

## ASSIGNMENT-3.4

Name: Praneeth banda

HT.No: 2303A51711

Batch: 20

### Task 1: Zero-shot Prompt – Fibonacci Series Generator

Scenario : In this task, a zero-shot prompting technique was used. A single comment prompt was written without providing any examples, instructing GitHub Copilot to generate a Python function that prints the first N Fibonacci numbers.

Prompt: # Write a Python function to print the first N Fibonacci numbers

Code:

```
# Task-1 : Fibonacci Series Generator
def fibonacci(n):
    # Generate Fibonacci series up to n terms
    a, b = 0, 1
    for _ in range(n):
        print(a, end=' ')
        a, b = b, a + b
    print()

# Get input from user
num_terms = int(input("Enter number of terms: "))
fibonacci(num_terms)
```

Result:

```
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" sers/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-3.4.py"
c:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST\assignment-3.4.py:43: SyntaxWarning: invalid sequence '\.'
  pattern = r'^[a-zA-Z0-9._%+~]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter number of terms: 5
0 1 1 2 3
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> █
```

Observation:

The zero-shot prompt was sufficient for Copilot to correctly infer the Fibonacci logic, even without any examples or additional context. However, the function behavior depended heavily on Copilot's prior training, and the output format was assumed rather than explicitly defined. This shows that zero-shot prompting works well for well-known problems but may lack consistency for ambiguous or complex tasks.

Task 2: One-shot Prompt – List Reversal Function. In this task, a one-shot prompting approach was used by providing a single example along with the instruction to help Copilot generate a correct list reversal function

Prompt: # Write a Python function to reverse a list

# Example: input [1, 2, 3] -> output [3, 2, 1] Code:

```
# Task-2 : List Reversal Function
def reverse_list(lst):
    # Reverse a list and return it
    return lst[::-1]

# Get input from user
user_list = input("Enter list elements (space-separated): ").split()
reversed_list = reverse_list(user_list)
print("Reversed list:", reversed_list)
```

Result:

```
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:/Users/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-3.4.py"
c:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST\assignment-3.4.py:44: SyntaxWarning: invalid escape sequence '\.'
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter list elements (space-separated): 23 34 43 12 25
Reversed list: ['25', '12', '43', '34', '23']
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> █
```

Observation:

Providing one example significantly improved Copilot's accuracy and confidence in choosing an optimal approach. The generated solution was concise and efficient, using Python slicing. Compared to zero-shot prompting, one-shot prompting reduced ambiguity and guided Copilot toward the expected output format and logic

### Task 3: Few-shot Prompt – String Pattern Matching

Scenario: This task used a few-shot prompting technique by providing multiple examples to help Copilot understand a specific string validation pattern.

Prompt: # Write a function to check if a string starts with a capital letter and ends with a period

# Example: "Hello." -> True

# Example: "hello." -> False

# Example: "Hello" -> False

Code:

```
"""# Task-3 : String Pattern Matching
def check_pattern(text):
    # Check if string starts with capital letter and ends with period
    return text[0].isupper() and text[-1] == '.'

# Example 1: "Hello." -> True (starts with 'H', ends with '.')
# Example 2: "hello." -> False (starts with 'h', not capital)
# Example 3: "Hello" -> False (ends with 'o', not period)

user_text = input("Enter a string: ")
if check_pattern(user_text):
    print("True")
else:
    print("False")
```

Result:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "c:
sers/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-3.4.py"
c:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST\assignment-3.4.py:44: SyntaxWarning: invalid esca
sequence '\.'
  pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter a string: Hello
False
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> █
```

Observation:

The few-shot prompt enabled Copilot to accurately identify the string pattern requirements and generate a precise validation function. The multiple examples clarified edge cases and reduced misinterpretation. This demonstrates that few-shot prompting is highly effective when pattern recognition or conditional logic is involved.

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

You are participating in a code review session. This task compares zero-shot and few-shot prompting by generating two versions of an email validation function and analyzing their differences

Prompt: Zero-Shot Prompt: # Write a Python function to validate an email address

Prompt: Few-Shot Prompt: # Write a Python function to validate an email address

# Example: "test@gmail.com" -> True

# Example: "testgmail.com" -> False

# Example: "test@com" -> False

Code:

```
# Task-4 : Zero-shot vs Few-shot - Email Validator
import re
def validate_email(email):
    # Simple email validation using regex
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None

# Get input from user
user_email = input("Enter an email: ")
if validate_email(user_email):
    print("Valid email")
else:
    print("Invalid email")
```

Result:

```
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python
sers/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-3.4.py"
Enter an email: bandaparneeth4@gmail.com
Valid email
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> █
```

## Observation

The zero-shot version produced a very basic and unreliable validation logic, while the few-shot prompt resulted in a more structured and realistic solution. The examples guided Copilot to include domain checks and input validation, significantly improving reliability. This comparison clearly highlights the advantage of few-shot prompting for real-world validation tasks.

Task 5: Prompt Tuning – Summing Digits of a Number. In this task, two different prompt styles were used to study how prompt tuning affects code quality and optimization.

Prompt: Style-1:Generic Prompt # Write a function to return the sum of digits of a number

Prompt with I/O Example: # Write a function to return the sum of digits of a number #

Example: sum\_of\_digits(123) -> 6

Code:

```
# Task-5 : Summing Digits of a Number
def sum_digits(num):
    # Sum all digits in a number
    return sum(int(digit) for digit in str(abs(num)))

# Get input from user
user_num = int(input("Enter a number: "))
result = sum_digits(user_num)
print("Sum of digits:", result)
```

Result:

```
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> & "C:/Program Files/Python313/python.exe" "
sers/Banda Praneeth/OneDrive/Desktop/AI ASSIST/assignment-3.4.py"
c:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST\assignment-3.4.py:44: SyntaxWarning: invalid es
sequence '\.'
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
Enter a number: 143
Sum of digits: 8
PS C:\Users\Banda Praneeth\OneDrive\Desktop\AI ASSIST> █
```

Observation:

The prompt that included an input-output example produced a cleaner and more optimized implementation. The example encouraged Copilot to generate concise and Pythonic code using built-in functions. This demonstrates that prompt tuning with examples not only improves correctness but also enhances code quality and efficiency.

