NAME:- SIDDHARTHA REDDY KOTHA || HALL TICKET NO.:- 2303A51491 || B- 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	Week2

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Task Description #1

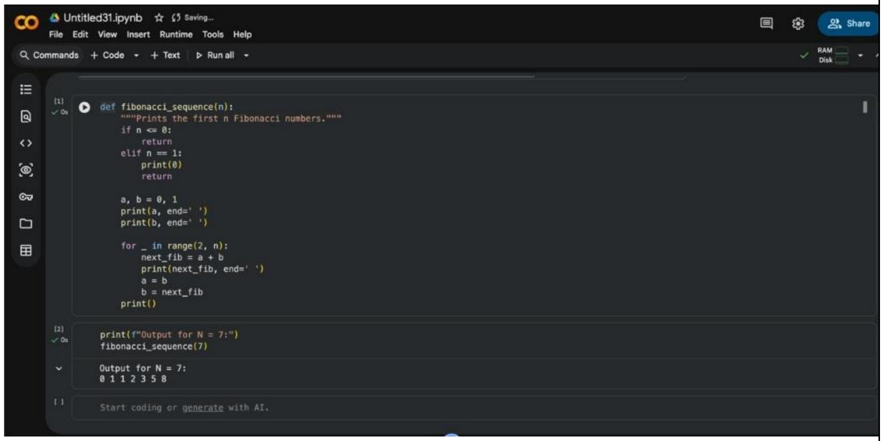
• Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.

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.



```
def fibonacci_sequence(n):
    """Prints the first n Fibonacci numbers."""
    if n <= 0:
        return
    elif n == 1:
        print(0)
        return

    a, b = 0, 1
    print(a, end=' ')
    print(b, end=' ')

    for _ in range(2, n):
        next_fib = a + b
        print(next_fib, end=' ')
        a = b
        b = next_fib
    print()
```

```
print(f"Output for N = 7:")
fibonacci_sequence(7)
```

```
Output for N = 7:
0 1 1 2 3 5 8
```

Start coding or generate with AI.

Task 2: One-shot Prompt – List Reversal Function

Task Description #2

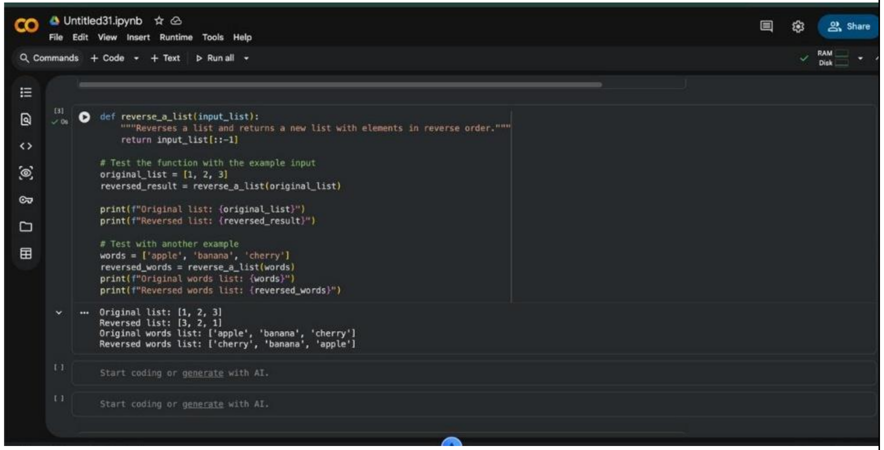
• Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

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.


Untitled31.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk

```python
def reverse_a_list(input_list):
    """Reverses a list and returns a new list with elements in reverse order."""
    return input_list[::-1]

# Test the function with the example input
original_list = [1, 2, 3]
reversed_result = reverse_a_list(original_list)

print(f"Original list: {original_list}")
print(f"Reversed list: {reversed_result}")

# Test with another example
words = ['apple', 'banana', 'cherry']
reversed_words = reverse_a_list(words)
print(f"Original words list: {words}")
print(f"Reversed words list: {reversed_words}")
```

```
Original list: [1, 2, 3]
Reversed list: [3, 2, 1]
Original words list: ['apple', 'banana', 'cherry']
Reversed words list: ['cherry', 'banana', 'apple']
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Task 3: Few-shot Prompt – String Pattern Matching

Task Description #3

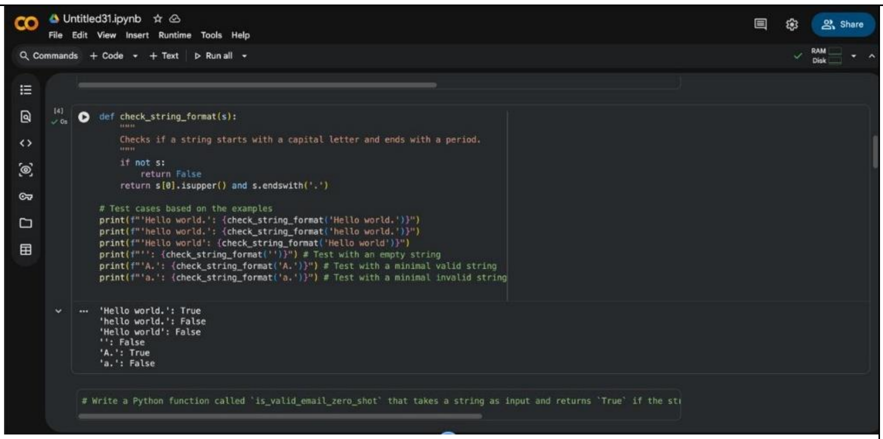
• Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.

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.



```python
def check_string_format(s):
    """
    Checks if a string starts with a capital letter and ends with a period.
    """
    if not s:
        return False
    return s[0].isupper() and s.endswith('.')

# Test cases based on the examples
print(f"'Hello world.': {check_string_format('Hello world.')}")
print(f"'hello world.': {check_string_format('hello world.')}")
print(f"'Hello world': {check_string_format('Hello world')}")
print(f"'': {check_string_format('')}") # Test with an empty string
print(f"'A.': {check_string_format('A.')}") # Test with a minimal valid string
print(f"'a.': {check_string_format('a.')}") # Test with a minimal invalid string
```

```
'Hello world.': True
'hello world.': False
'Hello world': False
'': False
'A.': True
'a.': False
```

```python
# Write a Python function called `is_valid_email_zero_shot` that takes a string as input and returns `True` if the st
```

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.

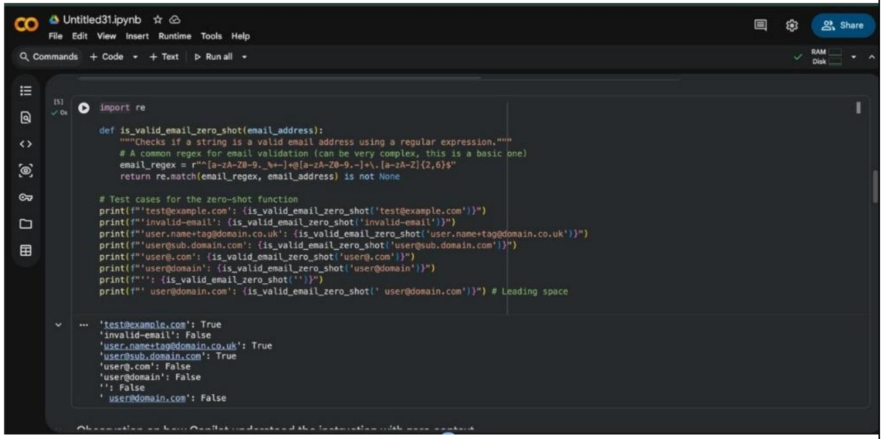
Expected Output #4

• 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.



```python
import re

def is_valid_email_zero_shot(email_address):
    """Checks if a string is a valid email address using a regular expression."""
    # A common regex for email validation (can be very complex, this is a basic one)
    email_regex = r"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$"
    return re.match(email_regex, email_address) is not None

# Test cases for the zero-shot function
print(f"'test@example.com': {is_valid_email_zero_shot('test@example.com')}")
print(f"'invalid-email': {is_valid_email_zero_shot('invalid-email')}")
print(f"'user.name+tag@domain.co.uk': {is_valid_email_zero_shot('user.name+tag@domain.co.uk')}")
print(f"'user@sub.domain.com': {is_valid_email_zero_shot('user@sub.domain.com')}")
print(f"'user@.com': {is_valid_email_zero_shot('user@.com')}")
print(f"'user@domain': {is_valid_email_zero_shot('user@domain')}")
print(f"'': {is_valid_email_zero_shot('')}")
print(f"' user@domain.com': {is_valid_email_zero_shot(' user@domain.com')}") # Leading space
```

```
'test@example.com': True
'invalid-email': False
'user.name+tag@domain.co.uk': True
'user@sub.domain.com': True
'user@.com': False
'user@domain': False
'': False
' user@domain.com': False
```

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

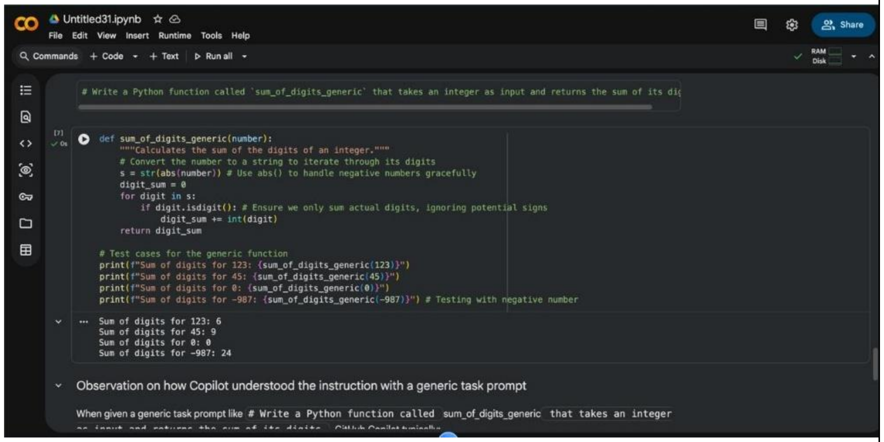
Style 2: Task + Input/Output example

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?



Untitled31.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
RAM Disk

```
# Write a Python function called `sum_of_digits_generic` that takes an integer as input and returns the sum of its dig
```

```
def sum_of_digits_generic(number):
    """Calculates the sum of the digits of an integer."""
    # Convert the number to a string to iterate through its digits
    s = str(abs(number)) # Use abs() to handle negative numbers gracefully
    digit_sum = 0
    for digit in s:
        if digit.isdigit(): # Ensure we only sum actual digits, ignoring potential signs
            digit_sum += int(digit)
    return digit_sum

# Test cases for the generic function
print(f"Sum of digits for 123: {sum_of_digits_generic(123)}")
print(f"Sum of digits for 45: {sum_of_digits_generic(45)}")
print(f"Sum of digits for 0: {sum_of_digits_generic(0)}")
print(f"Sum of digits for -987: {sum_of_digits_generic(-987)}") # Testing with negative number
```

```
Sum of digits for 123: 6
Sum of digits for 45: 9
Sum of digits for 0: 0
Sum of digits for -987: 24
```

Observation on how Copilot understood the instruction with a generic task prompt

When given a generic task prompt like `# Write a Python function called` sum_of_digits_generic `that takes an integer`

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