AI ASSISTED CODING

Lab 13: Code Refactoring: Improving Legacy Code with AI Suggestions

Roll no: 2503A51L42
Name: YASHWANTH
Batch: 25BTCAICSB20
Assignment number: 13.2

Task 1: Remove Repetition

Provide AI with the following redundant code and ask it to refactor

Prompt:

Refactor the given Python function calculate_area that repeats logic into a cleaner, modular version using either dictionary-based dispatch or separate functions for rectangle, square, and circle.

Code provided:

```
def calculate_area(shape, x, y=0):
    if shape == "rectangle":
        return x * y
    elif shape == "square":
        return x * x
    elif shape == "circle":
        return 3.14 * x * x
```

Code:

task1.py

assignment 13.2 > task1.py > calculate_area

```python
import math

def area_rectangle(length, width):
    return length * width

def area_square(side):
    return side * side

def area_circle(radius):
    return math.pi * radius * radius

def calculate_area(shape, **kwargs):
    dispatch = {
        'rectangle': lambda: area_rectangle(kwargs.get('length'), kwargs.get('width')),
        'square': lambda: area_square(kwargs.get('side')),
        'circle': lambda: area_circle(kwargs.get('radius')),
    }
    if shape not in dispatch:
        raise ValueError(f"Unknown shape: {shape}")
    return dispatch[shape]()

# Example usage:
print("Rectangle area:", calculate_area('rectangle', length=5, width=3))
print("Square area:", calculate_area('square', side=4))
print("Circle area:", calculate_area('circle', radius=2))

```

Output:

```
Rectangle area: 15
Square area: 16
Circle area: 12.566370614359172
```

Observation:

- The refactored calculate_area function uses dictionary-based dispatch and separate functions for each shape.
- This modular approach eliminates repeated logic, improves readability, and makes it easy to add new shapes or modify existing ones.

Task 2: Error Handling in Legacy Code

Legacy function without proper error handling

Prompt:

Refactore the given python function with open() and try-except with an example

Code provided:

```
class Student:
    def __init__(self, n, a, m1, m2, m3):
        self.n = n
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3
```

Code:

assignment 13.2 > task2.py > e

```python
class Student:
    def __init__(self, n, a, m1, m2, m3):
        self.n = n
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3

# Example: Write and read student details using open() and try-except
student = Student("Alice", 20, 85, 90, 88)
filename = "student_details.txt"

# Write details to file
try:
    with open(filename, "w") as f:
        f.write(f"Name: {student.n}\n")
        f.write(f"Age: {student.a}\n")
        f.write(f"Total Marks: {student.total()}\n")
    print(f"Details written to {filename}")
except Exception as e:
    print(f"Error writing to file: {e}")

# Read details from file
try:
    with open(filename, "r") as f:
        content = f.read()
    print("\nRead from file:")
    print(content)
except Exception as e:
    print(f"Error reading from file: {e}")

```

Output:

```
Details written to student_details.txt

Read from file:
Name: Alice
Age: 20
Total Marks: 263
```

Observation:

The code demonstrates safe file operations using open() and try-except blocks.

AI has successfully writes and reads student details, handling any file errors gracefully.

This approach ensures robust file handling and prevents program crashes due to file-related issues.

Task 3: Complex Refactoring

Provide this legacy class to AI for readability and modularity improvements:

Prompt:

Refactor the legacy Student class by improving variable naming (name, age, marks), adding docstrings, enhancing print readability, and modularizing marks handling (e.g., storing in a list and using sum).

Code provided:

```
class Student:
    def __init__(self, n, a, m1, m2, m3):
        self.n = n
        self.a = a
        self.m1 = m1
        self.m2 = m2
        self.m3 = m3
    def details(self):
        print("Name:", self.n, "Age:", self.a)
    def total(self):
        return self.m1+self.m2+self.m3
```

Code:

assignment 13.2 > task3.py > ...

```python
class Student:
    """
    Represents a student with name, age, and a list of marks.
    """
    def __init__(self, name, age, marks):
        """
        Initialize a Student object.
        Args:
            name (str): The name of the student.
            age (int): The age of the student.
            marks (list of int/float): List of marks for the student.
        """
        self.name = name
        self.age = age
        self.marks = marks

    def details(self):
        """
        Print the student's details in a readable format.
        """
        print(f"Student Details:\n  Name: {self.name}\n  Age: {self.age}")

    def total_marks(self):
        """
        Calculate and return the total marks.
        Returns:
            int/float: The sum of all marks.
        """
        return sum(self.marks)

# Example usage
if __name__ == "__main__":
    student = Student(name="Aarav Sharma", age=17, marks=[85, 90, 88])
    student.details()
    print(f"Total Marks: {student.total_marks()}")
```

Output:

```
Student Details:
  Name: Aarav Sharma
  Age: 17
Total Marks: 263
```

Observation:

AI made the program more readable and easy to understand and it added the doc string for more readability and it add sum fuction to calculate the total marks.

Task 4: Inefficient Loop Refactoring

Refactor this inefficient loop with AI help

Prompt:

Code provided:

nums = [1,2,3,4,5,6,7,8,9,10]

squares = []

for i in nums:

    squares.append(i * i)

Code:

assignment 13.2 > task4.py > ...

```python
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Using list comprehension to get squares
squares = [i * i for i in nums]

# Example: print squares of even numbers using list comprehension
even_squares = [i * i for i in nums if i % 2 == 0]
print("All squares:", squares)
print("Squares of even numbers:", even_squares)
```

Output:

```
All squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Squares of even numbers: [4, 16, 36, 64, 100]
```

Observation:

- AI suggested a list comprehension
- Ai added the comment line for better understanding
- It made the inefficient loop , efficient and working