

## AI ASSISTANT CODING

Name: S.VYSHNAVI

Ht no: 2303A52239

Batch: 35

Q.No. Question Expected

Time

to

complete

1

Lab 1: Environment Setup – GitHub Copilot and VS Code Integration +  
Understanding AI-assisted Coding Workflow

Lab Objectives:

- ❖ To install and configure GitHub Copilot in Visual Studio Code.

Week1 -

Monday

- ❖ To explore AI-assisted code generation using GitHub Copilot.

- ❖ To analyze the accuracy and effectiveness of Copilot's code suggestions.

- ❖ To understand prompt-based programming using comments and code

context

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- ❖ Set up GitHub Copilot in VS Code successfully.

- ❖ Use inline comments and context to generate code with Copilot.
- ❖ Evaluate AI-generated code for correctness and readability.
- ❖ Compare code suggestions based on different prompts and programming styles.

#### Task 0

- ❖ Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

#### Expected Output

- ❖ Install and configure GitHub Copilot in VS Code. Take screenshots of each step.

#### Task 1: AI-Generated Logic Without Modularization (String Reversal Without Functions)

##### ❖ Scenario

You are developing a basic text-processing utility for a messaging application.

##### ❖ Task Description

Use GitHub Copilot to generate a Python program that:

- Reverses a given string
- Accepts user input
- Implements the logic directly in the main code

- Does not use any user-defined functions

- ❖ Expected Output

- Correct reversed string

- Screenshots showing Copilot-generated code suggestions

- Sample inputs and outputs

## Task 2: Efficiency & Logic Optimization (Readability Improvement)

- ❖ Scenario

The code will be reviewed by other developers.

- ❖ Task Description

Examine the Copilot-generated code from Task 1 and improve it by:

- Removing unnecessary variables

- Simplifying loop or indexing logic

- Improving readability

- Use Copilot prompts like:

- “Simplify this string reversal code”

- “Improve readability and efficiency”

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

- ❖ Expected Output

- Original and optimized code versions

- Explanation of how the improvements reduce time complexity

### Task 3: Modular Design Using AI Assistance (String Reversal Using Functions)

#### ❖ Scenario

The string reversal logic is needed in multiple parts of an application.

#### ❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

- Uses a user-defined function to reverse a string
- Returns the reversed string
- Includes meaningful comments (AI-assisted)

#### ❖ Expected Output

- Correct function-based implementation
- Screenshots documenting Copilot's function generation
- Sample test cases and outputs

### Task 4: Comparative Analysis – Procedural vs Modular Approach (With vs Without Functions)

#### ❖ Scenario

You are asked to justify design choices during a code review.

#### ❖ Task Description

Compare the Copilot-generated programs:

- Without functions (Task 1)
- With functions (Task 3)

Analyze them based on:

- Code clarity
- Reusability
- Debugging ease
- Suitability for large-scale applications

#### ❖ Expected Output

Comparison table or short analytical report

### Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different

### Algorithmic Approaches to String Reversal)

#### ❖ Scenario

Your mentor wants to evaluate how AI handles alternative logic paths.

#### ❖ Task Description

Prompt GitHub Copilot to generate:

- A loop-based string reversal approach
- A built-in / slicing-based string reversal approach

#### ❖ Expected Output

- Two correct implementations
- Comparison discussing:
  - Execution flow
  - Time complexity
  - Performance for large inputs
  - When each approach is appropriate

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.

## **Solutions:**

# Python program to reverse a string

```
user_input = input("Enter a string to reverse: ")
print(f"Original: {user_input}\nReversed: {user_input[::-1]}")
```

# Python program to reverse a string using a user-defined function

# Demonstrates function-based approach with proper documentation

```
def reverse_string(text):
```

```
    """
```

```
    Reverses the given string using Python's slicing technique.
```

Args:

text (str): The input string to be reversed

Returns:

str: The reversed string

"""

return text[::-1]

# Main program execution

if \_\_name\_\_ == "\_\_main\_\_":

# Prompt user for input

user\_input = input("Enter a string to reverse: ")

# Call the function to get reversed string

result = reverse\_string(user\_input)

# Display the results

print(f"Original: {user\_input}")

print(f"Reversed: {result}")

## **## 1. CODE CLARITY**

### **### Without Functions (Task 1)**

```

```python
# Python program to reverse a string

user_input = input("Enter a string to reverse: ")
print(f"Original: {user_input}\nReversed: {user_input[::-1]}")
```

```

### **\*\*Observations:\*\***

- ✓ Extremely concise (4 lines)
- ✓ Easy to understand at a glance
- ✗ Mixes input/output logic with core functionality
- ✗ No documentation of what the reversal does

### **### With Functions (Task 3)**

```

```python
def reverse_string(text):
    """
    Reverses the given string using Python's slicing technique.
    ...
    """
    return text[::-1]

if __name__ == "__main__":

```



```

user_input = input("Enter a string to reverse: ")
result = reverse_string(user_input)
print(f"Original: {user_input}\nReversed: {result}")
...

```

### **\*\*Observations:\*\***

- ✓ Clear separation of concerns
- ✓ Comprehensive docstring documentation
- ✓ Explicit function purpose and behavior
- ✗ More verbose (27 lines with documentation)
- ✓ Follows Python best practices with `if \_\_name\_\_ == "\_\_main\_\_":`

**\*\*Winner: Task 3\*\*** - Better documentation and separation of concerns make intent clearer.

---

## **## 2. REUSABILITY**

### **### Without Functions (Task 1)**

**\*\*Reusability Score: 1/5\*\***

- + Cannot reuse the reversal logic in other parts of the program
- + Cannot import and use in other modules
- + Must duplicate code if reversal is needed elsewhere
- + Tightly coupled with input/output operations

### **\*\*Example Problem:\*\***

```
```python
```

```
# To reverse multiple strings, must repeat the logic
```

```
string1_reversed = string1[::-1]
```

```
string2_reversed = string2[::-1]
```

```
string3_reversed = string3[::-1]
```

```
```
```

### **### With Functions (Task 3)**

#### **\*\*Reusability Score: 5/5\*\***

- ✓ Function can be imported into other modules
- ✓ Can reverse any number of strings without duplication
- ✓ Logic is isolated and independent
- ✓ Can be used in different contexts (APIs, GUIs, batch processing)

## **\*\*Example Benefit:\*\***

```
```python
from reverse_string_function import reverse_string

strings = ["hello", "world", "python"]
reversed_strings = [reverse_string(s) for s in strings]
# Result: ['olleh', 'dlrow', 'nohtyp']
```
```

**\*\*Winner: Task 3\*\*** - Dramatically superior for code reuse and modularity.

---

## **## 3. DEBUGGING EASE**

### **### Without Functions (Task 1)**

#### **\*\*Debugging Score: 2/5\*\***

- + Hard to isolate which part has issues
- + No clear entry/exit points for testing
- + Cannot debug the reversal logic independently
- + Changes require modifying the entire script

## **\*\*Challenges:\*\***

```
```python
# Is the problem in input handling or reversal logic?
print(f"Original: {user_input}\nReversed: {user_input[::-1]}")
# Difficult to pinpoint issues
```
```

## **### With Functions (Task 3)**

### **\*\*Debugging Score: 5/5\*\***

- ✓ Can test the function independently
- ✓ Can add breakpoints specifically in the function
- ✓ Unit testing is straightforward
- ✓ Can isolate bugs to specific sections

## **\*\*Example Testing:\*\***

```
```python
# Easy to test the function directly
assert reverse_string("hello") == "olleh"
assert reverse_string("world") == "dlrow"
assert reverse_string("") == ""
```
```

**\*\*Winner: Task 3\*\*** - Function-based design enables systematic debugging and testing.

---

## **## 4. SUITABILITY FOR LARGE-SCALE APPLICATIONS**

### **### Without Functions (Task 1)**

#### **\*\*Large-Scale Suitability: 1/5\*\***

- + Not scalable to larger applications
- + No code organization or structure
- + Impossible to maintain in teams
- + Cannot be part of a larger project
- + No testing framework compatibility

#### **\*\*Why It Fails:\*\***

- Single-purpose scripts only
- Cannot integrate with frameworks
- No separation of business logic from I/O
- Violates Single Responsibility Principle

### ### With Functions (Task 3)

#### **\*\*Large-Scale Suitability: 5/5\*\***

- ✓ Easily integrates into larger projects
- ✓ Follows software engineering principles
- ✓ Compatible with unit testing frameworks (pytest, unittest)
- ✓ Can be part of modules and packages
- ✓ Supports code organization and architectural patterns

#### **\*\*Enterprise-Ready Features:\*\***

```
```python
```

```
# Can be used in web frameworks
```

```
from flask import Flask
```

```
from reverse_string_function import reverse_string
```

```
app = Flask(__name__)
```

```
@app.route('/reverse/<text>')
```

```
def api_reverse(text):
```

```
    return {"reversed": reverse_string(text)}
```

```
# Can be tested with pytest
```

```
def test_reverse_string():
    assert reverse_string("test") == "tset"

# Can be integrated into larger modules
# from utils.string_operations import reverse_string
...
```

**\*\*Winner: Task 3\*\*** - Essential for professional, scalable software development.

---

**## SUMMARY COMPARISON TABLE**

Criterion	Task 1 (No Functions)	Task 3 (With Functions)
----- ----- -----		
<b>**Lines of Code**</b>	4	27
<b>**Code Clarity**</b>	★★★ (Concise)	★★★★★ (Well-documented)
<b>**Reusability**</b>	★ (None)	★★★★★ (Excellent)
<b>**Debugging Ease**</b>	★★ (Difficult)	★★★★★ (Easy)

| **\*\*Large-Scale Apps\*\*** | ★ (Not suitable) |

★★★★★ (Ideal) |

| **\*\*Testing\*\*** | + (Not testable) | ✓ (Fully testable) |

| **\*\*Maintenance\*\*** | + (Difficult) | ✓ (Easy) |

---

## ## CONCLUSION

### ### Use Task 1 (No Functions) When:

- Writing quick, throwaway scripts
- Learning Python basics
- One-time utility tasks
- Minimal complexity required

### ### Use Task 3 (With Functions) When:

- Building production code
- Working in teams
- Planning to extend the application
- Requiring code quality and maintainability
- Needing unit tests
- Integrating into larger projects



**\*\*Best Practice Recommendation:\*\*** **\*\*Task 3 is the professional standard.\*\*** Even for simple programs, using functions establishes good habits and makes code enterprise-ready. The minimal overhead of function definition pays dividends in reusability, testing, and maintenance.

# Python program demonstrating different string reversal approaches

# Compares loop-based vs built-in/slicing-based methods

#

=====

# APPROACH 1: LOOP-BASED STRING REVERSAL

#

=====

def reverse\_string\_loop(text):

"""

Reverses a string using a manual loop approach.

Iterates through the string backwards and builds a new string.

Args:

text (str): The input string to be reversed

Returns:

str: The reversed string

```
"""
```

```
reversed_text = ""
```

```
for i in range(len(text) - 1, -1, -1):
```

```
    reversed_text += text[i]
```

```
return reversed_text
```

```
def reverse_string_loop_alt(text):
```

```
    """
```

Alternative loop-based approach using a for-each loop.

Converts string to list, then iterates in reverse.

Args:

text (str): The input string to be reversed

Returns:

str: The reversed string

```
"""
```

```
reversed_text = ""
```

```
for char in reversed(text):
```

```
    reversed_text += char
```

```
return reversed_text
```

```
#
```

```
=====
```

```
=====
```

```
# APPROACH 2: BUILT-IN / SLICING-BASED STRING REVERSAL
```

```
#
```

```
=====
```

```
=====
```

```
def reverse_string_slicing(text):
```

```
    """
```

```
    Reverses a string using Python's slicing technique.
```

```
    Most efficient and Pythonic approach.
```

```
    Args:
```

```
        text (str): The input string to be reversed
```

```
    Returns:
```

str: The reversed string

"""

return text[::-1]

def reverse\_string\_reversed\_builtin(text):

"""

Reverses a string using the built-in reversed() function.

Returns an iterator, so needs to be joined.

Args:

text (str): The input string to be reversed

Returns:

str: The reversed string

"""

return "".join(reversed(text))

#

=====

=====

# MAIN PROGRAM - DEMONSTRATION AND COMPARISON

```

#
=====

=====

if __name__ == "__main__":
    # Test string
    test_string = input("Enter a string to reverse: ")

    print("\n" + "=" * 60)
    print("STRING REVERSAL APPROACHES COMPARISON")
    print("=" * 60)
    print(f"\nOriginal String: '{test_string}'")
    print("-" * 60)

    # LOOP-BASED APPROACHES
    print("\n1. LOOP-BASED APPROACHES:")
    print("-" * 60)

    # Approach 1a: Manual index-based loop
    result1 = reverse_string_loop(test_string)
    print(f"\n Loop with Index (range):")
    print(f" Code: for i in range(len(text) - 1, -1, -1): ...")

```

```
print(f"  Result: '{result1}')
```

```
# Approach 1b: Using reversed() function with loop
```

```
result2 = reverse_string_loop_alt(test_string)
```

```
print(f"\n  Loop with reversed() function:")
```

```
print(f"  Code: for char in reversed(text): ...")
```

```
print(f"  Result: '{result2}')
```

```
# BUILT-IN / SLICING APPROACHES
```

```
print("\n\n2. BUILT-IN / SLICING APPROACHES:")
```

```
print("-" * 60)
```

```
# Approach 2a: Slicing (Most Pythonic)
```

```
result3 = reverse_string_slicing(test_string)
```

```
print(f"\n  Slicing (MOST PYTHONIC):")
```

```
print(f"  Code: text[::-1]")
```

```
print(f"  Result: '{result3}')
```

```
# Approach 2b: Using reversed() with join()
```

```
result4 = reverse_string_reversed_builtin(test_string)
```

```
print(f"\n  reversed() + join():")
```

```
print(f"  Code: ''.join(reversed(text))")
```

```
print(f"  Result: '{result4}')
```

```
# VERIFICATION
```

```
print("\n" + "=" * 60)
```

```
print("VERIFICATION - All methods produce same result:")
```

```
print("=" * 60)
```

```
all_equal = result1 == result2 == result3 == result4
```

```
print(f"All results equal: {all_equal} ✓" if all_equal else  
f"Results differ: FAILED X")
```

```
# PERFORMANCE COMPARISON SUMMARY
```

```
print("\n" + "=" * 60)
```

```
print("PERFORMANCE & READABILITY COMPARISON")
```

```
print("=" * 60)
```

```
print("""
```

Method	Speed	Readability	Recommendation
--------	-------	-------------	----------------

---

```
1. Loop with
```

Index	Slow	Medium	Learning/Detailed
-------	------	--------	-------------------

```
2. Loop with
```

reversed()	Medium	Medium	Educational
------------	--------	--------	-------------

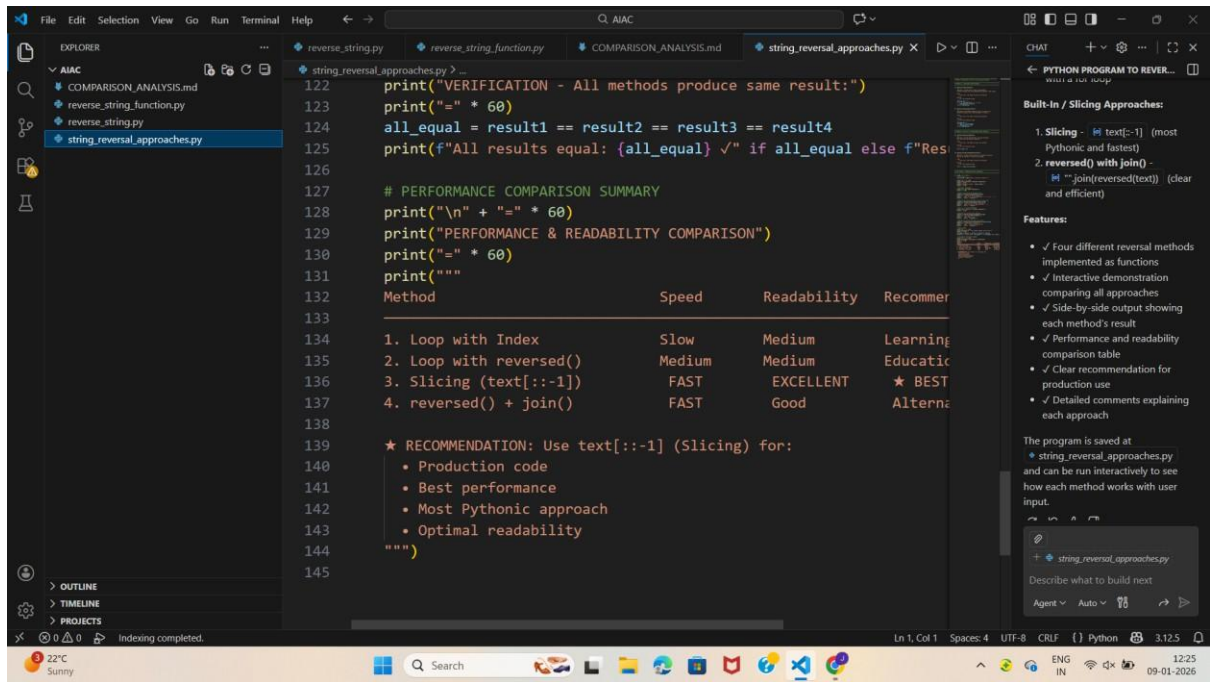
3. Slicing (text[::-1])	FAST	EXCELLENT	★ BEST CHOICE
4. reversed() + join()	FAST	Good	Alternative

★ RECOMMENDATION: Use text[::-1] (Slicing) for:

- Production code
- Best performance
- Most Pythonic approach
- Optimal readability

""")





```
98 # Approach 1b: Using reversed() function with loop
99 result2 = reverse_string_loop_alt(test_string)
100 print(f"\n Loop with reversed() function:")
101 print(f" Code: for char in reversed(text): ...")
102 print(f" Result: '{result2}'")
103
104 # BUILT-IN / SLICING APPROACHES
105 print("\n2. BUILT-IN / SLICING APPROACHES:")
106 print("-" * 60)
107
108 # Approach 2a: Slicing (Most Pythonic)
109 result3 = reverse_string_slicing(test_string)
110 print(f"\n Slicing (MOST PYTHONIC):")
111 print(f" Code: text[::-1]")
112 print(f" Result: '{result3}'")
113
114 # Approach 2b: Using reversed() with join()
115 result4 = reverse_string_reversed_builtin(test_string)
116 print(f"\n reversed() + join():")
117 print(f" Code: ''.join(reversed(text))")
118 print(f" Result: '{result4}'")
119
120 # VERIFICATION
121 print("\n" + "=" * 60)
122 print("VERIFICATION - All methods produce same result:")
123 print("=" * 60)
```

**Built-in / Slicing Approaches:**

1. Slicing - `text[::-1]` (most Pythonic and fastest)
2. `reversed()` with `join()` - `"".join(reversed(text))` (clear and efficient)

**Features:**

- ✓ Four different reversal methods implemented as functions
- ✓ Interactive demonstration comparing all approaches
- ✓ Side-by-side output showing each method's result
- ✓ Performance and readability comparison table
- ✓ Clear recommendation for production use
- ✓ Detailed comments explaining each approach

The program is saved at `string_reversal_approaches.py` and can be run interactively to see how each method works with user input.

```
73
74 # =====
75 # MAIN PROGRAM - DEMONSTRATION AND COMPARISON
76 # =====
77
78 if __name__ == "__main__":
79     # Test string
80     test_string = input("Enter a string to reverse: ")
81
82     print("\n" + "=" * 60)
83     print("STRING REVERSAL APPROACHES COMPARISON")
84     print("=" * 60)
85     print(f"\nOriginal String: '{test_string}'")
86     print("-" * 60)
87
88     # LOOP-BASED APPROACHES
89     print("\n1. LOOP-BASED APPROACHES:")
90     print("-" * 60)
91
92     # Approach 1a: Manual index-based loop
93     result1 = reverse_string_loop(test_string)
94     print(f"\n Loop with Index (range):")
95     print(f" Code: for i in range(len(text) - 1, -1, -1): ...")
96     print(f" Result: '{result1}'")
97
98 # Approach 1b: Using reversed() function with loop
```

**Built-in / Slicing Approaches:**

1. Slicing - `text[::-1]` (most Pythonic and fastest)
2. `reversed()` with `join()` - `"".join(reversed(text))` (clear and efficient)

**Features:**

- ✓ Four different reversal methods implemented as functions
- ✓ Interactive demonstration comparing all approaches
- ✓ Side-by-side output showing each method's result
- ✓ Performance and readability comparison table
- ✓ Clear recommendation for production use
- ✓ Detailed comments explaining each approach

The program is saved at `string_reversal_approaches.py` and can be run interactively to see how each method works with user input.

