
🟧 **/JavaScript_Theory**

- Primitive vs Non-Primitive
- Mutable vs Immutable
- Pass by Value vs Pass by Reference
- Higher Order Function vs Callback Function
- TDZ (Temporal Dead Zone)
- Ternary Operator
- Nullish Coalescing Operator
- Optional Chaining
- Map vs forEach
- find(), some(), every()
- Constructor Function vs Object Literals
- Object.entries / Object.fromEntries
- hasOwnProperty
- Pure Function
- Reference
- Trim
- Regex
- Async / Await
- Deep Copy of Object
- Destructure an Object
- Currying (Example)

🟩 **/JavaScript_Practicals**

- Sum multiple numbers using Arrow Function
- Create a Promise to check if a number is Odd or Even
- Swap Even & Odd Characters in a string ("abababab" → "bababa")
- Using while loop — sum of multiples of 3 for the first 5 even numbers
- Find count of 0 and 1 in array without if condition

- Find sum of even numbers using reduce
- Find count of digits in a number
- Practicals using Timer Functions (setTimeout, setInterval)
- People grouping by age → { '21': ['Alice', 'Charlie'], '22': ['Bob','Eva'] }
- Invoices → return customer products / handle “no products” or “no user”
- Recursive file system → return all file names
- Flight time using Binary Search (find exact or next slot)

🟦 **/NodeJS_Theory**

- Core Modules
- FS (File System)
- HTTP Requests
- Headers in HTTP
- What is Middleware
- Application Level Middleware
- app.use vs app.get
- req.query vs req.params
- Content Negotiation
- Modules (CommonJS vs ESModules)
- CORS
- Express Router()
- app.all / app.set / app.locals
- Custom Middleware
- Schema (Concept)

🟧 **/NodeJS_Practicals**

- Create Server using Express
- App-Level Middleware Example
- Query Params – Find Sum of Two Numbers
- Middleware to Block POST Request
- Middleware to Handle “Page Not Found”

- Function to Append a File
- Improve Basic Node.js (custom middleware, routing, cors)
- app.post Example
- Custom Middleware (Practical)
- Application-Level Middleware Practical
- CORS Setup
- Express Router Practical

🟪 **/MongoDB_Theory**

- Sharding
- Cardinality
- Vertical vs Horizontal Scaling
- Different Types of Joins
- Covered Queries
- Accumulator Operators
- Aggregation Framework
- $each Operator
- $in Operator
- Populate (Concept)
- Lookup (Aggregation Join)
- Timestamp
- Schema Structure

🟫 **/MongoDB_Practicals**

- Capped Collection
- Find name starting with ‘s’
- Find names starting with ‘S’ or ‘B’
- Find 2nd most expensive product
- Decrease price by 100
- Average price per category
- Count products by category

- Increase age of students in BCA by 5
- Difference between highest and lowest
- Count of books in each year
- Author names starting with ‘k’
- Lookup Example
- Populate Example
- Timestamp Example

🟧 **/React_Topics**

- Child-to-Parent Communication
- React Timer App using useEffect
- Lifecycle Methods (Class Components)
- useEffect (Hook)
- Custom Hook Example
- Lazy Component Import Example
- Can name be changed without setName in useState?
- Is React one-way or two-way data flow?

🟦 **/DSA_Topics**

- Find Middle Node (Linked List)
- Binary Search (Recursive Algorithm)
- Quick Sort (Recursive Algorithm)
- Array Sorting (O(n log n)) — [0,0,1,1,0,1,0,0,1]

🟥 **/Combined_Revision_List**

- Middleware & Routing in Express
- $in, $each, Lookup, Populate, Aggregation (MongoDB + Node Integration)
- Improve Array & Object Problem Solving
- Project-related Questions for all CRUD + Aggregation + Middleware Topics

---

🟩 **/Pending_Tasks**

JavaScript (P:4, T:5)

- Deep copy for object
- Destructure object
- Optional chaining
- Timer functions
- Currying

Node.js (P:5, T:5)

- Content Negotiation
- req.query vs req.params
- Middleware examples
- CORS setup
- Custom router practical

MongoDB (P:4, T:5)

- Difference between highest and lowest
- Count books in each year
- Author names starting with 'k'
- Covered queries

Feedback: Focus more on MongoDB practicals & JS logical questions.

People Grouping Example

```
const people = [
 { name: "Alice", age: 21 },
 { name: "Bob", age: 22 },
 { name: "Charlie", age: 21 },
 { name: "David", age: 23 },
 { name: "Eva", age: 22 }
];
```

Expected output:

```
{ '21': ['Alice', 'Charlie'], '22': ['Bob', 'Eva'], '23': ['David'] }
```

Invoices Example

```
const invoices = [
 { id: 201, customerId: 10, total: 1500, status: 'paid', items: ['monitor', 'cable'] },
 { id: 202, customerId: 11, total: 2300, status: 'unpaid', items: ['laptop'] },
```

```
 { id: 203, customerId: 10, total: 600, status: 'paid', items: ['keyboard', 'mouse'] },
 { id: 204, customerId: 12, total: 800, status: 'cancelled', items: ['webcam'] },
 { id: 205, customerId: 15, total: 800, status: 'cancelled', items: null },
];
```

Task:

Return the products of the given customerId.

- If no products → return “no products”.
- If no user → return “no user”.

Binary Search Flight Example

const flights = ['06:30', '08:15', '09:00', '10:45', '13:20', '15:50', '18:10'];

Find the **exact flight time**;

if not available → return the **next time slot**;

if none → return **“no flights available”** (use Binary Search).

Async/Await Salary Example (Error Finding)

```
function fetchEmployee(id) {
 return new Promise(resolve => {
  setTimeout(() => {
   const employees = [
    { id: 1, name: "Alice", base: 50000, bonus: "5000" },
    { id: 2, name: "Bob", base: 40000 },
    { id: 3, name: "Charlie", base: 60000, bonus: 7000 },
   ];
   resolve(employees.find(emp => emp.id = id));
  }, 500);
 });
}

function calculateGross(emp) {
 return emp.base + (emp.bonus || 0);
}

// Apply tax (10%)
function applyTax(amount) {
 return amount - amount * 0.1;
}

async function getNetSalary(id) {
 const emp = fetchEmployee(id);
 if (!emp) return "Employee not found";
 const gross = calculateGross(emp);
 return applyTax(gross);
}

(async () => {
 console.log("Alice Salary:", await getNetSalary(1));
 console.log("Bob Salary:", await getNetSalary(2));
```

```
  console.log("Charlie Salary:", await getNetSalary(3));
})();
```

Task: Find the errors.

Recursive File System

```
const fileSystem = {
  name: 'root',
  files: ['file1.txt', 'file2.txt'],
  folders: [
    {
      name: 'docs',
      files: ['doc1.pdf', 'doc2.pdf'],
      folders: [
        { name: 'personal', files: ['resume.docx'], folders: [] },
      ],
    },
    {
      name: 'images',
      files: ['photo1.jpg', 'photo2.jpg'],
      folders: [],
    },
  ],
};
```

Task: Return an array with values of key **"files"** using recursion.