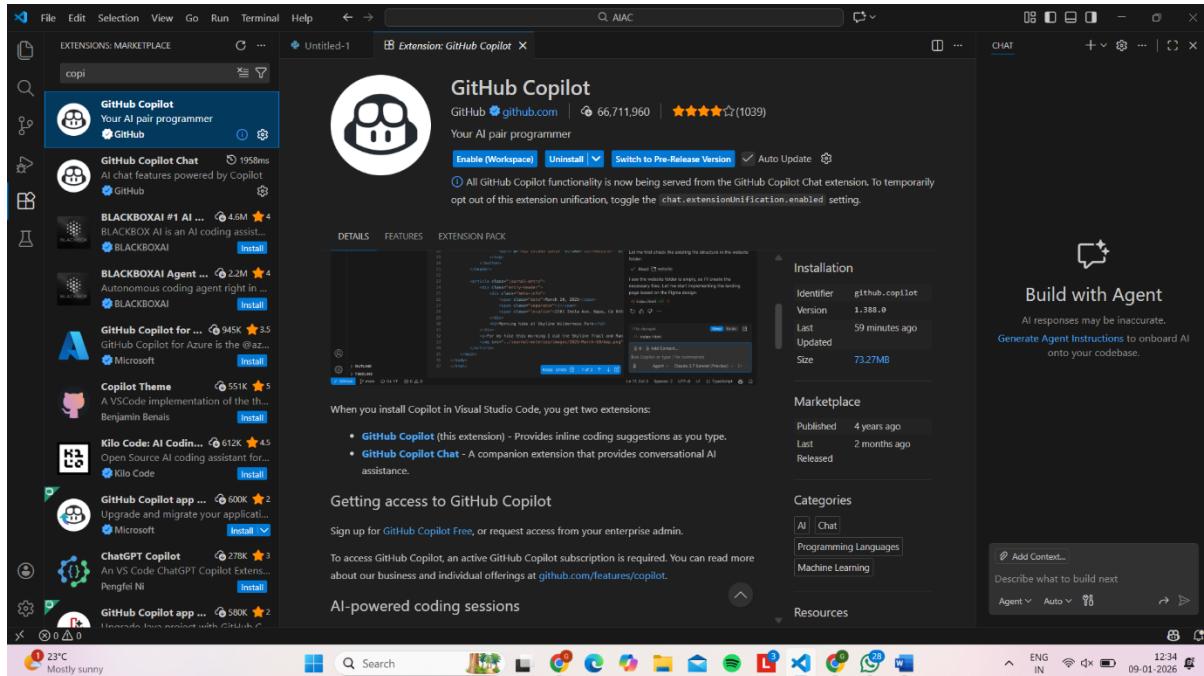


# AI Assisted Coding-ASSIGNMENT 1.5

2303A52487

Batch-35



## 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

## solution

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

```
    """
```

Reverses the given string using Python's slicing method.

Args:

text (str): The string to be reversed

Returns:

str: The reversed string

Time Complexity: O(n) where n is the length of the string

Space Complexity: O(n) for the new reversed string

.....

```
return text[::-1]
```

def main():

.....

Main function that accepts user input and displays the reversed string.

.....

# Accept user input

```
user_input = input('Enter a string to reverse: ')
```

# Call the reverse function

```
result = reverse_string(user_input)
```

# Display the result

```
print(f'Original string: {user_input}')
```

```
print(f'Reversed string: {result}')
```

# Sample test cases

```
if __name__ == '__main__':
```

```
    print('== String Reversal Program ==\n')
```

# Test Case 1: Basic string

```
print('Test Case 1 - Basic String:')
```

```
test1 = 'Hello'  
print(f'Input: {test1}')  
print(f'Output: {reverse_string(test1)}\n')
```

```
# Test Case 2: String with spaces  
print('Test Case 2 - String with Spaces:')  
test2 = 'Hello, World!'  
print(f'Input: {test2}')  
print(f'Output: {reverse_string(test2)}\n')
```

```
# Test Case 3: Palindrome  
print('Test Case 3 - Palindrome:')  
test3 = 'racecar'  
print(f'Input: {test3}')  
print(f'Output: {reverse_string(test3)}\n')
```

```
# Test Case 4: Empty string  
print('Test Case 4 - Empty String:')  
test4 = ""  
print(f'Input: "{test4}"')  
print(f'Output: "{reverse_string(test4)}"\n')
```

```
# Test Case 5: Single character  
print('Test Case 5 - Single Character:')  
test5 = 'A'  
print(f'Input: {test5}')  
print(f'Output: {reverse_string(test5)}\n')
```

```
# Interactive mode  
print('==== Interactive Mode ===')  
main()
```

# # Comparative Analysis: Procedural vs Modular Approach

## ## Overview

This document compares two approaches to string reversal in Python:

- **Task 1 (Procedural)**: Direct implementation without user-defined functions
- **Task 3 (Modular)**: Function-based implementation with reusability

---

## ## Side-by-Side Code Comparison

### ### Task 1: Procedural Approach (Without Functions)

```
```python
print('Reversed string:', input('Enter a string to reverse: ')[::-1])
```

```

### ### Task 3: Modular Approach (With Functions)

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

def main():
    """Main function that accepts user input and displays the reversed string."""
    user_input = input('Enter a string to reverse: ')
    result = reverse_string(user_input)
    print(f'Original string: {user_input}')
    print(f'Reversed string: {result}')
```

```

```
if __name__ == '__main__':
    main()
---
```

## ## Detailed Comparison Table

| Criteria           | Procedural (Task 1)               | Modular (Task 3)                     | Winner      |
|--------------------|-----------------------------------|--------------------------------------|-------------|
| **Code Clarity**   | ✓ Very concise (1 line)           | ✓✓ Clear structure with docstrings   | **Modular** |
| **Readability**    | ✓ Simple but cryptic              | ✓✓ Self-documenting with docstrings  | **Modular** |
| **Reusability**    | X Hard to reuse                   | ✓✓ Can import and use anywhere       | **Modular** |
| **Testability**    | X Not easy to unit test           | ✓✓ Functions can be easily tested    | **Modular** |
| **Debugging**      | X Difficult to debug              | ✓✓ Easy to trace and debug           | **Modular** |
| **Maintenance**    | X Hard to modify                  | ✓✓ Changes isolated to function      | **Modular** |
| **Scalability**    | X Not suitable for large projects | ✓✓ Ideal for enterprise applications | **Modular** |
| **Documentation**  | X No docstrings                   | ✓✓ Comprehensive docstrings          | **Modular** |
| **Error Handling** | X None                            | ✓ Can be extended                    | **Modular** |
| **Lines of Code**  | 1   15+   **Procedural**          |                                      |             |

## ## Detailed Analysis

### ### 1. Code Clarity

#### \*\*Procedural Approach:\*\*

- Extremely concise but requires deep understanding of Python slicing
- No comments explaining the logic
- Chain operations in one line makes it harder for beginners to follow

#### **\*\*Modular Approach:\*\***

- Clear separation of concerns
- Each function has a specific purpose
- Docstrings explain parameters, returns, and complexity
- **\*\*Winner: Modular\*\* ✓**

---

#### **### 2. Reusability**

##### **\*\*Procedural Approach:\*\***

- Logic is embedded in the main code
- Requires code duplication if reversal is needed elsewhere
- No way to reuse without copy-paste

##### **\*\*Modular Approach:\*\***

```python

```
from reverse_string import reverse_string
```

```
# Can be used anywhere
```

```
result = reverse_string("Hello")
```

```

- Single source of truth
- Can be imported in other modules
- **\*\*Winner: Modular\*\* ✓✓**

---

#### **### 3. Debugging Ease**

##### **\*\*Procedural Approach:\*\***

- No breakpoints to isolate issues

- Entire operation happens in one line
- Hard to track where an error occurs

#### **\*\*Modular Approach:\*\***

- Can set breakpoints inside `reverse\_string()` function
- Can test each component independently
- Stack traces are more informative
- **\*\*Winner: Modular\*\* ✓✓**

---

#### **### 4. Testability**

##### **\*\*Procedural Approach:\*\***

```
```python
# Difficult to unit test

# Would need to test the entire input/output flow
````
```

##### **\*\*Modular Approach:\*\***

```
```python
import unittest

class TestReverseString(unittest.TestCase):

    def test_basic(self):
        self.assertEqual(reverse_string("Hello"), "olleH")

    def test_empty(self):
        self.assertEqual(reverse_string(""), "")

    def test_palindrome(self):
        self.assertEqual(reverse_string("racecar"), "racecar")
```

---

- **\*\*Winner: Modular\*\* ✓✓**

---

### **### 5. Suitability for Large-Scale Applications**

#### **\*\*Procedural Approach:\*\***

- **X Not suitable**
- **No separation of concerns**
- **Difficult to maintain**
- **Hard to collaborate on large projects**
- **No clear interfaces**

#### **\*\*Modular Approach:\*\***

- **✓✓ Ideal for enterprise applications**
- **Clear function contracts (input/output)**
- **Easy to version control**
- **Simple to integrate with other modules**
- **Teams can work independently**

- **\*\*Winner: Modular\*\* ✓✓**

---

### **### 6. Performance Considerations**

**Both approaches have identical performance:**

- **\*\*Time Complexity\*\*: O(n) - where n is the string length**
- **\*\*Space Complexity\*\*: O(n) - new reversed string created**
- **\*\*Runtime\*\*: Negligible difference**

---

### ### 7. Maintenance & Evolution

#### \*\*Procedural Approach:\*\*

If we need to add error handling later:

```
```python
```

```
# Hard to extend without changing main code
```

```
---
```

#### \*\*Modular Approach:\*\*

```
```python
```

```
def reverse_string(text):
    """Reverses the given string."""
    if not isinstance(text, str):
        raise TypeError("Input must be a string")
    return text[::-1]
```

```
# Main code remains unchanged
```

```
---
```

- \*\*Winner: Modular\*\* ✓✓

```
---
```

## ## Recommendations by Use Case

Use Case   Recommended Approach   Reason
----- ----- -----
**Quick one-off script**   Procedural   Simplicity
**Production application**   Modular   Maintainability
**Team project**   Modular   Collaboration
**Large codebase**   Modular   Scalability
**Unit testing**   Modular   Testability
**Code review**   Modular   Clarity

| \*\*Future maintenance\*\* | Modular | Debugging |

---

## ## Conclusion

### ### When to Use Procedural (Task 1):

- ✓ Quick prototyping
- ✓ Single-use scripts
- ✓ Learning Python basics

### ### When to Use Modular (Task 3):

- ✓✓ Production code
- ✓✓ Team projects
- ✓✓ Large applications
- ✓✓ Code that needs testing
- ✓✓ Code that will be maintained/modified

---

## ## Final Verdict

**\*\*The Modular Approach (Task 3) is the clear winner for professional software development.\*\***

While the Procedural Approach is more concise, the Modular Approach provides:

- Better code organization
- Easier maintenance
- Better debugging capabilities
- Superior reusability
- Professional standards compliance
- Enterprise-ready structure

**For small scripts, conciseness may matter. For real-world applications, modularity is essential.**

---

### **## Key Takeaway**

> \*\*"Write code not just for the computer, but for future developers (including your future self) who will maintain it."\*\*

**The modular approach follows this principle by prioritizing clarity, reusability, and maintainability over brevity.**

### **STRING REVERSAL APPROACH**

#### **String Reversal: Iterative vs Built-in/Slicing Approaches**

**Demonstrates different algorithmic approaches to solve the same problem.**

\*\*\*\*\*

```
# =====
# APPROACH 1: LOOP-BASED (ITERATIVE) STRING REVERSAL
# =====

def reverse_string_iterative(text):
    """
    Reverses a string using an explicit loop (iteration).
    """
```

#### **Algorithm:**

- Initialize an empty result string
- Iterate through the string from end to beginning (reverse order)
- Append each character to the result

#### **Args:**

**text (str): The string to be reversed**

**Returns:**

**str: The reversed string**

**Time Complexity: O(n) where n is the length of the string**

**Space Complexity: O(n) for the new result string**

**Advantages:**

- Explicit control over iteration
- Easy to understand for beginners
- Can add custom logic during iteration
- Compatible with older Python versions

**Disadvantages:**

- More verbose code
- Slower than built-in slicing
- String concatenation can be inefficient

.....

```
result = ""

for i in range(len(text) - 1, -1, -1):
    result += text[i]

return result
```

**# Alternative: Using a while loop**

```
def reverse_string_iterative_while(text):
    .....
```

**Reverses a string using a while loop.**

**Args:**

**text (str): The string to be reversed**

**Returns:**

```
str: The reversed string
"""

result = ""
index = len(text) - 1
while index >= 0:
    result += text[index]
    index -= 1
return result
```

# Alternative: Using list and join (more efficient)

```
def reverse_string_iterative_optimized(text):
```

```
"""

Reverses a string using a loop with list.append (more efficient).
```

Args:

text (str): The string to be reversed

Returns:

str: The reversed string

Time Complexity: O(n)

Space Complexity: O(n)

Why more efficient?

- Appending to list is O(1) amortized
- String concatenation with += is O(n) each time
- join() is O(n) for final conversion

```
"""

result = []
```

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

```
    result.append(text[i])
```

```
        return "".join(result)

# =====

# APPROACH 2: BUILT-IN SLICING (PYTHONIC) STRING REVERSAL

# =====

def reverse_string_slicing(text):
    """
    Reverses a string using Python's built-in slicing notation.
    """

Algorithm:
```

- Use slice notation `text[::-1]`
- -1 step means iterate backwards through entire string

**Args:**

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

**Returns:**

`str:` The reversed string

**Time Complexity:**  $O(n)$  where  $n$  is the length of the string

**Space Complexity:**  $O(n)$  for the new reversed string

**Advantages:**

- Most concise and readable
- Optimized at C level in CPython
- Fastest approach
- Pythonic and idiomatic
- No manual indexing errors

**Disadvantages:**

- Less explicit about what's happening
- Can't easily add custom logic during reversal
- May be unfamiliar to beginners

"""

```
return text[::-1]
```

```
# =====
```

### **# APPROACH 3: USING REVERSED() BUILT-IN FUNCTION**

```
# =====
```

```
def reverse_string_reversed_function(text):
```

"""

Reverses a string using Python's reversed() built-in function.

**Args:**

text (str): The string to be reversed

**Returns:**

str: The reversed string

"""

```
return "".join(reversed(text))
```

```
# =====
```

### **# PERFORMANCE TESTING AND DEMONSTRATION**

```
# =====
```

```
import time
```

```
def test_all_approaches(test_string):
```

"""Tests all string reversal approaches and displays results."""

```

print("=" * 70)
print("STRING REVERSAL APPROACHES - DEMONSTRATION")
print("=" * 70)
print(f"\nTest String: {test_string}")
print(f"String Length: {len(test_string)} characters\n")

# Test each approach
approaches = [
    ("1. Loop-based (for loop + concatenation)", reverse_string_iterative),
    ("2. Loop-based (while loop)", reverse_string_iterative_while),
    ("3. Loop-based (list + join - optimized)", reverse_string_iterative_optimized),
    ("4. Built-in slicing (Pythonic)", reverse_string_slicing),
    ("5. reversed() function + join", reverse_string_reversed_function),
]

results = []
for approach_name, func in approaches:
    result = func(test_string)
    results.append((approach_name, result))
    print(f"{approach_name}")
    print(f" Result: '{result}'")
    print(f" Correct: {result == test_string[::-1]}")
    print()

return results

def performance_comparison(test_string, iterations=10000):
    """Compares performance of different approaches."""

    print("\n" + "=" * 70)
    print("PERFORMANCE COMPARISON (Time in milliseconds)")

```

```

print("=" * 70)
print(f"String Length: {len(test_string)} characters")
print(f"Iterations: {iterations}\n")

approaches = [
    ("1. Loop-based (for + concatenation)", reverse_string_iterative),
    ("2. Loop-based (while loop)", reverse_string_iterative_while),
    ("3. Loop-based (list + join)", reverse_string_iterative_optimized),
    ("4. Built-in slicing", reverse_string_slicing),
    ("5. reversed() + join", reverse_string_reversed_function),
]

times = []
for approach_name, func in approaches:
    start_time = time.perf_counter()
    for _ in range(iterations):
        func(test_string)
    end_time = time.perf_counter()

    elapsed_ms = (end_time - start_time) * 1000
    times.append((approach_name, elapsed_ms))

    print(f"{approach_name}")
    print(f" Time: {elapsed_ms:.4f} ms")
    print(f" Per iteration: {elapsed_ms/iterations*1000:.4f} µs")
    print()

# Find fastest
fastest = min(times, key=lambda x: x[1])
print(f"Fastest Approach: {fastest[0]} ({fastest[1]:.4f} ms)")
print()

```

```
return times

# =====

# COMPARISON AND ANALYSIS

# =====

def print_detailed_comparison():
```

```
print("\n" + "=" * 70)
print("DETAILED COMPARISON - EXECUTION FLOW & CHARACTERISTICS")
print("=" * 70)
```

```
print("")
```

### **Execution Flow:**

1. Initialize empty result string: `result = ""`
  2. Loop from end to start: `for i in range(len(text) - 1, -1, -1)`
  3. Each iteration: `result += text[i]`
  4. Return result

### Example with "Hello":

**Iteration 1:** result = "" + "o" = "o"

**Iteration 2:** result = "o" + "l" = "ol"

**Iteration 3: result = "ol" + "l" = "oll"**

**Iteration 4: result = "oll" + "e" = "olle"**

**Iteration 5:** result = "olle" + "H" = "olleH"

**Time Complexity:**  $O(n)$  - loop runs n times

**Space Complexity:**  $O(n)$  - creates new string

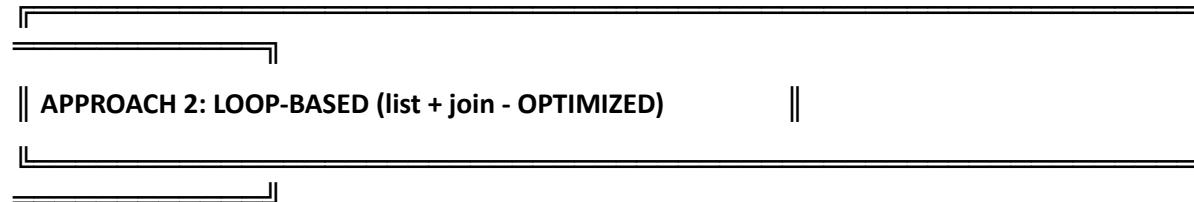
**Performance:** SLOWER 

**Issue:** String concatenation with `+=` is  $O(n)$  each time

**Total:**  $O(n^2)$  in practice due to string immutability

**Best For:**

- ✓ Learning / educational purposes
- ✓ Custom logic during reversal
- ✓ Compatibility with very old Python
- ✓ When you need explicit control



**Execution Flow:**

1. Initialize empty list: `result = []`
2. Loop from end to start: `for i in range(len(text) - 1, -1, -1)`
3. Each iteration: `result.append(text[i])`
4. Join all elements: `return "" .join(result)`

**Example with "Hello":**

Build list: `["o", "l", "l", "e", "H"]`

Join: `"olleH"`

**Time Complexity:**  $O(n)$  - loop runs n times

**Space Complexity:**  $O(n)$  - new list + string

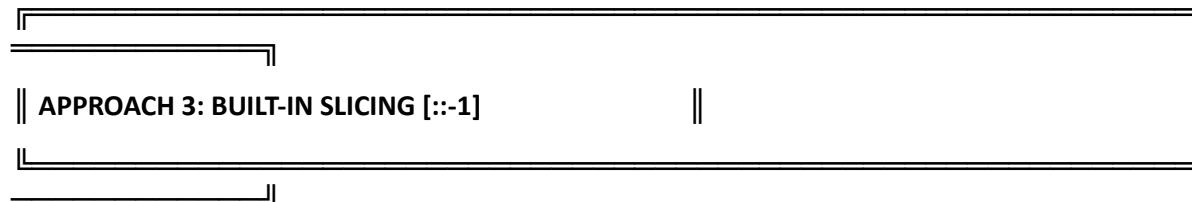
**Performance: FAST ✓**

**Why faster: `list.append()` is  $O(1)$ , `join()` is  $O(n)$**

**Total:  $O(n)$  which is optimal**

**Best For:**

- ✓ When you need explicit iteration logic
- ✓ Educational purposes (shows optimization technique)
- ✓ Adding custom processing during reversal
- ✓ Performance-conscious iterative code



**Execution Flow:**

1. Use Python slice notation: `text[::-1]`
2. `[:]` = from start to end
3. `-1` = step size (backwards)
4. Returns new reversed string

**Example with "Hello":**

`"Hello"[::-1]` = "olleH"

**Time Complexity:  $O(n)$  - must copy all characters**

**Space Complexity:  $O(n)$  - creates new reversed string**

**Performance: FASTEST ⚡ ⚡**

**Optimized at C level in CPython**

**Direct string reversal operation**

**Best For:**

- ✓ Production code (most Pythonic)
- ✓ General use cases
- ✓ Performance-critical code
- ✓ Readable and idiomatic Python
- ✓ Recommended by Python community



**Execution Flow:**

1. Create reverse iterator: `reversed(text)`
2. Join iterator into string: `"".join(...)`
3. Returns new reversed string

**Example with "Hello":**

```
reversed("Hello") → iterator  
"".join(iterator) = "olleH"
```

**Time Complexity:**  $O(n)$

**Space Complexity:**  $O(n)$

**Performance:** VERY FAST ✓✓

**Efficient iterator approach**

**Minimal overhead**

**Best For:**

- ✓ When you need an iterator
- ✓ Functional programming style

✓ Memory-efficient for large strings

✓ Pythonic alternative to slicing

""")

```
# =====  
# COMPREHENSIVE COMPARISON TABLE  
# =====
```

def print\_comparison\_table():

"""Prints comprehensive comparison table."""

print("\n" + "=" \* 70)

print("COMPREHENSIVE COMPARISON TABLE")

print("=" \* 70)

print("""

Criterion	Loop (concat)	Loop (list+join)	Slicing [::-1]
Time Complexity	$O(n^2)$ practical	$O(n)$ optimal	$O(n)$ optimal
Space Complexity	$O(n)$	$O(n)$	$O(n)$
Code Brevity	Medium (6 lines)	Medium (6 lines)	Very short (1)
Readability	Good	Good	Excellent
Performance	Slow ⚠️	Fast ✓	Fastest ⚡ ⚡
Pythonic Style	Not really	Somewhat	Yes ✓✓
Beginner Friendly	Yes ✓	Yes ✓	Somewhat
Extensibility	Easy ✓	Easy ✓	Hard
Production Ready	No	Yes ✓	Yes ✓✓
Large Input (1M)	SLOW ✗	FAST ✓	FAASTEST ⚡ ⚡

""")

```
# =====  
# WHEN TO USE EACH APPROACH  
# =====  
  
def print_recommendations():  
    """Prints recommendations for each approach."""  
  
    print("\n" + "=" * 70)  
    print("RECOMMENDATIONS - WHEN TO USE EACH APPROACH")  
    print("=" * 70)  
    print("")
```

#### USE LOOP-BASED (String Concatenation) WHEN:

- ✓ Learning Python / studying algorithms
- ✓ Need explicit control over each character
- ✓ Adding custom logic during reversal
- ✗ NOT recommended for production code
- ✗ NOT recommended for large strings

#### USE LOOP-BASED (List + Join) WHEN:

- ✓ Need explicit iteration with custom logic
- ✓ Processing each character before reversal
- ✓ Educational demonstrations
- ✓ Performance matters and explicit approach preferred
- ✓ Compatible with functional programming style

#### USE BUILT-IN SLICING [::-1] WHEN:

- ✓ Production code (RECOMMENDED)
- ✓ General string reversal needed
- ✓ Maximum performance required

- ✓ Clean and readable code preferred
- ✓ Most common use case
- ✓ Working with large strings
- ✓ Following Python best practices

USE reversed() FUNCTION WHEN:

- ✓ Working with iterators
  - ✓ Functional programming style
  - ✓ Memory efficiency important
  - ✓ Iterating without creating full string
  - ✓ Working with iterables (not just strings)
- """)

```
# =====
# MAIN EXECUTION
# =====

if __name__ == '__main__':
    # Test cases
    test_cases = [
        "Hello, World!",
        "Python",
        "racecar",
        "a" * 100, # Large string
    ]

    # Run demonstrations
    for test_string in test_cases:
        test_all_approaches(test_string)
```

```
# Performance comparison with larger string  
large_string = "Hello, World! " * 100 # 1400 characters  
performance_comparison(large_string, iterations=5000)
```

```
# Print detailed analysis  
print_detailed_comparison()
```

```
# Print comparison table  
print_comparison_table()
```

```
# Print recommendations  
print_recommendations()
```

```
print("\n" + "=" * 70)  
print("FINAL VERDICT")  
print("=" * 70)  
print("")
```

### RECOMMENDED FOR PRODUCTION: Slicing [::-1]

- Fastest performance
- Most Pythonic
- Cleanest code
- Best practices compliant

### RECOMMENDED FOR LEARNING: Loop-based approaches

- Understand algorithms
- Learn about optimization
- Educational value

### RECOMMENDED FOR PERFORMANCE: Either Slicing or reversed()

- Both have O(n) complexity
- Slicing is slightly faster in practice

""")

A screenshot of the Visual Studio Code (VS Code) interface. The workspace contains a project named 'assignment-1' which includes a file 'reverse\_string.py'. The code in the editor is:

```
1 # Accept user input
2 input_string = input('Enter a string to reverse: ')
3
4 # Reverse the string
5 reversed_string = input_string[::-1]
6
7 # Print the reversed string
8 print('Reversed string:', reversed_string)
```

The terminal at the bottom shows the command being run: `[Running] python -u "c:\Users\Wysnavi\OneDrive\Documents\AIAC\assignment-1\reverse_string.py"`. The output of the program is: `Enter a string to reverse: vishnavy`.

A screenshot of the Visual Studio Code (VS Code) interface. The workspace contains a project named 'assignment-1' which includes a file 'reverse\_string.py'. The code in the editor is identical to the first screenshot.

The terminal at the bottom shows the command being run: `PS C:\Users\Wysnavi\OneDrive\Documents\AIAC> 8 "c:\Users\Wysnavi\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "c:\Users\Wysnavi\OneDrive\Documents\AIAC\assignment-1\reverse_string.py"`. The output of the program is: `Enter a string to reverse: vishnavy`.



**File Edit Selection View Go Run Terminal Help**

**EXPLORER**

- OPEN EDITORS
  - reverse\_string.py
  - COMPARATIVE\_ANALYSIS.md
  - assignment-1
  - string\_reversal\_approaches.py
- AIAC
  - assignment-1
  - COMPARATIVE\_ANALYSIS.md
  - reverse\_string.py
  - string\_reversal\_approaches.py

**OUTLINE**

**TIMELINE**

**PROJECTS**

Indexing completed.

**Terminal**

```
assignment-1 > COMPARATIVE_ANALYSIS.md > # Comparative Analysis: Procedural vs Modular Approach
1  # Comparative Analysis: Procedural vs Modular Approach
10 ## Side-by-Side Code Comparison
17 ### Task 3: Modular Approach (With Functions)
35
36 ## Detailed Comparison Table
37
38 | Criteria | Procedural (Task 1) | Modular (Task 3) | Winner |
39 |-----|-----|-----|-----|
40 | **Code clarity** | ✓ Very concise (1 line) | ✓✓ Clear structure with docstrings | **Modular** |
41 | **Readability** | ✓ Simple but cryptic | ✓✓ Self-documenting with docstrings | **Modular** |
42 | **Reusability** | X Hard to reuse | ✓✓ Can import and use anywhere | **Modular** |
43 | **Testability** | X Not easy to unit test | ✓✓ Functions can be easily tested | **Modular** |
44 | **Debugging** | X Difficult to debug | ✓✓ Easy to trace and debug | **Modular** |
45 | **Maintenance** | X Hard to modify | ✓✓ Changes isolated to function | **Modular** |
46 | **Scalability** | X Not suitable for large projects | ✓✓ Ideal for enterprise applications | **Modular** |
47 | **Documentation** | X No docstrings | ✓✓ Comprehensive docstrings | **Modular** |
48 | **Error Handling** | X None | ✓ Can be extended | **Modular** |
49 | **Lines of Code** | 1 | 15+ | **Procedural** |
50
51 ---
52
53 ## Detailed Analysis
54
55 ### 1. Code Clarity
56 **Procedural Approach:***
57 - Extremely concise but requires deep understanding of Python slicing
58 - No comments explaining the logic
59 - Chain operations in one line makes it harder for beginners to follow
60
61 **Modular Approach:***
62 - Clear separation of concerns
63 - Each function has a specific purpose
64 - Docstrings explain parameters, returns, and complexity
65 - **Winner: Modular** ✓
66
67 ---
```

**CHAT**

REVERSING A STRING IN PYTHON

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

✓ Created string reversal approaches

file

> 1 file changed Keep Undo

+ COMPARATIVE\_ANALYSIS.md

Describe what to build next

Agent Auto

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Markdown

1209 ENG IN 09-01-2026

**File Edit Selection View Go Run Terminal Help**

**EXPLORER**

- OPEN EDITORS
  - reverse\_string.py
  - COMPARATIVE\_ANALYSIS.md
  - assignment-1
  - string\_reversal\_approaches.py
- AIAC
  - assignment-1
  - COMPARATIVE\_ANALYSIS.md
  - reverse\_string.py
  - string\_reversal\_approaches.py

**OUTLINE**

**TIMELINE**

**PROJECTS**

Indexing completed.

**Terminal**

```
assignment-1 > COMPARATIVE_ANALYSIS.md > # Comparative Analysis: Procedural vs Modular Approach
1  # Comparative Analysis: Procedural vs Modular Approach
2
3 ## Overview
4 This document compares two approaches to string reversal in Python:
5 - **Task 1 (Procedural)**: Direct implementation without user-defined functions
6 - **Task 3 (Modular)**: Function-based implementation with reusability
7
8 ---
9
10 ## Side-by-Side Code Comparison
11
12 ### Task 1: Procedural Approach (Without Functions)
13
14 print("Reversed string:", input("Enter a string to reverse: ")[::-1])
15
16
17 ### Task 3: Modular Approach (With Functions)
18
19 def reverse_string(text):
20     """Reverses the given string using Python's slicing method."""
21     return text[::-1]
22
23 def main():
24     """Main function that accepts user input and displays the reversed string."""
25     user_input = input("Enter a string to reverse: ")
26     result = reverse_string(user_input)
27     print(f"Original string: {user_input}")
28     print(f"Reversed string: {result}")
29
30 if __name__ == '__main__':
31     main()
32
33
34 ---
35
36 ## Detailed Comparison Table
37
```

**CHAT**

REVERSING A STRING IN PYTHON

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

✓ Created string reversal approaches

file

> 1 file changed Keep Undo

+ COMPARATIVE\_ANALYSIS.md

Describe what to build next

Agent Auto

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Markdown

1209 ENG IN 09-01-2026

The screenshot shows two instances of Microsoft Visual Studio Code running side-by-side. Both instances have the same workspace open, featuring an Explorer sidebar, an Open Editors panel, and a Terminal tab.

**Terminal Content:**

```
assignment-1 > reverse_string.py ...
```

```
48     # Test Case 3: Palindrome
49     print("Test Case 3 - Palindrome:")
50     test3 = 'racecar'
51     print(f'Input: {test3}')
52     print(f'Output: {reverse_string(test3)}\n')

53     # Test Case 4: Empty string
54     print("Test Case 4 - Empty String:")
55     test4 = ''
56     print(f'Input: {test4}')
57     print(f'Output: {reverse_string(test4)}\n')

58     # Test Case 5: Single character
59     print("Test Case 5 - Single Character:")
60     test5 = 'A'
61     print(f'Input: {test5}')
62     print(f'Output: {reverse_string(test5)}\n')

63     # Interactive mode
64     print('== Interactive Mode ==')
65     main()
```

**Terminal Tab:**

Ln 68, Col 11 | Spaces: 4 | UTF-8 | Python | 3.11.9 (Microsoft Store) | 12:07 | 09-01-2026 | ENG | IN

**Output Tab:**

```
==== String Reversal Program ====
Test Case 1 - Basic String:
Input: Hello
Output: olleH

Test Case 2 - String with Spaces:
Input: Hello, World!
Output: !dlroW ,olleH

Test Case 3 - Palindrome:
Input: racecar
Output: racecar
```

**Bottom Status Bar:**

Ln 68, Col 11 | Spaces: 4 | UTF-8 | Python | 3.11.9 (Microsoft Store) | 12:07 | 09-01-2026 | ENG | IN

The screenshot shows the Microsoft Visual Studio Code interface with the following details:

- File Structure:** The Explorer sidebar shows a project named "assignment-1" containing files "reverse\_string.py" and "reverse\_string.py ~ reverse\_string.py".
- Code Editor:** The main editor pane displays a Python script named "reverse\_string.py". The code implements a function `reverse\_string` that takes a string as input and returns its reverse. It includes several test cases: basic strings ("Hello"), strings with spaces ("Hello, World!"), palindromes ("racecar"), empty strings, and single characters ("A"). An interactive mode is also demonstrated.
- Terminal:** The Terminal tab is active, showing the output of running the script. The output matches the expected results for each test case.
- Output:** The Output tab shows no errors or warnings.
- Status Bar:** The status bar at the bottom indicates the current file is "reverse\_string.py", line 68, column 11, with 4 spaces, using UTF-8 encoding, and Python 3.11.9 (Microsoft Store).



**File Edit Selection View Go Run Terminal Help**

**OPEN EDITORS**

- reverse\_string.py
- COMPARATIVE\_ANALYSIS.md
- reverse\_string.py -- reverse\_string.py
- string\_reversal.approaches...
- AIAC
- assignment-1
- COMPARATIVE\_ANALYSIS.md
- reverse\_string.py
- string\_reversal.approaches.py

**EXPLORER**

**OUTLINE**

**TIMELINE**

**PROJECTS**

**REVERSE A STRING IN PYTHON**

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

✓ Created string reversal approaches

File

> 1 file changed Keep Undo

Describe what to build next

Agent Auto

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Markdown

1209 ENG IN 09-01-2026

---

**File Edit Selection View Go Run Terminal Help**

**OPEN EDITORS**

- reverse\_string.py
- COMPARATIVE\_ANALYSIS.md
- reverse\_string.py -- reverse\_string.py
- string\_reversal.approaches...
- AIAC
- assignment-1
- COMPARATIVE\_ANALYSIS.md
- reverse\_string.py
- string\_reversal.approaches.py

**EXPLORER**

**OUTLINE**

**TIMELINE**

**PROJECTS**

**REVERSE A STRING IN PYTHON**

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

✓ Created string reversal approaches

File

> 1 file changed Keep Undo

Describe what to build next

Agent Auto

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF {} Markdown

1209 ENG IN 09-01-2026

The screenshot shows a code editor interface with multiple windows and panels. The main window displays a Python script named `reverse_string.py` which contains code for reversing strings using procedural and modular approaches. The script includes comments explaining the differences between the two methods.

```
assignment-1 > COMPARATIVE_ANALYSIS.md > # Comparative Analysis: Procedural vs Modular Approach
  1  # Comparative Analysis: Procedural vs Modular Approach
  2  ## Detailed Analysis
  3  ### 1. Code Clarity
  4  ...
  5  ...
  6  ...
  7  ...
  8  ...
  9  ...
 10 ...
 11 ...
 12 ...
 13 ...
 14 ...
 15 ...
 16 ...
 17 ...
 18 ...
 19 ...
 20 ...
 21 ...
 22 ...
 23 ...
 24 ...
 25 ...
 26 ...
 27 ...
 28 ...
 29 ...
 30 ...
 31 ...
 32 ...
 33 ...
 34 ...
 35 ...
 36 ...
 37 ...
 38 ...
 39 ...
 40 ...
 41 ...
 42 ...
 43 ...
 44 ...
 45 ...
 46 ...
 47 ...
 48 ...
 49 ...
 50 ...
 51 ...
 52 ...
 53 ...
 54 ...
 55 ...
 56 ...
 57 ...
 58 ...
 59 ...
 60 ...
 61 ...
 62 ...
 63 ...
 64 ...
 65 ...
 66 ...
 67 ...
 68 ...
 69 ...
 70 ...
 71 ...
 72 ...
 73 ...
 74 ...
 75 ...
 76 ...
 77 ...
 78 ...
 79 ...
 80 ...
 81 ...
 82 ...
 83 ...
 84 ...
 85 ...
 86 ...
 87 ...
 88 ...
 89 ...
 90 ...
 91 ...
 92 ...
 93 ...
 94 ...
 95 ...
 96 ...
 97 ...
 98 ...
 99 ...
100 ...
```

The right-hand side of the interface features a sidebar titled "REVERSING A STRING IN PYTH..." which contains a detailed scenario and task description. The scenario discusses how AI handles alternative logic paths, while the task description outlines a comparison between loop-based and built-in slicing-based string reversal approaches, along with expected output and execution flow.