

AI Assisted Coding

Assignment 3.4

Name: Mujtaba Ahmed

Hallticket No: 2303A510C2

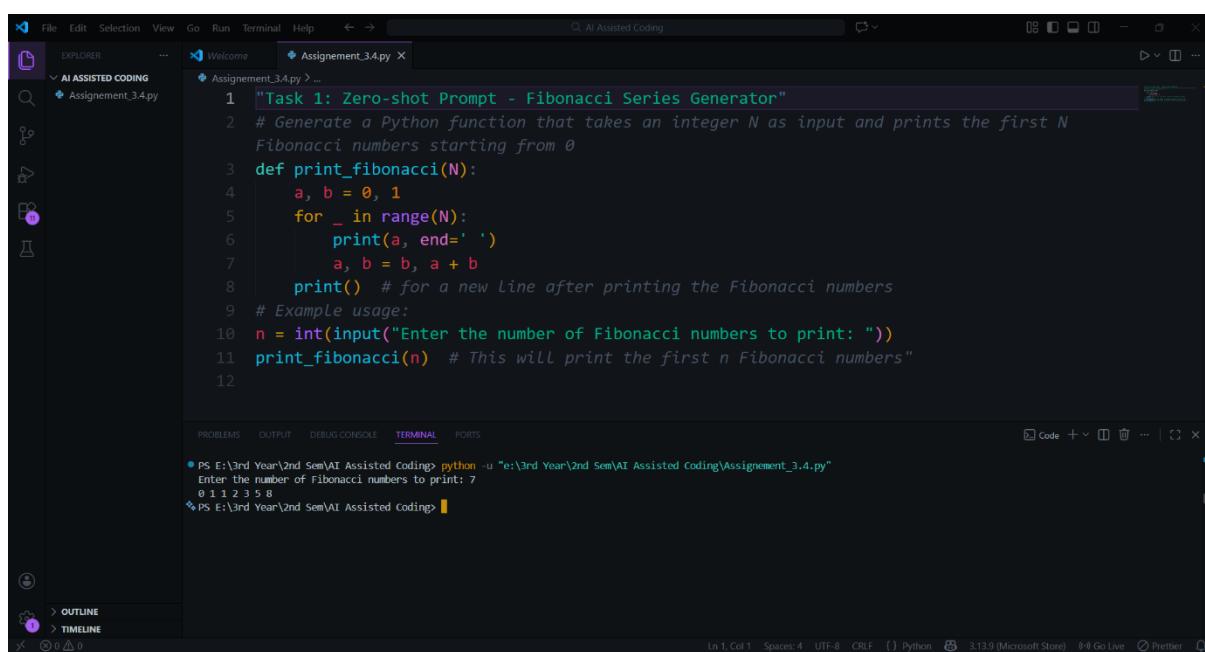
Batch Number : 19

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Prompt:

Generate a Python function that takes an integer N as input and prints the first N Fibonacci numbers starting from 0.

Code & Output:



The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a folder named "AI ASSISTED CODING" containing a file named "Assigment_3.4.py".
- Code Editor:** Displays the following Python code:

```
1  "#Task 1: Zero-shot Prompt - Fibonacci Series Generator"
2  # Generate a Python function that takes an integer N as input and prints the first N
   Fibonacci numbers starting from 0
3  def print_fibonacci(N):
4      a, b = 0, 1
5      for _ in range(N):
6          print(a, end=' ')
7          a, b = b, a + b
8      print() # for a new Line after printing the Fibonacci numbers
9  # Example usage:
10 n = int(input("Enter the number of Fibonacci numbers to print: "))
11 print_fibonacci(n) # This will print the first n Fibonacci numbers
12
```
- Terminal:** Shows the command line output of running the script:

```
PS E:\3rd Year\2nd Sem\AI Assisted coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assigment_3.4.py"
Enter the number of Fibonacci numbers to print: 7
0 1 1 2 3 5 8
PS E:\3rd Year\2nd Sem\AI Assisted coding>
```
- Status Bar:** Shows the following information: Ln 1, Col 1 | Spaces: 4 | UTF-8 | CR/LF | {} Python | 3.13.9 (Microsoft Store) | Go Live | Prettier

Explanation:

In this task, a zero-shot prompt was used where only the problem description was provided without any examples. Based on this instruction, the AI generated a function to print the Fibonacci series. The program starts with the initial values and iteratively calculates the next numbers in the sequence. This task demonstrates that the AI can correctly understand and solve a problem even when no examples are given.

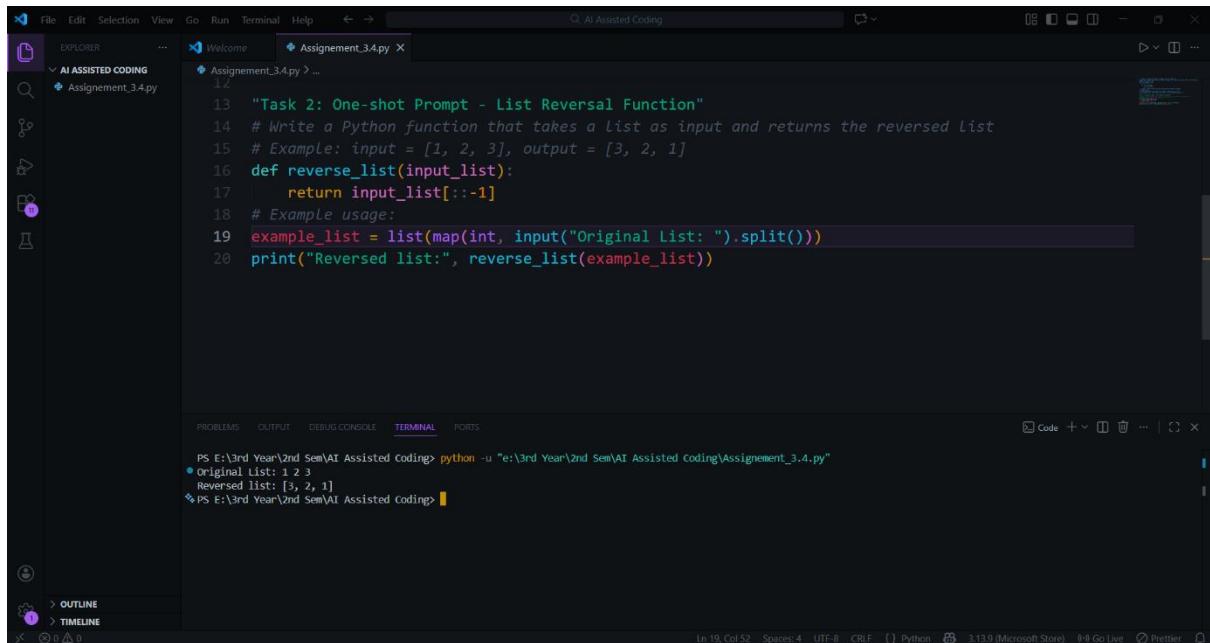
Task 2: One-shot Prompt – List Reversal Function

Prompt:

Write a Python function that takes a list as input and returns the reversed list.

Example: input = [1, 2, 3], output = [3, 2, 1]

Code & Output:



The screenshot shows the Visual Studio Code interface with the 'AI ASSISTED CODING' extension open. The code editor displays a Python script named 'Assignment_3.4.py' containing the following code:

```
12
13 "Task 2: One-shot Prompt - List Reversal Function"
14 # Write a Python function that takes a list as input and returns the reversed list
15 # Example: input = [1, 2, 3], output = [3, 2, 1]
16 def reverse_list(input_list):
17     return input_list[::-1]
18 # Example usage:
19 example_list = list(map(int, input("Original List: ").split()))
20 print("Reversed list:", reverse_list(example_list))
```

The terminal below shows the execution of the script and its output:

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
● Original list: 1 2 3
Reversed list: [3, 2, 1]
PS E:\3rd Year\2nd Sem\AI Assisted Coding>
```

Explanation:

In this task, a one-shot prompt was used by providing a single example along with the task description. The example helped the AI clearly understand the expected input and output format. As a result, the generated solution accurately reverses the list. This shows that adding one example improves the clarity and correctness of the AI-generated code.

Task 3: Few-shot Prompt – String Pattern Matching

Prompt:

Write a Python function `is_valid(s)` that returns True if a string starts with a capital letter and ends with a period.

Examples:

"Hello world." → True

"hello world." → False

"Hello world" → False

Code & Output:

The screenshot shows the Microsoft Visual Studio Code interface with the 'AI ASSISTED CODING' extension active. The 'TERMINAL' tab is selected, displaying a command-line session where the user runs a Python script named 'Assignment_3.4.py'. The script contains a function 'is_valid' that checks if a string starts with a capital letter and ends with a period. The terminal output shows three examples: 'Hello world.' (True), 'hello world.' (False), and 'Hello world' (False). The status bar at the bottom indicates the file is saved.

```
22
23 "Task 3: Few-shot Prompt - String Pattern Matching"
24 # Write a Python function is_valid(s) that returns True if a string starts with a capital letter
25 # and ends with a period, otherwise False
26 # Example: "Hello world." → True
27 # Example: "hello world." → False
28 # Example: "Hello world" → False
29 def is_valid(s):
30     return s[0].isupper() and s.endswith('.')
31 test_string = input("Enter a string: ")
32 print(is_valid(test_string)) # This will print True or False based on the input string

PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
● Enter a string: Hello world.
True
● PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
Enter a string: hello world.
False
● PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
Enter a string: Hello world
False
PS E:\3rd Year\2nd Sem\AI Assisted coding>
```

Explanation:

In this task, few-shot prompting was used by providing multiple examples. These examples guided the AI to identify both conditions correctly: the string must start with a capital letter and end with a period. The presence of multiple examples helped the AI generate a more precise and reliable solution compared to zero-shot or one-shot prompting.

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

Zero-shot Prompt:

Write a Python function to validate whether an email address is valid or not.

Code & Output:

```
34
35 "Task 4: Zero-shot vs Few-shot - Email Validator (Zero-shot Prompt)"
36 # Write a Python function to validate whether an email address is valid or not
37 def is_valid_email(email):
38     import re
39     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
40     return re.match(pattern, email) is not None
41 # Example usage:
42 email_input = input("Enter an email address: ")
43 if is_valid_email(email_input):
44     print("Valid email address")
45 else:
46     print("Invalid email address")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
● Enter an email address: sru@sru.edu.in
Valid email address.
● PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
Enter an email address: sru@mail.com
Invalid email address.
↳ PS E:\3rd Year\2nd Sem\AI Assisted coding>

Few-shot Prompt:

Write a Python function `is_valid_email(email)` that returns True for valid emails and False otherwise.

Examples:

"user@gmail.com" → True
"user123@yahoo.in" → True
"user@gmail.com" → False
"user@.com" → False

Code & Output:

```
48 "Task 4: Zero-shot vs Few-shot - Email Validator (Few-shot Prompt)"
49 # Write a Python function is_valid_email(email) that returns True for valid emails and False otherwise
50 # Example: "user@gmail.com" → True
51 # Example: "user123@yahoo.in" → True
52 # Example: "user@gmail.com" → False
53 # Example: "user@.com" → False
54 def is_valid_email(email):
55     import re
56     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
57     return re.match(pattern, email) is not None
58 # Example usage:
59 email_input = input("Enter an email address: ")
60 print(is_valid_email(email_input)) # This will print True or False based on the email validity

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
● Enter an email address: sru@sru.in
True
● PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
Enter an email address: sru@.in
False
↳ PS E:\3rd Year\2nd Sem\AI Assisted coding>

Explanation:

In the zero-shot prompt, the AI produced a basic email validation logic because no examples were provided. In contrast, the few-shot prompt included valid and invalid examples, which helped the AI understand the structure of an email address more clearly. As a result, the few-shot approach generated a more accurate and reliable email validation solution.

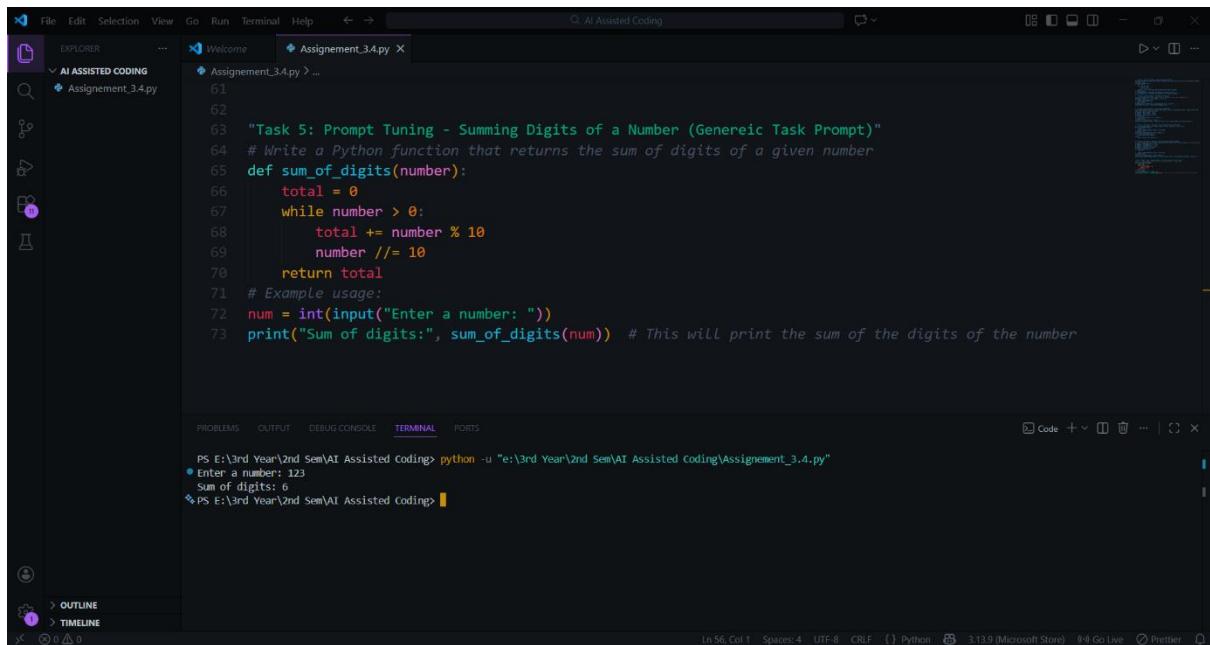
Task 5: Prompt Tuning – Summing Digits of a Number

Style 1: Generic Task Prompt

Prompt:

Write a Python function that returns the sum of digits of a given number.

Code & Output:



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar labeled "AI Assisted Coding". The Explorer sidebar on the left shows a folder named "AI ASSISTED CODING" containing a file "Assigment_3.4.py". The main editor area displays the following Python code:

```
61
62
63 "Task 5: Prompt Tuning - Summing Digits of a Number (Genereic Task Prompt)"
64 # Write a Python function that returns the sum of digits of a given number
65 def sum_of_digits(number):
66     total = 0
67     while number > 0:
68         total += number % 10
69         number //= 10
70     return total
71 # Example usage:
72 num = int(input("Enter a number: "))
73 print("Sum of digits:", sum_of_digits(num)) # This will print the sum of the digits of the number
```

Below the editor is a terminal window titled "TERMINAL". It shows the command "python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assigment_3.4.py"" being run, followed by the output: "Enter a number: 123", "Sum of digits: 6", and the prompt "PS E:\3rd Year\2nd Sem\AI Assisted Coding>". The bottom status bar indicates "Ln 56, Col 1" and "Python 3.13.9 (Microsoft Store)".

Style 2: Task + Input/Output Example Prompt

Prompt:

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

Example: input = 123, output = 6

Code & Output:

The screenshot shows the Microsoft Visual Studio Code interface with the "AI ASSISTED CODING" extension active. The code editor displays a Python file named "Assignment_3.4.py" containing the following code:

```
75
76
77 "Task 5: Prompt Tuning - Summing Digits of a Number (Task + Input/Output example)"
78 # Write a Python function sum_of_digits(n) that returns the sum of all digits in a number
79 # Example: input = 123, output = 6
80 def sum_of_digits_tuned():
81     return sum(int(digit) for digit in str(n))
82 # Example usage:
83 num_tuned = int(input("Enter a number: "))
84 print("Sum of digits (tuned):", sum_of_digits_tuned(num_tuned)) # This will print the sum of the
# digits of the number
```

The terminal below shows the execution of the script:

```
PS E:\3rd Year\2nd Sem\AI Assisted Coding> python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"
● Enter a number: 123
Sum of digits (tuned): 6
PS E:\3rd Year\2nd Sem\AI Assisted coding>
```

Explanation:

In this task, two different prompt styles were used. The generic prompt resulted in a straightforward solution, while the prompt with an input/output example produced a cleaner and more optimized implementation. This task highlights how prompt tuning can significantly improve code quality and readability.