



HANDS-ON GUIDE

IT track Day 1 Hands-on Guide

Hands-on AI-Assisted Development, IT track, Day 1

This guide collects the hands-on parts of Hands-on AI-Assisted Development, IT track, Day 1, so you can review them on your own PC. The slides keep moving forward, but you can go back in this guide at your own pace. When you are unsure about a step or get stuck, open the page for that exercise.

Contents

What you do today

What you use

Setup Step 1: Prepare VS Code and your folder

Setup Step 2: Create a GitHub account

Setup Steps 3 and 4: Accept the invitation and sign in to VS Code

Setup Step 5: Connection check

Key points for the morning

Exercise 1: A tour of the ways into Copilot

How to generalize client information

Exercise 2: Rewrite a prompt with the prompt pattern

Exercise 3: A Worklog app from a 5-line spec

Exercise 4: Add a filter by category

Exercise 5: Add total hours by category

Exercise 6: Switch models and send the same request twice

Exercise 7: Add an instructions file and compare before and after

Where to look when things go wrong

What you take home today

Rules to follow

What you do today

Today, you set up GitHub Copilot in VS Code on your loaner PC so that it answers in English, and you build a working web app from a 5-line spec. Then you have Copilot read the provided Worklog app first, add two features, and go on to switching models and the instructions file.

Time	Steps	Minutes
Morning	Environment setup (Setup Steps 1 to 5)	[100min]
Morning	The world of AI-driven development (listening only)	[30min]
Afternoon	Copilot basics and how to generalize client information (Exercises 1 and 2)	[95min]
Afternoon	Break	[10min]
Afternoon	A working web app from a 5-line spec (Exercise 3)	[60min]
Afternoon	Break	[10min]
Afternoon	Add features to an existing app (Exercises 4 and 5)	[75min]
Afternoon	Break	[10min]
Afternoon	Choosing models, the instructions file, and review (Exercises 6 and 7)	[90min]

The total is 480 minutes. The lunch break comes after the AI-driven development session. The instructor will tell you its length on the day.

Today has three goals. Copilot answers in VS Code. You can choose between Ask, Agent and Plan, and read the diff before you click Keep. You can have Copilot read the provided app first and then add features. Everyone must reach the first goal.

Put every file you create today in the tpro-work folder on your desktop. If your files end up in different places, you will not find your own work in the afternoon exercises.

What you use

You write code in VS Code and always run it in the browser. You do not start a server. You do not use Python, Node.js or the command line. You do not need administrator rights.

Tool	Role
VSCode	Where you write code. Version 1.116 and later include Copilot Chat by default.

Tool	Role
GitHub Copilot	The engine behind suggestions, chat and agents. It runs inside VS Code.
Browser (Edge, etc.)	Where the app runs. Double-click a saved HTML file to open it.

Contents of the handout ZIP

Extract the handout ZIP to your desktop with the name tpro-work. If the extracted folder contains another tpro-work folder, move its contents up one level. If your PC does not let you write to the desktop, create tpro-work directly under your Documents folder. Wherever this guide says "tpro-work on your desktop", read it as the place where you extracted the files.

Contents	Used for	When you use it
worklog-app/index.html	The existing app that records four columns: date, category, description and hours	Morning and afternoon
worklog-app/README.txt	A list of what this app can and cannot do yet	Afternoon
spec5.txt	The 5-line spec you paste into the chat in the afternoon	Afternoon
worklog-complete.html	The finished version of the app you build in the afternoon. Use it as a replacement if generation fails	Afternoon (only if needed)
lib folder (3 files)	xlsx.full.min.js, papaparse.min.js, chart.umd.js. About 1.1 MB in total	Day 2 and later
README.txt	Where to extract the files and how to open them	Any time

Files you create in this folder today

None of these are in the handouts. Create them yourself during the exercises and put them in the tpro-work folder.

File	Exercise where you create it
notes.txt	Exercise 1. Write one line on where the explanation differed from your understanding.
prompt.txt	Exercise 2. Fill in the 5 lines of the prompt pattern.
worklog.html	Exercise 3. The app you build from the 5-line spec.
Additions to worklog-app/index.html	Exercises 4 and 5. Add a filter field and a totals table.
model-memo.txt	Create it in Exercise 5 and write the cross-check results. In Exercise 6, add how the models differed.
before.txt / after.txt	Exercise 7. Paste the answers from before and after you add the instructions file.

File	Exercise where you create it
.github/copilot-instructions.md	Exercise 7. Create a .github folder directly under tpro-work and put the file there.
AGENTS.md	Extra task for Exercise 7. Put it directly under tpro-work.

Names of the screen areas used today

Today, only these 5 areas of the VS Code window are named.

Name	Location	How you use it today
Activity Bar	The column of icons on the far left	You only click Explorer and Chat.
Explorer	The contents of the open folder	Lists the files in tpro-work.
Editor	Center	Suggestions appear as light gray text.
Chat view	Right side	Today's main tool. You also start signing in from here.
Status Bar	Bottom edge	Shows the language of the open file. Check it when suggestions do not appear.

Setup Step 1: Prepare VS Code and your folder

About [8min]

Scenario

On your first day with a client's PC, you first decide on one place to work. Today, you keep every file you create in tpro-work on your desktop. If files are scattered, you will not find your own work in the afternoon exercises.

Steps

1. Start Visual Studio Code from the Start menu. If you cannot find it, raise your hand.
2. Open Help > About and check that the version is 1.116 or later.
3. Extract the handout ZIP to your desktop and check that the folder name is tpro-work.
4. Choose tpro-work from File > Open Folder. Check that worklog-app and lib appear in Explorer on the left.

Open the folder, not a single file. The folder you open is where Copilot starts reading nearby files. If you open a single file, Copilot does not get the content of other files.

Check your result

- tpro-work appears in the title bar or at the top of Explorer

- Explorer shows two folders: worklog-app and lib
- The About dialog shows version 1.116 or later

No deliverable. When you see all three on your own screen, move on.

If it does not work

Symptom	Fix
The version is older than 1.116	The Chat view does not appear. Install GitHub Copilot Chat from Extensions. You do not need administrator rights
You cannot write to the desktop	Create tpro-work directly under your Documents folder. Wherever the guide says desktop, read it as that folder
You opened a single file	The AI cannot read the files around it. Use File > Open Folder again
There is a tpro-work folder inside tpro-work	The extracted folder is nested twice. Move its contents up one level

If time allows

- From the View menu, open Appearance and look for where to switch the display language. Keep the display language in English. It does not affect today's steps.
- Open README.txt in worklog-app and read the list of what the app cannot do yet. You add features to it in the afternoon.

Setup Step 2: Create a GitHub account

The instructor will tell you the time on the day. This step takes the longest this morning.

Scenario

Copilot seats go to individual GitHub accounts. Today's invitation goes to your company email address, so you need an account with that address to get a seat.

Steps

1. Open github.com in the browser. If you are signed in with a personal account, sign out first.
2. Click Sign up and enter the email address your company gave you. Invitations do not reach personal addresses.
3. A password of 15 characters or more passes the rules without trouble. Follow the requirements shown on the screen on the day.
4. Choose a username of letters and numbers that no one else uses. Decide quickly: your family name in romaji plus some numbers.
5. Solve the puzzle on screen, then enter the verification code sent to your email.

Check your result

- Clicking the icon at the top right of github.com shows your username
- Your company email address is listed under Settings > Emails

No deliverable. You are done when your username shows at the top right after you sign in.

If it does not work

Symptom	Fix
The verification code does not arrive	Check your spam folder, then your company mail filter. If it does not arrive in 5 minutes, raise your hand
The username is taken and red text appears	Add two digits and it will pass. Do not spend time on this; decide quickly
The puzzle is slow	The network is just busy. Wait and do not close it
You are still signed in with an existing personal account	Sign out and create the account again

If time allows

In Settings, open the Copilot item in the left menu and see that nothing is assigned yet. The display changes in the next exercise.

Setup Steps 3 and 4: Accept the invitation and sign in to VS Code

The instructor will tell you the time on the day.

Scenario

A seat is assigned in two stages: you accept the invitation, then you sign in from VS Code. If a different account gets mixed in here, you accept the invitation but still cannot use Copilot. This is the most common problem in the course. Use the same account for accepting and for signing in.

Steps

1. Open the invitation email we sent in advance, with GitHub Copilot in the subject. If you cannot find it, tell the instructor.
2. First, sign in to the account you just created in the browser. If you stay signed in to another account, the invitation is accepted for the wrong account.
3. Open it from the button in the email and accept the invitation.
4. Go back to VS Code and open the chat from the Chat icon in the Activity Bar on the far left, or from the View menu.

5. Click Sign in in the view. A browser opens. Check that it is the account you used to accept the invitation, then authorize.
6. In the "Open Visual Studio Code?" dialog, click Open.

Check your result

- GitHub Settings > Copilot shows your seat. Menu names may differ on the day; follow the screen
- The Sign in button is gone from the VS Code Chat view, and you can type in the input box
- The account icon at the bottom left of VS Code shows your company email account

No deliverable. If you can type in the input box, Copilot is ready to use.

If it does not work

Symptom	Fix
The acceptance link says there is no invitation	This is almost always the wrong signed-in account. Sign out and sign in again
Your company email rewrites the link and it does not open	Copy the URL and paste it into the browser's address bar
You are stuck at the browser dialog that asks to return to VS Code	It is fine to click Open
The Chat view does not appear	Open it from the View menu. If it still does not appear, check the version. On versions older than 1.116, install the extension
Sign-in fails because of the company proxy	This cannot be fixed on the spot. Raise your hand. We record your PC and the symptom, and today you follow along on the instructor's screen

If time allows

In Settings > Copilot, find the setting that blocks suggestions matching public code. We cover it in the afternoon.

Setup Step 5: Connection check

About [9min]

Scenario

The afternoon exercises assume these three checks pass. If one fails, you cannot try anything on your own PC in the afternoon. You can judge each check yourself without the instructor coming around.

Steps

1. In Explorer, open index.html in worklog-app. If you ask about an empty folder, you will not get a useful answer. Open a file with content before you ask.
2. In the Chat view, type "What does the app in this file do? Answer in English." and send it.
3. Create a new test.html in tpro-work and save it. Then type html in the editor and wait a few seconds.
4. Click the model name below the chat input box and just check that a list opens. Switching models comes later in the afternoon.

Check your result

- The chat answers in English. Whether the content matches the actual code does not matter here
- Typing html in test.html shows a suggestion in light gray text
- Clicking the model name shows a list of several names

You now have one test.html in tpro-work on your desktop. It can be empty.

If it does not work

Symptom	Fix
The answer is not in English	Add "Answer in English." to the end of the prompt
Suggestions do not appear	Usually the file name is only test. Save it again with .html and check the Status Bar too
No answer comes back	The network is often busy, so wait 30 seconds and send it again. If it still does not come, raise your hand
The answer has nothing to do with index.html	Check that the file was open when you sent the prompt. If it is closed, Copilot does not get its content

If time allows

If you finished all three, go on to the tasks below without waiting.

Task	Steps	Check your result
A	Have the chat explain index.html in worklog-app, and find places in the answer that differ from the actual code	You can name at least one difference out loud
B	Open the model picker in the chat input box and write down the model names you can choose	You noted at least 3 model names. You use them later in the afternoon
C	Open index.html in the browser, press F12, and check the Console for errors	You confirmed that the Console shows no red lines

Task	Steps	Check your result
D	In GitHub Settings > Copilot, find the setting that blocks suggestions matching public code	You opened that settings screen

It also helps to send the same question again and see that the reply differs from the first one. The second half of the morning explains why.

Key points for the morning

In the second half of the morning, you stop and listen. Keep these 3 points in mind before the exercises, and the afternoon steps will make more sense.

An LLM splits the text you send into units called tokens and keeps choosing the next likely word by probability. This is why the same question can get different answers. It is not a bug. Nothing in it cross-checks whether the content is correct.

Information that is not in the context is not used. Only the open file, the selected range, the conversation so far and files you attach explicitly count as input. Internal company rules, client specifications and run results are not passed to the AI. If you ask with Ask, you get a plausible guess. Agent may search for and read files on its own, so a file you did not mean to share can end up in its input.

Inline suggestions, chat and agents use the same AI, but they differ in how much work you hand over.

Feature	What you hand over	What you do
Inline suggestions	A few lines continuing from the cursor	Accept or dismiss it right away
Chat	Questions about a selected range or one file	Decide what to pass in and verify the reply
Agent	Edits across several files, and redoing them	Set the scope before it starts and read the diff it produces

In chat, you open the file and paste it yourself. In an agent, the AI opens the files itself.

A line you approve without reading becomes a line you wrote. Because we do not use Git today, there are only 2 ways to go back: Ctrl+Z and undoing the chat's changes.

Exercise 1: A tour of the ways into Copilot

About [15min]

Scenario

At a client site, you usually start by reading what is inside the screens you took over. If you fix something without knowing what is in it, things break. Here you have Copilot read the provided Worklog app and get to the point where you can explain it in your own words. You also click through to the Agent switch you use later, so you remember where it is.

Operations to learn first

In the chat input box, Enter sends and Shift+Enter inserts a line break. Some people get flustered when a long text is sent before they finish, so learn this first.

Type / in the input box and a list of available commands appears. You do not need to memorize the names. You use these 4 today.

```
/explain  Explain what the selected code does
/fix      Show the cause of an error and how to fix it
/tests    Suggest tests for the selected code
/clear    Discard the chat history and reset the context
```

You switch modes with the Ask, Agent and Plan dropdown below the chat input box. Ask only answers questions and does not change files. Agent decides which files to open and fix, and rewrites them. Plan only produces a work plan and steps and does not change files. Agent's changes are not final until you read the diff and click Keep. If a change is wrong, go back with Undo. Keep means you accept the lines as your own. This dropdown is the only place to switch. Its position changes with the version, so if you cannot find it, look both above and below the input box.

Steps

1. In the tpro-work folder you opened this morning, open the worklog-app folder, then open index.html in the editor.
2. In the script tag, select the whole render function, line by line.
3. Type the following line in the chat input box and press Enter to send it.

```
/explain Explain what this function does in 3 lines, for someone reading it for the first time
```

4. Read the explanation. Create a new notes.txt in the tpro-work folder and write one line about one point that differs from your understanding or that the explanation did not cover well.
5. Open the Ask, Agent and Plan dropdown below the chat input box and choose Agent. When the display changes to Agent, switch back to Ask. Do not send anything in this step.

Check your result

- The /explain reply is in English and is about the render function. You can read from it that the function rebuilds the table and lays the rows out again
- notes.txt in the tpro-work folder has at least one line about a point that differed from your understanding or that the explanation did not cover well

- The dropdown below the chat input box changed to Agent, and you then switched it back to Ask

You end up with a notes.txt in the tpro-work folder on your desktop, with at least one line written in it. You do not create any other file. You do not change index.html.

If it does not work

Symptom	Fix
/explain does not appear in the command list	The list of commands changes with the version. With the code still selected, type "Explain the selected code in 3 lines" in plain English and you get the same result
The reply explains the whole file	The selection was lost. Select the function again, send /clear once, and send the request again
You cannot find the dropdown	Look above and below the input box. The position changes with the version. If you still cannot find it, raise your hand
The answer is not in English	Send "Answer in English" as a follow-up. The conversation carries over
You changed index.html by mistake	Undo with Ctrl+Z. You use this file in the later exercises

If time allows

- Send /explain on the same render function again and check that the wording differs from the first time. Even if the wording changes, check whether the substance is the same.
- Send /tests on the same selection and only check whether the suggested tests include "when no records have been entered". Do not implement anything.
- In the script in index.html, write the comment `// A function that returns the total hours`, press Enter, and watch the gray inline suggestion appear. Press Esc to dismiss it. Do not change the file.

How to generalize client information

Before Exercise 2, you decide which form of information is safe to enter. You decide it first because, if you learn the prompt pattern first, the sentence you write in practice may contain the client name as it is. All of today's exercises use dummy topics, so you will not break the rules here. The risk starts tomorrow, when you are back at your own desk.

Rule	Example of generalizing
Generalize client names, product names, project names and partner names to the industry or process level	Replace "Company A's order management system" with "a core system in the distribution industry"

Rule	Example of generalizing
Do not use real data, real files or screenshots	Do not paste a production CSV. Create a dummy file yourself with the same column meanings and pass that instead
Avoid specific values for figures and specifications, and write at the level of purpose, issues and expected effects	Replace "process 30,000 records a day in 30 minutes" with "the daily record count is high and processing time is a problem"
Before you submit, check yourself that the content is safe to send outside the company	Read it through once yourself before you press Send

The test is whether a search for that word could identify the client. If it could, generalize it. Do not assume that someone will stop you later.

Exercise 2: Rewrite a prompt with the prompt pattern

About [6min]

Scenario

When you use Copilot at your own desk from tomorrow, the biggest risk is pasting a client name or table name as it is when you are in a hurry. While you are calm, you practice writing in generalized form once. You do not send what you write, and you do not submit it.

The 4 elements of a prompt

Of the 4 elements (role, context, constraints and output format), constraints and output format change the result the most. Without constraints, you get code that does not run in today's environment. Without an output format, the whole file comes back and you cannot see what changed. You do not need to remember the order of the 4. Before you send, ask yourself only: "Did I write the constraints and the output format?"

Save the following 5 lines as prompt.txt in the tpro-work folder on your desktop. Today you keep adding prompts to this file.

```
You are a JavaScript code reviewer.
The target is a single HTML file that I open and run directly in the browser.
The symptom is that the numbers in the total hours field are lined up side by side instead of being added.
The constraint is that no external library is added.
Return the output in this order: 3 candidate causes, then only the corrected lines.
```

You replace 5 parts: the role, the target and environment, the symptom, the constraint and the output format. Only the symptom line changes each time. The other 4 lines can stay almost fixed. Adding one more constraint line works better than polishing the role line.

Steps

1. Open prompt.txt in the tpro-work folder on your desktop. If you have not created it yet, create it now and copy the 5 lines above into it.
2. Rewrite the 5 lines for the work you are involved in now. Generalize client names, product names, project names and partner names into industry and process terms.
3. When you finish, read it through once and check that no word remains that would let a search identify the client.
4. If one remains, replace it with a generalized word and save the file. Do not send it.

Check your result

- Of the 5 lines, the constraint line and the output format line are not empty
- No proper nouns such as company names, product names, system names or personal names remain. You have checked yourself that a search for those words would not identify the client
- The symptom line states what you can see, not the cause. It reads "the numbers appear side by side in the total field", not "reduce is broken"

You end up with prompt.txt in the tpro-work folder on your desktop, with all 5 lines filled in your own words. You do not send or submit it. You only keep it on your PC.

If it does not work

Symptom	Fix
You cannot think of what to write	Recall one question you asked someone in the past week and write that. Do not spend time looking for a topic
It feels like the question no longer makes sense once you generalize it	A dummy with the same structure is enough. Keep the meaning of the columns and change only the names
You cannot think of a constraint	Write the constraints of today's environment as they are: no external libraries, and everything in a single HTML file
You cannot find prompt.txt	Create it now, in the tpro-work folder on your desktop

If time allows

- Write the same request in two versions, one without the constraint line and one with it, and put them side by side. See which one you would rather send.
- Write only the symptom line in 3 ways: one that asserts a cause, one with facts only, and one with facts plus the steps to reproduce. Check that the third one works best.
- Send what you wrote only after you are back at your own desk. Before you send it, read it again and check that it is safe to send outside the company.

Exercise 3: A Worklog app from a 5-line spec

About [20min]

Scenario

If you can show a requirement a client gave you verbally as a working app the same day, the discussion moves faster. First, you feel for yourself how long it takes to go from an English spec to a working app.

One cycle of generating and checking

Write the spec, have the AI generate the app, open it in the browser, and describe what does not match. These 4 moves make one cycle. A cycle counts only if you got as far as opening it in the browser. Do not ask for the next change without opening it.

You write in VS Code and run in the browser. You do not start a server. After editing, press Ctrl+S to save and check that the dot next to the file name on the tab has disappeared. Then double-click the file in File Explorer. If the address bar starts with file:///, the file opened correctly.

Because the file is opened with file:///, JavaScript code that loads external files does not work. On Day 2, when you read CSV or Excel files, you pass them in by dragging and dropping them onto the page.

The 5-line spec to paste

Copy tpro-work/spec5.txt as it is and paste it into the chat. Do not retype it or change it.

1. Put one form for entering Date, Owner, Phase, Task and Time (min)
2. The "Add" button adds the input as one row to the list table below
3. Below the list table, show the total time in minutes
4. If any field is empty, do not add the row, and show "Some fields are empty" below the form
5. Put the HTML, CSS and JavaScript in one HTML file that works just by opening it in a browser. Name the file worklog.html

Each of the 5 lines always shows up in one place on the screen. Line 1 is the 5 input fields of the form, line 2 is the Add button and the table, line 3 is the total field, line 4 is the message below the form, and line 5 is file:/// in the address bar.

Steps

1. Open the tpro-work folder in VS Code and choose Agent below the input box in the Chat view.
2. Open tpro-work/spec5.txt and copy the 5 lines with Ctrl+A and Ctrl+C.
3. Paste it into the chat input box and send it.
4. Read the proposed diff from the top, and approve it only after you have read it.
5. In File Explorer, open tpro-work on your desktop and double-click worklog.html.
6. Enter 3 records of 30, 15 and 45 minutes and look at the total field and the table.

Check your result

- Double-clicking worklog.html opens it in your default browser, and the address bar starts with file:///
- After you add 3 records of 30, 15 and 45 minutes, the table has 3 rows and the total field shows 90
- If you leave any one field empty and click Add, no row is added and "Some fields are empty" appears

You end up with worklog.html in the tpro-work folder on your desktop. If the file name is different, send "Rename the file to worklog.html" in the chat to have it fixed. How the app looks differs from person to person. If the 3 checks above work, you are done.

If it does not work

Symptom	Fix
The screen is blank	Press F12 and look at the Console. If it shows a failure to load an external file, send "Fix it so it works with a single HTML file and no external libraries"
The file is empty	You have not approved the diff. Go back to the chat and approve it
It was split into 3 files: html, css and js	Send "Put everything back into a single HTML file"
The file was created outside tpro-work	Right-click the file in VS Code, choose Reveal in File Explorer (Windows) / Reveal in Finder (Mac), and check where it is
VS Code opens when you double-click the file	Right-click the file, choose Open with, and select a browser
The total field is concatenated into something like 3015	Send one sentence: "Add the totals as numbers."
The empty-field check does not work for some fields	Try each of the 3 input fields and name the field where the check fails
The date input is a plain text field	This is expected, because the spec does not state a type. Add "Make the date a date picker" and it changes

If time allows

- Send one more sentence, "Also show the total in hours," and read only the lines that changed.
- Send "Add a delete button to each row," and after the change, check that the total of the 3 records does not shift.
- Reload the page and confirm that your entries disappear. The spec does not mention saving, so disappearing is the correct behavior.

Exercise 4: Add a filter by category

Scenario

At a client site, the first change you are asked to make is usually a small addition to a screen that someone else wrote. There is no specification. You have only a working screen and a spoken request. From that starting point, you go through the whole cycle once with the Worklog app from the handouts, up to adding one feature without breaking the existing ones.

The same request returns different results depending on whether Copilot read the file first. If you send only "Make it possible to filter by category," the reply is a rewrite that does not use the existing render. After you paste it, the list may appear twice or the Add button may stop responding. If you first attach index.html and have Copilot explain it, and then ask for the same thing, the reply is a change that uses the existing render and records as they are.

Have Copilot read it first

Before adding a feature, have Copilot explain how the app is built now. Check the explanation yourself in the code.

1. Open index.html in the worklog-app folder in the browser. Confirm that the list appears and that you can add one record. Look at the app in its working state before you break anything.
2. In the Copilot Chat input box, type a half-width # and choose index.html.
3. Send the following prompt as it is. Do not have Copilot write code yet.

```
#index.html
Explain what this app does and how the list gets onto the screen.
You must include these 3 points.
- Where the work log data is stored
- The name of the function that draws the list
- The name of the function called when the Add button is clicked
Answer in English. Do not write any code yet.
```

4. Search for the function names from the reply inside index.html. If you cannot find them, you cannot trust the explanation.

The prompt asks for 3 points so that you can cross-check the reply yourself. You can verify a function name by searching for it.

Steps

1. Write the spec in the chat in 3 lines. Put a field for choosing a category above the list. When you choose a category, the list shows only that category. When you choose "All," the list goes back to the original.
2. Attach index.html with a half-width #, and send the 3-line spec. Add one more sentence at the end: "Use the existing render as it is."
3. Open the place named in the reply in VS Code and find it first, then paste. If you cannot tell, ask back: "Tell me which function to put it under." Copy the code all at once with the Copy button at the top right of the reply. If you select only part of it by hand, you miss lines when you paste.

4. Press Ctrl+S to save, then press Ctrl+Shift+R in the browser to reload. Change the selection and check that the list changes.
5. If it does not work, press F12 and paste the error message into the chat as it is. Do not rephrase it in your own words.

Change only one thing at a time. If you ask for two at once, you cannot tell which one broke it.

Keep a copy of the working version. When one feature works, copy index.html and save it under a new name that ends with a suffix such as _ok1. If things get out of hand, go back to that copy and rewrite the prompt. If you send the same sentence again, you get stuck at the same point. This course does not use Git, so a copy takes its place.

Check your result

- A field for choosing a category appears above the list, and choosing Design leaves only the Design rows
- Going back to "All" restores the record count from before filtering
- Even while a filter is on, you can add one record from the form. The existing features still work
- The change is added to index.html in a way that calls the existing render. render has not been replaced as a whole by another function

Check that the new feature works and that the existing features are not broken as two separate checks.

If it does not work

Symptom	Fix
The selection field appears but the list does not change	The event handler is not registered. Tell Copilot only "The list does not update when I change the selection" and have it fix the problem
The screen went blank	Press F12 and paste the first red line as it is. Do not guess the cause
The screen does not change although you saved	This is the browser cache. Press Ctrl+Shift+R to reload
The Add button stopped responding	The existing render has been replaced. Restore the file to its state before you pasted, add a line telling Copilot to use the existing render, and ask again
You do not know where to paste	If you ask back "Tell me which function to put it under," the reply even gives the line position

When it works, note in one line which sentence you think made the difference. This note is material for Exercise 7.

If time allows

- Show the number of records for the current selection above the list, like "Showing 12 records"
- Make it possible to filter by date range as well as by category. Add them one at a time

- Add a record while a filter is on, check whether that record appears in the list, and decide for yourself what the spec should be if it does not

Exercise 5: Add total hours by category

Scenario

Totals are the kind of processing that AI writes for you quickly. That makes it easy to accept the result without reading it, and the screen looks plausible even when a number is off by orders of magnitude. Here, you build the habit of cross-checking one of the numbers yourself.

Check that the number is correct, not just that a number appears on the screen.

Steps

1. Decide what to show. Show the total hours for each category in a table below the list. When a filter is on, total only the filtered result.
2. As in Exercise 4, attach index.html and send the spec. Add one more sentence at the end: "Do not break the previous filter feature."
3. Choose one category, add up the hours shown in the list yourself, and check that they match the total on the screen. Always do this part yourself.
4. Switch the category and check that the totals change with it. If they do not, describe the symptom as it is and have Copilot fix it.
5. Create a new file model-memo.txt in tpro-work and write the cross-check result in one line. Write whether the total matched or was off, and if it was off, what the cause was.

Check your result

- A table of total hours by category appears below the list
- For one chosen category, the total you added by hand matches the total on the screen. You confirmed the match yourself
- When you change the filter, the totals change with it
- model-memo.txt has the cross-check result in one line. If the total was off, the cause is written there too

If it does not work

Symptom	Fix
The totals stay at all records and do not follow the filter	Tell Copilot only "The totals do not change when I switch the category"
The total joins side by side, like 1.52.5	The values are being added as strings. Paste the symptom into the chat as it is and Copilot fixes it

Symptom	Fix
The totals table gets pushed into the list and the layout breaks	The spec does not say where to show it. Add "Show it as a separate table below the list"
The previous filter disappeared	You forgot the sentence in Step 2. Restore the file to its state before you pasted, and ask again
A decimal total is off by 0.1	This comes from rounding decimals. Do not dig into it today. Adding an instruction to round the displayed digits solves it

If time allows

- Sort the totals from largest to smallest
- Show the record count next to each total row, so that you can see how many records add up to how many hours
- Have Copilot deliberately break just one total, and test whether you notice it in your cross-check. Always copy the file before you break it

Exercise 6: Switch models and send the same request twice

Scenario

At a client site, your organization's settings may decide which models you can use. Even where you can choose, if you have never seen for yourself what changes when you switch, you get into the habit of changing the model every time something goes wrong. Send the same sentence twice and check on your own PC which parts change and which parts stay the same.

Here, a model means the type of AI that produces the chat answers. In Copilot Chat, the Claude family (Anthropic), the GPT and Codex family (OpenAI), the Gemini family (Google), and Auto, which Copilot chooses for you, are listed in the same place. The lineup changes every few months, so memorizing a ranking does not help at your next site. Remember two things: where the switch is, and what does not change when you switch.

What changes when you switch is the wording of the reply, the style of the code, how much of a long file the model can handle at once, and how fast the reply feels. What does not change is the information you did not give. If you have not attached the existing code, every model returns a suggestion that ignores the existing structure.

Steps

1. Open the tpro-work folder in VS Code, then open worklog.html inside it. If you do not have the file you built, use index.html in the worklog-app folder.
2. Open model-memo.txt that you created in Exercise 5, paste the prompt as one line, and keep it as a record. Use exactly the following sentence.

Explain how this file works to someone seeing it for the first time. Answer in English.

3. In the Copilot Chat input box, type a half-width #. A list of file candidates appears. Choose the target file to attach it. If the file name appears above the input box, the file is attached.
4. Send it without touching the model name. Read only the first 5 lines of the reply. You do not need to read all of it.
5. Click the model name shown in the input box to open the list of choices. This is where you switch. Switch to a family from a different provider, paste the sentence you kept, and send it again. Do not change a single character of the sentence.
6. Add one line to model-memo.txt about what was different. Do not write which one was better.

If you change the prompt and the model at the same time, you cannot tell which one made the difference. Change only one at a time.

Writing the difference is easier if you use one of these viewpoints: the length of the explanation (short and compact, or detailed and itemized), how the code is shown (the whole file, or only the changed parts), or the check of assumptions (asked a question first, or answered right away).

Check your result

- model-memo.txt has the prompt you sent, as it is, in one line. You can check later that you sent the same sentence both times
- The first request used the default model and the second used a family from a different provider. You can confirm in the list that the provider is different
- One line about the difference is there, written as "what was different," not as which model was better
- If you could not see a difference, writing "I could not see a difference with this request" counts as done

If it does not work

Symptom	Fix
No candidates appear when you type #	You typed a full-width #. Switch to half-width input and type it again
The file is not attached	Check whether the file name appears above the input box. If it does not, the file is not attached
The model list does not open	You are no longer signed in. Sign in to GitHub again from the account icon at the bottom left
Only one family appears in the list	Your organization's settings limit the choices. Send once with the default, and ask the person next to you to show you the difference on their screen
You added to the prompt by mistake the second time	Go back to the one line you saved and send it again in a new chat
The provider is the same even though you switched	You chose a different version from the same provider. Check the provider name in the list and choose again

Even if you send the same request twice to the same model, the answer may change a little. You do not get exactly the same answer every time. You may record that as a difference too.

If time allows

- Send the same sentence to a third family, compare the three, and add one more line about the differences
- Send the same sentence twice in a row to the same model, and see whether the reply varies even within one model
- Add "in 200 characters or fewer" at the end of the prompt, send it again, and see how well the length limit is followed

Exercise 7: Add an instructions file and compare before and after

Scenario

At a client site, several people work on the code of the same screen. If some people write careful preambles in their prompts and others leave them out, the same request returns code in different shapes, and review comments differ from person to person. You check on your own PC how far one instructions file closes that gap.

When you open a new chat thread, the previous conversation is not carried over, so you have to retype the language to use, the libraries to avoid, and how to write comments every time. If you put them in `tpro-work/.github/copilot-instructions.md`, Copilot Chat reads the file automatically in every conversation. You do not need to remember to paste it. When you want to change the rules, fix this one file.

`.github` is a fixed folder name that comes from GitHub. GitHub is a place to share source code. GitHub Copilot is a separate product that runs inside VS Code, and it is Copilot that reads the instructions file.

Steps

1. Open the `tpro-work` folder on your desktop in VS Code, and send only the following sentence to Copilot Chat. Do not change the wording.

```
Add a function that checks the hours input
```

2. Paste the code that appears into `before.txt` in the `tpro-work` folder and save it.
3. In the VS Code Explorer, click the `tpro-work` row once to select it, and create `.github` with the New Folder icon. If you forget the leading dot, the folder is not read. If you create it one level above `tpro-work`, it is not read either.
4. With `.github` selected, create a new file named `copilot-instructions.md`, and paste the full text below. Then add only 2 lines that you want to add for your own work, and press `Ctrl+S` to save.
5. Reopen the chat in a new thread and send exactly the same wording as in Step 1 again. Paste the code that appears into `after.txt` and save it.
6. Open `before.txt` and `after.txt` side by side (drag a tab to the right edge to split the screen in two), and compare 3 points: the variable names, the language of the comments, and how DOM elements are retrieved.

Full text of the instructions file

Copy this after you finish Step 3. Complete Step 1 first, or there is no before and after to compare.

This is the full text of copilot-instructions.md written for the Worklog app. It is divided under four headings: premise, language and style, review points, and how to present changes. The key is to write down even what you want to forbid.

```
# Project premise
- Write HTML, CSS and JavaScript in a single index.html. Do not split the file.
- Open index.html directly in the browser to run it. There is no build step.
- Do not use fetch. It does not work over file://, so receive files by drag and drop.

# Language and style
- Write in JavaScript (ES2020). Do not use TypeScript.
- Do not add external libraries. Use only the ones that are bundled.
- Write variable and function names in English, and write comments in English too.
- Get DOM elements only with document.getElementById and querySelector.
- Do not delete the existing render function. Redraw the screen by calling render.

# Review points
- Does the code add to records only after the input check passes?
- Is the text shown on screen kept in one place instead of scattered through the code?
- Does the code show a message in the on-screen message area instead of swallowing exceptions?

# How to present changes
- Start with one line that gives the changed file names and function names.
- Show each affected function in full, without ellipses.
```

Two instructions files

This shows how it differs from AGENTS.md, which you use in the extra exercise. The readers differ, so the location and the level of detail differ too.

Item	copilot-instructions.md	AGENTS.md
Location	Inside tpro-work/.github/	Directly under tpro-work
Read by	GitHub Copilot. Chat reads it in every conversation.	A shared format that several AI coding tools, including Copilot, read.
Contents	Style rules: language, naming, prohibitions, review points	Project premise: structure, how to run it, what you may touch
Approximate length	About 20 lines. Most important rules first	About 30 lines. Facts only, no judgments

Do not copy the same content into both files. When you fix one, the other becomes outdated and the results become unstable.

Check your result

- copilot-instructions.md appears in the list of referenced files above the answer. That confirms the file was read.
- The comments in after.txt are in English, and DOM lookups use getElementById or querySelector
- You can say in numbers how many items written in the instructions file were applied and how many were not
- before.txt and after.txt are saved in the tpro-work folder

Checking that the file was read is a separate task from checking that it was followed. Even if its name appears in the list of referenced files, not every line is followed.

If it does not work

Symptom	Fix
Cannot create the .github folder	Windows File Explorer sometimes rejects names that start with a dot. In the VS Code Explorer, click New Folder and type .github as the name.
The file name does not appear in the list of referenced files	Either you did not save the file, or you created .github one level above tpro-work. Check 2 things: the dot on the tab and the folder level in the Explorer.
No difference between before and after	You added detailed requirements to the request in Step 1. They overlap with the instructions file, so the difference disappears. Restore the original wording and send it again.
You continued the second request in the same thread	The earlier conversation affects the answer, so you cannot see the difference. Open a new thread and send the request again.
You wrote the file name as copilot-instruction.md	instructions is plural. If even one character differs, the file is not read.
A dot stays next to the tab name	The file is not saved. Press Ctrl+S to save it.

If time allows

- Add "At most 30 lines per function" to the instructions file, send a third request, and see how much effect the line-count rule has.
- Delete only the prohibition lines from the instructions file, send the same request, and see whether the result changes with and without negative rules.
- Place AGENTS.md directly under tpro-work and check whether both files appear in the list of referenced files. Its contents are below.

```
# Worklog app
A one-screen app where you enter the date, category, description and hours and show them in a list.

## Structure
- index.html ... Holds the screen, styles and logic in one file
- .github/copilot-instructions.md ... Coding instructions
```

```
## How to run it
- Open index.html in the browser. Do not start a server.
- After a change, save with Ctrl+S, then press Ctrl+Shift+R in the browser to reload.

## What you may touch
- Do not rename the fields in records (date, kind, memo, hours).
- Do not delete and rebuild the existing render. Add to it.

## Constraints
- Do not write code that communicates with the outside.
- Do not write real company names, person names or project names in the data.
```

Troubleshooting

These are common problems that come up across the exercises.

Symptom	First thing to check
The chat does not answer in English	Add "Answer in English." at the end of the prompt
The chat answer misses the point	Check whether the target file is open and the range is selected. Information you have not given to the chat is not used.
The conversation has gone off track	Send /clear once to discard the context, select the target again, and resend the request.
The gray suggestion text does not appear	Check the file extension. Also check that the language name appears in the Status Bar.
You approved the diff, but the file does not change	Check that you saved with Ctrl+S and that the dot on the tab is gone.
The change does not show in the browser	Save, then reload (F5). Reloading without saving does not change the page.
The page is blank in the browser	Check the Console tab (F12). If an external file failed to load, have Copilot fix the page so that it is complete in a single HTML file.
Double-clicking opens something other than the browser	Right-click the file, choose Open with, and select the browser.
You cannot find the generated file	In VS Code, right-click the file and choose Reveal in File Explorer (Windows) / Reveal in Finder (Mac).
The AI's change removed an existing feature	Add "Show only the changed lines. Do not change existing features." and send the request again.

Symptom	First thing to check
You want to undo a change	Use Ctrl+Z, or Undo in the chat. You do not use Git today, so these 2 are your only ways back.
Nothing you try fixes it	Restore it by copying worklog-complete.html and naming the copy worklog.html. For the Worklog app, go back to the copy with _ok1 in its name that you saved when the app last worked.
Copilot does not respond, or the model list does not appear	You have been signed out. Check whether the icon at the bottom right of VS Code has a slash through it, then sign in again from the account icon at the bottom left.
No file suggestions appear when you type #	It is a full-width #. Switch to a half-width #.
A file you meant to attach is not being read	Check whether the file name appears above the input box. If it does not, the file is not attached
The answer ignores the existing structure	You did not attach the target file. Attach it so that Copilot reads it, then ask again.

What you take home today

Everything you did today fits in one loop: ask, read the diff and Keep it, and check in the browser. You used the same loop whether you built from scratch or added to an existing app. The only difference was that, for an existing app, you had Copilot read the file before you asked.

You are left with the following.

- The tpro-work folder on the Desktop, and worklog.html, notes.txt, prompt.txt, model-memo.txt, before.txt and after.txt inside it
- worklog-app/index.html, with the filter and total hours added
- tpro-work/.github/copilot-instructions.md. It is read automatically from the next chat on.

You accepted the total because it matched the number you added by hand, not because it appeared on the screen. A line you approve without reading becomes a line you wrote. Checking that it works and deciding to accept it stay with you from today on.

What to keep until the next session

- The tpro-work folder. Leave index.html with your added features, and the instructions file, as they are. You use them at the start of Day 2.
- The Copilot sign-in state. The next session is 2 weeks away. Do not sign out.

Day 2 starts with a debugging exercise using the index.html you extended today. After that, you decide from the specification what to check, and create test cases and test data.

Rules to follow

All exercise material is fictional. Do not use real data or code from your own work, or client-specific names. When you use this at your desk later, follow the 4 rules below.

1. Generalize client names, product names, project names and partner names to the industry or process level
2. Do not use real data, real files or screenshots
3. Avoid specific values for figures and specifications, and write at the level of purpose, issues and expected effects
4. Before you submit, check yourself that the content is safe to send outside the company

If you are unsure, search for the word and judge whether the search can identify the client. If it can, generalize it. Read your text through once yourself before you press the Send button.

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.