AI-Assisted Coding

Name: G. Snehith
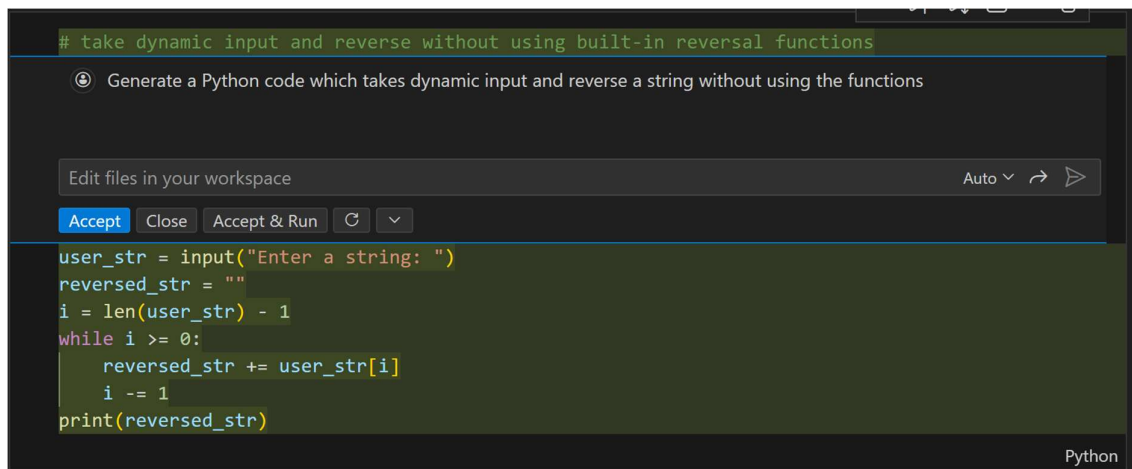
Roll number: 2303A52324

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
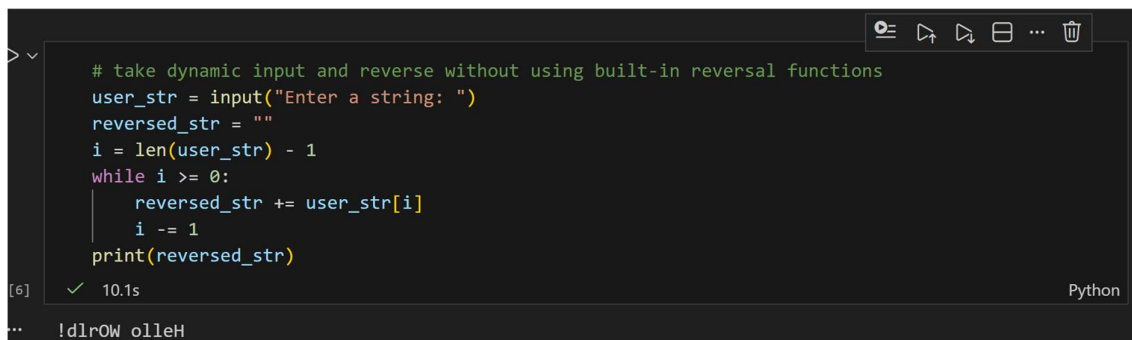
Screen Shot:

Inputs: Hello World!
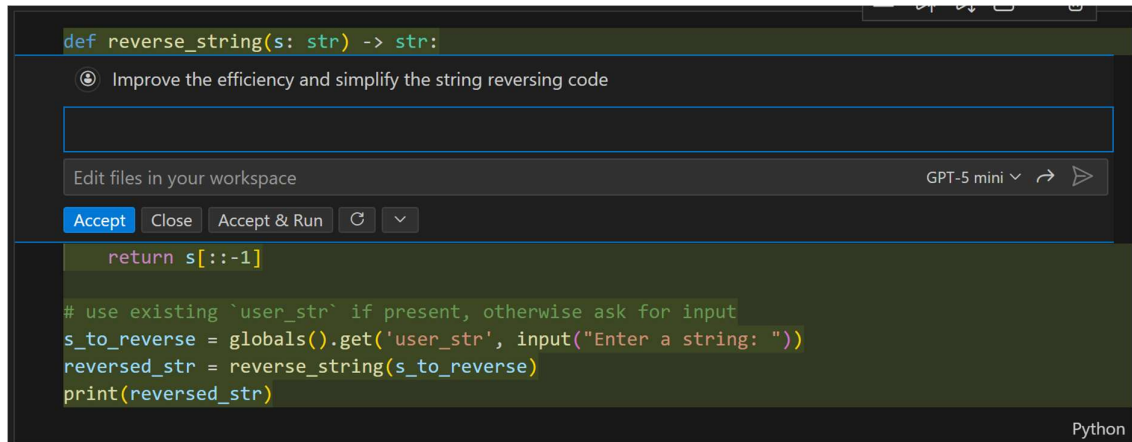
Output:

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)

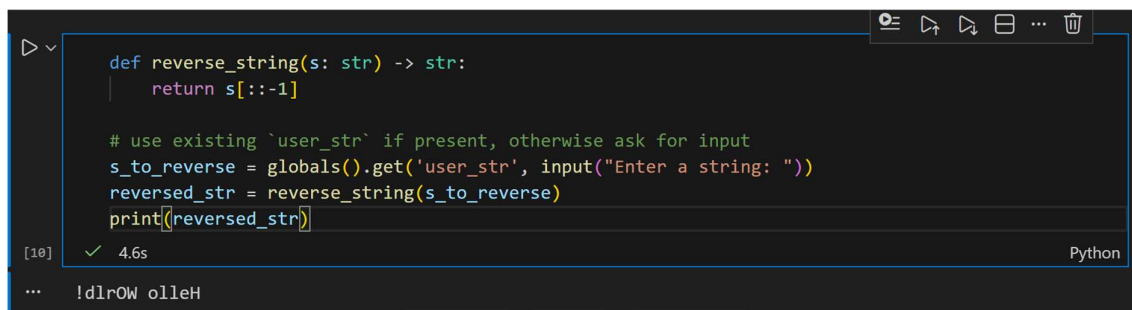Prompt: Improve the efficiency and simplify the string reversing code

Screen Shot:

```
def reverse_string(s: str) -> str:

    ⊕  Improve the efficiency and simplify the string reversing code

    Edit files in your workspace                                    GPT-5 mini ∨  ↗  ▷

    Accept   Close   Accept & Run   ⟳   ∨

        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)
    print(reversed_str)
                                                                      Python
```

Input: Hello World!

Output:

```
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)
print(reversed_str)
```
[10] ✓ 4.6s   Python
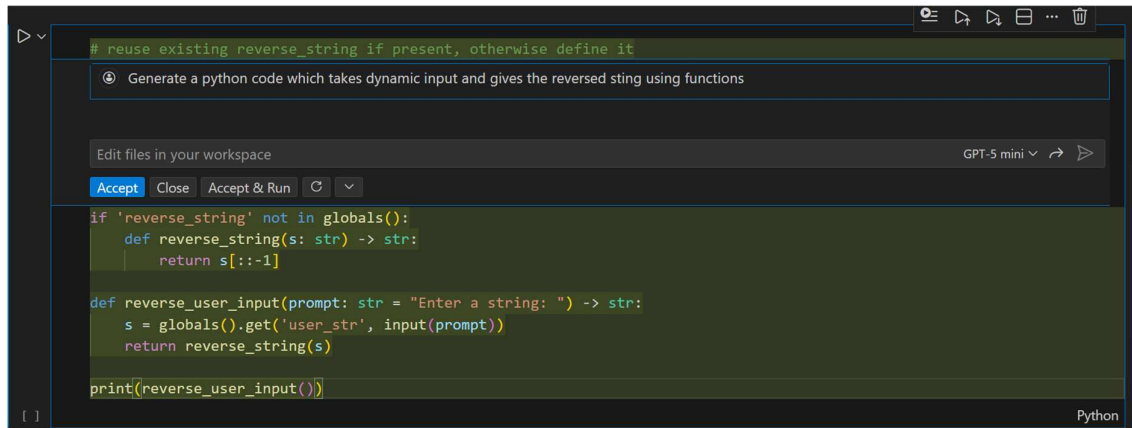···   !dlrOW olleH

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:

Input: How are you

Output:

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
```