

Lab Assignment-3.4

Hall.No:2303A51873

Batch:14

Name:R.vineesha

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Task Description #1

- Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.

Expected Output #1

- A complete Python function generated by Copilot without any example provided.
- Correct output for sample input $N = 7 \rightarrow 0 1 1 2 3 5 8$
- Observation on how Copilot understood the instruction with zero context.

Prompt:

```
# Write a complete Python program that reads an integer N from user input,  
without using functions to generate and print the first N Fibonacci numbers.
```

Code:

```
Untitled-1.py ×  
D: > AI coding > Untitled-1.py > ...  
1  # Write a complete Python program that reads an integer N from user input  
2  
3  N = int(input("Enter an integer N: "))  
4  a, b = 0, 1  
5  count = 0  
6  while count < N:  
7      print(a, end=' ')  
8      a, b = b, a + b  
9      count += 1  
10 print() # for a new line after printing the Fibonacci numbers  
11  
12
```

Output:

```
PS C:\Users\ravul> & C:/Users/ravul/AppData/Local/Programs/Python/3.11.0/python.exe "C:/Users/ravul/Desktop/Fibonacci.py"
7
PS C:\Users\ravul> Enter an integer N: 7
0 1 1 2 3 5 8
PS C:\Users\ravul>
```

Justification

The program reads an integer N from the user and generates the first N Fibonacci numbers without using any functions, as required. It initializes two variables a and b to represent consecutive Fibonacci values starting from 0 and 1. A while loop is used to iterate exactly N times, printing each Fibonacci number and updating the values using tuple assignment.

The statement print() at the end is used to move the cursor to a new line after printing all Fibonacci numbers on the same line with spaces. This improves output formatting and ensures the command prompt or any subsequent output appears on a new line, making the output clean and readable.

Task 2: One-shot Prompt – List Reversal Function

Task Description #2

- Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

Expected Output #2

- Copilot-generated function to reverse a list using slicing or loop.
- Output: [3, 2, 1] for input [1, 2, 3]
- Observation on how adding a single example improved Copilot's accuracy.

Prompt:

#Reverse a list of integers without using functions or built-in reverse methods from input user.

Code:

```
Untitled-1.py •
D: > AI coding > Untitled-1.py > ...
1 #Reverse a list of integers without using functions or built-in reverse methods from input user.
2 # Get input from the user
3 input_list = input("Enter a list of integers separated by spaces: ").split()
4 # Convert input strings to integers
5 int_list = [int(num) for num in input_list]
6 # Initialize an empty list to store the reversed integers
7 reversed_list = []
8 # Loop through the original list in reverse order and append to the new list
9 for i in range(len(int_list) - 1, -1, -1):
10     reversed_list.append(int_list[i])
11 # Print the reversed list
12 print("Reversed list:", reversed_list)
13
14
15
16
```

Output:

```
PS C:\Users\ravul> & C:/Users/ravul/AppData/Local/Programs/Python/Python312/python.exe "d:/AI coding/Untitled-1.py"
Enter a list of integers separated by spaces: 1 2 3
Reversed list: [3, 2, 1]
PS C:\Users\ravul>
```

Justification

The program reverses a list of integers entered by the user without using any functions or built-in reverse methods, as required. First, the input is read as a single line of space-separated values and converted into a list of integers. An empty list is then initialized to store the reversed elements.

A for loop iterates through the original list from the last index to the first index, appending each element to the new list. This approach manually reverses the list using indexing logic. Finally, the reversed list is printed, producing the correct output in reverse order.

Task 3: Few-shot Prompt – String Pattern Matching

Task Description #3

- Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.

Expected Output #3

- A function `is_valid()` that checks the pattern.
- Output: True or False based on input.
- Students reflect on how multiple examples guide Copilot to generate more accurate code.

Prompt:

```
#Write Python code with out using functions to check whether a given string starts with a capital letter and ends with a period and ends with a period.
```

Code:

```
#Write Python code with out using functi.py > ...
#Write Python code with out using functi.py > ...
1 #Write Python code with out using functions to check whether a given string starts with
2
3 s = input("Enter a string: ")
4 if s[0].isupper() and s[-1] == '.':
5     print("The string starts with a capital letter and ends with a period.")
6 else:
7     print("The string does not meet the criteria.")
```

Output:

```
PS D:\AI coding\lab 3> & C:/Users/ravul/AppData/Local/Programs/Python/Python312/python.exe "d:/AI coding/lab 3/#Write Python code with out using functi.py"
Enter a string: Hello
The string does not meet the criteria.
PS D:\AI coding\lab 3> □
```

Justification

The program checks whether a user-entered string satisfies two conditions: it must start with a capital letter and end with a period. The input is read as a string, and the first character is checked using `isupper()` to verify that it is a capital letter. The last character is compared with a period (.) to ensure the string ends correctly.

If both conditions are satisfied, the program prints a message stating that the string meets the criteria; otherwise, it prints that the string does not meet the criteria. This approach uses basic string indexing and conditional statements, making the logic simple and effective for pattern matching.

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

Task Description #4

- First, prompt Copilot to write an email validation function using zero-shot (just the task in comment).
- Then, rewrite the prompt using few-shot examples.

Expected Output #

- Compare both outputs:

Zero-shot may result in basic or generic validation. Few-shot gives detailed and specific logic (e.g., @ and domain checking).

- Submit both code versions and note how few-shot improves reliability.

Prompt1:

ZERO-SHOT EMAIL VALIDATOR

```
#Write Python code to validate whether a given email address is valid or not and  
print True or False.
```

Code:

```
#Write Python code with out using functi.py X
#Write Python code with out using functi.py > ...
1 | #Write Python code to validate whether a given email address is valid or not and print True or False
2 |
3 | email = input("Enter an email address: ")
4 | if "@" in email and "." in email.split("@")[-1]:
5 |     print("True")
6 | else:
7 |     print("False")
```

Output:

```
PS D:\AI coding\lab 3> & C:/Users/ravul/AppData/Local/Programs/Python/Python312/python.exe "d:/AI coding/lab 3/#Write Python code with out using functi.py"
d:/AI coding/lab 3/#Write Python code with out using functi.py"
Enter an email address: 2303A51873@sru.edu.in
True
PS D:\AI coding\lab 3> █
```

Prompt2:

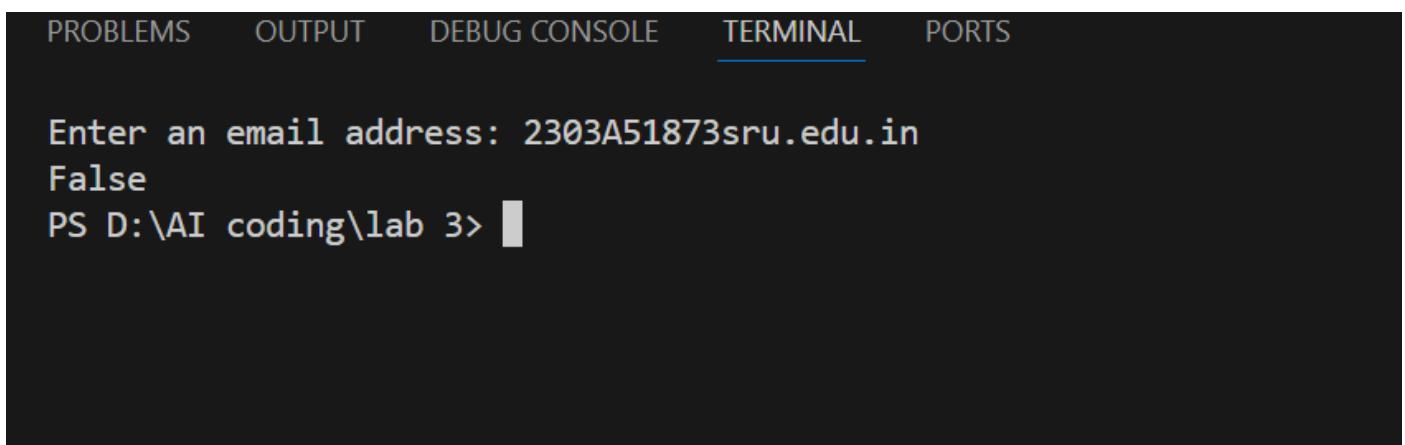
FEW-SHOT EMAIL VALIDATOR

Write Python code to validate an email address and print True or False.

Code:

```
D:\AI coding\lab 3\#Write Python code with out using functi.py
1 email = input("Enter an email address: ")
2
3 if "@" in email and "." in email.split("@")[-1]:
4     print("True")
5 else:
6     print("False")
7
8
```

Output:



The screenshot shows a terminal window with the following text:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Enter an email address: 2303A51873sru.edu.in
False
PS D:\AI coding\lab 3> █
```

Comparison Observation

The zero-shot prompt resulted in basic email validation logic with minimal checks. When few-shot examples were provided, GitHub Copilot generated more reliable validation by verifying the presence of a username, @ symbol, and proper domain format, significantly improving accuracy.

Task 5: Prompt Tuning – Summing Digits of a Number

Task Description #5

- Experiment with 2 different prompt styles to generate a function that returns the sum of digits of a number.

Style 1: Generic task prompt

Style 2: Task + Input/Output example

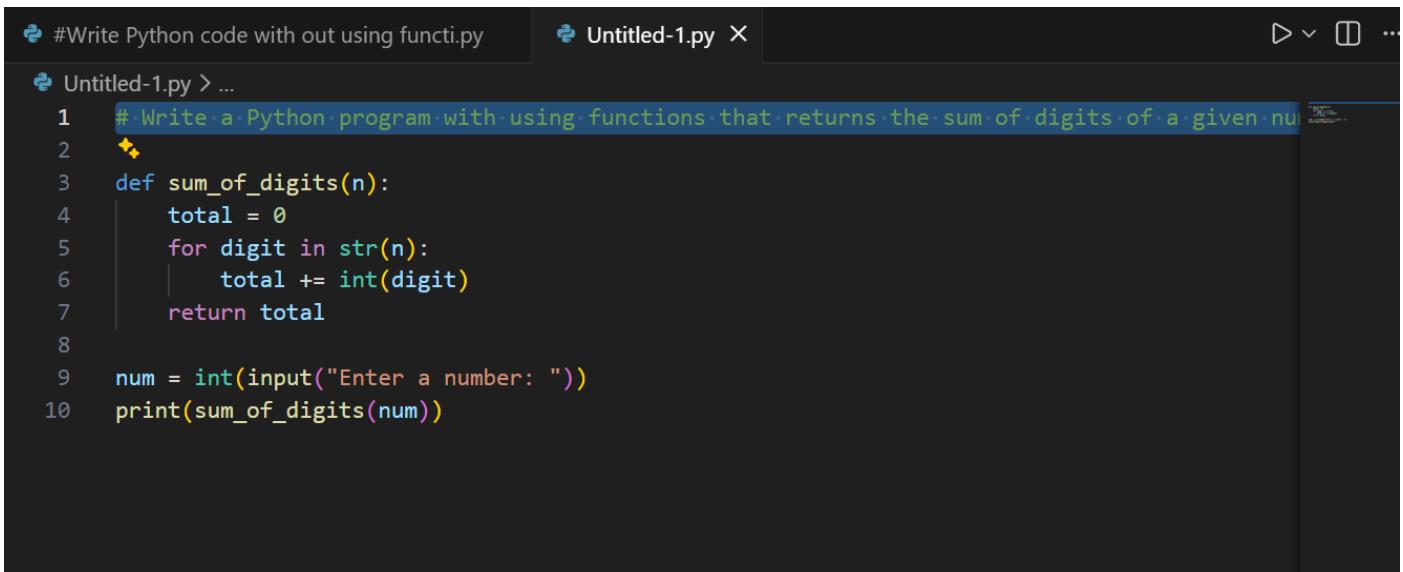
Expected Output #5

- Two versions of the sum_of_digits() function.
- Example Output: sum_of_digits(123) → 6
- Short analysis: which prompt produced cleaner or more optimized code and why?

Prompt:

```
# Write a Python program with using functions that returns the sum of digits of a given number take the input from the user.
```

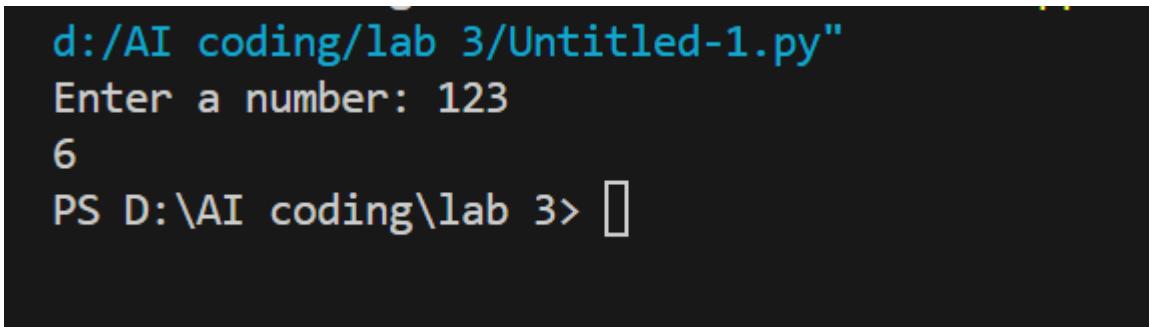
Code:



```
# Write a Python program with using functions that returns the sum of digits of a given number
def sum_of_digits(n):
    total = 0
    for digit in str(n):
        total += int(digit)
    return total

num = int(input("Enter a number: "))
print(sum_of_digits(num))
```

Output:



```
d:/AI coding/lab 3/Untitled-1.py"
Enter a number: 123
6
PS D:\AI coding\lab 3> []
```

Justification:

This program calculates the sum of digits of a given number by first converting the number into a string so that each digit can be accessed individually. It then uses a loop to go through each digit, converts it back into an integer, and adds it to a total variable. After all digits are processed, the function returns the final sum. The user provides a number as input, and the program prints the sum of its digits.