ASSIGNMENT-3.4

HT.no:2303A51556

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Dr. Rishabh Mittal		
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju		
CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week2**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches
Assignment Number: 3.4 (Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and	Week2

Few-shot Techniques

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Task Description #1

• Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.

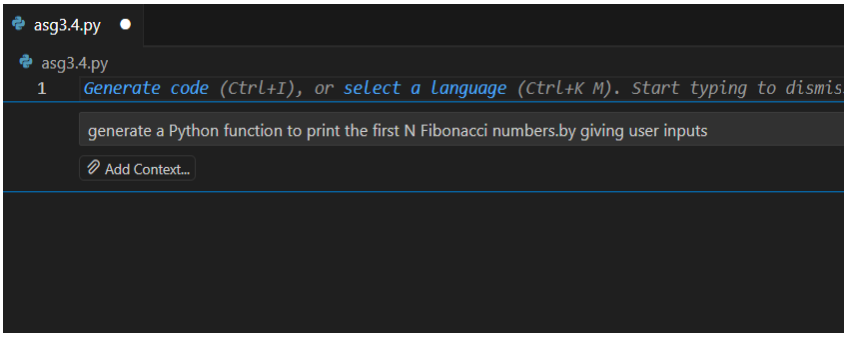
Expected Output #1

• A complete Python function generated by Copilot without any example provided.

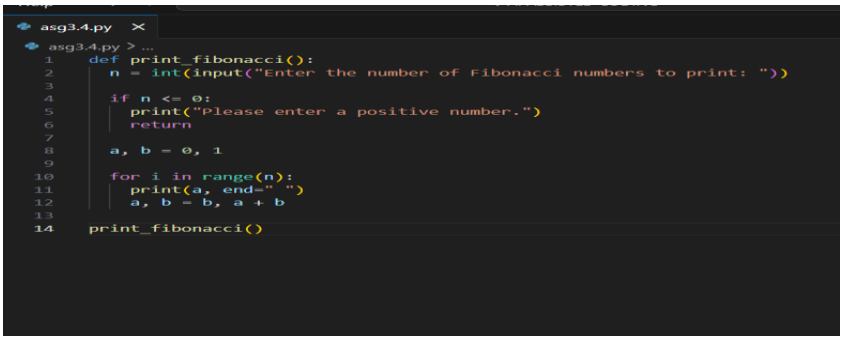
• Correct output for sample input N = 7 → 0 1 1 2 3 5 8

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

Prompt:



Code:



Output:

```
ANUSHA\AI ASSISTED CODING\asg3.4.py'
Enter the number of Fibonacci numbers to print: 7
0 1 1 2 3 5 8
```

Explanation:

Copilot understood the zero-shot instruction purely from the comment.It inferred the standard Fibonacci definition and produced a complete, working function.The solution uses tuple unpacking (a, b = b, a + b) and prints inline with end=" ".This shows Copilot's ability to reason from minimal context and generate correct code without examples.

Task 2: One-shot Prompt – List Reversal Function

Task Description #2

• Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

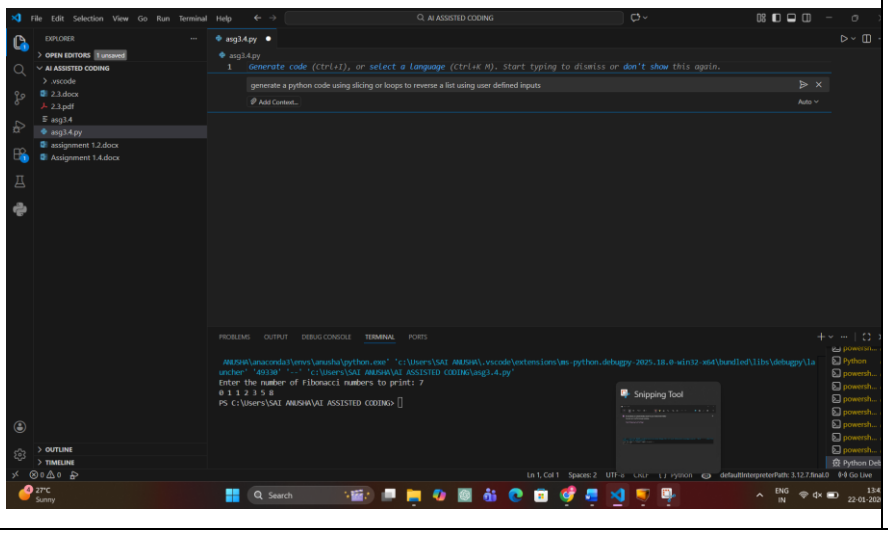
Expected Output #2

• Copilot-generated function to reverse a list using slicing or loop.

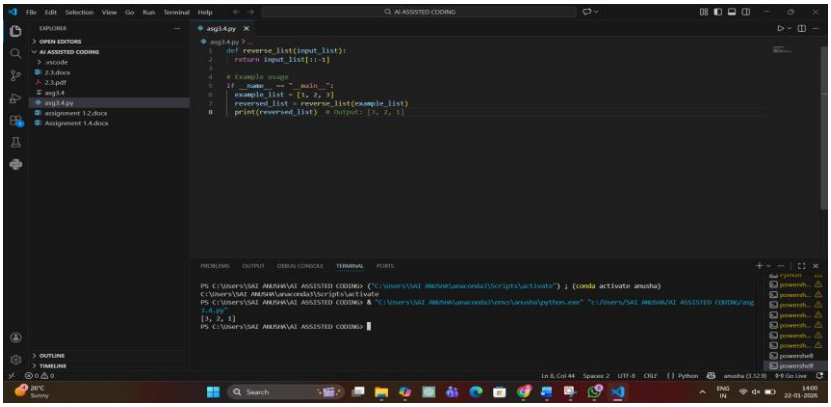
• Output: [3, 2, 1] for input [1, 2, 3]

• Observation on how adding a single example improved Copilot's accuracy.

Prompt:



Code and output:



Explanation:

With slicing, Copilot (or you) can generate a concise one-liner solution.With a loop, it demonstrates the step-by-step reversal logic, which is more explicit and useful for understanding how list traversal works.Both approaches are valid, and Copilot often defaults to slicing when given an example, since it's the most Pythonic.

Task 3: Few-shot Prompt – String Pattern Matching

Task Description #3

• 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.

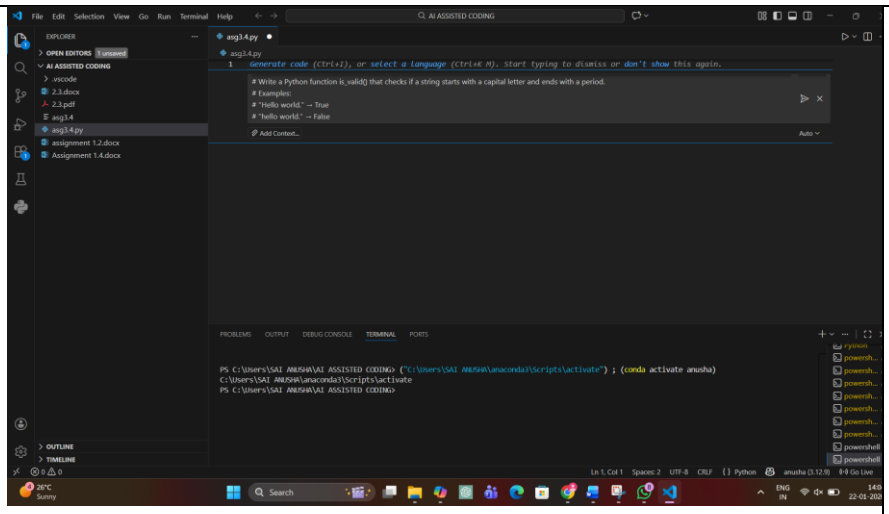
Expected Output #3

• A function is_valid() that checks the pattern.

• Output: True or False based on input.

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

Prompt:



Code:

```python
def is_valid(text):
    """
    Check if a string starts with a capital letter and ends with a period.
    Returns True or False.
    """
    text = text.strip()   # remove leading/trailing spaces

    if not text:          # handle empty input
        return False

    return text[0].isupper() and text.endswith('.')


# --- User-defined input (multiple sentences) ---
n = int(input("How many sentences do you want to check? "))

for i in range(n):
    user_text = input(f"Enter sentence {i+1}: ")
    if is_valid(user_text):
        print(f"Sentence {i+1} is VALID")
    else:
        print(f"Sentence {i+1} is NOT valid")
```

Output:

```
Enter sentence 1: hello world
Sentence 1 is NOT valid
```

Explanation:

Few-shot prompting helps Copilot understand a problem better by giving it a few clear examples along with the task description. Instead of only telling what to do, we show how the function should behave for different inputs. This makes the pattern more obvious, such as starting with a capital letter and ending with a period. With these examples, Copilot can learn the rules and generate more accurate and correct code. It also reduces

confusion and helps handle edge cases properly.

Task 4: Zero-shot vs Few-shot – Email Validator

Task Description #4

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

• Then, rewrite the prompt using few-shot examples.

Expected Output #4

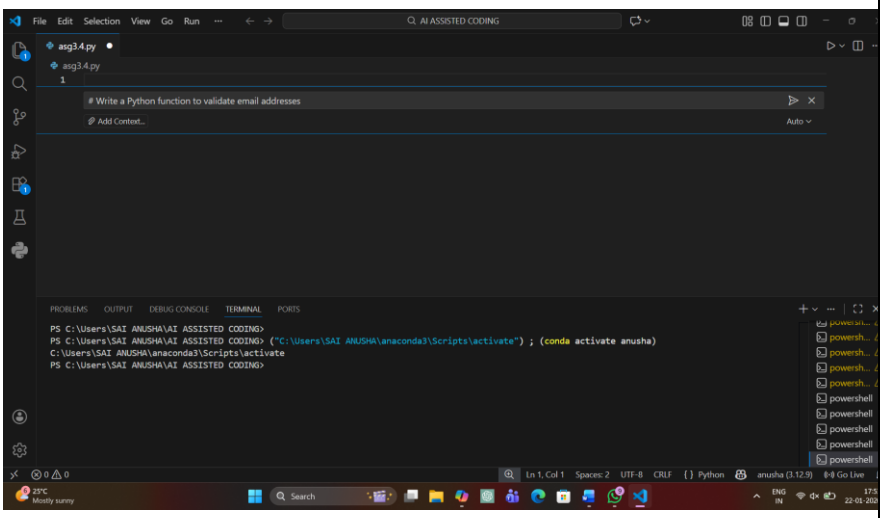
• Compare both outputs:

Zero-shot may result in basic or generic validation.

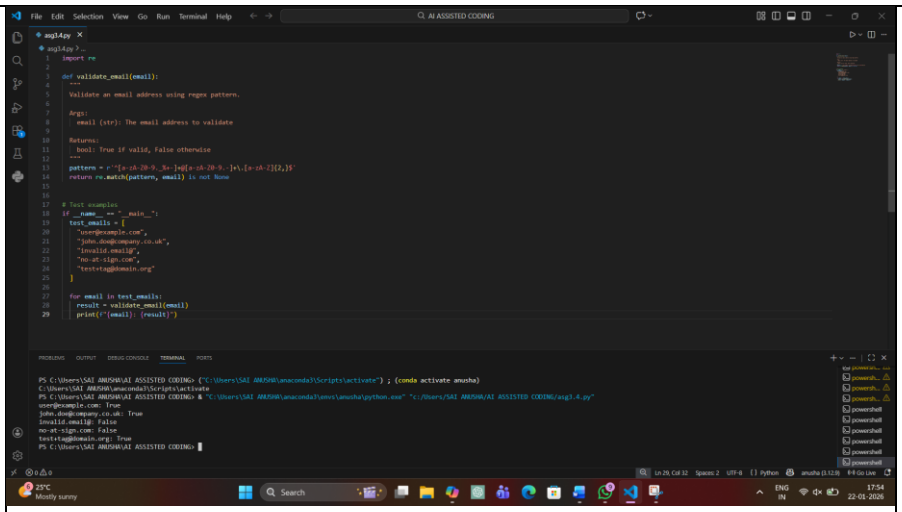
Few-shot gives detailed and specific logic (e.g., @ and domain checking).

• Submit both code versions and note how few-shot improves reliability.

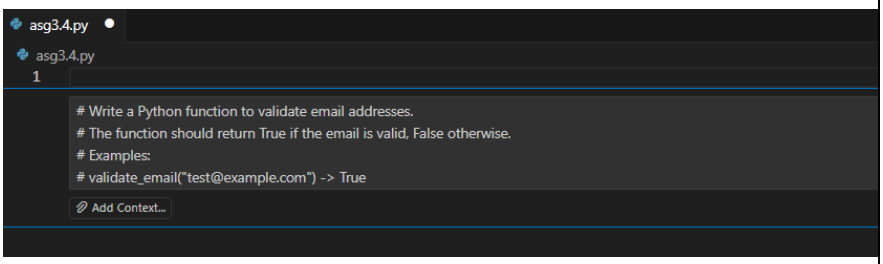
Prompt:

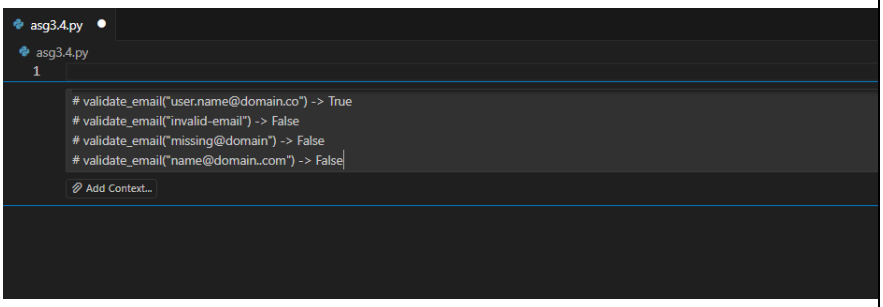


Code and output:

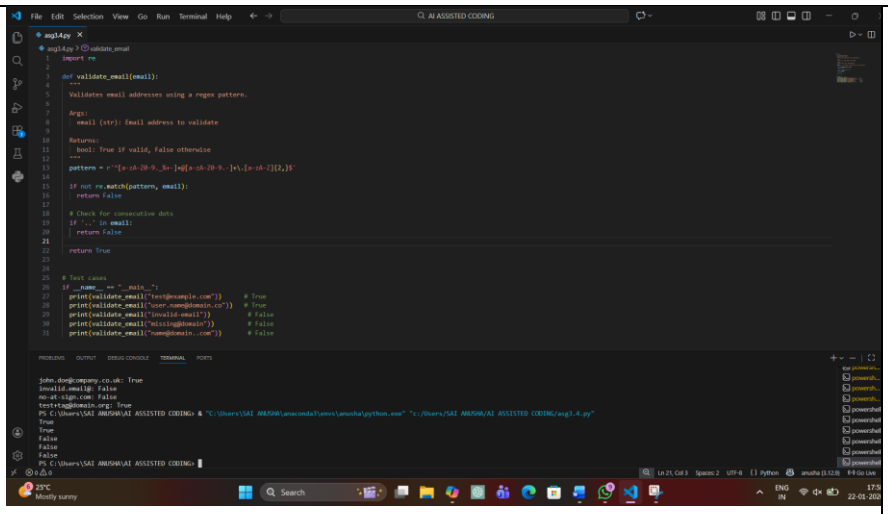


Prompt:





Code and output:



Explanation:

This is a zero-shot prompt because it only gives the task instruction without any examples or extra guidance. Copilot has to decide the logic on its own, which often results in very basic or incomplete solutions. Since no valid or invalid cases are shown, the model cannot clearly understand the exact rules for email validation. That is why zero-shot outputs are usually less accurate than few-shot ones.

Task 5: Prompt Tuning – Summing Digits of a Number

Task Description #5

• Experiment with 2 different prompt styles to generate a function that returns the sum of digits of a number.

Style 1: Generic task prompt

Style 2: Task + Input/Output example

Expected Output #5

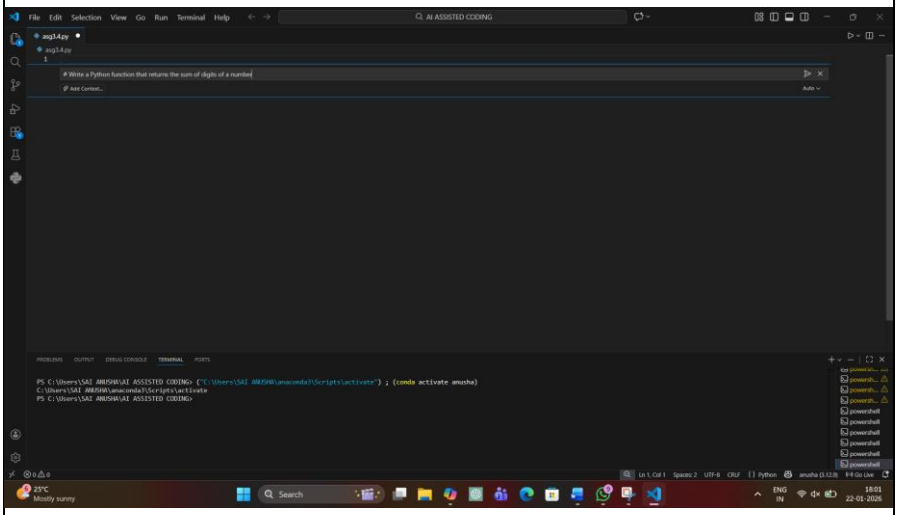
• Two versions of the sum_of_digits() function.

• Example Output: sum_of_digits(123) → 6

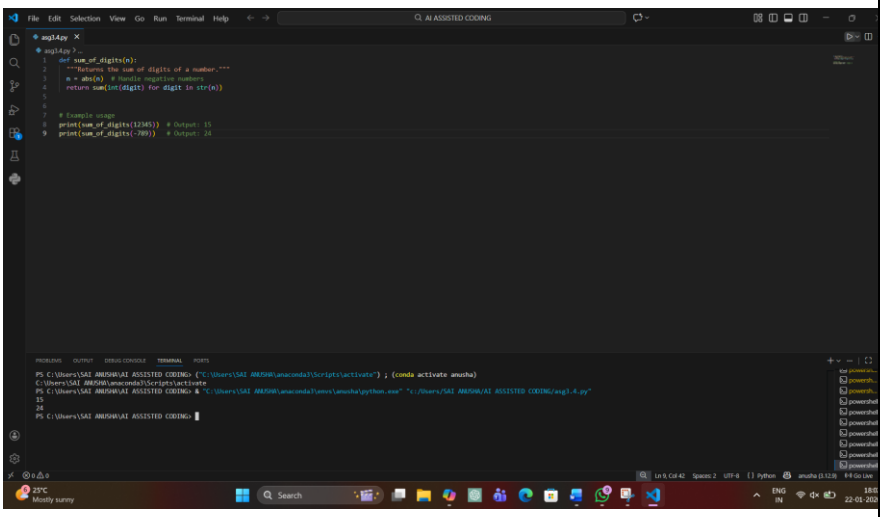
• Short analysis: which

prompt produced cleaner or more optimized code and why?

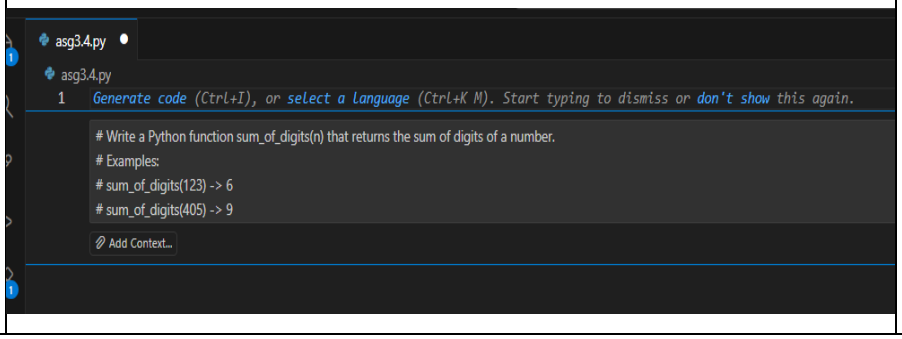
Generic task prompt:



Code and output:



Task + Input/Output example:



asg3.4.py

```
# sum_of_digits(123) -> 6
# sum_of_digits(405) -> 9
# sum_of_digits(0) -> 0
# sum_of_digits(9999) -> 36
```

Add Context...

Code:

asg3.4.py

```python
def sum_of_digits(n):

    Args:
      n: An integer

    Returns:
      The sum of all digits in n
    """
    return sum(int(digit) for digit in str(abs(n)))


# Test cases
if __name__ == "__main__":
    print(sum_of_digits(123))    # -> 6
    print(sum_of_digits(405))    # -> 9
    print(sum_of_digits(0))      # -> 0
    print(sum_of_digits(9999))   # -> 36
```

Output:

```
PS C:\Users\SAI ANUSHA\AI ASSISTED CODING> & "C:\Users\SAI ANUSHA\anaconda3\envs\anusha\python.exe" "c:/Users/SAI ANUSHA/AI ASSISTED CODING/asg3.4.py"
6
9
0
36
```

Explanation:

The generic prompt only describes the task, so Copilot usually generates a basic and longer solution using loops. However, when examples are included, Copilot clearly understands the expected behavior and often produces cleaner and more optimized code. The few-shot prompt leads to a shorter and more Pythonic solution because the model learns the pattern from the examples.

	Therefore, the second prompt style produces more accurate and efficient code. **Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots**	