# AI Assisted Coding

Assignment 3.4

Name: V.Harivamsh

Hall ticket no: 2303A51266

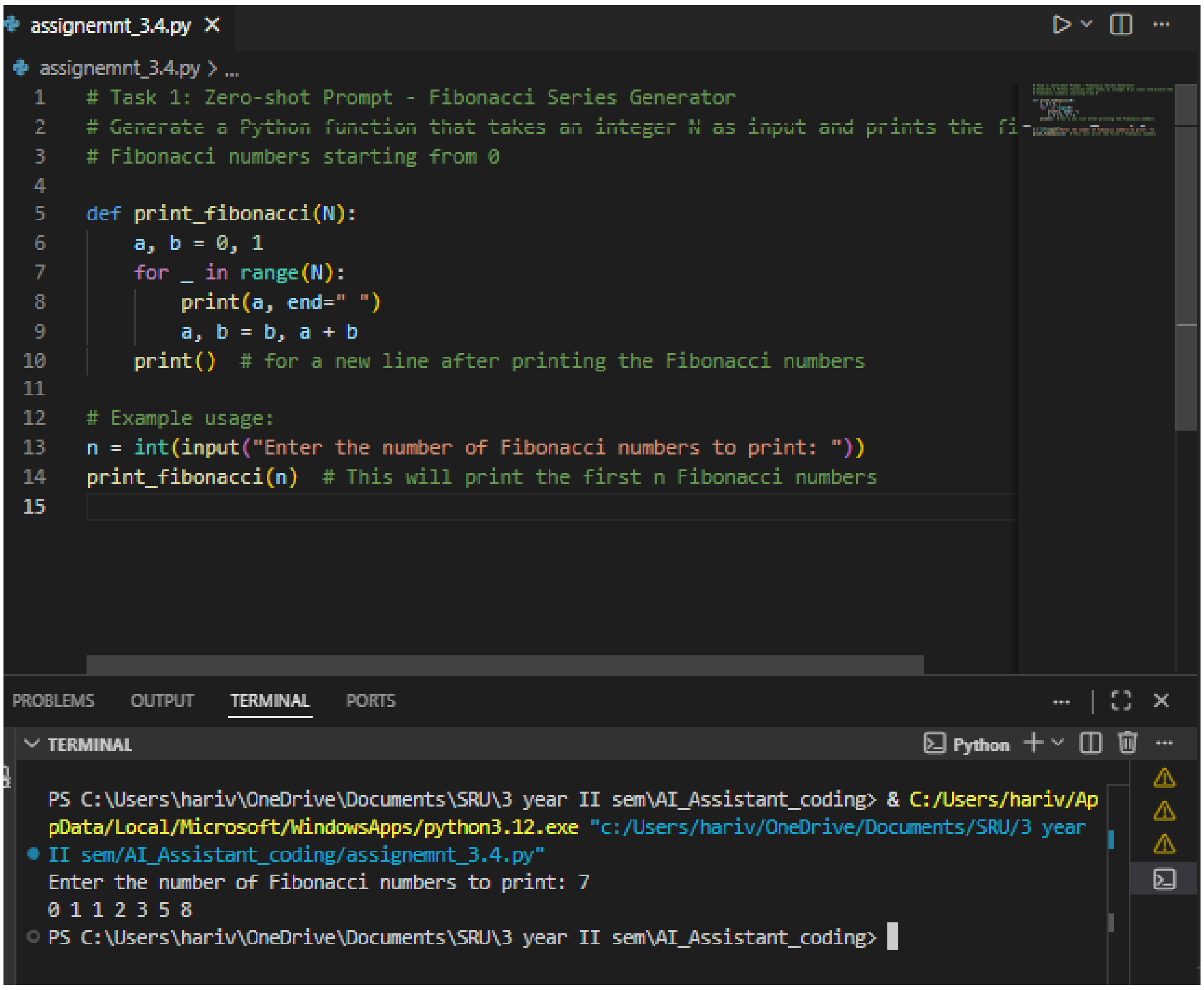
Batch no: 19

**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:**



```python
# Task 1: Zero-shot Prompt - Fibonacci Series Generator
# Generate a Python function that takes an integer N as input and prints the fi
# Fibonacci numbers starting from 0

def print_fibonacci(N):
    a, b = 0, 1
    for _ in range(N):
        print(a, end=" ")
        a, b = b, a + b
    print()  # for a new line after printing the Fibonacci numbers

# Example usage:
n = int(input("Enter the number of Fibonacci numbers to print: "))
print_fibonacci(n)  # This will print the first n Fibonacci numbers
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter the number of Fibonacci numbers to print: 7
0 1 1 2 3 5 8
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

**Explanation:**

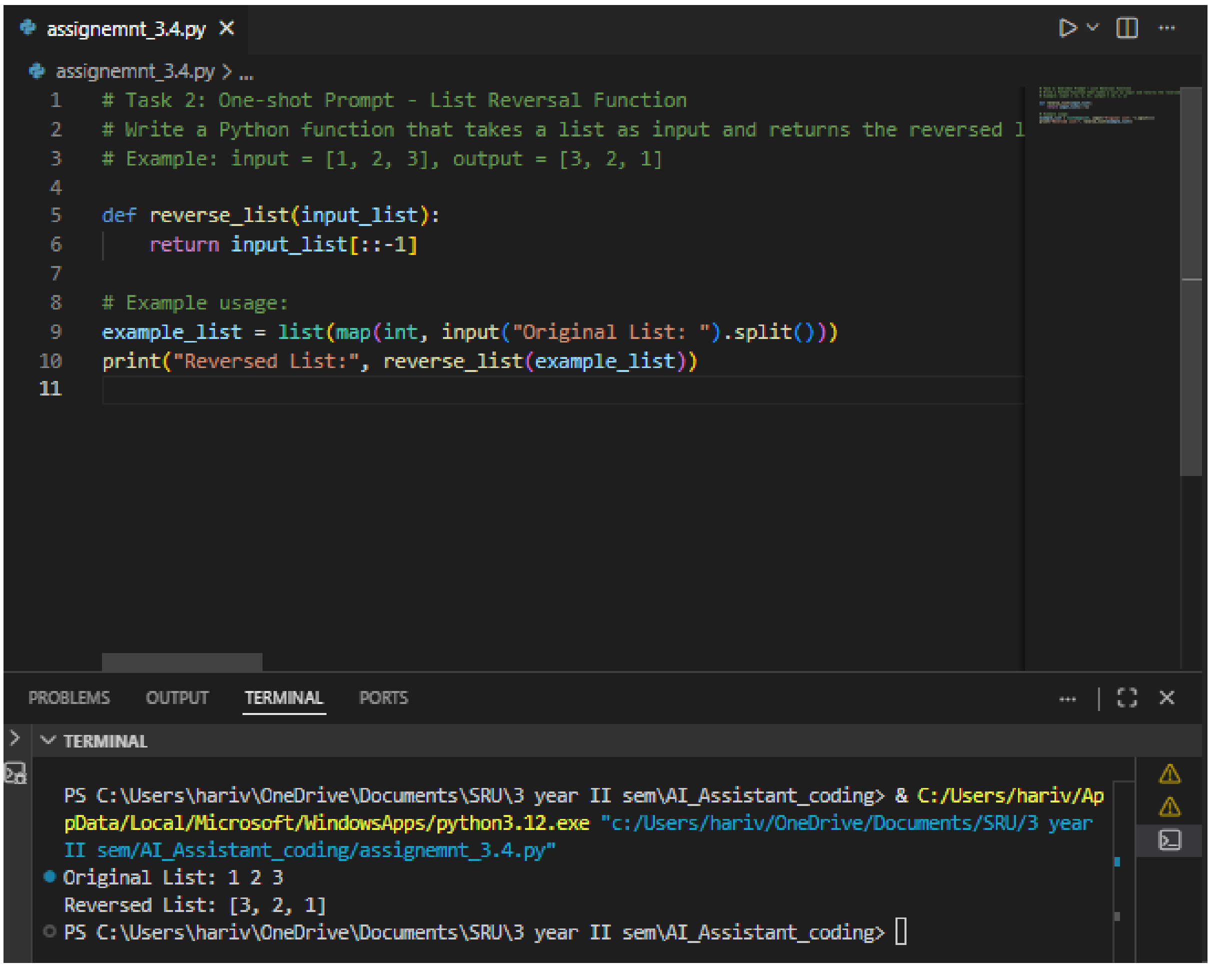
In this task, a zero-shot prompt was used, meaning only the problem statement was given without providing any examples. Using this instruction alone, the AI generated a function to display the Fibonacci series. The program begins with the initial values and then repeatedly computes the next numbers in the sequence. This shows that the AI is capable of understanding the problem and producing a correct solution even without example-based guidance.

## 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:**



```python
# Task 2: One-shot Prompt - List Reversal Function
# Write a Python function that takes a list as input and returns the reversed l
# Example: input = [1, 2, 3], output = [3, 2, 1]

def reverse_list(input_list):
    return input_list[::-1]

# Example usage:
example_list = list(map(int, input("Original List: ").split()))
print("Reversed List:", reverse_list(example_list))
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/Ap
pData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year
II sem/AI_Assistant_coding/assignemnt_3.4.py"
Original List: 1 2 3
Reversed List: [3, 2, 1]
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

**Explanation:**

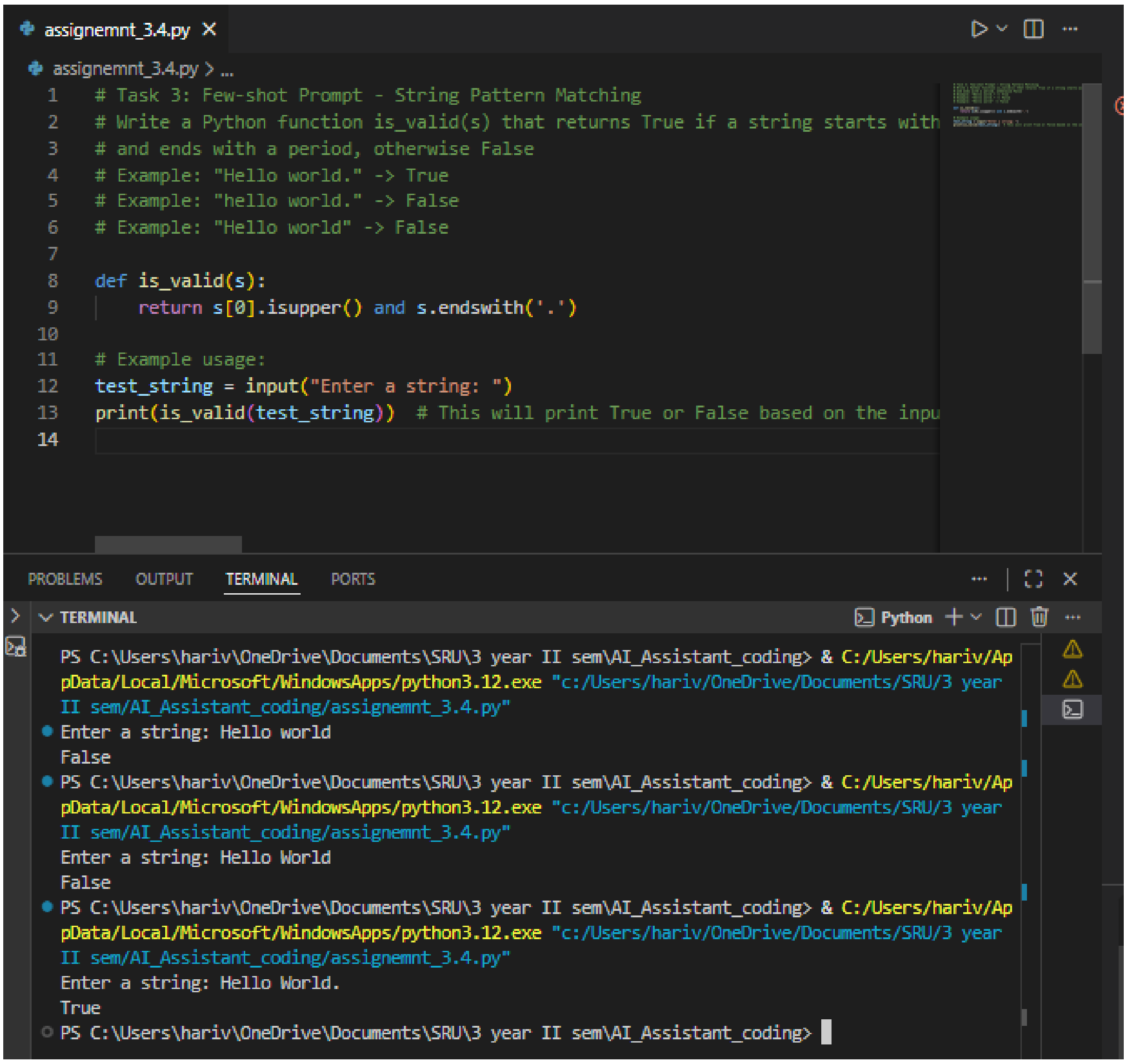
In this task, a one-shot prompt was used by including one example along with the problem description. This example helped the AI better understand the required input and output format. As a result, the generated solution correctly reverses the list. The task highlights how providing a single example can improve the clarity and accuracy of the AI's output.

**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:**



```python
# Task 3: Few-shot Prompt - String Pattern Matching
# Write a Python function is_valid(s) that returns True if a string starts with
# and ends with a period, otherwise False
# Example: "Hello world." -> True
# Example: "hello world." -> False
# Example: "Hello world" -> False

def is_valid(s):
    return s[0].isupper() and s.endswith('.')

# Example usage:
test_string = input("Enter a string: ")
print(is_valid(test_string))  # This will print True or False based on the inpu
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/Ap
pData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year
II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter a string: Hello world
False
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/Ap
pData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year
II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter a string: Hello World
False
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/Ap
pData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year
II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter a string: Hello World.
True
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

**Explanation:**

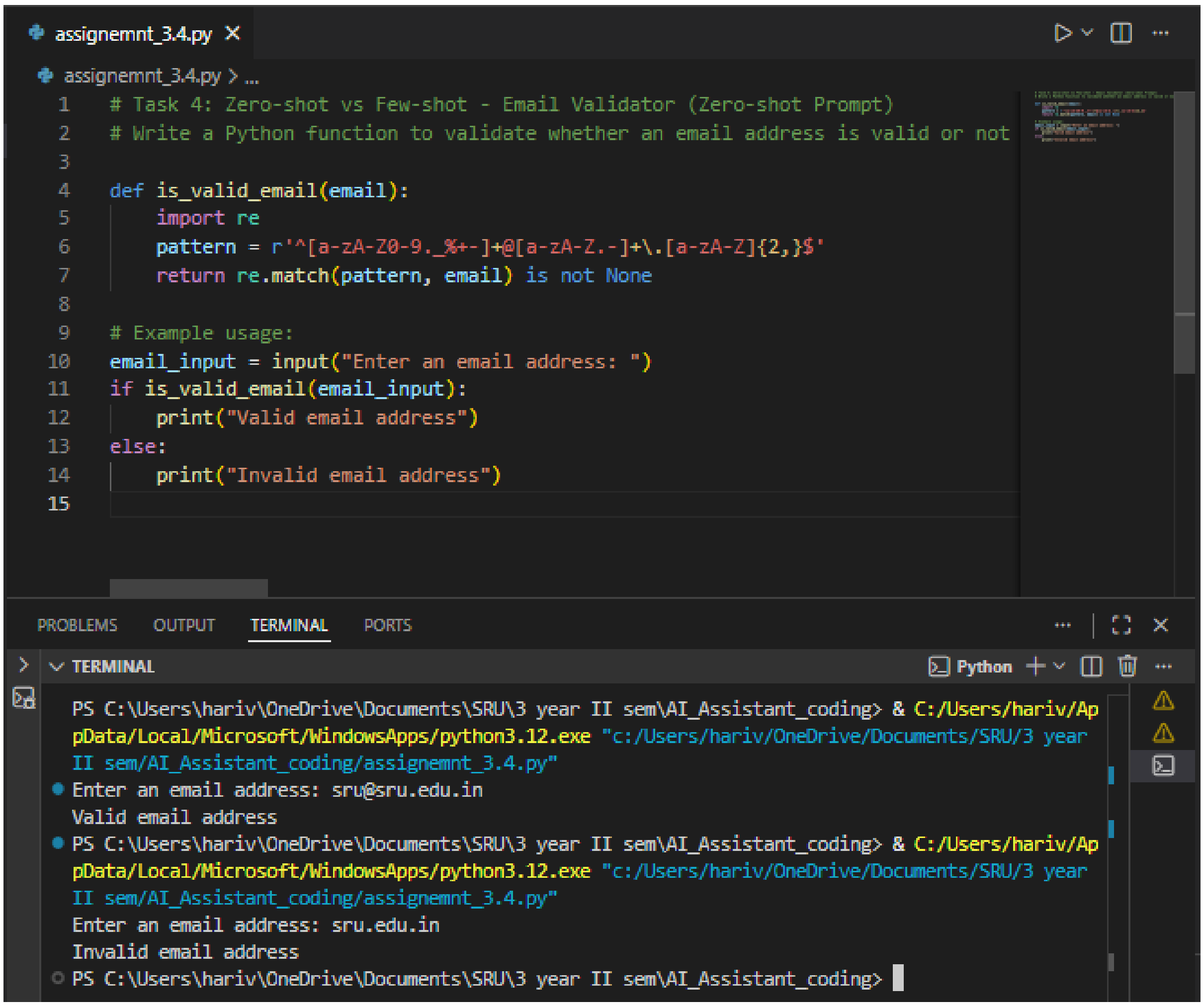
In this task, few-shot prompting was applied by giving the AI several examples. These examples helped it clearly recognize both required conditions: the string should begin with a capital letter and end with a period. Because of the multiple examples, the AI was able to produce a more accurate and dependable solution than with 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:**



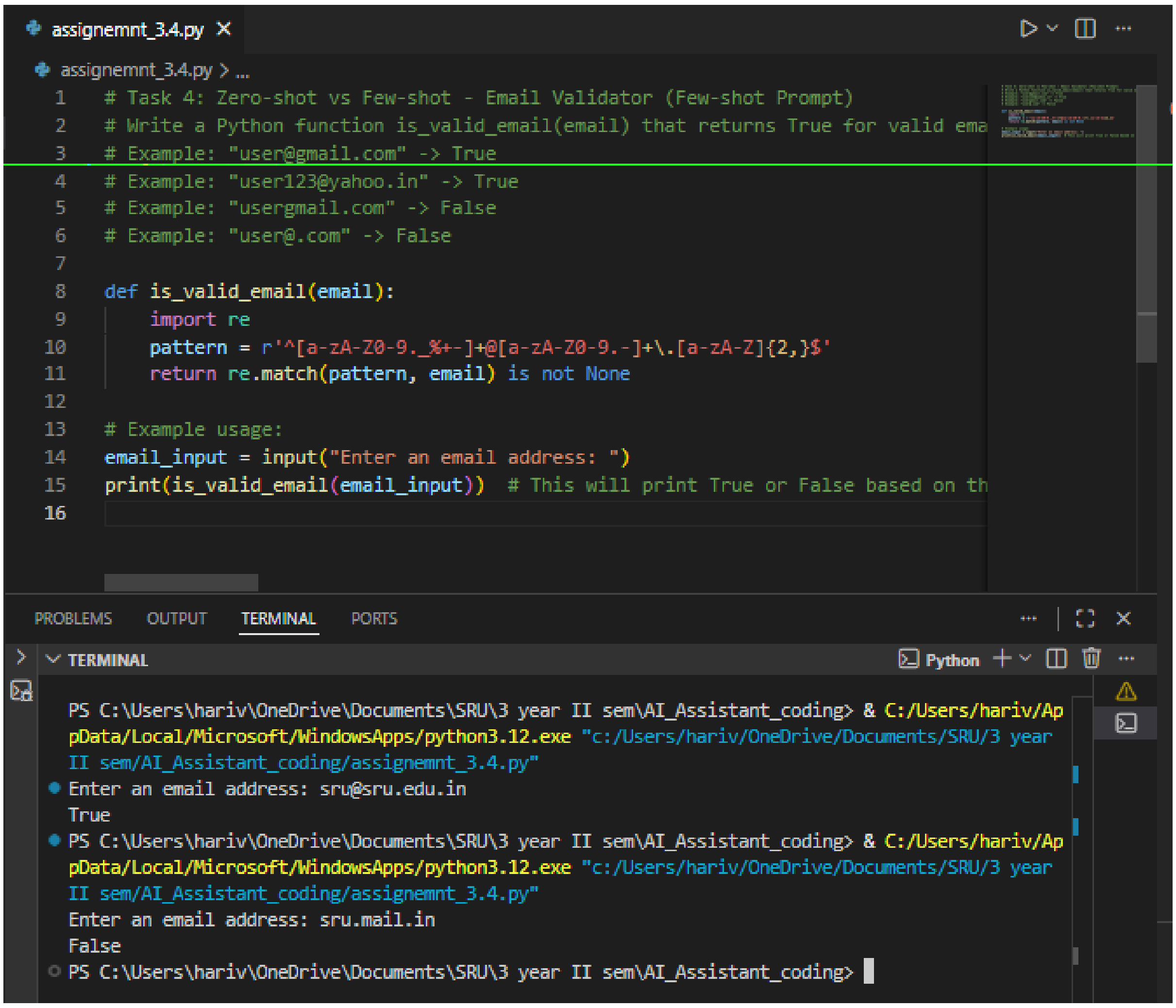
```python
# Task 4: Zero-shot vs Few-shot - Email Validator (Zero-shot Prompt)
# Write a Python function to validate whether an email address is valid or not

def is_valid_email(email):
    import re
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None

# Example usage:
email_input = input("Enter an email address: ")
if is_valid_email(email_input):
    print("Valid email address")
else:
    print("Invalid email address")

```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter an email address: sru@sru.edu.in
Valid email address
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter an email address: sru.edu.in
Invalid email address
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_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:**



```python
# Task 4: Zero-shot vs Few-shot - Email Validator (Few-shot Prompt)
# Write a Python function is_valid_email(email) that returns True for valid ema
# Example: "user@gmail.com" -> True
# Example: "user123@yahoo.in" -> True
# Example: "usergmail.com" -> False
# Example: "user@.com" -> False

def is_valid_email(email):
    import re
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None

# Example usage:
email_input = input("Enter an email address: ")
print(is_valid_email(email_input))  # This will print True or False based on th
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter an email address: sru@sru.edu.in
True
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter an email address: sru.mail.in
False
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

**Explanation:**

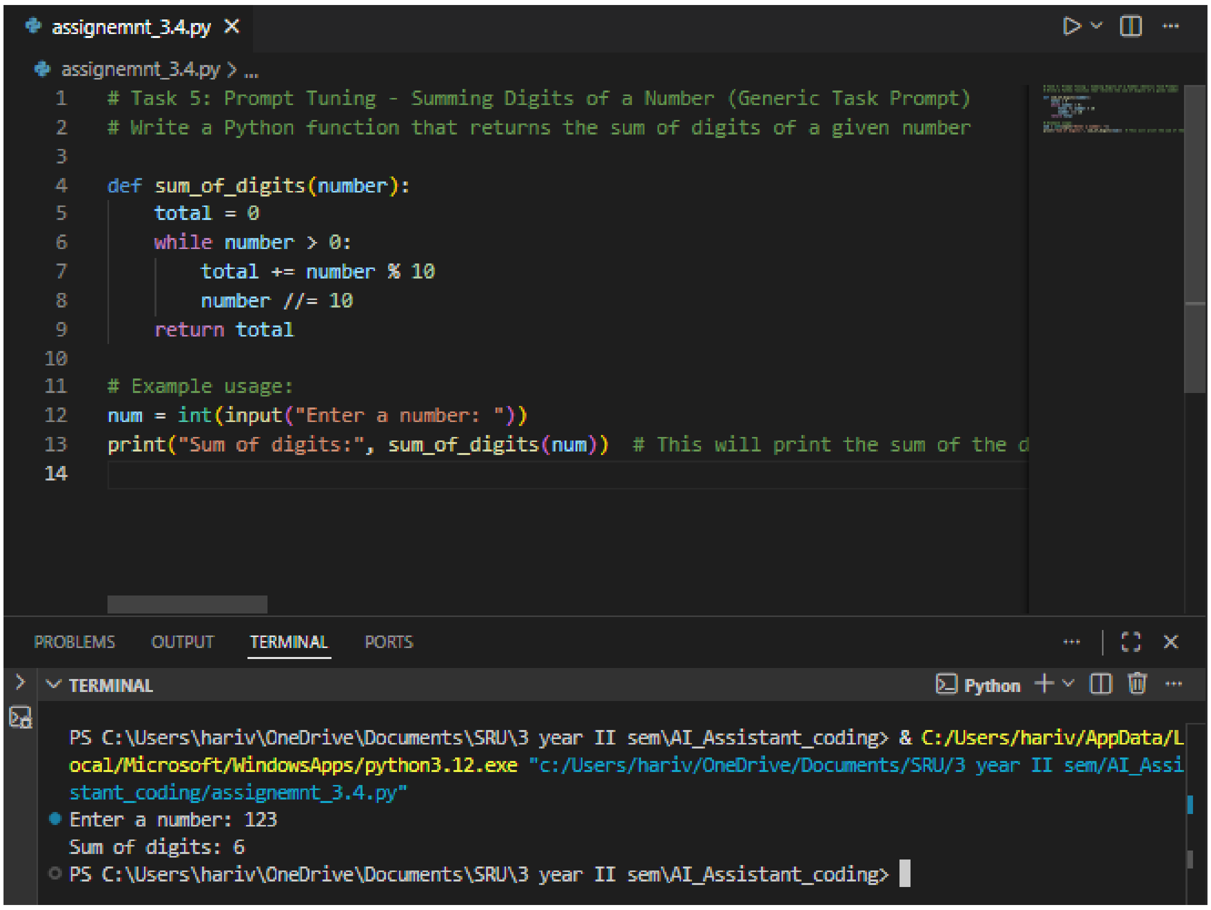
In the zero-shot prompt, the AI generated a simple email validation logic since no examples were given. However, in the few-shot prompt, both valid and invalid examples were provided, which helped the AI better understand the correct structure of an email address. Because of this added guidance, the few-shot approach resulted in 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:**



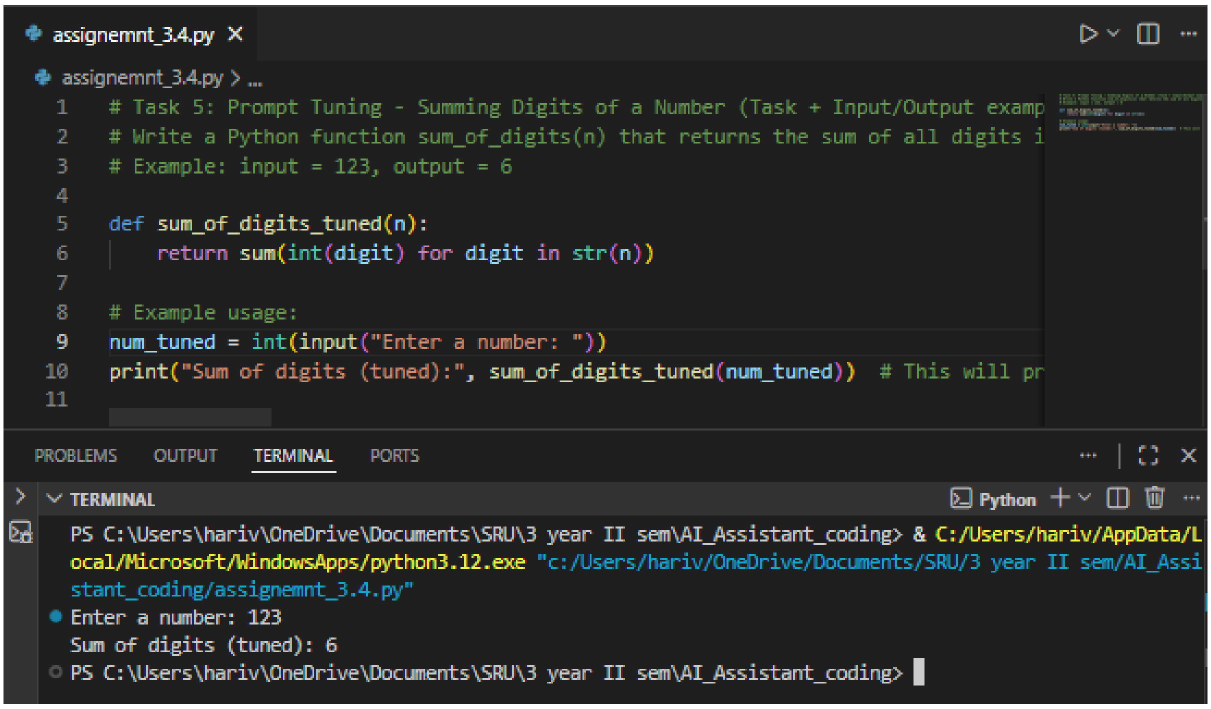
```python
# Task 5: Prompt Tuning - Summing Digits of a Number (Generic Task Prompt)
# Write a Python function that returns the sum of digits of a given number

def sum_of_digits(number):
    total = 0
    while number > 0:
        total += number % 10
        number //= 10
    return total

# Example usage:
num = int(input("Enter a number: "))
print("Sum of digits:", sum_of_digits(num))  # This will print the sum of the d

```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/L
ocal/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assi
stant_coding/assignemnt_3.4.py"
Enter a number: 123
Sum of digits: 6
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_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:**



```
# Task 5: Prompt Tuning - Summing Digits of a Number (Task + Input/Output examp
# Write a Python function sum_of_digits(n) that returns the sum of all digits i
# Example: input = 123, output = 6

def sum_of_digits_tuned(n):
    return sum(int(digit) for digit in str(n))

# Example usage:
num_tuned = int(input("Enter a number: "))
print("Sum of digits (tuned):", sum_of_digits_tuned(num_tuned))  # This will pr
```

```
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding> & C:/Users/hariv/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/hariv/OneDrive/Documents/SRU/3 year II sem/AI_Assistant_coding/assignemnt_3.4.py"
Enter a number: 123
Sum of digits (tuned): 6
PS C:\Users\hariv\OneDrive\Documents\SRU\3 year II sem\AI_Assistant_coding>
```

**Explanation:**

In this task, two different prompt styles were applied. The generic prompt led to a basic and direct solution, whereas the prompt that included an input/output example produced a cleaner and more optimized implementation. This clearly shows how refining a prompt can greatly enhance the quality and readability of the generated code.