

## 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

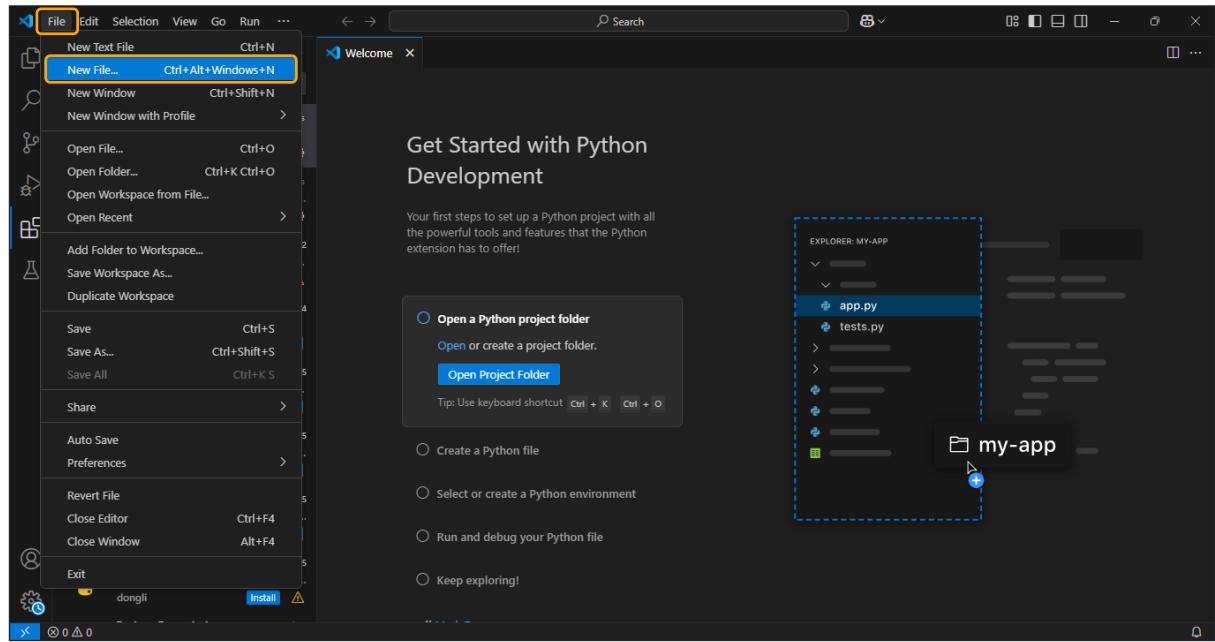
**Prerequisites:** None

#### Steps to be followed:

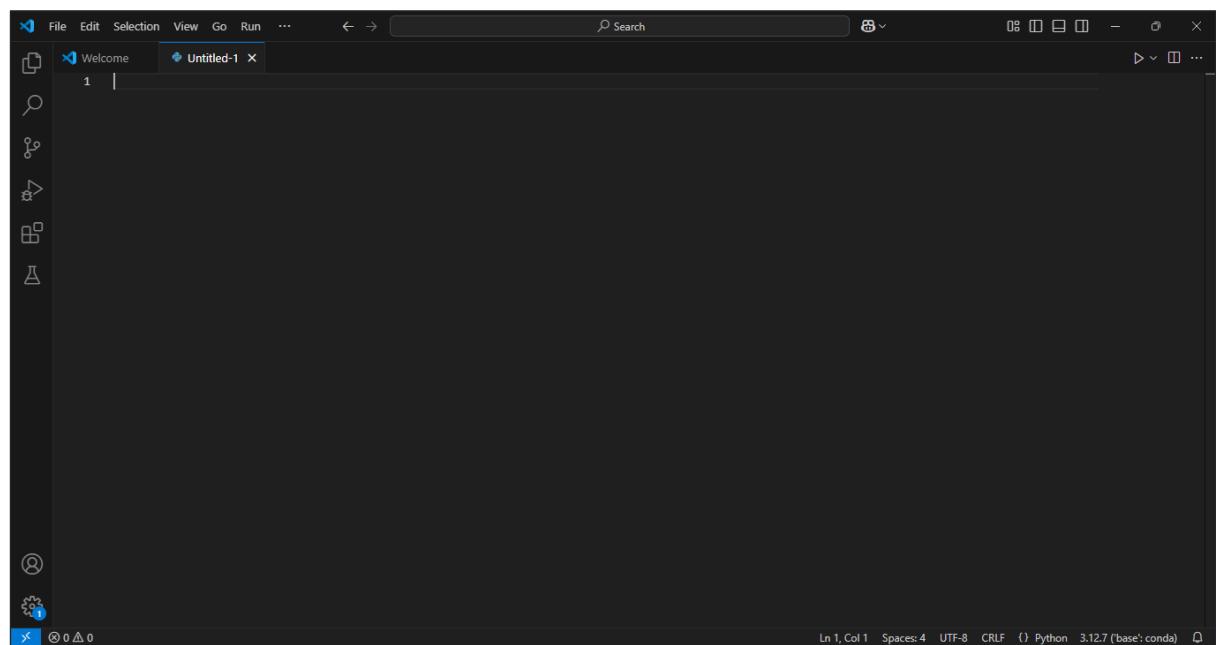
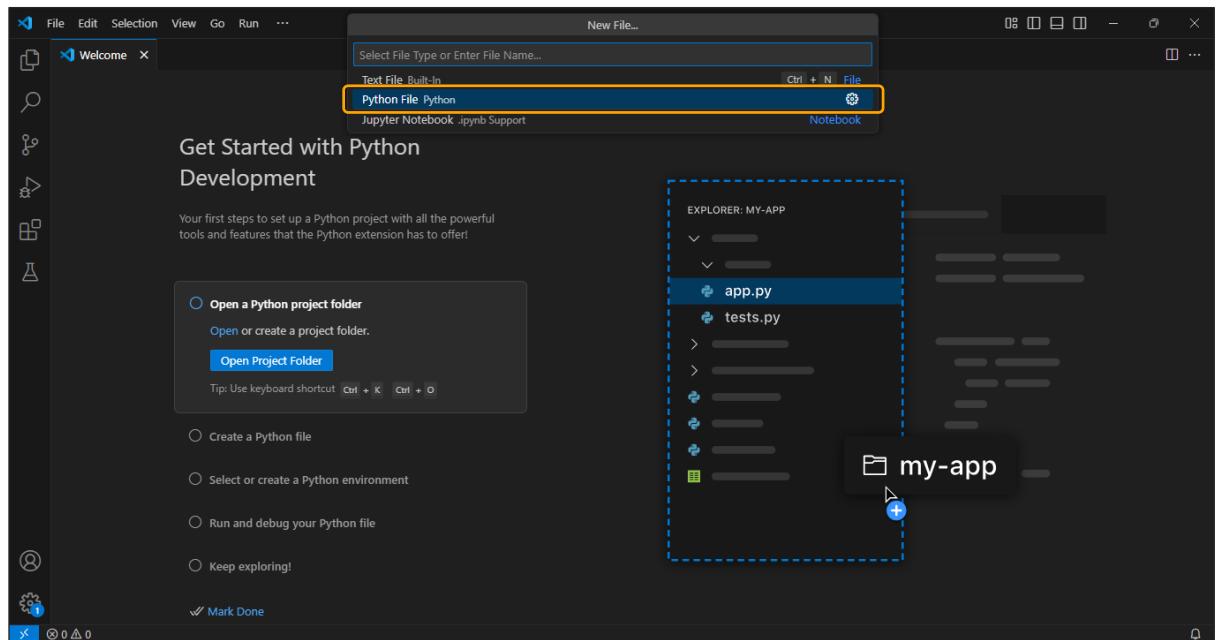
1. Launch VS Code and open a new file
2. Create a function using inline comments
3. Create a function using docstring-based prompts
4. Create a function using contextual hints
5. Create a function using partial code completion

#### Step 1: Launch VS Code and open a new file

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



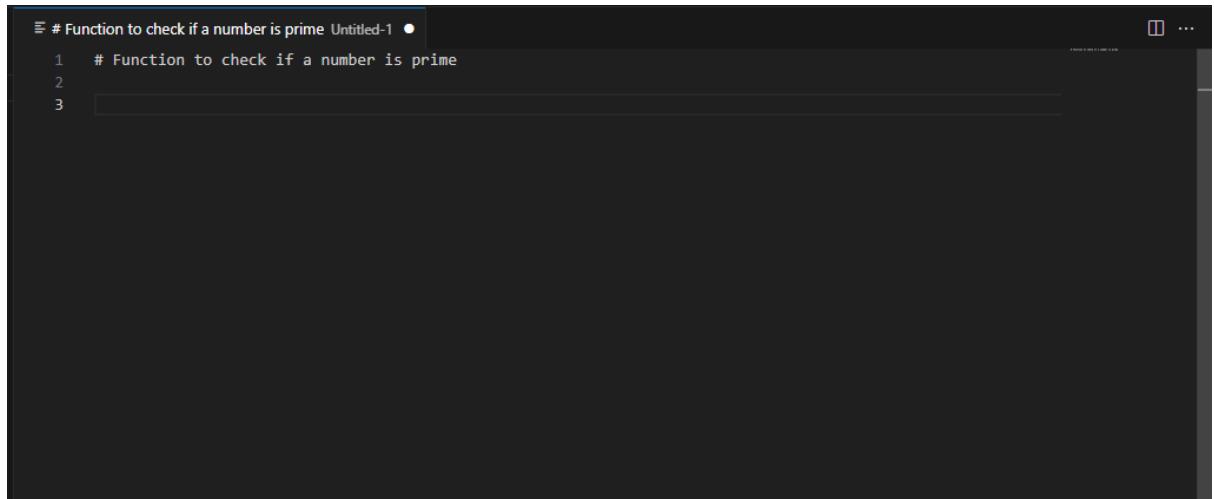
1.2 Select the **Python File** option from the name bar on top and a new Python file named Untitled-1 will open



## Step 2: Create a function using inline comments

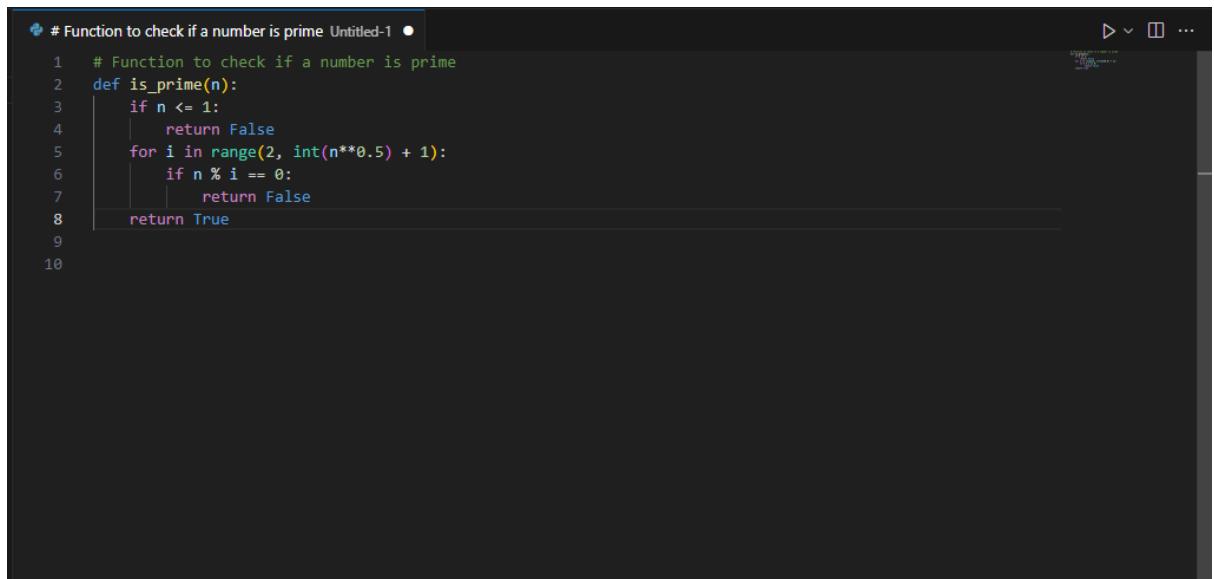
2.1 Type the following comment before defining a function and press enter to get suggestions:

```
# Function to check if a number is prime
```



A screenshot of a dark-themed code editor window. At the top, there is a title bar with the text "# Function to check if a number is prime Untitled-1". The main area contains three lines of code: "1 # Function to check if a number is prime", "2", and "3". A cursor is visible at the start of the second line.

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



A screenshot of a dark-themed code editor window. At the top, there is a title bar with the text "# Function to check if a number is prime Untitled-1". The main area contains ten lines of Python code. Lines 1 through 8 are user-defined, while lines 9 and 10 are AI-suggested. The AI-suggested code defines a function 'is\_prime' that checks if a number is prime by iterating from 2 to the square root of 'n' and checking for divisibility.

```
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
```

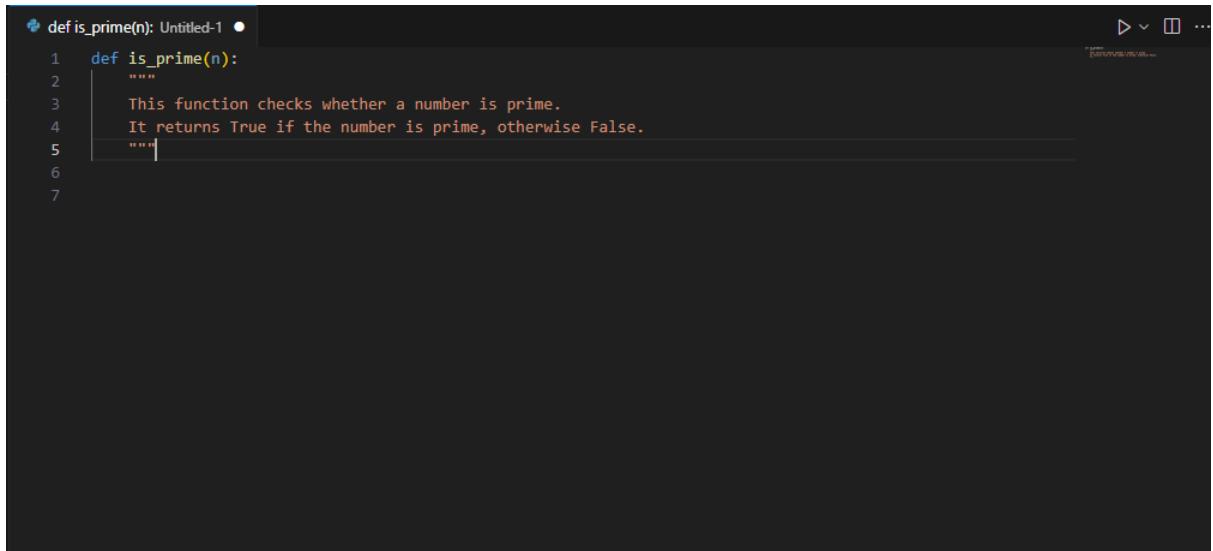
**Note:** You can press **tab** on your keyboard to accept the AI-suggested code or you can modify and run it with test values.

### 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:

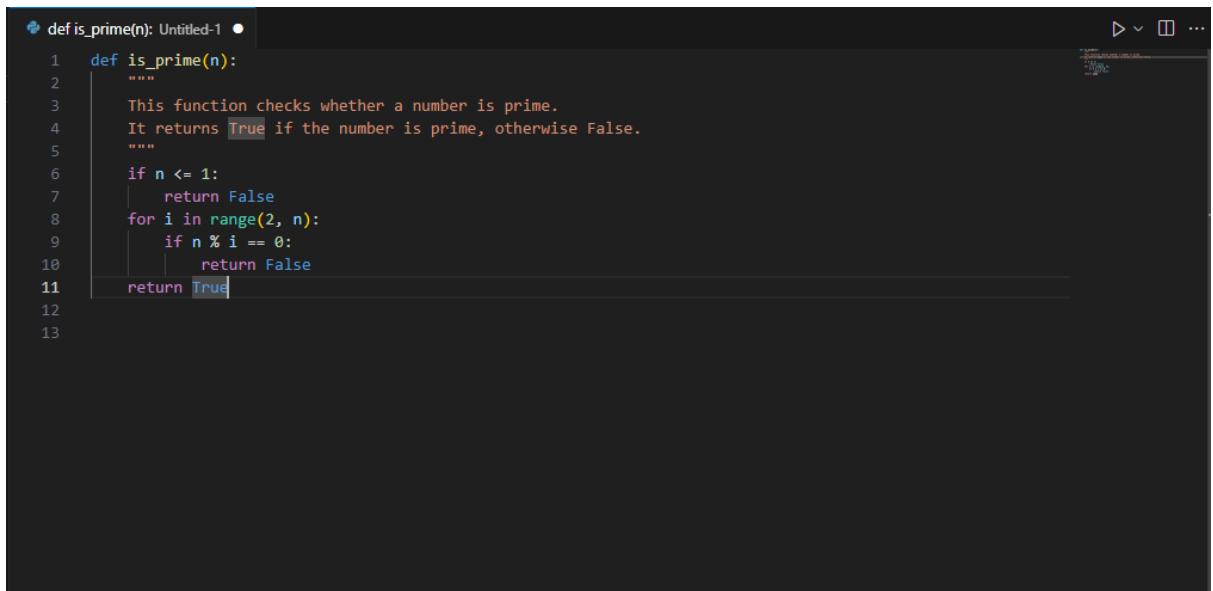
```
def is_prime(n):
    """
    This function checks whether a number is prime.
    It returns True if the number is prime, otherwise False.
    """
```



A screenshot of a code editor window titled "Untitled-1". The code defines a function "is\_prime" with a multi-line docstring. The docstring explains that the function checks if a number is prime and returns True or False. The code editor has a dark theme with syntax highlighting.

```
def is_prime(n): Untitled-1
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 a code editor window titled "Untitled-1". The code defines a function "is\_prime" with a docstring. Below the docstring, the function body is partially completed by Copilot, showing an if-else structure to check for numbers less than or equal to 1 and then iterating from 2 to n-1 to check for divisibility. The code editor has a dark theme with syntax highlighting.

```
def is_prime(n): Untitled-1
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     if n <= 1:
8         return False
9     for i in range(2, n):
10        if n % i == 0:
11            return False
12
13     return True
```

**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):
```

A screenshot of a code editor window titled "# Function to check prime numbers Untitled-1". The code is as follows:

```
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

A screenshot of a code editor window titled "# Function to check prime numbers Untitled-1". The code is as follows:

```
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 
```

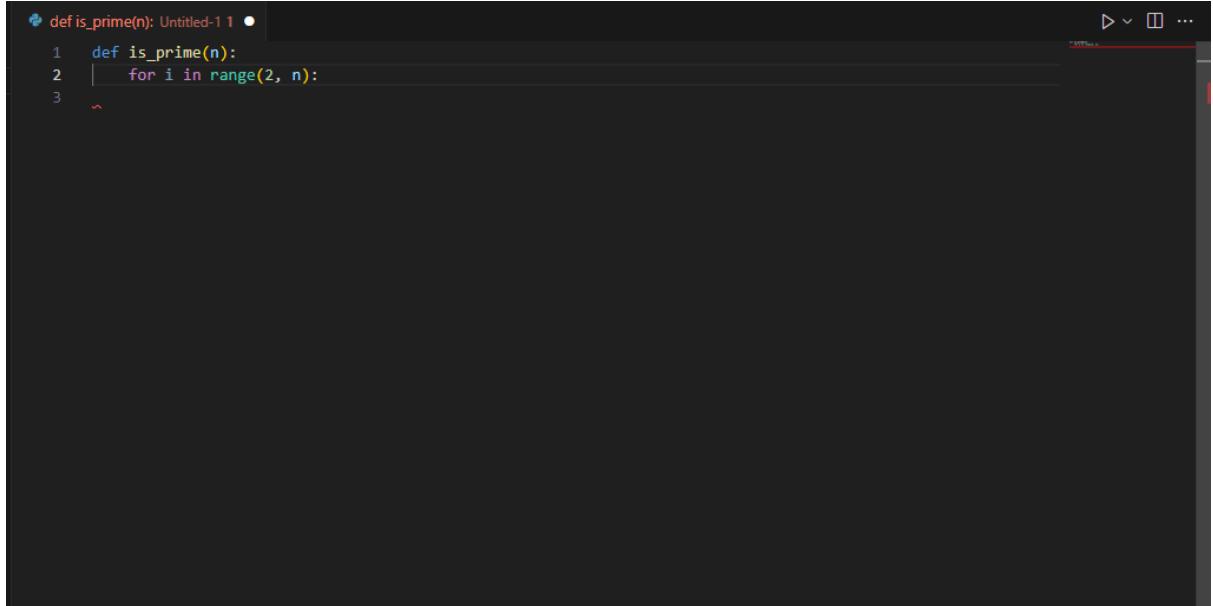
**Note:** Press tab to accept the function or modify and run it with test values

## Step 5: Create a function using partial code completion

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

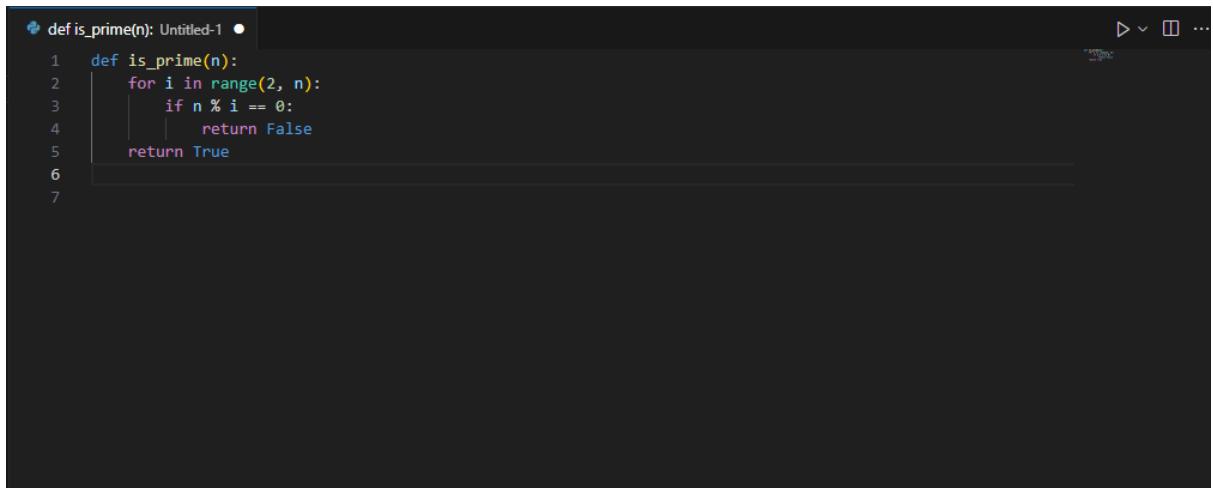
Example input:

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



A screenshot of a dark-themed code editor window. In the top left, there's a small icon with a blue dot and the text "Untitled-1". Below it, the code starts with "def is\_prime(n):". The next line, "for i in range(2, n):", is partially typed, with the closing brace of the for loop missing. The cursor is positioned at the end of the second line. The rest of the editor window is blank.

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



A screenshot of a dark-themed code editor window, identical to the previous one but with more completed code. The "is\_prime" function now includes a body: it loops through numbers from 2 to n-1, checking if any of them divide n evenly (n % i == 0). If any divisor is found, it immediately returns False. If no divisors are found after the loop, it returns True. The cursor is now at the end of the fifth line, after the final closing brace of the function.

**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.