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 > ...

```python
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):
    """Calculates area of given shape (rectangle, square, circle)."""
    if shape not in AREA_FUNCTIONS:
        raise ValueError(f"Unknown shape: {shape}")
    return AREA_FUNCTIONS[shape](x, y)
if __name__ == "__main__":
    print("Rectangle (5x3):", calculate_area("rectangle", 5, 3))
    print("Square (4):", calculate_area("square", 4))
    print("Circle (radius 2):", calculate_area("circle", 2))

```

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:

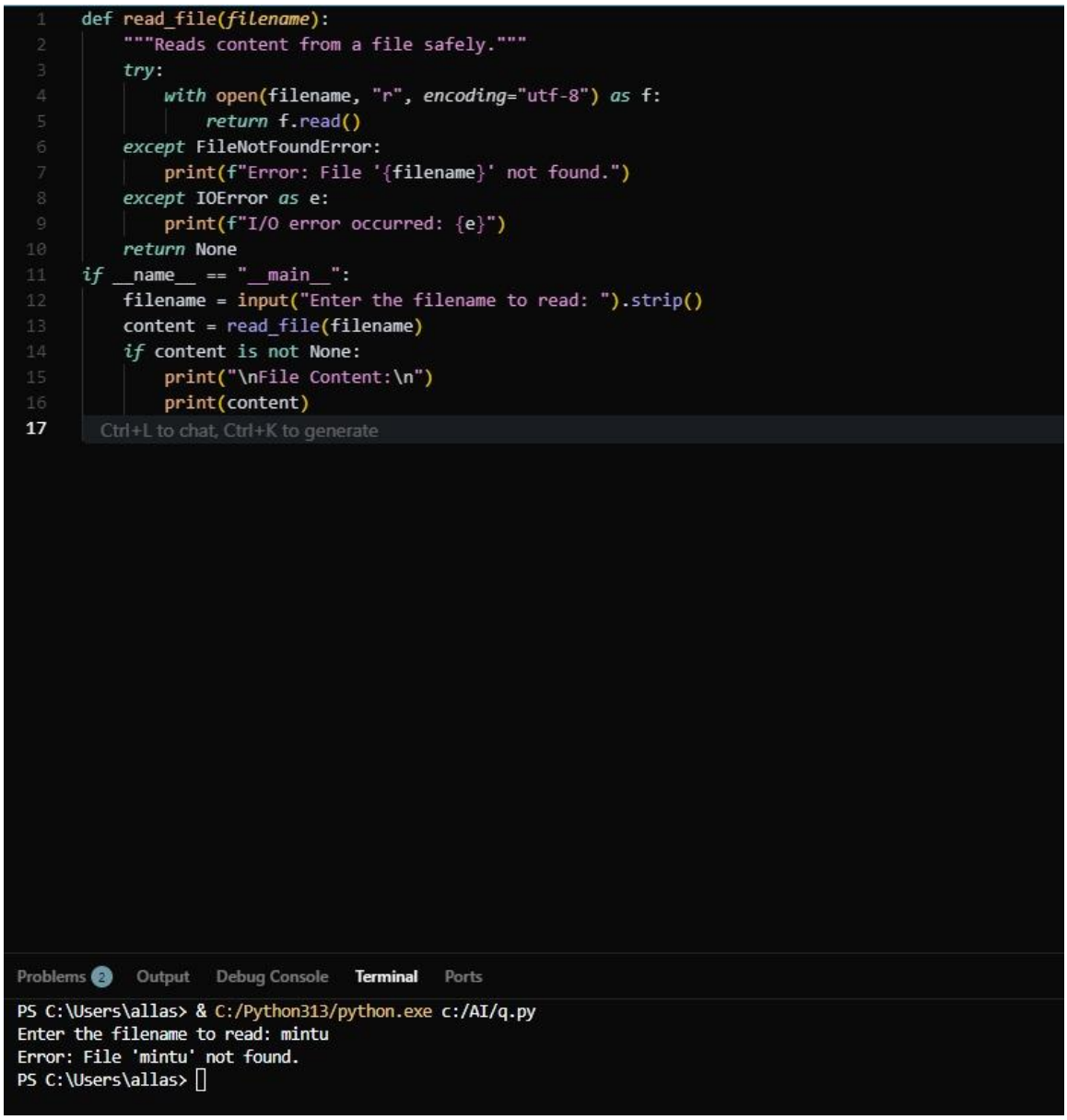
The refactored version eliminates repetitive code by using modular functions and dictionary dispatch. It is more readable, easier to extend, and follows clean coding practices.

Task 2 – Error Handling in Legacy Code

Prompt:

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

Code:



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:

```python
class Student:
    """
    A class to represent a student with basic details and marks.
    """
    def __init__(self, name: str, age: int, marks: list[int]):
        """
        Initialize a Student object.

        :param name: Name of the student
        :param age: Age of the student
        :param marks: List of marks in subjects
        """
        self.name = name
        self.age = age
        self.marks = marks

    def show_details(self) -> None:
        """
        Prints the details of the student in a readable format.
        """
        print(f"Student Details:\n  Name: {self.name}\n  Age: {self.age}")

    def total_marks(self) -> int:
        """
        Calculates the total marks of the student.

        :return: Sum of all marks
        """
        return sum(self.marks)
if __name__ == "__main__":
    student = Student("Alice", 20, [85, 90, 88])
    student.show_details()
    print(f"Total Marks: {student.total_marks()}")
```

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:

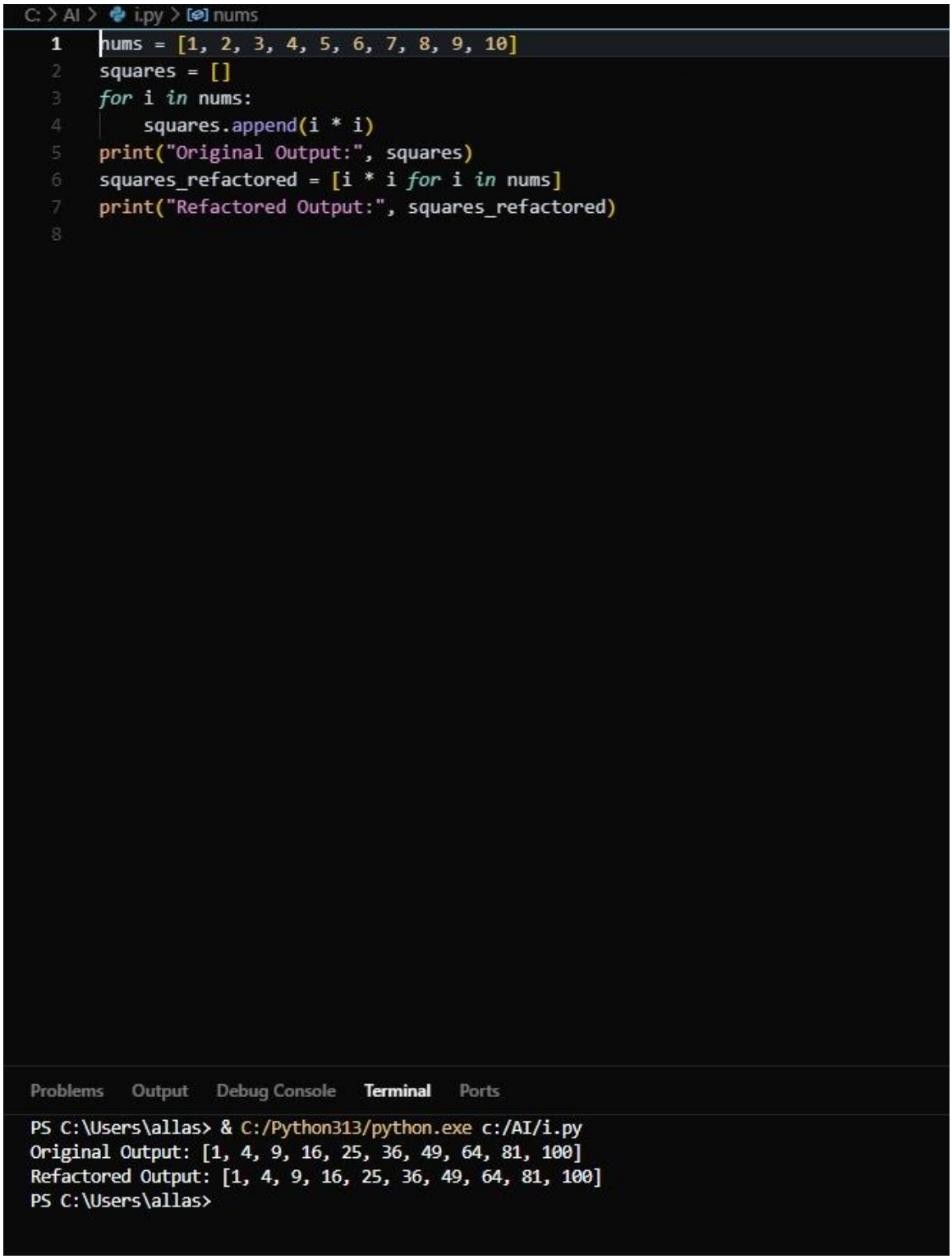
The refactored version improves variable naming, adds docstrings, stores marks in a list, and uses sum() for cleaner logic. The design is modular and maintainable.

Task 4 – Inefficient Loop Refactoring

Prompt:

Refactor this inefficient loop with AI help.

Code:



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.