A Practical Guide to

GitHub Copilot

Sina Mostafanejad

ACT-CMS Faculty Fellows Bootcamp
June 2024

What is GH Copilot?

``… an AI coding assistant that helps you write code faster and with less effort''

Main Features

- Code completion
- Chat
- Pull Request summaries
- Knowledge bases (Copilot Enterprise)

Interfaces

- IDE/text editor
- Command line interface (GH CLI)
- Chat interface through GitHub Mobile
- GitHub.com interface (Enterprise subscription)

Interfaces

- IDE/text editor
- Command line interface (GH CLI)
- Chat interface through GitHub Mobile
- GitHub.com interface (Enterprise subscription)

Getting Started with Copilot

- The following IDEs/text editors are supported
  - Microsoft Visual Studio
  - Visual Studio Code
  - Vim/Neovim
  - JetBrains IDEs
  - Microsoft Azure Data Studio

https://docs.github.com/en/copilot/using-github-copilot/getting-started-with-github-copilot

Getting Started with Copilot

- The following IDEs/text editors are supported
  - Microsoft Visual Studio
  - Visual Studio Code
  - Vim/Neovim
  - JetBrains IDEs
  - Microsoft Azure Data Studio

https://docs.github.com/en/copilot/using-github-copilot/getting-started-with-github-copilot

Supported Languages

- Many programming languages are supported including
  - Python,
  - JavaScript,
  - TypeScript,
  - Ruby,
  - Go,
  - C#
  - C/C++
- Copilot can also assist in query generation for databases.

Supported Languages

- Many programming languages are supported including

  - Python,
  - JavaScript,
  - TypeScript,
  - Ruby,
  - Go,
  - C#
  - C/C++

- Copilot can also assist in query generation for databases

Your First Suggestion

- Start writing code...
  - the suggestions start showing up as you write!
- Accept the suggestions by pressing the **tab** button
- Partially accept the suggestion by using **ctrl + -->**
- Hover the mouse over the suggestions to see alternative options
- Use **Alt + [** or **Alt +]** to switch between alternative suggestions
- Use **ctrl + Enter** to see a potential list of suggestions in a new panel

Code Completion

- You get suggestions from Copilot as you write code, so...

- **Exercise:**
  - Open the **00_code_completion.py** file and instruct Copilot to write you a simple [e.g., *add()*] function

Comments Are Valuable!

- Provide as much information as possible
- Offering examples is helpful, especially working with data or strings
- Top-level comments can give context about the overall intended code
- Useful for boilerplate code to get you started

Comments Are Valuable!

- **Exercise:**
  - Open the **01a_comments.py** file and use multiple comments to instruct Copilot to define an *add()* function, write unit test(s) for it and run the test(s).
  - Open the **01b_top_comment_solution.py** file and in a top-level comment ask Copilot to write a complete calculator class with add, subtract, … member functions. Provide as much detail as possible.

Be Specific, Please!

- All headers, modules and libraries are best to be included/imported manually.
- Be specific about the versions or libraries when asking Copilot
- **Exercise:**
  - Open the **02_specific_versions.py** file and instruct Copilot to write a "Hello World" print statement in Python 2.7 and 3.0 for you!

Context Matters!

- LLMs make inference based on the context
- If you keep relevant code files open in the IDE, Copilot uses their content to make better suggestions
- Closed files do not contribute to the context.

Chat Interface

- There is a Chat Interface within IDEs that can be used for chatting with Copilot.
  - Simply press the Copilot logo on the bottom right bar in the VSCode and select **GitHub Copilot Chat** to start, or
  - Press **Ctrl + Alt + I** to open the side chat panel
  - Navigate your conversation using up or down arrow buttons on the keyboard

Context Variables (#)

- Use the **#editor** context variable in the chat interface to provide additional context from the currently opened files in the VS/VSCode.
- Use **#file** to attach a file to your instruction/question to provide targeted context for better outcome

Context Variables (#)

- **Exercise:**

  - Work on **03_context.py** to define a **Calculator** class which
    - Implements an *addition* member function that wraps around the *add()* imported from the **00_code_completion.py** module.
    - After writing the class, call the *addition* function, and print the result.

  - **Hint:** Keep the **00_code_completion.py** open in your editor to provide context. You may need to rename the file before importing from it.

Context Variables (#)

- **Exercise:**
    - Using context variables (#), provide additional context for the **Calculator** class and ask Copilot in the chat how a **subtract** function can be added to the **00_code_completion.py** module or the **Calculator** class

Naming Conventions

- Give your functions and variables meaningful names
- Meaningful names create better codes
- Meaningful names generate better context and therefore, better suggestions from Copilot
- **Exercise:**
  - Open the **04_naming_convention.py** file and define a function with a random name [e.g., *asdfjkh23m()*] and see what Copilot suggest for its body.

Examples Help!

- As humans learn the new concepts better with specific examples, AI algorithms can do too.
- In your instructions and comments, try to provide specific examples (e.g., of the expected output, return values etc.)
- **Exercise**:
  - Open the **05_examples.py** file and instruct Copilot to write a function that takes two arrays of integers as input and *returns the sum of the two arrays*.

Inline Chat

- Chat can be done in an inline fashion
  - Press **ctrl + I** to see a pop-up chat bar.
  - Useful for quick fixes with code diffs and documentation

- Highlighting the relevant code narrows down context and helps with the suggestions

- Look for **Magic Sparkles** to get help from Copilot Inline chat

Inline Chat

- **Exercise:**
    - Open the **05_examples_solution.py** file and instruct Copilot through inline chat to write NumPy/Google style docstring for your function(s)

Slash Commands (/)

- The **Slash Commands** are designed for common tasks
  - **/doc** ---> Add documentations for objects
  - **/explain** ---> Explain the highlighted code
  - **/fix** ---> Provide a potential fix for the highlighted problematic code
  - **/generate** ---> Generate code as instructed
  - **/help** ---> Get help on Copilot Chat
  - **/optimize** ---> Analyze and enhance efficiency of the highlighted code
  - **/simplify** ---> Simplify the highlighted code
  - **/tests** ---> Write unit test for the highlighted code
  - **/clear** ---> Clear the chat

Copilot Agents (@)

- Agents can help with a large variety of tasks providing context on their own.
- Instead of providing context in our prompts, we can ask Copilot to build the context on its own.
- Currently there are 3 agents in Copilot:
  - **@workspace** ---> Context from workspace
  - **@vscode** ---> Questions related to VSCode and its structures
  - **@terminal** ---> Chat pertinent to the terminal commands

Copilot Agents (@)

- **@workspace** builds the context from our workspace and can be used for:
  - Looking for files, searching for modules, class or function definitions etc.
  - Adding new functionalities
  - Fixing the code issues and errors
  - Suggestions for refactoring/restructuring the code
- **Exercise:**
  - Close all files in the editor and open the Copilot chat interface. Use the **@workspace** agent and ask where is the *add()* function defined?

References

- https://docs.github.com/en/copilot/using-github-copilot/using-github-copilot-code-suggestions-in-your-editor
- https://github.blog/2024-03-25-how-to-use-github-copilot-in-your-ide-tips-tricks-and-best-practices/
- https://github.blog/2023-05-17-inside-github-working-with-the-llms-behind-github-copilot/
- https://github.blog/2023-05-17-how-github-copilot-is-getting-better-at-understanding-your-code/
- https://dev.to/github/a-beginners-guide-to-prompt-engineering-with-github-copilot-3ibp
- https://medium.com/@yar.dobroskok/github-copilot-workspace-new-development-experience-d69857fbd067