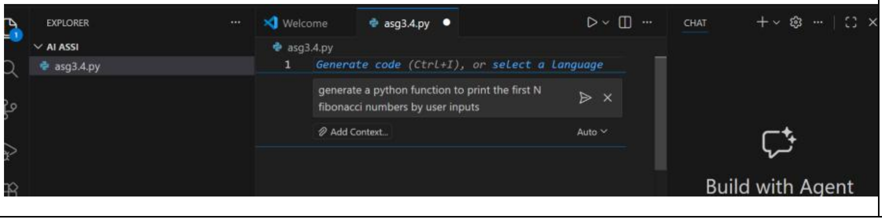
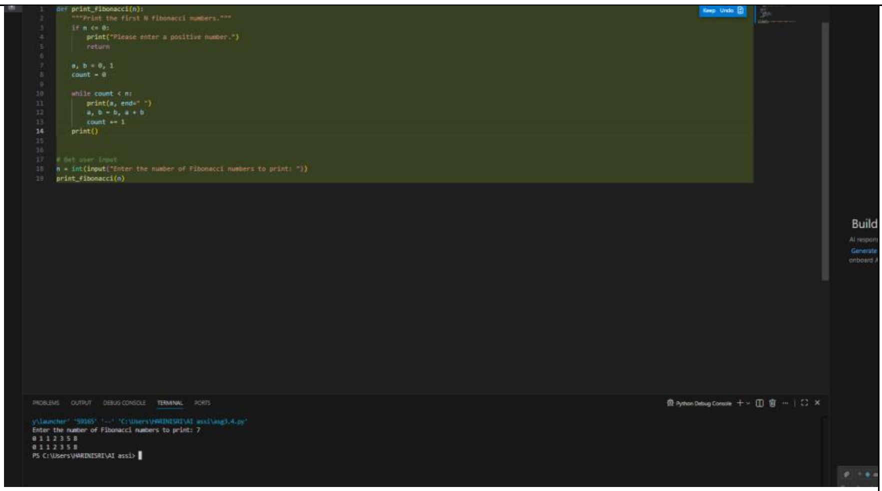
K.Harini sri
2303A51602
Batch_25

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 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.  `Generate code (Ctrl+I), or select a language` generate a python function to print the first N fibonacci numbers by user inputs EXPLORER AI ASSI asg3.4.py Welcome asg3.4.py Add Context... Auto CHAT Build with Agent	Week2



```python
def print_fibonacci(n):
    """Print the first N fibonacci numbers."""
    if n <= 0:
        print("Please enter a positive number.")
        return

    a, b = 0, 1
    count = 0

    while count < n:
        print(a, end=" ")
        a, b = b, a + b
        count += 1
    print()


# Get user input
n = int(input("Enter the number of Fibonacci numbers to print: "))
print_fibonacci(n)
```

```
y\launcher' '59165' '--' 'C:\Users\HARINISRI\AI assi\asg3.4.py'
Enter the number of Fibonacci numbers to print: 7
0 1 1 2 3 5 8
0 1 1 2 3 5 8
PS C:\Users\HARINISRI\AI assi>
```

Task 2 : one – shot Prompt – line Reversal Function

Task Description #2

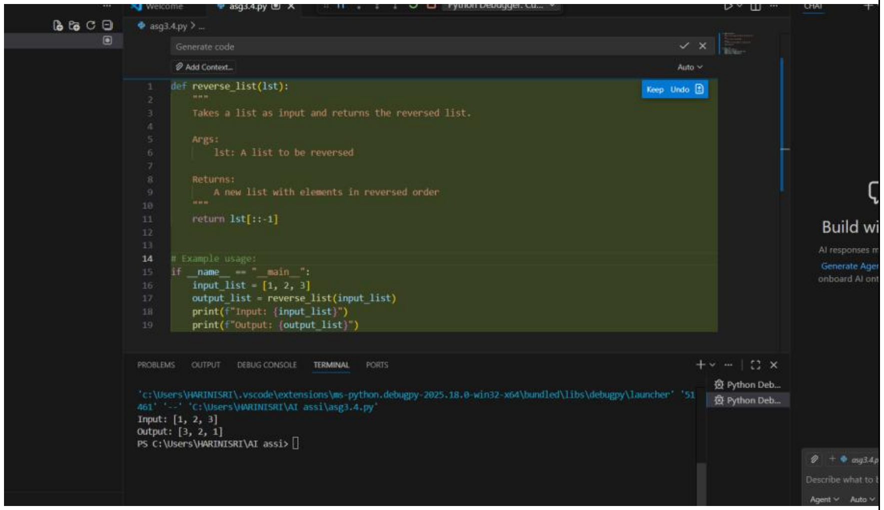
• Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

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.



```python
def reverse_list(lst):
    """
    Takes a list as input and returns the reversed list.

    Args:
        lst: A list to be reversed

    Returns:
        A new list with elements in reversed order
    """
    return lst[::-1]


# Example usage:
if __name__ == "__main__":
    input_list = [1, 2, 3]
    output_list = reverse_list(input_list)
    print(f"Input: {input_list}")
    print(f"Output: {output_list}")
```

```
'c:\Users\HARINISRI\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51
461' '--' 'C:\Users\HARINISRI\AI assi\asg3.4.py'
Input: [1, 2, 3]
Output: [3, 2, 1]
PS C:\Users\HARINISRI\AI assi>
```

Task 3: Few-shot Prompt – String Pattern Matching

Task Description #3

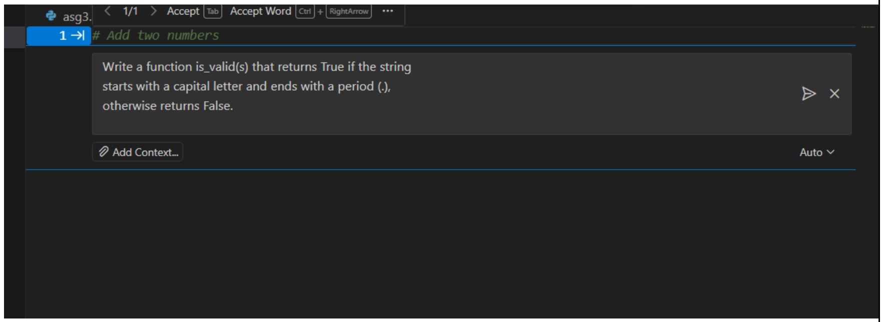
• Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.

Expected Output #3

• A function is_valid() that checks the pattern.

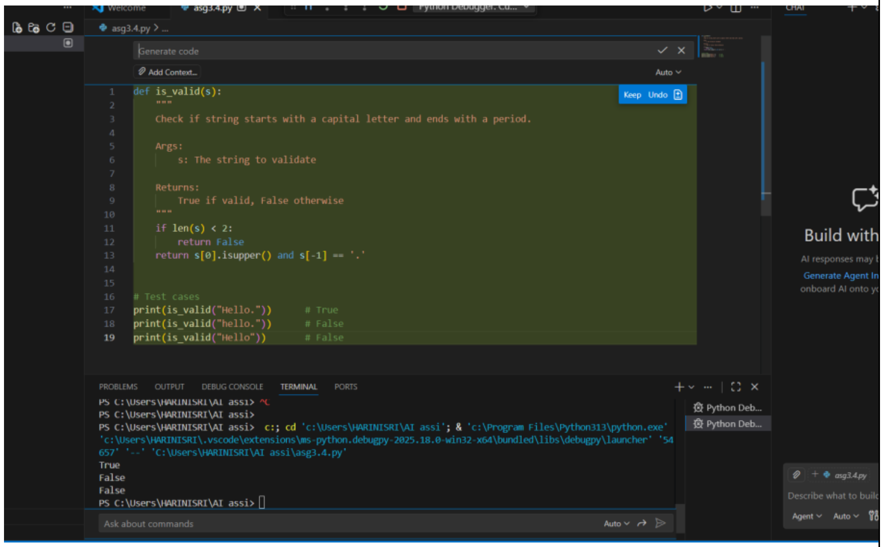
• Output: True or False based on input.

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



```
# Add two numbers
```

Write a function is_valid(s) that returns True if the string starts with a capital letter and ends with a period (.), otherwise returns False.



```python
def is_valid(s):
    """
    Check if string starts with a capital letter and ends with a period.

    Args:
        s: The string to validate

    Returns:
        True if valid, False otherwise
    """
    if len(s) < 2:
        return False
    return s[0].isupper() and s[-1] == '.'


# Test cases
print(is_valid("Hello."))      # True
print(is_valid("hello."))      # False
print(is_valid("Hello"))       # False
```

```
PS C:\Users\HARINISRI\AI assi> ^C
PS C:\Users\HARINISRI\AI assi>
PS C:\Users\HARINISRI\AI assi>  c:; cd 'c:\Users\HARINISRI\AI assi'; & 'c:\Program Files\Python313\python.exe'
'c:\Users\HARINISRI\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54
657' '--' 'C:\Users\HARINISRI\AI assi\asg3.4.py'
True
False
False
PS C:\Users\HARINISRI\AI assi> 
```

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.

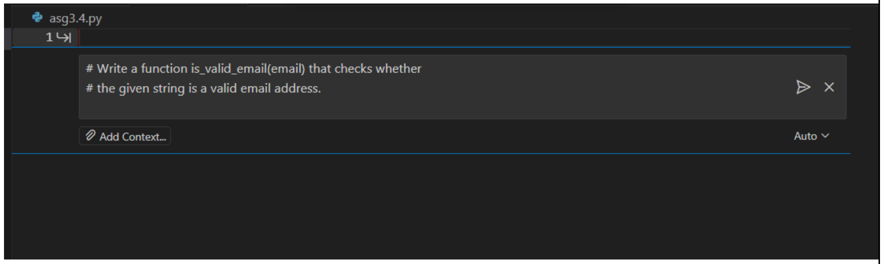
Expected Output #4

• Compare both outputs:

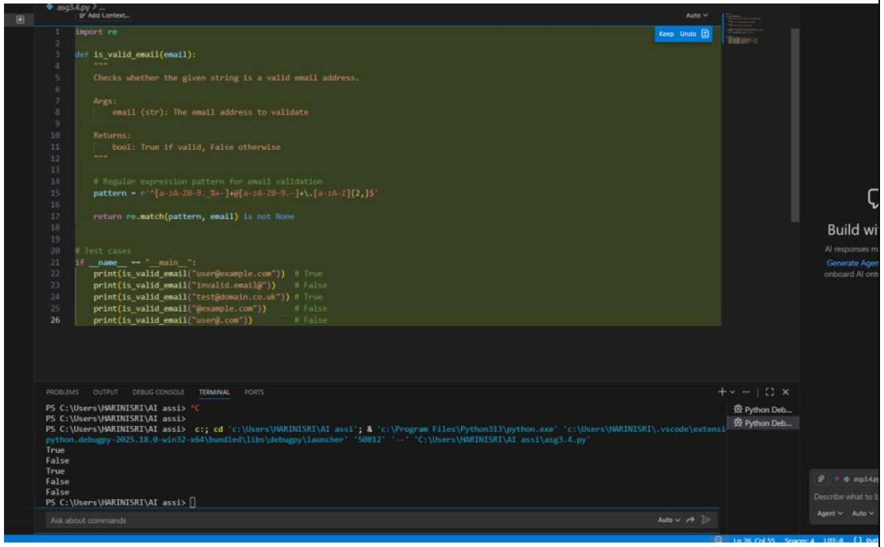
Zero-shot may result in basic or generic validation.

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

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



```
# Write a function is_valid_email(email) that checks whether
# the given string is a valid email address.
```



```python
import re

def is_valid_email(email):
    """
    Checks whether the given string is a valid email address.

    Args:
        email (str): The email address to validate

    Returns:
        bool: True if valid, False otherwise
    """

    # Regular expression pattern for email validation
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'

    return re.match(pattern, email) is not None


# Test cases
if __name__ == "__main__":
    print(is_valid_email("user@example.com"))  # True
    print(is_valid_email("invalid.email@"))    # False
    print(is_valid_email("test@domain.co.uk")) # True
    print(is_valid_email("@example.com"))      # False
    print(is_valid_email("user@.com"))         # False
```

```
PS C:\Users\HARINISRI\AI assi> ^C
PS C:\Users\HARINISRI\AI assi>
PS C:\Users\HARINISRI\AI assi>  c:; cd 'c:\Users\HARINISRI\AI assi'; & 'c:\Program Files\Python313\python.exe' 'c:\Users\HARINISRI\.vscode\extensi
python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50812' '--' 'C:\Users\HARINISRI\AI assi\asg3.4.py'
True
False
True
False
False
PS C:\Users\HARINISRI\AI assi>
```

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

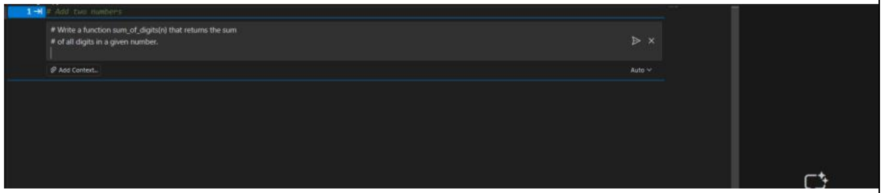
Style 2: Task + Input/Output example

Expected Output #5

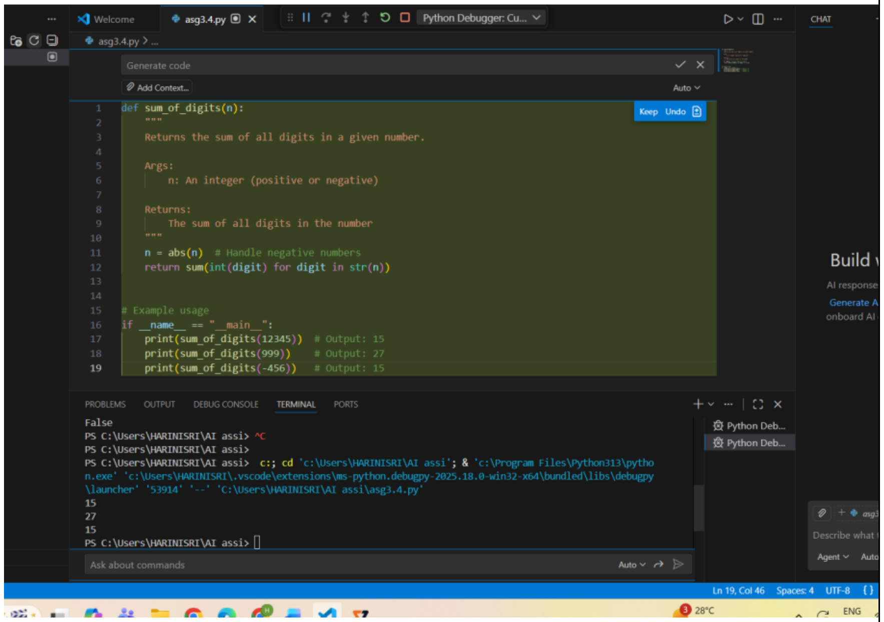
• Two versions of the sum_of_digits() function.

• Example Output: sum_of_digits(123) → 6

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



```
# Write a function sum_of_digits(n) that returns the sum
# of all digits in a given number.
```



```python
def sum_of_digits(n):
    """
    Returns the sum of all digits in a given number.

    Args:
        n: An integer (positive or negative)

    Returns:
        The sum of all digits in the number
    """
    n = abs(n)  # Handle negative numbers
    return sum(int(digit) for digit in str(n))


# Example usage
if __name__ == "__main__":
    print(sum_of_digits(12345))  # Output: 15
    print(sum_of_digits(999))    # Output: 27
    print(sum_of_digits(-456))   # Output: 15
```

```
False
PS C:\Users\HARINISRI\AI assi> ^C
PS C:\Users\HARINISRI\AI assi>
PS C:\Users\HARINISRI\AI assi>  c:; cd 'c:\Users\HARINISRI\AI assi'; & 'c:\Program Files\Python313\pytho
n.exe' 'c:\Users\HARINISRI\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy
\launcher' '53914' '--' 'C:\Users\HARINISRI\AI assi\asg3.4.py'
15
27
15
PS C:\Users\HARINISRI\AI assi>
```

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