

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:

The screenshot shows a code editor window with a Python file named 'assignemnt_3.4.py'. The code defines a function 'print_fibonacci' that generates a Fibonacci series up to a given number of terms. The terminal below shows the execution of the script and its output for 7 terms.

```
# Task 1: Zero-shot Prompt - Fibonacci Series Generator
# Generate a Python function that takes an integer N as input and prints the first N Fibonacci numbers starting from 0
# Example usage:
# Enter the number of Fibonacci numbers to print: 7
# 0 1 1 2 3 5 8
```

TERMINAL

```
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"
● 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:

The screenshot shows a code editor interface with a dark theme. At the top, there's a file tab labeled "assignment_3.4.py X". Below it is a code editor pane containing the following Python code:

```
# Task 2: One-shot Prompt - List Reversal Function
# Write a Python function that takes a list as input and returns the reversed list
# 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))
```

At the bottom of the screen, there's a terminal window titled "TERMINAL". It shows the command line output of running the script:

```
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/assignment_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:

The screenshot shows a code editor interface with a dark theme. At the top, there's a tab bar with 'assignemnt_3.4.py' and an 'X'. Below the tabs is a code editor area containing the following Python code:

```
(assignemnt_3.4.py)
# 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 input
```

Below the code editor is a terminal window titled 'TERMINAL'. It shows the following interaction:

```
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 string: Hello world
False
● 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 string: Hello World
False
● 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 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:

The screenshot shows the VS Code interface. The top part displays the code for `assignment_3.4.py`. The code defines a function `is_valid_email` that uses a regular expression to validate an email address. It then prompts the user for an email input and prints "Valid email address" if it's valid, or "Invalid email address" if it's not. The bottom part shows the terminal window with two runs of the script. In the first run, a valid email address (`sru@sru.edu.in`) is entered, resulting in "Valid email address". In the second run, an invalid email address (`sru.edu.in`) is entered, resulting in "Invalid email address".

```
assignment_3.4.py > ...
1 # Task 4: Zero-shot vs Few-shot - Email Validator (Zero-shot Prompt)
2 # Write a Python function to validate whether an email address is valid or not
3
4 def is_valid_email(email):
5     import re
6     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z.-]+\.[a-zA-Z]{2,}\$'
7     return re.match(pattern, email) is not None
8
9 # Example usage:
10 email_input = input("Enter an email address: ")
11 if is_valid_email(email_input):
12     print("Valid email address")
13 else:
14     print("Invalid email address")
15
```

TERMINAL

```
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/assignment_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/assignment_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:

The screenshot shows a VS Code interface with a code editor and a terminal window.

Code Editor:

```
# 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 emails
# Example: "user@gmail.com" -> True
# Example: "user123@yahoo.in" -> True
# Example: "user@gmail.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 the input
```

Terminal:

```
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/assignment_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/assignment_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:

The screenshot shows a code editor window with a dark theme. The file being edited is named 'assignment_3.4.py'. The code defines a function 'sum_of_digits' that takes a number as input and returns the sum of its digits. It uses a while loop to repeatedly take the remainder of the number divided by 10 and add it to a total, then divides the number by 10. An example usage is shown at the bottom, demonstrating how to run the function with user input.

```
assignment_3.4.py
assignment_3.4.py > ...
1 # Task 5: Prompt Tuning - Summing Digits of a Number (Generic Task Prompt)
2 # Write a Python function that returns the sum of digits of a given number
3
4 def sum_of_digits(number):
5     total = 0
6     while number > 0:
7         total += number % 10
8         number //= 10
9     return total
10
11 # Example usage:
12 num = int(input("Enter a number: "))
13 print("Sum of digits:", sum_of_digits(num)) # This will print the sum of the d
14
```

Below the code editor is a terminal window. The terminal tab is selected. The terminal shows the command 'python assignment_3.4.py' being run, followed by the prompt 'Enter a number:'. The user inputs '123', and the terminal outputs 'Sum of digits: 6'.

TERMINAL

```
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/assignment_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:

The screenshot shows a code editor interface with a Python file named `assignment_3.4.py`. The code defines a function `sum_of_digits_tuned` that calculates the sum of digits of a given number by converting it to a string and summing the integer values of each character. The terminal below shows the script being run and a sample input of 123 resulting in an output of 6.

```
assignment_3.4.py X
assignment_3.4.py > ...
1 # Task 5: Prompt Tuning - Summing Digits of a Number (Task + Input/Output example)
2 # Write a Python function sum_of_digits(n) that returns the sum of all digits in n
3 # Example: input = 123, output = 6
4
5 def sum_of_digits_tuned(n):
6     return sum(int(digit) for digit in str(n))
7
8 # Example usage:
9 num_tuned = int(input("Enter a number: "))
10 print("Sum of digits (tuned):", sum_of_digits_tuned(num_tuned)) # This will print 6
11
```

TERMINAL

```
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/assignment_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.