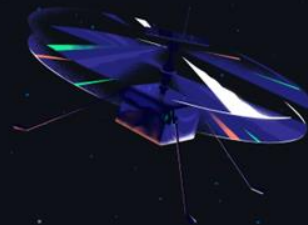




# Copilot for Business Developer training

Level: Intermediate

Let's build from here



## AGENDA

Copilot for Business - Introduction

Coding

Best practices & Prompt Engineering

Workshop (1 - 2 hours long)

Copilot X - Feature Overview

Secure coding

Wrap-up, Q&A

# Coding

GH Copilot vs GH Copilot Chat

Create Workspace

Create Notebook

Data Generation

Testing with GH Copilot

Code Refining and Refactoring

Block Files

Debugging

GH Rest APIs

# When to use Copilot vs Copilot Chat

## Copilot

Direct Code Writing

Seamless IDE Integration

Solo Development

## Copilot Chat

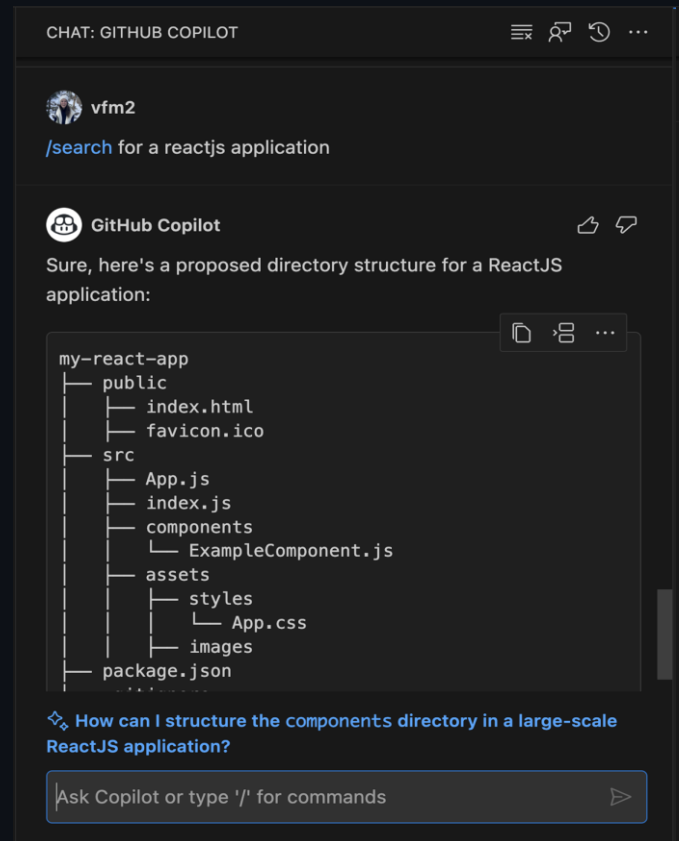
In-Depth Assistance

Learning & Teaching

Collaborative Scenarios

# Create Workspace

**/search** to query  
parameters  
**@Workspace /new** to  
create a workspace for  
your project



# @Workspace

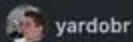
**@Workspace** to  
create a workspace for  
your project

✧ /help What can you do?

**@workspace** tell me how to  
start the development server  
in the current project



# Without using @Workspace



yardobr

how do I start the current project?



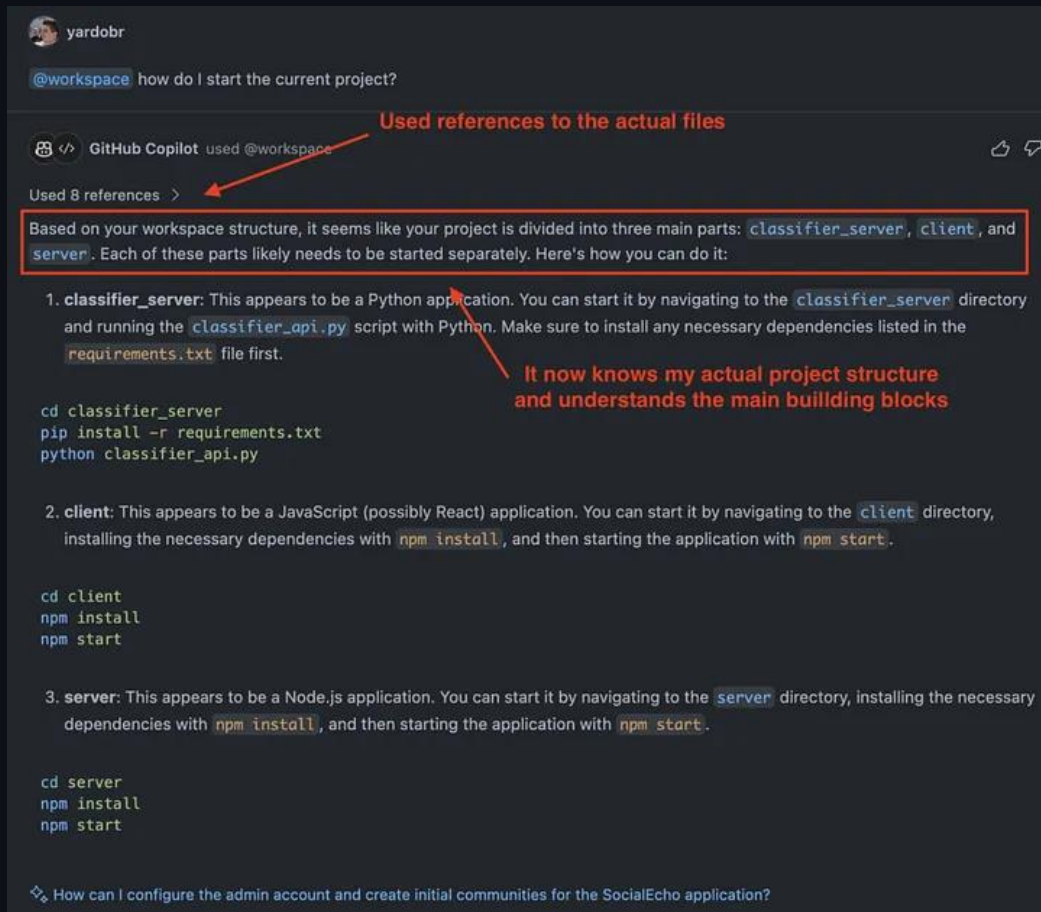
GitHub Copilot

To start a project in Visual Studio Code, you typically need to follow these steps:

1. Open the terminal in Visual Studio Code. You can do this by going to the top menu and selecting `View -> Terminal`, or by using the shortcut `Ctrl+`` (backtick).
2. Navigate to the project's directory. You can do this by typing `cd path/to/your/project` in the terminal, replacing `path/to/your/project` with the actual path to your project.
3. Depending on the type of your project, you will need to run a specific command to start it. Here are some common examples:
  - For a Node.js project, you might run `npm start` or `node app.js` (replace `app.js` with the entry point to your application).
  - For a Python project, you might run `python main.py` (replace `main.py` with the main Python file in your project).
  - For a Java project, you might run `javac Main.java` followed by `java Main` (replace `Main` with the main class in your project).

Please replace the commands with the ones specific to your project. If you're not sure what command to run, check the documentation of the technology you're using or the README file of your project.

# Using @Workspace



The screenshot shows a chat interface with a user named 'yardobr' asking '@workspace how do I start the current project?'. The response from GitHub Copilot is detailed, mentioning three project parts: classifier\_server, client, and server. Red annotations highlight that the model used references to actual files and understood the project structure.

yardobr

@workspace how do I start the current project?

Used references to the actual files

Used 8 references >

Based on your workspace structure, it seems like your project is divided into three main parts: `classifier_server`, `client`, and `server`. Each of these parts likely needs to be started separately. Here's how you can do it:

- 1. classifier\_server:** This appears to be a Python application. You can start it by navigating to the `classifier_server` directory and running the `classifier_api.py` script with Python. Make sure to install any necessary dependencies listed in the `requirements.txt` file first.

It now knows my actual project structure and understands the main building blocks

```
cd classifier_server
pip install -r requirements.txt
python classifier_api.py
```

- 2. client:** This appears to be a JavaScript (possibly React) application. You can start it by navigating to the `client` directory, installing the necessary dependencies with `npm install`, and then starting the application with `npm start`.

```
cd client
npm install
npm start
```

- 3. server:** This appears to be a Node.js application. You can start it by navigating to the `server` directory, installing the necessary dependencies with `npm install`, and then starting the application with `npm start`.

```
cd server
npm install
npm start
```

How can I configure the admin account and create initial communities for the SocialEcho application?



# @Workspace - use cases - Look around the workspace

@workspace Tell me about the current workspace structure

@workspace How do I run this app?

@workspace What is the automated testing tech stack used here?

@workspace Where is the docker-compose.yml for production located?

@workspace where is database connecting string configured?

@workspace how can I validate a date?

@workspace where are tests defined?

# @Workspace - use cases - making plans for complex edits

@workspace how can I add a rich tooltip to a button?

@workspace add date validation to #selection

@workspace add a new API route for the forgot password form

# @Workspace - use cases – add the new functionality

@workspace How do I add a new route in the frontend application?

@workspace Guide me how to create a new REST endpoint here

@workspace Generate code for a new microservice that will be responsible for the notifications

@workspace Create a new lazy-loaded page with a path “/faq”.

# @Workspace - use cases – fix the issues

@workspace when I access the /orders/:id route - I see a 404 error. what could be its root cause?

@workspace the calculations in the PriceTiers component is incorrect but the unit tests show the correct calculations. what's wrong?

@workspace my server responds with “413 Content too large” when I request the /subscriptions endpoint. please investigate and fix.

# @Workspace - use cases – refactor

@workspace how can I restructure my React components to be split by domains?

@workspace how do I make my Angular Products module lazy-loaded?

@workspace how can I split my environment variables located in /server/env into dev and production?

@workspace what are the potential performance bottlenecks of the UserAuthentication controller?

# @Workspace - use cases – Explaining higher-level concepts in a codebase

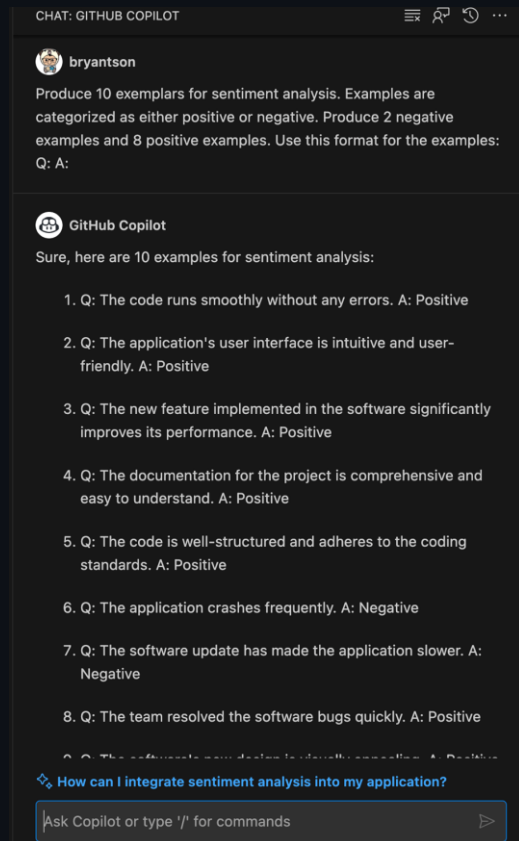
@workspace how is authentication implemented?

@workspace which API routes depend on this service?

How do I build this #codebase? -

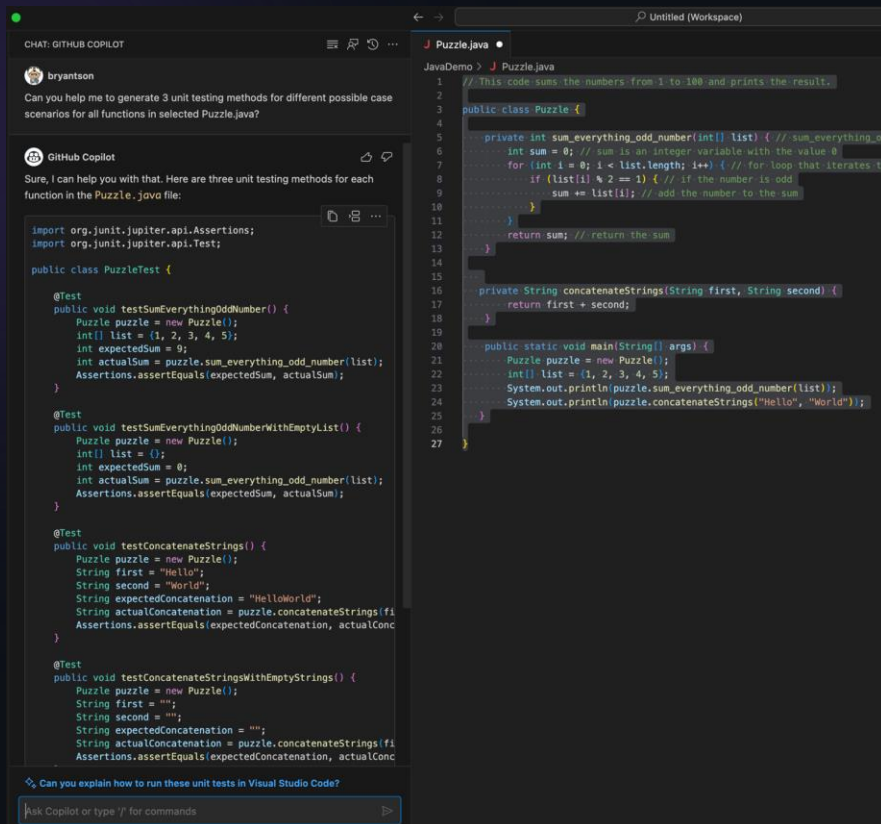
# Data Generation

Ask Copilot to  
generate data



# Unit Testing Generation - Copilot Chat

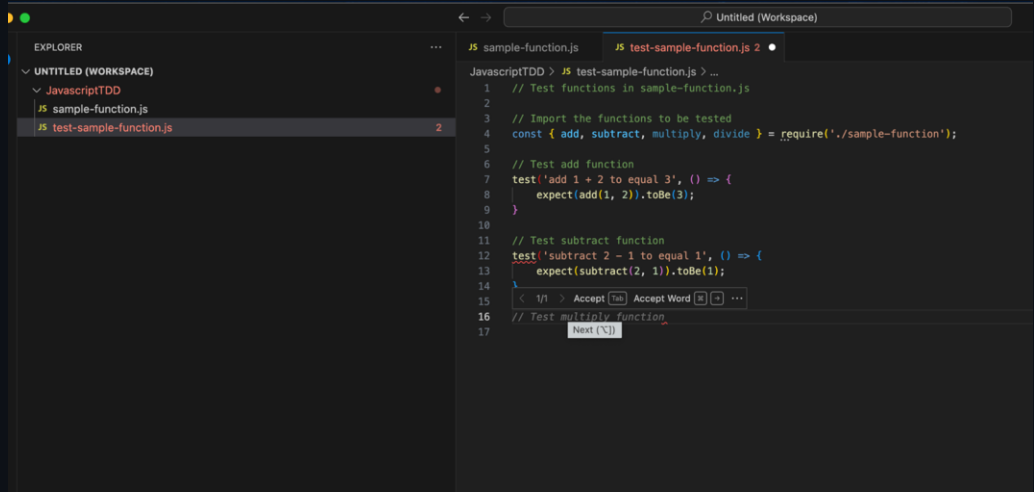
Works best for  
general **unit  
testing** case  
generation





# Test Driven Development (TDD)

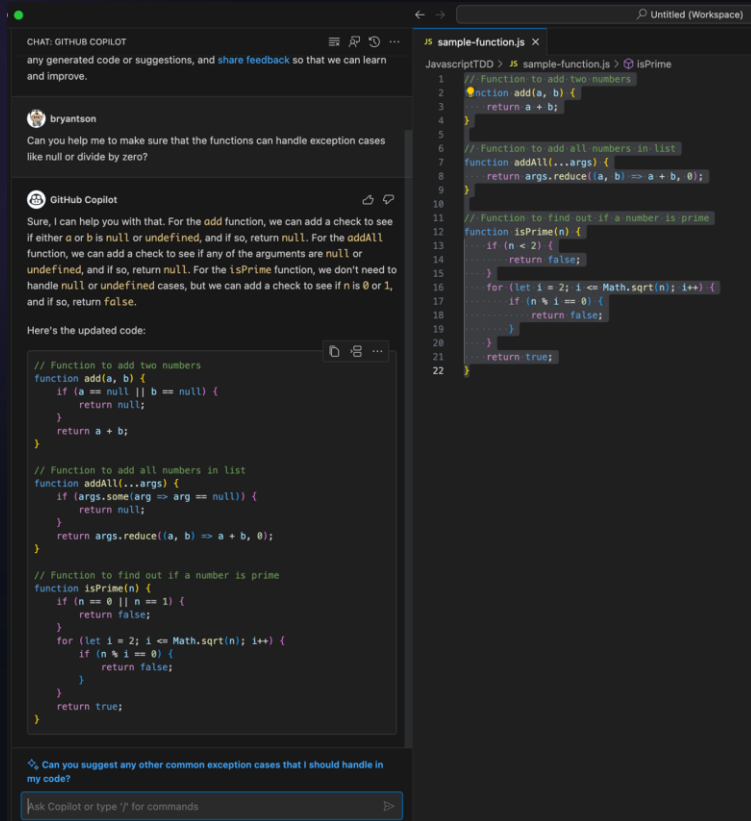
Test Driven  
Development  
(TDD) practice is  
possible



```
JavaScriptTDD > JS test-sample-function.js > ...
1 // Test functions in sample-function.js
2
3 // Import the functions to be tested
4 const { add, subtract, multiply, divide } = require('./sample-function');
5
6 // Test add function
7 test('add 1 + 2 to equal 3', () => {
8   expect(add(1, 2)).toBe(3);
9 })
10
11 // Test subtract function
12 test('subtract 2 - 1 to equal 1', () => {
13   expect(subtract(2, 1)).toBe(1);
14 })
15
16 // Test multiply function
17
```

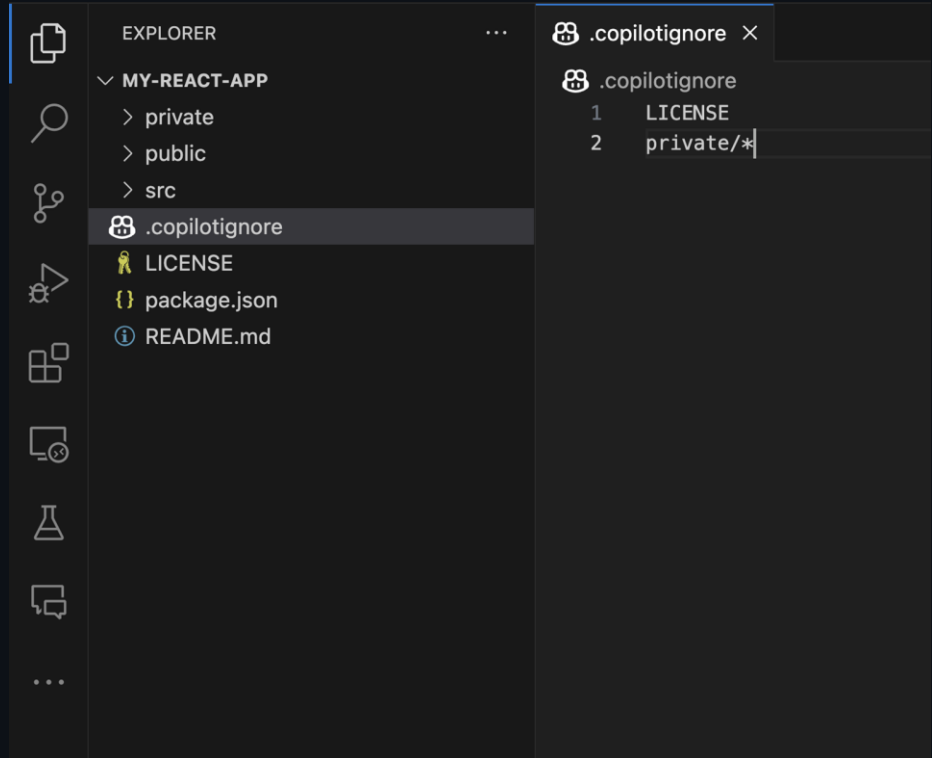
# Code Refactoring

Refactoring is  
possible through  
GitHub Copilot  
Chat



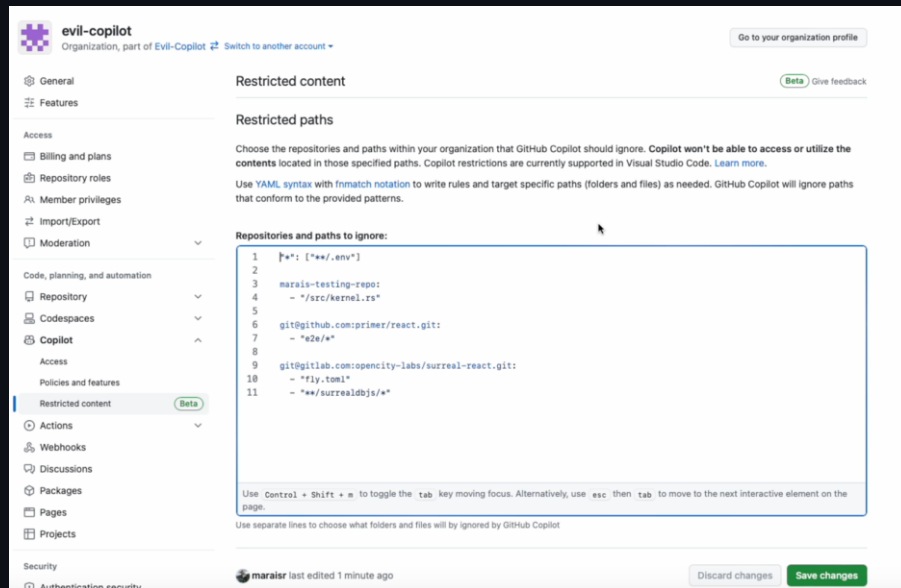
# Block files from Copilot

Use **.copilotignore** to block files and folders from being used by Github Copilot



# Block files from Copilot

Ignore specific  
repositories and  
paths in Restricted  
Content



# Block files from Copilot

YAML

Beside Inline 

```
- "/src/some-dir/kernel.rs"

- "secrets.json"

- "secret*"

- "*.cfg"

- "/scripts/**"
```

# Code Debugging

## Debugging with GitHub Copilot

```
42     def collisions(self):
43         if pygame.sprite.spritecollide(self.plane, self.collision_sprites,
44         or self.plane.rect.top <= 0:
45             for sprite in self.collision_sprites.sprites():
46                 if sprite.sprite_type == 'obstacle':
46+                 if sprite.sprite_type == 'obstacle':

/fix Expected ":"
Accept Discard [v] [refresh] Changed 1 line [thumbs up] [thumbs down]
```

```
47             sprite.kill()
48             self.active = False
49             self.plane.kill()
50
```

# GitHub Rest API

## GitHub Copilot Rest API

GET /orgs/{org}/copilot/billing

cURL JavaScript GitHub CLI

```
# GitHub CLI api
# https://cli.github.com/manual/gh_api

gh api \
-H "Accept: application/vnd.github+json" \
-H "X-GitHub-Api-Version: 2022-11-28" \
/orgs/ORG/copilot/billing
```

Example response Response schema

Status: 200

```
{
  "seat_breakdown": {
    "total": 12,
    "added_this_cycle": 9,
    "pending_invitation": 0,
    "pending_cancellation": 0,
```

# Best practices

Getting accuracy closer to expectation

- Prompt Engineering

Working at project(s) level

Leveraging Copilot to increase code quality

Addressing concern around security exploit



# Getting accuracy closer to expectation

## Problems



Copilot fails to produce answer or to keep repeating



Copilot generates incorrect result



Library/module version discrepancies issue



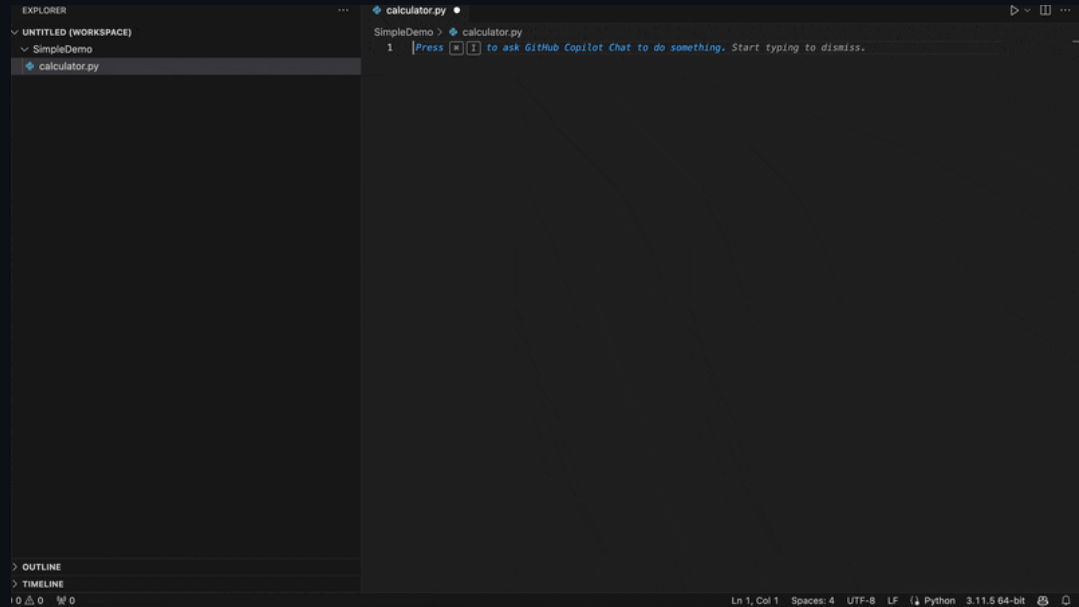
Copilot suggests non-optimal solution

# Problems #1:

Copilot fails to produce answer or keep repeating

## Some problems

- Fails to produce answer
- Hallucination - Keeps repeating

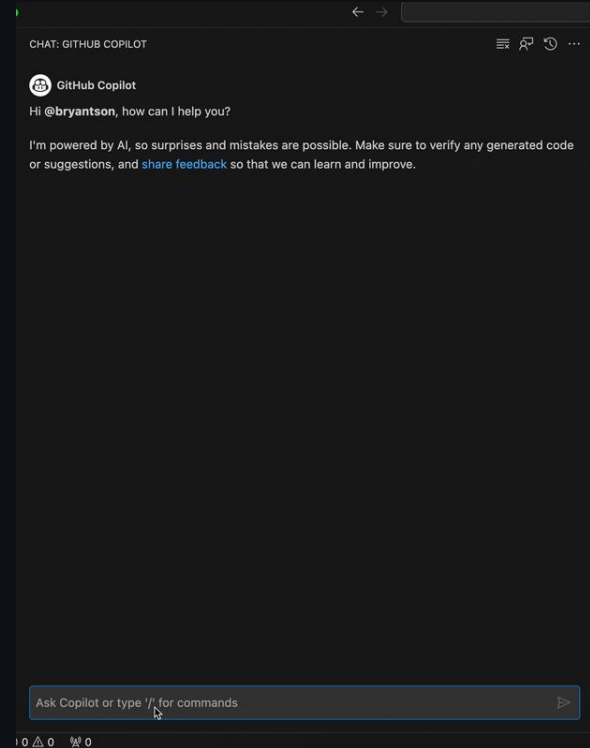


# Problems #2:

Copilot generates incorrect result

Reasons why...

- Not enough context
- Old trained data
- Generative in nature



# What is a Prompt?



*In the context of Copilot, a prompt is a piece of code or natural language description that is used to generate code suggestions. It is the input that Copilot uses to generate its output.*

- Copilot

# What is Prompt Engineering?



*Prompt engineering is the process of designing and creating high-quality prompts that can be used to generate accurate and useful code suggestions with Copilot.*

- Copilot

# Why is Prompt Engineering Important?



- Maximizes utility of Copilot
- Minimizes repeated iterations
- Determines code suggestion quality
- Skill prompts guide Copilot to understand context & nuances
- Refined interactions reduces misunderstandings

# Techniques



Zero-Shot  
Prompting



One-Shot  
Prompting



Few-Shot  
Prompting

# Techniques

## Zero-Shot Programming with Copilot



abelberhane

Write me unit tests for all of the operations in my calculator.js file.



GitHub Copilot

Sure! Here's an example of a test suite that covers all of the operations in a `calculator.js` file:

```
const { calculate } = require('./calculator');

describe('calculator', () => {
  describe('addition', () => {
    it('adds 1 + 2 to equal 3', () => {
      expect(calculate(1, 2, '+')).toBe(3);
    });

    it('adds -5 + 7 to equal 2', () => {
      expect(calculate(-5, 7, '+')).toBe(2);
    });

    it('adds 0.1 + 0.2 to equal 0.3', () => {
      expect(calculate(0.1, 0.2, '+')).toBeCloseTo(0.3);
    });
  });

  describe('subtraction', () => {
    it('subtracts 5 - 3 to equal 2', () => {
      expect(calculate(5, 3, '-')).toBe(2);
    });
  });
});
```



# Techniques

## One or few shot Programming with Copilot



abelberhane

Write me unit tests for all of the operations in my calculator.js file.

Use the following example: `test('adds 1 + 2 to equal 3', () => { expect(calculate(1, 2, '+')).toBe(3); });`



GitHub Copilot

Here are some examples of unit tests for all of the operations in a calculator.js file:

```
const { calculate } = require('./calculator');

test('adds 1 + 2 to equal 3', () => {
  const result = calculate(1, 2, '+');
  expect(result).toBe(3);
});

test('subtracts 5 - 3 to equal 2', () => {
  const result = calculate(5, 3, '-');
  expect(result).toBe(2);
});

test('multiplies 2 * 4 to equal 8', () => {
  const result = calculate(2, 4, '*');
  expect(result).toBe(8);
});

test('divides 10 / 5 to equal 2', () => {
  const result = calculate(10, 5, '/');
  expect(result).toBe(2);
});

test('throws an error for an invalid operation', () => {
  expect(() => calculate(1, 2, '%')).toThrow('Invalid operation');
});

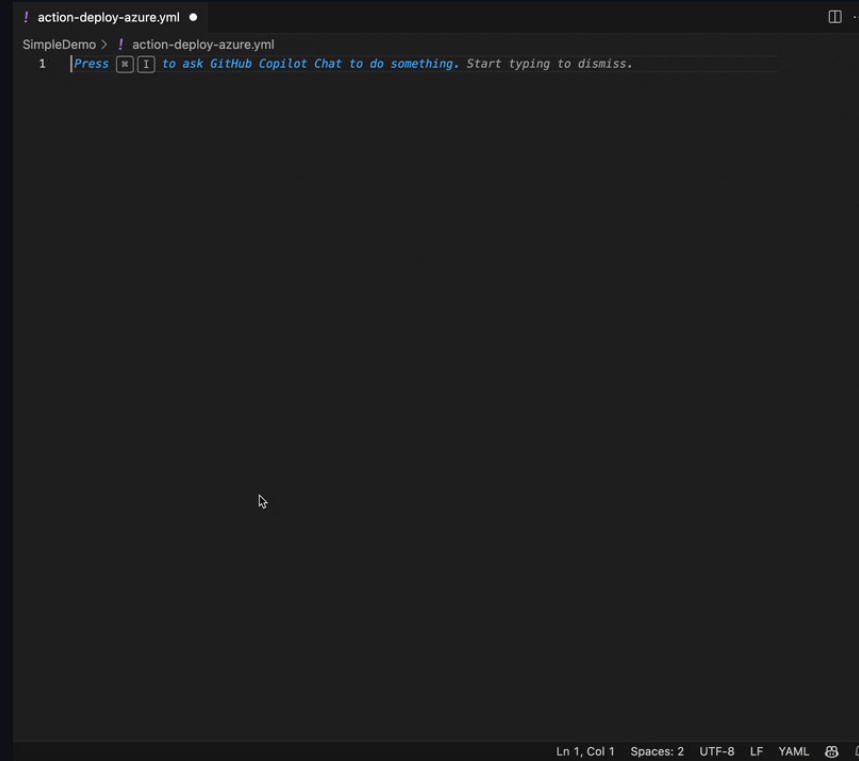
test('throws an error for division by zero', () => {
  expect(() => calculate(1, 0, '/')).toThrow('Division by zero');
});
```

# Problems #3:

Library/module version discrepancy

Old trained data

- While packages go through frequent updates, Copilot does not use latest data

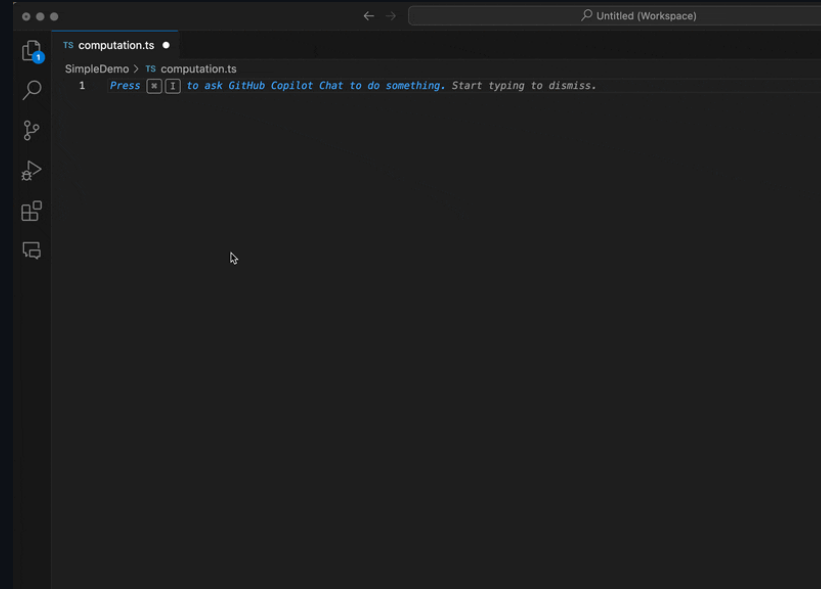


# Problems #4:

Copilot suggests non-optimal solution

Although solution works...

Suggested solution is not optimal because Quick Sort can be implemented in  $O(1)$  complexity, meaning no space required



# How to improve Problem #4

Copilot suggests non-optimal solution

How to improve

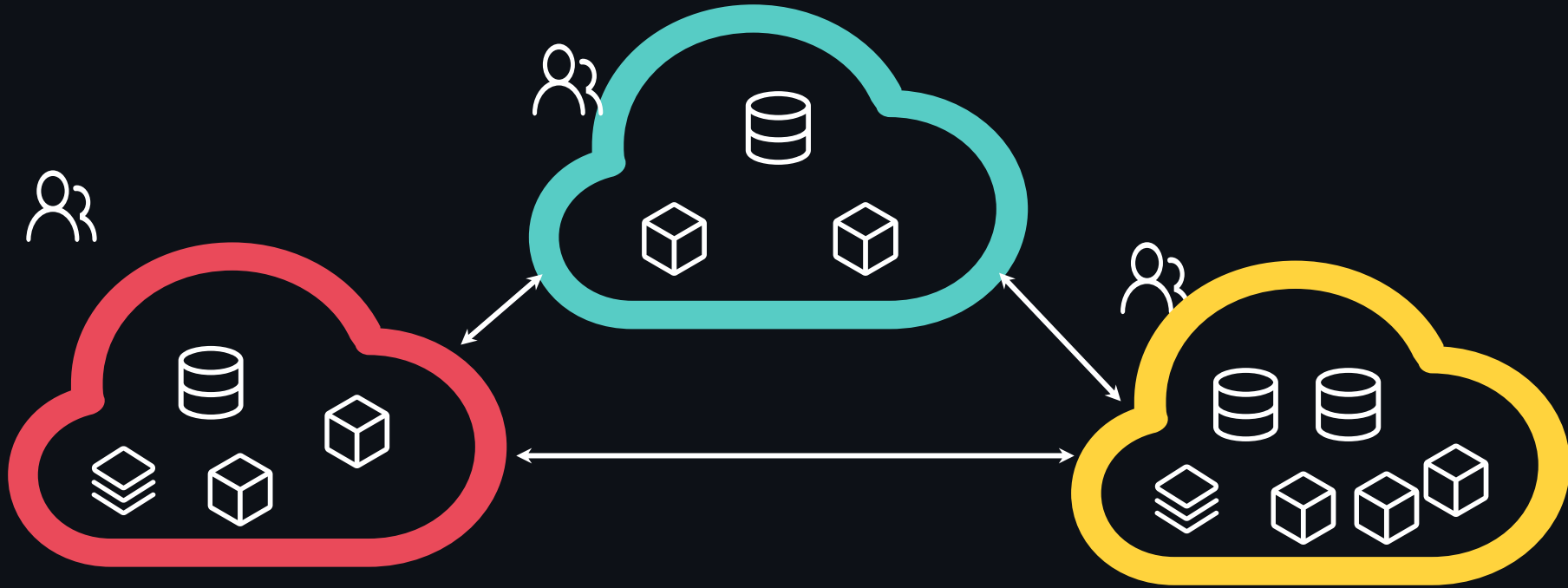
- Ask in Copilot Chat
- Use in-line suggestion from Copilot

A screenshot of a Visual Studio Code editor window. The title bar shows 'SimpleDemo > TS computation.ts 2' and 'quickSort'. The editor displays a TypeScript file with a quick sort algorithm. The code is as follows:

```
1 // Write a quick sort algorithm for me
2 function quickSort(arr) {
3   if (arr.length <= 1) {
4     return arr;
5   }
6   var pivot = arr[0];
7   var left = [];
8   var right = [];
9   for (var i = 1; i < arr.length; i++) {
10    if (arr[i] < pivot) {
11      left.push(arr[i]);
12    }
13    else {
14      right.push(arr[i]);
15    }
16  }
17  return quickSort(left).concat([pivot]).concat(quickSort(right));
18 }
```

The editor has a dark theme and a sidebar on the left with icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The top bar shows 'SimpleDemo > TS computation.ts 2' and 'quickSort'.

# How to work at project(s) level



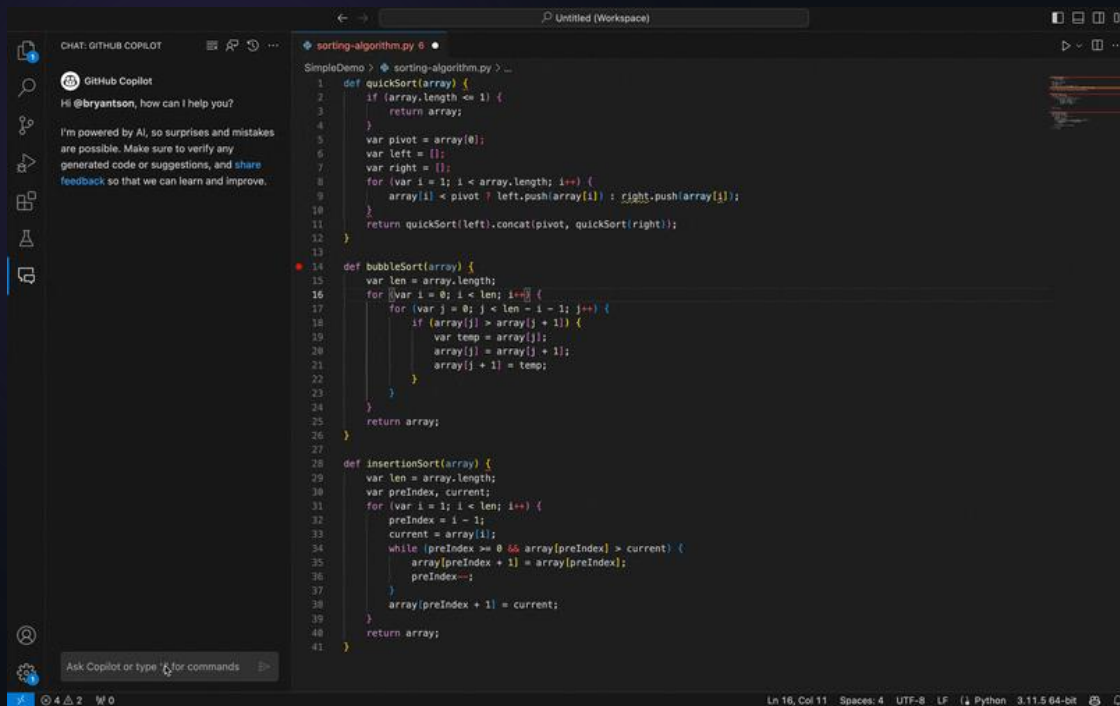
# Neighboring Tabs

```
TS sentiments.ts  write_sql.go  parse_expenses.py  addresses.rb

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

# Copilot Chat: Ask to create tests

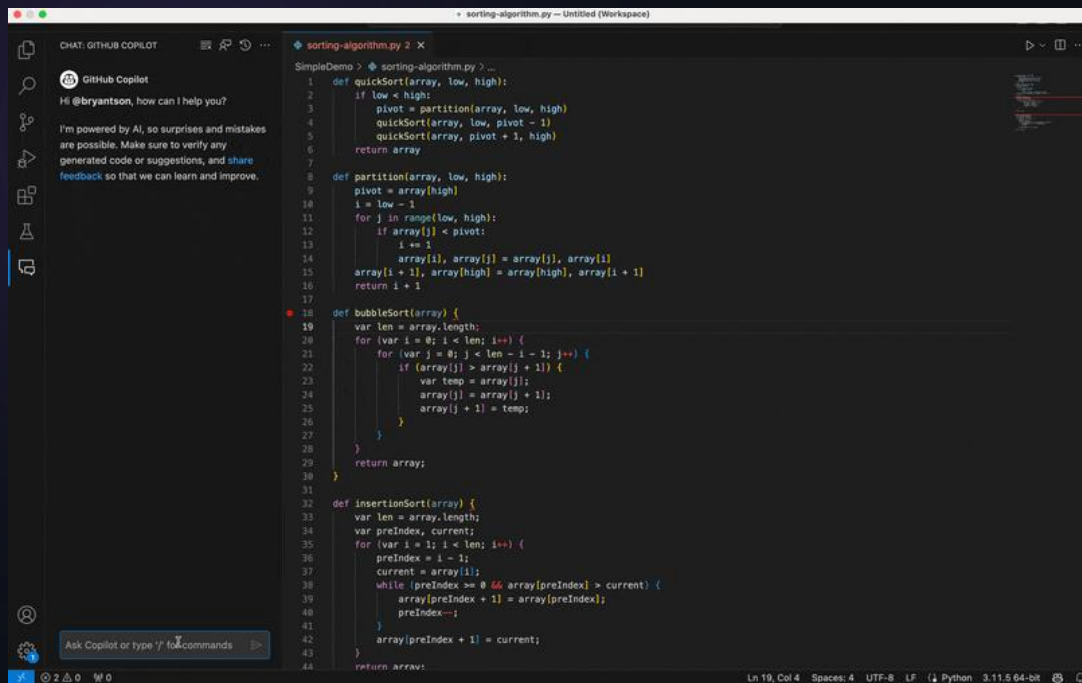
Ask GitHub  
Copilot Chat to  
refactor your  
code



GitHub Copilot Chat (default)

# Copilot Chat: Ask to generate tests

Ask GitHub  
Copilot Chat to  
generate tests

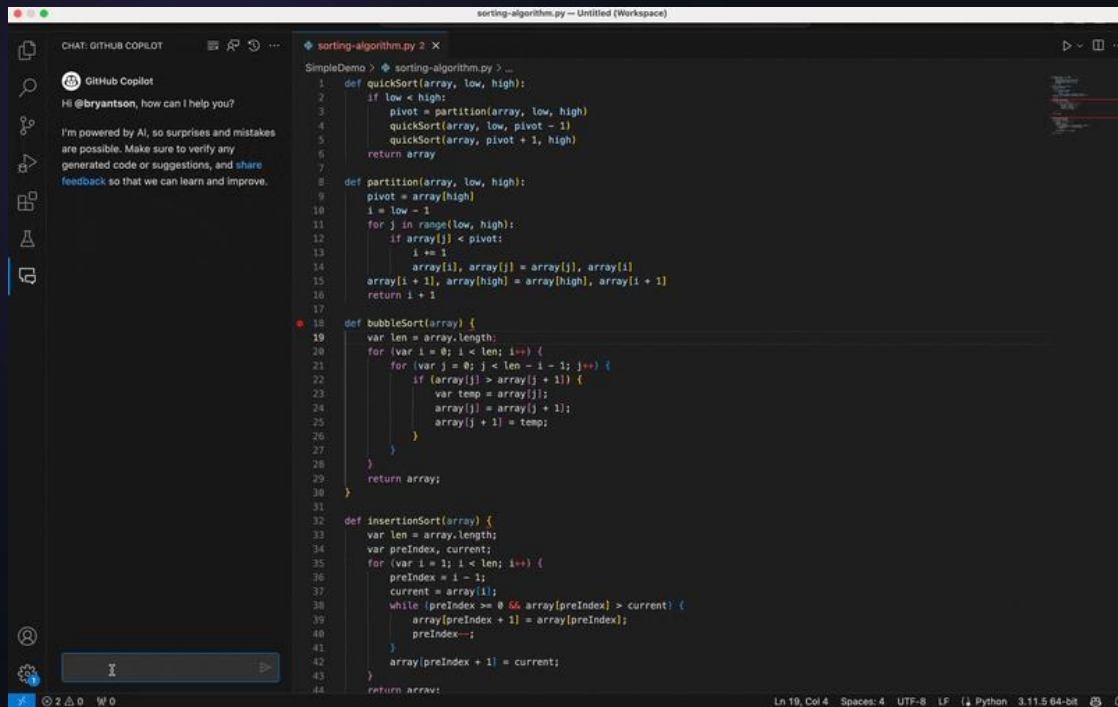


GitHub Copilot Chat (default)



# /fix, /tests and other / command options

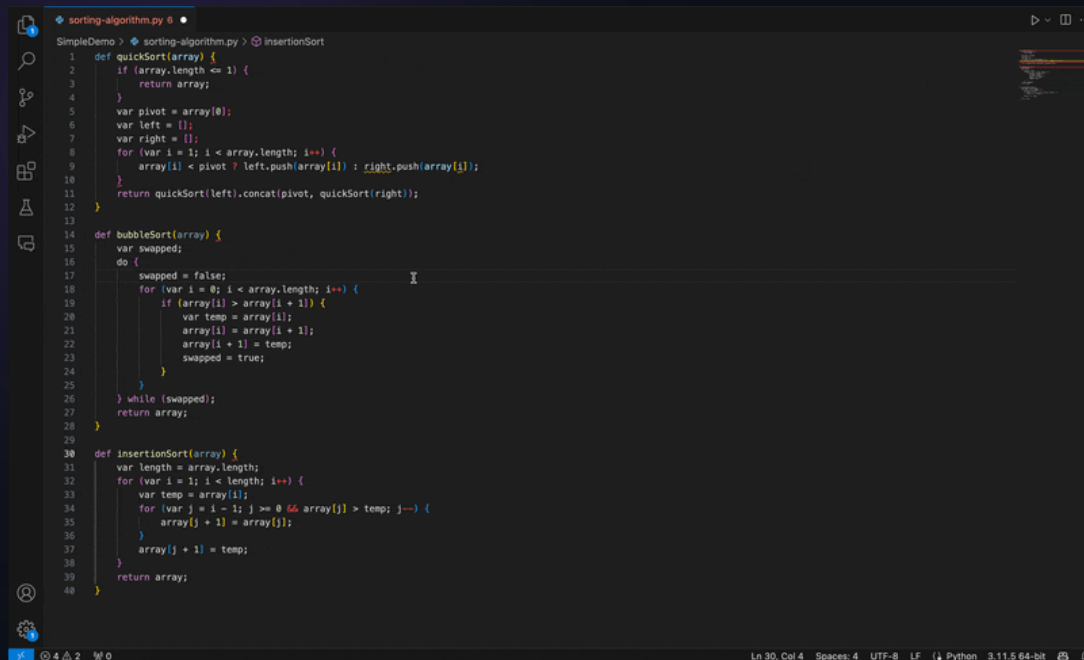
Leverage /  
command  
options that can  
help to improve  
your code



GitHub Copilot Chat (default)

# In-file Copilot options

Copilot now  
offers **in-file**  
**Copilot** feature  
to selectively  
improve



The screenshot shows a code editor with a file named 'sorting-algorithm.py'. The code contains three sorting functions: quickSort, bubbleSort, and insertionSort. The quickSort function is partially completed, and the bubbleSort function is also partially completed. The insertionSort function is shown at the bottom. The editor interface includes a sidebar on the left with icons for file explorer, search, and other tools. The status bar at the bottom indicates the current line and column (Ln 30, Col 4) and other settings like Spaces: 4, UTF-8, LF, Python, 3.11.5 64-bit.

```
1 def quickSort(array) {  
2   if (array.length <= 1) {  
3     return array;  
4   }  
5   var pivot = array[0];  
6   var left = [];  
7   var right = [];  
8   for (var i = 1; i < array.length; i++) {  
9     array[i] < pivot ? left.push(array[i]) : right.push(array[i]);  
10  }  
11  return quickSort(left).concat(pivot, quickSort(right));  
12 }  
13  
14 def bubbleSort(array) {  
15   var swapped;  
16   do {  
17     swapped = false;  
18     for (var i = 0; i < array.length; i++) {  
19       if (array[i] > array[i + 1]) {  
20         var temp = array[i];  
21         array[i] = array[i + 1];  
22         array[i + 1] = temp;  
23         swapped = true;  
24       }  
25     }  
26   } while (swapped);  
27   return array;  
28 }  
29  
30 def insertionSort(array) {  
31   var length = array.length;  
32   for (var i = 1; i < length; i++) {  
33     var temp = array[i];  
34     for (var j = i - 1; j >= 0 && array[j] > temp; j--) {  
35       array[j + 1] = array[j];  
36     }  
37     array[j + 1] = temp;  
38   }  
39   return array;  
40 }
```

GitHub Copilot Chat (default)

# Copilot Security Exploit?

## Prompt Injection

Carefully crafted prompts to make the model to ignore its original instruction or perform unintended actions



# Secure coding

Copilot and secure coding

Copilot + GitHub Advanced Security

Vulnerability detection

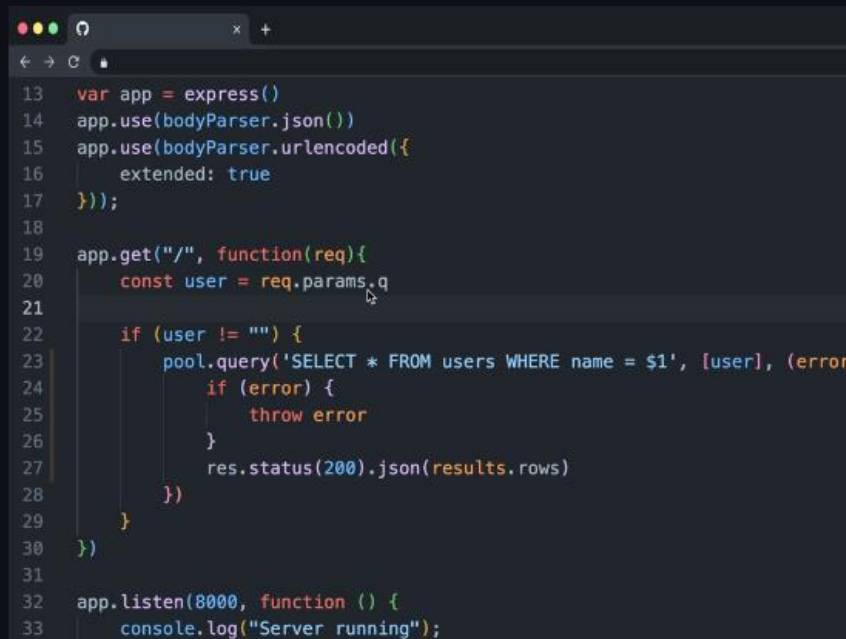
Vulnerability remediation

Copilot Trust Center

Workshop

# Copilot and secure coding

- AI-based vulnerability system that helps prevent insecure coding patterns (e.g. SQL script injection)
- Vulnerability filters are applied only to the Copilot-generated suggestion itself
  - ✗ It cannot detect downstream vulnerabilities introduced by the code e.g. on deployment infrastructure
  - i **We recommend taking the same precautions you take with code written by engineers (linting, code scanning, etc.)**
- Copilot Chat can be used to query code for known vulnerabilities



```
13 var app = express()
14 app.use(bodyParser.json())
15 app.use(bodyParser.urlencoded({
16   extended: true
17 }));
18
19 app.get("/", function(req){
20   const user = req.params.q
21
22   if (user != "") {
23     pool.query('SELECT * FROM users WHERE name = $1', [user], (error, results) => {
24       if (error) {
25         throw error
26       }
27       res.status(200).json(results.rows)
28     })
29   }
30 })
31
32 app.listen(8000, function () {
33   console.log("Server running");
34 })
```

# Copilot + GHAS

- Copilot is not a replacement of GHAS features
- Copilot can be used in tandem with GHAS features to detect and remediate vulnerabilities earlier during the SDLC
  - GHAS Code scanning results
  - GHAS Secret scanning

# Vulnerability detection

- Prompt for insecure coding patterns
- Use GHAS Code Scanning results
- Write custom CodeQL queries
- Increase your knowledge of secure coding patterns
- Create custom Secret Scanning patterns



stebje

It looks like this code is insecure. Help me understand what the issue is and how to resolve it.



GitHub Copilot

Yes, this code is vulnerable to SQL injection attacks. The issue is that the `search` parameter is directly concatenated into the SQL query string without any sanitization or validation. This means that an attacker could potentially inject malicious SQL code into the query, which could lead to data theft or destruction.

To fix this issue, you should use parameterized queries instead of concatenating strings. Parameterized queries use placeholders for the input values, which are then bound to the query separately. This ensures that the input values are properly sanitized and validated before being used in the query.

Here's an example of how to use parameterized queries with the `pg` library:

```
const search = req.params.q;

if (search !== '') {
  const sqquery = "SELECT * FROM users WHERE name = $1";
  const values = [search];
  pool.query(sqquery, values, (err, res) => {
    console.log(err, res);
    pool.end();
  });
}
```

In this example, the `$1` placeholder is used for the `search` parameter, and the actual value is passed in as an array of values to the `pool.query()` function. This ensures that the `search` parameter is properly sanitized and validated before being used in the query.

```
4
5 const pool = new Pool( config: {
6   user: 'postgres',
7   host: 'localhost',
8   database: 'postgres',
9   password: 'process.env.POSTGRES_PW',
10  port: 5432,
11 });
12
13 var app = express();
14 app.use(bodyParser.json());
15 app.use(bodyParser.urlencoded({ extended: true }));
16
17 app.get('/', function (req, res) {
18   const search = req.params.q;
19
20   if (search !== '') {
21     var sqquery = "SELECT * FROM users WHERE name = \"\" + search + \"\"";
22     pool.query( queryTextOrConfig: sqquery, callback: (err, res) => {
23       console.log( message: err, optionalParams[0]: res);
24       pool.end();
25     });
26   }
27 });
28
29 app.listen( port: 8000, callback: function () {
30   console.log( message: 'Example app listening on port 8000!');
31 });
```

# Vulnerability remediation

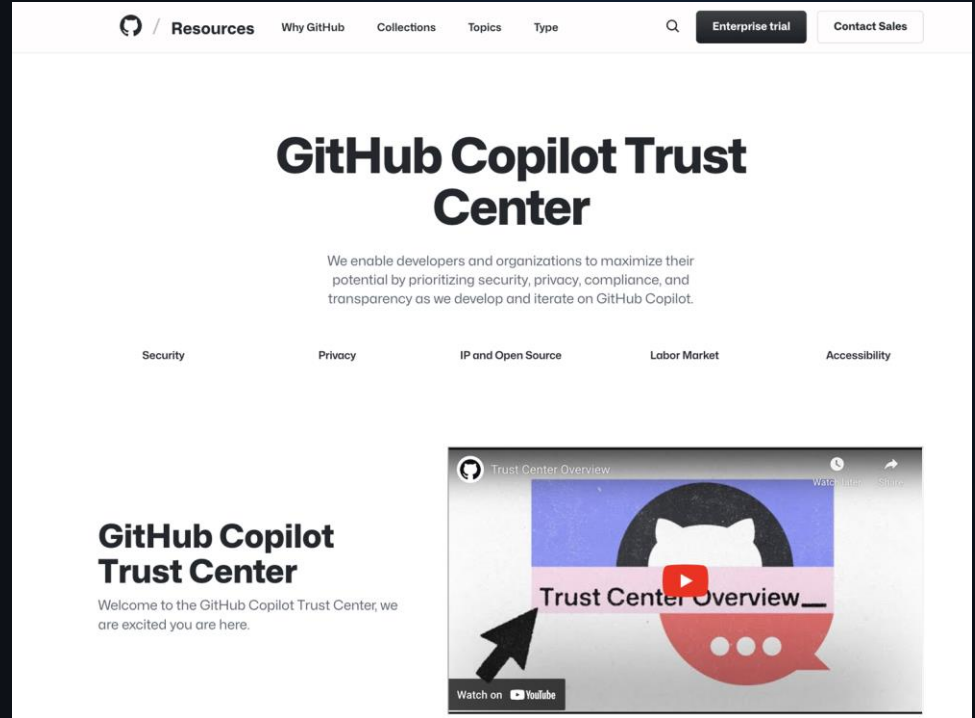
- Copilot helps prevent suggestions that contain insecure coding patterns in real-time
- Automatically fix insecure code based on Copilot suggestions
- Validate and improve existing CodeQL queries



# Security & Trust

## Copilot Trust Center

- Security
- Privacy
- Data flow
- Copyright
- Labor market
- Accessibility
- Contracting



The background of the slide is a deep blue and black cosmic scene. It features numerous small, bright white stars scattered across the field. On the right side, there are larger, more complex structures resembling nebulae or star-forming regions, with glowing blue and purple hues. Some stars exhibit prominent diffraction spikes, giving them a starburst appearance. The overall texture is grainy and ethereal, typical of astronomical imagery.

# Wrap Up



Thank you