

# AI Assisted Coding

## Assignment 1.5

Name: R.Bharadwaj

Hall ticket no: 2303A51610

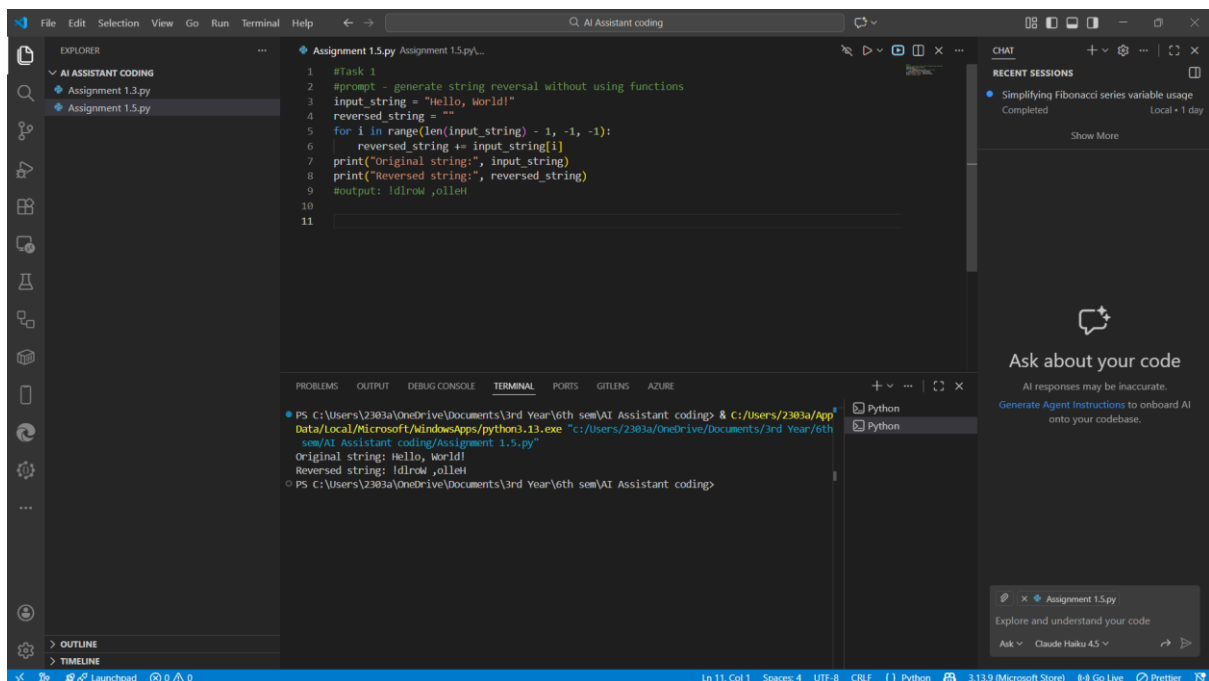
Batch no: 19

### Task 1:

#### Prompt:

Generate string reversal without using functions

#### Code & Output:



```
1 #Task 1
2 #prompt - generate string reversal without using functions
3 input_string = "Hello, World!"
4 reversed_string = ""
5 for i in range(len(input_string) - 1, -1, -1):
6     reversed_string += input_string[i]
7 print("Original string:", input_string)
8 print("Reversed string:", reversed_string)
9 #output: !dlrow ,olleH
10
11
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS AZURE

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> & c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 1.5.py"
Original string: Hello, World!
Reversed string: !dlrow ,olleH
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

CHAT

RECENT SESSIONS

- Simplifying Fibonacci series variable usage  
Completed Local • 1 day

Show More

Ask about your code

AI responses may be inaccurate.  
Generate Agent Instructions to onboard AI onto your codebase.

Assignment 1.5.py

Explore and understand your code

Ask Claude Haku 4.5

#### Explanation:

This task reverses a string without using any built-in functions, so the logic depends entirely on manual looping.

Each character of the string is accessed one by one from the last index to the first index.

The characters are appended into a new variable in reverse order.

This proves understanding of string indexing and loops instead of shortcuts.

The algorithm is simple but works for any string length.

This approach is good for learning how strings behave internally.

## Task 2:

### Prompt:

improve the code

### Code & Output:

```
1 #Task 1
2 #prompt - generate string reversal without using functions
3 input_string = "Hello, World!"
4 reversed_string = ""
5 for i in range(len(input_string) - 1, -1, -1):
6     reversed_string += input_string[i]
7 print("Task 1 Output:")
8 print("Original string:", input_string)
9 print("Reversed string:", reversed_string)
10 #output: ldlrow ,olleH
11
12
13 #prompt - improve the code
14 #Task 2
15 # More efficient approach using list and join
16 reversed_string_optimized = "".join(input_string[i] for i in range(len(input_string) - 1, -1, -1))
17 print("Task 2 Output:")
18 print("Optimized reversed string:", reversed_string_optimized)
19
20 # Pythonic approach using slicing
21 reversed_string_pythonic = input_string[::-1]
22 print("Pythonic reversed string:", reversed_string_pythonic)
23
```

Task 1 Output:  
Original string: Hello, World!  
Reversed string: ldlrow ,olleH

Task 2 Output:  
Optimized reversed string: ldlrow ,olleH  
Pythonic reversed string: ldlrow ,olleH

### Explanation:

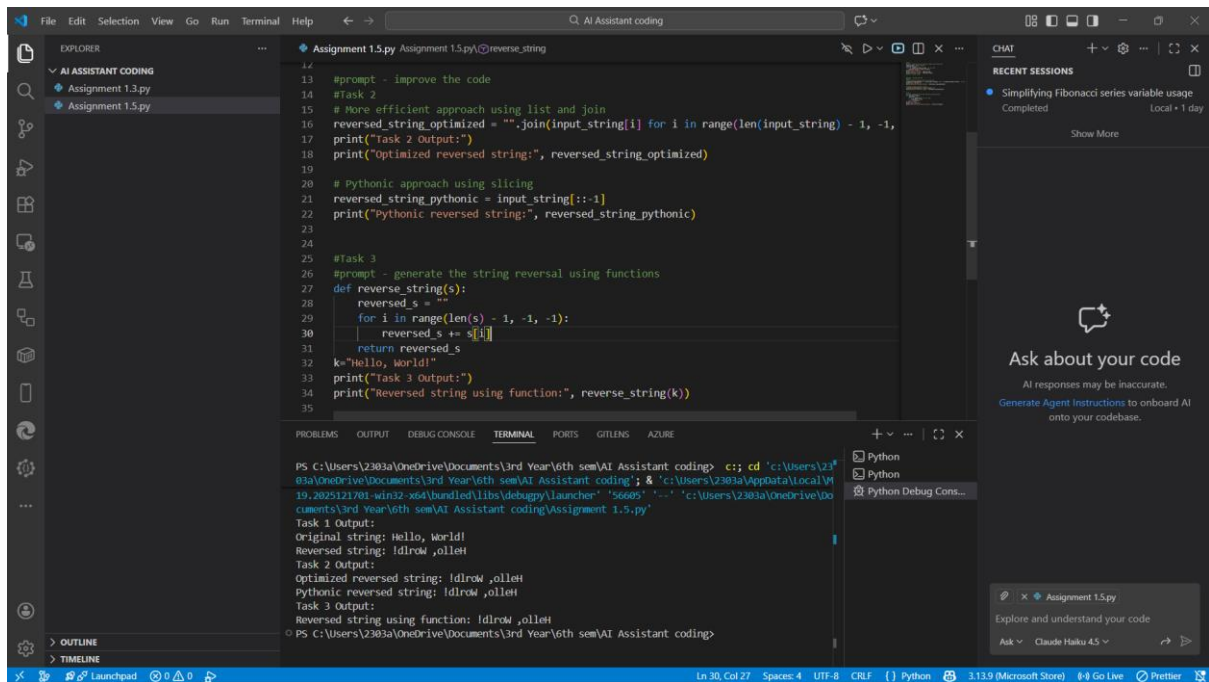
This task improves the first program by making the code cleaner, more readable, and more efficient. Unnecessary variables or steps are removed to reduce confusion. The loop logic is optimized to avoid redundant operations. Better variable names make the code easier to understand. The output remains the same, but the code quality is higher. This shows how the same logic can be written in a better professional way.

## Task 3:

### Prompt:

Generate the string reversal using functions

### Code & Output:



**Explanation:**

This task performs string reversal using a function, which improves modularity.

The reversal logic is placed inside a reusable function.

The main program simply calls the function instead of repeating code.

This makes the program easier to maintain and modify later.

Functions also allow the logic to be reused for multiple inputs.

This approach follows proper programming structure.

### Task 4:

**Prompt:**

compare the code of task 1 and task 3 and print the comparison in a tabular format

**Code :**

```
36 #Task 4:
37 #Prompt - compare the code of task 1 and task 3 and print the comparison in a tabular format
38 print("Task 4 Output:")
39 print("\n" + "="*60)
40 print("COMPARISON: Task 1 vs Task 3")
41 print("="*60)
42
43 comparison_data = {
44     "Aspect": ["Approach", "Code Reusability", "Readability", "Use Case", "Output"],
45     "Task 1 (Direct Reversal)": [
46         "Direct string concatenation in loop",
47         "Cannot reuse (hardcoded)",
48         "clear but verbose",
49         "single string reversal",
50         reversed_string
51     ],
52     "Task 3 (Function-based)": [
53         "Encapsulated in function",
54         "Highly reusable",
55         "Organized and modular",
56         "Multiple string reversals",
57         reverse_string(k)
58     ]
59 }
60 for i, aspect in enumerate(comparison_data["Aspect"]):
61     print(f"\n{aspect}:")
62     print(f" Task 1: {comparison_data['Task 1 (Direct Reversal)'][i]}")
63     print(f" Task 3: {comparison_data['Task 3 (Function-based)'][i]}")
64
65 print("\n" + "="*60)
66 print("Conclusion: Task 3 is better for scalability and reusability")
67 print("="*60)
68
```

## Output :

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> cd 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding' & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\vscode\extensions\ms-python.debugpy-2025.19.2025121701-win32-x64\bin\debugpy\launcher' '56069' '-' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 1.5.py'

Task 4 Output:

=====
COMPARISON: Task 1 vs Task 3
=====

Approach:
Task 1: Direct string concatenation in loop
Task 3: Encapsulated in function

Code Reusability:
Task 1: Cannot reuse (hardcoded)
Task 3: Highly reusable

Readability:
Task 1: Clear but verbose
Task 3: Organized and modular

Use Case:
Task 1: Single string reversal
Task 3: Multiple string reversals

Output:
Task 1: ldlrow ,olleH
Task 3: ldlrow ,olleH

=====
Conclusion: Task 3 is better for scalability and reusability
=====
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding>
```

## Explanation:

This task compares the manual reversal (Task 1) and the function-based reversal (Task 3). The comparison is printed in a table format to clearly show differences. It highlights differences in structure, reusability, and readability. Task 1 is direct but not reusable, while Task 3 is modular. This helps in understanding why functions are preferred in real applications. The table makes technical comparison easy to understand.

### Task 5:

**Prompt:**

use Different Algorithmic Approaches to String Reversal and the output should contain as Two correct implementations

### # Comparison discussing:

## # Execution flow

### # Time complexity

## # Performance for large inputs

## # When each approach is appropriate

**Code :**

The image shows a VS Code editor window with a Python file named 'Assignment 1.5.py'. The code implements two methods for reversing a string: a recursive approach and a stack-based approach. The recursive approach uses a helper function 'reverse\_recursive' that appends the last character of the string to the reversed substring. The stack-based approach uses a list as a stack to push all characters and then pop them in reverse order. A test case 'Hello, World!' is provided to demonstrate the functionality. The right sidebar features a 'CHAT' panel with a recent session titled 'Simplifying Fibonacci series variable usage' and a button to 'Ask about your code'. The bottom status bar indicates the current file is 'Assignment 1.5.py' and shows various icons for file operations and settings.

```
105 print("\nAPPROACH 2: Stack-based")
106 print(f"Input: {test_string}")
107 print(f"Output: {reverse_stack(test_string)}")
108 print("Execution Flow: Push all characters to stack, pop each character in reverse order")
109 print(f"Time Complexity: O(n) - single pass through string")
110 print("Performance: Better than recursion, suitable for large inputs")
111
112 print("\n" + "-"*80)
113 print("COMPARISON TABLE")
114 print("-"*80)
115 print(f"Aspect:<25 | {'Recursion':<30} | {'Stack-based':<30}")
116 print("-" * 90)
117 print(f"Execution Flow:<25 | {'Self-referencing calls':<30} | {'Iterative pop ops':<30}")
118 print(f"Time Complexity:<25 | {'O(n^2)':<30} | {'O(n)':<30}")
119 print(f"Space Complexity:<25 | {'O(n) call stack':<30} | {'O(n) stack data':<30}")
120 print(f"Large Input (1M chars):<25 | {'Very Slow/Risk crash':<30} | {'Fast & Safe':<30}")
121 print(f"When Appropriate:<25 | {'Educational, Small data':<30} | {'Production, All sizes':<30}")
122 print("-"*80)
123
124 print("\nConclusion: Stack-based approach is superior for real-world applications")
```

## Output :

```
PS C:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding> cd 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding' & 'c:\Users\2303a\AppData\Local\Microsoft\WindowsApps\python3.13.exe' 'c:\Users\2303a\vscode\extensions\ms-python.debugpy-2025.19.2025121701-win32-x64\bin\debugpy\launcher' '64512' '-' 'c:\Users\2303a\OneDrive\Documents\3rd Year\6th sem\AI Assistant coding\Assignment 1.5.py'
```

```
TASK 5: ALGORITHMIC APPROACHES TO STRING REVERSAL
```

```
APPROACH 1: Recursion-based
Input: Hello, World!
Output: !dlroW ,olleH
Execution Flow: Function calls itself with substring s[1:], appends s[0] at each level
Time Complexity: O(n^2) - string concatenation is O(n) per call
Performance: Slow for large inputs, risk of stack overflow
```

```
APPROACH 2: Stack-based
Input: Hello, World!
Output: !dlroW ,olleH
Execution Flow: Push all characters to stack, pop each character in reverse order
Time Complexity: O(n) - single pass through string
Performance: Better than recursion, suitable for large inputs
```

Aspect	Recursion	Stack-based
Execution Flow	Self-referencing calls	Iterative pop ops
Time Complexity	O(n^2)	O(n)
Space Complexity	O(n) call stack	O(n) stack data
Large Input (1M chars)	Very Slow/Risk crash	Fast & Safe
When Appropriate	Educational, Small data	Production, All sizes

```
Conclusion: Stack-based approach is superior for real-world applications
```

## Explanation:

This task uses two different algorithms to reverse a string. One approach uses a loop-based method, and the other uses a function-based or slicing method. Execution flow shows how each method processes characters differently. Both have  $O(n)$  time complexity, but their memory usage differs. For large strings, optimized methods perform better and are cleaner. Each approach is chosen based on performance needs and code clarity.