Perfect 🚀 Let's tackle **3. Core Modules in Node.js**.
These are **built-in modules** (no npm install needed) and they come up a lot in **interviews & real-world coding**.

---

3. Core Modules in Node.js

Node.js provides several **inbuilt modules** to handle **I/O operations, networking, utilities, events, and streams**.

3.1 File System (`fs`)

- Used to interact with the file system.
- Provides **sync & async** methods.

Example:

```
const fs = require('fs');

// Async read
fs.readFile('file.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log("Async Read:", data);
});

// Sync read
const data = fs.readFileSync('file.txt', 'utf8');
console.log("Sync Read:", data);
```

👉 **Interview Tip**:

- Always prefer **async methods** in production (non-blocking).
- Sync methods block event loop → bad for performance.

3.2 HTTP & HTTPS (`http`, `https`)

- Used to create servers or make requests.
- Forms the foundation of Express.js.

Example: Simple HTTP Server

```
const http = require('http');

const server = http.createServer((req, res) => {
```

```
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello from Node.js Server');
});

server.listen(3000, () => console.log("Server running on port 3000"));
```

👉 **Interview Tip**:

- Express is built on top of the `http` module.
- Understand raw `http` → helps in debugging and scaling.

3.3 Path (`path`)

- Helps manage **file and directory paths** (cross-platform safe).

Example:

```
const path = require('path');

console.log(path.basename('/home/user/file.txt'));  // file.txt
console.log(path.dirname('/home/user/file.txt'));   // /home/user
console.log(path.extname('/home/user/file.txt'));   // .txt
console.log(path.join('folder', 'subfolder', 'file.txt'));
// folder/subfolder/file.txt
```

👉 Prevents issues with different OS path separators (`/` vs `\`).

3.4 OS (`os`)

- Provides system-related information.

Example:

```
const os = require('os');

console.log("CPU Architecture:", os.arch());
console.log("OS Platform:", os.platform());
console.log("Total Memory:", os.totalmem());
console.log("Free Memory:", os.freemem());
console.log("Hostname:", os.hostname());
```

👉 Useful in **logging, monitoring, scaling decisions**.

3.5 Events (`events`)

- Node.js is **event-driven**.
- `EventEmitter` class allows custom event handling.

Example:

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

event.on('greet', (name) => {
  console.log(`Hello, ${name}`);
});

event.emit('greet', 'Shubham');
```

👉 **Interview Tip**:

- Express middleware & Socket.io internally use EventEmitter.

3.6 Stream (`stream`)

- Used to handle **large data efficiently**.
- Streams process data **in chunks**, not all at once (saves memory).
- Types of streams:
    - **Readable** (fs.createReadStream)
    - **Writable** (fs.createWriteStream)
    - **Duplex** (both read & write, e.g., sockets)
    - **Transform** (modify data, e.g., compression)

Example: Copy file using streams

```
const fs = require('fs');

const readStream = fs.createReadStream('input.txt');
const writeStream = fs.createWriteStream('output.txt');

readStream.pipe(writeStream); // stream data chunk by chunk
```

👉 Much faster & memory-efficient than reading entire file at once.

3.7 Buffer (`buffer`)

- Used to handle **binary data** (images, files, network packets).
- Especially useful with streams.

Example:

```
const buf = Buffer.from('Hello');
console.log(buf);          // <Buffer 48 65 6c 6c 6f>
console.log(buf.toString()); // Hello
```

3.8 Utility (`util`)

- Provides helper functions like `promisify`, `inherits`, debugging.

Example: Convert callback to Promise

```
const fs = require('fs');
const util = require('util');

const readFile = util.promisify(fs.readFile);

readFile('file.txt', 'utf8')
  .then(data => console.log(data))
  .catch(err => console.error(err));
```

📌 Interview Q&A

☑ **Q: Difference between `fs.readFile` and `fs.createReadStream`?**
👉 `fs.readFile` loads entire file in memory → bad for large files.
👉 `fs.createReadStream` reads file in chunks → efficient & memory safe.

☑ **Q: Why use streams in Node.js?**
👉 For **large data handling** (logs, video, file transfer) → prevents memory exhaustion.

☑ **Q: What is EventEmitter in Node.js?**
👉 A class that allows objects to **emit events** and **register listeners**.

☑ **Q: What is Buffer in Node.js?**
👉 A raw memory allocation outside V8 heap to store **binary data**.

📌 Key Takeaways

- **fs** → File system operations
- **http/https** → Web servers & API requests
- **path** → Cross-platform path handling

- **os** → System info
- **events** → EventEmitter pattern
- **stream** → Handle big data efficiently
- **buffer** → Work with binary data
- **util** → Helpers (promisify, debugging)

👉 Next up: **4. Package Management (npm, yarn, semver, package.json)**.

Do you want me to explain **npm & package.json deeply with real examples**, or should I keep it **interview-focused short notes**?