

Task 1:

Prompt: write a python program to reverse a string without functions

Code: def reverse_string(s):

```
reversed_str = ""  
for i in range(len(s) - 1, -1, -1):  
    reversed_str += s[i]  
return reversed_str  
  
input_string = input("Enter a string: ")  
reversed_string = reverse_string(input_string)  
print("Reversed string:", reversed_string)
```

output: Enter a string: nikhhil

Reversed string: lihhkin

Task 2:

Prompt: #simplify this string reversal code and improve readability and efficiency

Code:

```
input_string = input("Enter a string: ")  
reversed_string = input_string[::-1]  
print("Reversed string:", reversed_string)
```

output: Enter a string: nikhhil

Reversed string: lihhkin

Task 3:

Prompt: Give string reversal code with using functions

Code:

```
def reverse_string(s: str) -> str:  
    return s[::-1]
```

```

if __name__ == "__main__":
    input_string = input("Enter a string: ")
    reversed_string = reverse_string(input_string)
    print("Reversed string:", reversed_string)

```

Task 4:

Prompt: Analyse the code with function and without function and give a comparsion table

#Analyse the code with function and without function and give a comparsion table		
# Aspect	Without Functions	With Functions
# Code Clarity	Less clear, harder to follow logic	More organized, easier to understand
# Reusability	Low, code cannot be reused easily can be reused in different contexts	High, functions
# Debugging Ease	Difficult to isolate issues	Easier to debug specific functions
# Suitability for Large-Scale Applications	Not suitable, hard to maintain	Highly suitable
# , modular structure aids maintenance		
....		

Task 5:

Prompt: Give different approaches to reverse a string like a loop based and built in or slicing based.

Code:

```

def reverse_string_loop(s: str) -> str:
    reversed_str = ""
    for char in s:
        reversed_str = char + reversed_str
    return reversed_str

# Slicing-based approach

def reverse_string_slicing(s: str) -> str:
    return s[::-1]

if __name__ == "__main__":
    input_string = input("Enter a string: ")
    print("Reversed using loop:", reverse_string_loop(input_string))
    print("Reversed using slicing:", reverse_string_slicing(input_string))

```

# Aspect	Loop-based Approach	slicing-based Approach
# -		
# Execution flow	Iterates through each character and builds the reversed string step-by-step.	Uses Python's built-in slicing mechanism to reverse the string in one operation
# -		
# Time complexity	$O(n^2)$ due to string concatenation in a loop	$O(n)$ as slicing is optimized
# Performance for large inputs	Slower for large strings due to repeated concatenation	Faster for large strings
# When each approach is appropriate	Useful when you want to understand the underlying logic or when implementing custom reversal logic without built-ins.	Preferred in production code where performance matters and readability is important
# -		