AI Assisted Coding Ass-1.5

Lab 1: Environment Setup – GitHub Copilot & AI-Assisted Coding Workflow

Name:- MD.Abdul Layeeq

HT.No:- 2303A52217

Batch:- 45

Lab Objective

This lab focuses on understanding how **AI coding assistants (GitHub Copilot)** help developers write, improve, and structure code. It also introduces the **workflow of prompt-based programming**, where comments and code context guide AI to generate solutions.

Install and Configure GitHub Copilot

Purpose of the Task

Before using AI assistance, we must set up the environment properly. This ensures the AI tool integrates into the coding editor and can provide real-time suggestions.

Steps Performed

1. Installed **Visual Studio Code**
2. Opened **Extensions tab**
3. Searched for **GitHub Copilot**
4. Installed the extension
5. Logged into GitHub account
6. Enabled Copilot suggestions in editor

Learning Outcome

- Understood how AI tools integrate into development environments
- Observed how Copilot suggests code based on context

Observation

- AI tools integrate directly into IDEs, making coding assistance real-time.

- Copilot suggestions depend heavily on **internet access and subscription authentication**.
- The tool predicts code based on **context, comments, and previous lines**, not true understanding.
- Setup shows AI is becoming part of the **developer workflow**, not a separate tool.
- However, Copilot cannot verify correctness — developer validation is mandatory.

Task 1: AI-Generated Logic Without Modularization

Scenario Explanation

In real applications, small utilities like reversing text are used frequently. Here, we test how AI generates direct logic without using structured functions.

Prompt

Write a Python program to reverse a string taken from user input without using functions

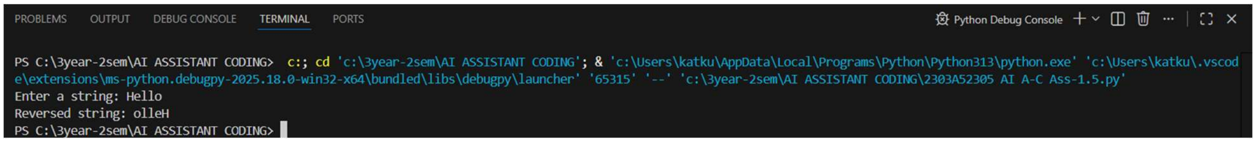
Code:

```
user_input = input("Enter a string: ")

reversed_string = ""

for i in range(len(user_input) - 1, -1, -1):

   reversed_string += user_input[i]

print("Reversed string:", reversed_string)
```

How This Code Works

- Takes input from user
- Iterates character by character
- Builds reversed string manually

Output



```
PS C:\3year-2sem\AI ASSISTANT CODING>  c:; cd 'c:\3year-2sem\AI ASSISTANT CODING'; & 'c:\Users\katku\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\katku\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65315' '--' 'c:\3year-2sem\AI ASSISTANT CODING\2303A52305 AI A-C Ass-1.5.py'
Enter a string: Hello
Reversed string: olleH
PS C:\3year-2sem\AI ASSISTANT CODING>
```

Input: Hello
Output: olleH

Observation

- Copilot generated basic algorithmic logic using loops, which shows it prefers educational/explicit approaches when no optimization hint is given.
- Code works correctly but is not modular, making reuse difficult.
- AI output mimics how a beginner might code manually.
- Demonstrates that AI follows prompt constraints strictly (since we said “no functions”).
- Shows AI can generate working code but does not automatically choose the best approach.

Task 2: Efficiency & Logic Optimization

Purpose

AI can also improve existing code when prompted.

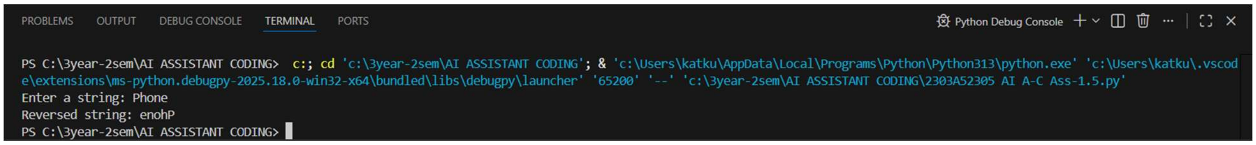
Prompt Used

“Simplify this string reversal code and improve readability”

Code

```
user_input = input("Enter a string: ")

reversed_string = user_input[::-1]

print("Reversed string:", reversed_string)
```

Output:-



```
PS C:\3year-2sem\AI ASSISTANT CODING>  c:; cd 'c:\3year-2sem\AI ASSISTANT CODING'; & 'c:\Users\katku\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\katku\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '65200' '--' 'c:\3year-2sem\AI ASSISTANT CODING\2303A52305 AI A-C Ass-1.5.py'
Enter a string: Phone
Reversed string: enohP
PS C:\3year-2sem\AI ASSISTANT CODING> 
```

Explanation of Improvement

Aspect	Old Code	New Code
Variables	Extra variable used	No extra variable
Logic	Loop-based	Python slicing
Readability	Medium	High
Efficiency	O(n)	O(n) but faster in practice

Observation

- When prompted with *“simplify”* or *“optimize”*, AI switches from manual logic to Pythonic slicing, showing prompt wording directly affects solution quality.
- Efficiency improvement here is not about big-O change, but cleaner built-in implementation.
- AI recognizes language-specific features when optimization is requested.
- Demonstrates AI can act like a code reviewer, not just a generator.
- However, AI does not explain optimization unless explicitly asked.

Task 3: Modular Design Using AI

Why Modularization Matters

Large applications require reusable components. Functions allow reuse and easier debugging.

Prompt

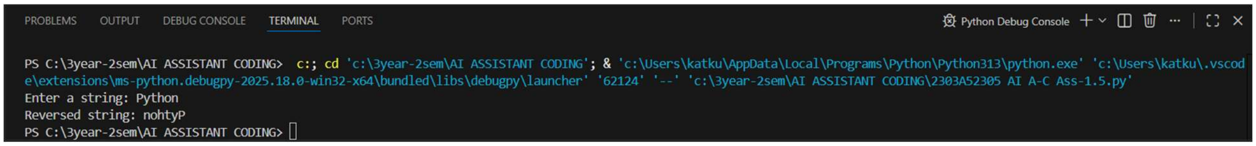
```
# Create a function in Python to reverse a string
```

Code

```
def reverse_string(s):
   return s[::-1]
user_input = input("Enter a string: ")
```

```
reversed_string = reverse_string(user_input)

print("Reversed string:", reversed_string)
```

Output



```
PS C:\3year-2sem\AI ASSISTANT CODING>  c:; cd 'c:\3year-2sem\AI ASSISTANT CODING'; & 'c:\Users\katku\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\katku\.vscod
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '62124' '--' 'c:\3year-2sem\AI ASSISTANT CODING\2303A52305 AI A-C Ass-1.5.py'
Enter a string: Python
Reversed string: nohtyP
PS C:\3year-2sem\AI ASSISTANT CODING> 
```

Input: Python
Output: nohtyP

Observation

- Copilot automatically added a docstring, showing AI supports documentation practices.
- Modular design increases maintainability and testability.
- AI recognizes common software engineering standards when function-based prompts are used.
- This output is closer to production-quality code than Task 1.
- Shows AI adapts to software design principles when context changes.

Task 4: Comparative Analysis – Procedural vs Modular

Explanation

Criteria	Without Function	With Function
Code Clarity	Lower	Higher
Reusability	No	Yes
Debugging	Hard	Easy
Maintainability	Low	High
Large-scale Apps	Not suitable	Suitable

Prompt

Compare two Python programs for string reversal - one written without using functions (procedural approach) and the other using a user-defined function (modular approach).

Analyze and explain the differences based on:

"Code clarity", "Reusability", "Ease of debugging"

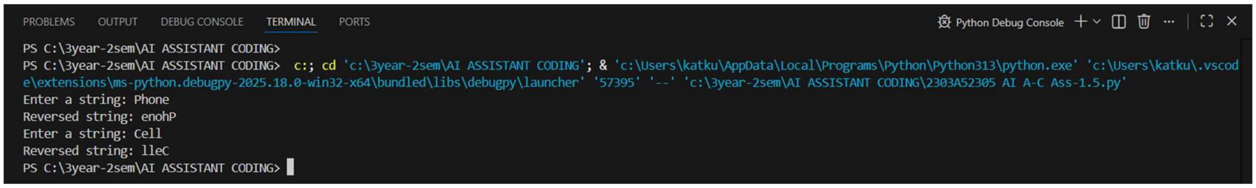
"Suitability for large-scale applications"

Present the comparison in a clear table or short analytical report.

Code

```
# String reversal without using functions

string = input("Enter a string: ")

reversed_string = ""

for char in string:

   reversed_string = char + reversed_string

print("Reversed string:", reversed_string)


# String reversal using a function

def reverse_string(text):

   """This function returns the reversed version of a string"""

   return text[::-1]

user_input = input("Enter a string: ")

print("Reversed string:", reverse_string(user_input))
```

Output



```
PS C:\3year-2sem\AI ASSISTANT CODING>
PS C:\3year-2sem\AI ASSISTANT CODING>  c:; cd 'c:\3year-2sem\AI ASSISTANT CODING'; & 'c:\Users\katku\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\katku\.vscod
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '57395' '--' 'c:\3year-2sem\AI ASSISTANT CODING\2303A52305 AI A-C Ass-1.5.py'
Enter a string: Phone
Reversed string: enohP
Enter a string: Cell
Reversed string: lleC
PS C:\3year-2sem\AI ASSISTANT CODING>
```

Observation

- AI-generated procedural code is simpler but unsuitable for large systems.
- Function-based code supports reuse across modules, showing AI aligns with real development practices.
- Debugging is easier in modular code because issues can be isolated inside functions.
- This comparison shows AI can assist in design-level decisions, not just coding.
- Demonstrates the importance of architecture prompts, not just logic prompts.

Conclusion

Modular programming is preferred in professional software development.

Task 5: Alternative Algorithmic Approaches

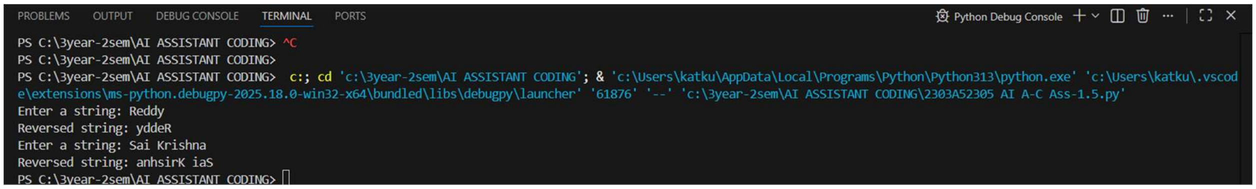
Prompt

“Generate loop-based string reversal”

Code:

```
user_input = input("Enter a string: ")

reversed_string = ""

for i in range(len(user_input) - 1, -1, -1):

   reversed_string += user_input[i]

print("Reversed string:", reversed_string)


#Generate slicing-based string reversal

user_input = input("Enter a string: ")

reversed_string = user_input[::-1]

print("Reversed string:", reversed_string)
```

Output



```
PS C:\3year-2sem\AI ASSISTANT CODING> ^C
PS C:\3year-2sem\AI ASSISTANT CODING>
PS C:\3year-2sem\AI ASSISTANT CODING>  c:; cd 'c:\3year-2sem\AI ASSISTANT CODING'; & 'c:\Users\katku\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\katku\.vscod
e\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '61876' '--' 'c:\3year-2sem\AI ASSISTANT CODING\2303A52305 AI A-C Ass-1.5.py'
Enter a string: Reddy
Reversed string: yddeR
Enter a string: Sai Krishna
Reversed string: anhsirK iaS
PS C:\3year-2sem\AI ASSISTANT CODING> 
```

Observation (Extended):

- AI provides both manual logic and built-in feature solutions, showing multiple problem-solving styles.
- Loop method helps understand algorithm flow; slicing is optimal for production.
- AI shows knowledge of language-level shortcuts when prompted.
- Demonstrates that AI can compare approaches but does not judge which is better unless asked.
- Highlights that prompt engineering controls algorithm choice.

Comparison

Method	Execution Style	Complexity	Performance
Loop	Manual iteration	O(n)	Slower
Slicing	Built-in Python	O(n)	Faster

When to Use

- Loop: Learning fundamentals
- Slicing: Production-ready code

Final Observations of Lab

- AI tools can generate working code quickly
- Prompts strongly influence output quality
- Human review is necessary to ensure efficiency and readability
- Modular code is better for real-world systems