

AI Programming

Copyright © 2026 by
Robert M. Dondero, Ph.D
Princeton University

Objectives

- The lecture will cover:
 - How to use generative AI to help you compose code
- This lecture will **not** cover:
 - How generative AI works
 - How to develop applications that offer users AI functionality

I'm not an expert!

Agenda

- **Generative AI for SW engineering**
- AI programming tools
- Prompt heuristics
- Copilot in VS Code
- Example
- Personal observations
- AI in COS 333

Generative AI for SWE

GenAI can be useful at all stages of SWE:

- Requirements analysis
- Design
- Implementation
- Debugging
- Testing
- Evaluation
- Maintenance

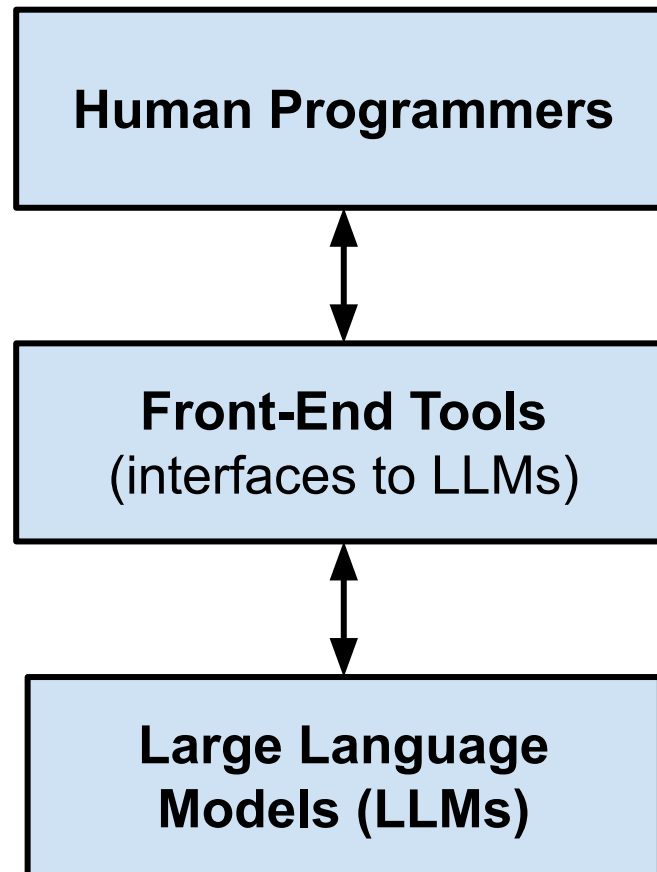
Our focus now (and your initial focus?)



Agenda

- Generative AI for SW engineering
- **AI programming tools**
- Prompt heuristics
- Copilot in VS Code
- Example
- Personal observations
- AI in COS 333

AI Programming Tools



AI Programming Tools

- LLMs
 - Known to be good for programming:
 - Claude Mythos / Fable / Opus family (Anthropic)
 - GPT-5.x family (OpenAI)
 - Gemini 3.x (Google)

AI Programming Tools

- Front-end tools
 - **Terminal-based AI tools**
 - Tools that you run from a shell prompt
 - **Examples:** Claude Code, GitHub Copilot, Aider, Cline

AI Programming Tools

- Front-end tools
 - **Integrated Development Environment (IDE) plugins**
 - Tools that you run as plugins within IDEs (VS Code, JetBrains, XCode, Eclipse, ...)
 - Provide:
 - Context-aware code assistance
 - Chat via prompts
 - **Examples:** GitHub Copilot, JetBrains AI Assistant, Amazon Q Developer, Tabnine, Supermaven, Codeium

AI Programming Tools

- Front-end tools
 - **AI-first code editors**
 - Tools that are combination editors and IDE plugins
 - **Examples:** Cursor, Devin Desktop, Zed

AI Programming Tools

- Use a **terminal-based** AI tool when you need the tool to have:
 - Broad control of your computer
- Use an **IDE plugin** AI tool when you need the tool to have:
 - Rich understanding of your project and rich interaction with your editor

AI Programming Tools

Task	Terminal-Based AI	IDE Plugin AI
Run shell commands	Excellent	Usually possible via integrated terminal
Manage (long-running) processes/services	Excellent	Awkward, usually limited/indirect
Work with multiple unrelated repos (files outside project)	Excellent	Often restricted
SSH into other computers	Excellent	Possible, but less natural
Use Unix pipes/composition	Excellent	Awkward
Operate on arbitrary server/infrastructure	Excellent	Usually less convenient
Automate outside the project	Excellent	Less natural

AI Programming Tools

Task	Terminal-Based AI	IDE Plugin AI
Edit code	Good	Excellent
Understand open files/tabs	Not inherently	Excellent
Understand editor selections	Not inherently	Excellent
See compiler/linter diagnostics inline	No	Excellent
Navigate symbols/references	Via external tools	Excellent
Debug interactively	Difficult, via CLI debugger	Excellent
Visually show file diffs	No	Excellent

AI Programming Tools

- Typical **terminal-based** AI prompts:
 - Figure out why our app has been restarting.”
 - “Upgrade all our repos from Python 3.12 to Python 3.14.”
- Typical **IDE Plugin** AI prompts:
 - “React component XXX is rendering unnecessarily; determine why and fix the problem.”
 - “Rename interface XXX to YYY and update every reference.”

AI Programming Tools

- For helping you write COS 333 project code I recommend an **IDE plugin**
- Use an IDE plugin in **two modes**:
 - Context aware code assistance
 - Probably will come to you naturally
 - Chat via prompts
 - You might benefit from some suggestions of heuristics...

Agenda

- Generative AI for SW engineering
- AI programming tools
- **Prompt heuristics**
- Copilot in VS Code
- Example
- Personal observations
- AI in COS 333

Prompt Heuristic 1

- Prompt **incrementally** by app feature
 - Prompts describe too many app features =>
 - Code might be difficult to understand
 - Code might be difficult to test
 - Example prompt (bad):
 - Develop a 3-tier app...

Prompt Heuristic 1

- Prompt **incrementally** by app feature (cont.)
 - Example prompts (good):
 - Develop the first endpoint ...
 - Develop the second endpoint...
 - ...
 - Develop client-side code...
 - Add authentication...
 - Check security...
 - ...

Prompt Heuristic 2

- Be specific
 - Describe **inputs** to the desired code
 - Describe proper **outputs** from the desired code
 - Describe proper handling of **boundary cases** by the desired code
 - Describe proper **error handling** by the desired code
 - Parameter validation
 - Exception handling
 - Provide **examples** as necessary

Prompt Heuristic 2

- Often recommended prompt pattern:

```
Role:  
Senior software engineer.  
  
Context:  
[project description]  
  
Task:  
[specific task]  
  
Requirements:  
[list of requirements]  
  
Constraints:  
[list of constraints]  
  
Output:  
[exact format required]  
  
Review:  
{introspection on design decisions,  
weaknesses, security vulnerabilities,  
etc.]
```

However...

Prompt Heuristic 3

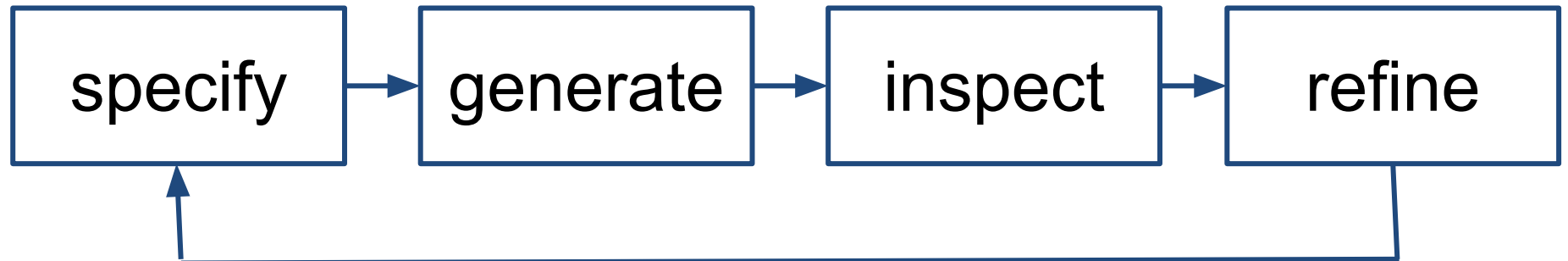
- Prompt at a **moderate** level of detail
 - Prompts too **shallow** =>
 - AI will guess
 - You'll waste time composing many follow-up prompts
 - AI might “drift” during long prompt sequences
 - Prompts too **deep** =>
 - You'll waste time composing the prompt!
 - AI might be overwhelmed
 - You might lose learning opportunities

Prompt Heuristic 3

- Prompt at a **moderate** level of detail (cont.)
 - Prompts at moderate level =>
 - Gives the model enough **structure** to be useful
 - Gives the model enough **flexibility** to adapt
 - **Guide** the AI, but don't **overwhelm** the AI

Prompt Heuristic 3

- Prompt at a **moderate** level of detail (cont.)



Moderate prompting keeps the loop fast and stable

Prompt Heuristic 3

- (Recall) often recommended prompt pattern:

OK, but your
goal should be
moderation,
not depth

```
Role:  
Senior software engineer.  
  
Context:  
[project description]  
  
Task:  
[specific task]  
  
Requirements:  
[list of requirements]  
  
Constraints:  
[list of constraints]  
  
Output:  
[exact format required]  
  
Review:  
{introspection on design decisions,  
weaknesses, security vulnerabilities,  
etc.]
```

Prompt Heuristic 4

- Require **testing**
 - Require AI to write test code
 - **Maybe before it writes the production code!!!**
 - (If possible) require AI to compile production and test code
 - (If possible) require AI to run the test code against the production code

Prompt Heuristic 5

- Remember to prompt for **debugging**
 - Show error messages
 - Require corrections

Prompt Heuristic 6

- Remember to prompt for **explanations**
 - Ask the AI to explain any code that you don't understand

Prompt Heuristic 7

- Record important **decisions**
 - Command the AI to keep a record of important decisions in a **history.md** (or whatever) file
 - Tell the AI to read the **history.md** file at the beginning of each conversation (or prompt)

Agenda

- Generative AI for SW engineering
- AI programming tools
- Prompt heuristics
- **Copilot in VS Code**
- Example
- Personal observations
- AI in COS 333

Copilot in VS Code

- GitHub Copilot in VS Code
 - Can choose underlying LLM
 - Default: Claude Sonnet 5.5
 - Alternatives: GPT-6 Luna, many others
 - Can fine-tune underlying model
 - Size of context window, think deeply vs respond quickly, ...

Copilot in VS Code

- GitHub Copilot (in VS Code) (cont.)
 - (con) Free tier is limited
 - Restricted access during peak times
 - Limited context depth
 - (pro) *GitHub Copilot Student*
 - Free GitHub Copilot Pro
 - Higher free tier limits
- **You should subscribe to GitHub Copilot Student!!!**

Copilot in VS Code

The screenshot displays the Visual Studio Code interface with the Copilot extension. The Explorer on the left shows a project named 'PennyCopilot7' with files like `__pycache__`, `.github`, `copilot-instructions.md`, `templates`, `home.html`, `searchform.html`, `searchresults.html`, `database.py`, `penny.sqlite`, `prompt1.txt`, `prompt2.txt`, `prompt3.txt`, `prompt4.txt`, `prompt5.txt`, `prompt6.txt`, `runserver.py`, `testdatabase.py`, and `testserver.py`.

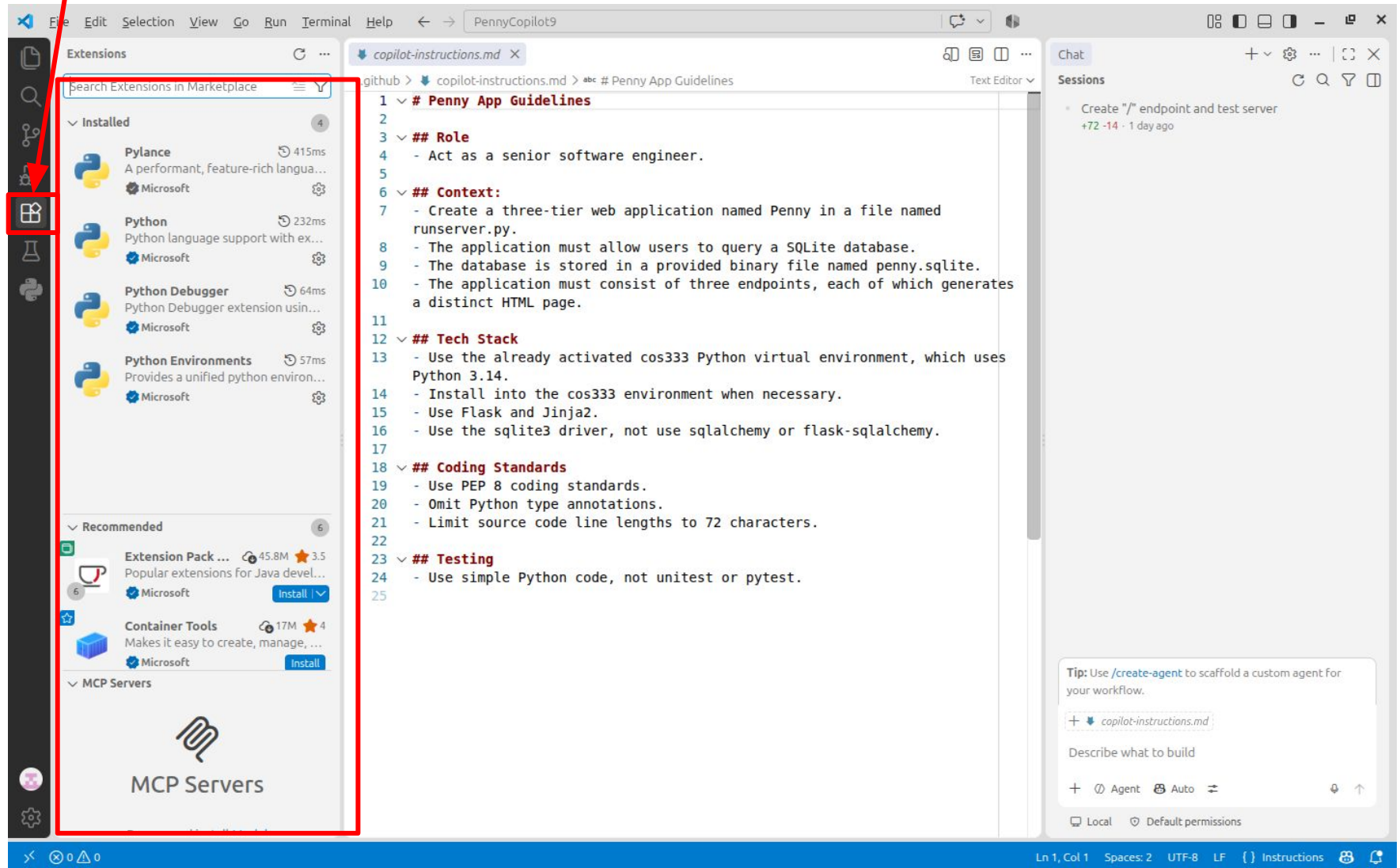
The main editor shows the `copilot-instructions.md` file with the following content:

```
1 # Penny App Guidelines
2
3 ## Role
4 - Act as a senior software engineer.
5
6 ## Context:
7 - Create a three-tier web application named Penny in a file named
  runserver.py.
8 - The application must allow users to query a SQLite database.
9 - The database is stored in a provided binary file named penny.sqlite.
10 - The application must consist of three endpoints, each of which
   generates a distinct HTML page.
11
12 ## Tech Stack
13 - Use the already activated cos333 Python virtual environment, which uses
   Python 3.14.
14 - Install into the cos333 environment when necessary.
15 - Use Flask and Jinja2.
16 - Use the sqlite3 driver, not use sqlalchemy or flask-sqlalchemy.
17
18 ## Coding Standards
19 - Use PEP 8 coding standards.
20 - Omit Python type annotations.
21 - Limit source code line lengths to 72 characters.
22
23 ## Testing
24 - Use simple Python code, not unittest or pytest.
25
```

The Chat panel on the right shows a conversation with Copilot. The task is 'Create "/" endpoint and testserver.py'. The chat history includes a review of `runserver.py` and the creation of `testdatabase.py`. The current message from Copilot is: 'Now create `testdatabase.py`.' Below this, it shows the command `python3 testdatabase.py && echo "---" && python3 t...` and the result: 'Both passed. Factored database access into `database.py` (exposes `query_books`, hides SQLite specifics), updated `runserver.py` to call `database.query_books` with no DB-specific code, and created `testdatabase.py` to test it directly. `testserver.py` still passes unchanged.'

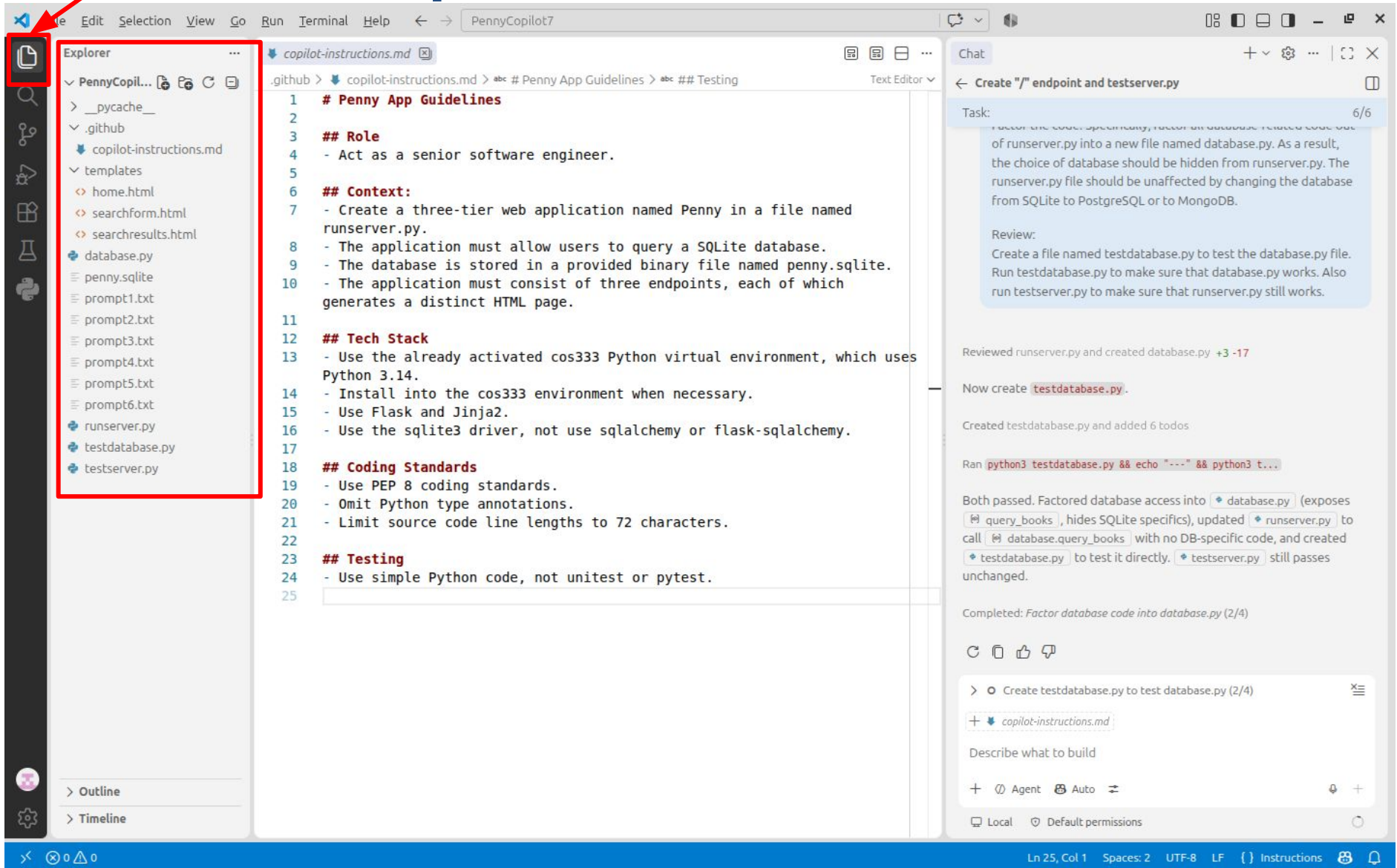
The bottom of the chat panel shows a list of tasks: 'Create testdatabase.py to test database.py (2/4)'. The status bar at the bottom indicates 'Ln 25, Col 1 Spaces: 2 UTF-8 LF {} Instructions'.

Extensions Copilot in VS Code



Explorer

Copilot in VS Code



Open file

Copilot in VS Code

The screenshot shows the VS Code interface with the following components:

- Explorer:** A sidebar on the left showing the file structure. The file `copilot-instructions.md` is highlighted with a red box and a red arrow pointing to it from the text "Open file".
- Editor:** The main workspace showing the content of `copilot-instructions.md`. The content is as follows:

```
1 # Penny App Guidelines
2
3 ## Role
4 - Act as a senior software engineer.
5
6 ## Context:
7 - Create a three-tier web application named Penny in a file named
  runserver.py.
8 - The application must allow users to query a SQLite database.
9 - The database is stored in a provided binary file named penny.sqlite.
10 - The application must consist of three endpoints, each of which
   generates a distinct HTML page.
11
12 ## Tech Stack
13 - Use the already activated cos333 Python virtual environment, which uses
   Python 3.14.
14 - Install into the cos333 environment when necessary.
15 - Use Flask and Jinja2.
16 - Use the sqlite3 driver, not use sqlalchemy or flask-sqlalchemy.
17
18 ## Coding Standards
19 - Use PEP 8 coding standards.
20 - Omit Python type annotations.
21 - Limit source code line lengths to 72 characters.
22
23 ## Testing
24 - Use simple Python code, not unittest or pytest.
25
```
- Chat:** A sidebar on the right showing a chat window. The task is "Create '/' endpoint and testserver.py". The chat content includes a task description, a review, and a completed task.

Task: Create "/" endpoint and testserver.py

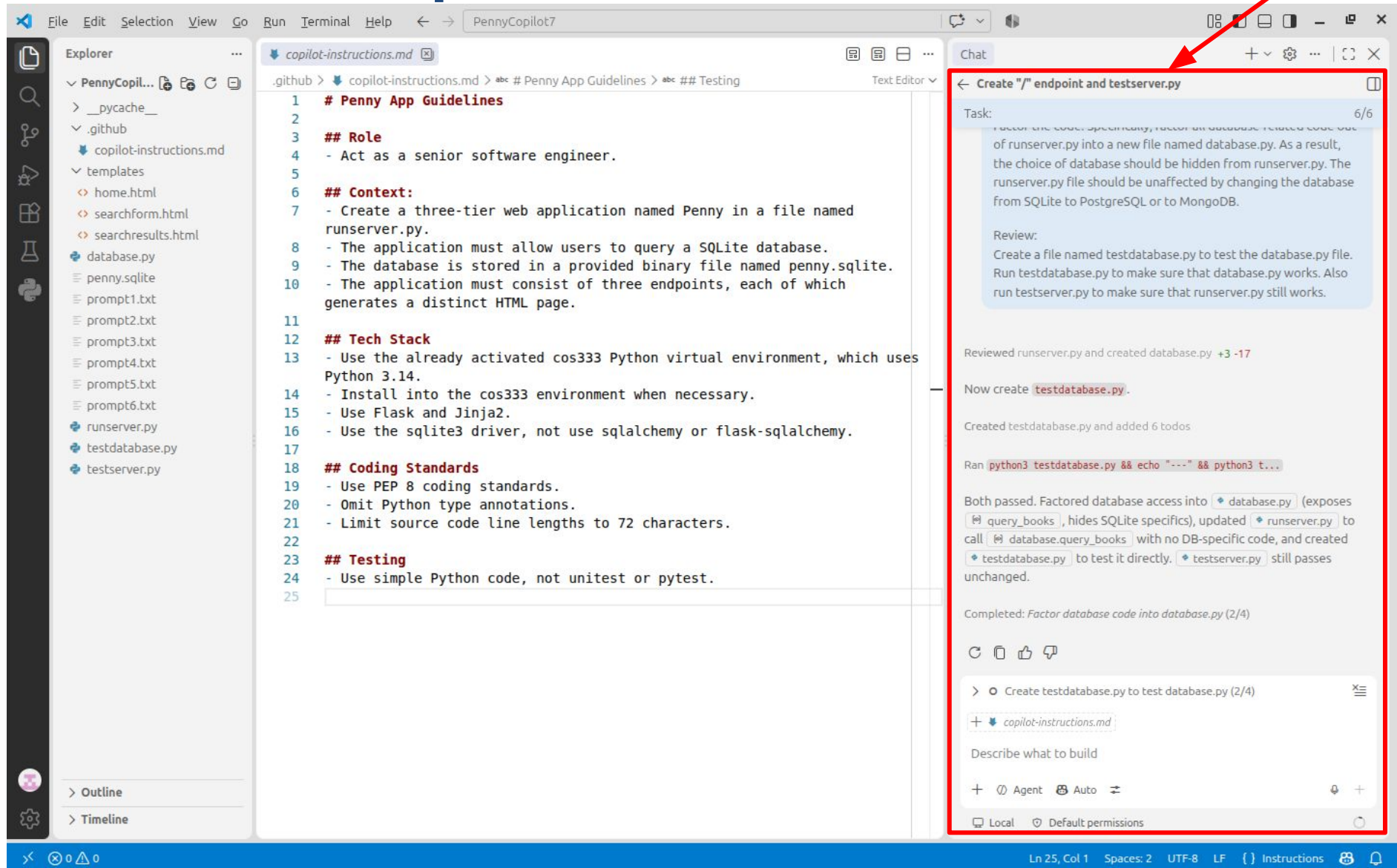
Review:

Create a file named testdatabase.py to test the database.py file. Run testdatabase.py to make sure that database.py works. Also run testserver.py to make sure that runserver.py still works.

Completed: Factor database code into database.py (2/4)

Copilot in VS Code

Copilot transcript



The screenshot displays the Visual Studio Code interface with the Copilot extension. The main editor shows a file named `copilot-instructions.md` with the following content:

```
1 # Penny App Guidelines
2
3 ## Role
4 - Act as a senior software engineer.
5
6 ## Context:
7 - Create a three-tier web application named Penny in a file named
  runserver.py.
8 - The application must allow users to query a SQLite database.
9 - The database is stored in a provided binary file named penny.sqlite.
10 - The application must consist of three endpoints, each of which
  generates a distinct HTML page.
11
12 ## Tech Stack
13 - Use the already activated cos333 Python virtual environment, which uses
  Python 3.14.
14 - Install into the cos333 environment when necessary.
15 - Use Flask and Jinja2.
16 - Use the sqlite3 driver, not use sqlalchemy or flask-sqlalchemy.
17
18 ## Coding Standards
19 - Use PEP 8 coding standards.
20 - Omit Python type annotations.
21 - Limit source code line lengths to 72 characters.
22
23 ## Testing
24 - Use simple Python code, not unittest or pytest.
25
```

The Chat window on the right, titled "Create "/" endpoint and testserver.py", shows the Copilot transcript. A red arrow points from the "Copilot transcript" text to the Chat window. The transcript includes:

- Task:** 6/6
- Review:** Create a file named `testdatabase.py` to test the `database.py` file. Run `testdatabase.py` to make sure that `database.py` works. Also run `testserver.py` to make sure that `runserver.py` still works.
- Completed:** Factored database code into `database.py` (2/4)
- Next steps:** Create `testdatabase.py` to test `database.py` (2/4)

The bottom status bar shows "Ln 25, Col 1 Spaces: 2 UTF-8 LF {} Instructions".

Copilot in VS Code

The screenshot displays the VS Code interface with the Copilot extension. The main editor shows a file named `copilot-instructions.md` with the following content:

```
1 # Penny App Guidelines
2
3 ## Role
4 - Act as a senior software engineer.
5
6 ## Context:
7 - Create a three-tier web application named Penny in a file named
  runserver.py.
8 - The application must allow users to query a SQLite database.
9 - The database is stored in a provided binary file named penny.sqlite.
10 - The application must consist of three endpoints, each of which
   generates a distinct HTML page.
11
12 ## Tech Stack
13 - Use the already activated cos333 Python virtual environment, which uses
   Python 3.14.
14 - Install into the cos333 environment when necessary.
15 - Use Flask and Jinja2.
16 - Use the sqlite3 driver, not use sqlalchemy or flask-sqlalchemy.
17
18 ## Coding Standards
19 - Use PEP 8 coding standards.
20 - Omit Python type annotations.
21 - Limit source code line lengths to 72 characters.
22
23 ## Testing
24 - Use simple Python code, not unittest or pytest.
25
```

The Explorer sidebar on the left shows the project structure, including files like `home.html`, `searchform.html`, `searchresults.html`, `database.py`, `penny.sqlite`, and `runserver.py`.

The Chat sidebar on the right shows a conversation with Copilot. The task is "Create '/' endpoint and testserver.py". The chat history includes a review of `runserver.py` and the creation of `testdatabase.py`. The current prompt in the input area is "Describe what to build", which is highlighted by a red box and an arrow pointing to it from the text "Copilot prompt input area".

Copilot in VS Code

The screenshot displays the VS Code interface with the Copilot LLM selector and chat window. The main editor shows a file named `copilot-instructions.md` with the following content:

```
1 # Penny App Guidelines
2
3 ## Role
4 - Act as a senior software engineer.
5
6 ## Context:
7 - Create a three-tier web application named Penny in a file named
  runserver.py.
8 - The application must allow users to query a SQLite database.
9 - The database is stored in a provided binary file named penny.sqlite.
10 - The application must consist of three endpoints, each of which
   generates a distinct HTML page.
11
12 ## Tech Stack
13 - Use the already activated cos333 Python virtual environment, which uses
   Python 3.14.
14 - Install into the cos333 environment when necessary.
15 - Use Flask and Jinja2.
16 - Use the sqlite3 driver, not use sqlalchemy or flask-sqlalchemy.
17
18 ## Coding Standards
19 - Use PEP 8 coding standards.
20 - Omit Python type annotations.
21 - Limit source code line lengths to 72 characters.
22
23 ## Testing
24 - Use simple Python code, not unittest or pytest.
25
```

The Chat window on the right shows a task: "Create '/' endpoint and testserver.py". The task description is: "Factor the code, separating, factor database access out of runserver.py into a new file named database.py. As a result, the choice of database should be hidden from runserver.py. The runserver.py file should be unaffected by changing the database from SQLite to PostgreSQL or to MongoDB." The review section states: "Create a file named testdatabase.py to test the database.py file. Run testdatabase.py to make sure that database.py works. Also run testserver.py to make sure that runserver.py still works." The chat history shows that the task was completed, and the database code was factored into `database.py`.

A red arrow points to the "Auto" button in the Copilot LLM selector, which is highlighted with a red box. The text "Copilot LLM selector" is written in red next to the arrow.

Copilot in VS Code

The screenshot displays the VS Code interface with the Copilot extension. The Explorer sidebar on the left shows a project named 'PennyCopilot7' with files like 'copilot-instructions.md', 'home.html', 'searchform.html', 'searchresults.html', 'database.py', 'penny.sqlite', 'prompt1.txt', 'prompt2.txt', 'prompt3.txt', 'prompt4.txt', 'prompt5.txt', 'prompt6.txt', 'runserver.py', 'testdatabase.py', and 'testserver.py'. The main editor shows the 'copilot-instructions.md' file with the following content:

```
1 # Penny App Guidelines
2
3 ## Role
4 - Act as a senior software engineer.
5
6 ## Context:
7 - Create a three-tier web application named Penny in a file named
  runserver.py.
8 - The application must allow users to query a SQLite database.
9 - The database is stored in a provided binary file named penny.sqlite.
10 - The application must consist of three endpoints, each of which
   generates a distinct HTML page.
11
12 ## Tech Stack
13 - Use the already activated cos333 Python virtual environment, which uses
   Python 3.14.
14 - Install into the cos333 environment when necessary.
15 - Use Flask and Jinja2.
16 - Use the sqlite3 driver, not use sqlalchemy or flask-sqlalchemy.
17
18 ## Coding Standards
19 - Use PEP 8 coding standards.
20 - Omit Python type annotations.
21 - Limit source code line lengths to 72 characters.
22
23 ## Testing
24 - Use simple Python code, not unittest or pytest.
25
```

The Chat sidebar on the right shows a conversation with Copilot. The task is 'Create "/" endpoint and testserver.py'. The chat history includes a review of 'runserver.py' and the creation of 'testdatabase.py'. The current status of the chat is 'Completed: Factor database code into database.py (2/4)'. A red arrow points from the text 'Copilot status (credits used)' to the status bar at the bottom right, which shows 'Ln 25, Col 1 Spaces: 2 UTF-8 LF {} Instruct...' and a red box around the Copilot icon.

Copilot status
(credits used)

Copilot in VS Code

- See me or a TA if you have setup questions

Aside

- **Microsoft Copilot** for Princeton (on the Web)
 - “Princeton Copilot”
 - <https://m365.cloud.microsoft/chat>
 - Alternative to ChatGPT
 - Underlying model: GPT-5

Aside

- *Microsoft Copilot* for Princeton (on the Web)
(cont.)
 - (pro) Readily available with Princeton login
 - (pro) Provides **privacy**
 - (pro) Essentially unlimited free tier
 - (pro) No setup
 - (con) (In my experience) not as good as ChatGPT

Aside

- **Microsoft Copilot** for Princeton (on the Web)
(cont.)
 - Don't use for coding
 - Not integrated into an IDE
 - Chat, but no dynamic code completion
 - Must upload local files
 - Must download generated code to local file
 - Minimal run-time environment

Agenda

- Generative AI for SW engineering
- AI programming tools
- Prompt heuristics
- Copilot in VS Code
- **Example**
- Personal observations
- AI in COS 333

Example

- To **learn** AI programming you must **do** AI programming
- And so...

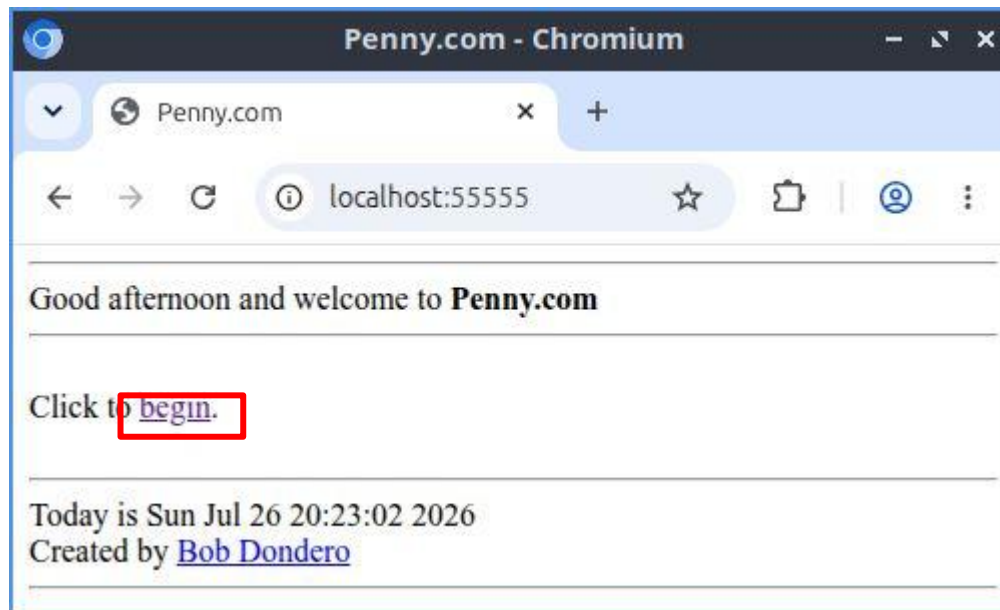
Example

- Desired:

```
$ python runserver.py 55555
* Serving Flask app 'penny'
  * Debug mode: on
WARNING: This is a development server. Do
not use it in a production deployment. Use a
production WSGI server instead.
  * Running on all addresses (0.0.0.0)
  * Running on http://127.0.0.1:55556
  * Running on http://10.8.19.130:55556
Press CTRL+C to quit
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: 118-894-031
```

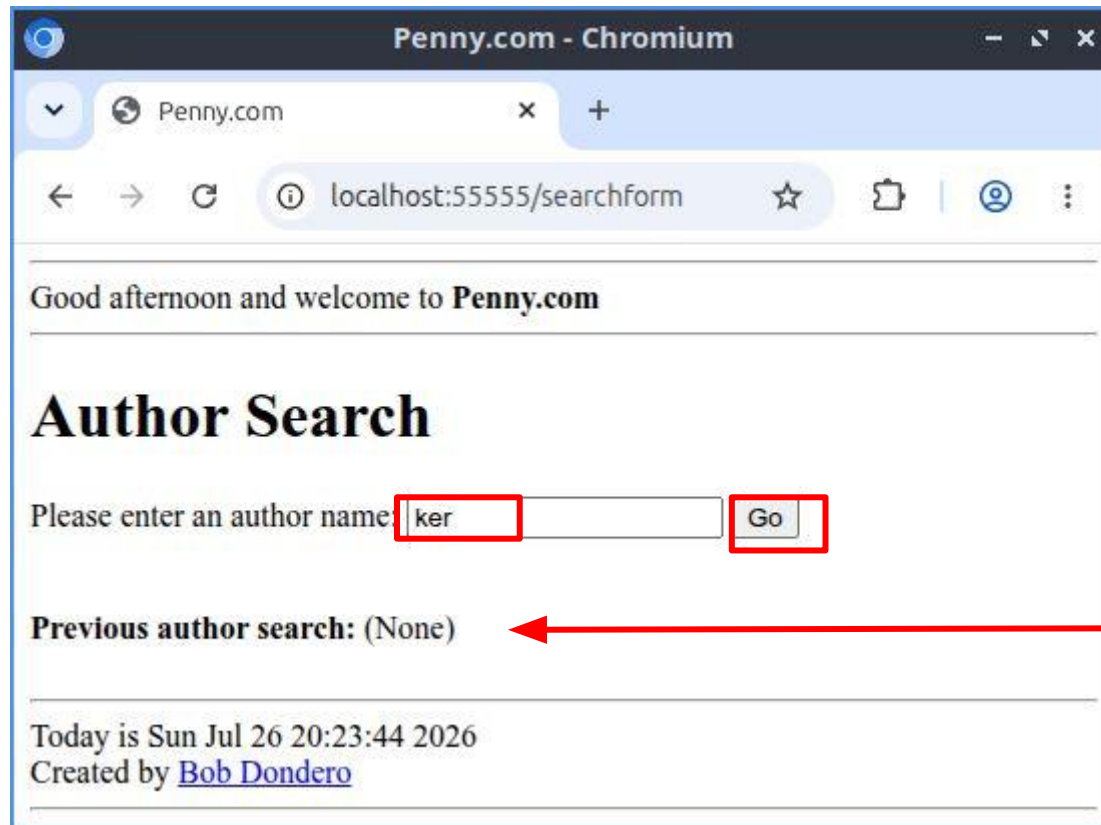
Example

- Desired:



Example

- Desired:



The screenshot shows a web browser window titled "Penny.com - Chromium". The address bar displays "localhost:55555/searchform". The page content includes a welcome message, a search form titled "Author Search", and a status bar at the bottom.

Good afternoon and welcome to **Penny.com**

Author Search

Please enter an author name:

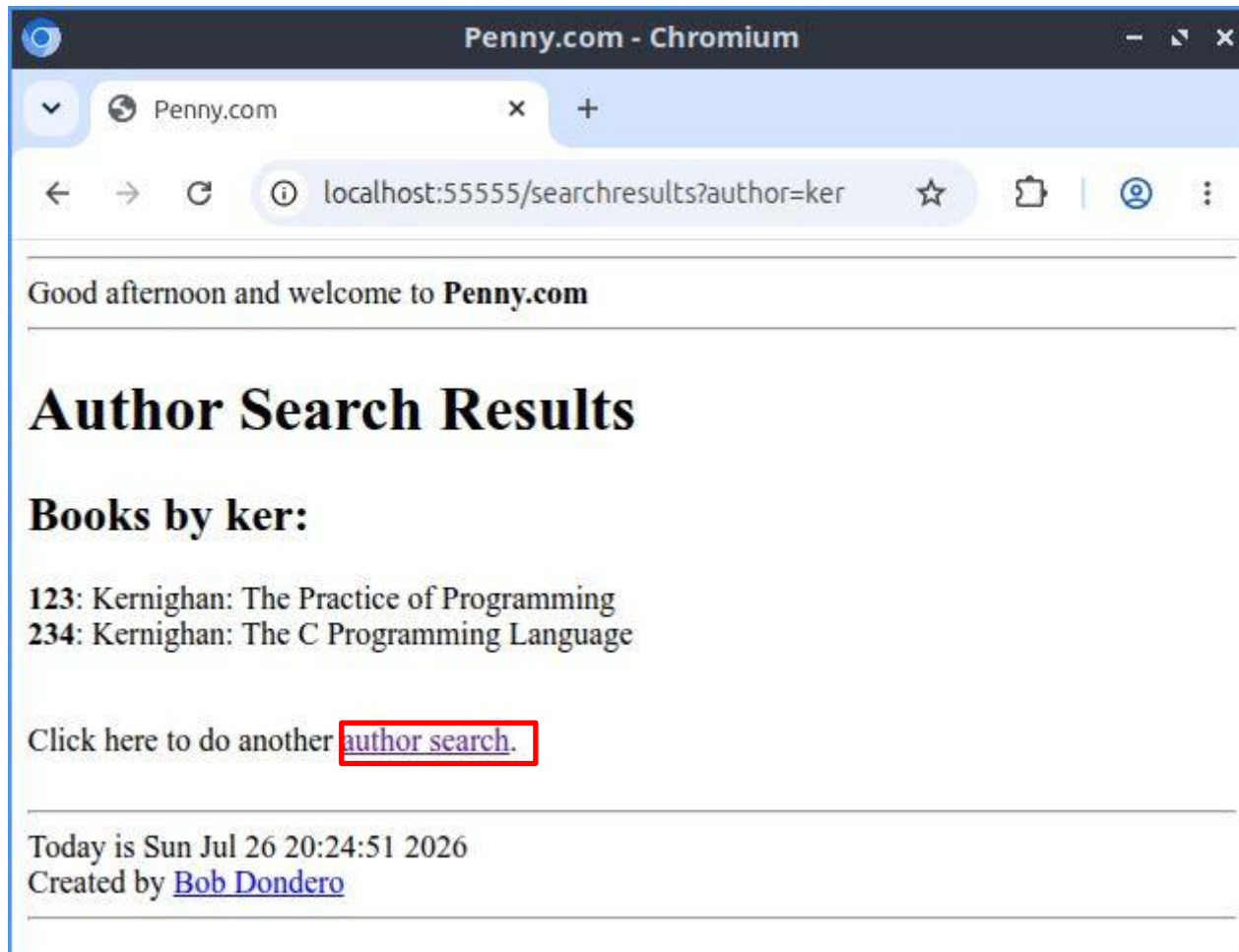
Previous author search: (None)

Today is Sun Jul 26 20:23:44 2026
Created by [Bob Dondero](#)

Note

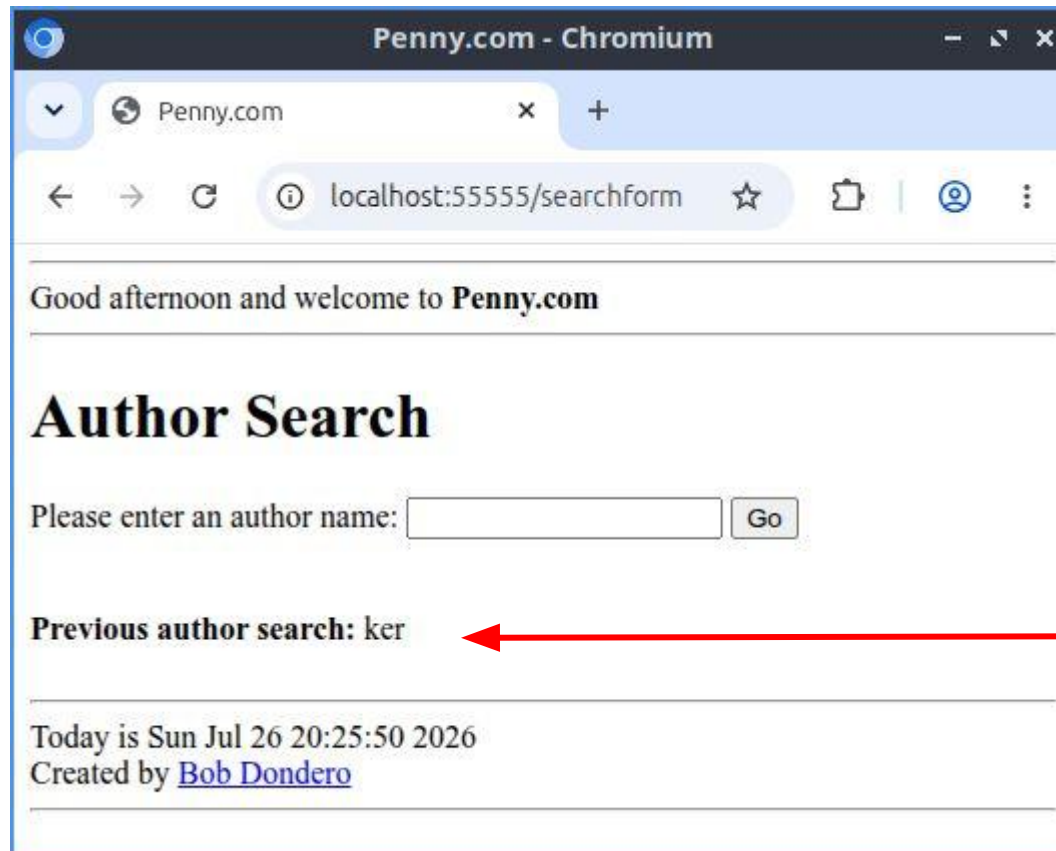
Example

- Desired:



Example

- Desired:



← Note

Example

- Assume we've created penny.sqlite, but no other files

Example

- Preliminary
 - In file `.github/copilot-instructions.md`

```
# Penny App Guidelines
```

```
## Role
```

```
- Act as a senior software engineer.
```

```
## Context:
```

```
- Create a three-tier web application named Penny in a file named  
runserver.py.
```

```
- The application must allow users to query a SQLite database.
```

```
- The database is stored in a provided binary file named  
penny.sqlite.
```

```
- The application must consist of three endpoints, each of which  
generates a distinct HTML page.
```

Example

- Preliminary

- In file .github/copilot-instructions.md (cont.)

Tech Stack

- Use the already activated cos333 Python virtual environment, which uses Python 3.14.
- Install into the cos333 environment when necessary.
- Use Flask and Jinja2.
- Use the sqlite3 driver, not use sqlalchemy or flask-sqlalchemy.

Coding Standards

- Use PEP 8 coding standards.
- Omit Python type annotations.
- Limit source code line lengths to 72 characters.

Testing

- Use simple Python code, not unittest or pytest.

Aside: Markdown and AI

- Markdown
 - <https://en.wikipedia.org/wiki/Markdown>
- GenAI and markdown (.md files) go together well
 - Structured yet readable
 - Concise; makes efficient use of tokens (vs. HTML, XML, ...)

<https://markdown-tools.com/markdown-ai-agents>

retrieved August 15, 2026

Question (11ai1a)

- Suppose you intend to use generative AI to help you develop the Penny app as shown. What is your first prompt?
 - Browse to <https://cos333attend.cs.princeton.edu> to answer

[1 point]

Example

- [Live demo]
- Repeat until finished:
 - Issue prompt
 - Watch Copilot work; interact as required
 - Study the resulting “production” code
 - Run the resulting “production” code
 - (If time) Study the resulting test code
 - (If time) Run the resulting test code

Example

- Example retrospective
 - Copilot consistent errors:
 - Failure to close DB connection
 - `with` ends transactions
 - `with` doesn't close connections
 - Failure to close DB cursor

Example

- Example retrospective (cont.)
 - Copilot variations (prior to creating the copilot-instructions.md file):
 - Create a new Python virtual env?
 - Use unittest?

Example

- Example retrospective (cont.)
 - Copilot variations (after creating the copilot-instructions.md file):
 - Transient syntax/indentation errors?
 - Specify the DB connection mode?
 - Instead of a “previous author search” cookie:
 - Use a global variable?
 - Use a Flask field?
 - Use a Flask session?
 - Create DB cursor?
 - Jinja2 templates external to runserver.py or internal?
 - Use CSS?

Example

- Example retrospective (cont.)
 - Generated code was not better over time

Agenda

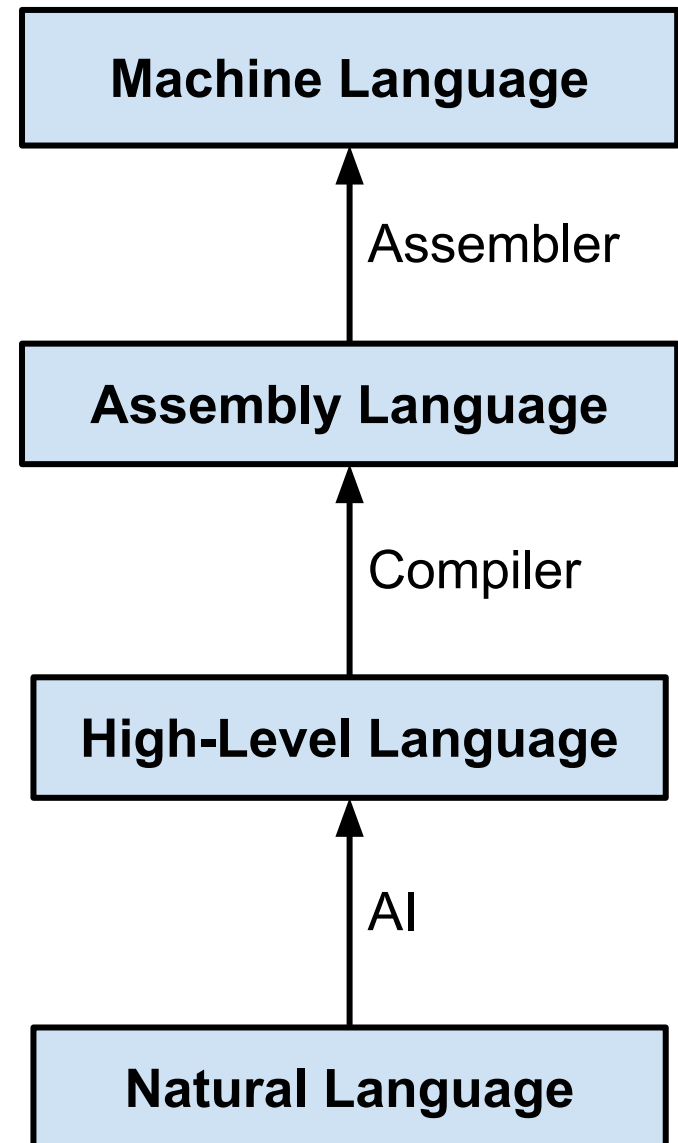
- Generative AI for SW engineering
- AI programming tools
- Prompt heuristics
- Copilot in VS Code
- Example
- **Personal observations**
- AI in COS 333

Personal Observations

- AI programming:
 - Shifts emphasis from **writing** code to **reading** code
 - Shifts emphasis from writing **code** to writing **specs**

Personal Observations

- AI programming:
 - Is **similar to** programming at a higher level of abstraction
 - Is **different from** programming at a higher level of abstraction



Aside: Vibe Coding

- *Vibe coding*

Vibe coding describes a chatbot-based approach to creating software where the **developer describes a project or task to a large language model (LLM)**, which generates code based on the prompt.

The developer does not review or edit the code, but solely uses tools and execution results to evaluate it and asks the LLM for improvements.

Unlike traditional AI-assisted coding or pair programming, **the human developer avoids examination of the code, accepts AI-suggested completions without human review, and focuses more on iterative experimentation than code correctness or structure.**

— https://en.wikipedia.org/wiki/Vibe_coding

Personal Observations

- AI programming:
 - Often produces code quickly
 - Often produces **usable** code not so quickly

Personal Observations

- AI programming:
 - Is **eliminating** some SWE jobs
 - Is **enhancing** some SWE jobs
 - Is **creating** some SWE jobs

Personal Observations

- **Very** personal observation...
- AI programming:
 - Isn't as much **fun**!

Agenda

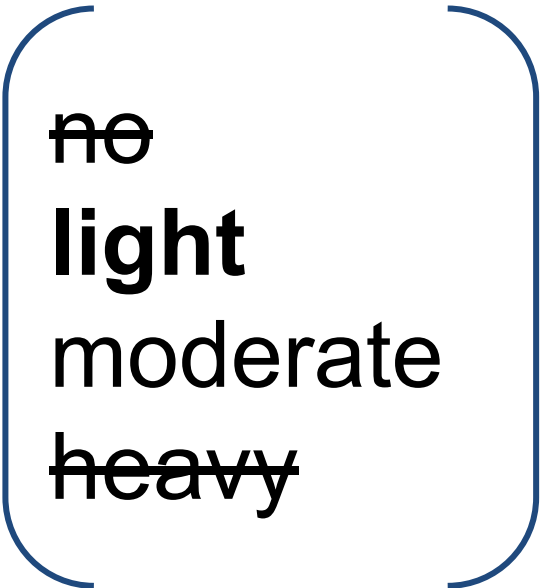
- Generative AI for SW engineering
- AI programming tools
- Prompt heuristics
- Copilot in VS Code
- Example
- Personal observations
- **AI in COS 333**

AI in COS 333

- For the **assignments**
 - You must not use AI
- For your **project**
 - Your **app** need not use AI
 - **You** must use AI
 - You must use AI to help you develop features of your project app
 - Specifically...

AI in COS 333

- For your **project**, you must:

Make  use of AI

- ~~no~~
- light**
- moderate
- ~~heavy~~

AI in COS 333

- For your **project**, you must:
 - **Tell your TA adviser** on a weekly basis when you get help from AI
 - **Understand** all of the code that you develop for your project, whether you develop it with or without the help of AI

Summary

- The lecture has covered:
 - How to use generative AI for programming