LAB ASSIGNMENT 13

NAME : M.TANVI

ROLL NO : 2403A510A4 BATCH 01

COURSE : ASSISTED CODING QUESTION

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.

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:

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:

```
    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.

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**

TASK 1

PROMPT

Here is some redundant Python code. Please refactor it to make it cleaner and more efficient. 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

TASK 2

PROMPT

Here’s an old Python function without proper error handling. Please improve it. def read_file(filename): f = open(filename, "r") data = f.read() f.close() return data

```python
def read_file_improved(filename):
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        print(f"Error: The file '{filename}' was not found.")
        return None
    except Exception as e:
        print(f"An error occurred while reading the file: {e}")
        return None

# Example usage with a non-existent file:
content = read_file_improved("non_existent_file.txt")
if content is None:
    print("Failed to read the file.")

# Example usage with a dummy file (you can create one to test)
# with open("my_dummy_file.txt", "w") as f:
#     f.write("This is a test file.")
# content = read_file_improved("my_dummy_file.txt")
# if content:
#     print("File content:")
#     print(content)
```

```
Error: The file 'non_existent_file.txt' was not found.
Failed to read the file.
```

TASK 3

PROMPT

Here’s a legacy Python class. Please improve its readability and make it more modular. 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

```python
class Student:
    def __init__(self, name, age, math_score, science_score, history_score):
        self.name = name
        self.age = age
        self.math_score = math_score
        self.science_score = science_score
        self.history_score = history_score

    def display_details(self):
        """Prints the student's name and age."""
        print(f"Name: {self.name}, Age: {self.age}")

    def calculate_total_score(self):
        """Calculates and returns the sum of the student's scores."""
        return self.math_score + self.science_score + self.history_score

# Example usage:
student1 = Student("Alice", 16, 90, 85, 92)
student1.display_details()
print(f"Total score: {student1.calculate_total_score()}")
```

```
Name: Alice, Age: 16
Total score: 267
```

TASK 4

PROMPT

Refactor this Python loop to make it more efficient. nums = [1,2,3,4,5,6,7,8,9,10] squares = [] for i in nums: squares.append(i * i)

Copy to Drive

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

# Refactored loop using a list comprehension
squares = [i * i for i in nums]

print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```