AI Automation / Task Orchestration

Design if/then/else logic for workflows where different inputs need different paths — handles variability without separate workflows.
Difficulty: Intermediate
Model: GPT-4 / Claude / Gemini
Use Case: Workflow Logic, Conditional Processing
Updated: May 2026
Why This Prompt Exists
Real-world workflows aren’t linear. Different inputs require different paths. Most automation tools support conditionals, but designing them correctly is non-obvious.

You get:

- separate workflows for each condition (duplication, maintenance nightmare)
- conditions that are too broad (wrong path taken)
- conditions that are too narrow (some inputs fall through cracks)
- no default path (unmatched inputs do nothing — silently fail)
- branch logic that’s impossible to debug (nested conditions too deep)

But conditional branches have patterns:

- if/then: single condition, single alternative path
- if/else if/else: multiple conditions, ordered by priority
- switch/case: one value mapped to many possible paths
- parallel conditions: multiple independent conditions (not mutually exclusive)
- default: catch-all for unmatched inputs

Without conditional design, workflows are brittle.

This prompt designs robust conditional branching logic.

The Prompt

```
Assume the role of a workflow logic architect who designs conditional branches.

Your task is to design if/then/else logic for a workflow with variable inputs.

Generate:

1. DECISION POINT
   - Where in the workflow branching occurs
   - What data determines the branch

2. CONDITIONAL LOGIC TABLE

| Priority | Condition | Branch Path | Expected Input % | Actions |
|----------|-----------|-------------|------------------|---------|
| 1 | [condition, e.g., "amount > $10,000"] | VIP Approval | 5% |
[notify manager, wait approval] |
| 2 | [condition, e.g., "amount > $1,000"] | Manager Review | 15% |
[email manager, auto-approve after 24h] |
| 3 | [else] | Auto-Approved | 80% | [create invoice, send receipt] |

3. CONDITION TYPE
   - Single condition (if/then)
   - Multiple ordered (if/else if/else)
```

   - Parallel (multiple independent conditions)
   - Value mapping (switch/case)

4. DEFAULT BEHAVIOR
   - What happens if no condition matches
   - Should this be an error, a default path, or a question to user?

5. NESTING DEPTH
   - Current nesting depth: [X]
   - Recommended max: 3 levels (deeper = hard to debug)
   - Flattening suggestions: [how to reduce nesting]

6. IMPLEMENTATION NOTES
   - Where to place the branch in the workflow
   - How to handle branch merging (if paths reconverge)

7. TEST CASES
   - Input that triggers each branch
   - Edge case near condition boundaries (e.g., exactly $1,000)

INPUTS:

Workflow description:
[E.G., "Purchase order approval process"]

Decision data:
[E.G., "Purchase amount, department, urgency flag"]

Branch types needed:

```
[E.G., "Auto-approve, manager review, VP approval, legal review"]

Volume estimates per branch:
[E.G., "80% auto, 15% manager, 4% VP, 1% legal"]

RULES:
- Order conditions from most specific to most general (prevent wrong matches)
- Always include a default/else branch (unmatched inputs need handling)
- Avoid deep nesting (more than 3 levels) — flatten into separate workflows
- Use priority ordering for mutually exclusive conditions
- Use parallel branches for non-mutually exclusive conditions
- Test boundary conditions (e.g., "exactly $1,000" — which branch?)
- Document the business reason for each condition
```

How To Use It

- Order conditions from most specific to most general (prevents wrong matches).
- Always include a default/else branch — unmatched inputs must go somewhere.
- Avoid deep nesting (more than 3 levels) — flatten into separate workflows.
- Use priority ordering for mutually exclusive conditions (only one can match).
- Use parallel branches for non-mutually exclusive conditions (multiple can match).
- Test boundary conditions (e.g., “exactly $1,000” — which branch takes it?).

Example Input

Workflow description:
“Purchase order approval — different amounts need different approval levels”

Decision data:
“Purchase amount, department, urgency flag”

Branch types needed:
“Auto-approve, manager review, VP approval, legal review”

Volume estimates per branch:
“80% auto, 15% manager, 4% VP, 1% legal”

Why It Works
Most workflows start linear and add conditions ad hoc — resulting in spaghetti logic that’s impossible to debug.

This framework improves outcomes by forcing:

- decision point identification (where does branching happen?)
- conditional logic table (explicit conditions, order, and actions)
- default branch design (unmatched inputs don’t fall through cracks)
- nesting depth awareness (deep nesting = unmaintainable)
- boundary testing (ensures correct behavior at thresholds)

Failure modes this prevents:

- No default branch — unmatched inputs do nothing (silent failure)
- Wrong condition order — e.g., “$1,000+ catches $10,000+ first”
- Deep nesting — conditions inside conditions inside conditions (impossible to debug)
- Missing business documentation — no one knows why each branch exists
- Boundary ambiguity — what happens at exactly $1,000?

This improves on: Ad hoc conditional logic added as an afterthought. Structured branching is maintainable and testable.

Related to: TO-01 (Dependencies) for data needed to evaluate conditions; TO-04 (Recovery) for branch-specific error handling.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also Orchestration Recovery Planner