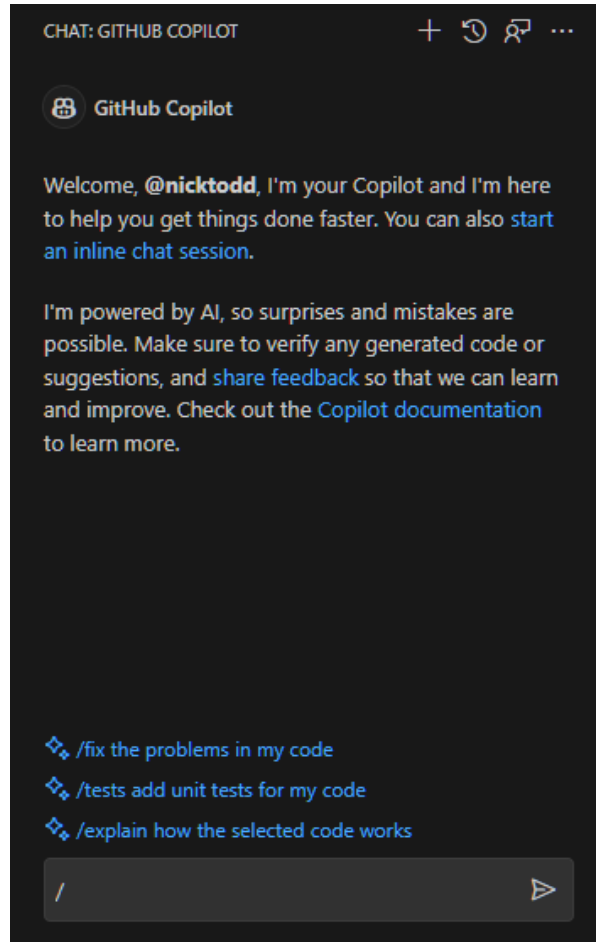
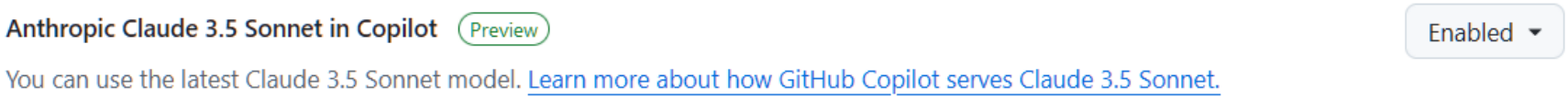
Developer Tools

GitHub CoPilot

- This was the first developer tool to gain significant traction
  - https://github.com/features/copilot
- Monthly or Annual Fee to use
- Currently uses GPT4 LLM by default
  - Other models are available

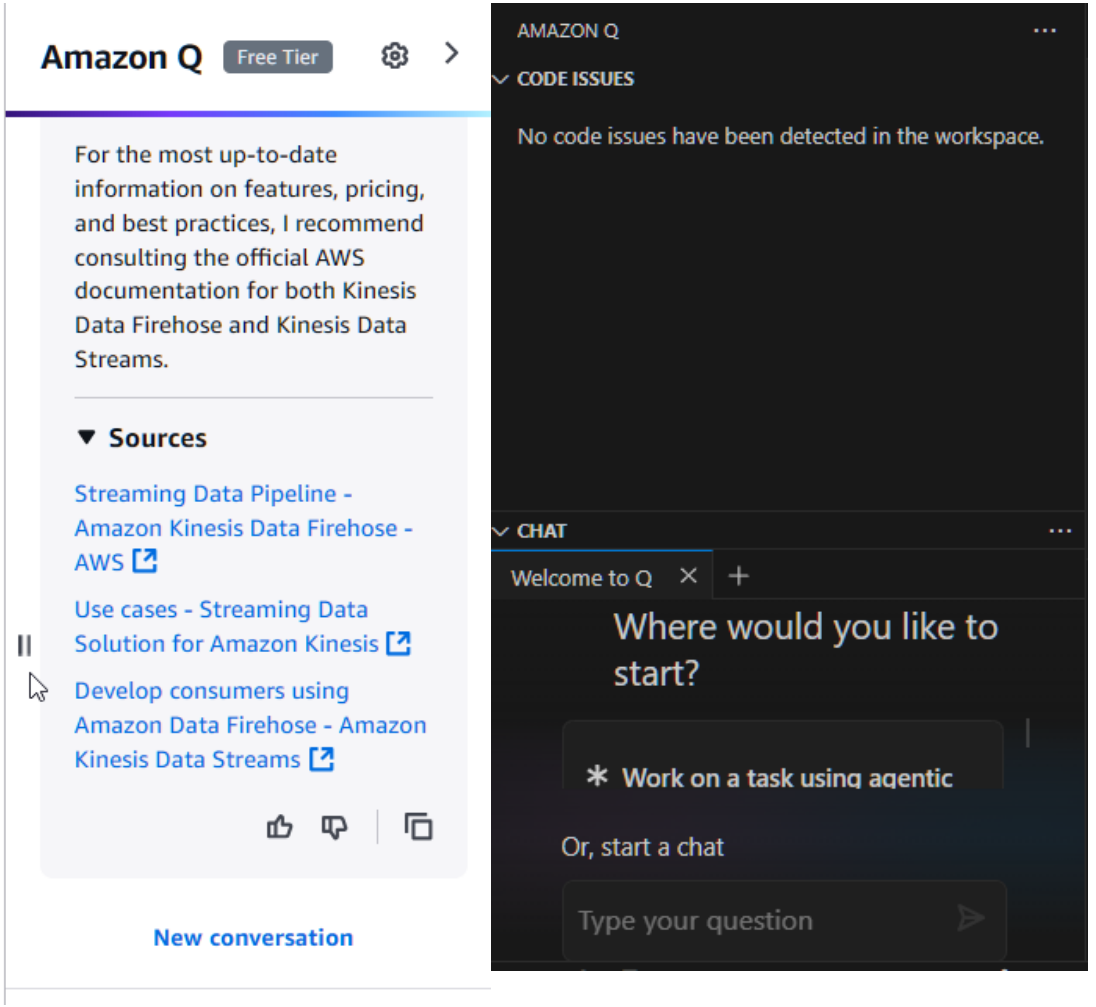




- Can be used in multiple IDEs

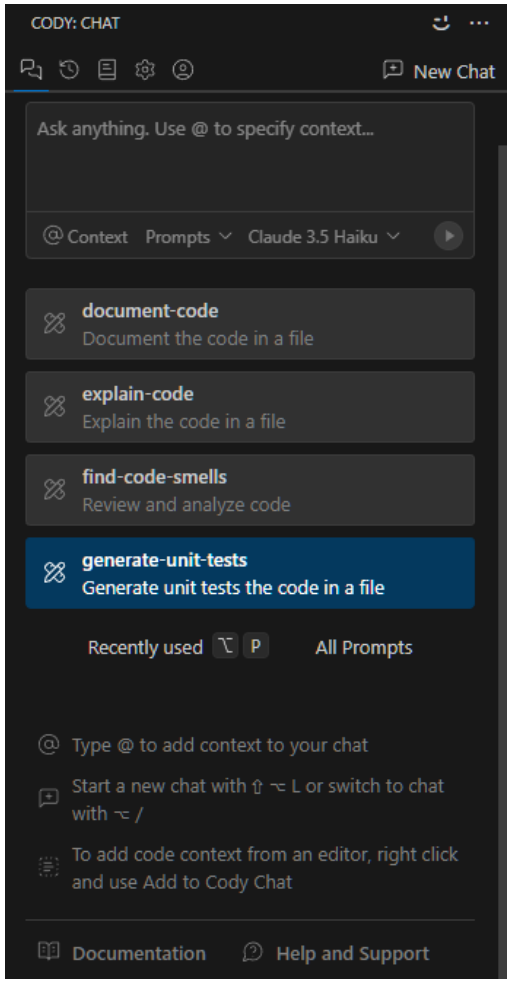
Amazon Q Developer

- The Amazon Q Developer tool provides Gen AI capabilities
  - https://aws.amazon.com/q/developer/
- Amazon Q Developer operates
  - In the IDE (similar to the other tools)
  - In the AWS Management Console
- You will require an AWS account in order to use it
- Multiple IDEs are supported



SourceGraph Cody

- Cody is the tool from SourceGraph
  - https://sourcegraph.com/cody
- Cody is different from the other offerings in that it allows the user to select an LLM for each session
- Has a free tier and then paid options

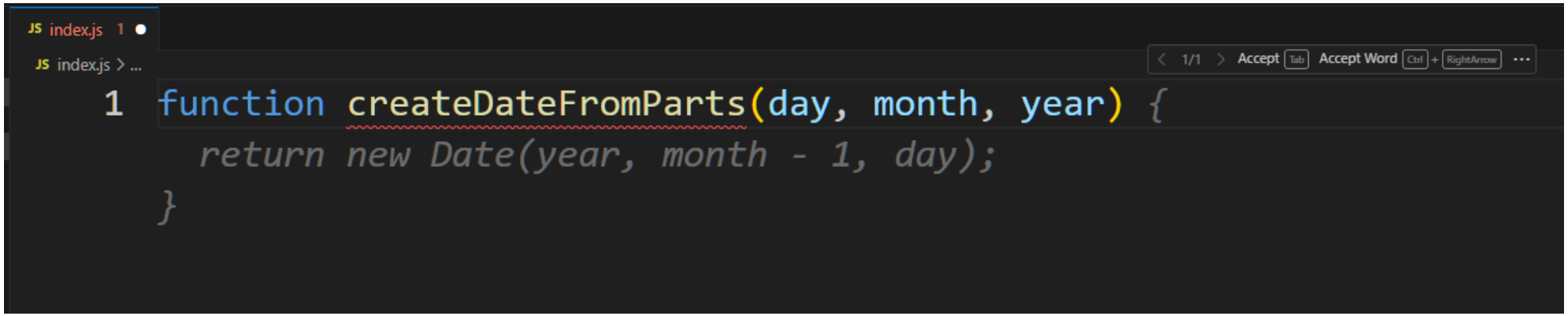


GitHub Copilot Features

- GitHub CoPilot has a growing set of features including
  - Autocompletion
  - Requesting suggestions
  - Context Menus
    - Generating documentation
    - Getting explanations
    - Fixing errors
    - Generating tests
  - Chat
  - Project creation

AutoCompletion

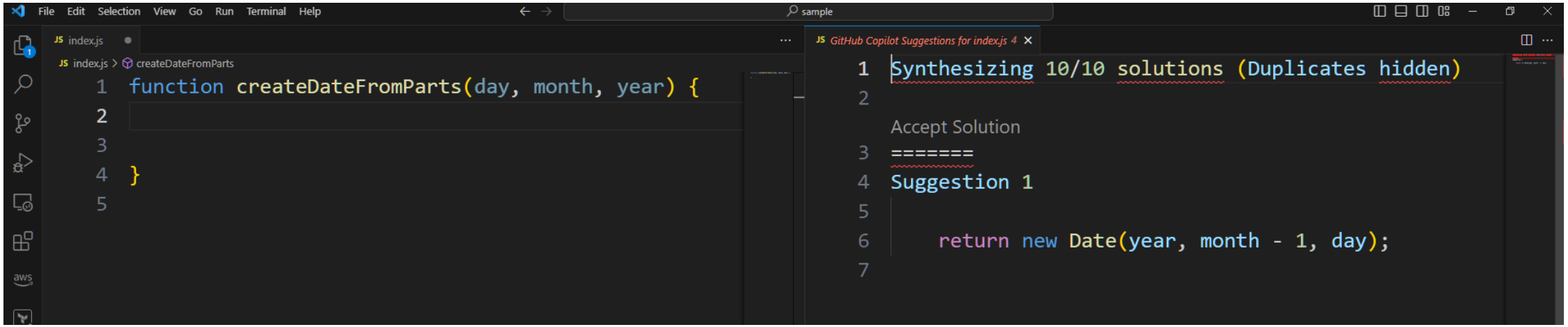
- One of the most useful features of CoPilot is the autocompletion capability
  - You begin typing some code and it suggests completions



- You can then **Accept** the suggestion using **Tab**

Request Suggestions

- Using **Ctrl-Enter**, you can request suggestions from GitHub CoPilot



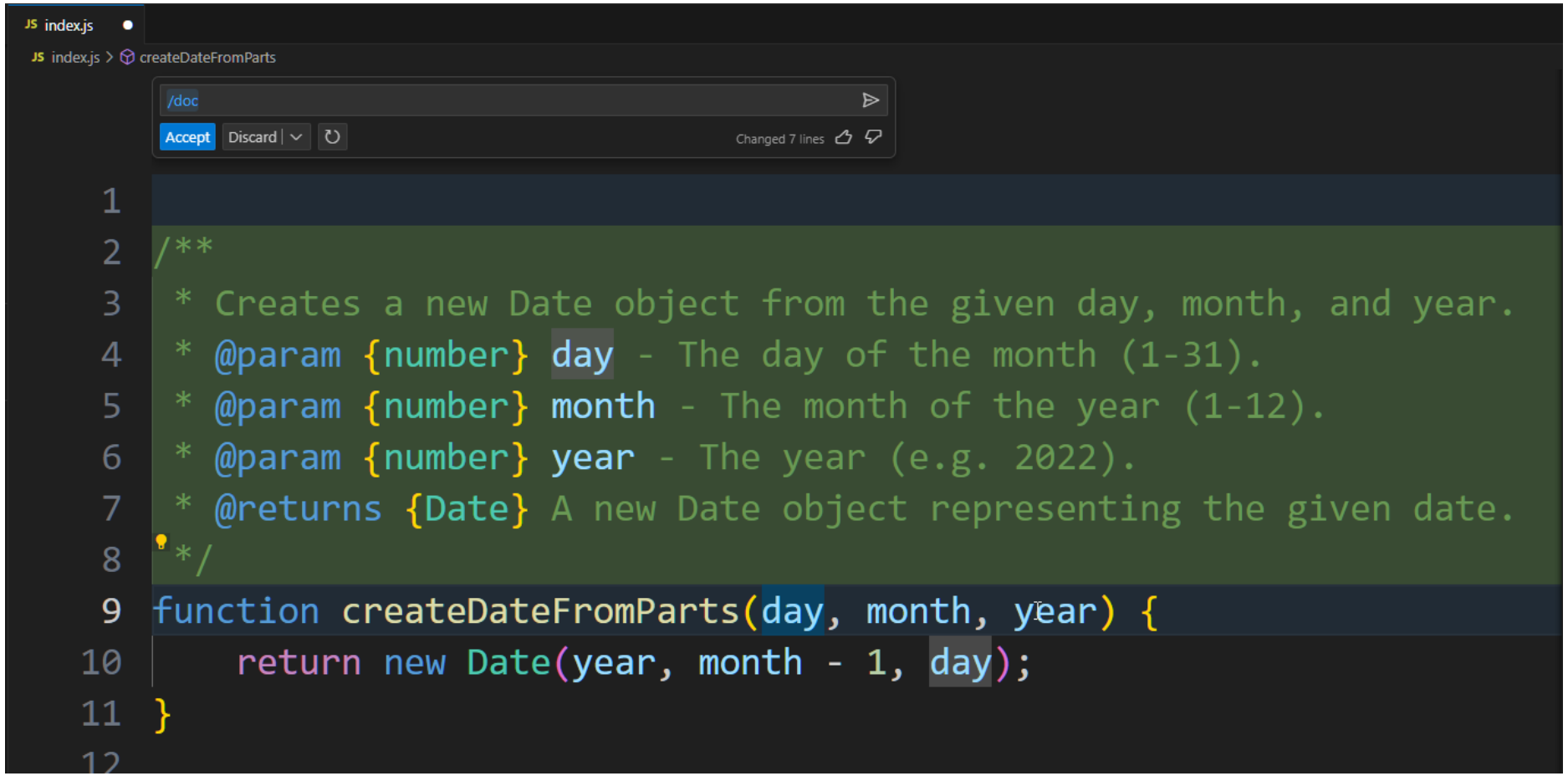
- Multiple suggestions are then displayed and you can select the preferred suggestion

Context Menu

- You can bring up the GitHub CoPilot context menu using **Ctrl-I**
- From this menu you can then enter one of the following commands
  - **/doc**
  - **/explain**
  - **/fix**
  - **/test**

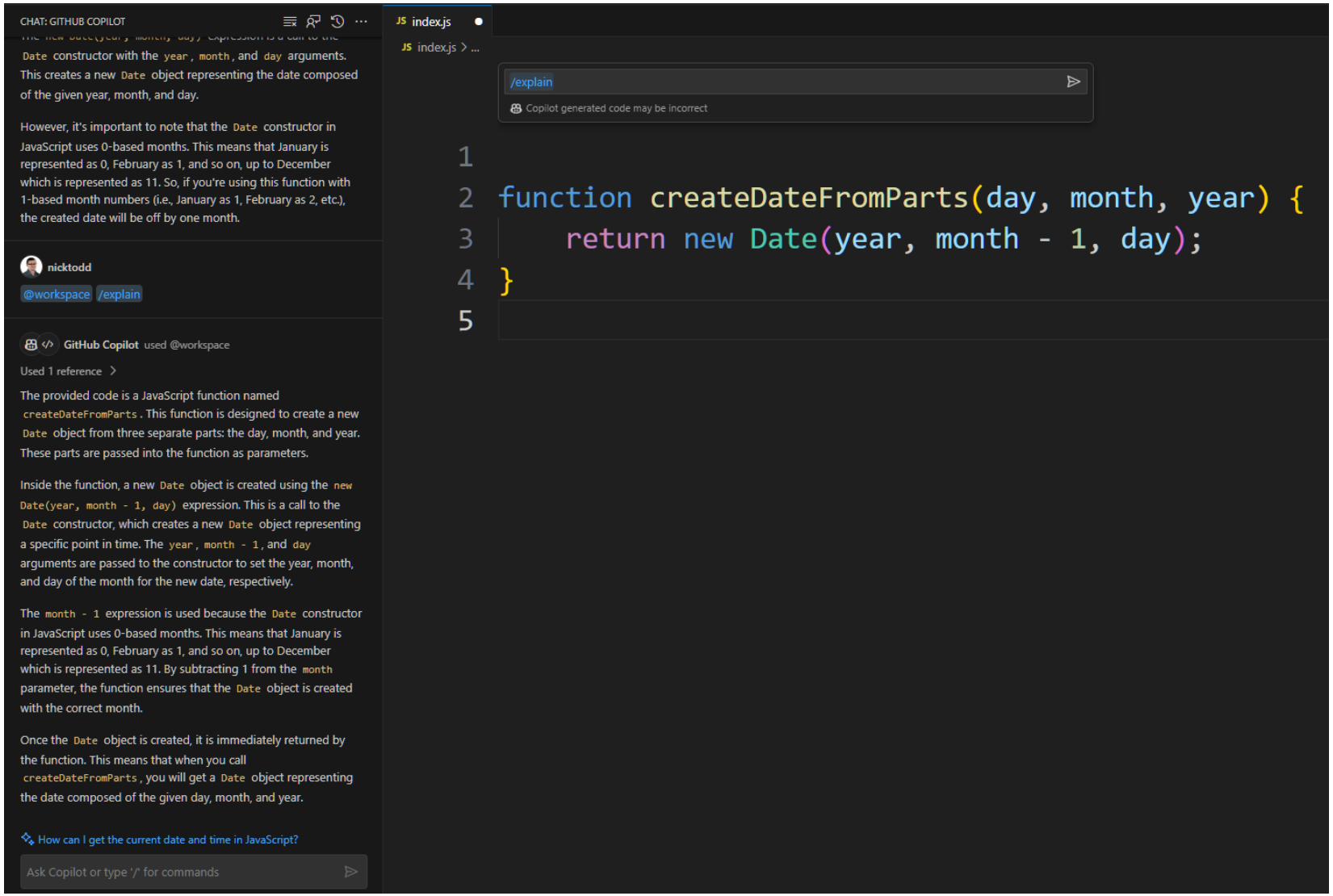
Using /doc to Generate Documentation

- Using **/doc** you can generate documentation for your code



Using /explain to Understand

- Using **/explain** describes the following code in the GitHub CoPilot Chat window



Fixing Problems

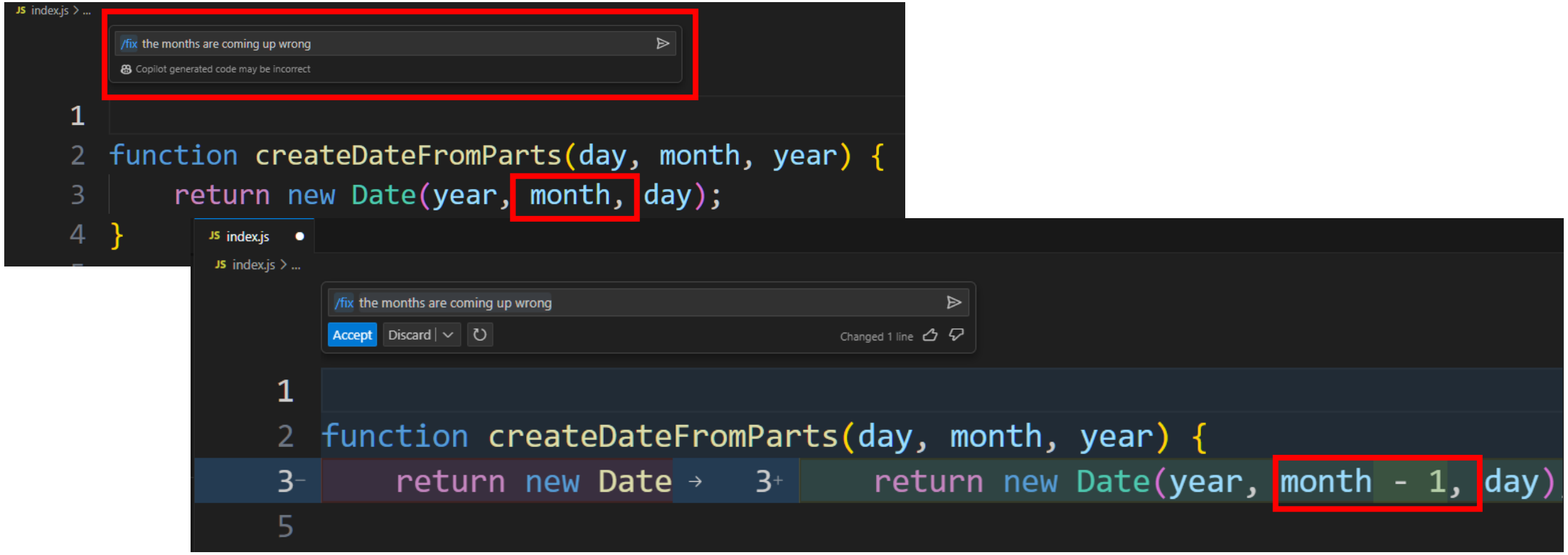
- Let’s introduce a deliberate problem with our function

```
function createDateFromParts(day, month, year)
{
    return new Date(year, month -1, day);
}
```

```
function createDateFromParts(day, month, year)
{
    return new Date(year, month, day);
}
```

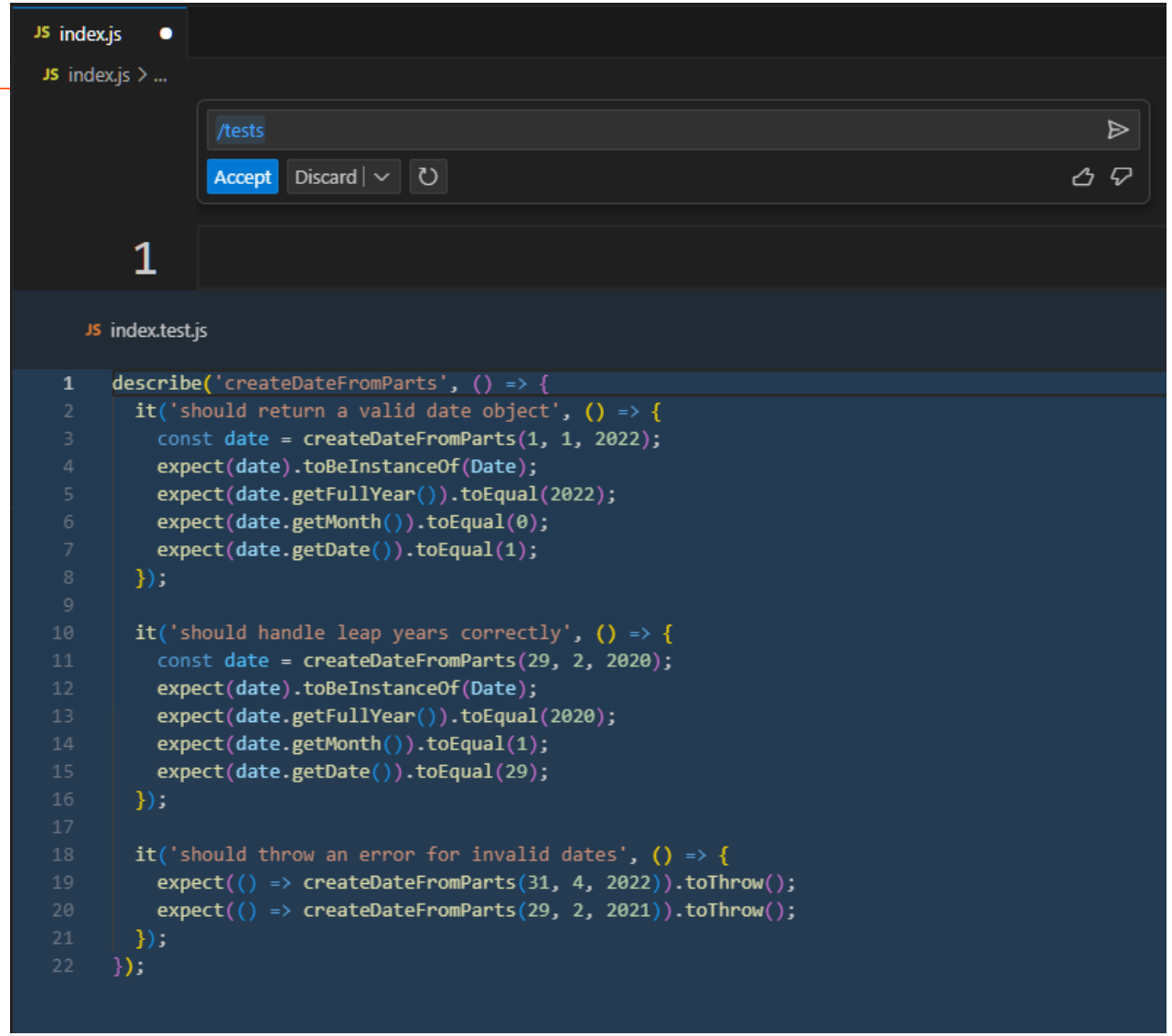
Using CoPilot to Fix the Issue

- The code now has the error
- I can ask GitHub CoPilot to fix the error with **/fix** followed by a description of the problem



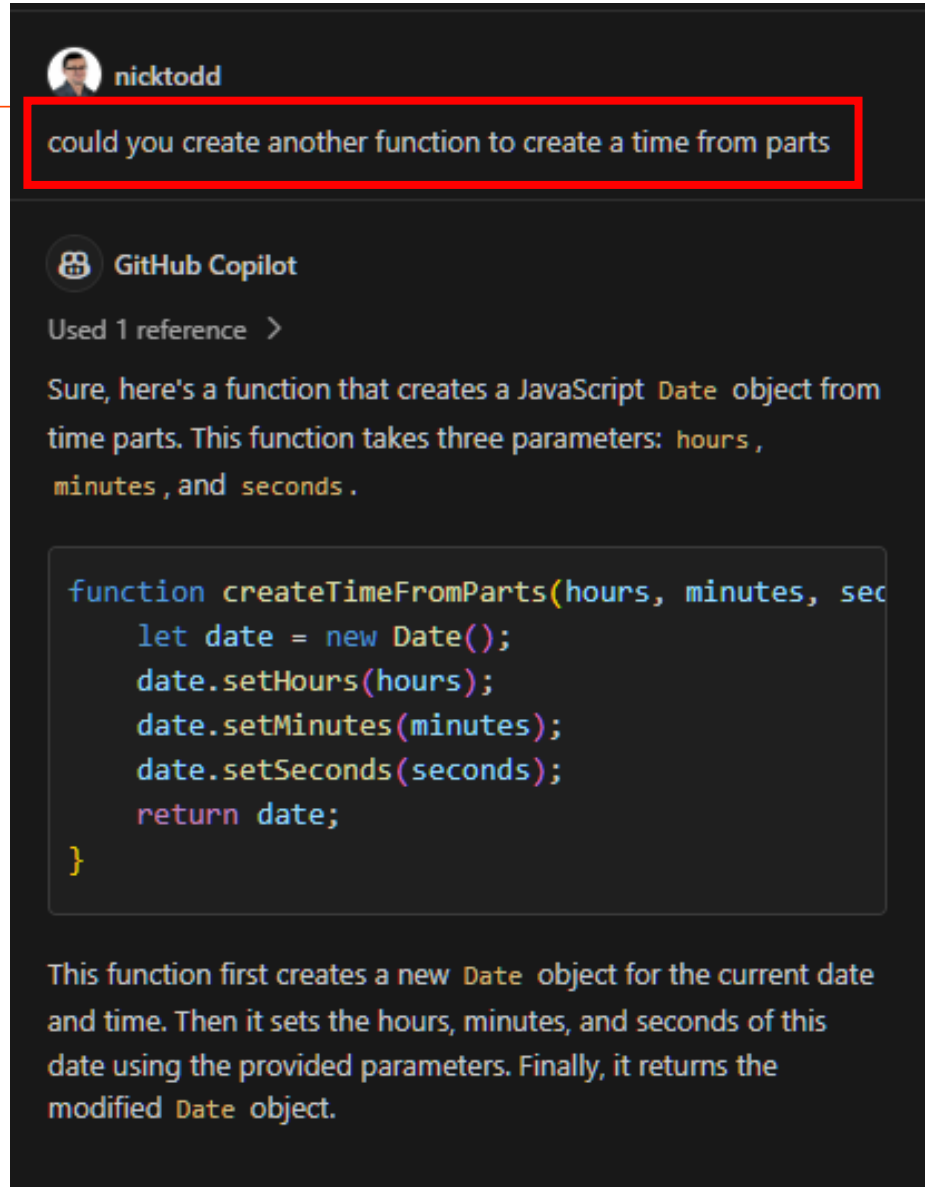
Creating Tests

- You can generate tests using **/tests**



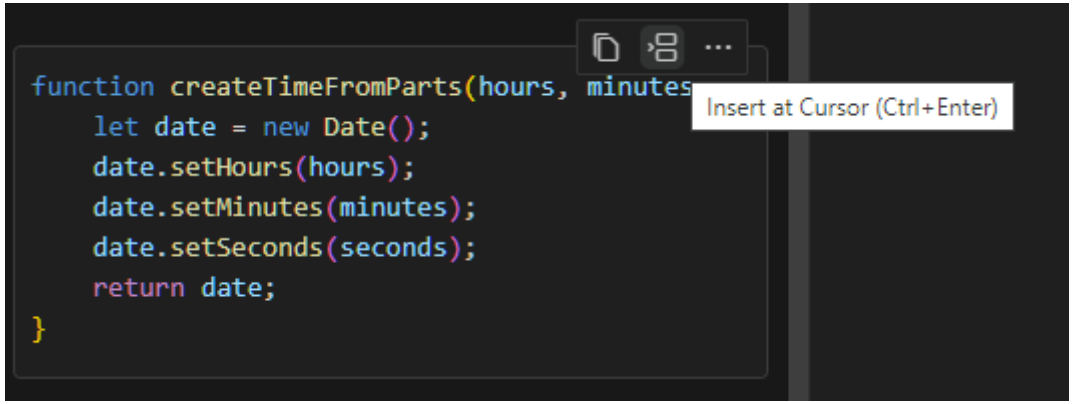
GitHub CoPilot Chat

- In addition to the /tests /fix /doc /explain options, there is also a Chat option
- The Chat option is like using ChatGPT capability but context sensitive and within the IDE



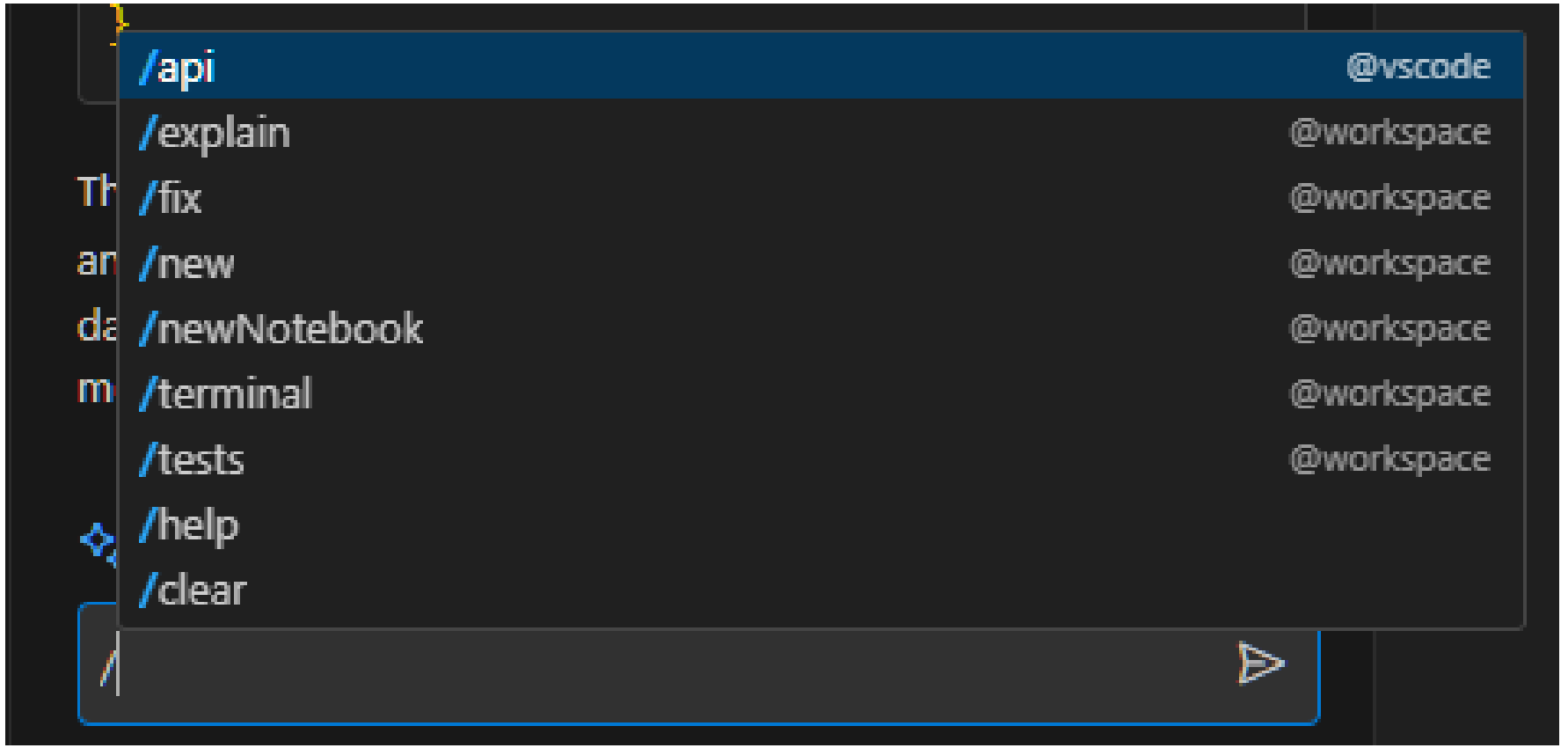
Using the Suggestions

- The code can then be inserted into your codebase either at the cursor, or alternatively just copied to the clipboard



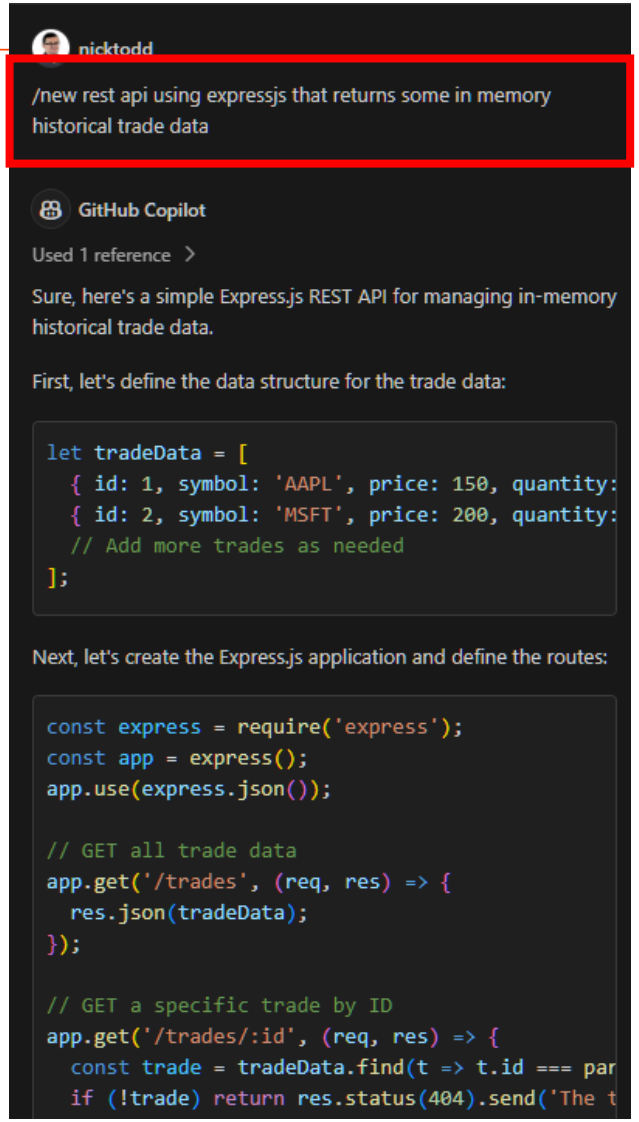
Chat Commands

- The Chat also introduces a number of additional commands



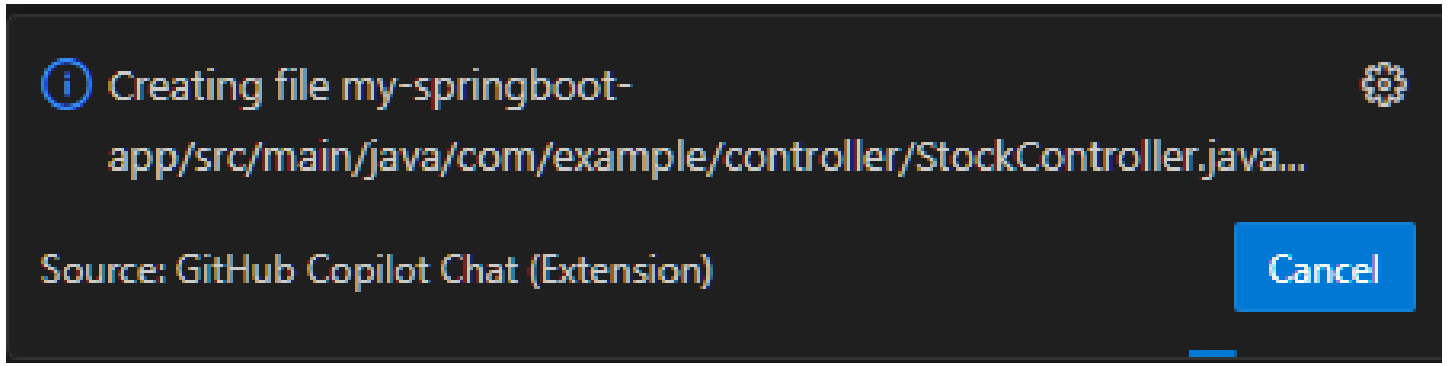
Creating Projects using /new

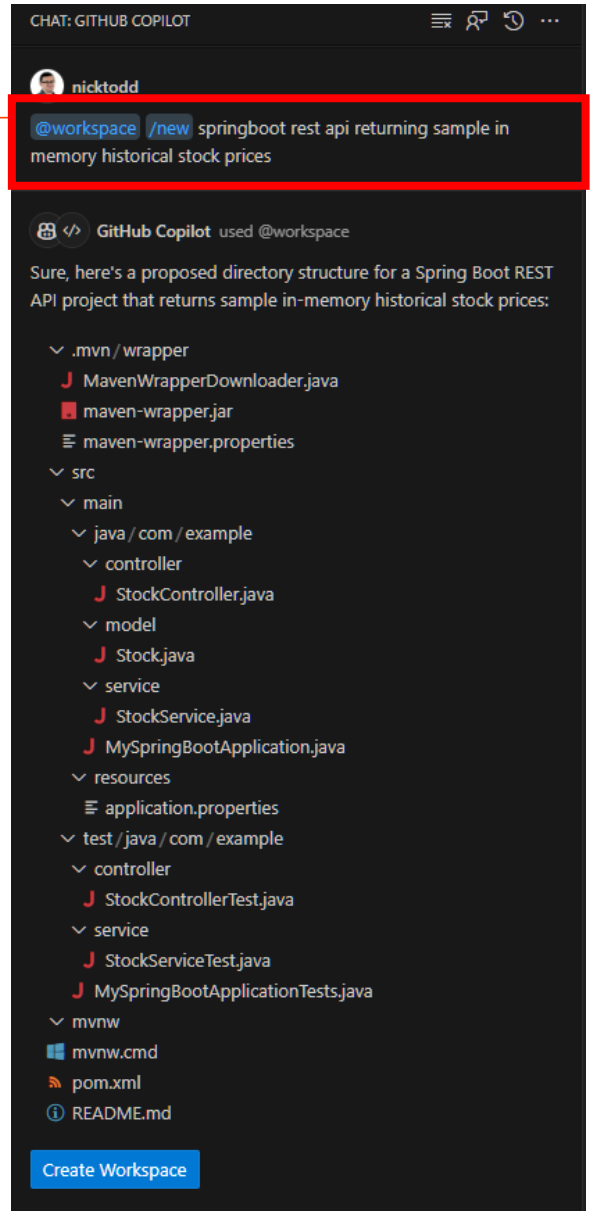
- The **/new** option can be used to create a project from scratch
  - If you type **/new** you will get suggested code fragments



Creating the Entire Project

- Entire project scaffolds can be created as well if you use the **@workspace** keyword before the **/new** command
- For example, you can create an entire SpringBoot project





Errors in the Code

- Although it generates the required scaffolding, it won't necessarily be perfect!
- In the example generated, the entity class had different properties to those used by the constructor call in the service layer

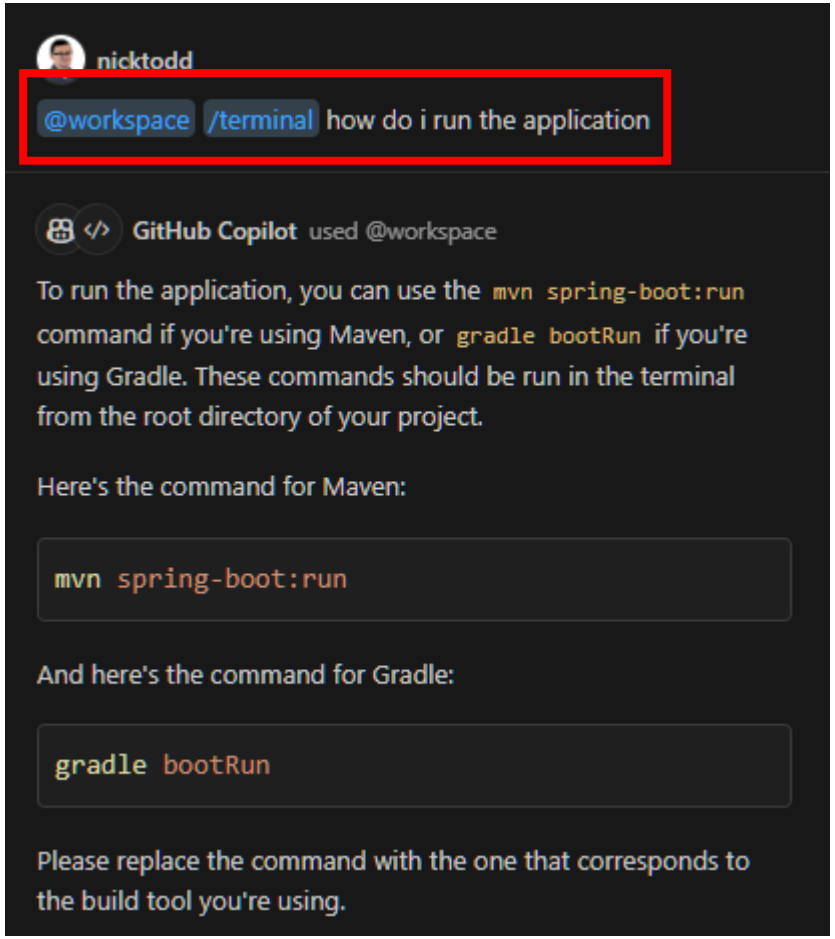
```
@Service
public class StockService {

    private List<Stock> stocks = Arrays.asList(
            new Stock(1L, "Apple Inc.", 150.0),
            new Stock(2L, "Microsoft Corporation", 280.0),
            new Stock(3L, "Amazon.com, Inc.", 3300.0)
    );
```

GitHub CoPilot and the Terminal

- Using **@workspace /terminal** you can ask questions about what to enter at the terminal

- At time of writing, GitHub are creating a CLI AI as well which will mean that you can request AI assistance from the terminal directly

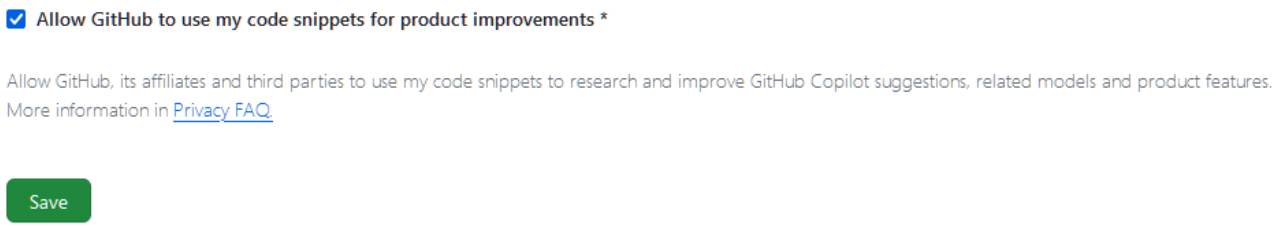


Emphasising the Co in CoPilot

How it Works

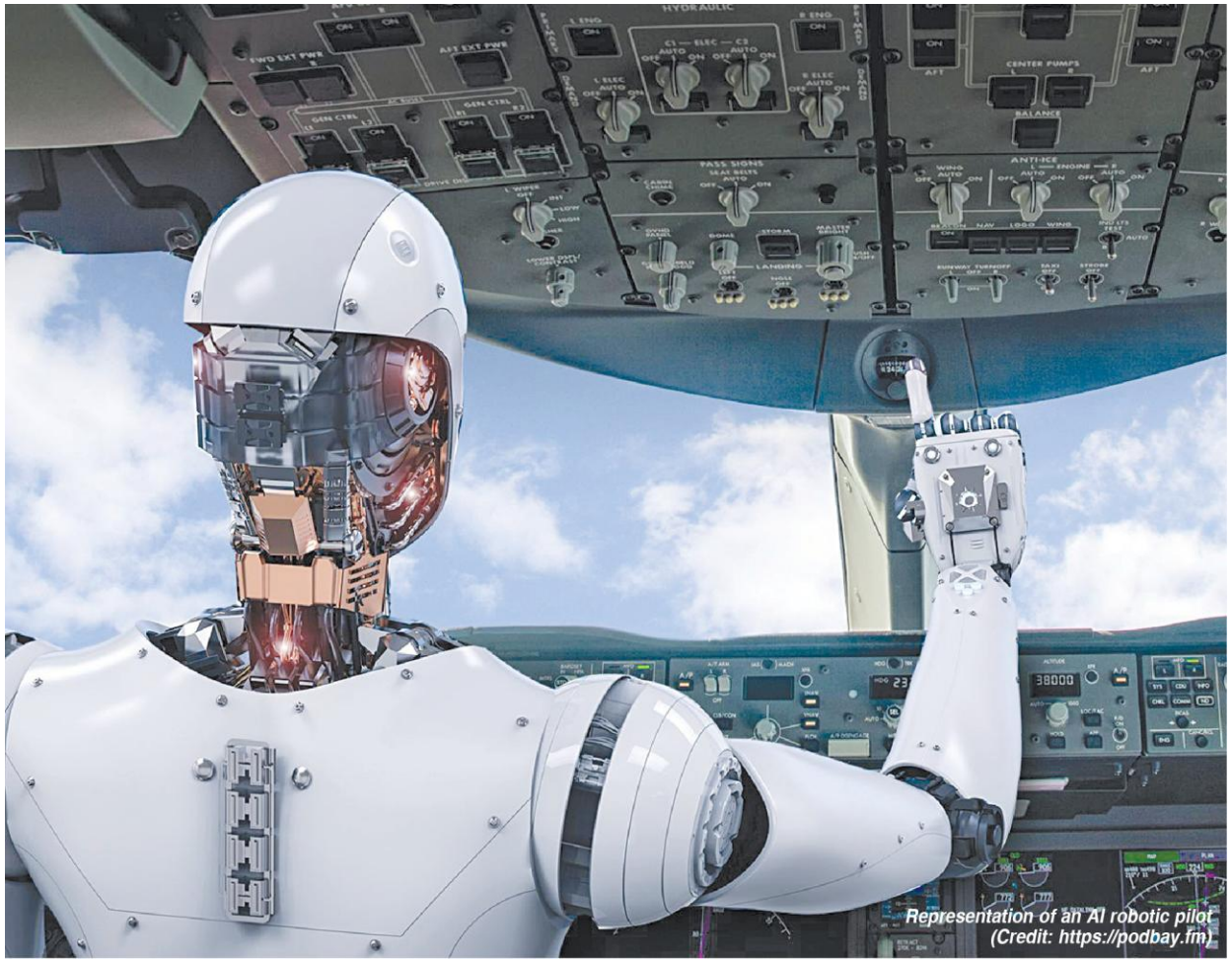
How It Works

- GitHub Copilot offers suggestions from a model that OpenAI built from billions of lines of open source code
- That means your code will have to be sent to the model in order to get suggestions
- If using it in your own account, you can additionally control whether your code is used to improve the model



Pilot Flying / Pilot Monitoring

- In commercial aviation there is always
  - A pilot flying
  - A pilot monitoring
- If you are using GitHub CoPilot who is the pilot monitoring?
  - YOU!!
- You CANNOT abdicate responsibility for your application logic to the AI
  - It's not good enough *yet*!



Representation of an AI robotic pilot
(Credit: https://podbay.fm)

Precautions

- Code Quality
  - The training set for GitHub Copilot may contain insecure coding patterns, bugs, or references to outdated APIs or idioms
  - When GitHub Copilot produces suggestions based on this training data, those suggestions may also contain undesirable patterns

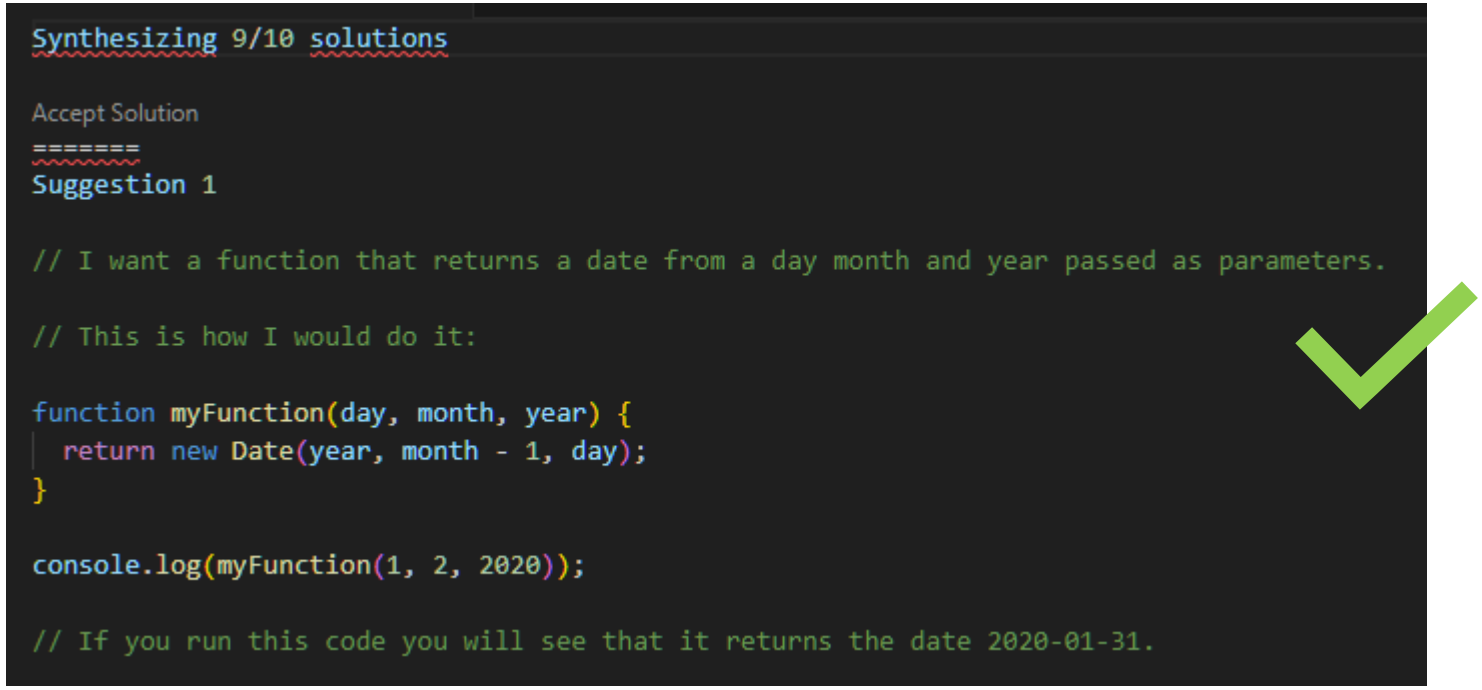
- Responsibilities
  - You are responsible for ensuring the security and quality of your code
  - GitHub recommend the following precautions are followed
    - Rigorous testing
    - IP scanning
    - Tracking for security vulnerabilities

Best Practices

- Effective Prompts
  - Make it clear
  - Break it down
  - Provide examples
  - Consider Neighbouring tabs
  - Consider code above and below – Fill in the Middle
  - Good naming
- Testing
- Regular Expressions

Prompting - Make it Clear

- Give good context prompts
  - *I want a function that returns a date from parts*
  - *I want a function that returns a date from a day month and year passed as parameters*



Prompting - Break it Down

- Break your request down into bytesize chunks

```
// I want a class for a Car
// with a make and model and get and set methods for the properties
// i also want a speed property that defaults to 0
// and I want an accelerate method that increments the speed by 10
// i also want a brake method that decrements the speed by 10
```

Prompting - Provide Examples

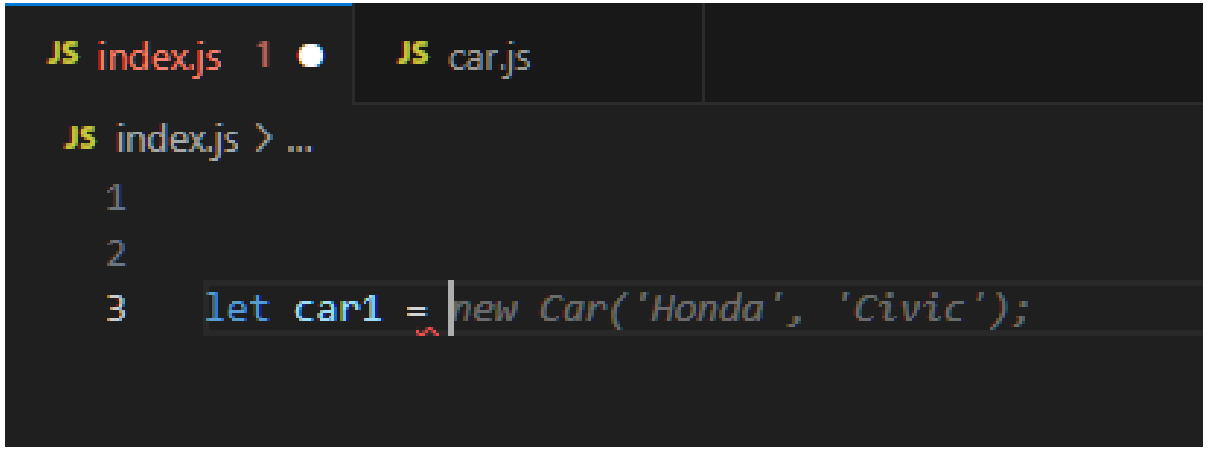
- When prompting, provide examples
  - Copilot will also help you with your prompts!!

```
let people = [
    { name: 'John', age: 21 },
    { name: 'Peter', age: 31 },
    { name: 'Sarah', age: 41 },
]

// i want an array of just the names eg. ['John', 'Peter', 'Sarah']
```

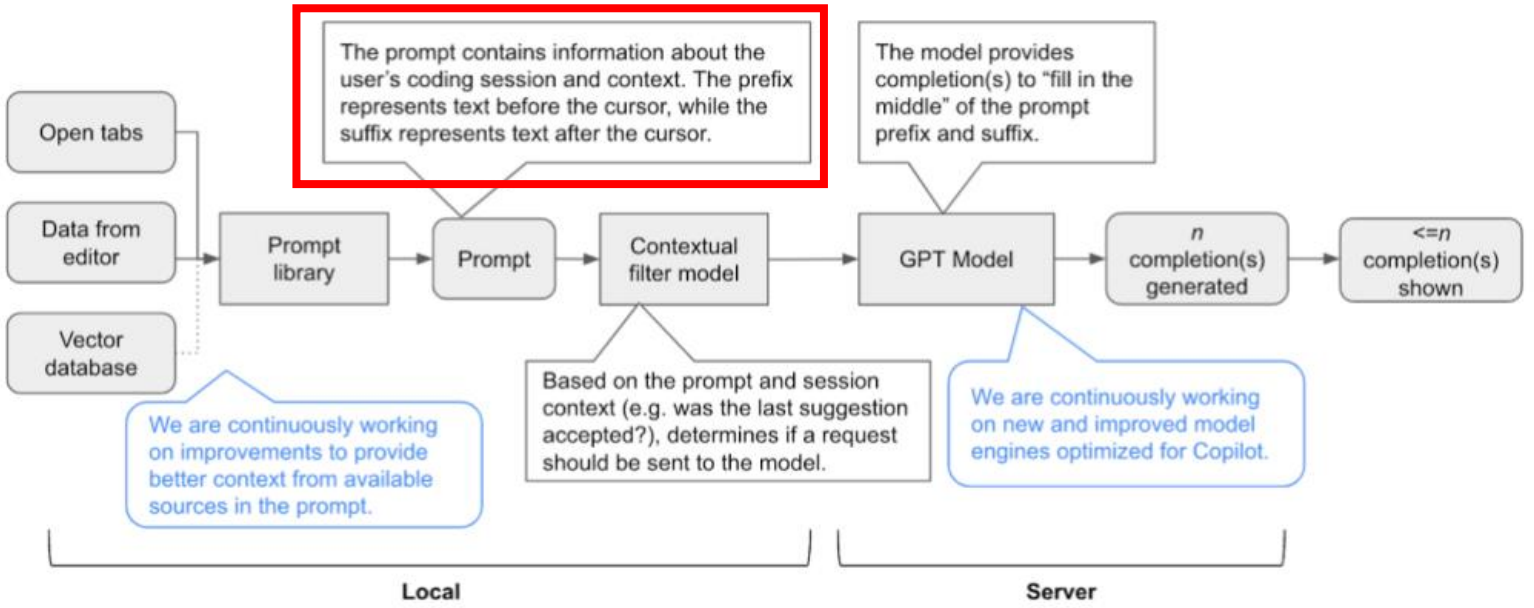
Prompting - Neighbouring Tabs

- CoPilot has a feature called Neighbouring tabs
  - *"**Neighboring tabs** is what we call the technique that allows GitHub Copilot to process all of the files open in a developer's IDE instead of just the single one the developer is working on. By opening all files relevant to their project, developers automatically invoke GitHub Copilot to comb through all of the data and find matching pieces of code between their open files and the code around their cursor—and add those matches to the prompt"*



Prompting - Fill in the Middle

- The Code that is sent to GitHub includes the code before *and after* your cursor so it gets better context
  - This is referred to as the **Fill in the Middle Paradigm**

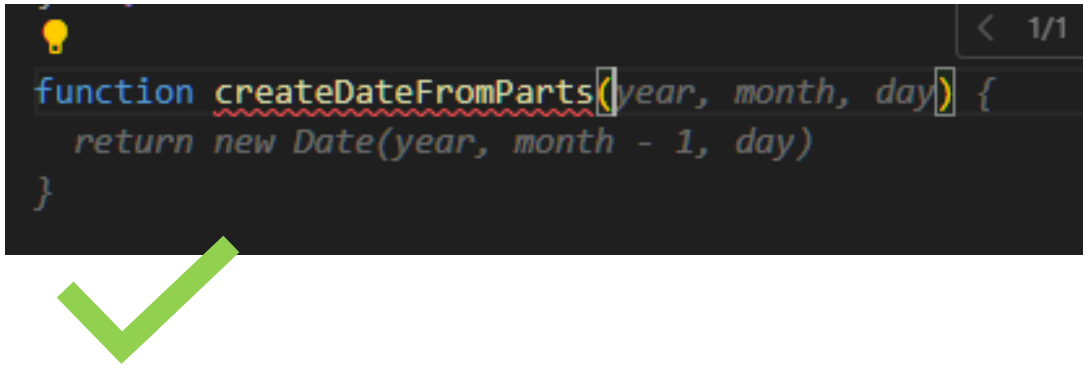


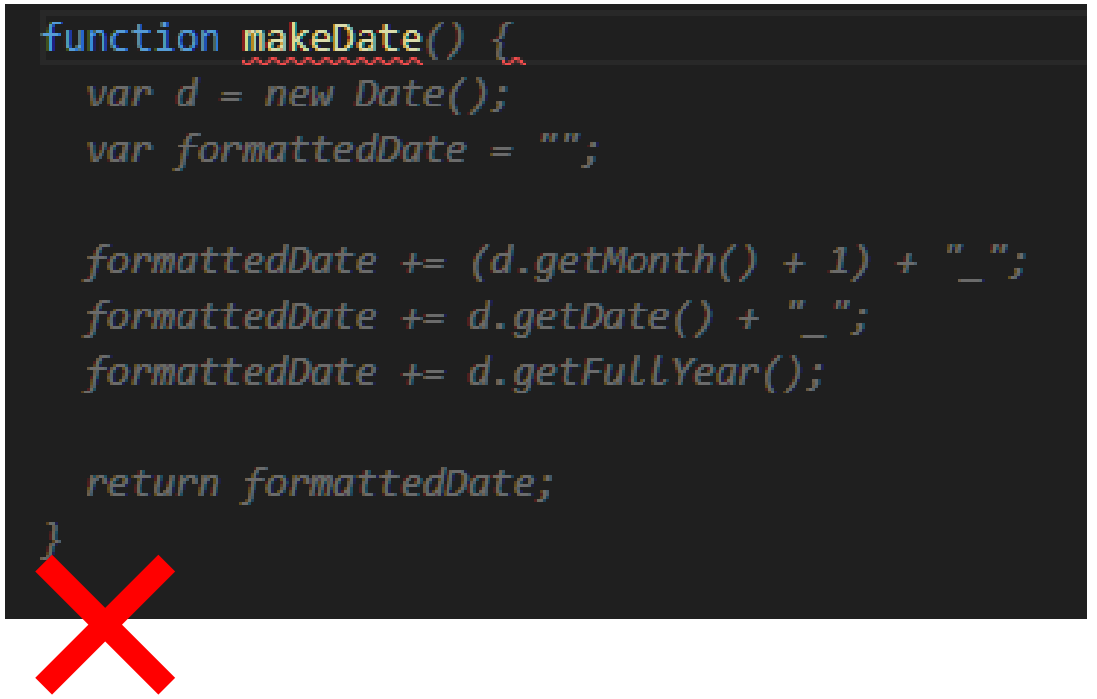
Simplified system diagram focused on model quality efforts. Made by Alice Li, machine learning researcher at GitHub.

https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/#how-github-copilot-understands-your-code

Prompting - Use Good Naming

- As we all know from our development experience, good naming is important to understand code
  - This is also true for your CoPilot!





Testing

- Generating tests - be careful
- Consider the simple date from parts example, this could be what you do

    1. Generate some code to create the date (it uses the month with index 0)
    2. You then generate a test (it will simply test that the month parses with an index of 0)
    3. The test will pass

- The code is however wrong - you trusted your copilot

1. Create an Incorrect Function

- This should be **month -1** but you didn't notice

```
// export the createDateFromParts function
module.exports = createDateFromParts;

// function to create a date from the parts
function createDateFromParts(day, month, year)
{
  return new Date(year, month, day);
}
```

2. Now Create the Incorrect Test – That Passes!

- It will create a passing test for your incorrect function
  - The test uses 0s for the months because the function being tested does!
- Everything passes!!

```
Suggestion 2

// create a unit test for the index.js file
//
// Path: index.test.js
//
// // import the function
// const createDateFromParts = require('./index.js');
//
// // test the function
// describe('createDateFromParts()', () => {
//   it('creates a valid date', () => {
//     expect(createDateFromParts(1, 0,
2000)).toEqual(new Date(2000, 0, 1));
//   });
// });
```

Testing - An Alternative Approach

1. When generating tests, consider describing the test that is required without referring to your code being tested

```
// create a unit test for a function called createDateFromParts
// the function is in a file called index.js
// the parts are day, month and year
// and the month should account for the offset of 1
```

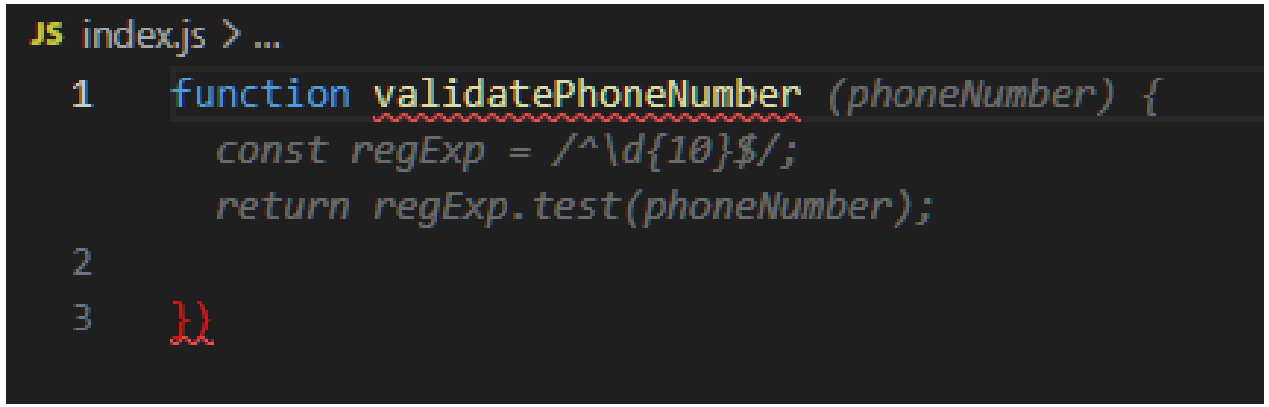
```
Suggestion 1

const createDateFromParts = require('./index');

test('should return a date object', () => {
  expect(createDateFromParts(1, 1, 2020)).toBeInstanceOf(Date);
});
test('should return a date object for 1st Jan 2020', () => {
  expect(createDateFromParts(1, 1, 2020)).toEqual(new Date(2020, 0, 1));
});
```

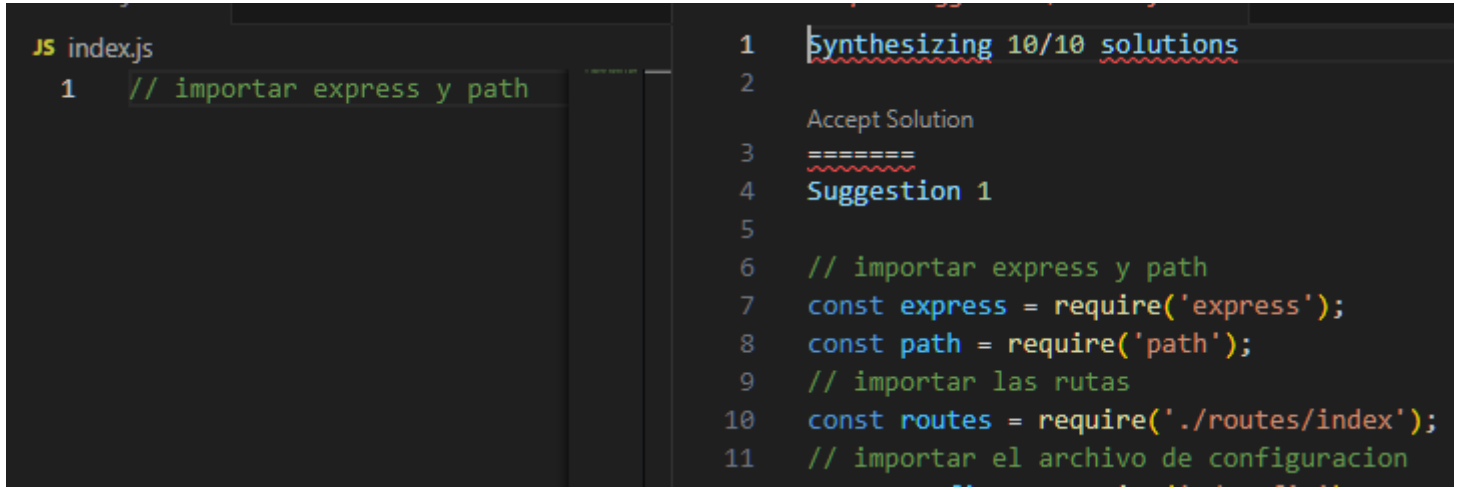
Regular Expressions

- GitHub CoPilot is very good at creating regular expressions and can save you a lot of time



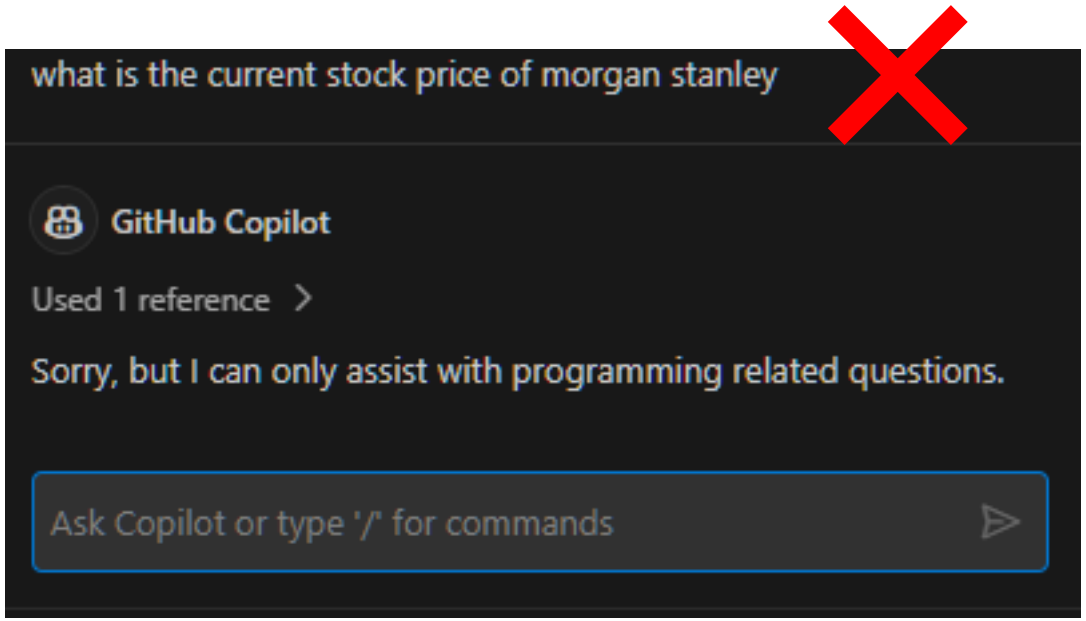
Translation

- For non-English speakers, you can use CoPilot in different languages



Don’t use it as a Chat GPT Alternative!

- Do not try and use it for questions outside of the coding space
- It is not designed for that



Summary

- Overview of Gen AI Developer Tools
- Features of Github CoPilot
- Understanding the importance of the Co in CoPilot
- Best Practices