

Assignment – 1.5

Name: U.VIGNESH

Batch:24

Ht No:2303A51964

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

❖ Scenario

You are developing a basic text-processing utility for a messaging application.

❖ Task Description

Use GitHub Copilot to generate a Python program that:

- Reverses a given string
- Accepts user input
- Implements the logic directly in the main code
- Does not use any user-defined functions
- ❖ Expected Output
- Correct reversed string
- Screenshots showing Copilot-generated code suggestions
- Sample inputs and outputs

The screenshot shows a code editor interface with several tabs at the top: "Welcome", "area of shapes.py 1", "derm5.py", and "#write a python program to reverse a str Untitled-1". The code in the active tab is:

```
1 #write a python program to reverse a string without functions
2 string = "Hello, World!"
3 reversed_string = ""
4 for char in string:
5     reversed_string = char + reversed_string
6 print("Reversed string:", reversed_string)
7 print("Original string:", string)
```

Below the code, there is a toolbar with tabs: PROBLEMS (1), OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, Filter (e.g. text, !excludeText...), and Code. The "OUTPUT" tab is selected, displaying the following execution results:

```
Original String: Hello, World!
Reversed String: !dlrow ,olleH

[Done] exited with code=0 in 0.532 seconds

[Running] python -u "C:\Users\souky\AppData\Local\Temp\tempCodeRunnerFile.python"
Reversed string: !dlrow ,olleH
Original string: Hello, World!

[Done] exited with code=0 in 0.337 seconds
```

Task 2: Efficiency & Logic Optimization (Readability Improvement)

❖ Scenario

The code will be reviewed by other developers.

❖ Task Description

Examine the Copilot-generated code from Task 1 and improve it by:

- Removing unnecessary variables
- Simplifying loop or indexing logic
- Improving readability
- Use Copilot prompts like:
 - “Simplify this string reversal code”
 - “Improve readability and efficiency”

Hint:

Prompt Copilot with phrases like

“optimize this code”, “simplify logic”, or “make it more readable”

❖ Expected Output

- Original and optimized code versions
- Explanation of how the improvements reduce time complexity

```
#simplify this string reversal code and improve readability and efficiency
string = "Hello, World!"
reversed_string = ""
for char in string:
    reversed_string = char + reversed_string
print("Reversed string:", reversed_string)
print("Original string:", string)
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, !excludeText...) Code

Reversed string: !dlroW ,olleH
Original string: Hello, World!

[Done] exited with code=0 in 0.337 seconds

[Running] python -u "C:\Users\souky\AppData\Local\Temp\tempCodeRunnerFile.python"
Reversed string: !dlroW ,olleH
Original string: Hello, World!

[Done] exited with code=0 in 0.433 seconds

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

❖ Scenario

The string reversal logic is needed in multiple parts of an application.

❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

- Uses a user-defined function to reverse a string
- Returns the reversed string
- Includes meaningful comments (AI-assisted)

❖ Expected Output

- Correct function-based implementation
- Screenshots documenting Copilot's function generation
- Sample test cases and outputs

```
... reverse a str Untitled-1 ● #simplify this string reversal code and Untitled-2 ● # Give string reversal code with using functions
1  # Give string reversal code with using functions
2  def reverse_string(s):
3      reversed_s = ""
4      for i in range(len(s)-1, -1, -1):
5          reversed_s += s[i]
6      return reversed_s
7  if __name__ == "__main__":
8      original_string = "Hello, World!"
9      reversed_string = reverse_string(original_string)
10     print("Reversed string:", reversed_string)
11     print("Original string:", original_string)
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, !excludeText...) Code

```
Original string: Hello, World!
Reversed string: !dlrow ,olleH

[Done] exited with code=0 in 0.38 seconds

[Running] python -u "C:\Users\souky\AppData\Local\Temp\tempCodeRunnerFile.python"
Reversed string: !dlrow ,olleH
Original string: Hello, World!
```

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

❖ Scenario

You are asked to justify design choices during a code review.

❖ Task Description

Compare the Copilot-generated programs:

- Without functions (Task 1)
- With functions (Task 3)

Analyze them based on:

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

❖ Expected Output

Comparison table or short analytical report

```

# Simplify this string reversal code and Untitled-2 ● # Give string reversal code with using f Untitled-3 ● # Analyze the code with functions and wi Untitled-4 9+ ●

1 # Analyze the code with functions and without functions and give a comparison table
2 # Aspects to consider:
3 # 1. Readability
4 # 2. Reusability
5 # 3. Maintainability
6 # 4. Efficiency
7 # 5. Testing
8 # 6. Debugging
9 # 7. Modularity
10 # 8. Scope Management
11 # 9. Performance
12 # 10. Collaboration
13 # 11. Documentation
14 # 12. Complexity Management
15 # 13. Encapsulation
16 # 14. Namespace Management
17 # 15. Code Organization
18 # Provide a detailed comparison in a table format
19 | Aspect | With Functions | Without Functions |
20 |-----|-----|-----|
21 | Readability | High - Functions provide clear structure | Low - Code can be cluttered and hard to follow |
22 | Reusability | High - Functions can be reused in different contexts | Low - Code duplication is common |
23 | Maintainability | High - Easier to update and modify specific parts | Low - Changes can affect the entire codebase |
24 | Efficiency | High - Functions can optimize performance | Low - May lead to redundant code |
25 | Testing | High - Functions can be tested individually | Low - Difficult to isolate and test specific parts |
26 | Debugging | High - Easier to locate and fix issues | Low - Hard to trace bugs in a monolithic codebase |
27 | Modularity | High - Code is organized into distinct units | Low - Code is often monolithic and intertwined |
28 | Scope Management | High - Variables have local scope within functions | Low - Global scope can lead to conflicts and unintended side effects |
29 | Performance | High - Functions can be optimized for performance | Low - May lead to inefficiencies due to code duplication |
30 | Collaboration | High - Functions allow multiple developers to work on different parts | Low - Difficult for multiple developers to coordinate changes |
31 | Documentation | High - Functions can be documented individually | Low - Documentation can be scattered and inconsistent |
32 | Complexity Management | High - Functions help break down complex problems | Low - Complex logic can become overwhelming |
33 | Encapsulation | High - Functions encapsulate behavior and data | Low - Data and behavior are often exposed globally |
34 | Namespace Management | High - Functions create local namespaces | Low - Global namespace can lead to naming conflicts |
35 | Code Organization | High - Functions promote organized code structure | Low - Code can become disorganized and hard to navigate |
36

```

```

# Simplify this string reversal code and Untitled-2 ● # Give string reversal code with using f Untitled-3 ● # Analyze the code with functions and wi Untitled-4 9+ ●

17 # 15. Code Organization
18 # Provide a detailed comparison in a table format
19 | Aspect | With Functions | Without Functions |
20 |-----|-----|-----|
21 | Readability | High - Functions provide clear structure | Low - Code can be cluttered and hard to follow |
22 | Reusability | High - Functions can be reused in different contexts | Low - Code duplication is common |
23 | Maintainability | High - Easier to update and modify specific parts | Low - Changes can affect the entire codebase |
24 | Efficiency | High - Functions can optimize performance | Low - May lead to redundant code |
25 | Testing | High - Functions can be tested individually | Low - Difficult to isolate and test specific parts |
26 | Debugging | High - Easier to locate and fix issues | Low - Hard to trace bugs in a monolithic codebase |
27 | Modularity | High - Code is organized into distinct units | Low - Code is often monolithic and intertwined |
28 | Scope Management | High - Variables have local scope within functions | Low - Global scope can lead to conflicts and unintended side effects |
29 | Performance | High - Functions can be optimized for performance | Low - May lead to inefficiencies due to code duplication |
30 | Collaboration | High - Functions allow multiple developers to work on different parts | Low - Difficult for multiple developers to coordinate changes |
31 | Documentation | High - Functions can be documented individually | Low - Documentation can be scattered and inconsistent |
32 | Complexity Management | High - Functions help break down complex problems | Low - Complex logic can become overwhelming |
33 | Encapsulation | High - Functions encapsulate behavior and data | Low - Data and behavior are often exposed globally |
34 | Namespace Management | High - Functions create local namespaces | Low - Global namespace can lead to naming conflicts |
35 | Code Organization | High - Functions promote organized code structure | Low - Code can become disorganized and hard to navigate |
36

```

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

❖ Scenario

Your mentor wants to evaluate how AI handles alternative logic paths.

❖ Task Description

Prompt GitHub Copilot to generate:

- A loop-based string reversal approach
- A built-in / slicing-based string reversal approach

❖ Expected Output

- Two correct implementations

➤ Comparison discussing:

- Execution flow
- Time complexity
- Performance for large inputs
- When each approach is appropriate

```

# Give string reversal code with using f Untitled-3
# Analyze the code with functions and wi Untitled-4 9+
# Give different approaches to reverse a string like a loop based and built in or slicing based.

1 # Give different approaches to reverse a string like a loop based and built in or slicing based.
2 def reverse_string_loop(s):
3     reversed_str = ""
4     for char in s:
5         reversed_str = char + reversed_str
6     return reversed_str
7 # Using slicing
8 def reverse_string_slicing(s):
9     return s[::-1]
10 if __name__ == "__main__":
11     test_string = "Hello, World!"
12     print("Original string:", test_string)
13     print("Reversed string (loop):", reverse_string_loop(test_string))
14     print("Reversed string (slicing):", reverse_string_slicing(test_string))

PROBLEMS 18 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, lexcludeText, t...
Original string: Hello, World!
[Done] exited with code=0 in 0.339 seconds

[Running] python -u "C:\Users\souky\AppData\Local\Temp\tempCodeRunnerFile.python"
Original string: Hello, World!
Reversed string (loop): !dlrow ,olleH
Reversed string (slicing): !dlrow ,olleH

[Done] exited with code=0 in 0.445 seconds

```

```

#simplify this string reversal code and Untitled-2
# Give string reversal code with using f Untitled-3
# Analyze the code with functions and wi Untitled-4 9+
# Give different approaches to reverse a string like a loop based and built in or slicing based.

1 # Give different approaches to reverse a string like a loop based and built in or slicing based.
2 def reverse_string_loop(s):
3     reversed_str = ""
4     for char in s:
5         reversed_str = char + reversed_str
6     return reversed_str
7 # Using slicing
8 def reverse_string_slicing(s):
9     return s[::-1]
10 if __name__ == "__main__":
11     test_string = "Hello, World!"
12     print("Original string:", test_string)
13     print("Reversed string (loop):", reverse_string_loop(test_string))
14     print("Reversed string (slicing):", reverse_string_slicing(test_string))
15 # comparison table of loop based approach vs slicing based approach
16 #Aspect | Loop Based Approach | Slicing Based Approach
17 #-----|-----|-----
18 # execution flow | Iterative, character by character | Direct slicing operation
19 # readability | Moderate, requires understanding of loop logic | High, concise and clear
20 # Time Complexity | O(n), where n is the length of the string | O(n), where n is the length of the string
21 # Performance for large inputs | Slower due to loop overhead | Faster due to optimized slicing
22 # When each approach is appropriate | When explicit control over the reversal process is needed | When simplicity and speed are prioritized
23
24

PROBLEMS 18 OUTPUT DEBUG CONSOLE TERMINAL PORTS Filter (e.g. text, lexcludeText, t...
Original string: Hello, World!
[Done] exited with code=0 in 0.339 seconds

[Running] python -u "C:\Users\souky\AppData\Local\Temp\tempCodeRunnerFile.python"
Original string: Hello, World!
Reversed string (loop): !dlrow ,olleH
Reversed string (slicing): !dlrow ,olleH

```

