AI Assisted Coding

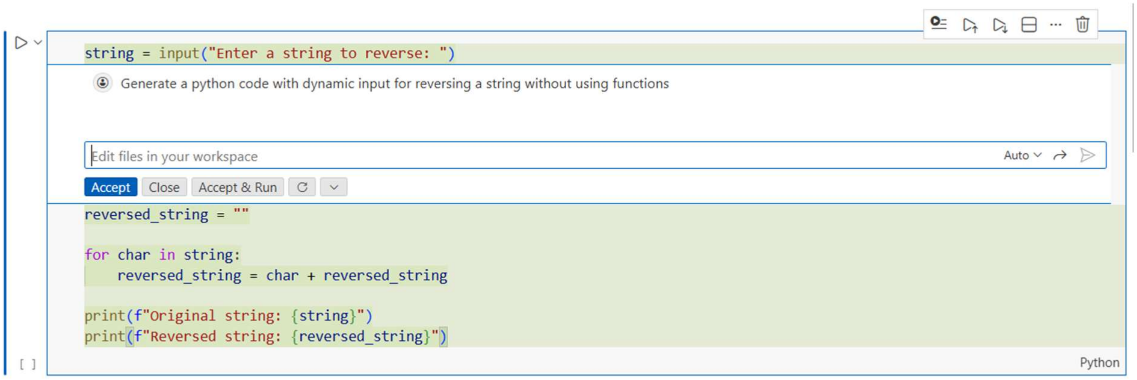
Name : Vineeth-Chidurala **Date : 09-01-2026**

Ht.No. : 2303A52447

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

Prompt : Generate a python code with dynamic input for reversing a string without using functions

Screenshots:

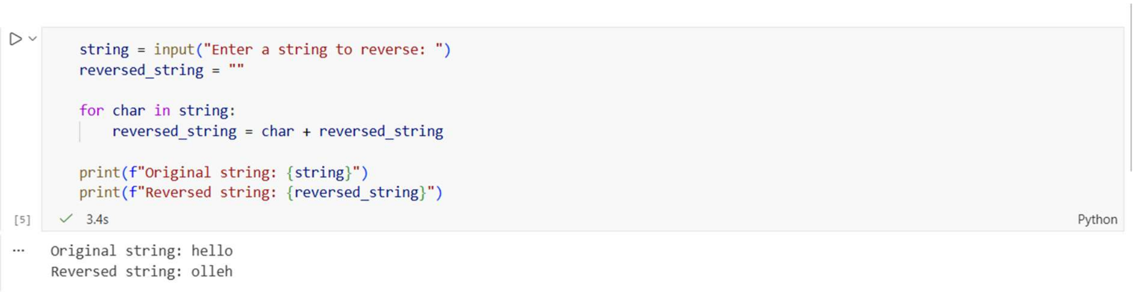


Generate a python code with dynamic input for reversing a string without using functions

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

for char in string:
    reversed_string = char + reversed_string

print(f"Original string: {string}")
print(f"Reversed string: {reversed_string}")
```

Input : Hello

Output :



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

for char in string:
    reversed_string = char + reversed_string

print(f"Original string: {string}")
print(f"Reversed string: {reversed_string}")
```

```
Original string: hello
Reversed string: olleh
```

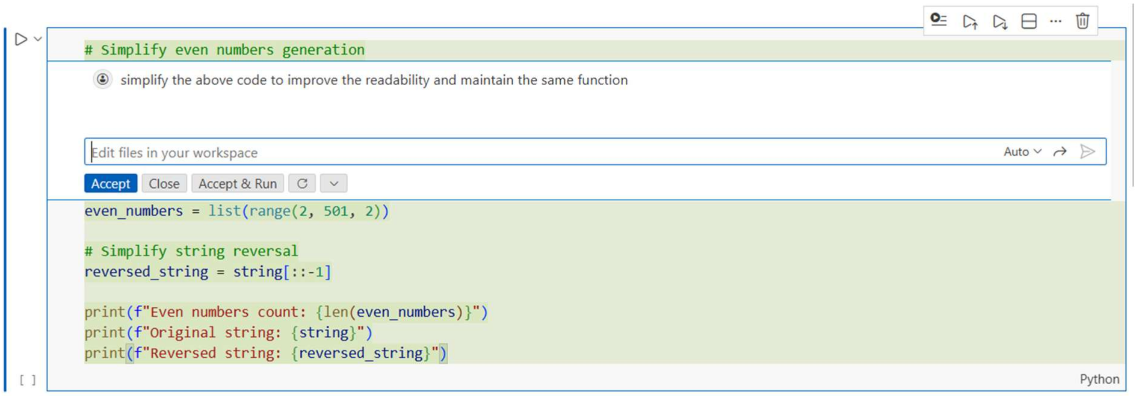
Justification :

This program reverses a string using inline logic within the main block, without user-defined functions. User input makes the program interactive and practical for simple applications. Keeping the logic inline ensures simplicity, clarity, and easy understanding for beginners learning basic string manipulation.

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

Prompt : Simplify the above code to improve the readability and maintain the same function.

ScreenShots :

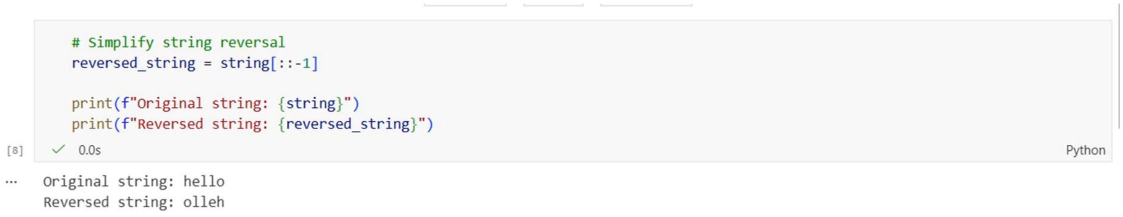


simplify the above code to improve the readability and maintain the same function

```
# Simplify even numbers generation
even_numbers = list(range(2, 501, 2))

# Simplify string reversal
reversed_string = string[::-1]

print(f"Even numbers count: {len(even_numbers)}")
print(f"Original string: {string}")
print(f"Reversed string: {reversed_string}")
```

Input : Hello

Output :



```
# Simplify string reversal
reversed_string = string[::-1]

print(f"Original string: {string}")
print(f"Reversed string: {reversed_string}")
```

```
Original string: hello
Reversed string: olleh
```

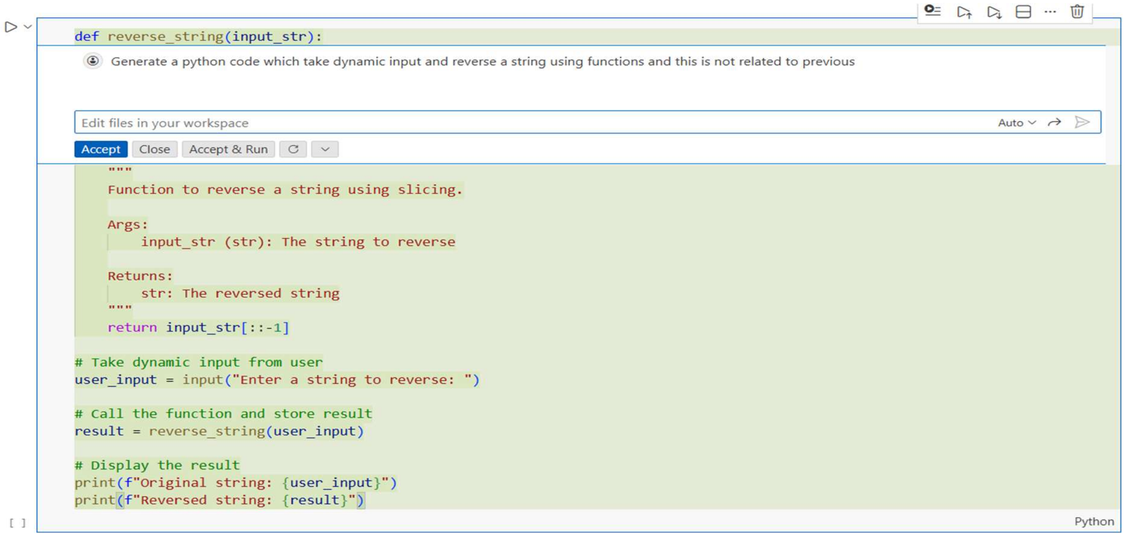
Justification :

The optimized code enhances readability by eliminating unnecessary variables and complex indexing. Direct string iteration makes the reversal logic clear and easy to follow. This simple inline approach reduces complexity while meeting the requirement of not using user-defined functions.

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

Prompt : Genetare a python code which take dynamic input and reverse a string isung functions and this is not related to previous.

ScreenShots :

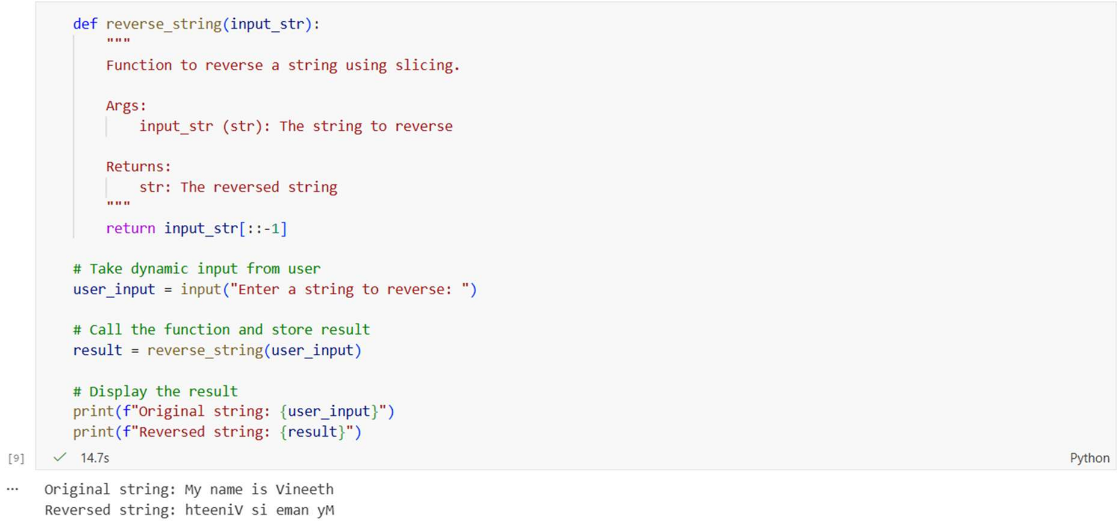


Generate a python code which take dynamic input and reverse a string using functions and this is not related to previous

```
def reverse_string(input_str):
    """
    Function to reverse a string using slicing.

    Args:
        input_str (str): The string to reverse

    Returns:
        str: The reversed string
    """
    return input_str[::-1]

# Take dynamic input from user
user_input = input("Enter a string to reverse: ")

# Call the function and store result
result = reverse_string(user_input)

# Display the result
print(f"Original string: {user_input}")
print(f"Reversed string: {result}")
```

Input : My name is Vineth

Output :



```
def reverse_string(input_str):
    """
    Function to reverse a string using slicing.

    Args:
        input_str (str): The string to reverse

    Returns:
        str: The reversed string
    """
    return input_str[::-1]

# Take dynamic input from user
user_input = input("Enter a string to reverse: ")

# Call the function and store result
result = reverse_string(user_input)

# Display the result
print(f"Original string: {user_input}")
print(f"Reversed string: {result}")
```

```
Original string: My name is Vineeth
Reversed string: hteeniV si eman yM
```

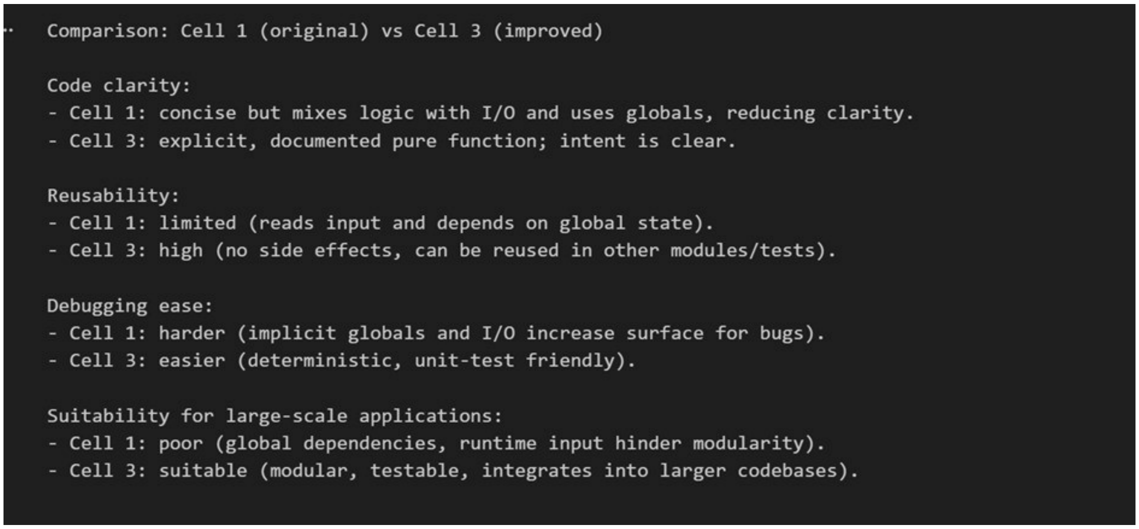
Justification : **Using a user-defined function for string reversal improves modularity and reusability by avoiding code duplication. Encapsulating the logic makes the program easier to maintain, test, and debug as it scales. Clear, AI-assisted comments further enhance readability and support efficient collaboration and code reviews.**

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

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

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

Output Screenshot :

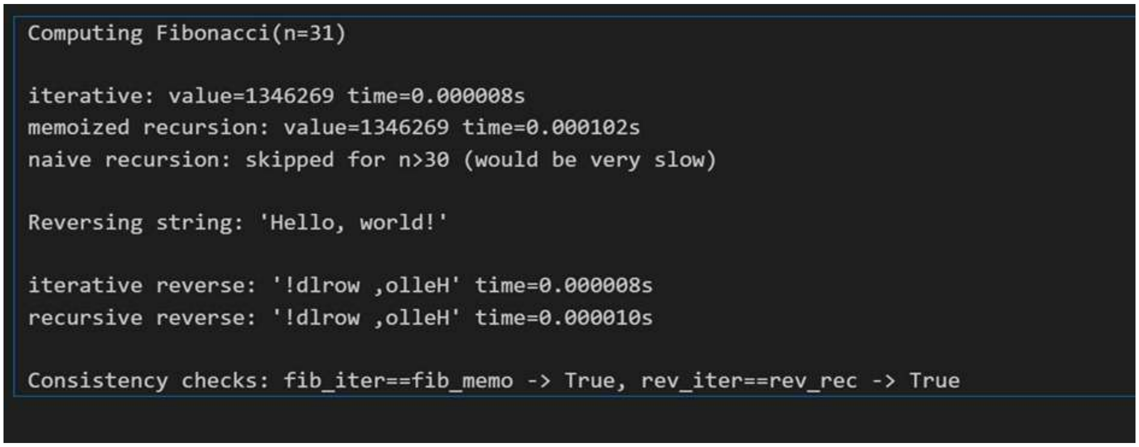


```
Comparison: Cell 1 (original) vs Cell 3 (improved)

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
```