

Lesson 04 Demo 03

Analyzing AI Responses to Different Prompt Styles

Objective: To analyze how different prompt styles influence AI-generated code responses and determine the most effective techniques for guiding AI assistance

Tools required: VS Code with GitHub Copilot

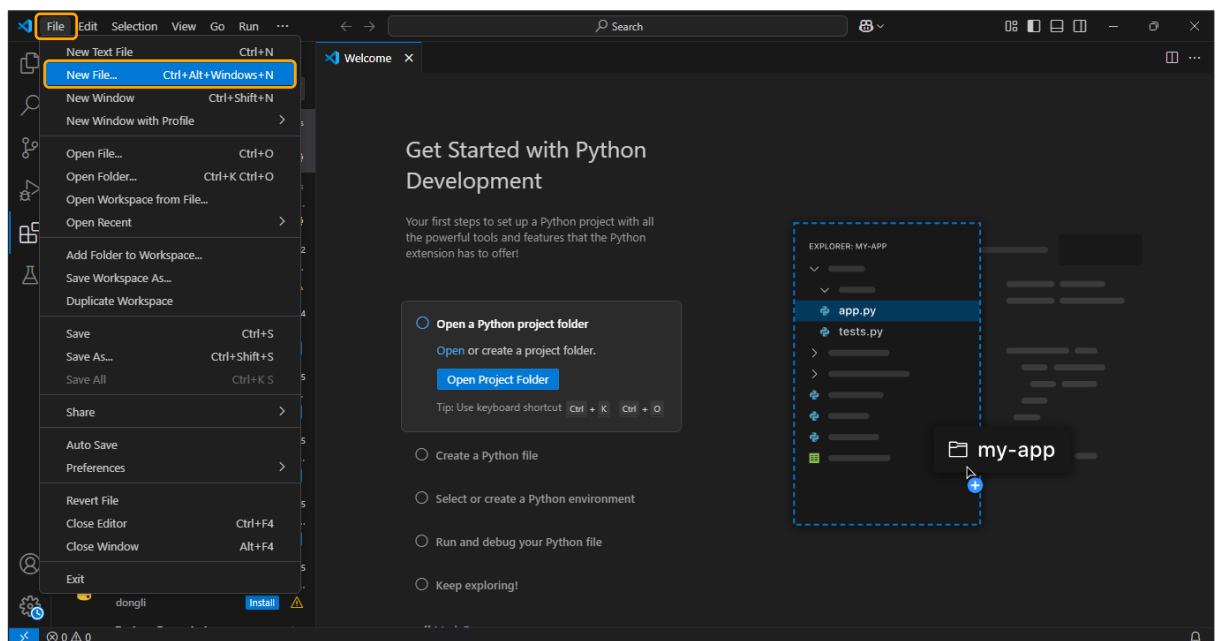
Prerequisites: None

Steps to be followed:

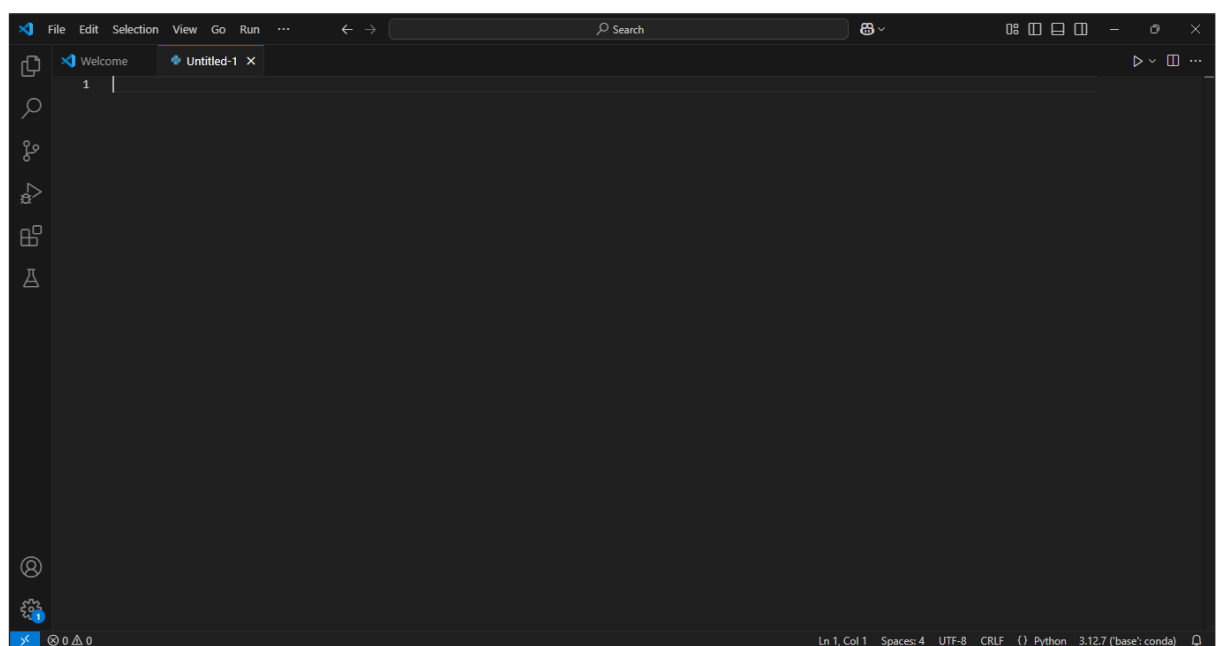
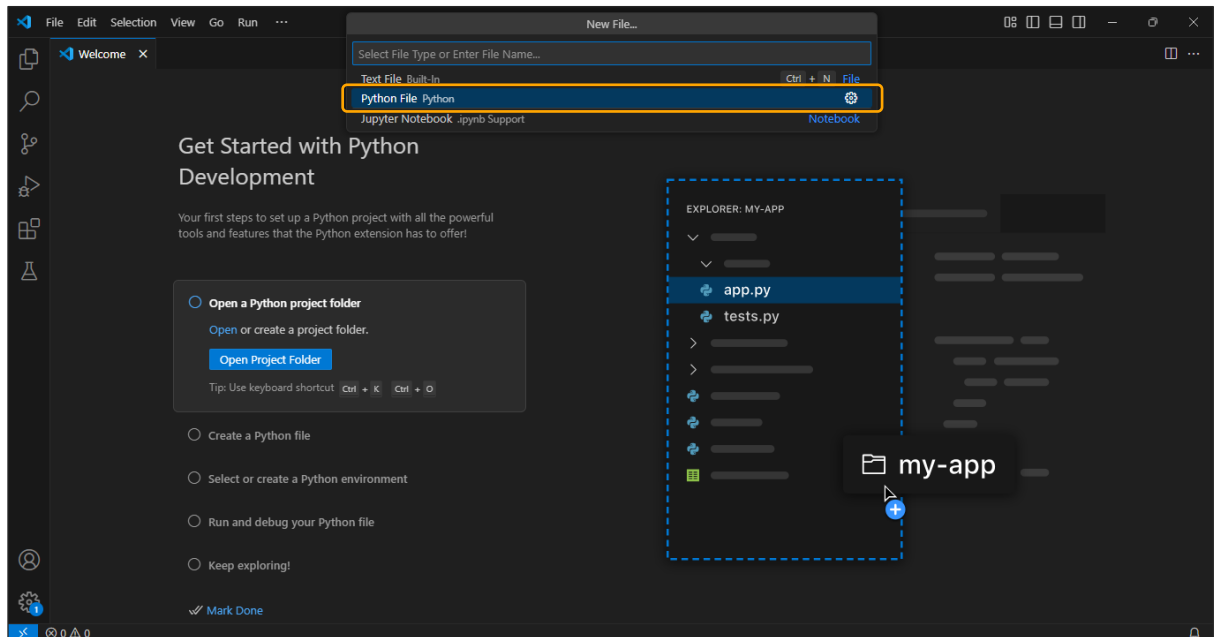
1. Launch VS Code and open a new file
2. Create a function using inline comments
3. Create a function using docstring-based prompts
4. Create a function using contextual hints
5. Create a function using partial code completion

Step 1: Launch VS Code and open a new file

1.1 Launch VS Code and click on **File** and then **New File**



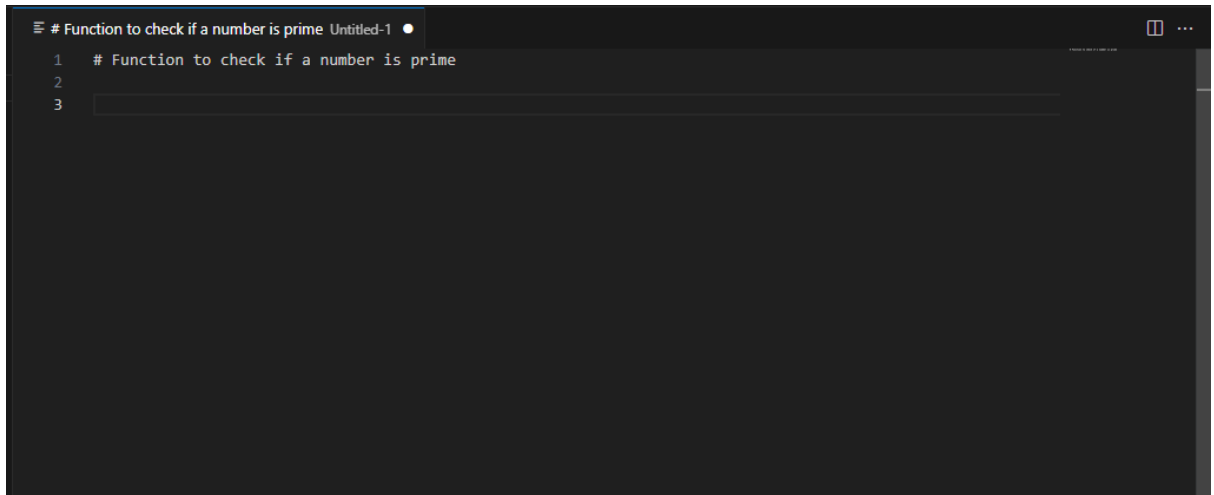
1.2 Select the **Python File** option from the name bar on top and a new Python file named Untitled-1 will open




Step 2: Create a function using inline comments

2.1 Type the following comment before defining a function and press enter to get suggestions:

Function to check if a number is prime

A screenshot of a code editor window titled "# Function to check if a number is prime Untitled-1". The editor shows three lines of code: line 1 is "# Function to check if a number is prime", line 2 is empty, and line 3 is an empty line with a cursor. The editor has a dark theme and a sidebar on the right.

2.2 Pause and observe how Copilot suggests the function's body:

A screenshot of the same code editor window, now showing the AI-suggested function body. The code is as follows:

```
1 # Function to check if a number is prime
2 def is_prime(n):
3     if n <= 1:
4         return False
5     for i in range(2, int(n**0.5) + 1):
6         if n % i == 0:
7             return False
8     return True
9
10
```

The code is color-coded: comments are green, function definitions are blue, and other code is purple. The editor has a dark theme and a sidebar on the right.

Note: You can press **tab** on your keyboard to accept the AI-suggested code or you can modify and run it with test values.

Step 3: Create a function using docstring-based prompts

- 3.1 Clear the previous function, type a new one with a structured docstring, and press enter to get suggestions

Example input:

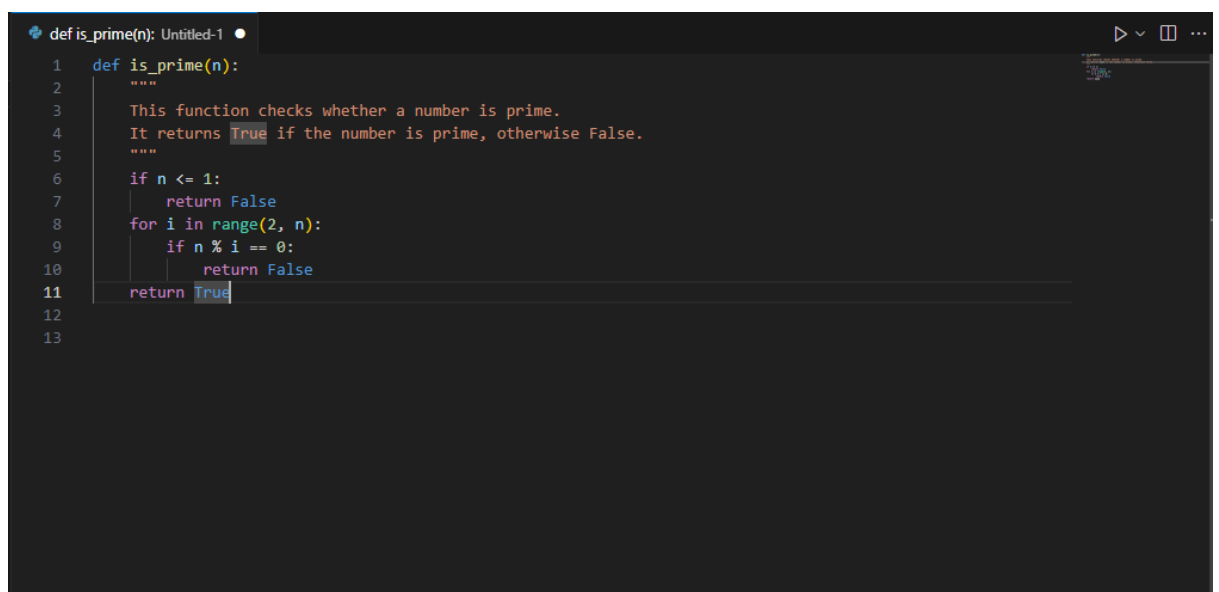
```
def is_prime(n):  
    """  
    This function checks whether a number is prime.  
    It returns True if the number is prime, otherwise False.  
    """
```



A screenshot of a code editor window titled 'def is_prime(n): Untitled-1'. The editor shows the following code:

```
1 def is_prime(n):  
2     """  
3     This function checks whether a number is prime.  
4     It returns True if the number is prime, otherwise False.  
5     """  
6  
7
```

- 3.2 Pause and observe how Copilot suggests the function's body



A screenshot of the same code editor window, now showing the function body suggested by Copilot. The code is as follows:

```
1 def is_prime(n):  
2     """  
3     This function checks whether a number is prime.  
4     It returns True if the number is prime, otherwise False.  
5     """  
6     if n <= 1:  
7         return False  
8     for i in range(2, n):  
9         if n % i == 0:  
10            return False  
11    return True  
12  
13
```

Note: Press tab to accept the function or modify and run it with test values

Step 4: Create a function using contextual hints

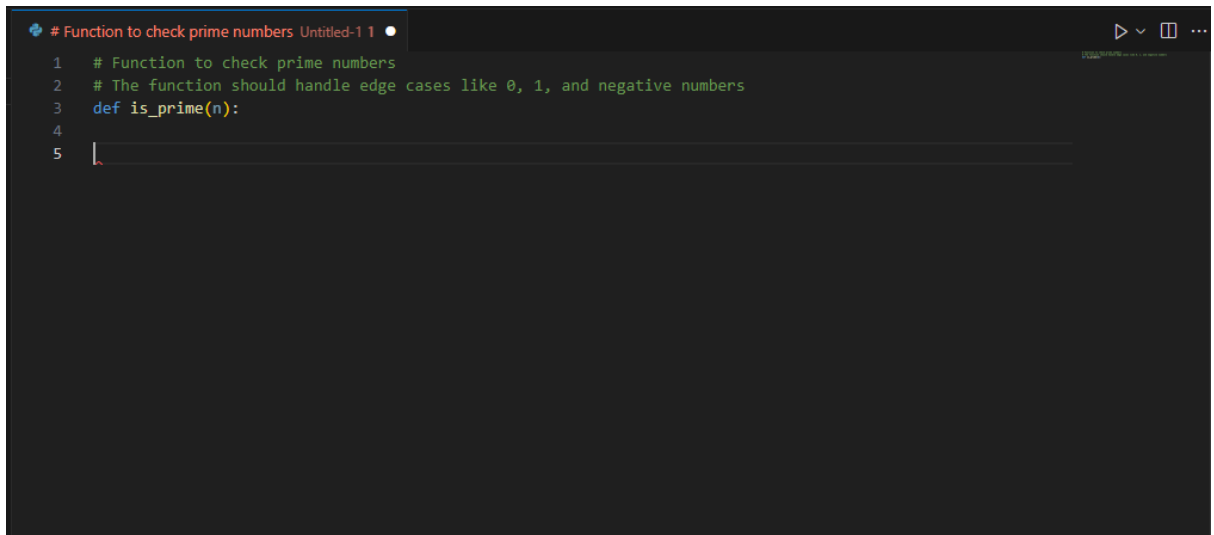
- 4.1 Delete the previous function, write a new one with more descriptive comments, and press enter to get suggestions:

Example input:

Function to check prime numbers

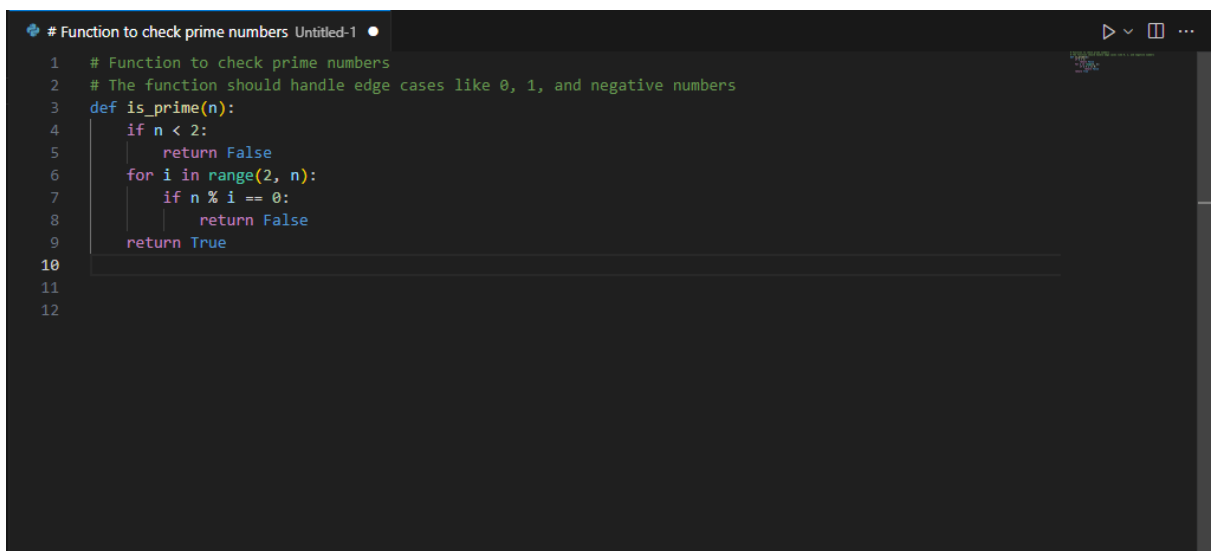
The function should handle edge cases like 0, 1, and negative numbers

def is_prime(n):



```
# Function to check prime numbers Untitled-1 1
1 # Function to check prime numbers
2 # The function should handle edge cases like 0, 1, and negative numbers
3 def is_prime(n):
4
5
```

- 4.2 Pause and observe how Copilot suggests the function's body



```
# Function to check prime numbers Untitled-1
1 # Function to check prime numbers
2 # The function should handle edge cases like 0, 1, and negative numbers
3 def is_prime(n):
4     if n < 2:
5         return False
6     for i in range(2, n):
7         if n % i == 0:
8             return False
9     return True
10
11
12
```

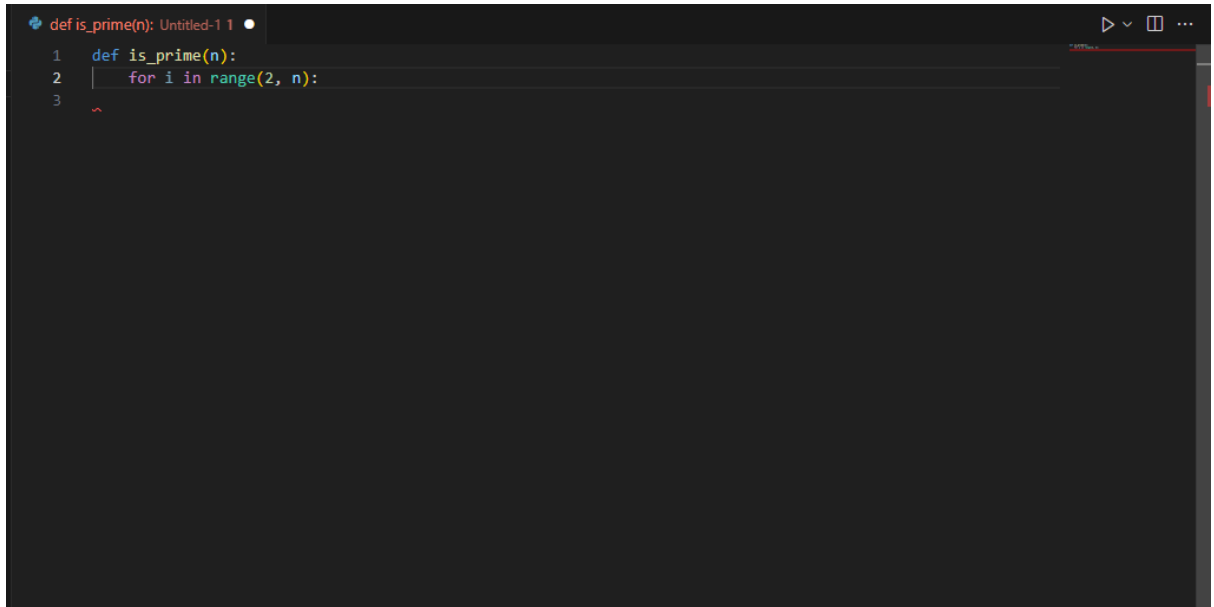
Note: Press tab to accept the function or modify and run it with test values

Step 5: Create a function using partial code completion

5.1 Type the function header and start the loop without completing it

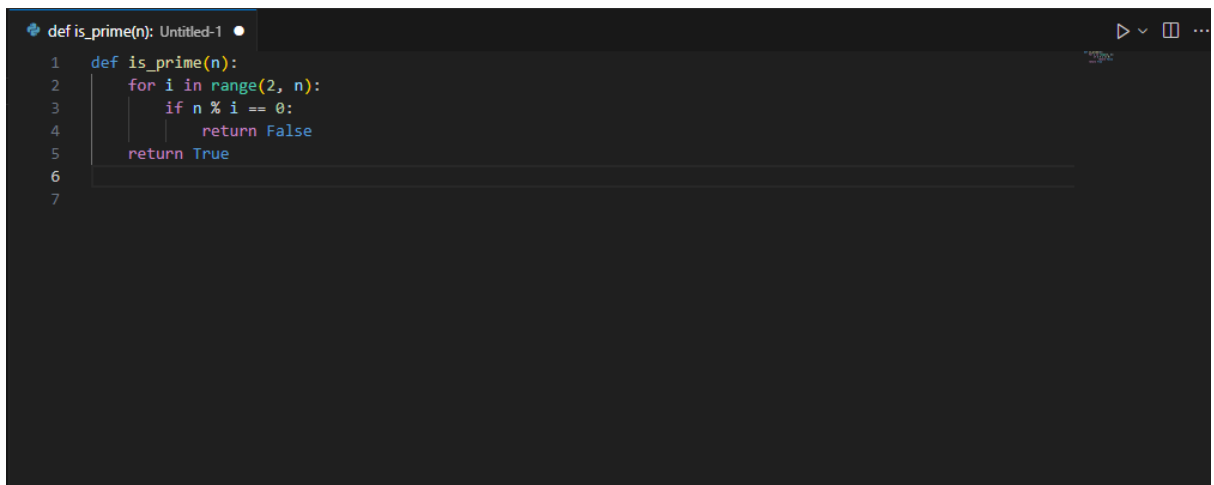
Example input:

```
def is_prime(n):  
    for i in range(2, n):
```



A screenshot of a code editor window titled 'def is_prime(n): Untitled-1'. The editor shows the following code:
1 def is_prime(n):
2 for i in range(2, n):
3
The cursor is positioned at the end of line 3. The editor has a dark theme and a sidebar on the right.

5.2 Pause and observe how Copilot suggests the function's body



A screenshot of a code editor window titled 'def is_prime(n): Untitled-1'. The editor shows the following code:
1 def is_prime(n):
2 for i in range(2, n):
3 if n % i == 0:
4 return False
5 return True
6
7
The cursor is positioned at the end of line 7. The editor has a dark theme and a sidebar on the right.

Note: Press tab to accept the function or modify and run it with test values

By following these steps, you have successfully tested different prompt styles and understood that structured docstrings and contextual hints lead to more accurate and optimized AI-generated code, while inline comments and partial code provide only basic

completions. Experimenting with these styles helps developers optimize AI-assisted coding for better accuracy, readability, and maintainability.