.

Difference Between HTML and DOM:

- **HTML**: HTML is a static markup language used to structure content on the web. It defines the elements, attributes, and layout of a web page.
- **DOM**: The DOM is a dynamic representation of the HTML document. It is an in-memory object model created by the browser when a page is loaded. The DOM allows scripts (like JavaScript) to manipulate the page's content and structure after the page has been rendered.

In short, HTML is the content structure, and the DOM is the dynamic model that JavaScript interacts with.

Difference Between innerHTML and textContent:

- **innerHTML**:
    - Represents the HTML content of an element, including HTML tags.

- Allows you to get or set HTML content, so any HTML tags inside the element are treated as HTML.
- Example:

document.getElementById('example').innerHTML = '<p>Hello</p>';

This would add a paragraph with the text "Hello".

- **textContent**:
  - Represents only the text content of an element, without any HTML tags.
  - Used for getting or setting plain text.
  - Example:

document.getElementById('example').textContent = 'Hello';

This would set the text to "Hello", ignoring any HTML tags.

74. What is Map object in Js?

Answer: The **Map** object in JavaScript is a collection of key-value pairs where both keys and values can be any data type. It preserves the order of insertion and is iterable.

Key Methods:

- **set(key, value)**: Adds or updates an entry.
- **get(key)**: Retrieves the value for a key.
- **has(key)**: Checks if a key exists.
- **delete(key)**: Removes an entry.
- **clear()**: Clears all entries.
- **size**: Returns the number of entries.

Example:

let map = new Map();

map.set('name', 'John');

console.log(map.get('name')); // 'John'

console.log(map.size); // 1

map.delete('name');

console.log(map.size); // 0

Differences from Objects:

- **Keys**: Maps can have any data type as keys, while objects can only use strings or symbols.
- **Order**: Maps preserve insertion order, objects do not guarantee it.

75.Explain Event Bubbling and Event Cpaturing.

Answer: **Event Bubbling** and **Event Capturing** are two phases of event propagation in the DOM when an event is triggered on an element.

Event Bubbling:

- The event starts from the target element and bubbles up to the root (document).
- It is the default behavior in most cases.
- Example: If you click on a button inside a div, the click event first triggers on the button, then on the div, and then on the document.

Event Capturing:

- The event starts from the root (document) and is captured as it moves down to the target element.
- It is less commonly used and can be enabled by specifying {capture: true} in event listeners.
- Example: The event is captured by the document first, then the div, and finally the button.