

AI Coding Lab - Refactored Tasks

Name: Alla Sreemanth Reddy

HT.NO:2403A510G1

Task 1 – Remove Repetition

Prompt:

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

Code:

```
C: > AI > n.py > ...

1  import math
2  def area_rectangle(x, y):
3      return x * y
4  def area_square(x):
5      return x * x
6  def area_circle(r):
7      return math.pi * r * r
8  AREA_FUNCTIONS = {
9      "rectangle": lambda x, y=0: area_rectangle(x, y),
10     "square": lambda x, y=0: area_square(x),
11     "circle": lambda x, y=0: area_circle(x),
12 }
13 def calculate_area(shape, x, y=0):
14     """Calculates area of given shape (rectangle, square, circle)."""
15     if shape not in AREA_FUNCTIONS:
16         raise ValueError(f"Unknown shape: {shape}")
17     return AREA_FUNCTIONS[shape](x, y)
18 if __name__ == "__main__":
19     print("Rectangle (5x3):", calculate_area("rectangle", 5, 3))
20     print("Square (4):", calculate_area("square", 4))
21     print("Circle (radius 2):", calculate_area("circle", 2))
22
```

Problems 2 Output Debug Console Terminal Ports

```
PS C:\Users\allas> & C:/Python313/python.exe c:/AI/n.py
Rectangle (5x3): 15
Square (4): 16
Circle (radius 2): 12.566370614359172
PS C:\Users\allas>
```

Output:

import math

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

```
def area_square(x):  
    return x * x
```

```
def area_circle(r):  
    return math.pi * r * r
```

```
AREA_FUNCTIONS = {  
    "rectangle": lambda x, y=0: area_rectangle(x, y),  
    "square": lambda x, y=0: area_square(x),  
    "circle": lambda x, y=0: area_circle(x),  
}
```

```
def calculate_area(shape, x, y=0):  
    return AREA_FUNCTIONS[shape](x, y)
```

Observation:

The refactored version eliminates repetitive code by using modular functions and dictionary dispatch. It is more readable, easier to extend, and follows clean coding practices.

Task 2 – Error Handling in Legacy Code

Prompt:

Legacy function without proper error handling. Refactor with context manager and try-except.

Code:

```

1  def read_file(filename):
2      """Reads content from a file safely."""
3      try:
4          with open(filename, "r", encoding="utf-8") as f:
5              return f.read()
6      except FileNotFoundError:
7          print(f"Error: File '{filename}' not found.")
8      except IOError as e:
9          print(f"I/O error occurred: {e}")
10     return None
11 if __name__ == "__main__":
12     filename = input("Enter the filename to read: ").strip()
13     content = read_file(filename)
14     if content is not None:
15         print("\nFile Content:\n")
16         print(content)
17 Ctrl+L to chat, Ctrl+K to generate

```

Problems 2 Output Debug Console Terminal Ports

```

PS C:\Users\allas> & C:/Python313/python.exe c:/AI/q.py
Enter the filename to read: mintu
Error: File 'mintu' not found.
PS C:\Users\allas> 

```

Output:

```

def read_file(filename):
    """Reads content from a file safely."""
    try:
        with open(filename, "r", encoding="utf-8") as f:
            return f.read()
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
    except IOError as e:
        print(f"I/O error occurred: {e}")
    return None

```

Observation:

The refactored version uses 'with open()' to ensure safe file handling and adds try-except blocks for error handling. This improves reliability and robustness of the function.

Task 3 – Complex Refactoring

Prompt:

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

Code:

```
1 class Student:
2     """
3     A class to represent a student with basic details and marks.
4     """
5     def __init__(self, name: str, age: int, marks: list[int]):
6         """
7         Initialize a Student object.
8
9         :param name: Name of the student
10        :param age: Age of the student
11        :param marks: List of marks in subjects
12        """
13        self.name = name
14        self.age = age
15        self.marks = marks
16
17    def show_details(self) -> None:
18        """
19        Prints the details of the student in a readable format.
20        """
21        print(f"Student Details:\n Name: {self.name}\n Age: {self.age}")
22
23    def total_marks(self) -> int:
24        """
25        Calculates the total marks of the student.
26
27        :return: Sum of all marks
28        """
29        return sum(self.marks)
30
31    if __name__ == "__main__":
32        student = Student("Alice", 20, [85, 90, 88])
33        student.show_details()
34        print(f"Total Marks: {student.total_marks()}")
35
36    Ctrl+L to chat, Ctrl+K to generate
```

Problems Output Debug Console Terminal Ports

```
PS C:\Users\allas> & C:/Python313/python.exe c:/AI/w.py
Student Details:
  Name: Alice
  Age: 20
Total Marks: 263
PS C:\Users\allas>
```

Output:

```
class Student:
    """Represents a student with name, age, and marks."""
    def __init__(self, name, age, marks):
        self.name = name
        self.age = age
        self.marks = marks

    def details(self):
        print(f"Name: {self.name}, Age: {self.age}")

    def total(self):
        return sum(self.marks)
```

Observation:

The refactored version improves variable naming, adds docstrings, stores marks in a list, and uses sum() for cleaner logic. The design is modular and maintainable.

Task 4 – Inefficient Loop Refactoring

Prompt:

Refactor this inefficient loop with AI help.

Code:

```
C: > AI > i.py > nums

1  nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
2  squares = []
3  for i in nums:
4      squares.append(i * i)
5  print("Original Output:", squares)
6  squares_refactored = [i * i for i in nums]
7  print("Refactored Output:", squares_refactored)
8

Problems  Output  Debug Console  Terminal  Ports
PS C:\Users\allas> & C:/Python313/python.exe c:/AI/i.py
Original Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Refactored Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\allas>
```

Output:

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

Observation:

The refactored version replaces the inefficient loop with a Python list comprehension, making the code concise, efficient, and Pythonic.