CDAC 2024

WPT

Interview Question set2

==

1. What is JavaScript, and what are its primary uses?

JavaScript is a high-level, interpreted programming language primarily used for creating dynamic and interactive content on websites. Its primary uses include web development (both client-side and server-side), mobile app development, and game development.

2. Differences between let, const, and var

- **let**: Block-scoped variable declaration, can be updated but not redeclared within the same scope.
- **const**: Also block-scoped, but the variable cannot be reassigned after its initial assignment (though objects it references can be mutated).
- **var**: Function-scoped or globally scoped, can be redeclared and updated. It is subject to hoisting.

3. What is hoisting in JavaScript?

Hoisting is JavaScript's behavior of moving variable and function declarations to the top of their containing scope during the compile phase. This means you can use functions and variables before they are declared in the code.

4. What are closures?

Closures are functions that capture the lexical scope in which they were created. They can access variables from their enclosing scope even after that scope has finished executing.

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

5. What is the event loop?

The event loop is a mechanism that allows JavaScript to perform non-blocking operations by using a single-threaded model. It manages asynchronous operations by placing callback functions in a queue to be executed after the current execution stack is clear.

6. What is the difference between == and ===?

- ==: Compares values for equality after performing type coercion.
- ===: Compares both value and type, ensuring no type conversion is performed.

7. How can you check the type of a variable?

You can use the `typeof` operator:

```
console.log(typeof variable);
```

8. What is the use of the this keyword?

`this` refers to the context in which a function is called. It can refer to different objects depending on how the function is invoked (e.g., as a method of an object, in the global context, or in a constructor).

9. Function declarations vs. function expressions

- **Function declaration**: Defined with the `function` keyword and hoisted, allowing calls before definition.

  ```
  function myFunction() {}
  ```

- **Function expression**: Created as part of an expression and not hoisted.

  ```
  const myFunction = function() {};
  ```

10. How does setTimeout work?

`setTimeout` schedules a function to be executed after a specified delay (in milliseconds). It does not block the execution of code.

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

11. What is asynchronous JavaScript?

Asynchronous JavaScript allows operations to run in the background without blocking the main thread. This is important for tasks like network requests and timers, ensuring a smooth user experience.

12. What is a callback function?

A callback function is a function passed as an argument to another function, executed after a certain task is completed. **Callback hell** refers to the difficulty of managing multiple nested callbacks. It can be avoided using promises or async/await.

13. Can you explain promises?

Promises represent the eventual completion (or failure) of an asynchronous operation and its resulting value. You can use `.then()` to handle fulfilled promises and `.catch()` for rejections.

```
let myPromise = new Promise((resolve, reject) => {
    // async operation
    resolve("Success!");
});
myPromise.then(result => console.log(result)).catch(err =>
console.error(err));
```

14. What is async/await?

`async/await` is a syntactic sugar over promises, making asynchronous code look synchronous. You define an asynchronous function with `async` and use `await` to pause execution until a promise is resolved.

```
async function fetchData() {
    const response = await fetch('url');
    const data = await response.json();
    console.log(data);
}
```

15. What is a higher-order function?

A higher-order function is a function that takes another function as an argument or returns a function as a result.

```
function higherOrderFunction(callback) {
    callback();
}
```

16. What are arrow functions?

Arrow functions are a concise syntax for writing function expressions. They do not have their own `this` context, which can be beneficial for certain callbacks.

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

17. What is a pure function?

A pure function is a function that, given the same input, will always return the same output without causing side effects (e.g., modifying external state).

18. What is prototypal inheritance?

Prototypal inheritance is a method by which objects can inherit properties and methods from other objects. JavaScript uses prototypes to allow for inheritance and reuse of properties.

19. How can you create an object?

You can create an object using object literals, constructors, or the `Object.create()` method.

```
const obj = { name: "Alice", age: 30 };
```

20. What is the purpose of the prototype property?

The prototype property allows you to add properties and methods to an object’s prototype, enabling inheritance. Properties on the prototype are shared across all instances of an object.

21. Explain object destructuring.

Object destructuring allows you to extract properties from objects into variables.

```
const person = { name: "Alice", age: 30 };
const { name, age } = person;
```

22. Shallow copy vs. deep copy

- **Shallow copy**: Copies the object’s top-level properties. Nested objects are shared.
- **Deep copy**: Creates a complete copy of an object, including all nested objects.

23. Value types vs. reference types

- **Value types**: Primitive data types (e.g., numbers, strings) that are copied by value.
- **Reference types**: Objects, arrays, and functions that are copied by reference.

24. What is lexical scope?

Lexical scope refers to the visibility of variables based on their physical location in the code. Inner functions have access to variables declared in their outer scopes.

25. What is an Immediately Invoked Function Expression (IIFE)?

An IIFE is a function that is executed immediately after it is defined. It helps to create a new scope.

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

26. How does the call stack work?

The call stack is a data structure that keeps track of function calls in JavaScript. When a function is invoked, it is pushed onto the stack, and when it returns, it is popped off.

27. What is function currying?

Currying is a technique of breaking down a function that takes multiple arguments into a series of functions that each take a single argument.

```
function multiply(a) {
    return function(b) {
        return a * b;
    };
}
const double = multiply(2);
console.log(double(5)); // 10
```

28. Difference between null and undefined

- **null**: An intentional absence of any object value.
- **undefined**: A variable that has been declared but not yet assigned a value.

29. What is the Document Object Model (DOM)?

The DOM is an interface that browsers implement to represent and interact with HTML and XML documents. It represents the structure of the document as a tree of objects.

30. How do you select elements in the DOM using JavaScript?

You can use methods like `document.getElementById()`, `document.querySelector()`, or `document.getElementsByClassName()` to select elements.

31. What is event delegation?

Event delegation is a technique where a single event listener is added to a parent element to manage events for multiple child elements. This improves performance and simplifies event management.

32. How can you create and remove elements in the DOM?

To create an element:

```
const newElement = document.createElement('div');
document.body.appendChild(newElement);
```

To remove an element:

```
const elementToRemove = document.getElementById('myElement');
elementToRemove.remove();
```

33. Purpose of addEventListener

`addEventListener` allows you to attach event handlers to elements without overwriting existing handlers. It is more flexible than inline event handlers.

34. How to prevent the default behavior of an event?

You can call `event.preventDefault()` inside the event handler.

```
element.addEventListener('click', (event) => {
    event.preventDefault();
});
```

35. Difference between innerHTML and textContent

- **innerHTML**: Allows you to get or set HTML content and parse it as HTML.
- **textContent**: Gets or sets the text content of an element without parsing HTML.

36. What is JSON, and how is it used in JavaScript?

JSON (JavaScript Object Notation) is a lightweight format for data interchange, derived from JavaScript object syntax. It's easy to read and write for humans and easy for machines to parse and generate. In JavaScript, JSON is often used for transmitting data between a server and a web application.

37. How do you parse JSON data in JavaScript?

You can parse JSON data using `JSON.parse()`, which converts a JSON string into a JavaScript object.

```
const jsonData = '{"name": "Alice", "age": 30}';
const obj = JSON.parse(jsonData);
```

38. How do you convert a JavaScript object into a JSON string?

You can convert a JavaScript object into a JSON string using `JSON.stringify()`.

```
const obj = { name: "Alice", age: 30 };
const jsonString = JSON.stringify(obj);
```

39. What are the benefits of using JSON over XML?

- **Simplicity**: JSON syntax is simpler and more compact than XML.
- **Data types**: JSON supports a wider variety of data types, including numbers and booleans, directly.
- **Parsing**: JSON is easier to parse in JavaScript since it directly maps to JavaScript objects.

40. How do you handle errors when parsing JSON data?

You can handle errors using a try-catch block when parsing JSON data to catch any syntax errors.

```
try {
    const obj = JSON.parse(jsonString);
} catch (error) {
    console.error("Invalid JSON:", error);
}
```

41. What are JavaScript modules?

JavaScript modules are reusable pieces of code that can be exported from one file and imported into another. They help in organizing code, promoting reusability, and managing dependencies. Modules can be defined using ES6 syntax (`import` and `export`) or CommonJS (e.g., `require`).

42. What is the difference between synchronous and asynchronous execution in JavaScript?

- **Synchronous execution**: Code runs in sequence, blocking subsequent operations until the current operation completes.
- **Asynchronous execution**: Code can run concurrently, allowing other operations to proceed while waiting for tasks (like network requests) to complete.

43. What are the bind, call, and apply methods in JavaScript?

- **bind**: Creates a new function with a specified `this` value and initial arguments. It returns a new function.

```
const boundFunction = myFunction.bind(obj, arg1);
```

- **call**: Calls a function with a specified `this` value and arguments passed individually.

```
myFunction.call(obj, arg1, arg2);
```

- **apply**: Similar to `call`, but arguments are passed as an array.

```
myFunction.apply(obj, [arg1, arg2]);
```

44. Can you explain the concept of the "closure scope chain"?

A closure scope chain refers to the nested scopes created by closures. When a function is created inside another function, it has access to its own scope, the outer function's scope, and the global scope. This allows inner functions to retain access to variables from their containing scopes.

45. How does JavaScript handle memory management and garbage collection?

JavaScript uses automatic memory management and garbage collection to reclaim memory occupied by objects that are no longer in use. The garbage collector identifies unreachable objects and frees up that memory, ensuring efficient memory usage.

46. What are the different types of JavaScript data types?

JavaScript has several data types, which can be categorized as:

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

47. How can you optimize JavaScript code for performance?

- Minimize DOM manipulations.
- Use event delegation.
- Reduce memory usage by avoiding memory leaks.
- Optimize loops and algorithm complexity.
- Use `requestAnimationFrame` for animations.
- Utilize caching for repeated computations.

48. What are service workers, and how do they work in modern web applications?

Service workers are scripts that run in the background, separate from a web page, allowing you to intercept network requests, cache resources, and enable offline capabilities. They enable features like push notifications and background sync in web applications.

49. What is the purpose of the fetch API?

The fetch API provides a modern way to make network requests in JavaScript, returning a promise that resolves to the response of the request. It simplifies making requests compared to older methods like `XMLHttpRequest`.

```
fetch('url')
    .then(response => response.json())
    .then(data => console.log(data));
```

50. What are WebSockets, and how do they differ from HTTP requests?

WebSockets provide a full-duplex communication channel over a single, long-lived connection, enabling real-time data exchange between the client and server. Unlike HTTP requests, which are request-response based and stateless, WebSockets allow for persistent connections, making them ideal for applications requiring real-time updates, like chat apps or live data feeds.

=====*=================*============*============*==========*========