The second person is the real test.

You have answered the same question five times. This time you build a small tool, send the link, and explain that it should save everyone time. Tomorrow the question comes back to you. The tool worked. The handoff didn't.

I have built operating tools with AI and watched teammates extend them into work I would not have designed myself. That is the part I care about: someone understands the rules well enough to change the work, then takes responsibility for the result.

This guide starts smaller. A fictional warehouse needs to decide which orders can ship. You will give a teammate a working tool, a recoverable failure and a rule they can change. All case records are invented for this exercise.

Decide what the tool is allowed to decide.

The practice warehouse has six orders waiting. Customer service wants a list it can discuss with the warehouse before promising shipment. The tool can recommend full-order allocations from a dated inventory snapshot. It cannot reserve inventory, release orders or tell a customer that a shipment happened.

> The business promise
> Every recommended unit is supported by the same available stock pool. Every held order has a specific reason and an owner action.

Write that promise before choosing a screen or asking for code. It tells the builder what must remain true when the tool gets prettier, faster or easier to use. It also gives the recipient a useful way to disagree: show the order that violates the promise.

Six orders, two stock pools, one missing item.

Stock item	Physical units	Unavailable
TRAY	17	4 reserved, 2 damaged
HOOK	8	2 reserved, 1 damaged

The starting rule keeps two additional units of each known item as a buffer. TRAY therefore has nine units available, and HOOK has three. The snapshot and requested ship date are both October 5, 2026. Units are individual pieces throughout, not cases or packs.

The orders request TRAY quantities of five, four and two, then HOOK quantities of three and one, then one LAMP. The input sequence is the priority. LAMP has no stock record. Missing stock is a question for the warehouse, never a convenient zero and never permission to invent availability.

The first answer should fit on paper.

Order	Request	Recommendation
O-101	5 TRAY	Allocate 5
O-102	4 TRAY	Allocate 4
O-103	2 TRAY	Hold: 0 available
O-104	3 HOOK	Allocate 3
O-105 / O-106	1 HOOK / 1 LAMP	Hold: shortage / missing stock

Nine TRAY units and three HOOK units are allocated. Three orders can proceed to warehouse review; three remain held. The output carries the snapshot date and rule version so a saved result can be understood later.

Do this calculation before looking at the tool's answer. Otherwise a plausible output becomes the answer key, and every later test simply agrees with it. The included expected results were written from this allocation sequence. They are separate files, not values exported by the program under test.

A held order does not disappear.

Full orders only. Process the stated sequence. A held order does not consume stock, so a later, smaller order may fit. Those are business choices. Another warehouse may insist that the first shortage blocks everyone behind it. That is a different tool until the policy changes explicitly.

The output gives each order its requested quantity, allocated quantity, remaining availability and reason. A missing item sends the user to the stock owner. A shortage sends them to review timing or allocation. A stale snapshot stops the entire run because every recommendation could be wrong.

> One controlled boundary
> Expected receipts, substitute items, carrier capacity and partial shipments are outside this exercise. Add one only when its source and decision rule are understood.

You already have the reference tool.

The kit includes a Python program and all six fictional orders. It needs Python 3.10 or newer and no additional packages, account or network connection. Open a terminal in the extracted skill folder, then run:

```
python scripts/promise.py
python scripts/check.py
```

The first command prints the recommendations as JSON, a structured text format an AI or another program can read. The second checks the expected allocations, a changed rule, the recovery cases and stock conservation. Your AI can run the commands if its environment supports Python. Otherwise read the included results and have it explain the same calculation.

A passing exercise proves this small reference behaves as specified. It does not establish that the real warehouse's exports, policies or permissions have been connected.

Give the next person a way back.

Now run the supplied stale snapshot. The program must stop before producing recommendations. The message identifies the date conflict and asks for a matching inventory snapshot. It must not quietly reuse the last successful result.

```
python scripts/promise.py --scenario stale
```

Recovery is a normal part of ownership. The user checks which source is wrong, obtains a valid snapshot, and reruns. In the exercise, return to the baseline fixture. In real work, simply changing a date to satisfy the program would conceal the problem.

The check script also introduces a duplicate order ID. That is rejected as an invalid input, with the duplicate named. A failure that explains what the person can fix is a much more useful handoff than an unexplained red badge.

Change the rule, predict the consequence.

The warehouse asks to hold three buffer units instead of two. Before running the variant, predict which orders will move. TRAY now starts with eight units and HOOK with two. No other rule changes.

Item	Buffer 2	Buffer 3
TRAY	O-101 and O-102	O-101 and O-103
HOOK	O-104	O-105
LAMP	O-106 held	O-106 held

Under the larger buffer, O-102 no longer fits after O-101. Because held orders do not consume stock, O-103 still fits. The same rule lets the smaller HOOK order proceed. The output changes in a way the teammate can explain, including the tradeoff for the earlier customer.

```
python scripts/promise.py --scenario buffer3
```

Configuration is a business decision.

A teammate who can change a number has editing access. A teammate who can explain why that number belongs there, predict the effect and prove the other rules still hold has begun to own the tool.

Keep the buffer in a small rules file with a version and a plain reason. Record the owner who approved a real policy change. The reference kit contains both versions for comparison; neither is a recommendation for your business. Reverting to the previous file should reproduce the earlier output exactly.

If the larger buffer delays an important order, that is a discussion for the policy owner. Do not let the AI rescue the demonstration by making an invisible exception. Exceptions need the same visible reasoning as the normal path.

Keep your hands off the keyboard.

Give a teammate the kit and a task: find the orders that need warehouse review, recover from the stale snapshot, then explain the buffer change. Sit nearby, but do not narrate every click. The moment you have to take over is useful evidence.

Record the exact point of friction. Could they find the output? Did they interpret held as cancelled? Did they know where the inventory came from? Did a successful check sound like permission to promise a shipment? Fix the instruction or behavior responsible, then repeat that step.

> Honest status
> This publication includes a runnable reference and a reproducible exercise. Human usability, adoption and time saved remain unmeasured until someone actually completes the trial.

Leave an owner, not another dependency.

A complete handoff contains the input contract, the rule, the result and the recovery path. It also names who can change policy, who supplies the data and who responds when the tool stops. Those can be the same person on a small team. They still need to be explicit.

The recipient should know which version they have, where its output goes and what has been checked. Put the first command beside a sample result. Put the recovery instruction beside the error. Make the approved configuration easy to compare with the one actually running.

For a live pilot, begin with a copy of one real workflow and review its recommendations before anybody acts. Expand after the recipient can operate it without borrowing your memory.

Three acts, one useful hour.

Spend the first twenty minutes writing the promise and working the baseline by hand. Use the next twenty to run the tool, trigger the stale-input failure and recover. Use the final twenty to change the buffer and compare the results. These are suggested time boxes, not measured completion times.

One person operates, another observes, and the policy owner explains the intended tradeoff. Swap the operator if time allows. Keep a short record of what worked, where help was needed and what remains untested. Use the included trial sheet rather than calling the session a rollout.

Then choose one repeated question from your own team. Keep the first version equally small. The next four pages are complete instructions you can give an AI to help build that handoff.

Start with the work someone will own.

Help me turn one repeated operational task into a small tool a teammate can run, recover and change. Work within the task I provide. First identify the user, the decision they need to make, the current input sources and the person who owns the business rule. Use the supplied materials before asking for missing details.

Write a plain-language promise for the output, including what it may recommend and what it must not imply. Separate confirmed rules from assumptions. Name any missing fact that could change the result. Choose a small example whose answer we can calculate without your implementation, and record that answer before building.

When using the practice kit, keep its fictional records separate from my business. Do not describe its successful runs as evidence of our adoption or savings.

Build the smallest transferable version.

Choose a simple format the intended teammate can actually use. Keep policy choices in a visible configuration or an equally clear control. Document units, identifiers, dates and input freshness wherever they affect the decision. Reject invalid inputs that make the result unsafe to interpret, and preserve unresolved rows with specific reasons.

Build the normal path and one likely recovery path. Make the error explain the next useful action. Include sample input, expected output and the exact command or opening step. Keep outputs distinguishable from inputs and identify which run produced them. Use existing access and tools where appropriate; do not add services or publish data just to complete the demonstration.

Before expanding the scope, finish one end-to-end result that the recipient can inspect and explain.

Test ownership, including a rule change.

Compare the result with the independently prepared example. Test a consequential boundary, such as missing stock or a stale snapshot, and confirm the tool refuses or limits its answer as promised. Deliberately introduce the failure you claim the check catches, then verify that the check fails. Restore the valid case and rerun.

Ask the intended owner to change one policy. State the expected effect before executing, inspect the actual effect and show how to reverse the change. Check that unrelated promises still hold. Do not silently repair the input or adjust the answer key to make the demonstration pass.

Prepare a short recipient trial with observable tasks. If no person has run it, label that stage untested and leave a clear trial plan.

Return a handoff that can survive tomorrow.

Deliver the working tool, complete instructions, sample inputs, expected results, a meaningful check and the recovery procedure. Include the rule owner, data owner, version and exact limitations. State which checks you executed and which results you merely reasoned through. Report failures directly with the evidence needed to reproduce them.

Provide a compact change record: what changed, why, affected outputs, checks run and reversal method. End with the smallest next trial that would establish whether the recipient can operate independently. Do not claim that a saved file, successful command or approving message proves the business outcome.

If the workflow would send messages, change live records or publish material, keep those actions separate from the local demonstration and use the authorization appropriate to the actual request.