AI ASSISTED CODING

LAB-13.2

NAME:CH.RAMCHARAN

ENROLLNO:2403A52069

BATCH:04

TASK-01:

Remove Repetition.

PROMPT:

Refactor the following redundant 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
```

CODE:

```python
1   import math
2
3   def calculate_area_refactored_functional(shape: str, x: float, y: float = 0) -> float:
4       """
5       Calculates the area of a shape using a dictionary-based dispatch.
6
7       This approach is more scalable than a long if/elif chain.
8
9       Args:
10          shape (str): The name of the shape ("rectangle", "square", "circle").
11          x (float): The primary dimension (width, side, or radius).
12          y (float, optional): The secondary dimension (height). Defaults to 0.
13
14      Returns:
15          float: The calculated area of the shape.
16      """
17      # Map shape names to lambda functions that perform the calculation
18      area_calculators = {
19          "rectangle": lambda w, h: w * h,
20          "square":    lambda side: side * side,
21          "circle":    lambda r: math.pi * r * r
22      }
23
24      calculator = area_calculators.get(shape)
25
26      if not calculator:
27          raise ValueError(f"Unsupported shape: {shape}")
28
29      if shape == "rectangle":
30          return calculator(x, y)
31      else: # For square and circle
32          return calculator(x)
33
34  # --- Example Usage ---
35  print(f"Rectangle Area: {calculate_area_refactored_functional('rectangle', 10, 5)}")
36  print(f"Square Area:    {calculate_area_refactored_functional('square', 7)}")
37  print(f"Circle Area:    {calculate_area_refactored_functional('circle', 3):.2f}")
38
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab13.4/13.4.4.py
Original loop result: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
List comprehension result: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab13.4/13.4.1.py
Rectangle Area: 50
Square Area:    49
Circle Area:    28.27
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The function calculate_area computes the area of a rectangle, square, or circle based on the given dimensions. It converts the shape name to lowercase for consistency, uses x as the main dimension and y as an optional width for rectangles, and calculates the area accordingly. For rectangles, both x and y are required, squares use x**2, and circles use math.pi * x**2. It raises an error if the shape is unknown or if required dimensions are missing. The docstring explains its usage and parameters.

TASK-02:

Error Handling in Legacy Code.

PROMPT:

The following python code that reads the file but it doesn't handle the errors . rewrite the code by correcting all the errors.

```python
def read_file(filename):
f = open(filename, "r")
data = f.read()

f.close()
return data
```

CODE:

```python
lab13.4 > 🐍 13.4.2.py > ...
1   import os
2
3   def read_file_legacy(filename: str) -> str:
4       """
5       Legacy function to read a file without proper error or resource handling.
6
7       WARNING: This function is unsafe. It can leak file descriptors if an
8       error occurs during the read operation and will crash if the file
9       is not found.
10      """
11      print(f"Attempting to read '{filename}' with legacy function...")
12      f = open(filename, "r")
13      data = f.read()
14      f.close()
15      return data
16
17  def read_file_safely(filename: str) -> str | None:
18      """
19      Refactored function to safely read a file using modern best practices.
20
21      This function uses:
22      1. A 'with' statement to ensure the file is automatically closed, even
23         if errors occur. This prevents resource leaks.
24      2. A 'try...except' block to gracefully handle exceptions like
25         FileNotFoundError, preventing the program from crashing.
26      3. Explicit 'utf-8' encoding to avoid potential UnicodeDecodeError on
27         different systems.
28
29      Args:
30          filename (str): The path to the file to read.
31
32      Returns:
33          str | None: The content of the file as a string, or None if an
34                      error occurred.
35      """
36      print(f"Attempting to read '{filename}' with safe function...")
37      try:
38          with open(filename, 'r', encoding='utf-8') as f:
39              return f.read()
```

0 ⚠ 0   🔍   Ln 1, Col 1   Spaces: 4

```
17    def read_file_safely(filename: str) -> str | None:
          with open(filename, "r", encoding="utf-8") as f:
39              return f.read()
40      except FileNotFoundError:
41          print(f"  Error: The file '{filename}' was not found.")
42          return None
43      except IOError as e:
44          print(f"  Error: An I/O error occurred while reading '{filename}': {e}")
45          return None
46
47  # --- Demonstration ---
48  if __name__ == "__main__":
49      existing_file = "sample.txt"
50      non_existent_file = "does_not_exist.txt"
51
52      # Create a dummy file for the successful case
53      with open(existing_file, "w", encoding="utf-8") as f:
54          f.write("Hello, world! This is a test.")
55
56      print("--- 1. Testing the safe function ---")
57      content = read_file_safely(existing_file)
58      if content:
59          print(f"  Success! Content: '{content}'")
60
61      print("\n--- 2. Testing the safe function with a non-existent file ---")
62      read_file_safely(non_existent_file)
63
64      # Clean up the dummy file
65      os.remove(existing_file)
66      print(f"\nCleaned up {existing_file}.")
```

OUTPUT:

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab13.4/13.4.2.py
--- 1. Testing the safe function ---
Attempting to read 'sample.txt' with safe function...
  Success! Content: 'Hello, world! This is a test.'

--- 2. Testing the safe function with a non-existent file ---
Attempting to read 'does_not_exist.txt' with safe function...
  Error: The file 'does_not_exist.txt' was not found.

Cleaned up sample.txt.
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The refactored function safely reads a file using with open(), ensuring the file is automatically closed, and uses try-except to handle errors like missing files or read failures. It provides clear error messages instead of crashing, making the code more robust and reliable.

TASK-03:

Complex refactoring

PROMPT:

Rewrite the following code by adding the proper variable names and refactor it in a proper way.

```
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:

```
lab13.4 >  13.4.3.py >  Student
 1    from typing import List
 2
 3    class Student:
 4        """
 5        Represents a student with their personal details and academic marks.
 6
 7        This class encapsulates student information, providing methods to
 8        display details and calculate total marks.
 9        """
10
11        def __init__(self, name: str, age: int, marks: List[float]):
12            """
13            Initializes a new Student object.
14
15            Args:
16                name (str): The full name of the student.
17                age (int): The age of the student in years.
18                marks (List[float]): A list of numerical marks obtained by the student
19                                     in various subjects or assessments.
20            """
21            # AI improves naming: n -> name, a -> age, m1, m2, m3 -> marks (list)
22            self.name = name
23            self.age = age
24            self.marks = marks
25
26        def display_details(self):
27            """
28            Prints the student's name and age in a readable format.
29            """
30            # AI improves print readability using f-strings
31            print(f"--- Student Details ---")
32            print(f"Name: {self.name}")
33            print(f"Age: {self.age}")
34            print(f"Marks: {self.marks}") # Displaying the list of marks
35
36        def calculate_total_marks(self) -> float:
37            """
38            Calculates the sum of all marks obtained by the student.
39
```

```
3    class Student:
36        def calculate_total_marks(self) -> float:

40            Returns:
41                float: The total sum of marks. Returns 0.0 if no marks are present.
42            """
43            # AI uses sum() for better readability and efficiency
44            return sum(self.marks)
45
46        def calculate_average_marks(self) -> float:
47            """
48            Calculates the average of all marks obtained by the student.
49
50            Returns:
51                float: The average marks. Returns 0.0 if no marks are present.
52            """
53            if not self.marks:
54                return 0.0
55            return sum(self.marks) / len(self.marks)
56
57    # --- Example Usage ---
58    if __name__ == "__main__":
59        student1 = Student("Alice Smith", 18, [85.5, 90.0, 78.5])
60        student1.display_details()
61        print(f"Total Marks: {student1.calculate_total_marks():.2f}")
62        print(f"Average Marks: {student1.calculate_average_marks():.2f}")
63
64        print("\n--- Another Student ---")
65        student2 = Student("Bob Johnson", 19, [70, 65, 80, 75])
66        student2.display_details()
67        print(f"Total Marks: {student2.calculate_total_marks():.2f}")
68        print(f"Average Marks: {student2.calculate_average_marks():.2f}")
```

OUTPUT:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab13.4/13.4.3.py
 --- Student Details ---
Name: Alice Smith
Age: 18
Marks: [85.5, 90.0, 78.5]
Total Marks: 254.00
Average Marks: 84.67

 --- Another Student ---
 --- Student Details ---
Name: Bob Johnson
Age: 19
Marks: [70, 65, 80, 75]
Total Marks: 290.00
Average Marks: 72.50
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The refactored Student class improves readability and modularity by using meaningful names, storing marks in a list, and adding docstrings. The details method prints information clearly with formatted strings, and total efficiently sums marks using sum(). The design is now more flexible and easier to extend.

TASK-04:

Inefficient Loop Refactoring

PROMPT:

I have a Python loop that computes squares of numbers and appends them to a list, but it seems inefficient. Can you rewrite it in a shorter, more Pythonic way and explain why it's better?

nums = [1,2,3,4,5,6,7,8,9,10]
squares = []
for i in nums:
squares.append(i * i

CODE:

```
lab13.4 > 🐍 13.4.4.py > ...
  1   # Original (inefficient) loop
  2   nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
  3   squares_old = []
  4   for i in nums:
  5   │   squares_old.append(i * i)
  6   print(f"Original loop result: {squares_old}")
  7
  8   # Refactored using a list comprehension
  9   squares_new = [i * i for i in nums]
 10   print(f"List comprehension result: {squares_new}")
 11
```

OUTPUT:

```
PS C:\Users\ramch\OneDrive\Desktop\ai> & C:/Users/ramch/AppData/Local/Programs/Python/Python312/python.exe c:/Users/ramch/OneDrive/Desktop/ai/lab13.4/13.4.4.py
Original loop result: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
List comprehension result: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\ramch\OneDrive\Desktop\ai>
```

OBSERVATION:

The list comprehension makes the code shorter, more readable, and efficient by replacing the explicit loop and append() method with a single expression.