

ASSIGNMENT-5

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

Prompt: #write a python function to print the first N Fibonacci numbers.

Code and Output:

```
1  #write a python program function to print the frist n Fibonacci numbers.
2  def print_fibonacci(n):
3      a, b = 0, 1
4      for _ in range(n):
5          print(a, end=' ')
6          a, b = b, a + b
7  # Example usage:
8  n = 10
9  print_fibonacci(n)
10
```

The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

c:\; cd 'c:\Users\dell\OneDrive\Desktop\AI Assistent-lab'; & 'c:\Program Files\Python311\python.exe' 'c:\Users\dell\OneDrive\Desktop\AI Assistent-lab\ai-1ab-3.4.pyt'
0 1 1 2 3 5 8 13 21 34
PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab>
```

Justification:

- Copilot correctly understood the task without any example
- Generated a valid Fibonacci logic using iteration
- Shows that zero-shot prompting works well for common problems

Task-2: One-shot Prompt – List Reversal Function

Prompt: #write a python function to reverse a list

#Example: Input: [3,2,1] **Output:** [1,2,3]

Code and Output:

```
ai la-3.4.pyt > ...
1 #write a python fuction to reverse a list using slicing or loop
2 def reverse_list(lst):
3     # Using slicing to reverse the list
4     return lst[::-1]
5 # Example usage:
6 example_list = [1, 2, 3]
7 reversed_list = reverse_list(example_list)
8 print(f"Original list: {example_list}")
9 print(f"Reversed list: {reversed_list}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
c:\ cd 'c:\Users\dell\OneDrive\Desktop\AI Assistent-lab'; & 'c:\Program Files\Python311\python.exe' 'c:\User
s\dell\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52781' '--' 'c:\Users\dell\OneDrive\Desktop\AI Assistent-lab\ai l
a-3.4.pyt'
Original list: [1, 2, 3]
Reversed list: [3, 2, 1]
PS c:\Users\dell\OneDrive\Desktop\AI Assistent-lab> []
```

Justification:

- Adding one example improved accuracy
- Copilot used Python slicing, which is clean and efficient
- One-shot prompts guide Copilot toward better solutions

Task-3: String Pattern Matching

Prompt:

```
#write a python function o check if a string starts with a capital letter and ends with
a period.
```

```
#Example: Input: "Coding." Output: True
```

```
#Example: Input: "coding." Output: False
```

```
#Example: Input: "Coding" Output: False
```

Code and Output:

```
1 #write a python fuction to check if a string starts with a capital letter and ends with a period
2 def check_string_format(s):
3     if s and s[0].isupper() and s.endswith('.'):
4         return True
5     return False
6 # Example usage:
7 print(check_string_format("Hello world."))
8 print(check_string_format("hello world."))
9 print(check_string_format("Hello world"))
10 print(check_string_format(""))
```

The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following command and its output:

```
c:\ cd 'c:\Users\dell\OneDrive\Desktop\AI Assistant-lab'; & 'c:\Program Files\Python311\python.exe' 'c:\Users\dell\OneDrive\Desktop\AI Assistant-lab\3.4\3.4.3.pyt'
True
False
False
False
```

Justification:

- Multiple examples helped Copilot understand the pattern clearly
- Few-shot prompting produces more accurate and specific logic

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

Prompt for zero-shot: write a python function to check an email validation.

Code and Output:

```

1 #write a python function to check an email validation function using zero-shot
2 import re
3 def is_valid_email(email):
4     # Define a regex pattern for validating an email
5     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}+$'
6     # Use re.match to check if the email matches the pattern
7     if re.match(pattern, email):
8         return True
9     return False
10 # Example usage:
11 print(is_valid_email("test@example.com")) # True
12 print(is_valid_email("invalid-email")) # False
13 print(is_valid_email("test@example")) # False
14 print(is_valid_email("test@.com")) # False

```

The screenshot shows the VS Code interface with the terminal tab selected. The command `python 3.4(emailValidation).py` is run, followed by four test cases: "test@example.com", "invalid-email", "test@example", and "test@.com". The output shows the results: True, False, False, and False respectively.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\dell\OneDrive\Desktop\AI Assistent-lab> c:; cd 'c:\Users\dell\OneDrive\Desktop\AI Assistent-lab'; & 'c:\Program Files\Python311\python.exe' 'c:\Users\dell\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '58874' '---' 'c:\Users\dell\OneDrive\Desktop\AI Assistent-lab\3.4\3.4.4.pyt'
True
False
False
False

```

Prompt for few-shot: #write a python function to check an email validation.

#Example: Input: "coding@gmail.com" Output: True

#Example: Input: "codinggmail.com" Output: False

#Example: Input: "coding@gmaiicom" Output: False

Code and Output:

The screenshot shows the VS Code interface with the terminal tab selected. The code defines a function `validate_email` that checks if an input string is a valid email address using a regular expression. It then prompts the user for an email address and prints whether it is valid or not. The user enters "coding@gmail.com", which is valid, and "coding", which is not valid.

```

3.4(emailValidation).py > ...
1 #write a python function to check an email validation.
2 #Example: Input: "coding@gmail.com" Output: True
3 #Example: Input: "codinggmail.com" Output: False
4 #Example: Input: "coding@gmaiicom" Output: False
5 import re
6 def validate_email(email):
7     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}+$'
8     if re.match(pattern, email):
9         return True
10    else:
11        return False
12 # Get user input
13 user_input = input("Enter an email address: ")
14 # Validate the email and print the result
15 is_valid = validate_email(user_input)
16 print(is_valid)

PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & c:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/mailValidation.py"
Enter an email address: coding@gmail.com
True
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & c:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/mailValidation.py"
Enter an email address: coding
False
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & c:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/mailValidation.py"
Enter an email address: coding
False
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>

```

Justification:

- Zero-shot prompting gives basic and incomplete validation because no examples are provided.
- Few-shot prompting gives better logic and more accurate results by checking username and domain using examples.

Zero-shot -- Basic, incomplete validation

Few-shot -- Better logic, checks username & domain

Task-5: Prompt Tuning – Summing Digits of a Number

Prompt1: write a python function that returns the sum of digits of a number.

Code and Output:

The screenshot shows a terminal window with the following content:

```
3.4(sumOfDigits).py > ...
1 # write a python function that returns the sum of digits of a number.
2 def sum_of_digits(n):
3     total = 0
4     for digit in str(n):
5         total += int(digit)
6     return total
7 # Get user input
8 user_input = int(input("Enter a number: "))
9 # calculate the sum of digits and print the result
10 result = sum_of_digits(user_input)
11 print(result)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/AI(vs)/sumOfDigits.py"
Enter a number: 15
6
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/AI(vs)/sumOfDigits.py"
Enter a number: 15
13
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

0

Prompt2: #generate a python function to calculate sum_of_digits

#Example Input: sum_of_digits(123) → 6

Code and Output:

```
3.4(sum).py > ...
1 #generate a python function to calculate sum_of_digits
2 #Example Input: sum_of_digits(123) → 6
3 def sum_of_digits(n):
4     total = 0
5     for digit in str(n):
6         total += int(digit)
7     return total
8 # Get user input
9 user_input = int(input("Enter a number: "))
10 # Calculate the sum of digits and print the result
11 result = sum_of_digits(user_input)
12 print(result)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + ⌂ ⌂ ⌂
```

```
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/sum.py"
Enter a number: 12
3
PS C:\Users\akhil\OneDrive\Documents\AI(vs)> & C:/Users/akhil/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/akhil/OneDrive/Documents/sum.py"
Enter a number: 157
13
PS C:\Users\akhil\OneDrive\Documents\AI(vs)>
```

Justification:

- Prompt2 produced cleaner and optimized code
- Example guided Copilot to use Pythonic one-line solution
- Prompt tuning improves code quality and readability