Copilot Chat & Inline Suggestions

Copilot Fundamentals | Beginner

Session 02 of 18 | Duration: 3 hours (1hr content + 2hr lab)

Agenda & Time Plan

Time	Activity	Duration
0:00	Recap & Context Setting	3 min
0:03	The Four Chat Surfaces	10 min
0:13	Chat Participants: @workspace, @terminal, @vscode	8 min
0:21	Slash Commands Deep Dive	8 min
0:29	Context Management: #file, #selection, #editor	7 min
0:36	Model Selection & Cost Awareness	7 min
0:43	Conversational Workflows	7 min
0:50	Copilot CLI	5 min
0:55	Live Demo: Debugging with Chat	5 min
1:00	Wrap-Up & Lab Preview	3 min
1:03	☕ Break	10 min
1:13	Lab Start	—
3:00	Wrap-up	—

Breaks: 10-minute break between trainer content and lab. Additional 5-minute breaks at trainer's discretion.

Inline Completions vs. Copilot Chat

	Inline Completions	Copilot Chat
Mode	Reactive: suggests as you type	Proactive: you ask, Copilot answers
Trigger	Automatic while coding	You open a Chat surface
Scope	Current cursor position	Whole file, project, or concept
Best for	Flow-state coding	Explaining, debugging, generating, refactoring

Switch between them as the task changes. Each gives you different context.

The Four Chat Surfaces

Surface	Shortcut	Best For	History
Sidebar	`Ctrl+Alt+I`	Extended conversations, @workspace	✅ Preserved
Inline	`Ctrl+I` (in editor)	Code-specific edits, explanations	Ephemeral
Quick Chat	`Ctrl+Shift+Alt+L`	Fast one-off questions	❌ Dismissed
Terminal	`Ctrl+I` (in terminal)	CLI help, error debugging	Ephemeral

Use the sidebar for a longer thread, inline chat for a focused edit, quick chat for a one-off question, and terminal chat for command output.

Sidebar Chat: Your Primary Surface

- Full-height panel with messaging-style interface
- **Preserves conversation history** within the session
- Supports multiple threads (click `+` for a new thread)
- Best for longer conversations, @workspace queries

Try it:

```
What does this project do?
```

Then follow up:

```
What programming languages are used?
```

Context carries over between messages.

Inline Chat: Focused Code Edits

- Select code → `Ctrl+I` → type your request
- Result appears as a **diff** right in the editor
- Accept, reject, or modify without leaving the editor

Try it:

1. Select a function
2. Press `Ctrl+I`
3. Type: `add input validation`
4. Review the inline diff

> Inline Chat is **context-anchored**. Copilot sees exactly what you selected.

Chat Participants

Point Copilot at specific knowledge sources with `@`:

Participant	Scope	Example
@workspace	Entire project	*"Where is auth configured?"*
@terminal	Terminal output & commands	*"What does this error mean?"*
@vscode	VS Code settings & features	*"How do I change the theme?"*

```
@workspace How is error handling implemented in this project?
```

Use `@workspace` for a project question. For a specific file, name that file explicitly.

Slash Commands

Command	Purpose	Example
`/explain`	Explain selected code	`/explain What does this regex do?`
`/fix`	Fix bugs in selected code	`/fix This throws a null reference`
`/tests`	Generate test cases	`/tests for the validate() function`
`/doc`	Generate documentation	`/doc Add JSDoc to this class`
`/new`	Scaffold new code	`/new Express route for user signup`

- Combine with selections for precise results
- Use `/fix` + error message for rapid debugging

Context References

Tell Copilot exactly what to look at with `#` references:

Reference	What It Adds
`#file:path/to/file.ts`	Contents of a specific file
`#selection`	Currently selected code
`#editor`	Visible content in the editor

```
Based on the patterns in #file:services/user_service.py,
create a new service method for order cancellation.
```

Name the context that matters. It reduces guesswork.

Model Selection & Cost

- Click the model picker in the Chat panel to switch models
- **Auto** lets Copilot choose a model based on the task
- Model costs are usage-based: token consumption × per-model rate

Model type	Good for	Cost note
Lightweight models	Quick questions, simple fixes	Lower-cost per token
Balanced models	General coding and explanations	Good default for most work
Frontier / deep-reasoning models	Complex architecture and design	Substantially more expensive

Start on **Auto**. Switch when you need a specific model's strengths.
⚠️ Available models and pricing change. Verify live docs: supported models and models and pricing.

Conversational Workflows

Build on the previous response when it has the right direction:

1. **Ask** → Get initial code
2. **Refine** → "Make it async" or "Add error handling"
3. **Constrain** → "Use only the standard library"
4. **Validate** → "Does this handle empty inputs?"

```
Write a function to parse CSV data.
→ Make it handle quoted fields with commas inside.
→ Add type hints and a docstring.
→ Now write tests for the edge cases we discussed.
```

Keep the useful parts and ask for the next specific change.

Copilot in the Terminal

GitHub Copilot includes a **standalone CLI tool** for terminal assistance — shell command explanation, generation, and interactive help.

The current standard is the **standalone `copilot` CLI binary** (`npm install -g @github/copilot` or via your package manager).

The CLI is covered fully in **Session 04 — GitHub Copilot in the CLI**, including:

- Interactive command explanation and suggestion
- Shell integration and context-aware completions

> Skip the CLI demo here. Point trainees to Session 04 for hands-on practice.

Lab Time

Hands-on Exercises (2 hours)

Lab Overview

Exercise	Topic	Duration
1	Code Explanation & Debugging	40 min
2	Test Generation	30 min
3	Workspace & Terminal Chat	30 min
4	Model Comparison	20 min

Deliverable: A debugged and tested module, plus documented model comparison observations.

Key Takeaways

1. **Four Chat surfaces** — Sidebar, Inline, Quick Chat, Terminal — each for a different context
2. **@workspace** searches your entire project — use it for cross-file questions
3. **Slash commands** (`/explain`, `/fix`, `/tests`, `/doc`) accelerate common tasks
4. **Context references** (`#file`, `#selection`) give Copilot precise input
5. **Model selection** lets you match model strengths to your task
6. **Iterative conversations** work better than one-shot prompts. Refine instead of restarting.

Questions?

Take a moment to discuss with your trainer

Next Up

Session 03: Prompt Engineering Fundamentals

Thank you!