ASSIGNMENT-13.3

Name : P.Reshmitha Reddy

Rollno : 2403A510A7

Batch : 05

Course : AI Assisted Coding

Task Description #1 – Remove Repetition
Task: Provide AI with the following redundant code and ask it to
refactor
Python Code
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
Expected Output
• Refactored version with dictionary-based dispatch or separate
functions.
• Cleaner and modular design.

Prompt : #redundant code and ask it to refactor.

Code :

```python
def area_rectangle(x, y):
    return x * y

def area_square(x):
    return x * x

def area_circle(x):
    return 3.14 * x * x

def calculate_area(shape, x, y=0):
    """
    Calculates the area of a given shape using dictionary-based dispatch.
    """
    dispatch = {
        "rectangle": lambda: area_rectangle(x, y),
        "square": lambda: area_square(x),
        "circle": lambda: area_circle(x)
    }
    if shape not in dispatch:
        raise ValueError(f"Unknown shape: {shape}")
    return dispatch[shape]()
if __name__ == "__main__":
    print(calculate_area("rectangle", 3, 4))
    print(calculate_area("square", 5))
    print(calculate_area("circle", 2))
```

Output :

```
PS C:\Users\Reshm\Desktop\AIAC> & C:/Users/Reshm/anaconda3/python.exe c:/Users/Reshm/Desktop/AIAC/13.3.py
12
25
12.56
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

- The code uses separate functions for each shape's area calculation, improving modularity and readability.
- The calculate_area function uses a dictionary-based dispatch to remove repetitive conditional statements.
- The script includes a main block with test cases for rectangle, square, and circle, which prints the correct area for each.

- The code is clean, easy to extend (add more shapes), and demonstrates good Python practices for avoiding repetition.

Task Description #2 – Error Handling in Legacy Code
Task: Legacy function without proper error handling
Python Code
def read_file(filename):
f = open(filename, "r")
data = f.read()

f.close()
return data
Expected Output:
AI refactors with with open() and try-except:

Prompt :# Legacy function without proper error handling
Python Code

Code :

```python
def read_file(filename):
    """
    Reads the contents of a file safely with error handling.
    """
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
    except IOError as e:
        print(f"IO error occurred: {e}")
    return None

# Example usage
if __name__ == "__main__":
    content = read_file("example.txt")  # Replace with your filename
    if content is not None:
        print(content)
```

Output :

```
PS C:\Users\Reshm\Desktop\AIAC> & C:/Users/Reshm/anaconda3/python.exe c:/Users/Reshm/Desktop/AIAC/13.3.py
Error: File 'example.txt' not found.
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

- The read_file function safely reads a file's contents and handles errors using try-except blocks.
- If the file does not exist, it prints a clear error message: Error: File 'example.txt' not found.If another I/O error occurs (like permission denied), it prints the specific error.
- The function returns None if an error occurs, preventing the program from crashing.
- The example usage demonstrates how to call the function and print the file's contents only if reading was successful.
- This approach improves code robustness and user feedback compared to legacy code without error handling.

Task Description #3 – Complex Refactoring

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

improvements:

Python Code

```
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
```

Expected Output:

- AI improves naming (name, age, marks).
- Adds docstrings.
- Improves print readability.
- Possibly uses sum(self.marks) if marks stored in a list.

Prompt : # Refactor the following legacy Python class for readability and modularity.

Code :

```python
class Student:
    # ...existing code...

    def details(self):
        """Prints the student's name and age in a readable format."""
        print(f"Name: {self.name}, Age: {self.age}")

    def total(self):
        """Returns the total marks obtained by the student."""
        return sum(self.marks)
    """
    Represents a student with name, age, and marks in three subjects.
    """

    def __init__(self, name: str, age: int, marks: list[int]):
        """
        Initializes a Student object.

        Args:
            name (str): The student's name.
            age (int): The student's age.
            marks (list[int]): List of marks in three subjects.
        """
        self.name = name
        self.age = age
        if len(marks) != 3:
            raise ValueError("Exactly three marks must be provided.")
```

Output :

```
PS C:\Users\Reshm\Desktop\AIAC> & C:/Users/Reshm/anaconda3/python.exe c:/Users/Reshm/Desktop/AIAC/13.3.py
Name: Alice, Age: 20
Total Marks: 253
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

- The Student class is intended to be modular and readable, with improved naming (name, age, marks) and docstrings.
- There is a duplication of the details and total methods—each is defined twice. This will cause the second definition to override the first, and the indentation of the second total method is incorrect (it is outside the class).
- The class docstring is misplaced; it should be at the top of the class, not between methods.

- The constructor (__init__) checks for exactly three marks, which is good for data integrity.
- The example usage at the bottom demonstrates object creation and method calls.

Task Description #4 – Inefficient Loop Refactoring
Task: Refactor this inefficient loop with AI help
Python Code
nums = [1,2,3,4,5,6,7,8,9,10]
squares = []
for i in nums:
squares.append(i * i)
Expected Output: AI suggested a list comprehension

Prompt : #Refactor this inefficient loop with AI help
Python Code

Code :

```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = [i * i for i in nums]
print(squares)
```

Output :

```
PS C:\Users\Reshm\Desktop\AIAC> & C:/Users/Reshm/anaconda3/python.exe c:/Users/Reshm/Desktop/AIAC/13.3.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\Reshm\Desktop\AIAC>
```

Observation :

- The code uses a list comprehension to efficiently generate a list of squares for numbers 1 through 10.

- This approach is more concise and readable compared to a traditional for-loop with append.