Phase 2: Control Flow & Functions

IT Ticket Priority Sorter Project

Project Overview

Build a console-based help desk ticket system that accepts, organizes, prioritizes, and filters support requests. This phase focuses heavily on decision-making logic and writing clean, reusable functions. You'll practice conditionals, loops, and function design patterns that show up constantly in real-world JavaScript.

The Scenario

Your IT department receives tickets from across the facility—production floor issues, network problems, Oracle database alerts, ERP questions, and general desktop support. You need a system that accepts new tickets, automatically assigns priority based on rules, lets you work through the queue, and provides filtered views of what's pending.

Data Structure

Each ticket is an object:

```
{
  id: 5001,
```

```
  title: "Oracle database connection timeout",
  description: "Epicor users getting disconnected every 15 minutes",
  category: "database",       // database, network, hardware, software, erp, other
  priority: "critical",       // critical, high, medium, low
  status: "open",             // open, in-progress, resolved, closed
  submittedBy: "Maria Garcia",
  department: "Production",
  createdAt: "2026-02-07T08:30:00",
  updatedAt: "2026-02-07T08:30:00",
  assignedTo: "",
  notes: []
}
```

Your ticket queue is an array:

```
const tickets = [];
let nextTicketId = 5001;
```

Features to Build

1. Create Ticket with Auto-Priority

This is the core function. It accepts ticket details and automatically assigns a priority based on category and keywords in the title or description.

```
createTicket(
  "PLC communication failure on Cell 3",
  "Keyence scanner not reading parts, line is down",
  "hardware",
  "Juan Martinez",
  "Die Cast"
)
```

Priority Rules (implement with conditionals):

Condition	Priority
Category is "database" AND description contains "down" or "timeout"	critical
Category is "network" AND title contains "down" or "failure"	critical
Title or description contains "line down" or "production stopped"	critical
Category is "erp" AND department is "Production" or "Shipping"	high
Category is "database"	high

Condition	Priority
Category is "network"	high
Category is "hardware"	medium
Category is "software"	medium
Everything else	low

The function should check conditions in order—first match wins.

> ***Concepts practiced:*** *if/else if/else chains, logical operators (&&, ||), string methods (toLowerCase, includes), function parameters*

2. Keyword Detection Helper

Before building the full `createTicket`, write a helper function that checks if a string contains any word from a list of keywords.

```
containsKeyword("The Oracle server is down and users cannot connect", ["down", "timeout", "failure"])
// Returns: true

containsKeyword("Need help with Excel formula", ["down", "timeout", "failure"])
// Returns: false
```

> ***Concepts practiced:*** *functions that return booleans, loops through arrays, early returns*

3. Update Ticket Status

Create a function to move tickets through the workflow. It should enforce valid transitions.

```
updateStatus(5001, "in-progress", "Wesley")
updateStatus(5001, "resolved")
```

Valid transitions:

- open → in-progress (requires assignedTo)
- in-progress → resolved
- in-progress → open (kicked back)
- resolved → closed
- resolved → in-progress (reopened)

Invalid transitions should return an error message.

> ***Concepts practiced:*** *switch statements, validation logic, multiple parameters with defaults*

4. Add Note to Ticket

Tickets need a history of updates and troubleshooting steps.

```
addNote(5001, "Checked redo logs - no issues found", "Wesley")
// Adds to the notes array: { text: "...", author: "Wesley", timestamp: "2026-02-07T09:15:00" }
```

> ***Concepts practiced:*** *finding objects, pushing to nested arrays, working with dates*

5. Filter Functions

Build a set of functions that return filtered views of the ticket queue.

Get open tickets:

```
getOpenTickets()
// Returns all tickets where status is "open" or "in-progress"
```

Get tickets by priority:

```
getByPriority("critical")
// Returns all critical tickets regardless of status
```

Get tickets by category:

```
getByCategory("database")
```

Get tickets by department:

```
getByDepartment("Production")
```

Get my tickets:

```
getAssignedTo("Wesley")
// Returns tickets assigned to a specific person
```

> ***Concepts practiced:*** *filter patterns, reusable function design*

6. Sort Functions

Build functions to sort the ticket array by different criteria.

Sort by priority:

```
sortByPriority(tickets)
// Returns array sorted: critical → high → medium → low
```

This one is interesting because priority isn't alphabetical. You'll need to define a priority order.

Sort by date (oldest first):

```
sortByAge(tickets)
// Oldest tickets first (for working the queue)
```

Sort by date (newest first):

```
sortByNewest(tickets)
// Most recent first
```

> ***Concepts practiced:*** *array sorting with custom comparators, working with date strings*

7. Get Next Ticket

Returns the highest priority, oldest ticket that's still open. This is what you'd grab when you're ready to work the queue.

```
getNextTicket()
// Returns the single most urgent ticket, or a message if queue is empty
```

> ***Concepts practiced:*** *combining filter and sort, returning single items vs arrays*

8. Ticket Summary Dashboard

Generate a report of the current queue state.

```
generateDashboard()



====== IT Ticket Dashboard ======
Open Tickets: 12
In Progress: 4
Resolved (pending close): 3

By Priority:
  Critical: 2 ⚠️
  High: 5
  Medium: 6
  Low: 6

Oldest Open Ticket: 3 days (ID: 5003)

Critical Tickets:
  #5007 - Oracle database connection timeout (Production)
  #5012 - Network switch failure IDF East (Network)
```

> ***Concepts practiced:*** *combining multiple operations, formatted output, calculating time differences*

9. Bulk Operations

Close all resolved tickets:

```
closeAllResolved()
// Changes status from "resolved" to "closed" for all resolved tickets
// Returns count of tickets closed
```

Reassign tickets:

```
reassignTickets("Wesley", "Mike")
// Moves all of Wesley's tickets to Mike
```

> ***Concepts practiced:*** *for loops that modify data, counting operations, batch updates*

Starter Data

```
const tickets = [
  {
    id: 5001,
    title: "Epicor running slow",
    description: "Reports taking 5+ minutes to generate",
    category: "erp",
    priority: "high",
    status: "open",
    submittedBy: "Linda Chen",
    department: "Accounting",
    createdAt: "2026-02-05T14:20:00",
```

```
  updatedAt: "2026-02-05T14:20:00",
  assignedTo: "",
  notes: []
},
{
  id: 5002,
  title: "Printer not working",
  description: "HP printer in Quality Lab showing offline",
  category: "hardware",
  priority: "medium",
  status: "open",
  submittedBy: "Carlos Ruiz",
  department: "Quality",
  createdAt: "2026-02-06T09:45:00",
  updatedAt: "2026-02-06T09:45:00",
  assignedTo: "",
  notes: []
},
{
  id: 5003,
  title: "Cannot access shared drive",
  description: "Getting access denied on S: drive",
  category: "network",
  priority: "high",
  status: "in-progress",
  submittedBy: "Maria Santos",
  department: "Shipping",
  createdAt: "2026-02-04T11:00:00",
  updatedAt: "2026-02-06T16:30:00",
  assignedTo: "Wesley",
  notes: [
    { text: "Checking AD permissions", author: "Wesley", timestamp: "2026-02-06T16:30:00" }
  ]
},
{
  id: 5004,
```

```
    title: "Keyence scanner line down",
    description: "Cell 4 scanner not communicating with PLC, production stopped",
    category: "hardware",
    priority: "critical",
    status: "open",
    submittedBy: "Jorge Mendez",
    department: "Production",
    createdAt: "2026-02-07T07:15:00",
    updatedAt: "2026-02-07T07:15:00",
    assignedTo: "",
    notes: []
  }
];

let nextTicketId = 5005;
```

Build Order

1. **containsKeyword helper** — Small, testable function to start
2. **createTicket with auto-priority** — The main logic challenge of this phase
3. **updateStatus** — Practice with switch and validation
4. **addNote** — Working with nested data
5. **Filter functions** — Build all five, notice the patterns
6. **sortByPriority** — Custom sort logic
7. **sortByAge and sortByNewest** — Date sorting
8. **getNextTicket** — Combine filter and sort
9. **generateDashboard** — Putting it all together

10. **Bulk operations —** Loops that modify data

Function Design Principles

Single responsibility: Each function does one thing. `containsKeyword` just checks for keywords—it doesn't create tickets or assign priority.

Pure functions where possible: Filter and sort functions should return new data without modifying the original array (until you get to bulk operations, which intentionally modify).

Consistent return types: If a function returns an array, it should always return an array (even if empty). Don't return null or a string message when the array is empty.

Early returns for validation: Check for invalid input at the top of your function and return early rather than nesting everything in conditionals.

```
// Prefer this:
function updateStatus(ticketId, newStatus) {
  const ticket = findById(ticketId);
  if (!ticket) {
    return "Ticket not found";
  }
  // ... rest of logic
}

// Over this:
function updateStatus(ticketId, newStatus) {
  const ticket = findById(ticketId);
  if (ticket) {
    // ... deeply nested logic
  } else {
    return "Ticket not found";
```

```
  }
}
```

Stretch Goals

- **SLA tracking:** Add a dueBy field calculated from priority (critical = 4 hours, high = 8 hours, etc.) and flag overdue tickets
- **Ticket templates:** Create functions that generate common ticket types with pre-filled fields
- **Escalation:** Auto-escalate tickets that have been open too long
- **Statistics:** Calculate average resolution time, tickets per category, busiest department
- **Search:** Full-text search across title, description, and notes

Pair Programming Suggestions

- **Driver/navigator rotation:** Switch who's typing every 20-30 minutes
- **Split the filters:** One person builds getOpenTickets and getByPriority, the other builds getByCategory and getByDepartment, then compare approaches
- **Priority logic challenge:** Each person writes the createTicket priority logic independently, then compare—you'll likely find different but equally valid approaches
- **Code review:** After building updateStatus, review each other's transition validation logic