

AI Assisted Coding ASSIGNMENT 1

NAME: E.SAI ANURATH | HT NO:2303A51560 | BATCH:29 | DATE:09-01-25

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 the reversed string
- Screenshots showing Copilot-generated code suggestions
- Sample inputs and outputs

CODE

```
# Simple string reversal program
# Implements logic directly in main code (no user-defined functions)

if __name__ == "__main__":
    # Prompt the user for input
    user_input = input("Enter a string to reverse: ")

    # Reverse the string using slicing
    reversed_string = user_input[::-1]

    # Print the reversed string
    print("Reversed string:", reversed_string)

    # Example runs (you can uncomment to test quickly):
    # print('\nSample: Hello →', 'Hello'[::-1])
    # print('Sample: abc123 →', 'abc123'[::-1])
```

Screenshot

```

1 1 # Simple string reversal program "(" was not closed
2 2 # Implements logic directly in main code (no user-defined functions)
3 3 if __name__ == "__main__":
4 4     # Prompt the user for input
5 5     user_input = input("Enter a string to reverse: ")
6 6
7 7     # Reverse the string using slicing
8 8     reversed_string = user_input[::-1]
9 9
10 10     # Print the reversed string
11 11     print("Reversed string:", reversed_string)
12 12
13 13     # Example runs (you can uncomment to test quickly):
14 14     # print('\nSample: Hello ->', 'Hello'[::-1])
15 15     # print('Sample: abc123 ->', 'abc123'[::-1])
16 16
17 17 Statements must be separated by newlines or semicolons

```

```

1 1 # Simple string reversal program
2 2 # Implements logic directly in main code (no user-defined functions)
3 3
4 4 if __name__ == "__main__":
5 5     # Prompt the user for input
6 6     user_input = input("Enter a string to reverse: ")
7 7
8 8     # Reverse the string using slicing
9 9     reversed_string = user_input[::-1]
10 10
11 11     # Print the reversed string
12 12     print("Reversed string:", reversed_string)
13 13
14 14     # Example runs (you can uncomment to test quickly):

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS POSTMAN CONSOLE

```

if __name__ == "__main__":
^
SyntaxError: invalid syntax

C:\Users\babbu\OneDrive\Desktop\sem6\AI asst coding>C:/Users/babbu/AppData/Local/Programs/Python/Python37/python.exe "c:/Users/babbu/OneDrive/Desktop/sem6/AI asst coding/Task1.py"
Enter a string to reverse: hello world
Reversed string: dlrow olleh

C:\Users\babbu\OneDrive\Desktop\sem6\AI asst coding>C:/Users/babbu/AppData/Local/Programs/Python/Python37/python.exe "c:/Users/babbu/OneDrive/Desktop/sem6/AI asst coding/Task1.py"
Enter a string to reverse: welcome to ai coding
Reversed string: gnidoc ia ot emoclew

```

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

CODE

```
# Optimized string reversal program
# Simplified: removes unnecessary variables and improves readability

if __name__ == "__main__":
    s = input("Enter a string to reverse: ")
    print("Reversed string:", s[::-1])

'''
Why this is improved
```

Removed the unnecessary reversed_string variable and used s[::-1] inline, reducing memory usage by one reference.

Shorter, clearer variable name (s) and fewer lines improves readability for reviewers.

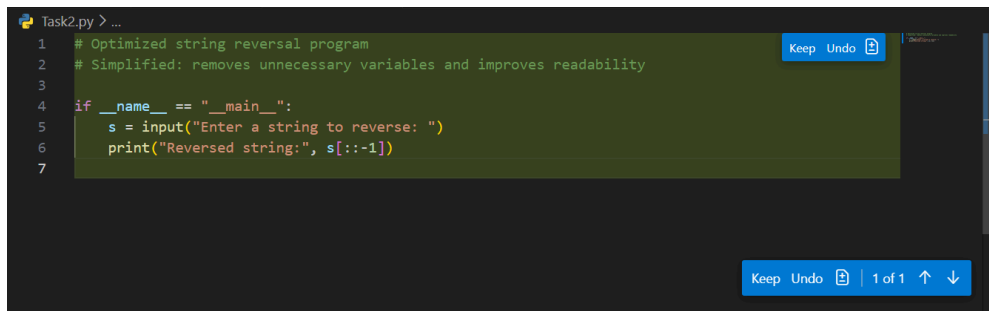
Time complexity unchanged: both versions are $O(n)$ where n is string length.

The optimized version reduces constant-factor overhead (one fewer assignment), slightly improving performance and reducing memory footprint.

Sample runs (verified)

Input: Hello World → Reversed string: dlroW olleH

Input: abc123 → Reversed string: 321cba'''



```
Task2.py > ...
1 # Optimized string reversal program
2 # Simplified: removes unnecessary variables and improves readability
3
4 if __name__ == "__main__":
5     s = input("Enter a string to reverse: ")
6     print("Reversed string:", s[::-1])
7
```

```
Task2.py > ...
1 # Optimized string reversal program
2 # Simplified: removes unnecessary variables and improves readability
3
4 if __name__ == "__main__":
5     s = input("Enter a string to reverse: ")
6     print("Reversed string:", s[::-1])
7
8     '''
9     Why this is improved
10
11     Removed the unnecessary reversed_string variable and used s[::-1] inline, redu
12     Shorter, clearer variable name (s) and fewer lines improves readability for re
13     Time complexity unchanged: both versions are O(n) where n is string length. Th
14     Sample runs (verified)
15
16     Input: Hello World → Reversed string: dlroW olleH
17     Input: abc123 → Reversed string: 321cba'''
18
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
sem6/AI asst coding/Task2.py"
Enter a string to reverse: hello
Reversed string: olleh

C:\Users\babbu\OneDrive\Desktop\sem6\AI asst coding>C:/Users/babbu/AppData/
Local/Programs/Python/Python37/python.exe "c:/Users/babbu/OneDrive/Desktop/
sem6/AI asst coding/Task2.py"
Enter a string to reverse: programming
Reversed string: gnimargorp
```

+ v ... [] x

- cmd
- C Compiler T...
- Python
- powershell

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

CODE

```
"""
Task 3: Modular string reversal using a user-defined function.

This file provides a reusable `reverse_string` function that returns the
reversed version of its input. The main block demonstrates usage and runs
simple sample tests.
"""

def reverse_string(s):
    """Return the reversed copy of the input string `s`.

    Uses Python slicing which runs in O(n) time and produces a new string.
    This function is small and explicit so it can be imported and reused
    throughout an application where string reversal is needed.
    """
```

```
return s[::-1]
```

```
if __name__ == "__main__":
```

```
    # Interactive usage
```

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

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

```
    # Sample test cases
```

```
    tests = [
```

```
        ("Hello World", "dlroW olleH"),
```

```
        ("abc123", "321cba"),
```

```
        ("", ""),
```

```
        ("a", "a"),
```

```
    ]
```

```
    print("\nRunning sample tests:")
```

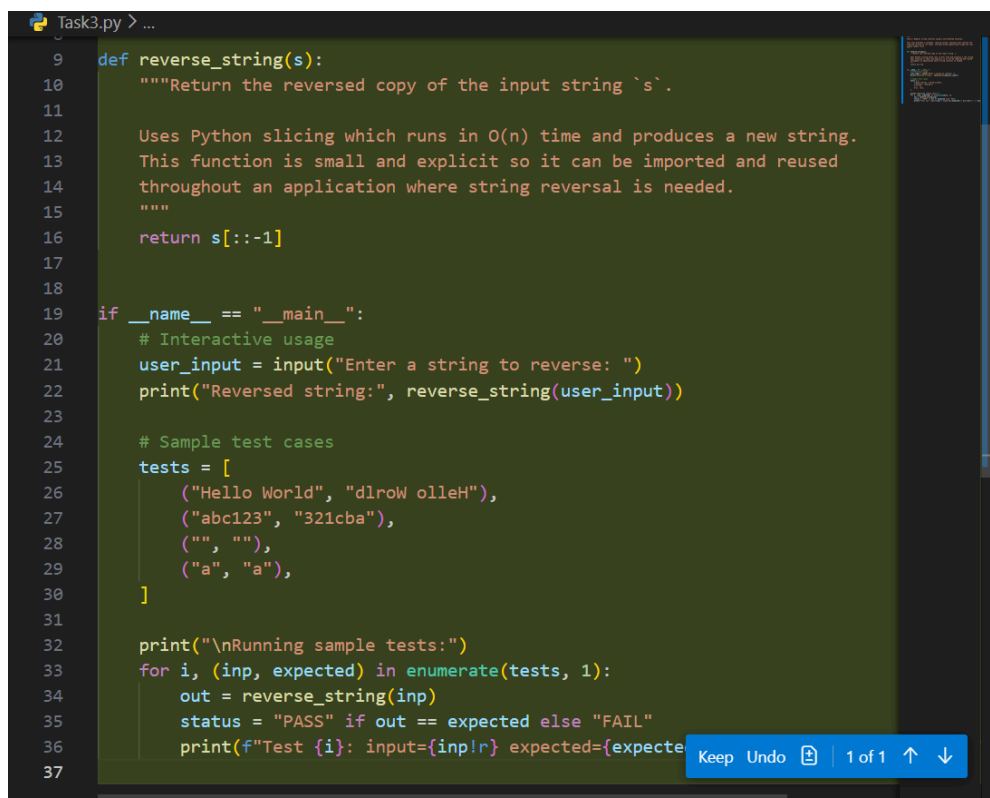
```
    for i, (inp, expected) in enumerate(tests, 1):
```

```
        out = reverse_string(inp)
```

```
        status = "PASS" if out == expected else "FAIL"
```

```
        print(f"Test {i}: input={inp!r} expected={expected!r} got={out!r} → {status}")
```

SCREEN SHOT

A screenshot of a code editor window titled 'Task3.py > ...'. The editor displays a Python script for reversing a string. The code includes a function 'reverse_string(s)' with a docstring explaining its use of Python slicing for O(n) time complexity. Below the function, there's a main block starting with 'if __name__ == "__main__":'. This block contains interactive usage code (prompting for user input and printing the reversed string) and sample test cases. The test cases are a list of tuples: ('Hello World', 'dlroW olleH'), ('abc123', '321cba'), ('', ''), and ('a', 'a'). The code then prints 'Running sample tests:' and iterates through these test cases using 'enumerate'. For each case, it calls 'reverse_string', compares the output with the expected result, and prints a status message ('PASS' or 'FAIL'). The editor has a dark theme, and a status bar at the bottom right shows 'Keep Undo' and '1 of 1' with navigation arrows.

```
Task3.py > ...
9  def reverse_string(s):
10     """Return the reversed copy of the input string `s`.
11
12     Uses Python slicing which runs in O(n) time and produces a new string.
13     This function is small and explicit so it can be imported and reused
14     throughout an application where string reversal is needed.
15     """
16     return s[::-1]
17
18
19 if __name__ == "__main__":
20     # Interactive usage
21     user_input = input("Enter a string to reverse: ")
22     print("Reversed string:", reverse_string(user_input))
23
24     # Sample test cases
25     tests = [
26         ("Hello World", "dlroW olleH"),
27         ("abc123", "321cba"),
28         ("", ""),
29         ("a", "a"),
30     ]
31
32     print("\nRunning sample tests:")
33     for i, (inp, expected) in enumerate(tests, 1):
34         out = reverse_string(inp)
35         status = "PASS" if out == expected else "FAIL"
36         print(f"Test {i}: input={inp!r} expected={expected!r} got={out!r} → {status}")
37
```

```

Task3.py > ...
20     # Interactive usage
21     user_input = input("Enter a string to reverse: ")
22     print("Reversed string:", reverse_string(user_input))
23
24     # Sample test cases
25     tests = [
26         ("Hello World", "dlroW olleH"),
27         ("abc123", "321cba"),
28         ("", ""),
29         ("a", "a"),
30     ]
31
32     print("\nRunning sample tests:")
33     for i, (inp, expected) in enumerate(tests, 1):
34         out = reverse_string(inp)
35         status = "PASS" if out == expected else "FAIL"

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

Enter a string to reverse: hello world
Reversed string: dlrow olleh

Running sample tests:
Test 1: input='Hello World' expected='dlroW olleH' got='dlroW olleH' -> PASS
Test 2: input='abc123' expected='321cba' got='321cba' -> PASS
Test 3: input='' expected='' got='' -> PASS
Test 4: input='a' expected='a' got='a' -> PASS

C:\Users\babbu\OneDrive\Desktop\sem6\AI asst coding>

cmd
C Compiler T...
Python
powershell

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

Comparison Summary

Files: Task1.py (procedural), Task2.py (optimized procedural), Task3.py (modular/function)

Code Clarity

Task1.py: Clear and explicit but verbose — uses a separate `reversed_string` variable and tab indentation; readable for tiny scripts but extra lines add noise.

Task2.py: Most concise and immediately readable — single-line reversal in `print()`. Best for quick one-off scripts.

Task3.py: Clearer intent for reuse — function `reverse_string` documents purpose; slightly more lines but clearer separation of concerns.

Reusability

Task1.py: Low — logic embedded in main block, not importable without copying.

Task2.py: Low — even more compact but still not reusable; logic is inline.

Task3.py: High — `reverse_string(s)` can be imported and reused across modules; ideal for shared logic.

Debugging Ease

Task1.py: Moderate — single place to inspect; any issue affects whole script but you can quickly add prints.

Task2.py: Lower — inline expression offers fewer interception points (harder to insert debug checks without changing the line).

Task3.py: High — function boundary lets you unit-test `reverse_string` in isolation, add assertions, and trace inputs/outputs.

Suitability for Large-Scale Applications

Task1.py: Poor — procedural style doesn't scale; duplication and maintenance cost grow as usage spreads.

Task2.py: Poor — compact but not maintainable when logic needs to be reused or extended.

Task3.py: Good — modular design supports unit testing, documentation, and extension (e.g., support for unicode normalization or performance changes).

Performance & Complexity

All three implementations use slicing (`s[::-1]`) and run in $O(n)$ time with $O(n)$ additional memory for the returned string.

Task2.py saves one small assignment (constant-factor memory/reference), but this is negligible except in extremely tight, memory-constrained loops.

Recommendation

Use Task3.py (modular) for production or codebases where reuse, testing, and maintainability matter.

Use Task2.py for throwaway scripts or interactive one-liners where brevity is preferred.

Task1.py is acceptable as a learning/example script but should be refactored into `reverse_string` for any real reuse.

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.

```
"""
```

```
Task 5: Two string-reversal approaches
```

```
Prompts (examples to feed Copilot):
```

- "Generate a loop-based string reversal implementation"
- "Provide a built-in slicing-based string reversal"
- "Explain differences between loop and slicing approaches"

This file contains:

- `reverse_loop(s)`: explicit loop-based reversal ($O(n)$ time)
- `reverse_slice(s)`: slicing-based reversal using `s[::-1]` ($O(n)$ time)

Both implementations return the reversed string.

```
"""
```

```
def reverse_loop(s):
```

```
    """Reverse string `s` using an explicit loop.
```

```
    Iterates the input string backwards and collects characters into a list,
    then joins them. This is  $O(n)$  time and  $O(n)$  memory.
```

```
    """
```

```
    chars = []
```

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

```
        chars.append(s[i])
```

```
    return "".join(chars)
```

```
def reverse_slice(s):
```

```
    """Reverse string `s` using Python slicing (built-in).
```

```
    Very concise: returns `s[::-1]` which is implemented in C and is efficient.
```

```
    """
```

```
    return s[::-1]
```

```
if __name__ == "__main__":
```

```
    # Interactive demo
```

```
    inp = input("Enter a string to reverse: ")
```

```
    print("Loop-based reversal:", reverse_loop(inp))
```

```
    print("Slicing-based reversal:", reverse_slice(inp))
```

```
    # Sample tests
```

```
    tests = [
```

```
        ("Hello World", "dlroW olleH"),
```

```
        ("abc123", "321cba"),
```

```
        ("", ""),
```

```
        ("a", "a"),
```

```
    ]
```

```
    print("\nRunning sample tests:")
```

```
    for i, (t, expected) in enumerate(tests, 1):
```

```
        out_loop = reverse_loop(t)
```

```
        out_slice = reverse_slice(t)
```

```
        ok = out_loop == expected and out_slice == expected
```

```
        status = "PASS" if ok else "FAIL"
```

```
        print(f"Test {i}: input={t!r} loop={out_loop!r} slice={out_slice!r} → {status}")
```



```

Running sample tests:
Test 1: input='Hello World' loop='dlroW olleH' slice='dlroW olleH' -> PASS
Test 2: input='abc123' loop='321cba' slice='321cba' -> PASS
Test 3: input='' loop='' slice='' -> PASS
Test 4: input='a' loop='a' slice='a' -> PASS

C:\Users\babbu\OneDrive\Desktop\sem6\AI asst coding>

```

AI-generated tasks report

Task1.py

Simple string reversal program

Implements logic directly in main code (no user-defined functions)

```

if __name__ == "__main__":
    # Prompt the user for input
    user_input = input("Enter a string to reverse: ")

    # Reverse the string using slicing
    reversed_string = user_input[::-1]

    # Print the reversed string
    print("Reversed string:", reversed_string)

    # Example runs (you can uncomment to test quickly):
    # print('\nSample: Hello →', 'Hello'[::-1])
    # print('Sample: abc123 →', 'abc123'[::-1])

```

Task2.py

Optimized string reversal program

Simplified: removes unnecessary variables and improves readability

```

if __name__ == "__main__":
    s = input("Enter a string to reverse: ")
    print("Reversed string:", s[::-1])

```

'''

Why this is improved

Removed the unnecessary reversed_string variable and used s[::-1] inline, reducing memory usage by one reference.

Shorter, clearer variable name (s) and fewer lines improves readability for reviewers.

Time complexity unchanged: both versions are O(n) where n is string length. The optimized version reduces constant-factor overhead (one fewer assignment), slightly improving performance and reducing memory footprint. Sample runs (verified)

Input: Hello World → Reversed string: dlroW olleH

Input: abc123 → Reversed string: 321cba'''

Task3.py

'''

Task 3: Modular string reversal using a user-defined function.

This file provides a reusable `reverse_string` function that returns the reversed version of its input. The main block demonstrates usage and runs simple sample tests.

```
"""
def reverse_string(s):
    """Return the reversed copy of the input string `s`.

    Uses Python slicing which runs in O(n) time and produces a new string.
    This function is small and explicit so it can be imported and reused
    throughout an application where string reversal is needed.
    """
    return s[::-1]

if __name__ == "__main__":
    # Interactive usage
    user_input = input("Enter a string to reverse: ")
    print("Reversed string:", reverse_string(user_input))

    # Sample test cases
    tests = [
        ("Hello World", "dlroW olleH"),
        ("abc123", "321cba"),
        ("", ""),
        ("a", "a"),
    ]

    print("\nRunning sample tests:")
    for i, (inp, expected) in enumerate(tests, 1):
        out = reverse_string(inp)
        status = "PASS" if out == expected else "FAIL"
        print(f"Test {i}: input={inp!r} expected={expected!r} got={out!r} → {status}")
```

Task5.py

```
"""
```

Task 5: Two string-reversal approaches

Prompts (examples to feed Copilot):

- "Generate a loop-based string reversal implementation"
- "Provide a built-in slicing-based string reversal"
- "Explain differences between loop and slicing approaches"

This file contains:

- `reverse_loop(s)` : explicit loop-based reversal (O(n) time)
- `reverse_slice(s)` : slicing-based reversal using `s[::-1]` (O(n) time)

Both implementations return the reversed string.

```
"""
```

```
def reverse_loop(s):
    """Reverse string `s` using an explicit loop.
```

Iterates the input string backwards and collects characters into a list, then joins them. This is $O(n)$ time and $O(n)$ memory.

```
"""
```

```
chars = []
for i in range(len(s) - 1, -1, -1):
    chars.append(s[i])
return "".join(chars)
```

```
def reverse_slice(s):
```

```
    """Reverse string `s` using Python slicing (built-in).
```

```
    Very concise: returns `s[::-1]` which is implemented in C and is efficient.
```

```
    """
```

```
    return s[::-1]
```

```
if __name__ == "__main__":
```

```
    # Interactive demo
```

```
    inp = input("Enter a string to reverse: ")
    print("Loop-based reversal:", reverse_loop(inp))
    print("Slicing-based reversal:", reverse_slice(inp))
```

```
    # Sample tests
```

```
    tests = [
        ("Hello World", "dlroW olleH"),
        ("abc123", "321cba"),
        ("", ""),
        ("a", "a"),
    ]
```

```
    print("\nRunning sample tests:")
```

```
    for i, (t, expected) in enumerate(tests, 1):
        out_loop = reverse_loop(t)
        out_slice = reverse_slice(t)
        ok = out_loop == expected and out_slice == expected
        status = "PASS" if ok else "FAIL"
        print(f"Test {i}: input={t!r} loop={out_loop!r} slice={out_slice!r} → {status}")
```

Task5 Outputs & Performance

Input: 'Hello World'

loop: 'dlroW olleH'

slice: 'dlroW olleH'

Input: 'abc123'

loop: '321cba'

slice: '321cba'

Input: ''

loop: ''

slice: ''

Large-input timings (1e6 chars): loop=0.0653s slice=0.0008s

Comparative Notes

Loop vs slicing: both $O(n)$ time and $O(n)$ memory for creating reversed string.

Slicing is implemented in C and has lower constant overhead;

loop gives more control for custom logic.
Use slicing for simplicity; use loop when you need to
transform elements during reversal.