Lab:3

Name:M.Ramsai

Roll Number: 2503A51L53

Course Code: 24CS002PC215

Course Title: AI Assisted Coding

Assignment Number: 3

Academic Year: 2025-2026

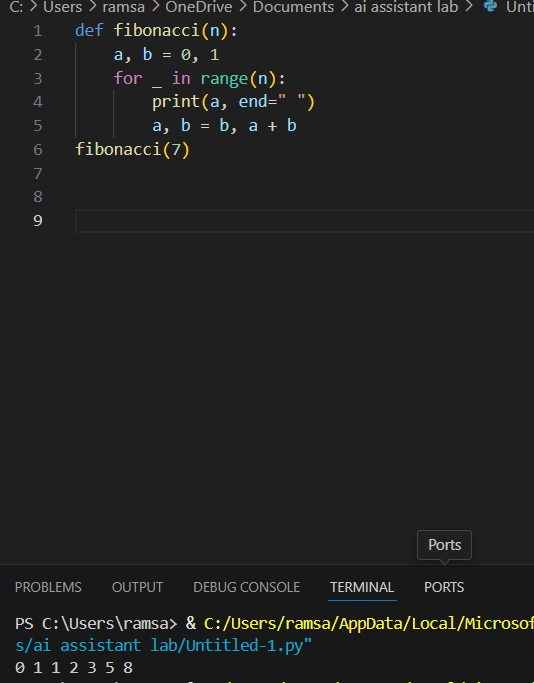
TASK1

* PROMPT GIVEN: Write a Python function that prints the first N Fibonacci numbers.
* Python function generated by Copilot without any example provided.

A screen shot of a computer

AI-generated content may be incorrect.

* output for sample input N = 7.



* Observation on how Copilot understood the instruction with zero context.

When provided with the single-line comment prompt, GitHub Copilot correctly interpreted the request and generated a complete Python function to print the first N Fibonacci numbers. Despite receiving no examples or contextual code, Copilot inferred the standard iterative approach, initialized the first two Fibonacci values (0 and 1), and used a loop to generate the sequence. For the test input N = 7, the function produced the expected output 0 1 1 2 3 5 8, demonstrating that Copilot understood the mathematical logic of the Fibonacci series purely from the natural language instruction.

**TASK2**

* PROMPT GIVEN: One-shot Prompt – List Reversal Function..

A computer screen shot of a list

AI-generated content may be incorrect.

* Output: [3, 2, 1] for input [1, 2, 3]

A screen shot of a computer

AI-generated content may be incorrect.

Observation on how adding a single example improved Copilot’s accuracy:

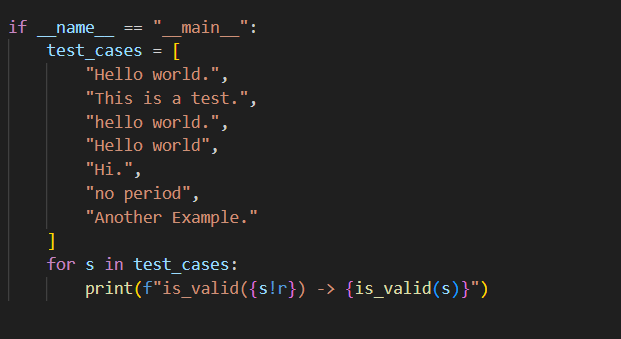
When a user provides even **one concrete example**, Copilot shifts from general guessing to **targeted generation**. That example acts as an anchor, helping Copilot:

* **Clarify ambiguity**: Abstract instructions like “reverse a list” could be interpreted in multiple ways. But when paired with an example like [1, 2, 3] → [3, 2, 1], Copilot instantly understands the expected behavior.
* **Match output format**: The example sets expectations for structure, style, and formatting—whether it’s code, text, or data.
* **Reduce error margin**: With a reference point, Copilot avoids common mistakes like reversing in-place when a new list is expected, or misinterpreting the data type.

**TASK3**

Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.

* THE CODE:



* OUTPUT:

A black screen with yellow and green text

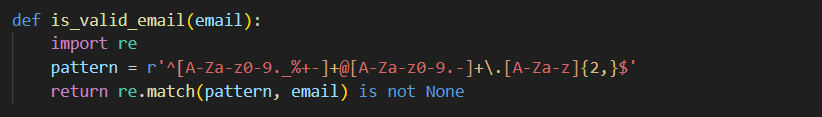
AI-generated content may be incorrect.

* **Students reflect on how multiple examples guide Copilot to generate more accurate code**.

1. By providing multiple examples, Copilot can better understand the intended pattern and logic for the function.
2. This helps Copilot distinguish between valid and invalid cases, resulting in more accurate code generation.
3. For instance, Copilot learns that the string must start with a capital letter and end with a period, and can avoid common mistakes.
4. The more examples given, the more reliable and precise Copilot's suggestions become..

**TASK 4**

* **First, prompt Copilot to write an email validation function using zero-shot (just the task in comment).**

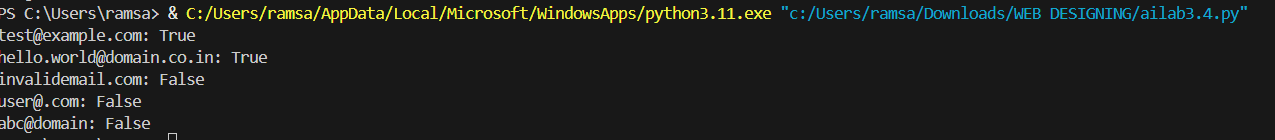
****

* **rewrite the prompt using few-shot examples.**

A computer screen with text and images

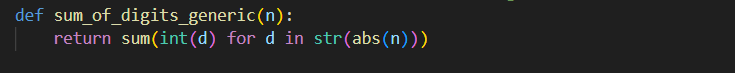
AI-generated content may be incorrect.

* **Output:**

****

**TASK 5**

* **Experiment with 2 different prompt styles to generate a function that returns the sum of digits of a number.**
* **Two versions of the sum\_of\_digits() function.**

**1.**

**2.**

**A screen shot of a computer code

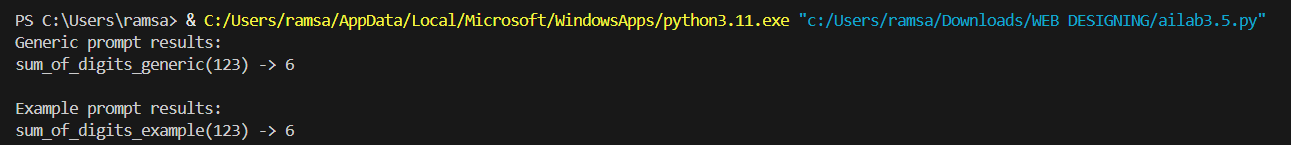
AI-generated content may be incorrect.**

* **Example Output: sum\_of\_digits(123) ➝ 6..**

**Program:**

**A computer screen with text

AI-generated content may be incorrect. Output:**

****

* **Short analysis: which prompt produced cleaner or more optimized code and why?**

1.The generic task prompt produced a concise and optimized solution using Python's built-in sum and generator expression:

2. return sum(int(d) for d in str(abs(n)))

3.This is both clean and efficient for this simple problem.

4.The input/output example prompt led to a more explicit, step-by-step approach using a loop and accumulator variable.

5.While both are correct, the generic prompt is more Pythonic and compact, whereas the example-based prompt is easier to understand for beginners.

6.For simple tasks, generic prompts often yield more optimized code. For complex logic, examples help Copilot generate more reliable and specific solutions.