Great 🚀 Let's dive into **11. Event-Driven Programming in Node.js**.
This is a **core Node.js concept** because Node itself is built on the **event-driven architecture** (non-blocking I/O, async ops, callback/event loop).

---

11. Event-Driven Programming in Node.js

📌 What is Event-Driven Programming?

- **Definition**: A programming paradigm where the flow of the program is determined by **events** (user actions, messages, I/O signals, timers).
- Instead of executing code sequentially, Node.js **reacts** to events.

👉 Example:

- HTTP server → reacts to requests.
- Streams → emit data, end, error events.
- Database queries → emit success/error events.

📌 Node.js Event Loop (Quick Refresher)

- The **event loop** listens for events and dispatches callbacks.
- Node.js uses **libuv** under the hood.
- Key phases: **timers** → **pending callbacks** → **idle** → **poll** → **check** → **close callbacks**.

👉 Interviewers often ask:

- "How does Node.js handle multiple requests with a single thread?"
- Answer: **event loop + non-blocking async I/O**.

📌 EventEmitter Class

Node provides the `events` **module** with the `EventEmitter` class to implement event-driven behavior.

Example 1: Basic EventEmitter

```
const EventEmitter = require('events');
const emitter = new EventEmitter();

// Listener
emitter.on('greet', (name) => {
  console.log(`Hello, ${name}!`);
});
```

```
// Emit
emitter.emit('greet', 'Shubham');
```

👉 Output:

```
Hello, Shubham!
```

Example 2: Multiple Listeners

```
emitter.on('data', (msg) => console.log('Listener 1:', msg));
emitter.on('data', (msg) => console.log('Listener 2:', msg));

emitter.emit('data', 'Event-driven programming is powerful!');
```

Example 3: Once Listener

```
emitter.once('init', () => {
  console.log('This runs only once.');
});

emitter.emit('init');
emitter.emit('init'); // Will not run again
```

Example 4: Removing Listeners

```
function sayHello() {
  console.log('Hello again!');
}
emitter.on('hello', sayHello);

emitter.emit('hello');
emitter.removeListener('hello', sayHello);
emitter.emit('hello'); // No output
```

📌 Real-World Uses of EventEmitter

1. **HTTP Server**

```
const http = require('http');
const server = http.createServer();

server.on('request', (req, res) => {
  res.writeHead(200);
  res.end('Hello, Event-driven World!');
});

server.listen(3000);
```

2. **Streams**

```
const fs = require('fs');
const readStream = fs.createReadStream('file.txt');

readStream.on('data', chunk => console.log('Chunk:', chunk.toString()));
readStream.on('end', () => console.log('File read completed.'));
```

3. **Custom Business Logic**

- Notifications
- Logging system
- Message queues

📌 Error Handling with Events

- Always listen for `error` events to prevent crashes.

```
emitter.on('error', (err) => {
  console.error('Error caught:', err.message);
});

emitter.emit('error', new Error('Something went wrong!'));
```

📌 Interview Questions

1. **What is EventEmitter in Node.js?**
   → A class that helps implement event-driven architecture by emitting and listening to events.

2. **How is Node.js inherently event-driven?**
   → It uses an event loop with async non-blocking I/O.

3. **Difference between `on()` and `once()`?**
   → `on()` → multiple times, `once()` → only first time.

4. **What happens if you don't handle an `error` event?**
   → The process crashes.

5. **Real-world use case of EventEmitter in Node.js?**
   → Streams, WebSocket connections, notification system, logging, message brokers.

☑ **Key Takeaway**:

- Node.js thrives on **event-driven architecture**.
- **EventEmitter** is at the core (used in streams, HTTP, sockets).
- **Always handle errors** in events.
- This is often asked in interviews because it tests **fundamentals**.

👉 Next up is **12. Streams in Node.js** (which builds directly on top of EventEmitter).

Do you want me to **cover Streams as the next logical step**, or would you prefer me to **jump to performance topics (clustering, scaling, worker threads)** since those are usually heavier in interviews?