

AI ASSISTED CODING

Lab Assignment-3.4

Name : T.Shylasri

H.T.NO:2303A51876

Batch-14(LAB-4)

Date:22-01-2026

ASSIGNMENT-3.4

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.

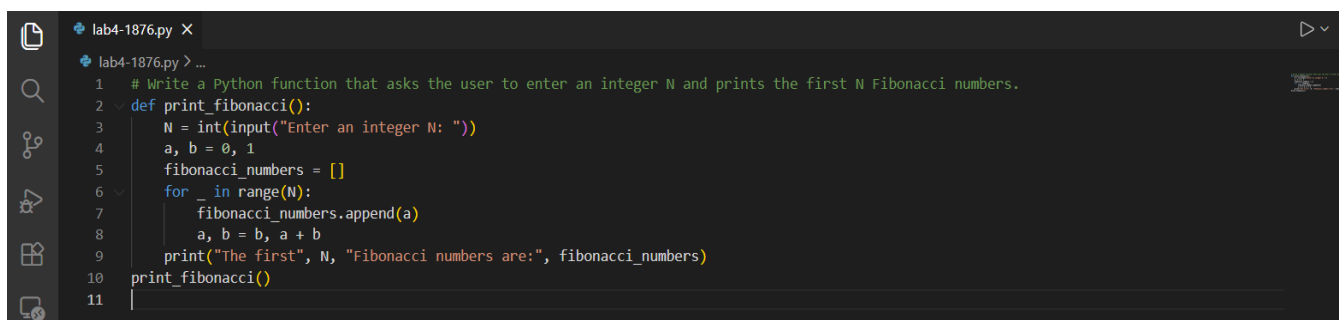
Expected Output #1

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

PROMPT:

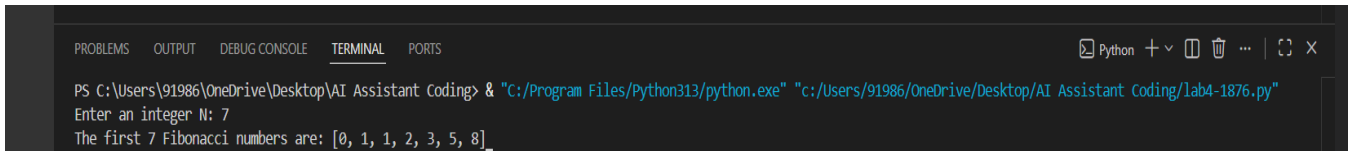
Generate a Python function that asks the user to enter an integer N and prints the first N Fibonacci numbers.

Code:

A screenshot of a code editor window titled 'lab4-1876.py'. The editor shows a Python function named 'print_fibonacci()' that prompts the user for an integer N and prints the first N Fibonacci numbers. The code is as follows:

```
1 # Write a Python function that asks the user to enter an integer N and prints the first N Fibonacci numbers.
2 def print_fibonacci():
3     N = int(input("Enter an integer N: "))
4     a, b = 0, 1
5     fibonacci_numbers = []
6     for _ in range(N):
7         fibonacci_numbers.append(a)
8         a, b = b, a + b
9     print("The first", N, "Fibonacci numbers are:", fibonacci_numbers)
10 print_fibonacci()
11
```

OUTPUT:



```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter an integer N: 7
The first 7 Fibonacci numbers are: [0, 1, 1, 2, 3, 5, 8]
```

Justification:

Zero-shot prompting evaluates Copilot's ability to understand and solve a problem using **only a textual instruction without any examples**. In this task, no sample input or output was provided. Copilot successfully generated the Fibonacci logic because the Fibonacci series is a well-known and frequently encountered problem. This shows that zero-shot prompts are effective when the task is standard and widely recognized. However, zero-shot prompting may be less reliable for complex or ambiguous problems.

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:

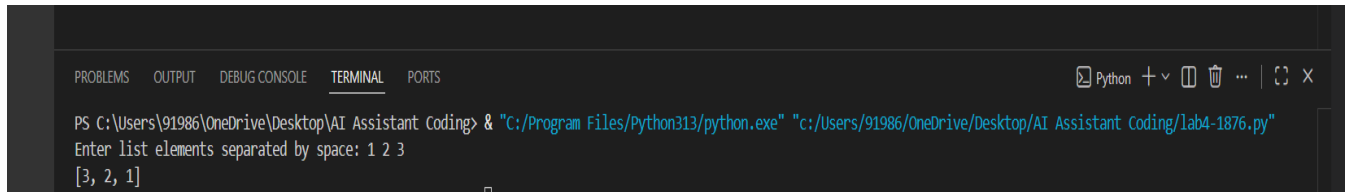
Generate a Python function that takes a list input from the user and reverses it.

Code:



```
lab4-1876.py X
lab4-1876.py > ...
1 # Write a Python function that takes a list input from the user and reverses it.
2 def reverse_list():
3     lst = list(map(int, input("Enter list elements separated by space: ").split()))
4     print(lst[::-1])
5 reverse_list()
```

Output:



```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter list elements separated by space: 1 2 3
[3, 2, 1]
```

Justification:

One-shot prompting improves accuracy by providing a **single example** along with the instruction. The example clarifies the expected behavior of the function, such as input format and output structure. In this task, the example $[1, 2, 3] \rightarrow [3, 2, 1]$ guided Copilot to correctly choose an appropriate list reversal technique. Compared to zero-shot prompting, one-shot prompting reduces misunderstanding and ensures the generated code aligns with user expectations.

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

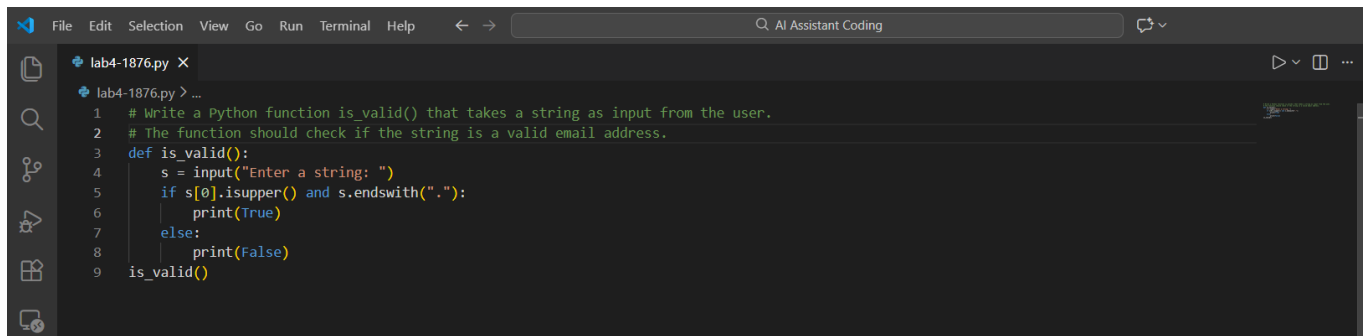
- A function `is_valid()` that checks the pattern.
- Output: True or False based on input.
- Students reflect on how multiple examples guide Copilot to generate more accurate code.

Prompt:

Write a Python function `is_valid()` that takes a string as input from the user.

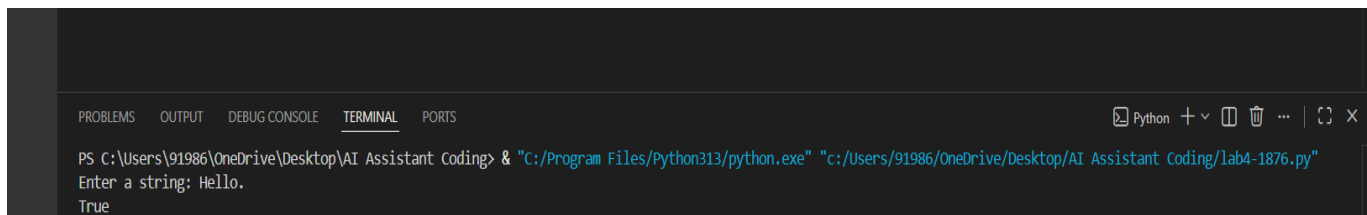
The function should check if the string is a valid email address.

CODE:

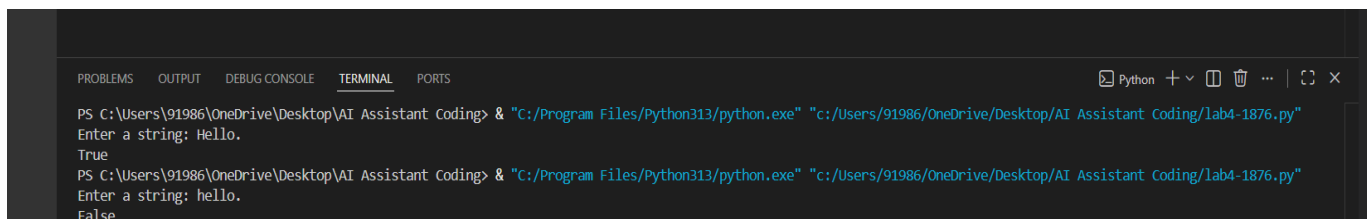


```
1 # Write a Python function is_valid() that takes a string as input from the user.
2 # The function should check if the string is a valid email address.
3 def is_valid():
4     s = input("Enter a string: ")
5     if s[0].isupper() and s.endswith("."):
6         print(True)
7     else:
8         print(False)
9 is_valid()
```

OUTPUT:



```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter a string: Hello.
True
```



```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter a string: Hello.
True
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter a string: hello.
False
```

JUSTIFICATION:

Few-shot prompting supplies **multiple examples**, allowing Copilot to identify patterns and rules more clearly. In this task, several valid and invalid string examples were given. These examples helped Copilot understand both **positive and negative cases**, such as capitalization and punctuation rules. As a result, the generated function accurately validates the string pattern. Few-shot prompting is especially useful for conditional logic and pattern-based problems where rules must be inferred from examples.

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 #4

- Compare both outputs:

Zero-shot may result in basic or generic validation.

Few-shot gives detailed and specific logic (e.g., @ and domain checking).

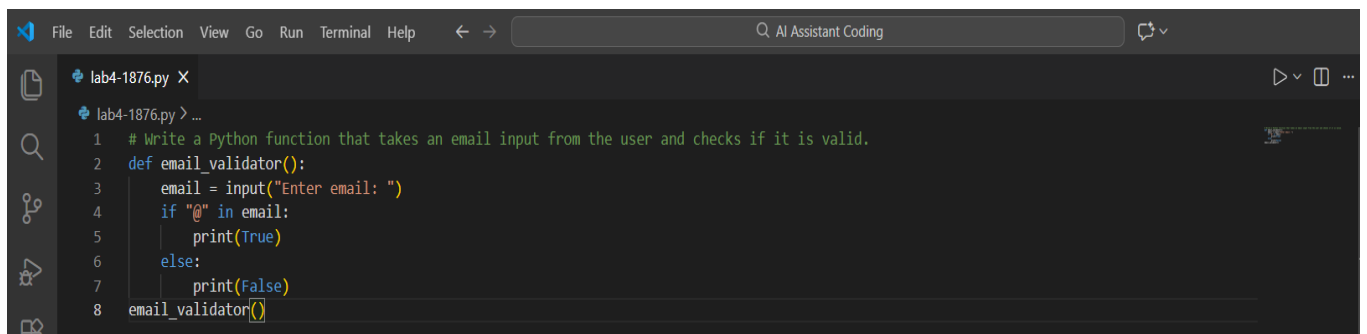
- Submit both code versions and note how few-shot improves reliability.

Part A: Zero-shot Prompt

PROMPT:

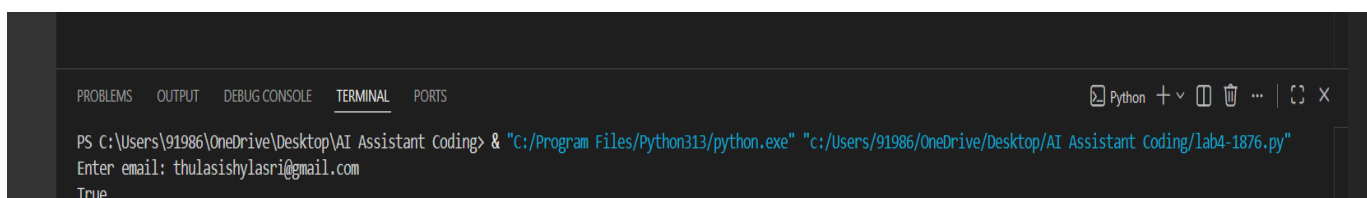
Write a Python function that takes an email input from the user and checks if it is valid.

CODE:

A screenshot of a code editor window. The title bar shows 'lab4-1876.py X'. The editor contains the following Python code:

```
1 # Write a Python function that takes an email input from the user and checks if it is valid.
2 def email_validator():
3     email = input("Enter email: ")
4     if "@" in email:
5         print(True)
6     else:
7         print(False)
8 email_validator()
```

OUTPUT:

A screenshot of a terminal window. The title bar shows 'Python + - [icon] [icon] [icon] [icon] X'. The terminal content is as follows:

```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter email: thulasishylasri@gmail.com
True
```

Part B: Few-shot Prompt

PROMPT:

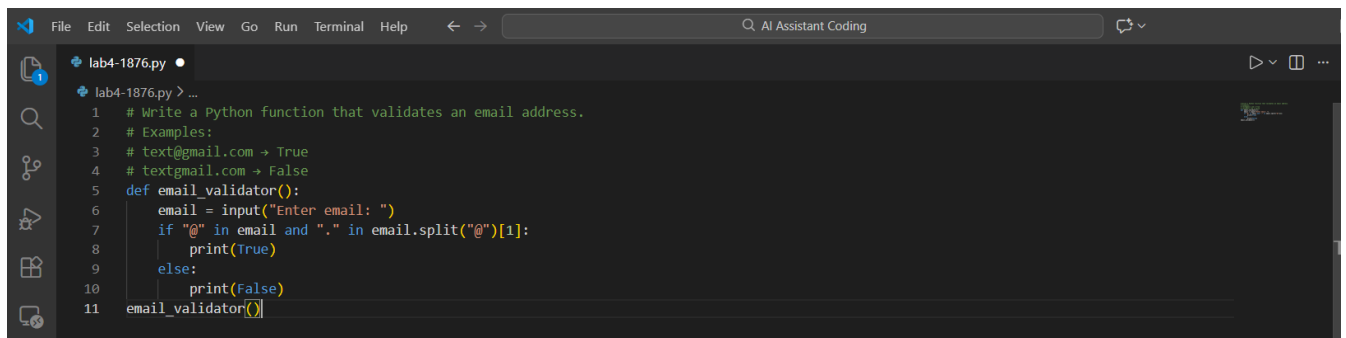
Write a Python function that validates an email address.

Examples:

text@gmail.com → True

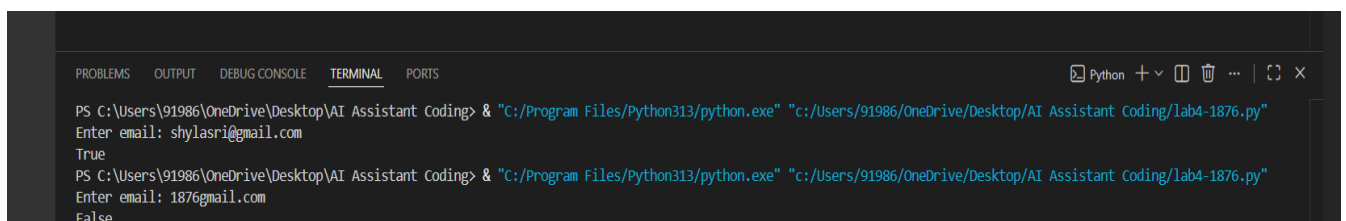
textgmail.com → False

CODE:



```
lab4-1876.py
lab4-1876.py > ...
1 # Write a Python function that validates an email address.
2 # Examples:
3 # text@gmail.com → True
4 # textgmail.com → False
5 def email_validator():
6     email = input("Enter email: ")
7     if "@" in email and "." in email.split("@")[1]:
8         print(True)
9     else:
10        print(False)
11 email_validator()
```

OUTPUT:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter email: shylasri@gmail.com
True
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter email: 1876@gmail.com
False
```

JUSTIFICATION:

This task highlights the **difference in code quality** between zero-shot and few-shot prompting.

- In the **zero-shot approach**, Copilot generated a basic solution that only checks for the presence of @, which is insufficient for real-world email validation.
- In the **few-shot approach**, multiple examples (valid and invalid emails) were provided. This guided Copilot to include additional checks such as domain validation and proper formatting.

This comparison demonstrates that few-shot prompting significantly improves **reliability, accuracy, and practical usefulness** of the generated code.

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

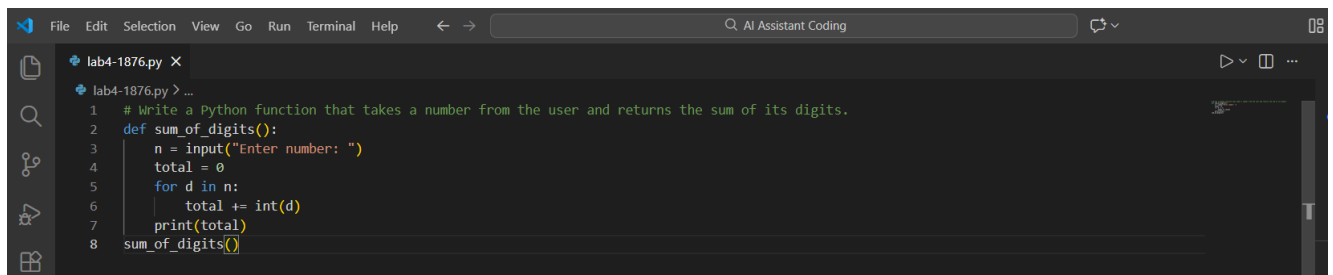
Expected Output #5

- 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:

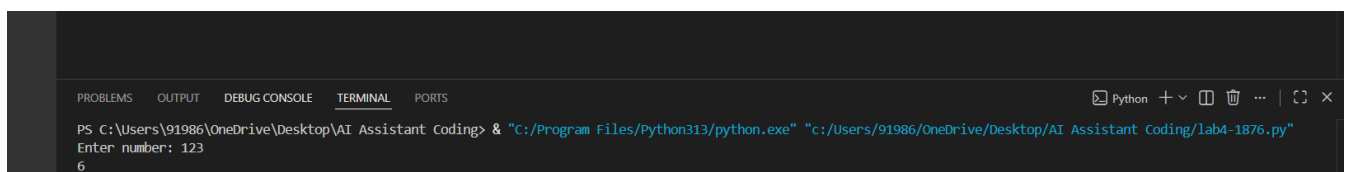
Write a Python function that takes a number from the user and returns the sum of its digits.

CODE:

A screenshot of a code editor window with a dark theme. The title bar shows 'lab4-1876.py'. The editor contains the following Python code:

```
1 # Write a Python function that takes a number from the user and returns the sum of its digits.
2 def sum_of_digits():
3     n = input("Enter number: ")
4     total = 0
5     for d in n:
6         total += int(d)
7     print(total)
8 sum_of_digits()
```

OUTPUT:

A screenshot of a terminal window with a dark theme. The title bar shows 'Python'. The terminal displays the following output:

```
PS C:\Users\91986\OneDrive\Desktop\AI Assistant Coding> & "C:/Program Files/Python313/python.exe" "c:/Users/91986/OneDrive/Desktop/AI Assistant Coding/lab4-1876.py"
Enter number: 123
6
```

JUSTIFICATION:

Prompt tuning examines how different prompt styles affect the quality of generated code.

- The **generic prompt** resulted in a longer, loop-based implementation.
- The **example-based prompt** encouraged Copilot to generate a more concise and optimized solution using Python's built-in functions.

This shows that adding input-output examples helps Copilot infer efficient coding practices. Prompt tuning is useful for improving readability, performance, and simplicity of the generated code.

