JavaScript — short-answer fundamentals

1. Difference between var, let, and const:
var is function-scoped and hoisted; it can be redeclared. let and const are block-scoped; they are not hoisted in the same way (temporal dead zone) and cannot be redeclared in the same scope. const creates a read-only binding (object/array contents can still be mutated).

2. Explain closures:
A closure is a function that retains access to its lexical scope even when executed outside that scope. Example: returning an inner function that references variables from the outer function — those variables persist as long as the inner function exists.

3. Promises vs async/await:
Promises represent eventual values and use .then()/.catch(). async/await is syntactic sugar around promises making asynchronous code read like synchronous code; await pauses execution inside async functions until the promise resolves.

4. What is the event loop?
The event loop coordinates execution of call stack and callback queues. It lets Node.js (and browsers) perform non-blocking I/O — when async operations complete they queue callbacks/microtasks which the event loop processes when the stack is empty.

Node.js / Backend

1. Example simple Express CRUD endpoint (create + list users):

```
// server.js
const express = require('express');
const bodyParser = require('body-parser');
const app = express();
app.use(bodyParser.json());

let users = []; // in-memory for demo

// Create user
app.post('/users', (req, res) => {
const { name, email } = req.body;
if (!name || !email) return res.status(400).json({ error: 'name & email required' });
const user = { id: Date.now().toString(), name, email };
users.push(user);
res.status(201).json(user);
});

// List users
app.get('/users', (req, res) => res.json(users));

app.listen(3000, () => console.log('Server listening on 3000'));
```

2. Middleware & error handling (brief):
Middleware are functions that receive (req, res, next) and can modify request/response or call

next() to continue. Central error-handling middleware has four args (err, req, res, next) and should send consistent error responses and log details.

3. Streams & when to use them:
Streams process data chunk-by-chunk (readable/writable) and are memory-efficient for large payloads (file uploads/downloads, piping responses).

4. Security essentials:

Use HTTPS, helmet for headers, rate limiting, input validation/sanitization, parameterized queries to avoid SQL injection, and secure cookie settings for sessions.

Databases (SQL & NoSQL)

1. SQL vs NoSQL (short):
SQL (relational) uses structured schemas, ACID transactions, and joins — good for structured relational data. NoSQL (document/key-value/column) is schema-flexible, horizontally scalable — good for unstructured or changing data and high throughput.

2. Example normalized table for users and orders:

users(id PK, name, email)

orders(id PK, user_id FK -> users.id, total, created_at)

3. Example SQL: get top 5 users by total order value

```
SELECT u.id, u.name, SUM(o.total) AS total_spent
FROM users u
JOIN orders o ON u.id = o.user_id
GROUP BY u.id, u.name
ORDER BY total_spent DESC
LIMIT 5;
```

4. Indexing:
Create indexes on columns used in WHERE, JOIN, and ORDER BY to improve read performance; be mindful of write cost and storage overhead.

System design / Architecture (short)

Design a simple lead-capture flow:

Frontend form -> POST to API Gateway -> Node.js service (validates + persists in DB) -> Message queue (e.g., RabbitMQ) -> Worker sends email + stores analytics. Use retries, idempotency keys, and monitoring.

Stateless services:
Keep services stateless (store sessions in Redis or JWT) to allow horizontal scaling and easier deployment.

AI Awareness (short)

1. What is AI / ML / Deep Learning:
AI is designing systems that perform tasks that normally require human intelligence. ML is a subset where models learn from data. Deep Learning uses neural networks with many layers for tasks like vision and language.

2. Prompt engineering basics:
Be explicit about format, constraints, examples, and desired style. For models that continue to learn, avoid asking for private info; validate outputs and add verification steps for high-stakes use.

3. Safety & ethics (short):
Address bias in data, privacy (avoid storing PII unnecessarily), explainability, and user consent; ensure models are evaluated for fairness and safety.

Typical coding question — sample answer (reverse string)

Problem: Reverse a string.
Solution (JS):

```
function reverseString(s) {
return s.split('').reverse().join('');
}
```