



# GitHub Copilot

**Sukumar P**

*Github CSA team*

<https://www.linkedin.com/in/sukumar-p/>

Level 100

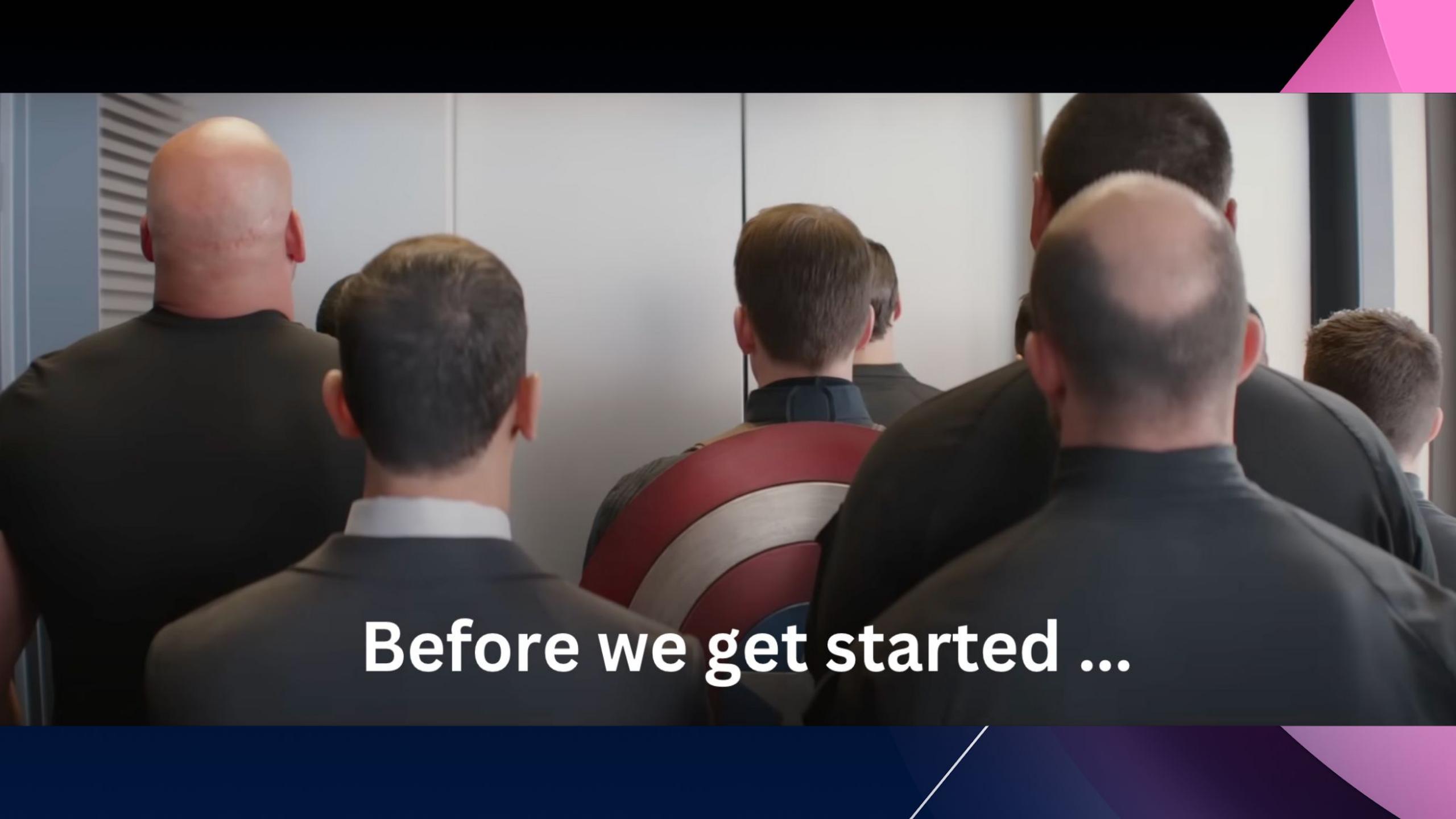
Let's boost your productivity with GitHub's AI-powered developer platform



# Agenda

- Copilot flow
- Code completion
- Next Edit Suggestion
- Inline Chat vs Normal Chat
- Slash commands, Chat participants, Context variables
- Custom instructions
- Reusable prompts
- Code-adjacent tasks
- Public-code blocking
- Ask vs Edit
- Copilot voice chat
- Saving chat conversations
- Best Practices
- Model selection
- Debugging with Copilot
- Q&A



A photograph showing the backs of several men in dark suits and ties, all looking towards a white wall where a presentation is likely being displayed. One man in the foreground has a red, white, and blue striped shoulder patch on his suit jacket.

**Before we get started ...**



I would like to know your experience with Copilot



# GitHub Copilot

## What is GitHub Copilot?



### Increase developer productivity

And satisfaction by focusing on real problems



### Accelerate innovation

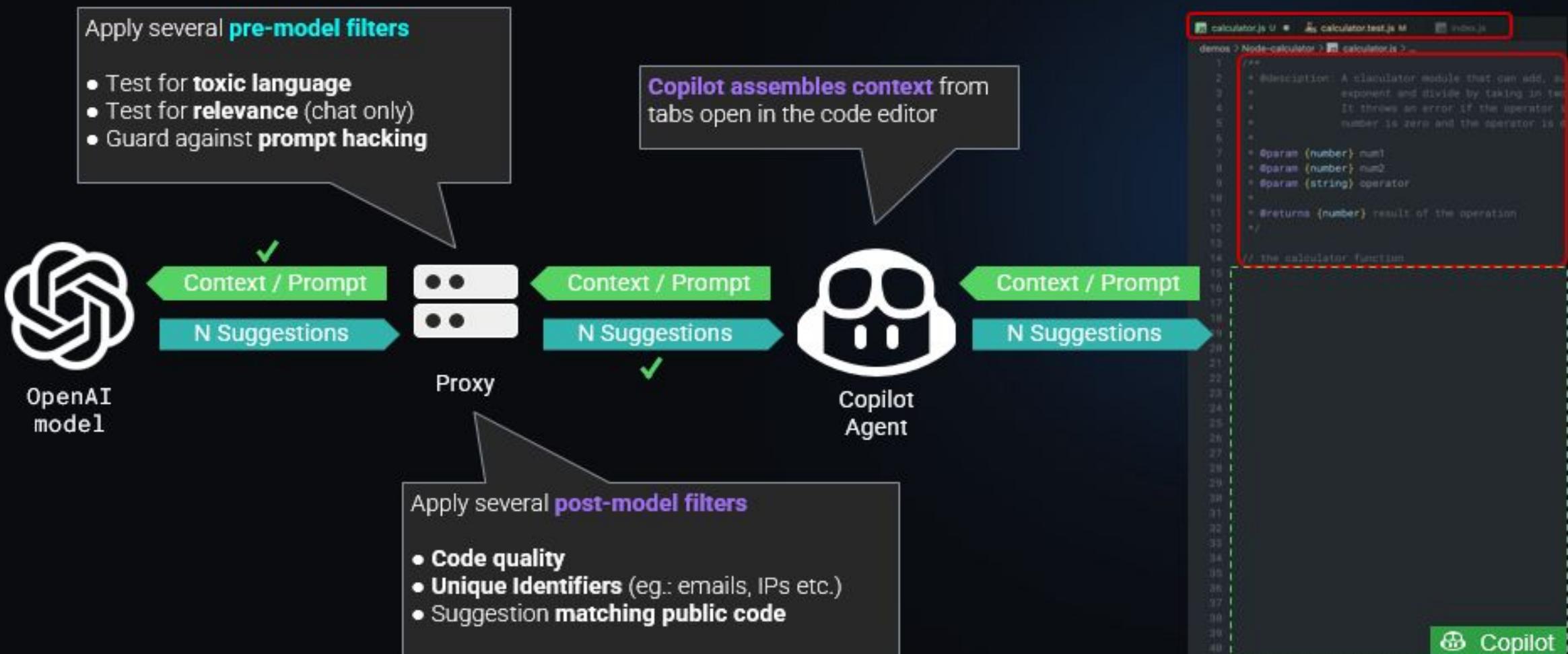
Prototype & innovate more rapidly



### Bridge skill gaps

Learn new languages and techniques

# How Copilot Fulfills a Request



# Code Completion



```
sentiment.ts -GO write_sql.go parse_expenses.p  
1 #!/usr/bin/env ts-node  
2  
3 import { fetch } from "fetch-h2";  
4  
5 // Determine whether the sentiment of te  
6 // Use a web service  
7 async function isPositive(text: string):  
8   const response = await fetch(`http://te  
9     method: "POST",  
10    body: `text=${text}`,  
11    headers: {  
12      "Content-Type": "application/x-www-  
13    },  
14  );  
15  const json = await response.json();  
16  return json.label === "pos";  
17 }|
```



Code completion

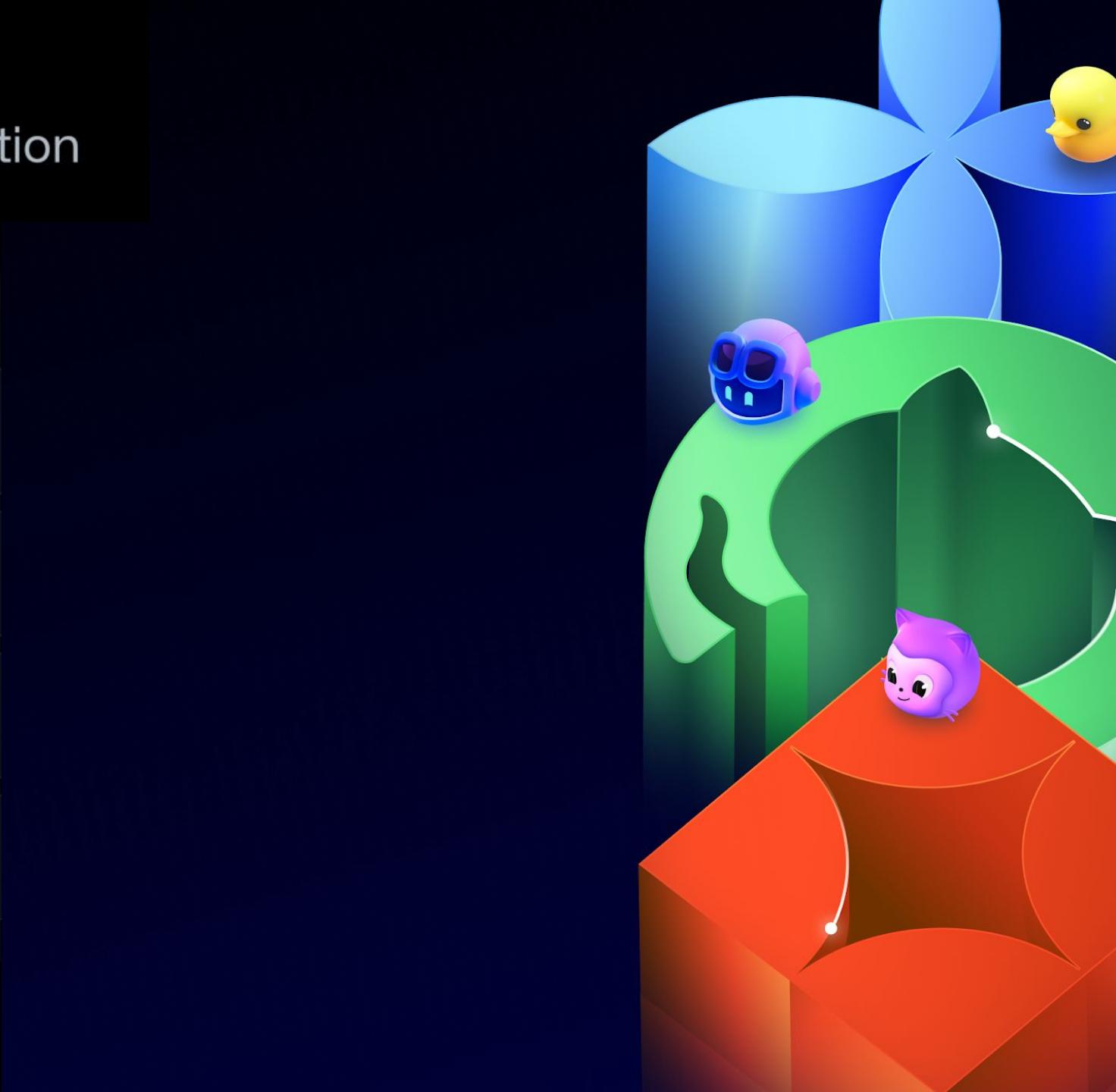
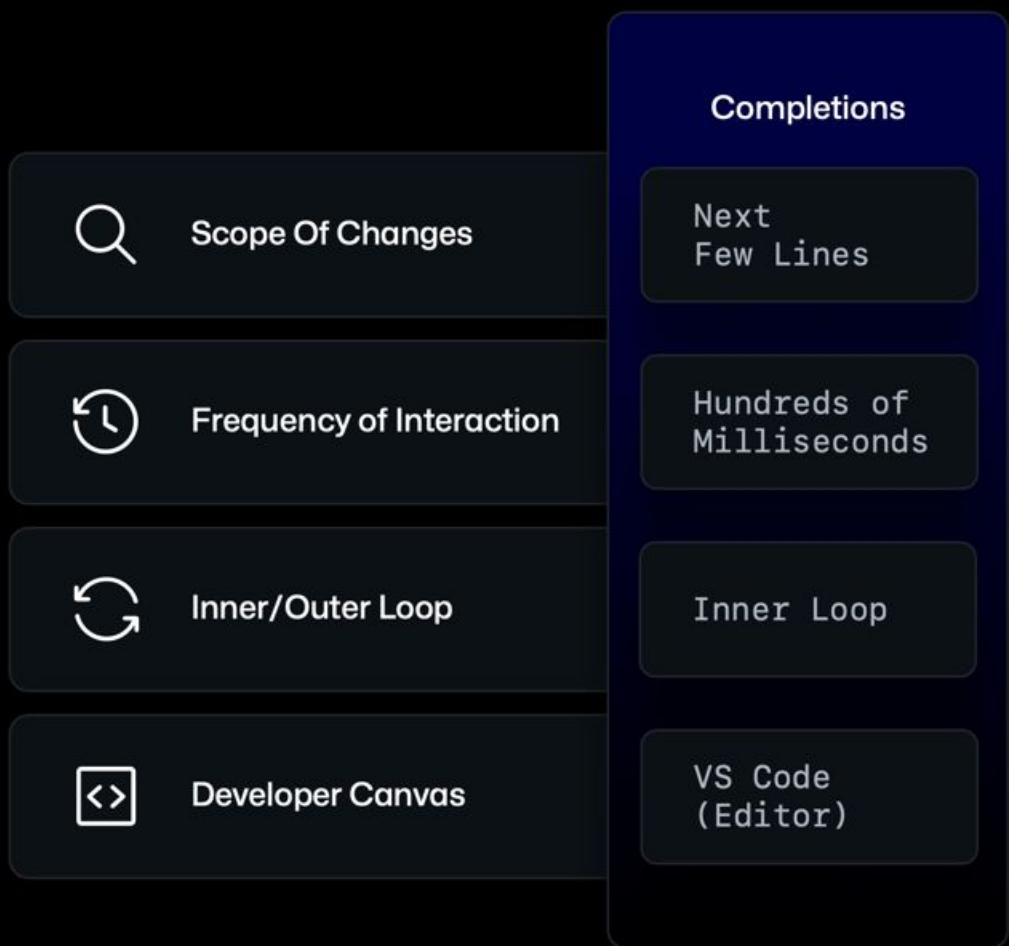


# Your AI pair programmer

Real time code suggestions

Comments to code

# GitHub Copilot / Code Generation



# Code Completion

Keep flying in your favorite editor



VS Code



Visual Studio



Neovim



JetBrains IDEs



Xcode



Eclipse

GA

GA

Next Edit Suggestions



# Predicts where the next edit should be

Based on the edits you're making, Copilot NES both predicts the location of the next edit you'll want to make and what that edit should be.



VS Code

A screenshot of the Visual Studio Code interface demonstrating the Copilot Next Edit Suggestion feature. A context menu is open over some code, listing options: 'Copilot Next Edit Suggestion', 'Go To / Accept' (with Tab key), 'Reject' (with Escape key), 'Feedback', and 'Settings'. The code editor shows the following Java-like code:

```
public Point(double x, double y) {  
    private double z;  
  
    public Point3D(double x, double y, double z) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

# GitHub Copilot Inline Chat

Quickly access Chat inline to ask questions and generate code suggestions

The screenshot shows a dark-themed code editor interface for a Java file named Book.java. The file contains the following code:src > main > java > com > github > demo > model > Book.java > ...  
6 public class Book {  
10 private String author;  
11  
12 private String cover;  
13A modal window titled "GitHub Copilot" is open, displaying a suggestion for adding comments to the code. The suggestion text is: "comment this code by including a comment line before each block". Below this, it says "Made changes." and shows the modified code with new comment lines added:14 // Default constructor  
15 public Book() {  
16 }  
17  
18 // Constructor with author and title param  
19 public Book(String author, String title) {  
20 this.author = author;  
21 this.title = title;  
22 }  
23  
24 // Constructor with author, title, and cover  
25 public Book(String author, String title, S  
26 this.author = author;  
27 this.title = title;  
28 this.cover = cover;  
29 }  
30 }At the bottom of the modal, there are "Accept" and "Cancel" buttons.

# Chat Experience

You can use shortcuts to code faster and better (context awareness) with:

/ Slash commands

```
/help  
/explain  
/fix  
/tests  
/new  
/newNotebook  
/search  
/startDebugging
```

@ Chat participants

```
@workspace  
@terminal  
@vscode  
@azure
```

# Context variables

```
#editor  
#selection  
#[file_name]  
#changes  
#terminalSelection  
#terminalLastCommand  
#problems  
#kb
```

@workspace /tests

</> Workspace

## Plan

1. Import necessary modules for testing.
2. Mock the `fetch` function to simulate API responses.
3. Write test cases to cover:
  - o Positive sentiment response.
  - o Negative sentiment response.
  - o Neutral sentiment response.
  - o Error handling.

## Code

TS sentiment.test.ts

```
1+ import { isPositive } from './sentiment';
2+ import { fetch } from 'fetch-h2';
3+
4+ // sentiment.test.ts
5+
6+
7+ jest.mock('fetch-h2');
8+
9+ describe('isPositive', () => {
10+   it('should return true for positive sentiment', async () => {
11+     (fetch as jest.MockedFunction<typeof fetch>).mockResolvedValue({
```

# Chat & Edit



The screenshot shows the GitHub Copilot Chat interface integrated into a code editor. On the left, a sidebar displays a list of recent projects and a search bar. The main area has two tabs: "Copilot Chat" and "GitHub Copilot". In the "Copilot Chat" tab, a user named "monalisa" asks for unit test functions for a selected code snippet. The "GitHub Copilot" tab shows the generated Python code for parsing expenses from a string. The code editor window displays the "module.py" file with the following content:

```
1 import datetime
2
3 def parse_expenses(expenses_string):
4     """Parse the list of expenses and return the list of triples (date, amount, currency).
5     Ignore lines starting with #.
6     Parse the date using datetime.
7     Example expenses_string:
8         2023-01-02 -34.01 USD
9         2023-01-03 2.59 DKK
10        2023-01-03 -2.72 EUR
11
12    expenses = []
13
14    for line in expenses_string.splitlines():
15        if line.startswith("#"):
16            continue
17        date, value, currency = line.split(" ")
18        expenses.append(datetime.datetime.strptime(date, "%Y-%m-%d"),
19                         float(value),
20                         currency))
21
22    return expenses
23
24
25
26
27
28
29
30
31
expenses_data = '''2023-01-02 -34.01 USD
2023-01-03 2.59 DKK
2023-01-03 -2.72 EUR'''
```

The status bar at the bottom indicates: Ln 17, Col 3, Spaces: 2, UTF-8, LF, () TypeScript.

Copilot Chat



AI



# Context-aware conversations

- ✓ Answer coding questions
- ✓ Explain code
- ✓ Write unit tests
- ✓ Translate languages

GA

Chat / Multi-file edits



AI



# Code across your codebase in a single view

Tell GitHub Copilot what to do, and watch it update multiple files right in your editor.



VS Code



VS



JetBrains

Working Set (8 files)

Accept

Discard



# styles.css public\css

# dark.css public\themes

# default.css public\themes

<> contact.ejs src\views

<> index.ejs src\views

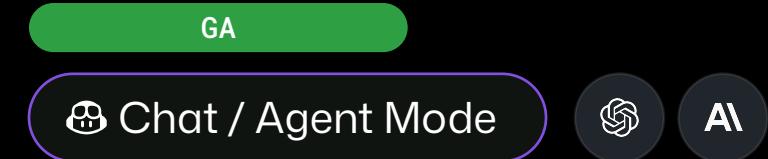
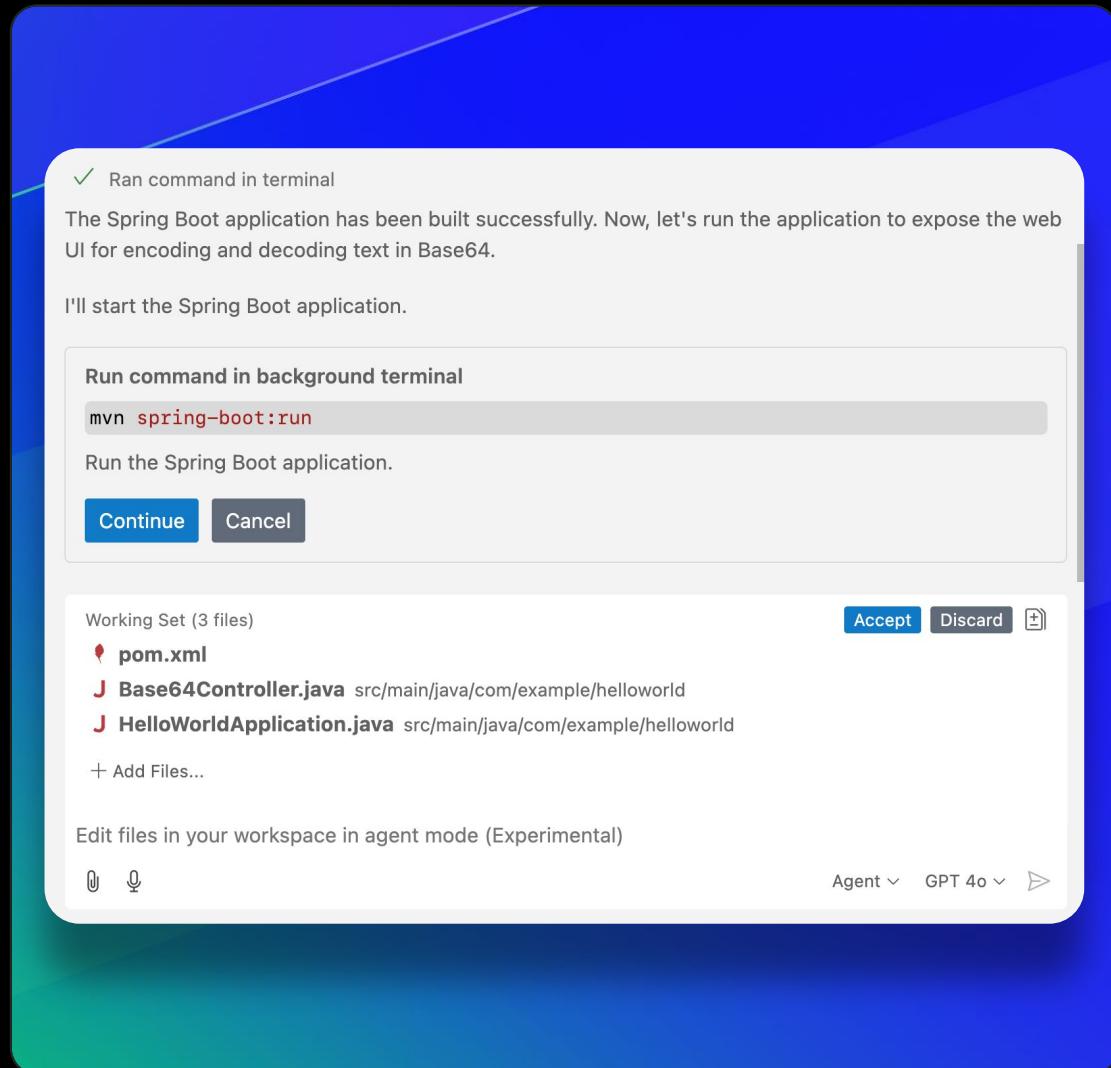
TS app.ts src

+ Add Files...

Edit files in your workspace

GPT 4o ▾





# Agent mode for Copilot Edit

File & codesearch tools for discovery  
Edits files & iterates on errors/tests  
Terminal execution and follow-up fixes



GA

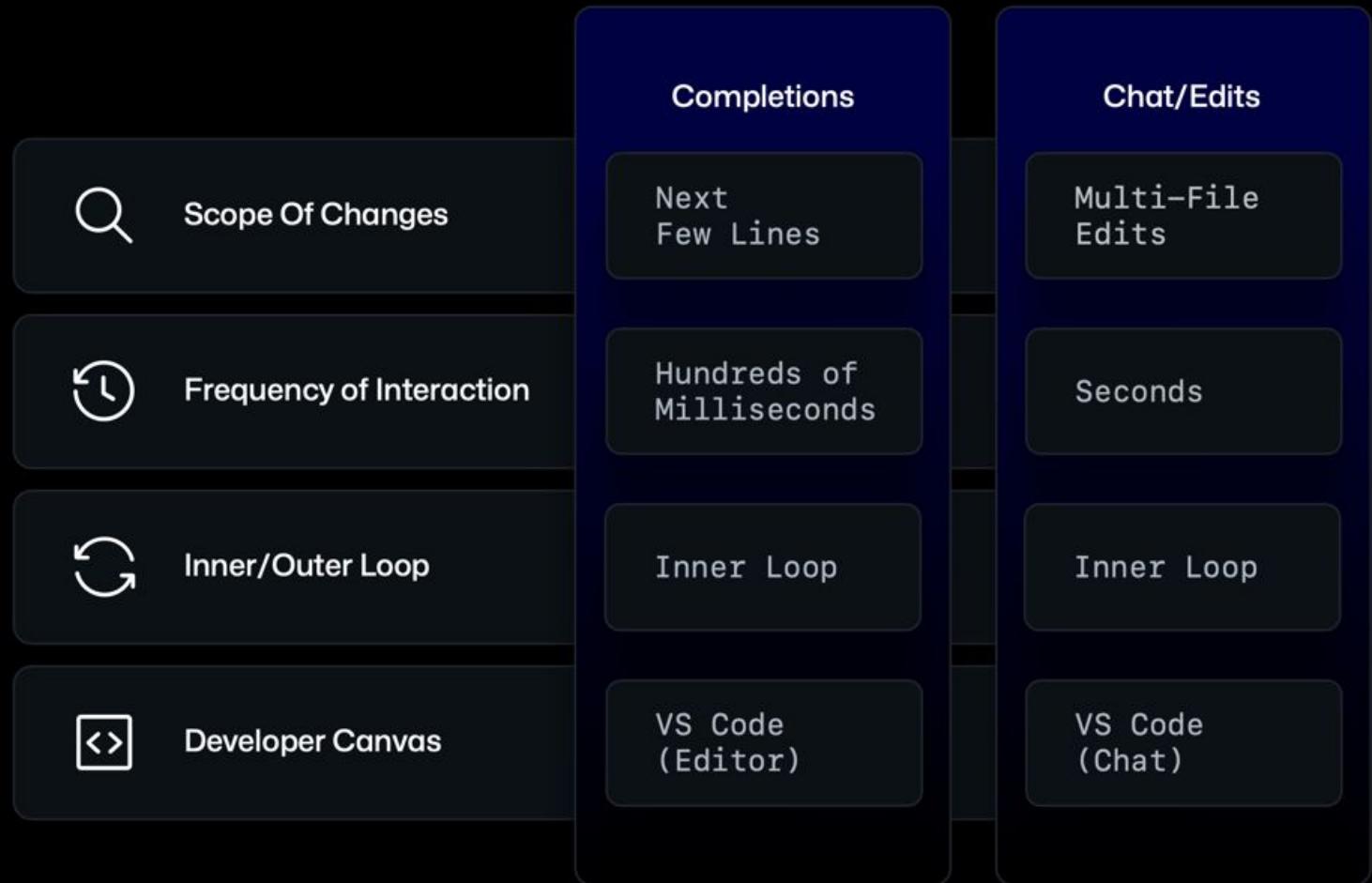
PREVIEW - JUNE 2025

# What is Agent mode ?

## Autonomous Pair Programmer

- ➡️ Runs Copilot in “**plan → act → observe → iterate**” loop on your workspace.
- 📝 Can **read, edit, run, and test code** across multiple files without manual steps.
- 🧠 Acts like an **eager junior dev** with an infinite knowledge
- 🔗 Fully **integrated** inside supported IDEs; no extra CLI or context-switching.

# GitHub Copilot / Code Generation



GA

Chat / Custom instructions



AI



# Tailor-made answers, defined by you

Specify custom instructions to personalize chat responses in VS Code and Visual Studio based on your preferred tools, organizational knowledge, and coding best practices.

Create a new API route for sharing a story 

 GitHub Copilot

Used 5 references

-  [github.copilot.chat.codeGeneration.instruction...](#)
-  [copilot-instructions.md](#) [.github](#)
-  [page.tsx: 39-50](#) app
-  [schema.sql](#) scripts
-  [spec.md](#)

# **copilot-instructions.md** vs **\*.instructions.md**

- Single file
  - Automatically attached in the chat conversation
  - Stored in **.github** directory
- Multiple files specific to file type
  - To be attached to the chat explicitly
  - Stored in **.github/instructions** directory

api.prompt.md

```
.github > prompts > api.prompt.md
1 always add error handle
2
3 following snake case for endpoint
4
5 always add documentation next to api definition
6
7 documentation must be both in french and in english
8
9 always add amadeus as first section of the endpoint address
10
11 always put the api definition in this #file:src/api.js
12
13 implement the backend in both postgres and in-memory databases in
#file:src/database/postgres.js and #file:src/database/in-memory.js
14
15 avoid sql injection and select *
```

+ Add Files...    api.prompt.md

new endpoint to get books by title

attività

GA

Chat / Reusable prompts



AI



**Build and share  
reusable prompt  
instructions with  
additional context.**

# Copilot for code-adjacent tasks

- Writing a CI/CD file
- Writing an IaC file
- Creating Mermaid diagrams

# Public code blocking

- Enable/Disable it, IP indemnity still applies

# Chat vs Edit Mode

**Edit mode:** Multi-file edits.

**Ask mode:** Safe. Just for responses.  
Doesn't change any code directly.

# Utilize /slash commands

/explain

/tests

/fix

@workspace

/help

Explains how the selected code works

Generates unit tests for the selected code

Proposes a fix for the problems in the selected code

Preface your prompts/commands to ask a question about the files in your workspace

Learn how to use Copilot Chat

@workspace /tests

Workspace

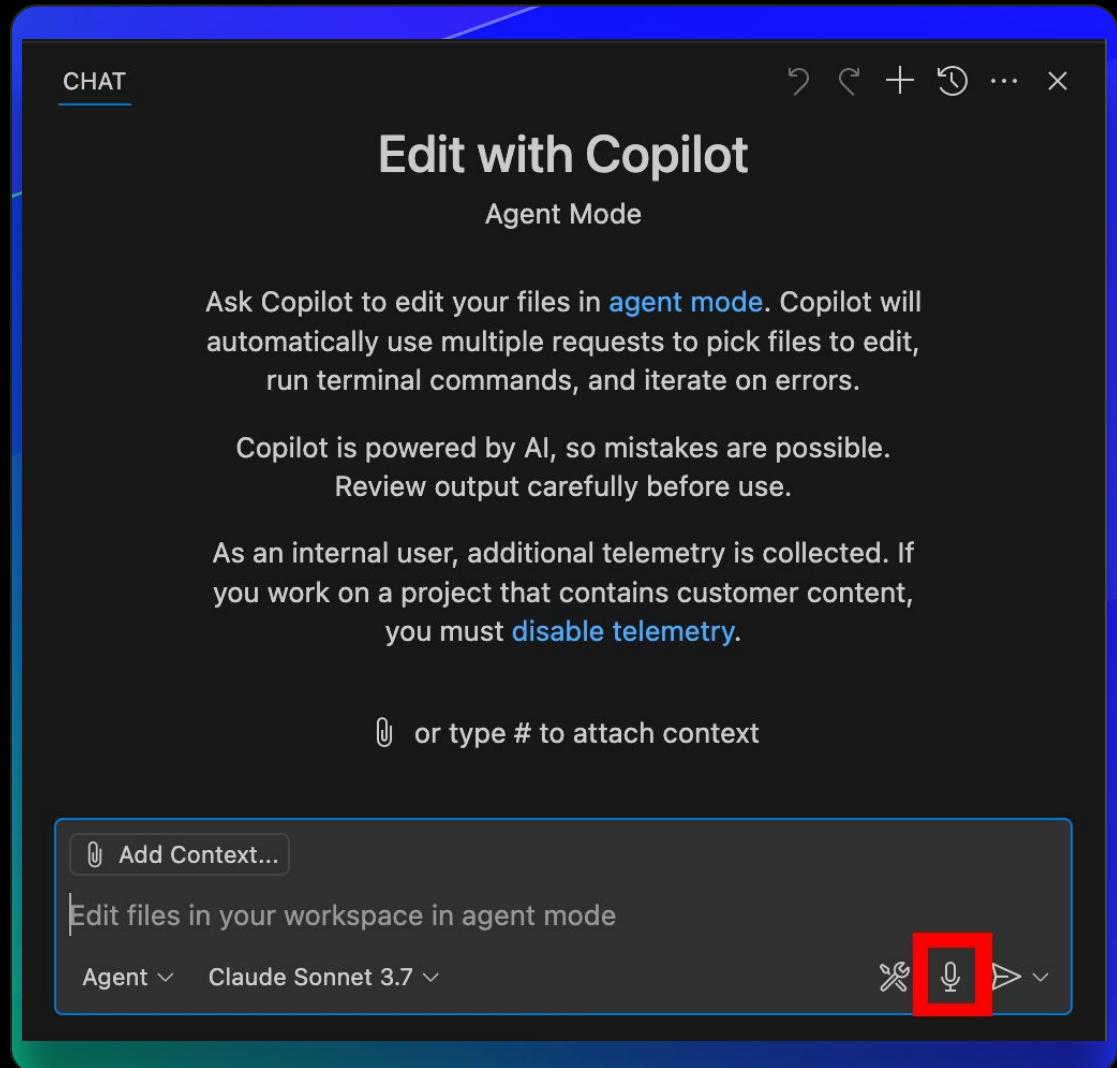
## Plan

1. Import necessary modules for testing.
2. Mock the `fetch` function to simulate API responses.
3. Write test cases to cover:
  - Positive sentiment response.
  - Negative sentiment response.
  - Neutral sentiment response.
  - Error handling.

## Code

TS sentiment.test.ts

```
1+import { isPositive } from './sentiment';
2+import { fetch } from 'fetch-h2';
3+
4+// sentiment.test.ts
5+
6+
7+jest.mock('fetch-h2');
8+
9+describe('isPositive', () => {
10+    it('should return true for positive sentiment',
11+        (fetch as jest.MockedFunction<typeof fetch>)
```



Chat / Voice chat



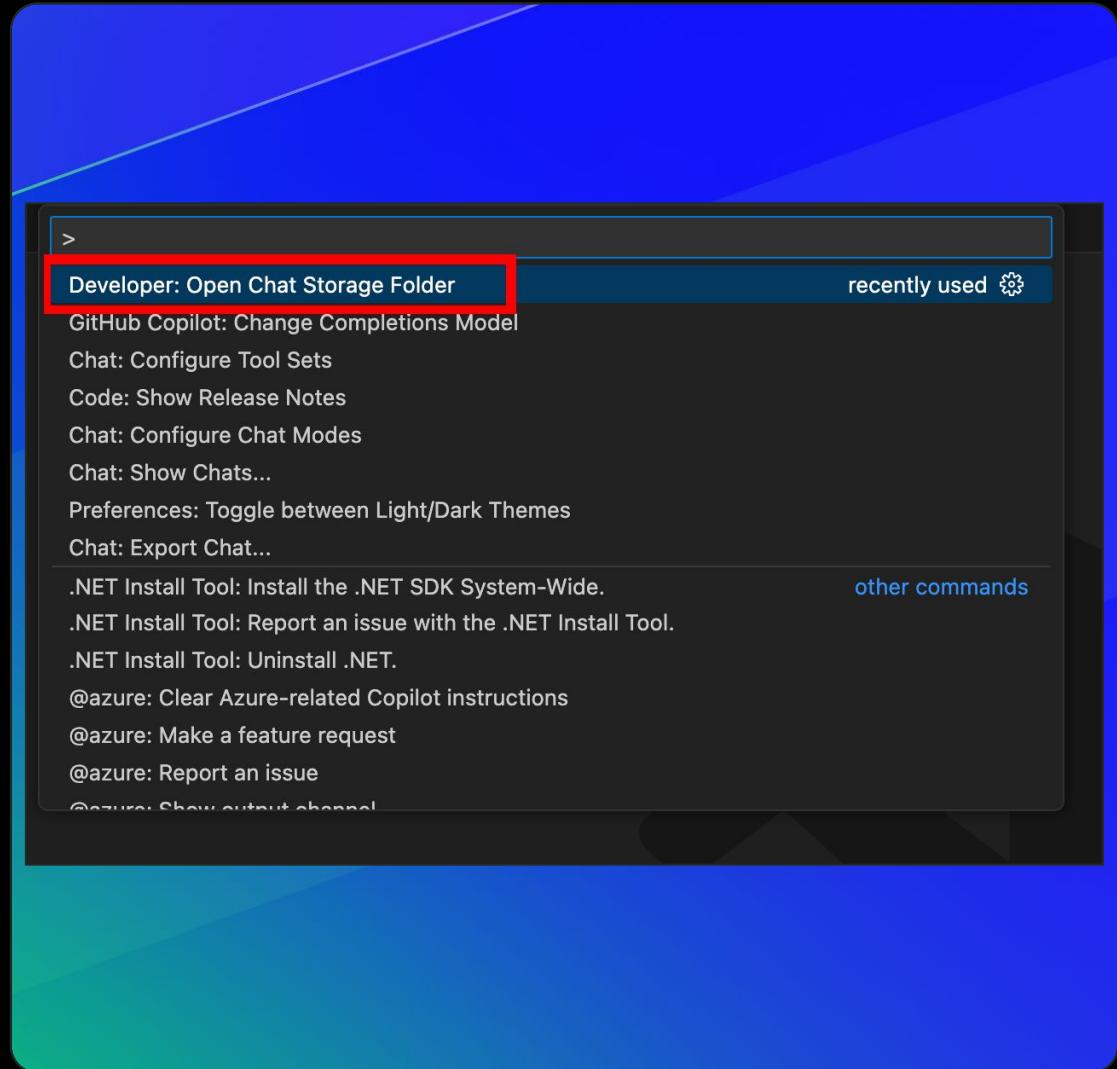
AI

# Talk to Copilot

With the help of VS Code Speech extension, directly talk to copilot chat and convert speech to text

# Utilize **#hash** context variables

#terminalSelection	#new	#problems	#changes	#filename	#codebase
Last command from the active terminal	Scaffold new workspace in VS Code	Check errors for particular files	Get diffs of changed file	File as context to the chat	Codebase as context



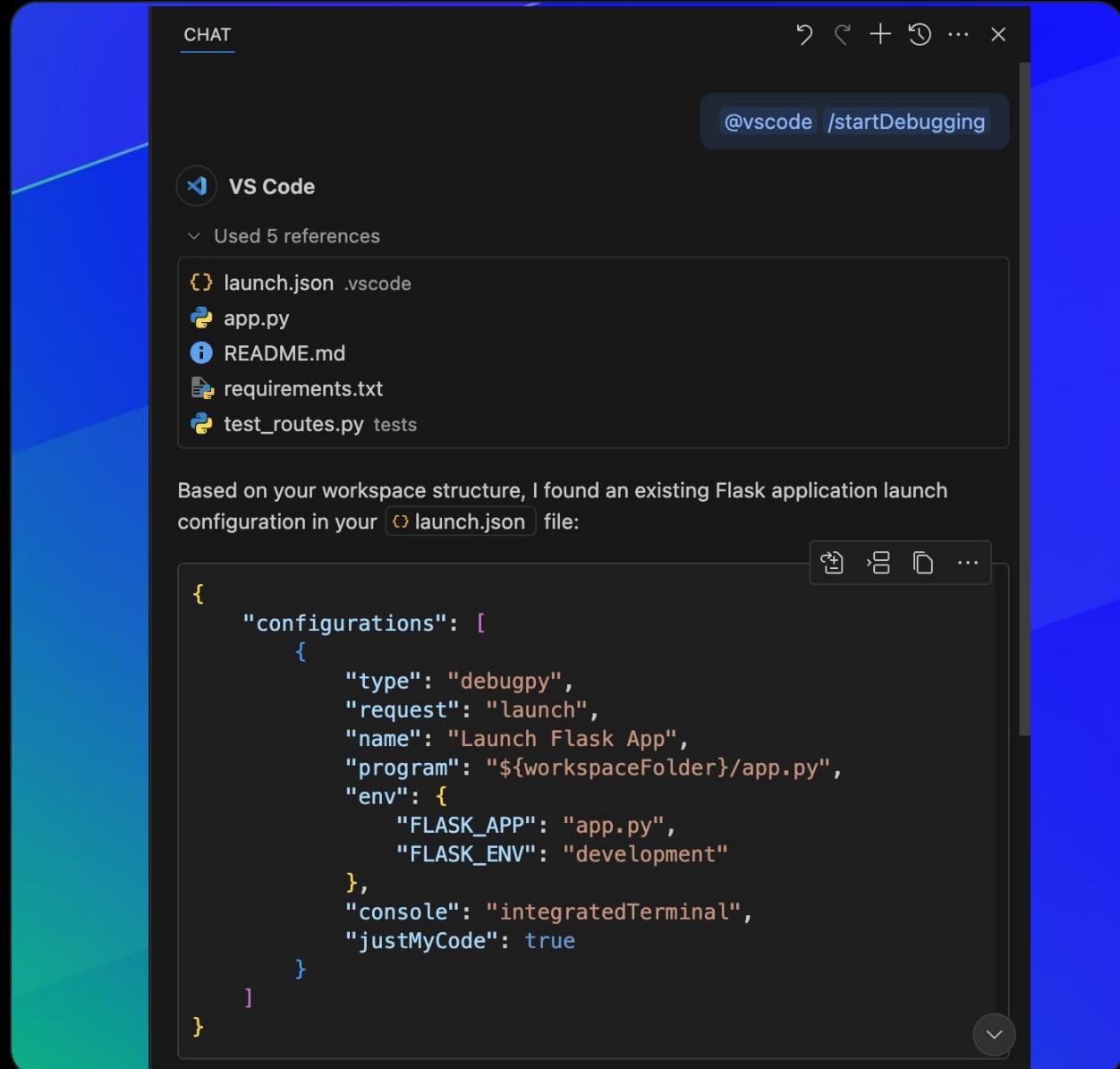
# /save - to save the conversation

Chat conversations written to disk as .json.  
Can save the conversation in Markdown format

# Copilot best practices

- Use the appropriate model for the use case (premium vs base)
- Custom-instructions for all tasks
- Re-usable prompts when needed
- Always review the code generated by Copilot

<https://docs.github.com/en/copilot/using-github-copilot/ai-models/choosing-the-right-ai-model-for-your-task>



## Chat / Debugging

# Generate debug configuration

Generate the debug configuration - **launch.json** file with **/startDebugging**.

Run the app in debug mode using the config

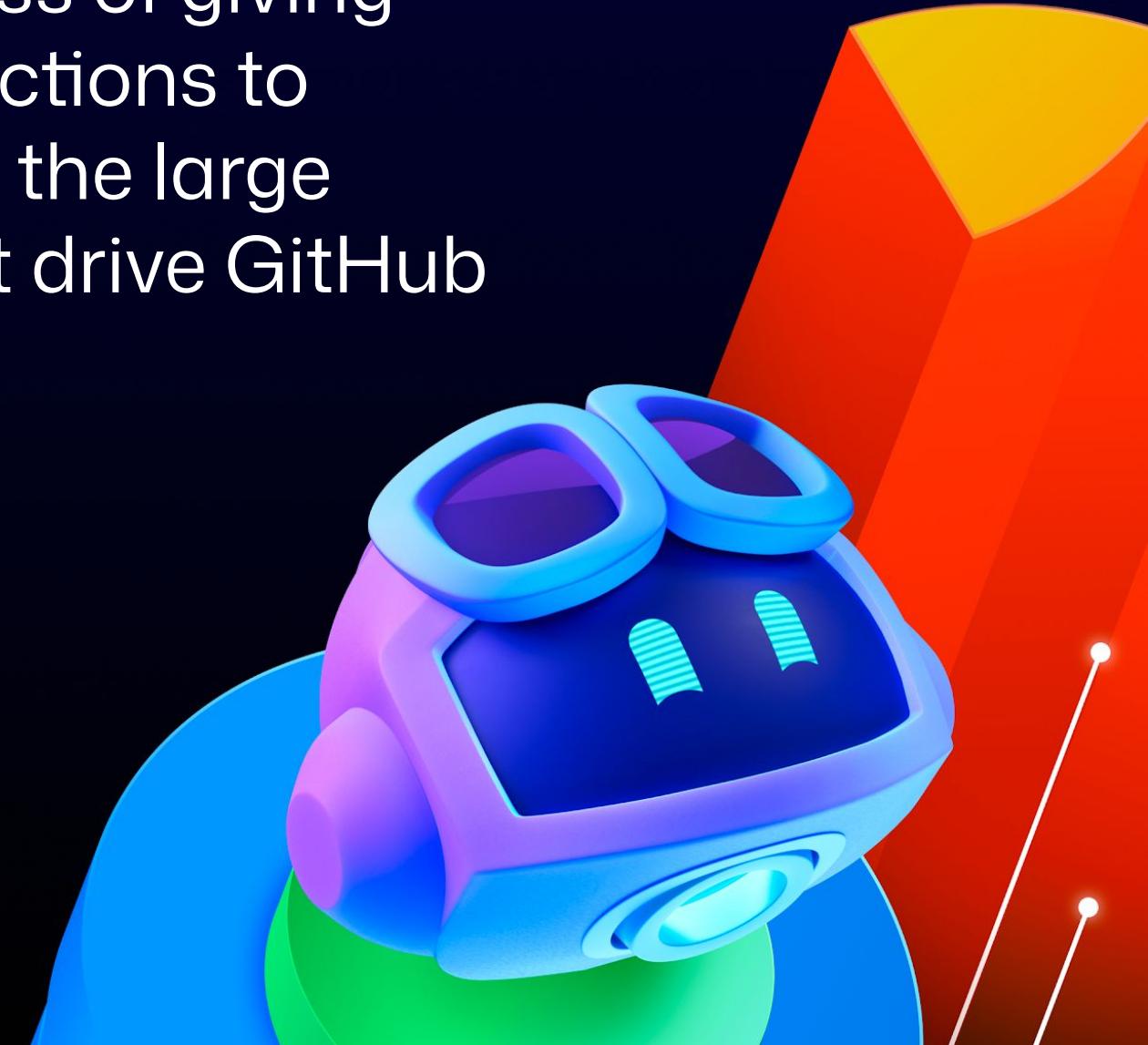
# Prompt Crafting



**Prompt crafting** is the process of giving clear natural language instructions to computer programs, such as the large language models (LLMs) that drive GitHub Copilot functionalities.

- Zero-shot prompting
- Few-shot prompting
- Chain of Thought prompting

<https://www.promptingguide.ai/techniques>



# Context is key

- Copilot relies on both the underlying model - trained on billions of lines of code - *and* the context in your editor to provide high quality suggestions
- Code completion prioritizes the immediate context, as well as context found in open files
- Chat has the ability to bring in even more context via the **@workspace** command to include additional project files that may not be open in the IDE

## LLM is **probabilistic**, not deterministic

Again, LLMs are sophisticated probability engines, so you might not get the same results for the exact same prompts every time

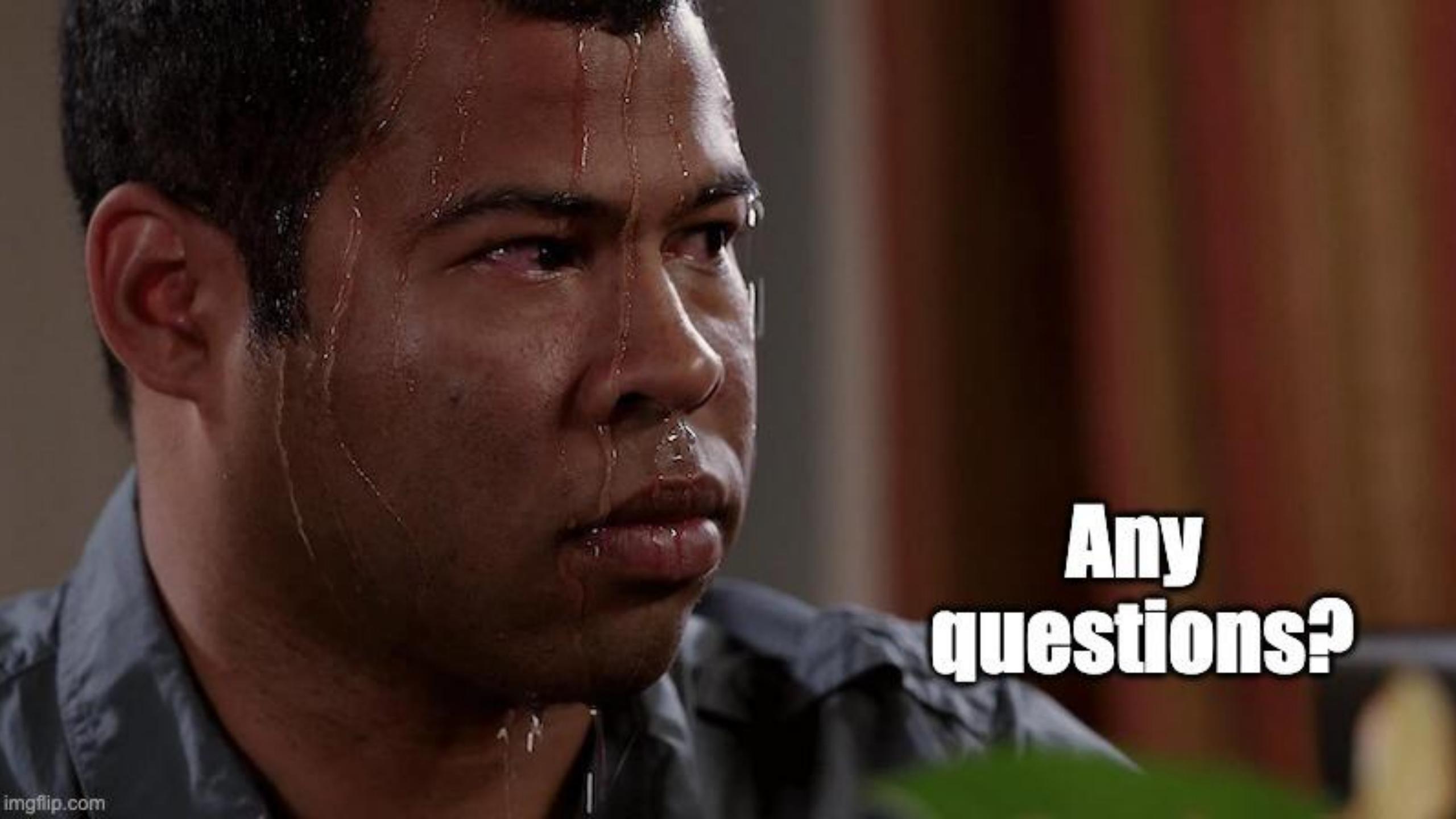
You don't succeed,  
iterate

```
1 //Write a JavaScript function that finds
2 //the maximum value in an array.
3 function max(array) {
4     return Math.max.apply(null, array);
5 }
```



Improvise. Adapt. Overcome

```
1 //Create a JavaScript function, `findMax`, that takes an array of strings as input
2 //and returns the string with the maximum length.
3 //Ensure that the function works correctly for arrays that are empty,
4 //as well as arrays that contain both strings and numbers.
5 function findMax(array) {
6     var max = array[0].length;
7     array.map(v => max = Math.max(max, v.length));
8     result = array.filter(v => v.length == max);
9     return result;
10 }
```

A close-up, profile shot of a man's face, looking directly at the viewer. He has dark skin, short hair, and is sweating heavily, with sweat droplets visible on his forehead, nose, and chin. He is wearing a light-colored, collared shirt. The background is blurred.

**Any  
questions?**



Thank you