

# AI Assisted Coding

## Assignment 3.4

Name: P. Vineeth Kumar

Hall ticket no: 2303A51256

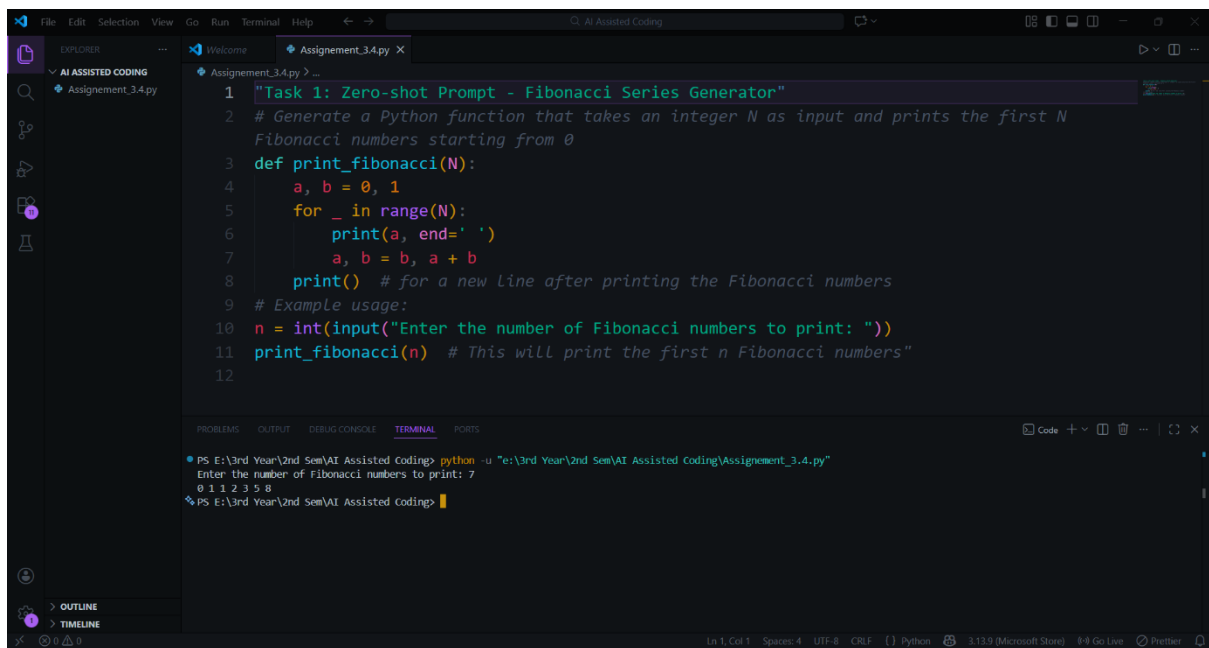
Batch no: 19

### 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:



```
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
```

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 the number of Fibonacci numbers to print: 7
0 1 1 2 3 5 8
PS E:\3rd Year\2nd Sem\AI Assisted Coding>
```

#### 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.

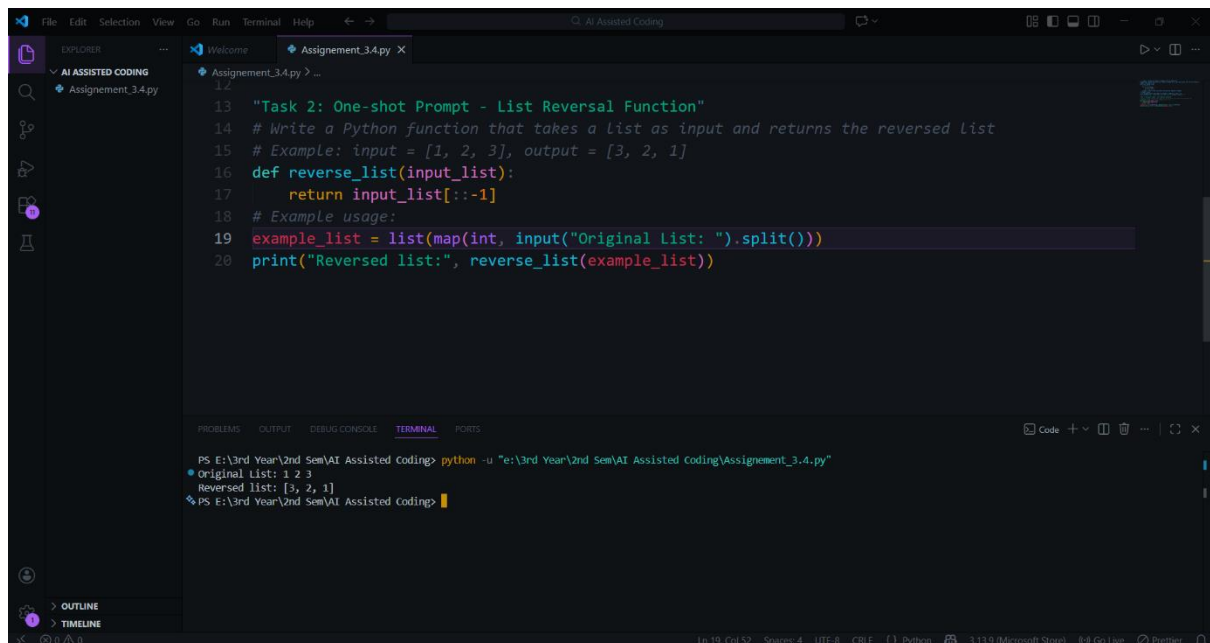
## Task 2: One-shot Prompt – List Reversal Function

### 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 a Visual Studio Code editor window with a file named 'Assignment\_3.4.py'. The code defines a function 'reverse\_list' that takes 'input\_list' and returns 'input\_list[::-1]'. It includes a comment about a one-shot prompt and an example usage where it takes user input, splits it into a list, and prints the reversed list. The terminal at the bottom shows the command 'python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment\_3.4.py"' being executed, with the output 'Original List: 1 2 3' and 'Reversed list: [3, 2, 1]'.

```
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))
```

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.

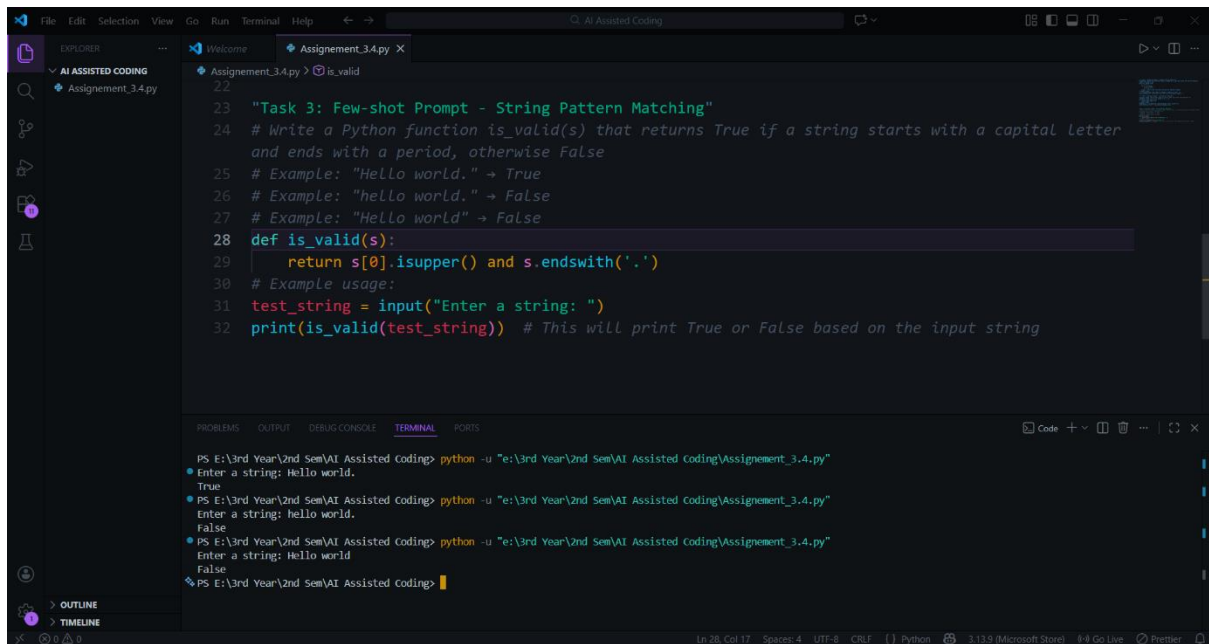
Examples:

"Hello world." → True

"hello world." → False

"Hello world" → False

### Code & Output:



The screenshot shows a Visual Studio Code editor window with a file named 'Assignment\_3.4.py'. The code defines a function `is_valid` that checks if a string starts with a capital letter and ends with a period. It includes three example prompts and their corresponding outputs. The terminal at the bottom shows the execution of the script, demonstrating the function's behavior for different inputs.

```
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
   and ends with a period, otherwise False
25 # Example: "Hello world." → True
26 # Example: "hello world." → False
27 # Example: "Hello world" → False
28 def is_valid(s):
29     return s[0].isupper() and s.endswith('.')
30 # Example usage:
31 test_string = input("Enter a string: ")
32 print(is_valid(test_string)) # This will print True or False based on the input string
```

Terminal Output:

```
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:

The screenshot shows a VS Code editor with a file named `Assignment_3.4.py`. The code defines a function `is_valid_email(email)` that uses a regular expression to validate email addresses. The prompt is labeled "Task 4: Zero-shot vs Few-shot - Email Validator (Zero-shot Prompt)". The terminal output shows the program running and validating two email addresses: `sr@gmail.com` (valid) and `sr@gmail` (invalid).

```
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")
```

Terminal Output:

```
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: sr@gmail.com
Valid email address
Enter an email address: sr@gmail
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

"usergmail.com" → False

"user@.com" → False

## Code & Output:

The screenshot shows a VS Code editor with a file named `Assignment_3.4.py`. The code defines a function `is_valid_email(email)` that uses a regular expression to validate email addresses. The prompt is labeled "Task 4: Zero-shot vs Few-shot - Email Validator (Few-shot Prompt)". The prompt includes four examples of valid and invalid email addresses. The terminal output shows the program running and validating four email addresses: `sr@gmail.com` (True), `sr@gmail.in` (True), `sr@gmail` (False), and `sr@.in` (False).

```
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: "usergmail.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
```

Terminal Output:

```
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: sr@gmail.com
True
Enter an email address: sr@gmail.in
True
Enter an email address: sr@gmail
False
Enter an email address: sr@.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.

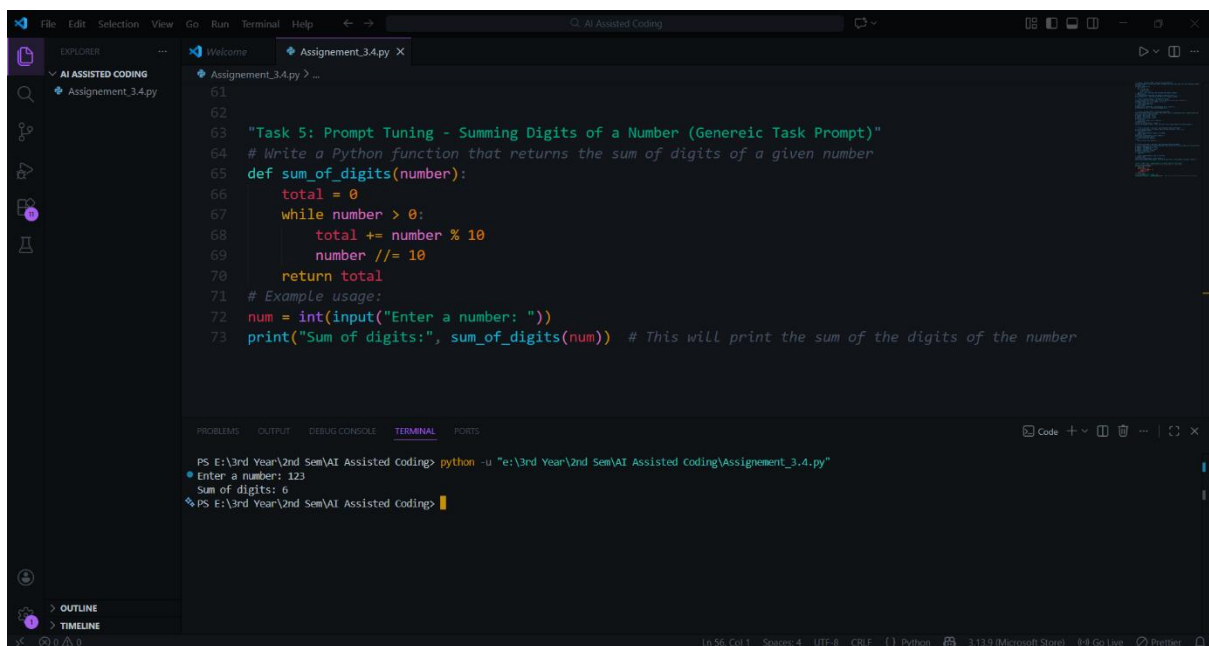
## Task 5: Prompt Tuning – Summing Digits of a Number

### Style 1: Generic Task Prompt

#### Prompt:

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

#### Code & Output:



```
61
62
63 "Task 5: Prompt Tuning - Summing Digits of a Number (Generic 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
```

```
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: 6
PS E:\3rd Year\2nd Sem\AI Assisted Coding>
```

### 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 a Visual Studio Code editor window with a file named 'Assignment\_3.4.py'. The code defines a function `sum_of_digits_tuned(n)` that calculates the sum of digits in a number. It includes a comment about prompt tuning and an example usage. The terminal at the bottom shows the command `python -u "e:\3rd Year\2nd Sem\AI Assisted Coding\Assignment_3.4.py"` being executed, with input '123' and output 'Sum of digits (tuned): 6'.

```
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(n):
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
```

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 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.