

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	23CS002 PC304	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)			

	Question	Expected Time to complete

		<i>mpl ete</i>
1	<p>Lab 4: Advanced Prompt Engineering – Zero-shot, One-shot, and Few-shot Techniques</p> <p>Task 1: Zero-shot Prompt – Fibonacci Series Generator</p> <p>Task Description #1</p> <ul style="list-style-type: none"> Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers. <p>Expected Output #1</p> <ul style="list-style-type: none"> A complete Python function generated by Copilot without any example provided. Correct output for sample input $N = 7 \rightarrow 0 1 1 2 3 5 8$ Observation on how Copilot understood the instruction with zero context. <p>Prompt :</p> <p>Write a Python function that prints the first N Fibonacci numbers.</p>	Week 2

The screenshot shows the VS Code interface with the following components:

- Editor Area:** Displays the file `t1.py` containing Python code for generating Fibonacci numbers.
- Output Area:** Shows the terminal output where the script is run and prints the first 10 Fibonacci numbers: `0 1 1 2 3 5 8 13 21 34`.
- Copilot Panel:** On the right, a panel titled "GENERATING FIBONACCI NUMBERS" provides context for the task and shows the AI's thought process and code generation steps.

Code in `t1.py`:

```

1 def print_fibonacci(n):
2     """
3         Prints the first N Fibonacci numbers.
4
5     Args:
6         n (int): The number of Fibonacci numbers to print.
7
8         if n <= 0:
9             print("Please enter a positive integer.")
10            return
11
12        a, b = 0, 1
13        for _ in range(n):
14            print(a, end=" ")
15            a, b = b, a + b
16        print() # For a new line after printing the sequence
17
18    # Example usage
19    print_fibonacci(10)

```

Copilot Panel Content:

Write a Python function that prints the first Fibonacci numbers.

✓ Optimizing tool selection...
✓ Edited `t1.py` 19 -1
I have added a function `print_fibonacci`. It prints the first N Fibonacci numbers. I know if you need any further modifications!

> 1 file changed → 19 -1

Describe what to build next

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

Write a Python function to reverse a list.

Example:

Input: [1, 2, 3]

Output: [3, 2, 1]

The screenshot shows a code editor interface with a Python file named `t1.py`. The code defines a `reverse_list` function that takes a list as input and returns its reverse. An example usage is shown where the function is called with the list `[1, 2, 3]` and the output is printed as `[3, 2, 1]`. To the right of the code editor, there is a sidebar titled "GENERATING FIBONACCI NUMBERS IN ...". It displays the input `[1, 2, 3]` and output `[3, 2, 1]`. Below this, a message says: "I have added a function `reverse_list` to reverse a given list, along with an example usage. Let me know if you need further assistance!" At the bottom of the interface, there is a terminal window showing the command `PS C:\Users\PC\Documents\web projects\CC\AI ASST\AI 3.4> & C:/Python312/python.exe "c:/Users/PC/Documents/web projects/CC/ASST/AI 3.4/t1.py"` and the output `0 1 1 2 3 5 8 13 21 34`.

```
PS C:\Users\PC\Documents\web projects\CC\AI ASST\AI 3.4> & C:/Pyd  
exe "c:/Users/PC/Documents/web projects/CC/AI ASST/AI 3.4/t1.py"  
0 1 1 2 3 5 8 13 21 34  
PS C:\Users\PC\Documents\web projects\CC\AI ASST\AI 3.4>
```

Observation

Adding **one example** helped Copilot:

- Choose slicing instead of complex logic
- Produce shorter and cleaner code

One-shot prompting improves **accuracy and simplicity**.

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

Write a function `is_valid(s)` that checks whether:

Example 1: "Hello." → True

Example 2: "hello." → False

Example 3: "Hello" → False

A valid string starts with a capital letter and ends with a period.

The screenshot shows a code editor interface with several windows open. The main window displays a Python script named `t1.py` containing the following code:

```
18 def is_valid(s):
19     """
20     Checks whether a string is valid.
21
22     A valid string starts with a capital letter and ends with a period.
23
24     Args:
25         s (str): The string to check.
26
27     Returns:
28         bool: True if the string is valid, False otherwise.
29     """
30     return s[0].isupper() and s.endswith('.')
31
32 # Example usage
33 print(fibonacci(10))
```

The code includes a docstring explaining that a valid string must start with a capital letter and end with a period. It defines an `is_valid` function that checks these conditions. Below the code, there's an example usage of the `fibonacci` function.

On the right side of the interface, there's a dark-themed sidebar with the title "GENERATING FIBONACCI NUMBERS IN PYTHON". It contains the following text:

Example 3: "Hello" → False

A valid string starts with a capital letter and ends with a period.

I have added the `is_valid` function to check if a string starts with a capital letter and ends with a period, along with example usage. Let me know if you need further assistance!

At the bottom of the sidebar, there's a message: "Describe what to build next".

At the very bottom of the interface, there's a terminal window showing command-line output:

```
PS C:\Users\PC\Documents\web projects\CC\AI ASST\AI 3.4> & C:/Python312/python.exe "c:/Users/PC/Documents/web projects/CC/AI ASST/AI 3.4/t1.py"
0 1 1 2 3 5 8 13 21 34
PS C:\Users\PC\Documents\web projects\CC\AI ASST\AI 3.4> & C:/Python312/python.exe "c:/Users/PC/Documents/web projects/CC/AI ASST/AI 3.4/t1.py"
0 1 1 2 3 5 8 13 21 34
True
False
False
```

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.

Zero-shot Prompt

```
# Write a Python function to validate an email address.
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. In the center, there is a floating window titled "GENERATING FIBONACCI NUMBERS IN PYTHON" which contains the text "Write a Python function to validate email address.". Below this window, the main code editor shows a Python file named "t1.py" with the following code:

```
def is_valid_email(email):
    email_regex = r'^[a-zA-Z0-9.-]+@[a-zA-Z0-9.-]+\.[a-zA-Z0-9.-]+$'
    return re.match(email_regex, email) is not None

# Example usage
print(is_valid("Hello")) # True
print(is_valid("Hello.")) # False
print(is_valid(".")) # False
print(is_valid_email("example@example.com")) # True
print(is_valid_email("invalid-email")) # False
```

The code editor has tabs for "OUTLINE", "TIMELINE", "PROJECTS", "LOGICAL STRUCTURE", and "JAVA PROJECTS". The status bar at the bottom indicates "1 file changed +17 -0" and "t1.py".

Comparison Observation

Aspect	Zero-shot	Few-shot
Logic	Very basic	Detailed & reliable
Accuracy	Low	High
Validation	Weak	Strong

Few-shot prompting clearly **improves reliability**.

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?

Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if require d, screenshots

Style 1: Generic Prompt

```
# Write a Python function to return the sum of digits  
of a number.
```

The screenshot shows a code editor interface with a Python file named `t2.py`. The code defines a function `sum_of_digits` that calculates the sum of the digits of a given number. The function includes docstrings and example usage. To the right of the code editor is a terminal window showing the command `python t2.py` being run and its output, which is the sum of the digits for the input numbers 123 and -456.

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

    Args:
        number (int): The number whose digits will be summed.

    Returns:
        int: The sum of the digits.
    """
    return sum([int(digit) for digit in str(abs(number))])

# Example usage
print(sum_of_digits(123)) # 6
print(sum_of_digits(-456)) # 15
```

```
PS C:\Users\PC\Documents\web projects\CC\AI ASST\AI 3.4> & C:/Python312/python.exe "c:/Users/PC/Documents/web projects/CC/AI ASST/AI 3.4/t2.py"
6
15
```

Analysis

Style-2 prompt produced:

- Cleaner code
- One-line optimized solution
- Better readability

👉 **Prompt tuning improves code quality.**

Final Conclusion

- **Zero-shot** → Works but may be basic
- **One-shot** → Improves clarity
- **Few-shot** → Best accuracy and logic
- **Prompt tuning** → Produces optimized code

GitHub Copilot responds better when **examples are included**.