Assignment-10.3

2303A52176
Y.Rishwanth Reddy
Batch:41

Task 1

Default Code:
Problem Statement 1: AI-Assisted Bug Detection
Scenario: A junior developer wrote the following Python function to
calculate factorials:

```
def factorial(n):
    result = 1
    for i in range(1, n):
        result = result * i
    return result
```

Error:

```
The Bug: The factorial function was
using range(1, n) which excludes n from
the calculation.

Example of the bug:

factorial(5) would return 24 (1×2×3×4)
instead of 120 (1×2×3×4×5)
The Fix: Changed range(1, n) to range(1,
n + 1) to include n in the
multiplication.
```

AI Code:

```
def factorial(n):
    result = 1
    for i in range(1, n + 1):
        result = result * i
    return result
```

Comparison:
The generated code gives the correct output for the number
Output for print(factorial(5)):

```
rishwanthyeruva@Rishwanths-MacBo
o ai assisted coding % /usr/loca
/python3 "/Users/rishwanthyeruva
top/ai assisted coding/10.3.py"
120
```

Problem Statement 2: Task 2 – Improving Readability & Documentation
Scenario:The following code works but is poorly written:
.

```
def calc(a, b, c):
    if c == "add":
        return a + b
    elif c == "sub":
        return a - b
    elif c == "mul":
        return a * b
    elif c == “div”:
```

```python
# TASK 2 PROMPT (One-Shot)
# Critique the function for readability and documentation,
# then rewrite it with meaningful names, docstring,
# input validation, and exception handling.

def calculate(a: float, b: float, operation: str) -> float:
    """
    Performs arithmetic operations.
    """
    if not isinstance(operation, str):
        raise TypeError("Operation must be a string")

    if operation == "add":
        return a + b
    elif operation == "sub":
        return a - b
    elif operation == "mul":
        return a * b
    elif operation == "div":
        if b == 0:
            raise ZeroDivisionError("Division by zero not allowed")
        return a / b
    else:
        raise ValueError("Invalid operation")
```

Output

```
--------------------------------------------------
TASK 2 OUTPUT:
Issues Identified : Poor naming, no documentation, no error handling
Fix Applied       : Added descriptive names, docstring, validation
Add Result        : 15
Multiply Result   : 50
--------------------------------------------------
```

Justification:

The original function had unclear variable names and no documentation.AI suggested better naming and added a proper docstring.Error handling and input validation were included.This made the function more readable and reliable.

Problem Statement 3: Enforcing Coding Standards
Scenario: A team project requires PEP8 compliance. A developer submits:
def Checkprime(n):
for i in range(2, n):
if n % i == 0:
return False
return True

PROMPT:
Identify PEP8 violations and refactor the code. while preserving functionality.

Code:
PROMPT:
Identify PEP8 violations and refactor the code. while preserving functionality.

```
# TASK 3 PROMPT (Zero-Shot)
# Identify PEP8 violations and refactor the code
# while preserving functionality.

def check_prime(n: int) -> bool:
    """
    Checks whether a number is prime.
    """
    if n <= 1:
        return False

    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

OUTPUT:

```
TASK 3 OUTPUT:
PEP8 Issues      : Function name, indentation, spacing
Fix Applied      : snake_case name, proper indentation
Is 7 Prime?     : True
Is 10 Prime?    : False
-------------------------------------------------
```

Justification: The original code violated PEP8 naming and indentation rules.
AI identified these style issues accurately.The function was refactored using snake_case and proper formatting.Functionality was preserved with improved code quality.

Problem Statement 4: AI as a Code Reviewer in Real Projects
Scenario:
In a GitHub project, a teammate submits:
def processData(d):

PROMPT: Review the function for readability, reusability. edge cases, and type safety. Refactor accordingly.
CODE:

```
# TASK 4 PROMPT (Few-Shot)
# Review the function for readability, reusability,
# edge cases, and type safety. Refactor accordingly.

from typing import List, Union

def double_even_numbers(
    numbers: List[Union[int, float]],
    multiplier: int = 2
) -> List[Union[int, float]]:
    """
    Doubles even numbers in a list.
    """
    if not isinstance(numbers, list):
        raise TypeError("Input must be a list")

    return [
        num * multiplier
        for num in numbers
        if isinstance(num, (int, float)) and num % 2 == 0
    ]
```

```
TASK 4 OUTPUT:
Issues Identified : Poor naming, no validation, no type hints
TASK 4 OUTPUT:
Issues Identified : Poor naming, no validation, no type hints
Issues Identified : Poor naming, no validation, no type hints
Fix Applied       : Clear name, type hints, validation, reusability
Processed List  : [4, 8, 12]
------------------------------------------------
```

Justification:
The original function lacked clarity and input validation.AI recommended meaningful names and type hints.Validation and reusability were added.This improved robustness and real-world usability.

Problem Statement 5: AI-Assisted Performance Optimization
Scenario: You are given a function that processes a list of integers, but it runs slowly on large datasets:

def sum_of_squares(numbers):
total = 0
for num in numbers:
total += num ** 2
return total

PROMPT: Analyze the time complexity and optimize the function.using Pythonic constructs.

CODE:

```
# TASK 5 PROMPT (Zero-Shot)
# Analyze the time complexity and optimize the function
# using Pythonic constructs.
def sum_of_squares_optimized(numbers) -> int:
    """
    Returns sum of squares using optimized generator expression.
    """
    return sum(x * x for x in numbers)
```

OUTPUT:

```
TASK 5 OUTPUT:
Performance Issue: Loop-based accumulation (slower)
Fix Applied      : Used generator expression with sum()
Optimized Result : 285
--------------------------------------------------
```

Justification:
The original function used a manual loop which was slower.AI analyzed the time complexity and suggested optimization. A generator expression with sum() was used. This improved performance while keeping the code readable.