



HANDS-ON GUIDE

Hands-on AI-Assisted Development, IT track, Day 3 Hands-on Guide

Hands-on AI-Assisted Development, IT track, Day 3

This guide lists the 9 exercises of Day 3 and the practical exam task in the order you do them, to keep at hand. The slides keep moving ahead, so when you stall, come back to the matching exercise in this guide. It does not give the full text of the prompts you send. It shows only the elements to include, so fill in the content in your own words.

Contents

What you do today

What you use

Exercise 1: Check your files

E2E scope and the shape of e2e.js

Exercise 2: E2E test from import to reload

Exercise 3: Read e2e.js

Exercise 4: Break it and see what fails

Division of work with Playwright

Exercise 5: Add run results to the test report

Skills and custom agents

The 3 places to store them

Exercise 6: Make case design a skill and delegate the review

Exercise 7: Make the regression check a small skill

Exercise 8: Instructions for writing tests

Exercise 9: Improve a skill

Certification criteria and the submission route

Literacy Check

Practical exam task

Instructor feedback and the Certificate of Completion

The README that links the 3 days

Where to look when things go wrong

What you take home today

Rules to follow

What you do today

Today you turn the tests you built through Day 2 into a form you can run from start to finish, and keep the steps and the reviewer for each run in files. The afternoon is for Certification. After the Literacy Check, you import the September data, whose format has changed, on your own, and submit your decisions with reasons.

Time	Steps	Minutes
Morning	Day 2 review and today's preparation (Exercise 1)	[10min]
Morning	E2E tests that run in the browser alone (Exercises 2 to 5)	[90min]
Morning	Break	[10min]
Morning	Locking in the steps and the reviewer (Exercises 6 to 9)	[120min]
Afternoon	Certification and Literacy Check	[40min]
Afternoon	Break	[10min]
Afternoon	Practical exam	[120min]
Afternoon	Break	[10min]
Afternoon	Feedback session, responding to comments, and the Certificate of Completion	[60min]
Afternoon	The 3 days' deliverables and next month's steps	[10min]

The total is 480 minutes. The lunch break comes after "Locking in the steps and the reviewer". The instructor will tell you its length on the day.

What you use

This morning you use the files in the tpro-work folder that you made up to Day 2. No new data is handed out. The data for the afternoon practical exam is handed out in a separate ZIP when the exam starts.

Handouts	Contents
day3/README.txt	Where to extract, and the list of files you create today
day3/02_literacy-check.html	You open it in the Literacy Check in the afternoon. Do not open it until then.

Files you create in this folder today

File	Exercise where you create it
e2e.js , e2e-scenarios.md	Exercises 2 to 4. The E2E test from import to reload.
report.md , report.pdf	Exercise 5. The Day 2 test report with the run results added.
.github/skills/test-design/	Exercises 6 and 9. The case design skill.
.github/agents/security-reviewer.agent.md	Exercise 6. A read-only reviewer.
.github/skills/regress/SKILL.md	Exercise 7. A small skill for the regression check.
.github/instructions/tests.instructions.md	Exercise 8. Instructions that apply only to the tests folder.
submit_(surname in Roman letters).md	Practical exam. The file your submission is made from.
README.md	Wrap-up. 4 lines of steps for next month.

Exercise 1: Check your files

About [3min]

Steps

1. Extract the handout ZIP into the tpro-work folder, and check that a day3 folder now exists inside tpro-work.
2. Open tests.html in the browser and check that the SUMMARY line appears.
3. Check that testcases.md , decision-log.md and security-notes.md are directly under tpro-work. Then write "All 3 OK" in the chat, or the names of any missing files.

If it does not work

Symptom	Fix
tests.html is a blank white page	Paste the first line of the Console (F12) into the chat. The instructor will look at it during the break.
tests.html does not run, testcases.md is missing, or you were absent on Day 2	Write in the chat. The instructor will send you the full set as it was at the end of Day 2 in a private chat. Copy it into tpro-work by following the steps in the README.txt inside.
decision-log.md and security-notes.md are missing	You can continue by creating empty files.

E2E scope and the shape of e2e.js

As covered in the Day 2 test design, the gaps that AI has trouble finding cluster around order and state. Examples are dropping the same CSV twice and dropping an exported file back in. Neither happens in tests that call one function at a time. E2E means checking through the screen, in the order of the operations. The instructor does not just show it on their own screen. You can build an E2E test on your own PC too, one that runs entirely in the browser.

E2E protects only the flow that crosses the screen and the state. That is the chain of import, warnings, list, total, export and reload. It does not repeat what the unit tests already protect.

What e2e.js does	How it works
When it runs	It is loaded only when you open <code>index.html?e2e</code> . If you open it without that, the app is the same as usual.
Import	It holds the CSV rows as strings and passes them to <code>WorkLogApp.importText(text, fileName)</code> , the later stage you split out in Exercise 9 on Day 2. Then it counts the warnings in the warning area and the rows in the list.
Export	It replaces <code>URL.createObjectURL</code> to receive the Blob, turns it into a string with <code>Blob.text()</code> , and passes it to <code>importText</code> again.
Cleanup	For each scenario, it resets records to the initial 8 records. This app has no storage mechanism, so resetting records is enough.

Some things do not pass through e2e.js. There are 3: real drag and drop, saving a download, and encoding detection. The string that `e2e.js` holds is already decoded, so the stage before reading the file is not exercised. Encoding detection is contained in one function in that earlier stage, so a person checks it, or Playwright, which the instructor shows next, covers it. For the new format in the practical exam too, `e2e.js` passes the already decoded string to `importText` .

You may want to open `index.html` in an iframe and operate it, but with `file://` you cannot touch the contents of the screen across pages. So the test runs inside the same page. Selenium and Playwright need Node, so they cannot be installed on today's PCs. From the rows where the Layer column of `testcases.md` is E2E, decide on the 1 case that would hurt most if it failed.

Exercise 2: E2E test from import to reload

The working time is [26min]. The breakdown is: choose 4 cases [4min], have them written as Given, When, Then [2min], rewrite the Then lines [4min], have `e2e.js` built [3min], run it and compare the results [5min], isolate and fix the 1 that fails [5min], write the difference in warning codes [3min].

Purpose

The Worklog app has no way to save data, so it returns to the initial 8 records when you reload the page. The exported CSV contains all the records that were in the list at that moment. When you drop it onto the app again, rows identical to records already in the list come in a second time. You work out which requirement of the Import Specification governs this case, comparing it with `open-questions.md` from Exercise 5 on Day 2.

There are 4 scenarios. You rewrite the Then of 2 of them yourself, and you run all 4 with `e2e.js`.

What you bring from the earlier exercises

These are the rows of `testcases.md` whose Layer is E2E, including the 3 order and state rows you wrote in Exercise 6 on Day 2, the export from Exercise 12 on Day 2, and `WorkLogApp.importText`, which you split out in Exercise 9 on Day 2.

Files you use

- `tpro-work/testcases.md`
- `day3/data/worklog_2026-07-08.csv`
- `tpro-work/e2e-scenarios.md` You create it from scratch.
- `tpro-work/e2e.js` Agent creates it.
- `tpro-work/index.html`, `tpro-work/open-questions.md`, `tpro-work/decision-log.md`

Steps

- Choose 4 rows whose Layer is E2E in `testcases.md`, and write one line each on why you chose them [4min]
- Have Ask write the 4 scenarios in Given, When and Then form. Have it specify the data by row number in `worklog_2026-07-08.csv`, and write each Then as how many records appear where on the screen and how many lines the exported file has [2min]
- Rewrite the Then of 2 of the 4 scenarios yourself, 3 lines each, and save them in `e2e-scenarios.md`. Write how many rows are in the warning area, how many are in the list, and how many lines the exported file has [4min]
- Have Agent create `e2e.js` and add one line to `index.html` that loads it only when `?e2e` is used. Read the diff and Keep [3min]
- Open `index.html?e2e` and check the results of the 4 scenarios against their Then, one by one [5min]
- Have the AI narrow down the cause of the 1 scenario that failed to the scenario, the app or `e2e.js`. Have it fix only the place you decided on, and run the scenarios again [5min]
- List the warnings from the first import and the warnings after the reload, by warning code. Write 2 lines in `decision-log.md` on why some codes remain and why some disappear [3min]

Put the following conditions in the prompt for Step 4.

Element	What you write
When it is loaded	Only when the page is opened with <code>?e2e</code> . Run scenarios 1 to 4 of <code>#e2e-scenarios.md</code> in order

Element	What you write
Entry point	Keep the CSV rows as strings inside <code>e2e.js</code> and pass them to <code>WorkLogApp.importText</code> . Do not go through the first stage, which reads the file, or the character encoding detection
How to check	Check the counts in the warning area and in the list with assert, for each warning code
Export	Replace <code>URL.createObjectURL</code> to receive the Blob, and pass the result of <code>Blob.text()</code> to <code>importText</code> again
Cleanup	Reset records to the initial 8 records for each scenario
Not used	<code>iframe</code> and <code>type="module"</code>
How to show results	Show PASS and FAIL at the top of the screen, in the same form as <code>tests.html</code>

Split Step 6 into 2 round trips. In the first round trip, write which scenario failed, its Then and the actual value, and have the AI list the possible causes in 3 groups: how the scenario is written, how the app is implemented, and how `e2e.js` is built. Do not ask for a fix yet. In the second round trip, have it fix only the place you decided on, and do not let it change any other files.

What to think through yourself

- What do the 4 scenarios protect? Do they overlap with what the unit tests already protect?
- Where do you fix the 1 scenario that failed: the scenario, the app or `e2e.js` ?
- How will you handle rows identical to records already in the list when you reload? What will you do for now, before the answer to `open-questions.md` comes back?

Resulting files and where to save them

- `tpro-work/e2e-scenarios.md` The 4 scenarios. For 2 of them, the Then is written in your own words
- `tpro-work/e2e.js` Your own E2E script, loaded only with `?e2e`. In the practical exam, you add 1 scenario for the new format
- `tpro-work/index.html` One line that loads `e2e.js` is added
- `tpro-work/decision-log.md` 2 lines on the difference in warning codes are added

Check your result

- When you open `index.html?e2e`, the results of the 4 scenarios appear, and without `?e2e`, `e2e.js` is not loaded
- `e2e.js` runs the scenarios from `WorkLogApp.importText` and does not create a File
- The Then of the 2 scenarios you rewrote yourself is written as counts
- After the E2E run, the list is back to the initial 8 records
- You can say where to fix the 1 scenario that failed, and why
- `decision-log.md` has 2 lines on the difference in each warning code before and after the reload

If it does not work

Symptom	Fix
<code>WorkLogApp.importText</code> is not found	The split was not finished in round 2 of Exercise 9 on Day 2. The drop handling is inside an anonymous function. Cut out the stage that receives a string and put it on <code>WorkLogApp</code> , and include that in the request for <code>e2e.js</code>
The export starts a download and the run stops	The replacement of <code>URL.createObjectURL</code> takes effect after the export function is called. Have the replacement placed first
In the reload scenario, the number of rows in the list does not match the Then	Look at the Source column and <code>open-questions.md</code> , and decide yourself whether this is handled in the scenario, the app or the open questions. The instructor will not give the answer
The Japanese text is garbled and every row becomes a warning	<code>e2e.js</code> calls the first stage, which reads the file, instead of <code>importText</code> . Have it changed to run from the second stage

If time allows

- Add 1 more scenario besides the 4. Items you decided are Manual, such as how the file looks in Excel, are marked and left in `e2e-scenarios.md`, not automated

Exercise 3: Read e2e.js

About [10min]

Scenario

`e2e.js` is code written by Agent. Even when all 4 scenarios PASS, some may not check any of the counts written in their Then. You repeat in E2E the cross-check you did with the unit tests on Day 2.

Prompt to send

```
#e2e.js #e2e-scenarios.md
For each scenario, show in a table the line number of the assert that checks each line of Then.
For a Then line that is not checked, write None. Do not change any files.
```

Steps

1. Send the prompt above in Ask and have it produce the table.
2. Pick one row marked None and look in `e2e.js` yourself to see whether it really does not check that Then.
3. If it really does not, have Agent add an assert that checks only that Then, and reopen the page with `?e2e`.

Check your result

- `decision-log.md` has the Then line that was marked None and 1 line saying whether you added a check

Exercise 4: Break the code and see whether the tests fail

About [12min]

Scenario

When all tests PASS, you cannot tell whether the code is really protected or the tests are simply not looking at anything. You disable the 15-minute-increment check in just one place and compare how tests.html and ?e2e fail.

Steps

1. Copy `worklog-core.js` and keep the copy as `worklog-core_ok.js`.
2. Give Ask `#worklog-core.js` and ask "Which line is the if that rejects values not in 15-minute increments?" Then, right after the `if (` on that line, add `false &&` and save. Do not comment out the whole line. The braces would no longer match, and the whole file could not be loaded.
3. Reopen `tests.html` and `index.html?e2e`, and write one line each in `decision-log.md`: the case IDs that failed in the unit tests, and the FAIL lines in E2E.
4. Copy `worklog-core_ok.js` back over `worklog-core.js`, and check that `tests.html` is back to the original SUMMARY line.

Check your result

- `decision-log.md` has 2 lines on how the unit tests and E2E failed
- `worklog-core.js` is back to its original state, and the SUMMARY line is the same as in Exercise 1

Normally both fail. Rows that are not on the increments get imported and the count changes, so E2E fails too. A unit test points to the broken rule by case ID, while E2E only says that the count does not match. If the unit tests did not fail, `tests.html` has no case for 15-minute increments, so write the missing case in `decision-log.md`.

Division of roles with Playwright

This is a [6min] demo. `e2e.js` calls the `import` function directly, so real drag and drop and real downloads are not exercised. Playwright is the tool that checks those, and it needs Node. It cannot be installed on participants' computers, so the instructor runs only 2 scenarios on the instructor's computer to show it.

Tool	Path covered	When to run
<code>e2e.js</code>	From <code>importText</code> onward. It starts from an already decoded string	Every time you fix something
Playwright	It goes through receiving the file, detecting the character encoding and saving the download, all with real operations	Once before a release

On the instructor's computer, 2 scenarios are run: from `import` to the number of warnings, and from `export` to reload. When you put them next to the results of running the same 2 scenarios with `e2e.js`, you can see that the

counts are the same and the paths covered are different. The instructor also shows once how to have Copilot's Agent write a Playwright test.

While you watch, mark in your own `e2e-scenarios.md` the scenarios that can only be checked with real operations. In the final README, you keep them as Manual items. If you want to install it on your own computer too, you need administrator rights and Node. That means asking a department in your company that has such an environment.

Exercise 5: Add run results to the test report

About [14min]

Scenario

The test report on Day 2 was submitted with every case marked as not run. Today you ran tests.html and E2E, so you add the run results to the report and submit it again. For Manual items, write exactly whether you looked at them or not.

Steps

1. Paste the SUMMARY line of `tests.html` and the results of the 4 scenarios of `?e2e` into `report.md`.
2. Give Ask `#report.md` and have it rewrite the Not run and why section, split into the results you ran and the Manual items. You decide the pass/fail column yourself.
3. Recreate `report.html` and `report.pdf` with the same 4 steps as on Day 2.

Check your result

- `report.pdf` contains the run results, and FAIL and HOLD are written separately
- A HOLD row only means the specification is not decided yet, so write it separately as PENDING, not as a failure.

Skills and custom agents

The 4 stages of test case design you went through on Day 2, and the 3 lines of review points you sent to Ask in Exercise 12 on Day 2, disappear when you close the chat. Next month you would have to build the same text again. Make the set of steps and column definitions a skill, and make the reviewer a custom agent.

Type	Location	Contents	How it is invoked
Skill	<code>.github/skills/name/SKILL.md</code>	Steps and assets. Because it is a folder, you can put column definitions and samples in <code>references</code> and refer to them from the body	Loaded when Copilot looks at the description and judges the skill relevant. You can also call it yourself with <code>/name</code>

Type	Location	Contents	How it is invoked
Custom agent	<code>.github/agents/name.agent.md</code>	Its job, and the tools and model it is allowed to use	When the parent judges it necessary, it runs as a subagent in a separate context and returns only the result

In a skill's `SKILL.md`, name and description are required. The name uses only lowercase letters and hyphens and is the same as the folder name. For a custom agent, write description, tools and model in the frontmatter. If you limit tools to read-only ones, the reviewer cannot change files on its own by mistake.

There is also a `.prompt.md` that fixes one prompt, but you do not create it today. The official VS Code documentation shows a notice that it is deprecated and not loaded in Agent Host sessions, and that points you to migrate to skills. You build a skill from the start.

How the model is decided

Priority	Where it is decided
1	When you specify a name at the time of the call
2	The name written in the model field of <code>.agent.md</code>
3	The model chosen in the parent conversation

A child's model cannot go beyond the parent's cost limit. If you specify one that does, it is not run and a list of usable models comes back. Only the direction where the parent is the higher model and the child is the lighter one works. In Exercise 6, you hit this limit on purpose once.

`SKILL.md` follows the common specification of agentskills.io. Copilot also reads skills placed in `.claude/skills/`, so you can use the same assets as in other tools. Choose one instruction from today's conversation that you are likely to type again next month, and decide ahead of time whether it should be a skill or an agent.

3 locations

The files you create today are listed by when they are read. All 3 go under `.github`.

Location	Folder	When it is read	What you make today
Instructions	<code>.github/instructions/</code>	Automatically in every chat, and when you work with files that match <code>applyTo</code>	Instructions for tests
Skill	<code>.github/skills/</code>	When it is judged relevant, and when you call it with <code>/</code>	test-design, regress

Location	Folder	When it is read	What you make today
Agent	<code>.github/agents/</code>	When the parent calls it as a subagent	security-reviewer

A `.prompt.md` file that saves a single prompt also works, but it has been officially announced that `SKILL.md` is its replacement. Today you build a skill even for a single prompt.

Using /create-skill

1. In the model picker, choose a model that has at least one model above it and at least one below it.
2. In the input box, type `/create-skill`, describe the skill you want in 1-2 sentences in English, and send it.
3. If it asks for the name, when to use the skill and the files to reference, answer one question at a time. Some versions do not ask.
4. Read the body of `SKILL.md`, check that the steps match the assumptions you wrote, and then click Keep.
5. Type `/` in the input box and check that the skill name appears in the list.

Exercise 6: Turn case design into a skill and delegate the review

The working time is [26min]. The breakdown is: match the parent model and have the skill created [3min], add assumptions and column definitions [4min], try the skill [3min], create the agent [6min], run the two side by side [4min], try a model above the parent's [3min], add 1 item to the tests [3min].

Purpose

Turn the Day 2 case design flow into a skill. The AI writes the path it took as the steps, but it does not write the assumptions that a person corrected along the way. Find 3 places on Day 2 where you corrected the AI's output, and add them.

Next, turn the 3 lines of viewpoints you sent to Ask by hand in Day 2 Exercise 12 into a read-only reviewer. When you call the two with the same request, they come back in different ways. A skill is a procedure loaded into the parent's context, so its result appears in the parent's reply. Only security-reviewer, which runs as a subagent, comes back in a separate section. You see this difference on screen.

If the model setting is wrong, the subagent does not run. Make the mistake once on purpose and look at what comes back. For that you need a model above the parent in the list, so first set the parent to a model that is not at the top.

What you bring from the earlier exercises

You need the 4-step flow from Day 2 Exercise 6, `testcases.md` and `open-questions.md`, `security-notes.md` from Day 2 Exercise 12, and `model-memo.txt` from Day 2.

Files you use

- `tpro-work/.github/skills/test-design/SKILL.md` and `references/columns.md` Both are new files.

- `tpro-work/.github/agents/security-reviewer.agent.md` New file.
- `tpro-work/security-notes.md` , `tpro-work/testcases.md` , `tpro-work/open-questions.md`
- `tpro-work/worklog-core.js` , `tpro-work/index.html` , `tpro-work/tests/core.tests.js`
- `tpro-work/model-memo.txt`

Steps

1. In the model picker, set the parent model to one that is not at the top of the list, one that has at least one model above it and one below it. Have `/create-skill` create the test-design skill. The materials are `#testcases.md` , `#open-questions.md` and the 4 steps you went through on Day 2. Read the diff and click Keep [3min]
2. Read the `SKILL.md` that was created, and add 3 lines of missing assumptions yourself. Put the definitions of the 8 columns in `references/columns.md` and make the body refer to it [4min]
3. Apply `/test-design` to R-08 (holidays) in `#import-spec_v1.md` , and check that the cases come out as the column definitions say, with a Source for each [3min]
4. In the Command Palette, use `Chat: New Custom Agent` to create `.github/agents/security-reviewer.agent.md` . Write one line for description, and in the body write the 3 viewpoint lines from `security-notes.md` as instructions. For tools, list only read-only ones. For model, copy the name of a model below the parent from the picker's display [6min]
5. Tell the parent Agent to run security-reviewer as a subagent to review `toCsv` and the places that show text on the screen, and then to list the gaps in the R-07 cases with the test-design skill. Have it show the 2 results side by side without editing them, and do not let it change any files. Check that one section appears and that the skill's result appears in the parent's reply. Write the AI credits shown when you hover over the section as one line in `model-memo.txt` [4min]
6. Change the model of security-reviewer to a model above the parent, and send exactly the same request as in Step 5. Check that it does not run and that the models you can use come back. Then change model back [3min]
7. Pick one finding or gap, have it added to the Import section of `testcases.md` and to the tests, and check the result in `tests.html` . Have it take the expected value only from the row you added to `testcases.md` [3min]

In an agent file, write the following 3 items in the range enclosed by 3-hyphen lines, and write the body below them.

```
---
description: when you want it to be called, in one sentence
tools: read-only tools only
model: the model name exactly as shown in the picker
---
Below this line, write the 3 viewpoint lines from security-notes.md as instructions
```

What to think through yourself

- The 3 assumptions to add to `SKILL.md` . Which assumptions did a person correct along the way that the AI did not write?
- The description of security-reviewer. When do you want it to be called?
- Which model to assign to the child. Which one below the parent's model?

Resulting files and where to save them

- `tpro-work/.github/skills/test-design/SKILL.md` and `references/columns.md` In the practical exam, you apply `/test-design` to the new format.
- `tpro-work/.github/agents/security-reviewer.agent.md` In the practical exam, you have it review the part you added for the new format.
- `tpro-work/model-memo.txt` One line about the AI credits is added.
- `tpro-work/testcases.md` and `tests/core.tests.js` Each gains one item.

Check your result

- `SKILL.md` has 3 assumption lines you added yourself and refers to `references/columns.md`
- The output of `/test-design` has 8 columns and includes the Source
- tools in `security-reviewer.agent.md` lists only read-only tools, and model is exactly as shown in the picker
- One section for security-reviewer appears, and the test-design result appears in the parent's reply. You can explain in one sentence why a skill does not get its own section
- The parent's model is not the top one in the list
- You have seen that, when you specify a model above the parent, it does not run and the models you can use come back

If it does not work

Symptom	Fix
<code>/create-skill</code> does not appear	Your version is too old, or the feature is blocked by your organization's settings. If you have the same version as the instructor's machine, create the skill from New Skill File... in the Command Palette, and have Agent write the body from the same prompt. If your version is older, watch the demonstration on the instructor's screen while you create <code>SKILL.md</code> and <code>.agent.md</code> by hand, and check the section and the model with the instructor's screen capture
Custom agents and subagents do not appear in the list	It may be your organization's settings. You cannot fix this with your own settings, so raise your hand. You check this exercise with the instructor's screen capture
The parent is still on the top model, so you cannot specify a model above it	Move the parent model down one in the picker, and then send the same request again
Only one model appears because of your organization's settings	Use the instructor's screen capture instead of the check with a model above the parent's
<code>SKILL.md</code> is not read	name and the folder name are different. Make name only lowercase letters and hyphens, and the same as the folder name

Symptom	Fix
The subagent is not called	Check that the mode is Agent and that agent (runSubagent) is enabled in the tool selection in the input box
The name written in model has no effect	If it differs from the display by even one character, it has no effect. Copy it again from the picker
security-reviewer tries to make fixes	tools still contains edit tools. Review the frontmatter
You want the skill's result to get its own section too	In <code>SKILL.md</code> , writing <code>context: fork</code> makes the skill run in its own subagent, but this is an experimental feature and you must enable a setting. We do not cover it today

If time allows

- Add a third agent: one that builds a column mapping table first when it receives a CSV in a different format. Decide for yourself whether to use it in the practical exam.

Exercise 7: Turn the regression check into a small skill

About [16min]

Scenario

Each time you fixed the import, you reopened `tests.html` and `?e2e` and copied the results. Typing the same prompt every time is a sign to move it into a file. Make it a skill that needs no reference files and has just one procedure.

Prompt to send

```
/create-skill Create a skill named regress that compares the SUMMARY line and ?e2e results I paste with the previous results and reports only the rows that changed.
```

Steps

1. Send the prompt above.
2. Open the `.github/skills/regress/SKILL.md` that was created, read it to check that it says what to compare against and the form of the report, add any missing lines, and save.
3. Paste the SUMMARY line, send `/regress`, and compare the report that comes back with the numbers in the Exercise 5 test report.

Check your result

- `.github/skills/regress/SKILL.md` exists, and `/regress` returns a side-by-side comparison

A skill cannot open the browser. Paste the SUMMARY line and the ?e2e results yourself, and then send /regress.

Exercise 8: Instructions for writing tests

About [14min]

Scenario

Since Day 2, every time you had it write tests, you have said "Take the expected values from testcases.md without looking at the implementation." If you make this an instructions file that applies only when you work with files in the tests folder, it takes effect even if you forget to say it.

Prompt to send

```
/create-instructions Create instructions that apply only when working with files in the tests folder.  
Take expected values from testcases.md and do not copy them from the contents of worklog-core.js.
```

Steps

- 1. Send the prompt above.
- 2. Check that applyTo at the top of the created file is tests/** . If it is not, fix it and save.
- 3. Send Agent "Add one R-08 case to tests/core.tests.js" and check whether the file you just created appears in the list of referenced files above the reply.

Check your result

- There is an instructions file for tests under .github/instructions, and its applyTo is tests/**
- It appears in the list of referenced files when you have tests written

If it does not work

Symptom	Fix
/create-instructions does not appear	Your version is old. Create a new file in .github/instructions by hand and have Agent write the body
The instructions are read in every conversation	applyTo at the top is missing. Open and check the first 3 lines

Exercise 9: Refine the skill

About [12min]

Scenario

A skill is at its roughest on the day you make it. If you use it and keep adding the points you fixed, neither you next month nor the person next to you will get stuck at the same point.

Steps

1. Apply /test-design to R-09 (duplicates) and find one point you want to fix in the test case table that comes back.
2. Turn that point into a one-line rule so you do not have to fix it next time, and add it inside the matching step of SKILL.md. Write it as "do ...", not as "do not ...".
3. Send the same request again and check that the line you added is working.

Check your result

- SKILL.md has a line you added after using the skill
- In the second output, that deviation is gone

How certification is assessed and how to submit

Certification is not meant to fail anyone. But one instructor reads the practical exams of 20 people from text alone, so we decide in advance what to look at. Today you submit only text that puts your thinking into words. You do not submit code or real data. Instead, you write your decisions and evidence, the SUMMARY line from `tests.html`, and the counts of the first import and the counts by warning code, under fixed headings.

Time	What is checked
Literacy Check	Your knowledge of the 3 days.
Practical exam task	Your decisions and the evidence for them. The check is whether the counts that follow from Import Y/N and Warning Y/N in the judgment table agree with the first-import counts you reported
Feedback session	Whether you can resolve the comments you get back on your own

What is not checked: the total values and how the code looks. If decisions differ, the totals naturally differ, and if the decisions and counts agree, the instructor can verify the totals on their side. Counts are compared only for the first import. After the reload, only whether an explanation is written is checked.

How to submit

- Submit through the single submission form that the organizer prepares. The instructor posts the URL in the Zoom chat. The form has only 2 questions, "Name" and "Text to submit", and you send both your Literacy Check answers and your practical exam submission to this form
- You submit the practical exam twice. In the first [15min], send only the judgment table, marked "Early". At the end, send the whole thing

- You send the judgment table first so that the instructor can read it during the practical exam and send back comments. The comments arrive in a private Zoom chat
- Use the format of the 8 headings and the warning code table in `submit_template.md`, which is in the ZIP for the practical exam. During this part, you only check where the headings and tables are on the instructor's screen. Everyone uses the warning codes from the same table

Decide which heading of the submission template you find hardest to write. Shift your practical exam time toward it. The instructor answers questions about how Certification results are handled after checking with the organizers.

Literacy Check

It takes [24min]. The breakdown is: answer the 20 questions [20min], review the unanswered ones [2min], paste into the submission form [2min].

1. Open `day3/02_literacy-check.html` in the browser. It has 20 multiple-choice questions
2. You may look at the slides from the 3 days, this Hands-on Guide, your own notes and the contents of `tpro-work`. The only thing to avoid during this time is asking Copilot
3. When you finish, review the unanswered questions and click "Compile answers". Paste the text that appears into "Text to submit" on the submission form, write your name in "Name", and send it

Q1 asks how to choose between Ask, Agent and Plan, which was covered on Day 1. The questions on how `AGENTS.md` is handled by default, the difference between `.prompt.md` and `SKILL.md`, and how a subagent's model is determined also check what was covered over the 3 days.

Symptom	Fix
The button does not respond	Another app opened when you double-clicked the file. Right-click it and reopen it in the browser
You sent the form without writing "Name"	On the same form, write "Name" and send it again. The instructor uses the one that arrives later

If you finish early, reread today's `decision-log.md`.

The 3 most-missed questions and the practical exam conditions

In the [10min] after answering, the instructor gives back the correct answers and reasons, 2 lines each, for the 3 most-missed questions only. Individual scores are not discussed. The instructor then goes straight into explaining the practical exam scenario and conditions, and hands out the ZIP for the practical exam for the first time here. Extract it to `day3/jitsugi`.

Practical exam task

It takes [120min]. No new techniques appear. You import using only what you built over the 3 days: the instructions, the contract, `tests.html`, `e2e.js`, the skills and `security-reviewer`. You then submit your decisions in

writing. The conditions and the prompt patterns for each stage are in `docs/practical-exam-guide.html` in the ZIP for the practical exam. Extract it to `day3/jitsugi`.

Stage	Time	Good enough when
1 Early judgment table	[15min]	You filled in J1 to J9 and sent only the judgment table
2 New-format import	[22min]	You matched one row with your hand calculation
Group check 1	[5min]	If your import does not work, get the skeleton
3 New-format tests	[22min]	You added 10 or more
4 One E2E scenario	[14min]	One scenario ran with ?e2e
Group check 2	[5min]	You checked the headings of the submission
5 Delegate to the reviewer	[8min]	You wrote one line on a finding
6 Collect observed values	[9min]	You copied the counts and the SUMMARY line
7 Finish and submit	[18min]	You sent everything

If you are not confident about the implementation, you may get the instructor's skeleton from the start. It costs no points. Just note that you used it under "Harness used" in your submission. You submit twice: the judgment table alone ("Early"), and the finished whole. Do not put totals, real client names, real data or code in the submission text.

Instructor feedback and the Certificate of Completion

It takes [60min]. During the break just before, the instructor checks everyone's submission and sends back comments by private Zoom chat.

Comments you receive

Each comment is one fixed-format sentence. It says only which row of the judgment table and which code in the first import counts do not match, and it ends with "Please check one of the two." It does not mean that your decision is wrong. It means that you wrote either the decision or the observed values incorrectly. If there is no mismatch, you receive the single word "Verified".

How the feedback session runs

1. The instructor explains what will and will not be checked, and reads out the names of the people who go to the feedback room: those with 2 or more mismatches or with missing template headings, up to 8 people [4min]
2. The feedback room is a Zoom breakout room, and each person gets 4 minutes. You share your screen, and the instructor tells you out loud how to fix it, without touching the keyboard. You do the typing yourself. If you are in the main room, handle the comments you received, and either fix one open issue or add one line explaining why you do not [32min]

3. Everyone returns to the main room and resubmits the submission text, with one line added on how you handled the comments, on the same form [4min]
4. The instructor gives the whole group feedback on just one stumbling point shared by 3 or more people [4min]

Certificate of Completion

It takes [6min]. There is no pass or fail by score. Completion means that you can check what you built over the 3 days yourself and submit it. Because the course is online, certificates are not handed over in person: the instructor shows one on screen, and the organizers send the PDF later. Follow the organizers' notice for where and when it is sent. If your name is spelled incorrectly, write it in the chat on the spot.

The README that links the 3 days

It takes [10min]. What stays in `tpro-work` after the 3 days is not only a working app. It also holds the instructions, the contract, the test runner, E2E, the skills and the agents. When the hours CSV arrives at the end of next month, you will have to start over from the chat again unless you have written down which files to open in which order.

1. The instructor shows the deliverables of the 3 days on one map. Day 1: `index.html` and the instructions. Day 2: `testcases.md` , `worklog-core.js` and `tests.html` . Day 3: `e2e.js` , test-design, regress and security-reviewer [2min]
2. In `tpro-work/README.md` , write 4 lines of steps for next month: what to drop onto which file, what to check with `tests.html` and `?e2e` , and where to make changes if the format changes [4min]
3. Following the steps in the README, open `tests.html` and `index.html?e2e` , and check that both show results [2min]
4. Right-click the `tpro-work` folder, compress it into a ZIP, and put it in your own save location. Check that `.github` is included [2min]

Do not let AI write just these 4 lines. If AI writes them, the places where you were unsure disappear. If `.github` is not in the ZIP, you copied file by file. Compress the whole folder instead. On a Mac, a folder that starts with a dot, such as `.github` , is not visible in Finder, so press `Cmd+Shift+.` to toggle the display and check.

If time allows, add one line at the end of the README naming the file to fix when the client's answer to `open-questions.md` arrives.

Troubleshooting

Symptom	What to check	Fix
The screen is blank	The first line in the Console (F12)	It is the load order or <code>type="module"</code> . Put <code>lib/papaparse.min.js</code> first, and do not use module

Symptom	What to check	Fix
Agent writes everything back into one file	Whether <code>copilot-instructions.md</code> was saved, and the structure lines in <code>AGENTS.md</code>	Fix both, then send the request again in a new chat
The expected values in the tests become the same values as the implementation's	References in the reply	If an implementation file is listed, close its tab, remove the attachment, and send the request again in a new chat
The diff you clicked Keep on included <code>tests/</code>	File names in the diff	Undo the change, and send the request again with "Do not edit anything under <code>tests/</code> " at the top
The numbers in the total are joined side by side	Whether hours is still a string	Describe only the symptom and have it fix the problem
Japanese text in the CSV is garbled	The encoding shown in the Status Bar at the bottom right	Choose Reopen with Encoding. On the app side, check that the encoding is handled only in the first stage
Nothing happens with <code>?e2e</code>	The address bar, and the line in <code>index.html</code> that loads it	Check the spelling of <code>?e2e</code> and the path to <code>e2e.js</code>
The menus for skills and custom agents do not appear	The VS Code version	Check through the instructor's on-screen demonstration and screen capture instead
You fell behind during an exercise	Recovery slot	Raise your hand. The instructor sends you the relevant file by private Zoom chat. Record the names of the files you used in <code>decision-log.md</code>

VS Code chat has 3 modes: Ask, Agent and Plan. The Edit mode that existed before is gone. The source is <https://code.visualstudio.com/docs/agents/concepts/agents>. Skills are described at <https://code.visualstudio.com/docs/agent-customization/agent-skills>, and custom agents at <https://code.visualstudio.com/docs/agent-customization/custom-agents>.

What you take home today

Check that the `tpro-work` folder contains the following.

Location	Contents	Use next month
<code>README.md</code>	4 lines of steps for next month	Open it first when the CSV arrives

Location	Contents	Use next month
<code>.github/copilot-instructions.md</code>	Instructions that include the 2 lines you rewrote on Day 2	Added automatically to every later chat
<code>docs/core-contract.md</code>	The contract for the 7 functions	When the format changes, hand it over before you have a plan made
<code>worklog-core.js</code> , <code>worklog-config.js</code>	Import and export logic, and the constants for the category map and holidays	Fix config when the holidays or the category map change
<code>tests.html</code> , <code>tests/</code>	The test runner you built yourself, and its tests	After a fix, reopen it and check FAIL_IDS
<code>e2e.js</code> , <code>e2e-scenarios.md</code>	The E2E that runs with <code>? e2e</code>	Rerun the flow from import to reload
<code>.github/skills/test-design/</code>	The 4 steps of case design and the column definitions	When a new specification arrives, apply <code>/test-design</code> to it
<code>.github/agents/security-reviewer.agent.md</code>	A read-only reviewer	Call it every time you make a change that puts a value that comes from outside on the screen or in an export
<code>testcases.md</code> , <code>open-questions.md</code> , <code>decision-log.md</code> , <code>security-notes.md</code>	Cases, questions for the client, a day's decisions, and viewpoints	When the client's answer arrives, fix the matching row in open-questions.md
<code>.github/skills/regress/</code>	A small skill for regression checks	After a fix, paste the SUMMARY line and send <code>/regress</code>
<code>.github/instructions/tests.instructions.md</code>	Instructions that apply only to the tests folder	Read automatically when you have tests written
<code>report.pdf</code>	The test report with the run results added	When the next format arrives, rebuild it in the same way

Rules to follow

Do not use real data from your own work in the exercises. The data used today and the requester of the Import Specification are all fictional. They have no connection to real companies, departments or people. In the Day 1 survey, some participants said that they are usually prohibited from giving code to AI at work. When you use this at work, your organization's rules take priority.

Before you paste into the submission form, generalize your text with these 4 rules.

Rule	How to apply it
Generalize client names, product names, project names and partner names to the industry or process level	Write something like "importing data into an attendance system in the finance industry", not a specific company or system name
Do not use real data, real files or screenshots	Do not paste your CSV, code or screenshots. Write only the counts, warning codes and decisions
Avoid specific values for figures and specifications, and write at the level of purpose, issues and expected effects	In open issues, do not write actual values from your workplace specifications. Write something like "the import specification still has undecided points"
Before you submit, check yourself that the content is safe to send outside the company	Read it again when you finish. Leave out any part you are unsure about.

The same applies when you put the harness you took home into a shared folder at your company. Do not write client names or actual system names in `SKILL.md` or `security-reviewer.agent.md`. If you are not sure about an item, do not put it there; keep it with you.

This material is a copyrighted work created by Givery, Inc. Participants may use it only for their own learning and for reference in their own work. Do not distribute it outside the company, post it on social media, or share it with third parties.



Givery, Inc. Do not redistribute this material.