



**GitHub**  
Copilot

# Copilot Guide

**Aurimas Aleksandras Nausėdas**

# Agenda

1. Copilot
2. More than autocomplete
3. Creating
4. Choosing alternative suggestions
5. Generating code from a comment
6. Using a framework
7. Configuring Copilot
8. Questions



# Terms

- Copilot
- GitHub

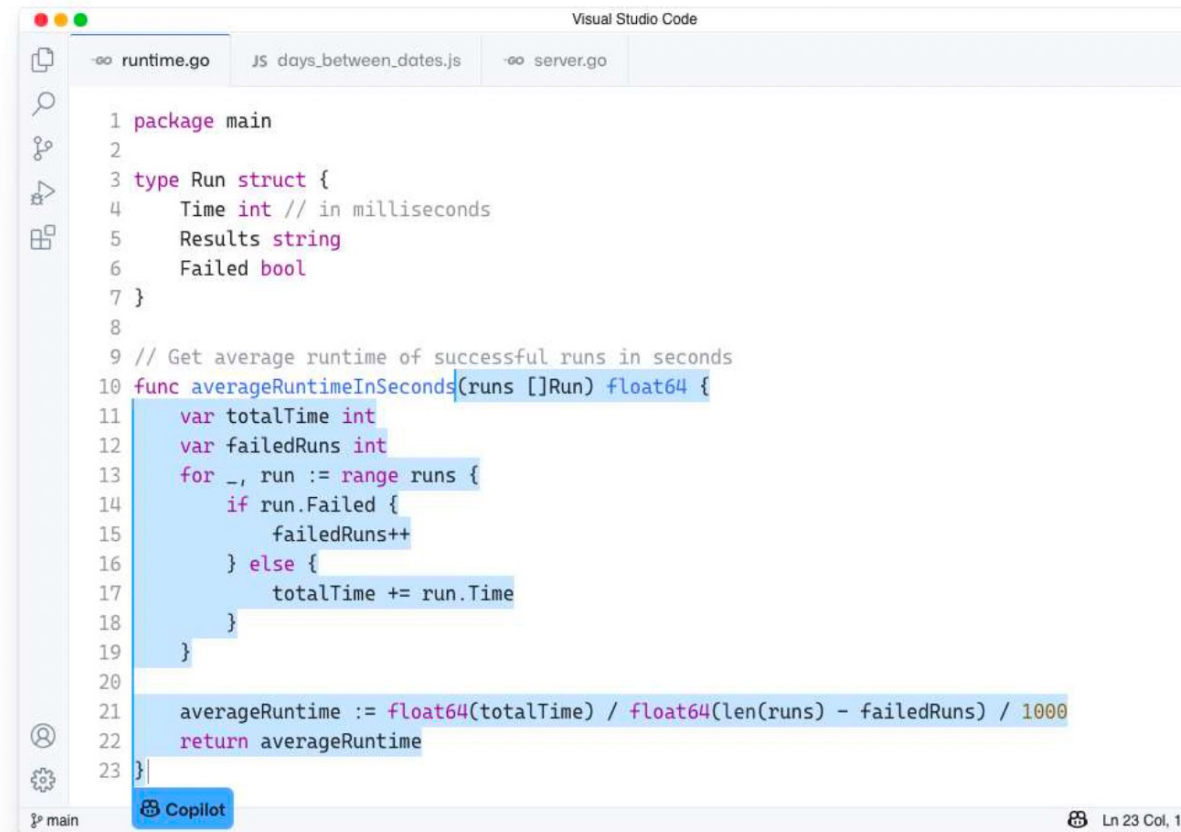
# What is Copilot?

GitHub Copilot is an AI-powered code completion tool developed by GitHub in collaboration with OpenAI. Trained on billions of lines of public code, GitHub Copilot puts the knowledge you need at your fingertips, saving you time and helping you stay focused.

- Extends your editor:  
GitHub Copilot is available today as a Visual Studio Code extension.
- Speaks all the languages you love:  
GitHub Copilot works with a broad set of frameworks and languages.
- You're the pilot:  
With GitHub Copilot, you're always in charge.

# More than autocomplete

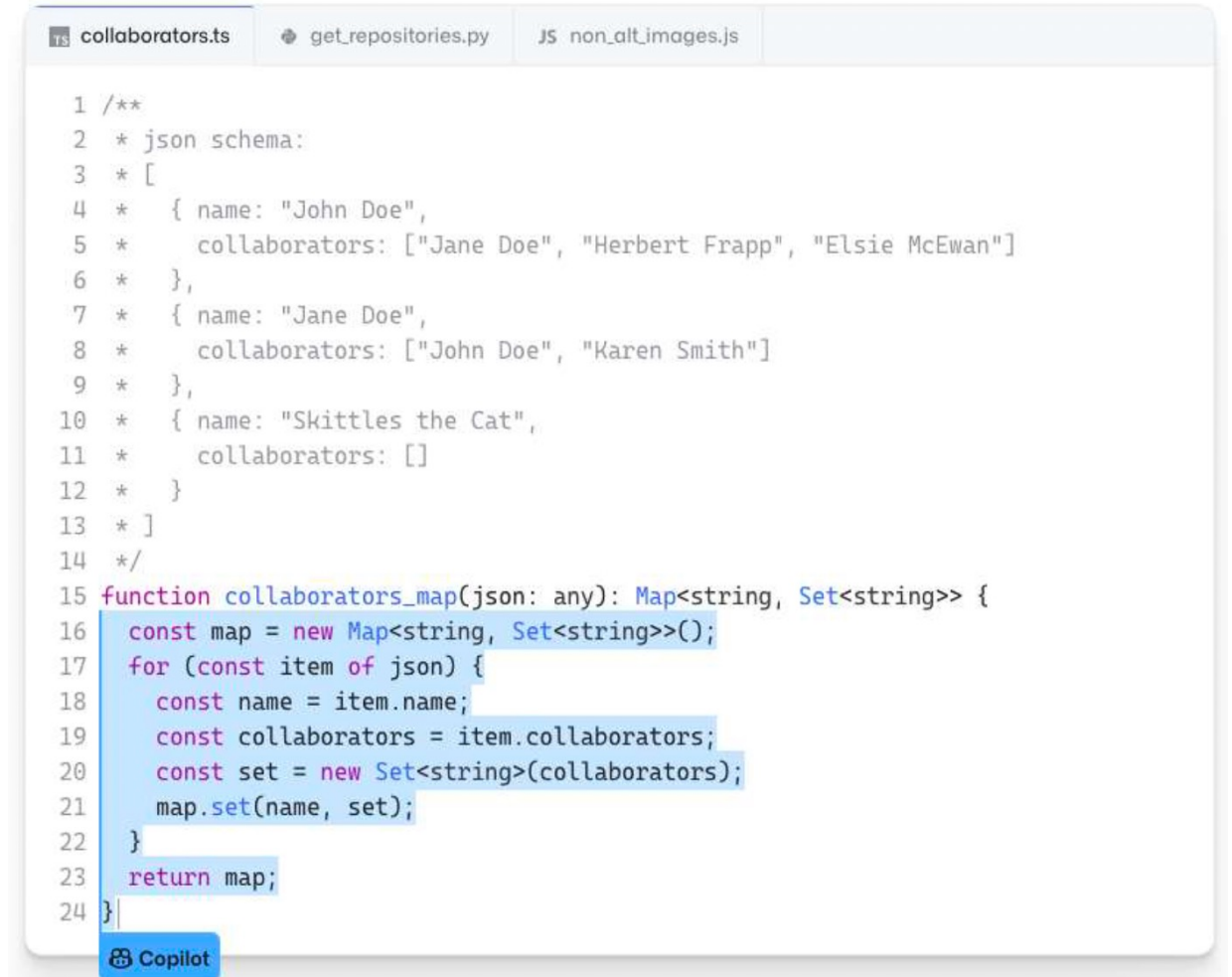
GitHub Copilot also is powered by Codex, the AI system created by OpenAI. GitHub Copilot understands significantly more context than most code assistants. So, whether it's in a docstring, comment, function name, or the code itself, GitHub Copilot uses the context you've provided and synthesizes code to match. Together with OpenAI, we're designing GitHub Copilot to get smarter at producing safe and effective code as developers use it.



```
1 package main
2
3 type Run struct {
4     Time int // in milliseconds
5     Results string
6     Failed bool
7 }
8
9 // Get average runtime of successful runs in seconds
10 func averageRuntimeInSeconds(runs []Run) float64 {
11     var totalTime int
12     var failedRuns int
13     for _, run := range runs {
14         if run.Failed {
15             failedRuns++
16         } else {
17             totalTime += run.Time
18         }
19     }
20
21     averageRuntime := float64(totalTime) / float64(len(runs) - failedRuns) / 1000
22     return averageRuntime
23 }
```

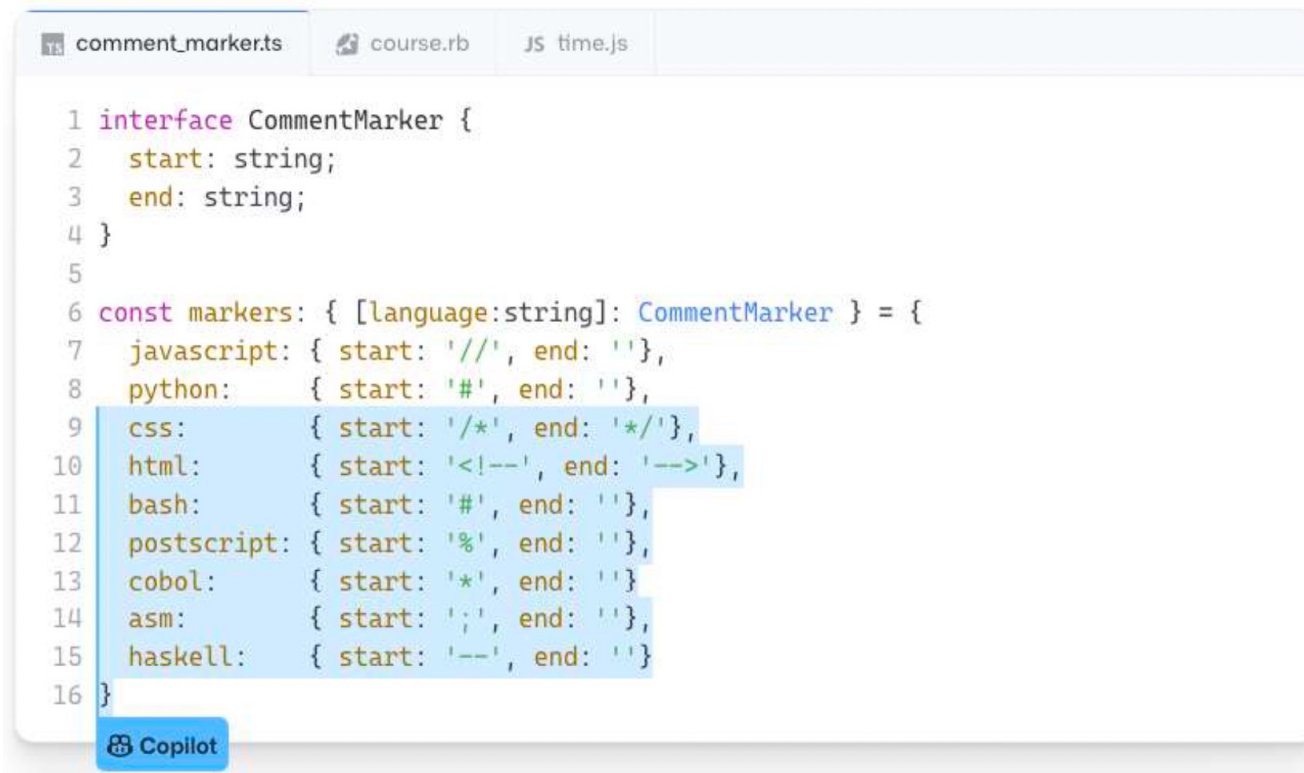
# More than autocomplete

**Convert comments to code.** Write a comment describing the logic you want, and let GitHub Copilot assemble the code for you.



```
1 /**
2  * json schema:
3  * [
4  *   { name: "John Doe",
5  *     collaborators: ["Jane Doe", "Herbert Frapp", "Elsie McEwan"]
6  * },
7  *   { name: "Jane Doe",
8  *     collaborators: ["John Doe", "Karen Smith"]
9  * },
10 *   { name: "Skittles the Cat",
11 *     collaborators: []
12 * }
13 * ]
14 */
15 function collaborators_map(json: any): Map<string, Set<string>> {
16   const map = new Map<string, Set<string>>();
17   for (const item of json) {
18     const name = item.name;
19     const collaborators = item.collaborators;
20     const set = new Set<string>(collaborators);
21     map.set(name, set);
22   }
23   return map;
24 }
```

# More than autocomplete



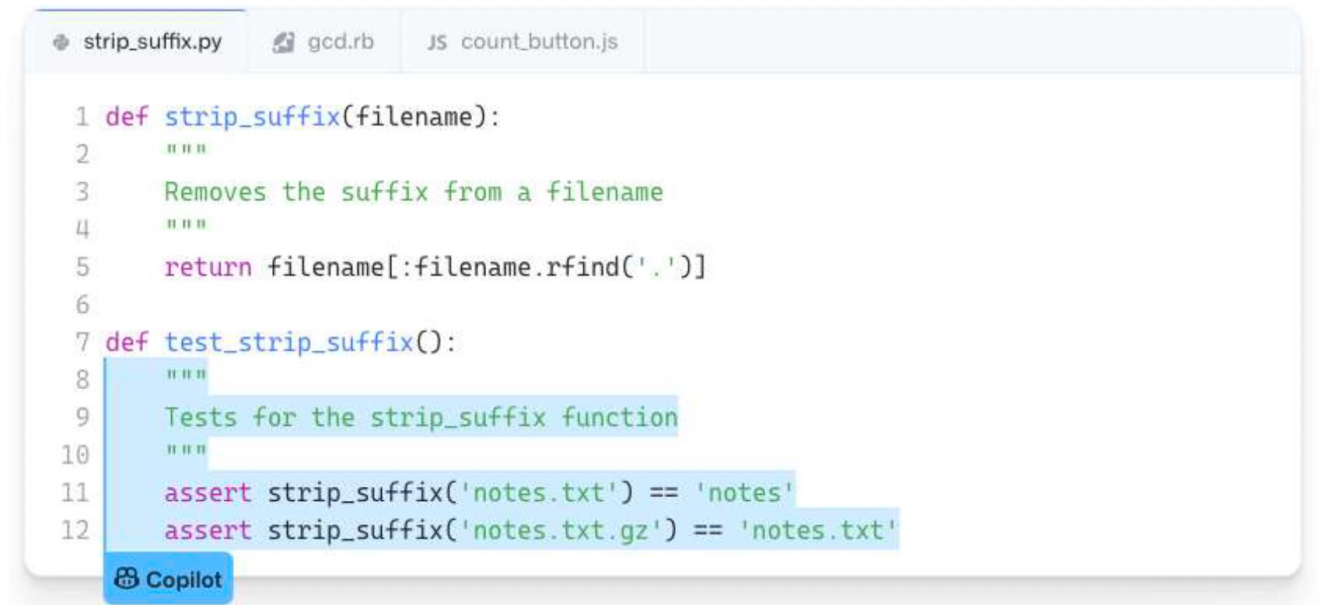
```
comment_marker.ts  course.rb  JS time.js

1 interface CommentMarker {
2   start: string;
3   end: string;
4 }
5
6 const markers: { [language:string]: CommentMarker } = {
7   javascript: { start: '//', end: ''},
8   python:     { start: '#', end: ''},
9   css:        { start: '/*', end: '*/'},
10  html:        { start: '<!--', end: '-->'},
11  bash:        { start: '#', end: ''},
12  postscript: { start: '%', end: ''},
13  cobol:       { start: '*', end: ''},
14  asm:         { start: ';', end: ''},
15  haskell:     { start: '--', end: ''}
16 }
```

**Autofill for repetitive code.** GitHub Copilot works great for quickly producing boilerplate and repetitive code patterns. Feed it a few examples and let it generate the rest!

# More than autocomplete

**Tests without the toil.** Tests are the backbone of any robust software engineering project. Import a unit test package, and let GitHub Copilot suggest tests that match your implementation code.



The screenshot shows a code editor with three tabs: `strip_suffix.py`, `gcd.rb`, and `JS count_button.js`. The active tab is `strip_suffix.py`, which contains the following Python code:

```
1 def strip_suffix(filename):
2     """
3     Removes the suffix from a filename
4     """
5     return filename[:filename.rfind('.')]
6
7 def test_strip_suffix():
8     """
9     Tests for the strip_suffix function
10    """
11    assert strip_suffix('notes.txt') == 'notes'
12    assert strip_suffix('notes.txt.gz') == 'notes.txt'
```

Lines 8 through 12 are highlighted in blue, indicating they were suggested by GitHub Copilot. A Copilot icon is visible in the bottom left corner of the editor window.



# More than autocomplete

**Show me alternatives.** Want to evaluate a few different approaches? GitHub Copilot can show you a list of solutions. Use the code as provided, or edit it to meet your needs.



The screenshot shows the Visual Studio Code editor with a Python file named `main`. The code defines a function `max_sum_slice(xs)` that calculates the maximum sum of a subarray. A Copilot suggestion menu is visible above the code, showing options to navigate between suggestions (Next, Previous) and to accept the current suggestion (Accept, Tab). The code is as follows:

```
1 def max_sum_slice(xs):  
2     max_ending = max_so_far = 0  
3     for x in xs:  
4         max_ending = max(0, max_ending + x)  
5         max_so_far = max(max_so_far, max_ending)  
6     return max_so_far
```

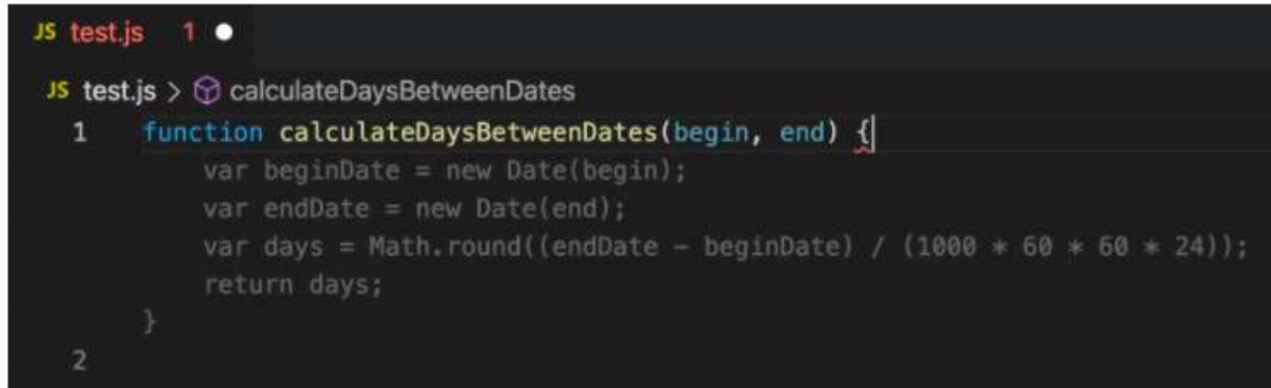
The status bar at the bottom indicates the current position is Line 6, Column 21.

# Creating

1. Create a new JavaScript (.js) file.
2. Type the following function header:

```
function calculateDaysBetweenDates(begin, end) {
```

3. GitHub Copilot will automatically suggest an entire function body in grayed text, as shown below. The exact suggestion may vary.



The screenshot shows a code editor with a dark background. At the top, it says 'JS test.js 1'. Below that, it shows 'JS test.js > calculateDaysBetweenDates'. The function header 'function calculateDaysBetweenDates(begin, end) {' is highlighted in blue. The function body is suggested in grayed text: 'var beginDate = new Date(begin);', 'var endDate = new Date(end);', 'var days = Math.round((endDate - beginDate) / (1000 \* 60 \* 60 \* 24));', and 'return days;'. The closing brace '}' is also highlighted in blue. The line number '2' is visible at the bottom left of the code block.

```
JS test.js 1 •
JS test.js > calculateDaysBetweenDates
1  function calculateDaysBetweenDates(begin, end) {
    var beginDate = new Date(begin);
    var endDate = new Date(end);
    var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));
    return days;
  }
2
```

4. Press `Tab` to accept the suggestion.

# Choosing alternative suggestions

For any given input, GitHub Copilot can provide multiple suggestions. As the developer you are always in charge; you can select which suggestion to use, or reject them all.

1. Clear the file (or start a new one), and type the following again:

```
function calculateDaysBetweenDates(begin, end) {
```

2. GitHub Copilot will again show you a suggested completion.

3. Instead of pressing `Tab` :

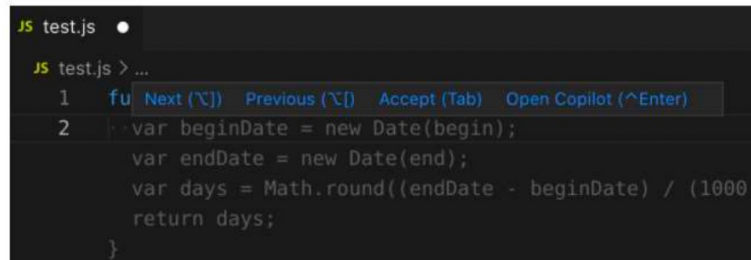
- On macOS, press `Option + ]` (or `Option + [`).
- On Windows or Linux, press `Alt + ]` (or `Alt + [`).

GitHub Copilot will cycle through other alternative suggestions.

4. When you see a suggestion you like, press `Tab` to accept it.

5. If you don't like any of the suggestions, press `Esc` .

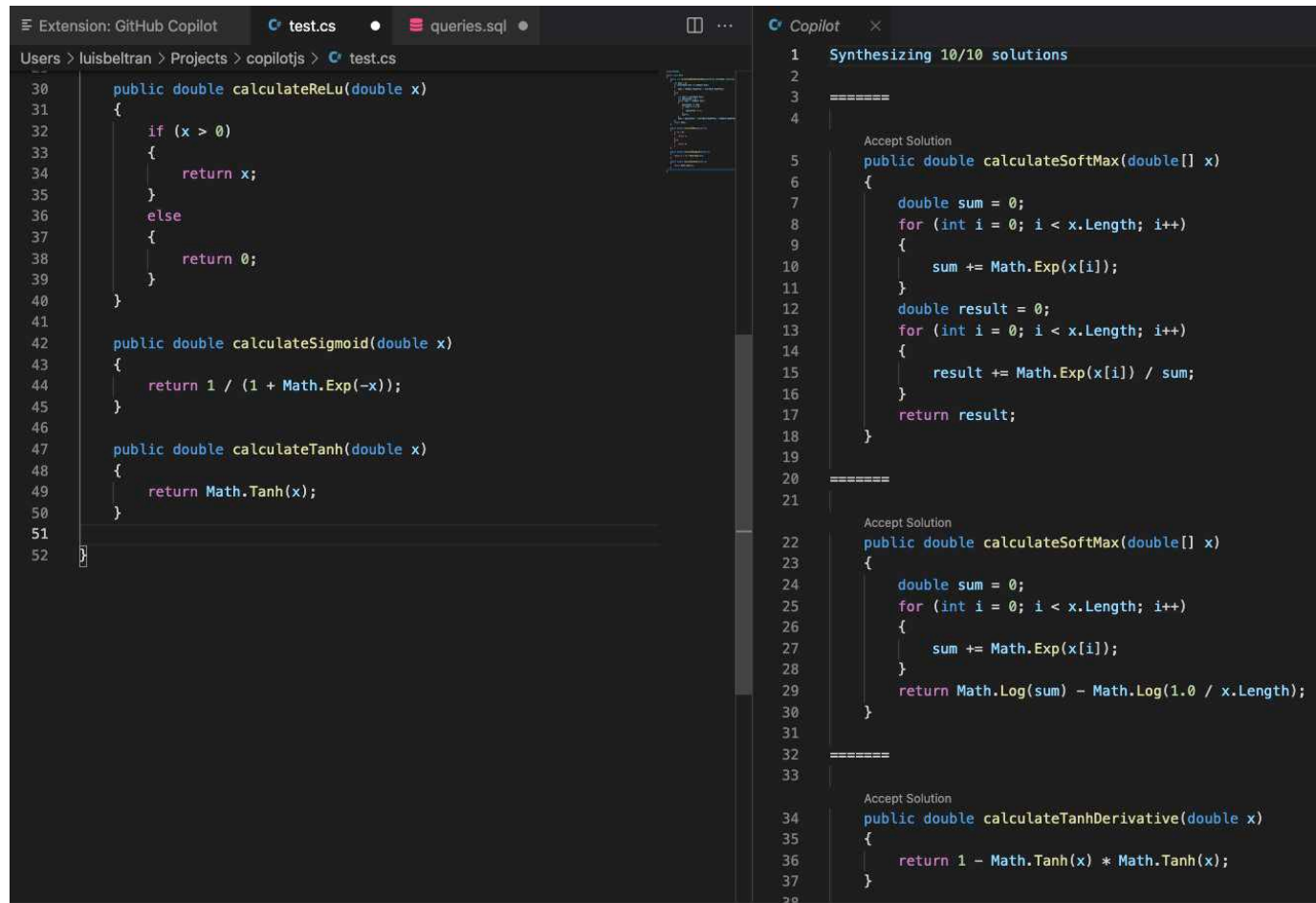
You can also hover over a suggestion to see the GitHub Copilot command palette for choosing suggestions.



The screenshot shows a code editor with a file named 'test.js'. The code contains a function 'calculateDaysBetweenDates' that takes 'begin' and 'end' dates and returns the number of days between them. A GitHub Copilot suggestion is shown, cycling through different completions for the function body. The suggestion palette is visible, showing options like 'Next', 'Previous', 'Accept', and 'Open Copilot'.

```
JS test.js > ...  
1 fu Next (^) Previous (^) Accept (Tab) Open Copilot (^Enter)  
2 var beginDate = new Date(begin);  
  var endDate = new Date(end);  
  var days = Math.round((endDate - beginDate) / (1000 * 60 * 60 * 24));  
  return days;  
}
```

# Choosing alternative suggestions



The screenshot shows the Visual Studio Code editor with a C# file named `test.cs` open. The file contains three methods: `calculateReLU`, `calculateSigmoid`, and `calculateTanh`. The GitHub Copilot sidebar is open on the right, displaying "Synthesizing 10/10 solutions". It shows three alternative suggestions for the `calculateSoftMax` method, each preceded by "Accept Solution".

```
30 public double calculateReLU(double x)
31 {
32     if (x > 0)
33     {
34         return x;
35     }
36     else
37     {
38         return 0;
39     }
40 }
41
42 public double calculateSigmoid(double x)
43 {
44     return 1 / (1 + Math.Exp(-x));
45 }
46
47 public double calculateTanh(double x)
48 {
49     return Math.Tanh(x);
50 }
51
52
```

**Synthesizing 10/10 solutions**

Accept Solution

```
5 public double calculateSoftMax(double[] x)
6 {
7     double sum = 0;
8     for (int i = 0; i < x.Length; i++)
9     {
10         sum += Math.Exp(x[i]);
11     }
12     double result = 0;
13     for (int i = 0; i < x.Length; i++)
14     {
15         result += Math.Exp(x[i]) / sum;
16     }
17     return result;
18 }
19
```

Accept Solution

```
22 public double calculateSoftMax(double[] x)
23 {
24     double sum = 0;
25     for (int i = 0; i < x.Length; i++)
26     {
27         sum += Math.Exp(x[i]);
28     }
29     return Math.Log(sum) - Math.Log(1.0 / x.Length);
30 }
31
```

Accept Solution

```
34 public double calculateTanhDerivative(double x)
35 {
36     return 1 - Math.Tanh(x) * Math.Tanh(x);
37 }
38
```

# Generating code from a comment

GitHub Copilot can understand significantly more context than most code assistants, and can generate entire functions from something as simple as a comment.

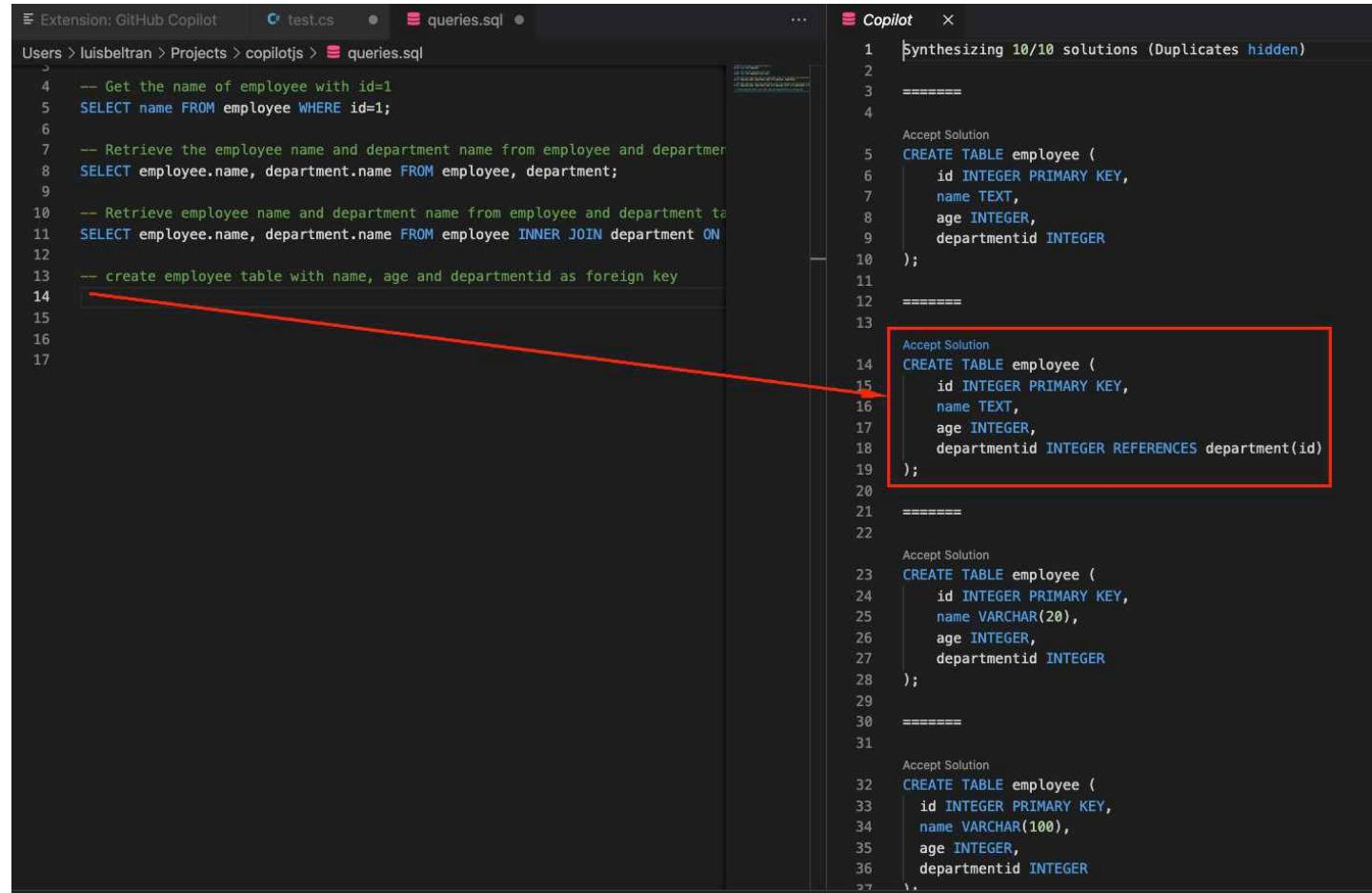
1. Create a new JavaScript file, and type the following:

```
// find all images without alternate text
// and give them a red border
function process() {
```

2. GitHub Copilot will automatically suggest an implementation:

```
JS test.js 1 ●
JS test.js > process
1 // find all images without alternate text
2 // and give them a red border
3 function process() {
    var images = document.getElementsByTagName('img');
    for (var i = 0; i < images.length; i++) {
        if (!images[i].alt) {
            images[i].style.border = '1px solid red';
        }
    }
}
```

# Generating code from a comment



The screenshot shows the GitHub Copilot extension in VS Code. On the left, the `queries.sql` file contains several SQL queries and a comment: `-- create employee table with name, age and departmentid as foreign key`. On the right, the Copilot pane shows three generated solutions. The second solution, which is highlighted with a red box, is the one generated from the comment. A red line connects the comment in the left pane to this specific solution in the right pane.

```
Users > luisbeltran > Projects > copilotjs > queries.sql
4  -- Get the name of employee with id=1
5  SELECT name FROM employee WHERE id=1;
6
7  -- Retrieve the employee name and department name from employee and department
8  SELECT employee.name, department.name FROM employee, department;
9
10 -- Retrieve employee name and department name from employee and department table
11 SELECT employee.name, department.name FROM employee INNER JOIN department ON
12
13 -- create employee table with name, age and departmentid as foreign key
14
15
16
17
```

Copilot

1 Synthesizing 10/10 solutions (Duplicates hidden)

2

3 =====

4

Accept Solution

5 CREATE TABLE employee (

6 id INTEGER PRIMARY KEY,

7 name TEXT,

8 age INTEGER,

9 departmentid INTEGER

10 );

11

12 =====

13

Accept Solution

14 CREATE TABLE employee (

15 id INTEGER PRIMARY KEY,

16 name TEXT,

17 age INTEGER,

18 departmentid INTEGER REFERENCES department(id)

19 );

20

21 =====

22

Accept Solution

23 CREATE TABLE employee (

24 id INTEGER PRIMARY KEY,

25 name VARCHAR(20),

26 age INTEGER,

27 departmentid INTEGER

28 );

29

30 =====

31

Accept Solution

32 CREATE TABLE employee (

33 id INTEGER PRIMARY KEY,

34 name VARCHAR(100),

35 age INTEGER,

36 departmentid INTEGER

37 );

# Using a framework

GitHub Copilot is especially useful for working with APIs and frameworks you're unfamiliar with. Here, we'll use GitHub Copilot to create a simple Express server that returns the current time.

1. Create a new JavaScript file, and type the following comment and press `Enter`.

```
// Express server on port 3000
```



2. GitHub Copilot will generate lines of code to create the Express app. Press `Tab` and `Enter` to accept each line.
3. Type the following comment and press `Enter`.

```
// Return the current time
```

4. GitHub Copilot will generate code for the default handler. Press `Tab` to accept each line.

# Questions

## **How do I get the most out of GitHub Copilot?**

It works best when you divide your code into small functions, use meaningful names for functions parameters, and write good docstrings and comments as you go. It also seems to do best when it's helping you navigate unfamiliar libraries or frameworks.



# Questions

## **What data has GitHub Copilot been trained on?**

It has been trained on a selection of English language and source code from publicly available sources, including code in public repositories on GitHub.

## **Why was GitHub Copilot trained on data from publicly available sources?**

Training machine learning models on publicly available data is considered fair use across the machine learning community. The models gain insight and accuracy from the public collective intelligence.

## **Will GitHub Copilot help me write code for a new platform?**

When a new platform or API is launched for the first time, developers are the least familiar with it. There is also very little public code available that uses that API, and a machine learning model is unlikely to generate the code without fine tuning. In the future, OpenAI will provide ways to highlight newer APIs and samples to raise their relevance in GitHub Copilot's suggestions.

# Questions

## **Does GitHub Copilot recite code from the training set?**

GitHub Copilot is a code synthesizer, not a search engine: the vast majority of the code that it suggests is uniquely generated and has never been seen before. OpenAI found that about 0.1% of the time, the suggestion may contain some snippets that are verbatim from the training set. Many of these cases happen when you don't provide sufficient context (in particular, when editing an empty file), or when there is a common, perhaps even universal, solution to the problem. OpenAI are building an origin tracker to help detect the rare instances of code that is repeated from the training set, to help you make good real- time decisions about GitHub Copilot's suggestions.

# Questions

## **Who owns the code GitHub Copilot helps me write?**

GitHub Copilot is a tool, like a compiler or a pen. The suggestions GitHub Copilot generates, and the code you write with its help, belong to you, and you are responsible for it. OpenAI recommends that you carefully test, review, and vet the code, as you would with any code you write yourself. The code you create with GitHub Copilot's help belongs to you. While every friendly robot likes the occasional word of thanks, you are in no way obligated to credit GitHub Copilot. Just like with a compiler, the output of your use of GitHub Copilot belongs to you.

# Questions

## **Does GitHub Copilot ever output personal data?**

Because GitHub Copilot was trained on publicly available code, its training set included public personal data included in that code. From our internal testing, we found it to be extremely rare that GitHub Copilot suggestions included personal data verbatim from the training set. In some cases, the model will suggest what appears to be personal data - email addresses, phone numbers, access keys, etc. - but is actually made-up information synthesized from patterns in training data. For the technical preview, OpenAI implemented a rudimentary filter that blocks emails when shown in standard formats, but it's still possible to get the model to suggest this sort of content if you try hard enough.

# Questions

## What data is collected?

The GitHub Copilot collects activity from the user's Visual Studio Code editor, tied to a timestamp, and metadata. This metadata consists of the extension settings and the standard metadata collected by the Visual Studio Code extension telemetry package:

- Visual Studio Code machine ID (pseudonymized identifier)
- Visual Studio Code session ID (pseudonymized identifier)
- Visual Studio Code version
- Geolocation from IP address (country, state/province and city, but not the IP address itself)
- Operating system and version
- Extension version
- The VS Code UI (web or desktop)

# Questions

**The activity collected consists of events that are triggered when:**

- An error occurs (it records the error kind and relevant background; e.g. if it's an authentication error the key expiry date is recorded)
- OpenAI models are accessed to ask for code suggestions (it records editor state like position of cursor and snippets of code)—this includes cases when the user takes an action to request code suggestions
- Code suggestions are received or displayed (it records the suggestions, post-processing, and metadata like model certainty and latency)
- Code suggestions are redacted due to filters that ensure AI safety
- The user acts on code suggestions (e.g. to accept or reject them)
- The user has acted on code suggestions and then it records whether or how they persisted in the code

# Questions

## **Can GitHub Copilot introduce insecure code or offensive outputs in its suggestions?**

There's a lot of public code in the world with insecure coding patterns, bugs, or references to outdated APIs or idioms. When GitHub Copilot synthesizes code suggestions based on this data, it can also synthesize code that contains these undesirable patterns. This is something OpenAI and Microsoft care a lot about at GitHub, and in recent years they provided tools such as Actions, Dependabot, and CodeQL to open source projects to help improve code quality. Similarly, as GitHub Copilot improves, they work to exclude insecure or low-quality code from the training set. Of course, you should always use GitHub Copilot together with testing practices and security tools, as well as your own judgment.

The technical preview includes filters to block offensive words and avoid synthesizing suggestions in sensitive contexts. Due to the pre-release nature of the underlying technology, GitHub Copilot may sometimes produce undesired outputs, including biased, discriminatory, abusive, or offensive outputs. If you see offensive outputs, please report them so that we can improve our safeguards. GitHub takes this challenge very seriously and they are committed to addressing it with GitHub Copilot.

# Questions

## **How will advanced code generation tools like GitHub Copilot affect developer jobs?**

Bringing in more intelligent systems has the potential to bring enormous change to the developer experience. OpenAI expects this technology will enable existing engineers to be more productive, reducing manual tasks and helping them focus on interesting work. They also believe that GitHub Copilot has the potential to lower barriers to entry, enabling more people to explore software development and join the next generation of developers.



# Questions

## **How is the data that GitHub Copilot collects used?**

In order to generate suggestions, GitHub Copilot transmits part of the file you are editing to the service. This context is used to synthesize suggestions for you. GitHub Copilot also records whether the suggestions are accepted or rejected. This telemetry is used to improve future versions of the AI system, so that GitHub Copilot can make better suggestions for all users in the future.

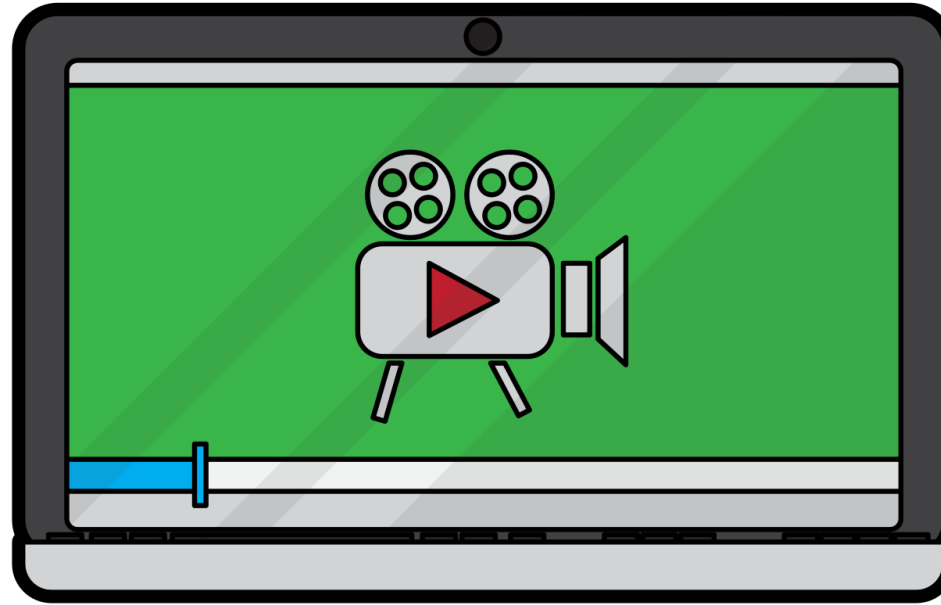
OpenAI do not reference your private code when generating code for other users.

All data is transmitted and stored securely. Access to the telemetry is strictly limited to individuals on a need-to-know basis. Inspection of the gathered source code will be predominantly automatic, and when humans read it, it is specifically with the aim of improving the model or detecting abuse.

# Conclusion

Here are a few suggestions for how you can continue to improve your skills with Copilot:

- 1. Practice, practice, practice!**
2. Experiment with different programming languages
3. Collaborate with others
4. Learn from its suggestions
5. Keep up-to-date with Copilot updates



## Videos:

What is GitHub Copilot? (2) – <https://www.youtube.com/watch?v=lqXNhakuwVc>

Get Started with the Future of Coding: GitHub Copilot (14) - <https://www.youtube.com/watch?v=Fi3AJZZregl>

GitHub Copilot X Explained | A big step forward... (18) - [https://www.youtube.com/watch?v=8\\_0DJ9FOlOM&t=1s](https://www.youtube.com/watch?v=8_0DJ9FOlOM&t=1s)

CoPilot Review: My Thoughts After 6 Months (10) - <https://www.youtube.com/watch?v=RDd71UIlgpg>

SciSpace Copilot - an AI tool for research papers (9) - <https://www.youtube.com/watch?v=MJp0WH4-gw0>

Effortless Python with GitHub Copilot (12) - <https://www.youtube.com/watch?v=DSHfHT5qnGc>

GitHub Copilot tips and tricks (4) - <https://www.youtube.com/watch?v=1qs6QKk0DVc>

GitHub Copilot X tested with REAL scenarios (30)- <https://www.youtube.com/watch?v=s7AGkcSMial>