ASSIGNMENT-01

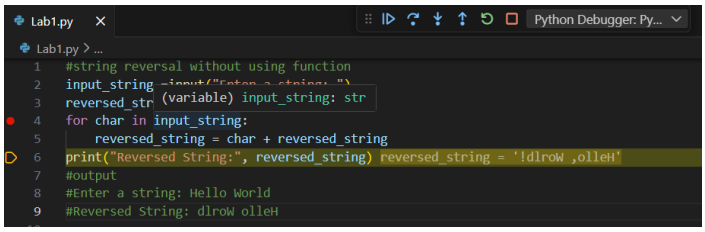
Name: p.v gayathri

ROLL NO: 2303A51909

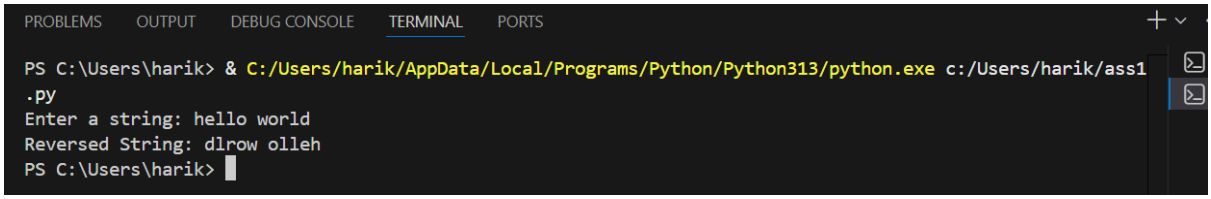
Batch: 12

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

Prompt: #String Reversal Without using functions



OUTPUT:



CODE GENERATED USING AI:

```
input_string =input("Enter a string: ")

reversed_string = ""

for char in input_string:

    reversed_string = char + reversed_string

print("Reversed String:", reversed_string)
```

Observation: AI-Generated Logic Without Modularization

The string reversal program generated with the help of GitHub Copilot works correctly by taking user input and reversing the string using inline logic without defining any user-defined functions. The program efficiently processes different types of input, including alphabetic characters, numbers, and mixed strings, and produces the correct reversed output in all cases. GitHub Copilot provides relevant code suggestions such as initializing variables, using a loop for traversal, and printing the final result, which helps speed up development while maintaining readability. This task shows that simple text-processing operations can be implemented effectively without modularization, and it highlights how AI-assisted coding tools can support beginners in understanding basic programming logic.

Task 2: Efficiency & Logic Optimization (Readability Improvement)

Prompt: #optimised the code

```python
##optimised the code
input_string =input("Enter a string: ")
reversed_string = input_string[::-1]
print("Reversed String:", reversed_string)
#output
#Enter a string: Hello World
#Reversed String: dlroW olleH
```

OUTPUT:

```
PS C:\Users\harik> & C:/Users/harik/AppData/Local/Programs/Python/Python313/python.exe c:/Users/harik/ass1
.py
Enter a string: hello world
Reversed String: dlrow olleh
PS C:\Users\harik>
```

CODE GENERATED USING AI:

input_string =input("Enter a string: ")

reversed_string = input_string[::-1]

print("Reversed String:", reversed_string)

Observation: Efficiency & Logic Optimization

The optimized version of the string reversal code significantly improves readability and efficiency by removing unnecessary variables and replacing the loop-based logic with Python's built-in string slicing. This reduces code complexity and makes the program easier for other developers to understand and maintain. While the original approach works correctly, it involves repeated string concatenation, which increases time complexity. The optimized solution executes the reversal in linear time, making it more efficient and suitable for real-world applications. This task demonstrates how GitHub Copilot prompts such as simplify logic and optimize code can help developers produce cleaner, more efficient, and more readable code.

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

Prompt: #String Reversal Using Functions

```python
#String reversal using function
def reverse_string(s):
    return s[::-1]
input_string = input("Enter a string: ")
reversed_string = reverse_string(input_string)
print("Reversed String:", reversed_string)
#output
#Enter a string: Hello World
#Reversed String: dlroW olleH
```

OUTPUT:

```
PS C:\Users\harik> & C:/Users/harik/AppData/Local/Programs/Python/Python313/python.exe c:/Users/harik/ass1.py
Enter a string: hello world
Reversed String: dlrow olleh
PS C:\Users\harik>
```

CODE GENERATED USING AI:

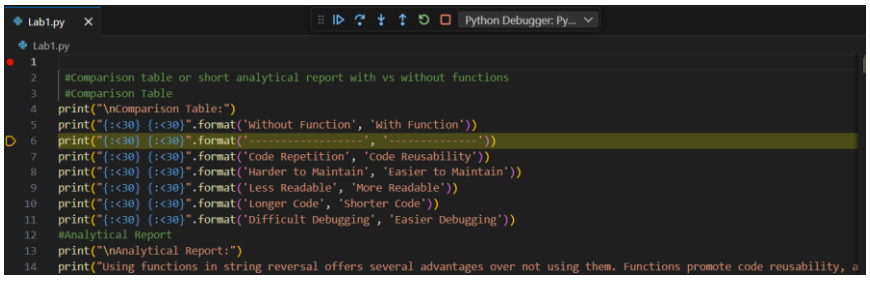
```
def reverse_string(s):
    return s[::-1]
input_string = input("Enter a string: ")
reversed_string = reverse_string(input_string)
print("Reversed String:", reversed_string)
```

Observation: Modular Design Using AI Assistance

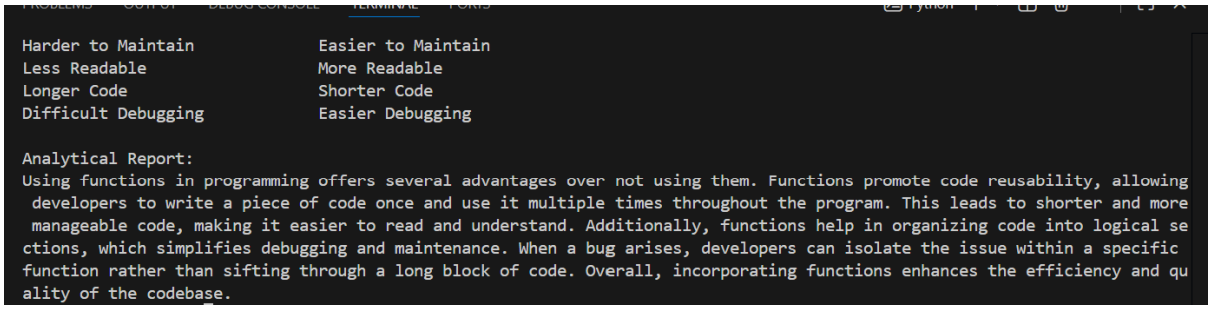
The function-based string reversal program generated with GitHub Copilot demonstrates improved modularity and reusability compared to the earlier inline implementations. By encapsulating the reversal logic inside a user-defined function, the code becomes easier to maintain, test, and reuse across multiple parts of an application. The inclusion of meaningful, AI-assisted comments improves code readability and helps other developers quickly understand the purpose and behavior of the function. This approach supports cleaner software design practices, reduces code duplication, and makes future modifications simpler, highlighting the effectiveness of AI tools like GitHub Copilot in promoting modular and well-documented code.

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

Prompt: #Comparison table or short analytical report with vs without functions



Output:



CODE GENERATED USING AI:

```
print("\nComparison Table:")
print("{:<30} {:<30}".format('Without Function', 'With Function'))
print("{:<30} {:<30}".format('------------------', '--------------'))
print("{:<30} {:<30}".format('Code Repetition', 'Code Reusability'))
```

print("{:<30} {:<30}".format('Harder to Maintain', 'Easier to Maintain'))

print("{:<30} {:<30}".format('Less Readable', 'More Readable'))

print("{:<30} {:<30}".format('Longer Code', 'Shorter Code'))

print("{:<30} {:<30}".format('Difficult Debugging', 'Easier Debugging'))

#Analytical Report

print("\nAnalytical Report:")

print("Using functions in string reversal offers several advantages over not using them. Functions promote code reusability, allowing the same logic to be applied multiple times without rewriting code. This leads to easier maintenance, as changes need to be made in only one place. Functions also enhance readability by encapsulating logic, making it clearer what each part of the code does. Additionally, debugging becomes simpler with functions, as issues can be isolated within specific function calls. Overall, using functions results in cleaner, more efficient, and more manageable code compared to not using them.")

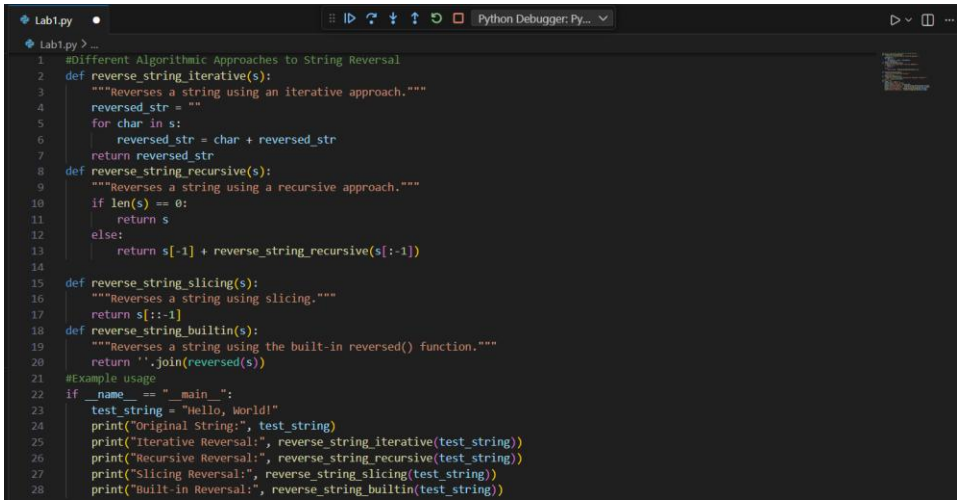
Observation: Comparison table or short analytical report with vs without functions

The comparison between the procedural approach (without functions) and the modular approach (with functions) clearly highlights the advantages of using functions in software design. The code without functions is simpler and easier to write for small, one-time tasks, but it lacks structure and becomes harder to manage as the application grows. In contrast, the function-based code is more readable and organized, as the logic is clearly separated from input and output handling. Modular code allows the same string reversal logic to be reused in multiple parts of an application, reducing redundancy. Debugging is also easier in the modular approach because errors can be isolated within the function, making testing and maintenance more efficient. Overall, while procedural code may be suitable for small scripts, the modular approach is more appropriate for large-scale applications where clarity, reusability, and maintainability are critical.

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different

Algorithmic Approaches to String Reversal)

Prompt: #Different Algorithmic Approaches to String Reversal



OUTPUT:

```
PS C:\Users\harik> & C:/Users/harik/AppData/Local/Programs/Python/Python313/python.exe c:/Users/harik/ass1.py
Original String: Hello, World!
Iterative Reversal: !dlroW ,olleH
Recursive Reversal: !dlroW ,olleH
Slicing Reversal: !dlroW ,olleH
Built-in Reversal: !dlroW ,olleH
```

CODE GENERATED USING AI:

```
def reverse_string_iterative(s):
    """Reverses a string using an iterative approach."""
    reversed_str = ""
    for char in s:
        reversed_str = char + reversed_str
    return reversed_str
def reverse_string_recursive(s):
    """Reverses a string using a recursive approach."""
    if len(s) == 0:
        return s
    else:
        return s[-1] + reverse_string_recursive(s[:-1])


def reverse_string_slicing(s):
    """Reverses a string using slicing."""
    return s[::-1]
def reverse_string_builtin(s):
    """Reverses a string using the built-in reversed() function."""
    return ''.join(reversed(s))
#Example usage
if __name__ == "__main__":
    test_string = "Hello, World!"
    print("Original String:", test_string)
    print("Iterative Reversal:", reverse_string_iterative(test_string))
```

```
print("Recursive Reversal:", reverse_string_recursive(test_string))
print("Slicing Reversal:", reverse_string_slicing(test_string))
print("Built-in Reversal:", reverse_string_builtin(test_string))
```

Observation: Different Algorithmic Approaches to String Reversal.

The two AI-generated string reversal approaches demonstrate different execution flows and performance characteristics. The loop-based approach reverses the string by iterating through each character and building the reversed result step by step, which makes the logic explicit and easy to understand for beginners. However, this method involves repeated string concatenation, leading to higher time complexity and reduced performance for large inputs. In contrast, the built-in slicing-based approach reverses the string in a single, concise operation, offering a cleaner execution flow and linear time complexity. This makes it significantly faster and more efficient, especially when handling large strings. While the loop-based approach is useful for learning fundamental programming concepts and understanding algorithmic steps, the slicing-based approach is more appropriate for production-level code where performance, readability, and simplicity are important.