1. What is the difference between instance methods and static methods?

Ans: **Instance methods** are called on an instance of the class. They can access and modify the instance's attributes

Static methods are called on the class itself, not on an instance. They do not have access to instance attributes or methods unless explicitly passed.

2. How does Javascript handle concurrency?

Ans: JavaScript handles concurrency using an event-driven, non-blocking, asynchronous programming model facilitated by its single-threaded event loop and callback mechanism. The event loop continuously checks the call stack and event queue, executing callbacks from the queue when the stack is empty. JavaScript uses non-blocking I/O operations to prevent the program from being stuck on a single task, and employs callbacks, Promises, and async/await for managing asynchronous operations. Additionally, Web APIs handle asynchronous tasks, which are placed into task queues prioritized by the event loop, ensuring efficient execution order. This model allows JavaScript to manage concurrency effectively despite being single-threaded.

3. What is async/await? How does it differ from using the promise instance methods?

Ans: async/await in JavaScript makes working with Promises easier and the code more readable. The async keyword is used to declare a function that returns a Promise, and await pauses the function until the Promise resolves, allowing you to handle errors with try/catch blocks. This approach looks like regular, synchronous code, unlike using Promises directly, which involves chaining .then() and .catch() methods and can get messy. Both async/await and Promises can run tasks in parallel using Promise.all(). Overall, async/await makes writing and managing asynchronous code simpler and cleaner.

4. Can you use await outside of an async function?

Ans: No, you cannot use await outside of an async function in JavaScript. The await keyword is designed to work only within async functions. If you try to use await outside of an async function, you will get a syntax error.

5. What is callback hell and why is it considered a problem?

Ans: Callback hell occurs in asynchronous programming when multiple levels of nested callbacks make code hard to read, maintain, and debug. It complicates error handling, flow control, and violates modular code principles.