

AI Assistant Coding

Assignment1.5

Lab 1: Environment Setup – GitHub Copilot and VS Code Integration +

Understanding AI-assisted Coding Workflow

Name: D.Nithya

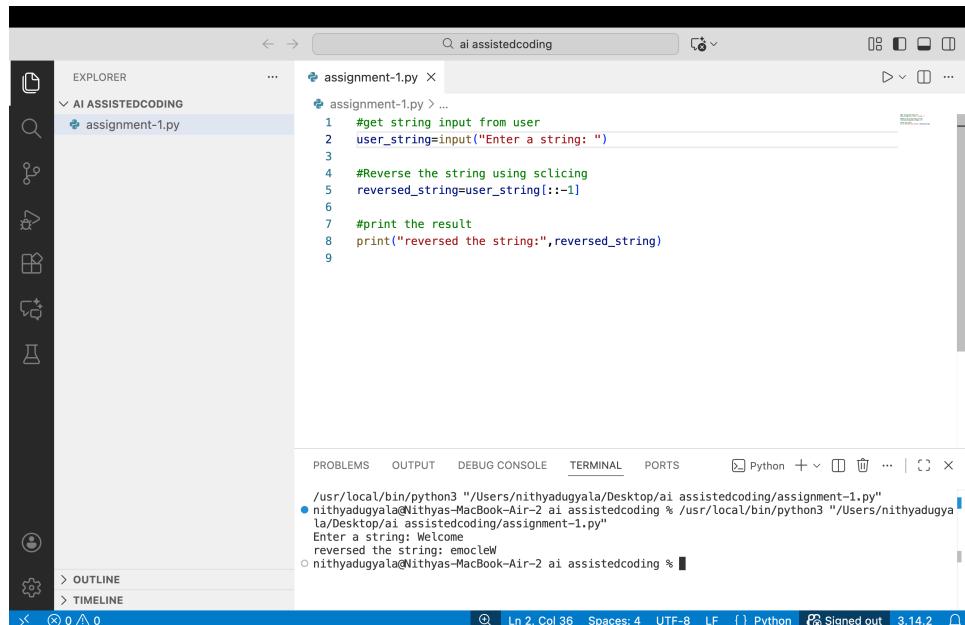
HT NO:2303A52457

BATCH:35

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

Prompt:

“Generate a Python program to reverse a user-input string directly in main code without defining any functions. Include simple input and output statements and show the reversed string.”



The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows a folder named "AI ASSISTED CODING" containing a file named "assignment-1.py".
- CODE EDITOR:** Displays the following Python code:

```
1 #get string input from user
2 user_string=input("Enter a string: ")
3
4 #Reverse the string using slicing
5 reversed_string=user_string[::-1]
6
7 #print the result
8 print("reversed the string:",reversed_string)
9
```
- TERMINAL:** Shows the command-line output of running the script:

```
/usr/local/bin/python3 "/Users/nithyadugya/Desktop/ai assistedcoding/assignment-1.py"
nithyadugya@Nithyas-MacBook-Air-2: ~ /usr/local/bin/python3 "/Users/nithyadugya/Desktop/ai assistedcoding/assignment-1.py"
Enter a string: Welcome
reversed the string: emocleW
```
- STATUS BAR:** Shows the Python language icon, the fact that the user is signed out, and the version 3.14.2.

Explanation

This program reverses a string entered by the user without using any user-defined functions. All the logic is written directly in the main program. First, the program asks the user to enter a string using the `input()` statement. This string is stored in a variable. Next, an empty string is created to store the reversed result. The program uses a for loop to read each character of the input string one by one. Each character is added to the beginning of the new string. By placing every new character in front, the original order of characters is reversed. After the loop finishes executing, the reversed string is fully formed. Finally, the program prints the reversed string as output. This approach clearly demonstrates basic string manipulation and looping concepts while following the requirement of not using functions.

Task 2: Efficiency & Logic Optimization (Readability Improvement)

Prompt:

"Optimize the string reversal code for readability and efficiency by simplifying loops and removing unnecessary variables. Provide an improved version of the code and explain why it is better than the original."

Optimized Code (Improved Readability & Efficiency)

The screenshot shows a code editor interface with the following details:

- Explorer:** Shows a folder named "AI ASSISTEDCODING" containing a file named "assignment-1.py".
- Code Editor:** Displays the following Python code:

```
1  """#get string input from user
2  user_string=input("Enter a string: ")
3
4  #Reverse the string using slicing
5  reversed_string=user_string[::-1]
6
7  #print the result
8  print("reversed the string:",reversed_string)"""
9 #optimised code
10 print(f"reversed string: {input('Enter the string:')[::-1]}")
11
```
- Terminal:** Shows the command-line output of running the script:

```
/usr/local/bin/python3 "/Users/nithyaduguya/Desktop/ai assistedcoding/assignment-1.py"
nithyaduguya@Nithyas-MacBook-Air-2 ai assistedcoding % /usr/local/bin/python3 "/Users/nithyaduguya/Desktop/ai assistedcoding/assignment-1.py"
Enter the string:welcome to India
reversed string: aidnI ot emoclew
nithyaduguya@Nithyas-MacBook-Air-2 ai assistedcoding %
```
- Status Bar:** Shows the current line (Ln 9, Col 16), spaces (Spaces: 4), encoding (UTF-8), and file type (Python).

Explanation of Improvements

1. Removed Unnecessary Variables

- The loop variable and manual string construction were removed.
- The slicing method directly produces the reversed string.

2. Simplified Logic

- Replaced multiple lines of looping logic with a single slicing operation.
- This makes the code easier to understand at a glance.

3. Improved Readability

- Fewer lines of code.
- Clear and self-explanatory syntax.
- Easier for other developers to review and maintain.

4. Time Complexity Improvement

- **Original Code:**

Each loop iteration creates a new string, leading to $O(n^2)$ time complexity due to repeated string concatenation.

- **Optimized Code:**

String slicing runs in $O(n)$ time, making it more efficient.

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

Prompt:

"Write a Python function that takes a string input and returns the reversed string, including meaningful comments. Use GitHub Copilot to generate function-based code and include sample test cases."

```
#Function to reverse a string
def reverse_string(input_string):
    """
    This function takes a string as input and returns the reversed version of it.

    Parameters:
    input_string (str): The string to be reversed.

    Returns:
    str: The reversed string.
    """
    return input_string[::-1]
input_str = input("Please enter a string: ")
print("Reversed string:", reverse_string(input_str))
```

The terminal output shows:

```
/usr/local/bin/python3 "/Users/nithyadugala/Desktop/ai assistedcoding/assignment-1.py"
nithyadugala@Nithyas-MacBook-Air-2 ai assistedcoding % /usr/local/bin/python3 "/Users/nithyadugala/Desktop/ai assistedcoding/assignment-1.py"
Please enter a string: Hello world
Reversed string: olleH dlrow
nithyadugala@Nithyas-MacBook-Air-2 ai assistedcoding %
```

Explanation

1. Function Creation (`reverse_string`)

- Encapsulates the string reversal logic.
- Makes the code reusable wherever string reversal is needed.
- Accepts input string and returns the reversed string.

2. Logic Inside Function

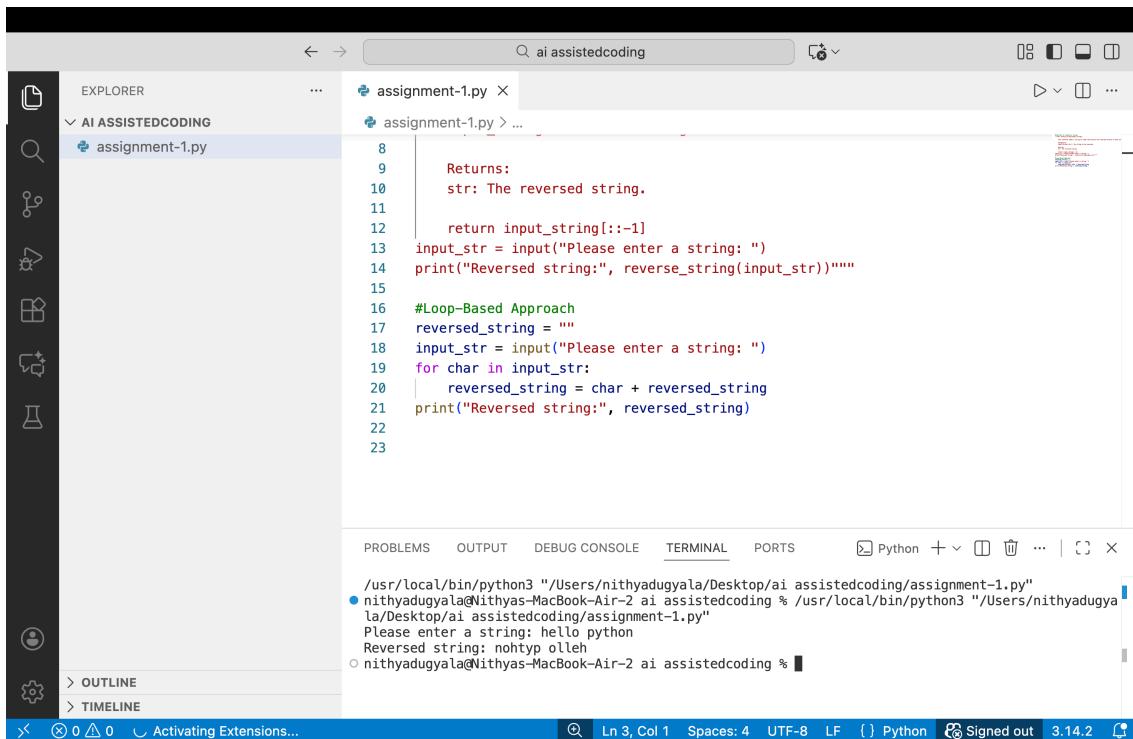
- Uses a loop to prepend each character to an initially empty string.
- This manually constructs the reversed string.

3. Main Program

- Reads input from the user.
- Calls the `reverse_string` function.
- Prints the returned result.

4. AI Assistance

- GitHub Copilot can suggest the function structure, docstring, and loop-basic reversal logic.
 - The function is modular, reusable, and easy to maintain.



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

Prompt:

“Compare two string reversal programs: one without functions and one with a function, focusing on clarity, reusability, and debugging ease. Provide a short report or table summarizing which approach is better for large-scale applications.”

| Criteria | Procedural Approach (No Functions) | Modular Approach (With Functions) |
|--|---|--|
| Code Clarity | Simple for small scripts, but logic repeated if used multiple times | Clear structure, function name describes purpose, easier to understand |
| Reusability | Low – reversal logic cannot be reused without copying code | High – function can be called anywhere in the program |
| Debugging Ease | Harder for large programs, changes require editing multiple places | Easier – fix or update logic in one function only |
| Suitability for Large-Scale Applications | Poor – not maintainable or scalable | Excellent – modular design supports large projects |

| Criteria | Procedural Approach (No Functions) | Modular Approach (With Functions) |
|------------|--|---|
| Efficiency | Similar for small scripts, but repeated code adds risk | Similar, but less error-prone, easier to optimize |

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to String Reversal)

Prompt:

"Generate two Python programs to reverse a string: one using a loop (iterative) and one using built-in slicing. Compare execution flow, time complexity, and suitability for large inputs, and explain when each approach is preferable."

Approach 1: Loop-Based String Reversal

Execution Flow

1. Program reads input from the user.
2. Initializes an empty string reversed_string.
3. Iterates through each character of user_input.
4. Prepends the current character to reversed_string.
5. Prints the final reversed string.

Time Complexity:

- $O(n^2)$ in languages where string concatenation is costly (like Python) because each char + reversed_string creates a new string.
- For small strings, this is negligible.

```

assignment-1.py
12     return input_string[::-1]
13
14     input_str = input("Please enter a string: ")
15     print("Reversed string:", reverse_string(input_str))"""
16
17     #Loop-Based Approach
18     """reversed_string = ""
19     input_str = input("Please enter a string: ")
20     for char in input_str:
21         reversed_string = char + reversed_string
22     print("Reversed string:", reversed_string)"""
23
24     #Built-In / Slicing-Based String Reversal
25     input_str = input("Please enter a string: ")
26     reversed_string = ''.join(reversed(input_str))
27     print("Reversed string:", reversed_string)
28

```

The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows a folder named "AI ASSISTEDCODING" containing "assignment-1.py".
- EDITOR:** Displays the Python code for reversing a string, comparing a loop-based approach and a built-in slicing-based approach.
- TERMINAL:** Shows the command `/usr/local/bin/python3 "/Users/nithyadugayala/Desktop/ai assistedcoding/assignment-1.py"`, followed by the user input "Please enter a string: India", and the reversed output "Reversed string: aidnI".
- STATUS BAR:** Shows "Ln 26, Col 43" and "3.14.2".

Use Case:

- Good for learning purposes and step-by-step logic demonstration.
- Not ideal for very large strings

Approach 2: Built-In / Slicing-Based String Reversal

Execution Flow

1. Program reads input from the user.
2. Uses Python slicing `[::-1]` to reverse the string in a single step.
3. Prints the final reversed string.

Time Complexity:

- $O(n)$ – very efficient even for large strings.

Use Case:

- Best for production code or large-scale applications.
- Cleaner, shorter, and more maintainable than the loop-based approach.

Comparison Table

| Criteria | Loop-Based Approach | Slicing-Based Approach |
|----------------------------------|--|--------------------------------|
| Code Complexity | Longer, step-by-step | Short, single-line |
| Execution Flow | Iterative prepending of characters | Direct slicing operation |
| Time Complexity | $O(n^2)$ (due to repeated concatenation) | $O(n)$ |
| Performance (Large Input) | Slower, may cause overhead for large strings | Very fast, scales well |
| Readability | Moderate | High |
| Use Case | Learning, step-by-step explanation | Production, large applications |