Copilot Chat & inline suggestions

Use the right context, then check the change

Session M1-02 | 3 hours

Agenda

Segment	Time
Inline suggestions and Chat	6 min
Choose a Chat surface	10 min
Select context deliberately	11 min
Explain, fix, test, document	10 min
Conversational refinement	8 min
Model and metering decisions	7 min
Debugging demo and handoff	8 min

Two interaction modes

Inline suggestions	Chat
Draft at the cursor	Respond to an explicit request
Best for the next small edit	Best for explanation and multi-step work
Context comes from the editing surface	Context can be selected or named

Both produce proposals that need review.

Choose the surface by the work

Work	Useful surface
Multi-step diagnosis	Sidebar Chat
Change selected code	Inline Chat
Ask one short question	Lightweight or quick surface
Explain recent command output	Terminal context

Use the controls available in the editor.

Sidebar and inline Chat

Sidebar Chat

- keeps a task thread visible;
- supports follow-up questions;
- works well for cross-file reasoning.

Inline Chat

- starts from selected code;
- shows an edit close to the source;
- fits a narrow change.

Quick and terminal context

A lightweight surface is useful when history is unnecessary.

Terminal context is useful for:

- a failed test command;
- an unfamiliar error;
- the next command to check the result.

Read the actual output before acting on the summary.

Context should be explicit

Weak:

```
Fix this.
```

Reviewable:

```
The weighted-average test expects 95 but receives 190.
Inspect the selected method and focused test.
Identify the defect without changing other methods.
```

Workspace, file, selection, terminal

Context	Use it for
Selection	One function or block
Named file	Known cross-file dependency
Workspace search	Broad structure or usage
Terminal output	Recent command results

Include the context the task needs.

Explain before changing

Ask for:

1. the current formula;
2. the role of each guard;
3. the failing example;
4. the exact mismatch.

An explanation can expose a wrong assumption before code changes.

Fix narrowly

```
Fix only weighted_average so it divides by total weight.
Preserve the method signature and length check.
Add no dependency.
```

Scope is part of correctness.

Generated tests need review

Check whether each test:

- states accepted behavior;
- would fail for the original defect;
- calculates the expected value correctly;
- avoids invented requirements;
- uses existing tools.

Check the expected behavior too. A generated test can encode the same mistake as the code.

Refine one concern at a time

```
Explain
   ↓
Diagnose
   ↓
Fix one behavior
   ↓
Add focused tests
   ↓
Review the complete method
```

Focused turns make mistakes easier to locate.

Know when to restart

Continue the thread when:

- the task is unchanged;
- the useful context remains accurate;
- the current structure is sound.

Start fresh when stale assumptions keep returning.

Compare against the task

If policy allows two available model choices:

1. use the same synthetic prompt;
2. keep the same context;
3. run the same tests;
4. compare correctness and useful citations.

Judge the results by the checks, not how well they read.

Access policy

The lab requires Chat access allowed by your organization.

If Chat is unavailable, **stop the lab and restore access**.
Manual work does not replace the Chat and inline-suggestion practice.

If only a second model is unavailable, continue with Chat and compare the assisted result with the reference solution.

Demo: weighted average

```
weighted_sum = sum(
    value * weight
    for value, weight in zip(values, weights)
)
return weighted_sum / len(values)
```

For `[80, 100]` with weights `[1, 3]`, the denominator must be `4`.

Failure to fix

```
Expected: 95
Actual:   190
```

The focused test gives Chat a concrete symptom. The reviewed fix changes the denominator to the total weight.

Lab handoff

You will use one calculator project to:

1. reproduce the defect;
2. explain and diagnose it;
3. apply a narrow fix;
4. review generated tests;
5. use workspace and terminal context;
6. decide zero-total-weight behavior;
7. run the full suite.

Deliverable: corrected code, passing tests, and a Chat decision record.