

# AI-Assisted Coding

Name: A. DINESH

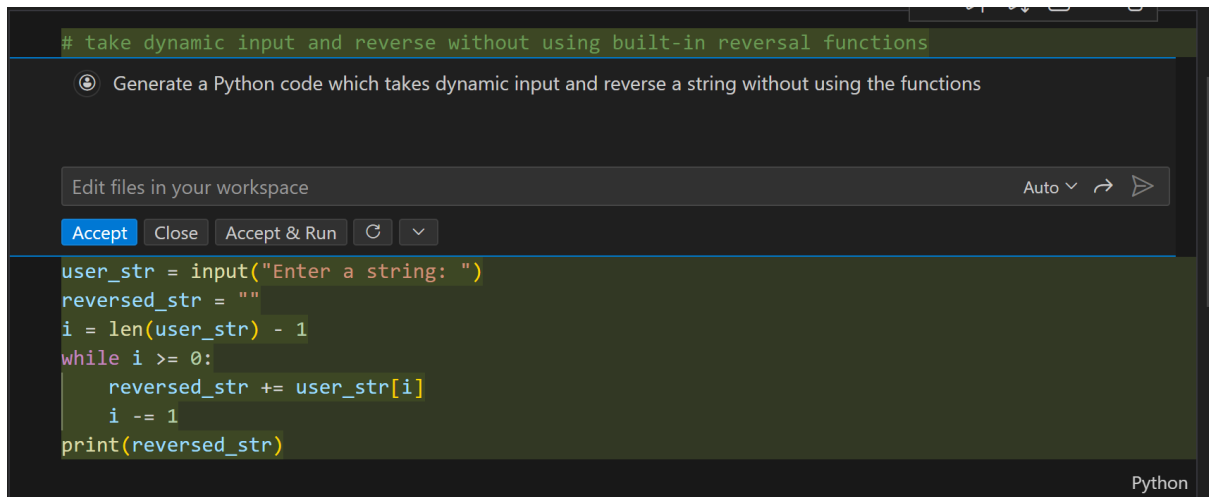
Roll Number: 2303A52329

Task: 1.5

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

Prompt: Generate a python code which take dynamic input and reverse without using built-in reversal functions

Screen Shot:



The screenshot shows an AI-assisted coding interface. At the top, a prompt is entered: "Generate a Python code which takes dynamic input and reverse a string without using the functions". Below the prompt, the generated Python code is displayed in a dark-themed editor. The code uses a while loop to iterate over the string from the end to the beginning, building the reversed string character by character. The interface includes buttons for "Accept", "Close", and "Accept & Run".

```
# take dynamic input and reverse without using built-in reversal functions

Generate a Python code which takes dynamic input and reverse a string without using the functions

Edit files in your workspace Auto ↩ ➡

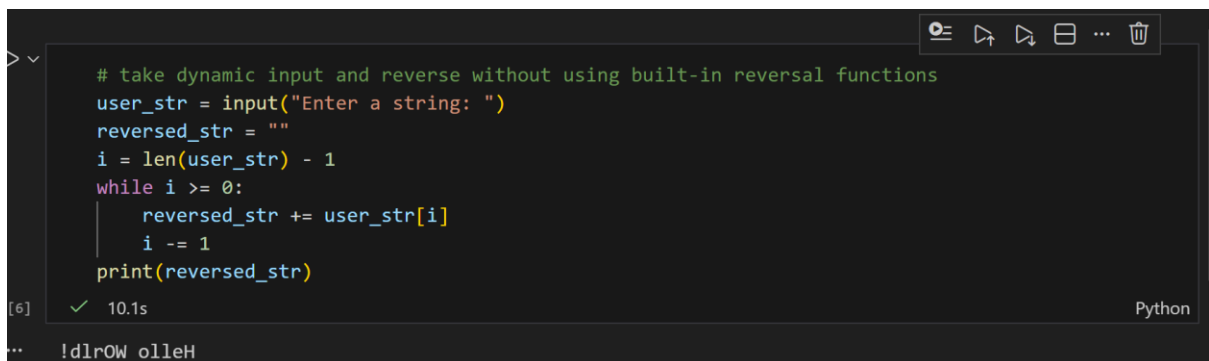
Accept Close Accept & Run ↻ ⌵

user_str = input("Enter a string: ")
reversed_str = ""
i = len(user_str) - 1
while i >= 0:
    reversed_str += user_str[i]
    i -= 1
print(reversed_str)
```

Python

Inputs: Hello World!

Output:



The screenshot shows a Python terminal window. The same Python code from the previous screenshot is pasted into the terminal. The output of the program is displayed at the bottom: "dlrOW olleH", which is the reverse of "Hello World!". The terminal shows a successful execution with a green checkmark and a timing of 10.1s.

```
# take dynamic input and reverse without using built-in reversal functions
user_str = input("Enter a string: ")
reversed_str = ""
i = len(user_str) - 1
while i >= 0:
    reversed_str += user_str[i]
    i -= 1
print(reversed_str)
```

[6] ✓ 10.1s Python

... !dlrOW olleH

Justification:

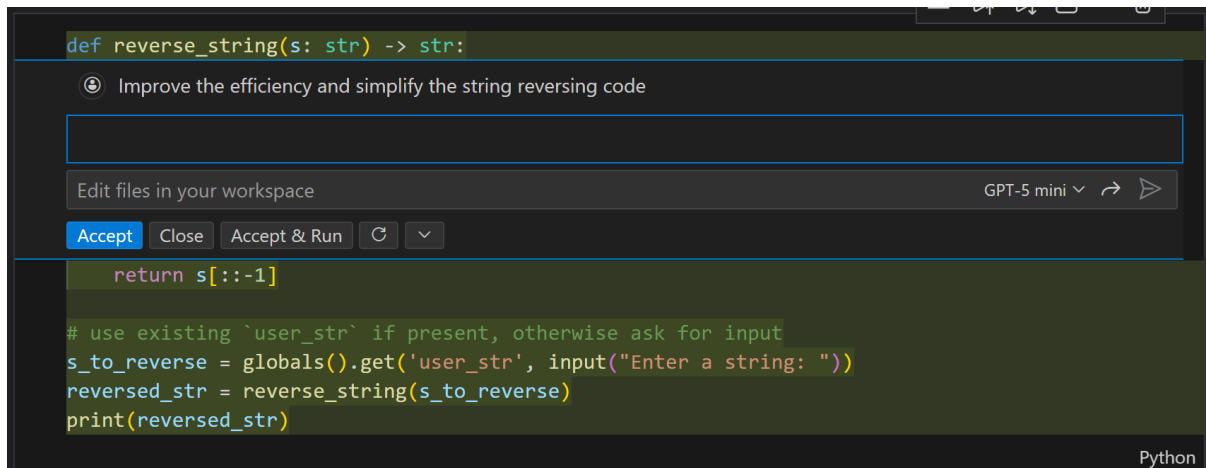
This program reverses a string by **implementing the logic directly in the main code block**, without using any user-defined functions, which satisfies the requirement of **no modularization**. Accepting input from the user makes the utility interactive and suitable for a basic messaging application scenario. Writing the logic inline keeps the program **simple**,

transparent, and easy to trace, which is ideal for beginners and for understanding how string manipulation works at a fundamental level.

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

Prompt: Improve the efficiency and simplify the string reversing code

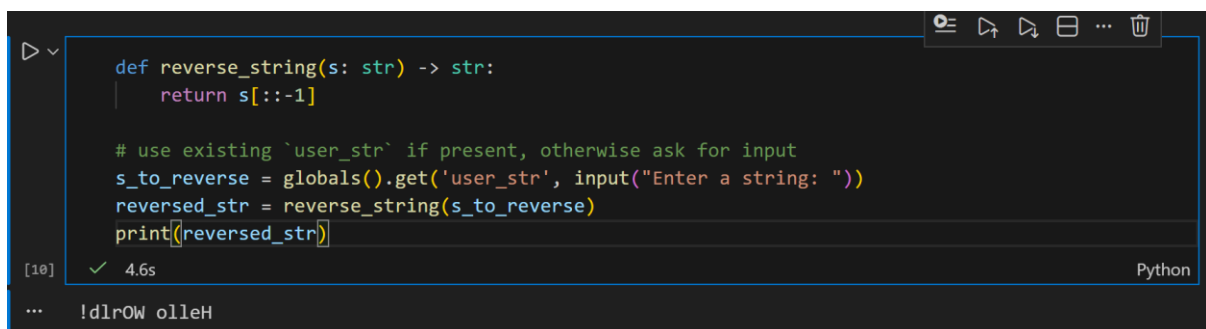
Screen Shot:



The screenshot shows a code editor with a dark theme. At the top, a function is defined: `def reverse_string(s: str) -> str:`. Below this, a prompt is displayed: "Improve the efficiency and simplify the string reversing code". A text input field is empty. Below the input field, there is a toolbar with buttons: "Accept", "Close", "Accept & Run", a refresh icon, and a dropdown arrow. To the right of the toolbar, it says "Edit files in your workspace" and "GPT-5 mini". Below the toolbar, the code `return s[::-1]` is visible. Further down, there is a comment: `# use existing 'user_str' if present, otherwise ask for input`, followed by `s_to_reverse = globals().get('user_str', input("Enter a string: "))`, `reversed_str = reverse_string(s_to_reverse)`, and `print(reversed_str)`. The editor is labeled "Python" in the bottom right corner.

Input: Hello World!

Output:



The screenshot shows the same code editor as before, but now the code is being executed. The code is: `def reverse_string(s: str) -> str:`, `return s[::-1]`, `# use existing 'user_str' if present, otherwise ask for input`, `s_to_reverse = globals().get('user_str', input("Enter a string: "))`, `reversed_str = reverse_string(s_to_reverse)`, and `print(reversed_str)`. The editor is labeled "Python" in the bottom right corner. Below the code, there is a status bar showing "[10] ✓ 4.6s". At the bottom, the output is displayed: "... !dlrOW olleH".

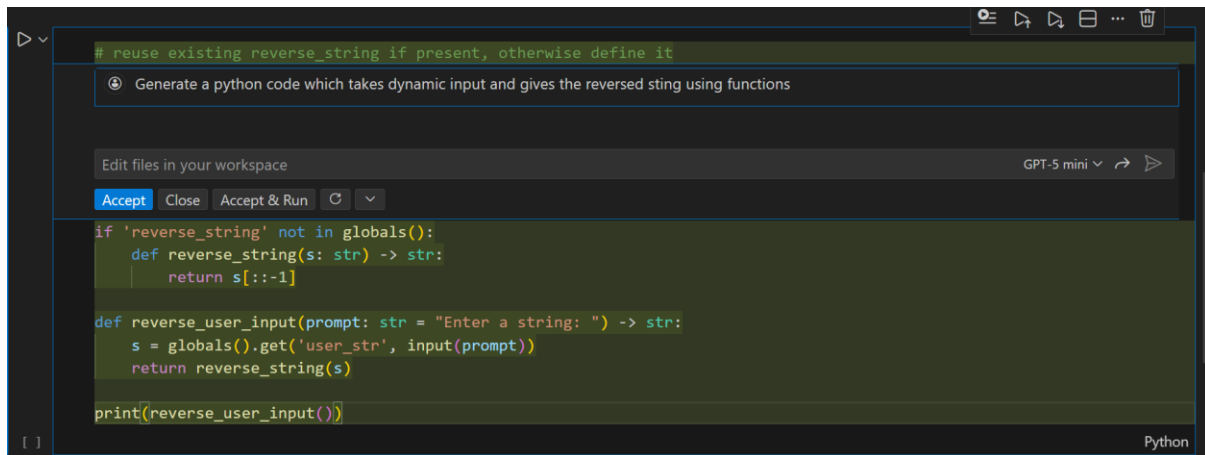
Justification:

The optimized code improves readability and maintainability by removing unnecessary variables and avoiding complex index-based logic. Iterating directly over the string makes the reversal process easier to understand during code review. The simplified structure reduces cognitive load for other developers while still fulfilling the requirement of implementing the logic inline without using user-defined functions.

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

Prompt: Generate a python code which takes dynamic input and gives the reversed sting using functions

Screen Shot:



```
# reuse existing reverse_string if present, otherwise define it
Generate a python code which takes dynamic input and gives the reversed sting using functions

Edit files in your workspace GPT-5 mini → ▶

Accept Close Accept & Run ↻ ▼

if 'reverse_string' not in globals():
    def reverse_string(s: str) -> str:
        return s[::-1]


def reverse_user_input(prompt: str = "Enter a string: ") -> str:
    s = globals().get('user_str', input(prompt))
    return reverse_string(s)

print(reverse_user_input())

Python
```

Input: How are you

Output:



```
# reuse existing reverse_string if present, otherwise define it
if 'reverse_string' not in globals():
    def reverse_string(s: str) -> str:
        return s[::-1]

def reverse_user_input(prompt: str = "Enter a string: ") -> str:
    s = globals().get('user_str', input(prompt))
    return reverse_string(s)

# force fresh input, do not reuse existing 'user_str'
print(reverse_string(input("Enter a string: ")))

[13] ✓ 72s Python

... uoy era woH
```

Justification:

Using a **user-defined function** for string reversal improves **modularity and reusability**, allowing the same logic to be used in multiple parts of the application without code duplication. Encapsulating the logic inside a function makes the program **easier to maintain, test, and debug**, especially as the application grows. Meaningful, AI-assisted comments enhance **code readability** and help other developers quickly understand the purpose and flow of the logic during collaboration or review.

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

Prompt: Compare the task 1 and task 2 and give a analytical report based on:

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

Output Screen Shot:

```
.. Comparison: Cell 1 (original) vs Cell 3 (improved)

Code clarity:
- Cell 1: concise but mixes logic with I/O and uses globals, reducing clarity.
- Cell 3: explicit, documented pure function; intent is clear.

Reusability:
- Cell 1: limited (reads input and depends on global state).
- Cell 3: high (no side effects, can be reused in other modules/tests).

Debugging ease:
- Cell 1: harder (implicit globals and I/O increase surface for bugs).
- Cell 3: easier (deterministic, unit-test friendly).

Suitability for large-scale applications:
- Cell 1: poor (global dependencies, runtime input hinder modularity).
- Cell 3: suitable (modular, testable, integrates into larger codebases).
```

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

Prompt: Generate a python code to Generate Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to String Reversal)

Output Screenshot :

```
Computing Fibonacci(n=31)

iterative: value=1346269 time=0.000008s
memoized recursion: value=1346269 time=0.000102s
naive recursion: skipped for n>30 (would be very slow)

Reversing string: 'Hello, world!'

iterative reverse: '!dlrow ,olleH' time=0.000008s
recursive reverse: '!dlrow ,olleH' time=0.000010s

Consistency checks: fib_iter==fib_memo -> True, rev_iter==rev_rec -> True
```