LAB ASSIGNMENT-13.3

Name : G.Sudeeksha

Roll No : 2403A510E0

Course : AI Assisted Coding

Branch : CSE

Batch : 05

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
```

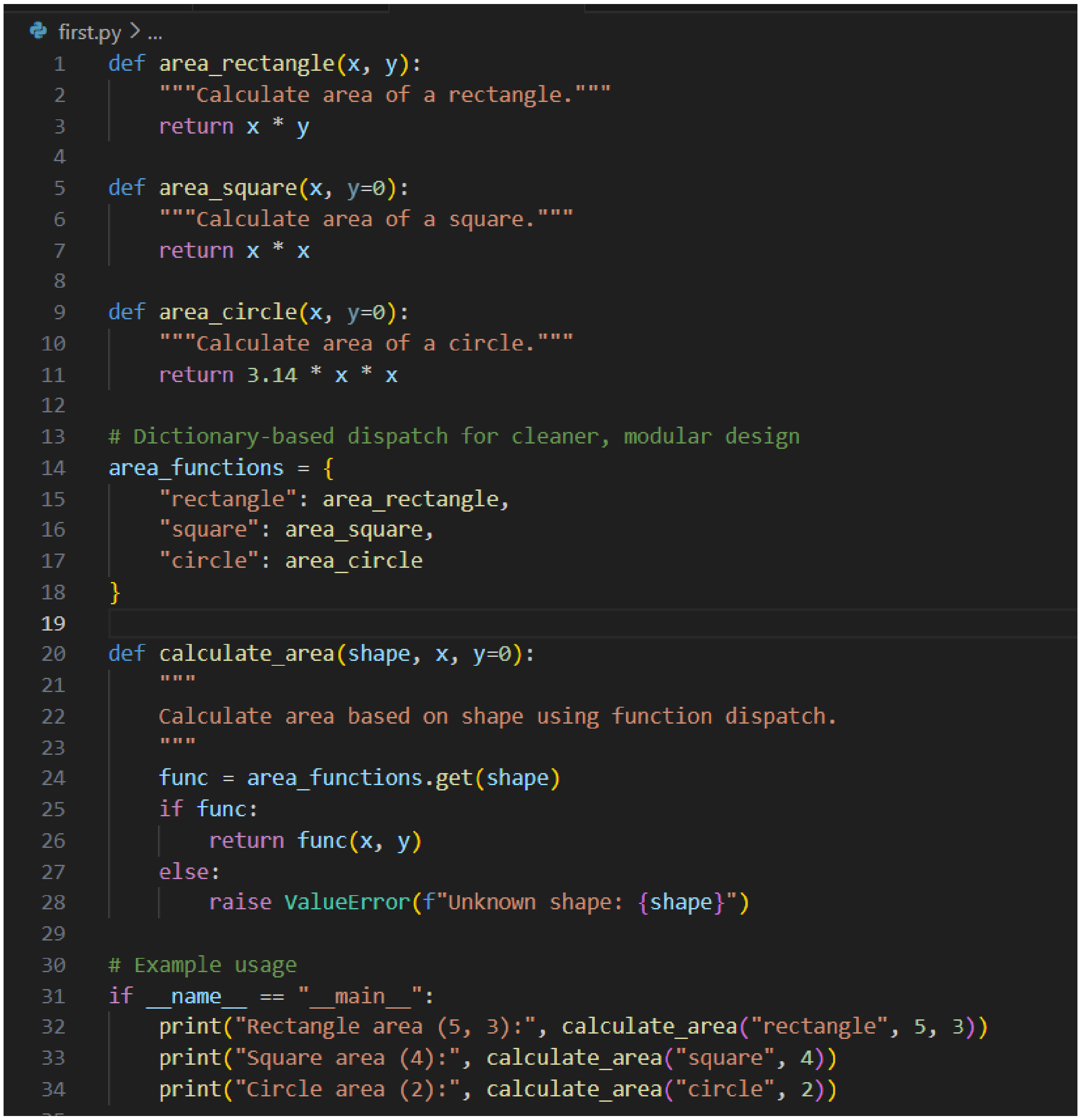
Output

- Refactored version with dictionary-based dispatch or separate functions.
- Cleaner and modular design.

#Prompt :

To make the redundant code and ask it to refactor

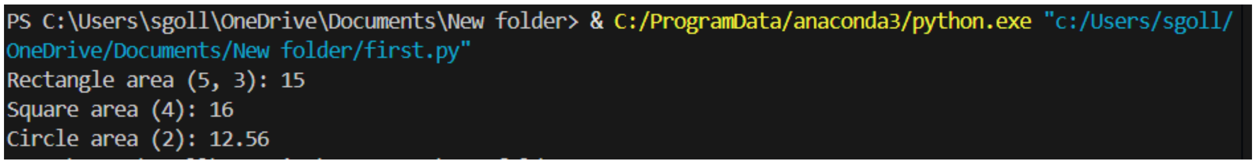
#Code :



first.py > ...

```python
def area_rectangle(x, y):
    """Calculate area of a rectangle."""
    return x * y

def area_square(x, y=0):
    """Calculate area of a square."""
    return x * x

def area_circle(x, y=0):
    """Calculate area of a circle."""
    return 3.14 * x * x

# Dictionary-based dispatch for cleaner, modular design
area_functions = {
    "rectangle": area_rectangle,
    "square": area_square,
    "circle": area_circle
}

def calculate_area(shape, x, y=0):
    """
    Calculate area based on shape using function dispatch.
    """
    func = area_functions.get(shape)
    if func:
        return func(x, y)
    else:
        raise ValueError(f"Unknown shape: {shape}")

# Example usage
if __name__ == "__main__":
    print("Rectangle area (5, 3):", calculate_area("rectangle", 5, 3))
    print("Square area (4):", calculate_area("square", 4))
    print("Circle area (2):", calculate_area("circle", 2))
```

#Output :


```
PS C:\Users\sgoll\OneDrive\Documents\New folder> & C:/ProgramData/anaconda3/python.exe "c:/Users/sgoll/OneDrive/Documents/New folder/first.py"
Rectangle area (5, 3): 15
Square area (4): 16
Circle area (2): 12.56
```

#Observation :

The refactored code demonstrates a modular and maintainable approach to calculating the area of different shapes. By defining separate functions for each shape and using a dictionary-based dispatch mechanism, the code eliminates redundancy and enhances readability. This design makes it easy to add new shapes in the future and centralizes the logic for area calculation. The use of descriptive function names and docstrings further improves clarity, while the main block provides clear examples of usage. Overall, the refactoring results in cleaner, more scalable, and easier-to-manage code.

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()

.close()

return data

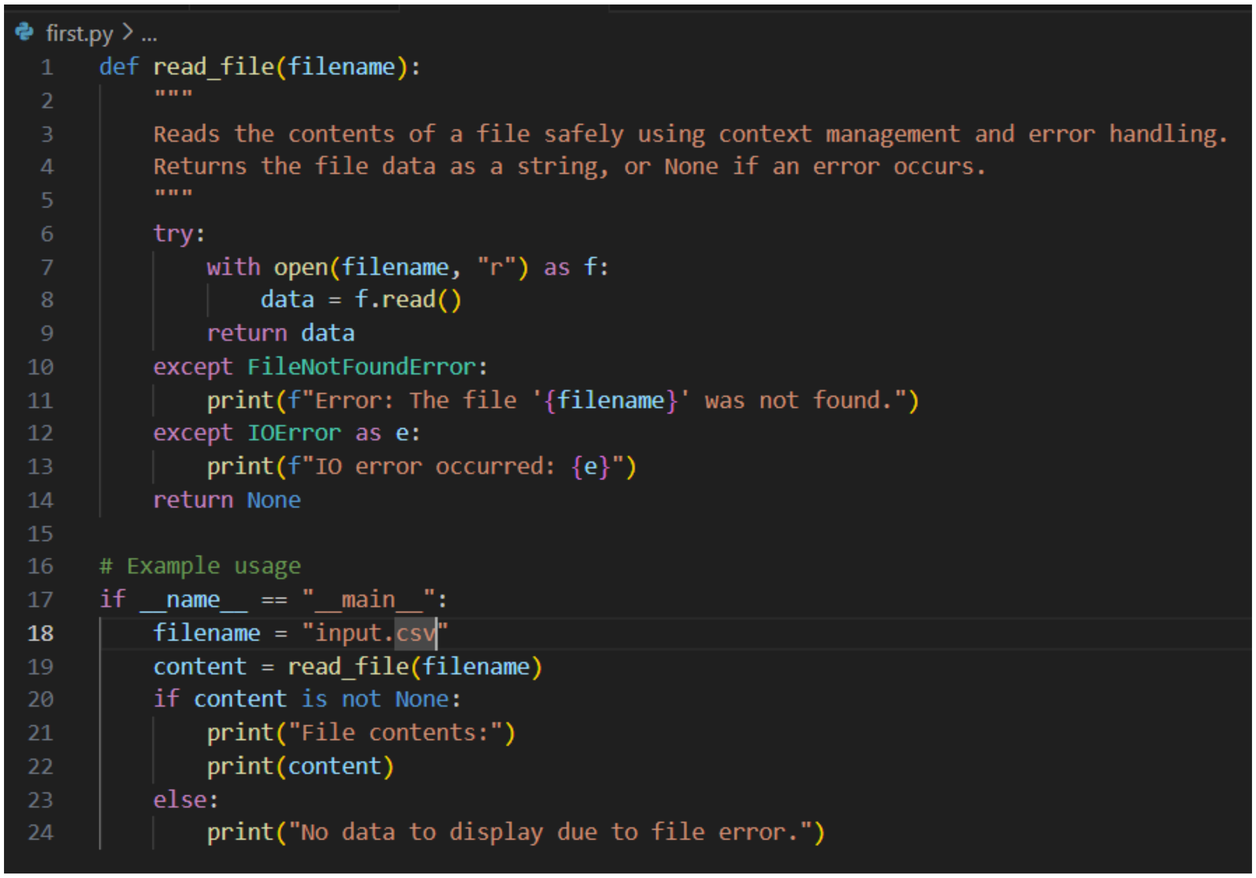
Output:

AI refactors with with open() and try-except:

#Prompt :

Write a python code by sing the Legacy function without proper error handling

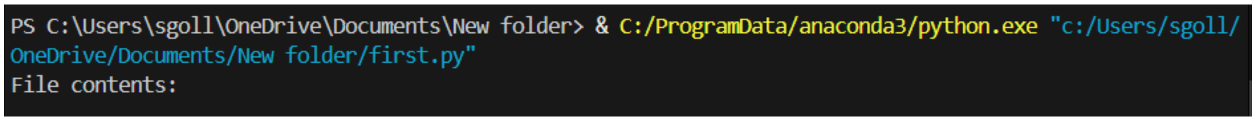
#Code :



first.py > ...

```python
def read_file(filename):
    """
    Reads the contents of a file safely using context management and error handling.
    Returns the file data as a string, or None if an error occurs.
    """
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        print(f"Error: The file '{filename}' was not found.")
    except IOError as e:
        print(f"IO error occurred: {e}")
    return None

# Example usage
if __name__ == "__main__":
    filename = "input.csv"
    content = read_file(filename)
    if content is not None:
        print("File contents:")
        print(content)
    else:
        print("No data to display due to file error.")
```

#Output :



```
PS C:\Users\sgoll\OneDrive\Documents\New folder> & C:/ProgramData/anaconda3/python.exe "c:/Users/sgoll/
OneDrive/Documents/New folder/first.py"
File contents:
```

#Observation :

The refactored read_file function demonstrates robust and modern Python practices for file handling. By using a with open() context manager, it ensures that files are properly closed after reading, even if an error occurs. The inclusion of a try-except block provides clear and user-friendly error messages for common issues such as missing files or input/output errors, and prevents the program from crashing unexpectedly. Returning None in error cases allows the calling code to handle failures gracefully. Overall, this approach results in safer, more maintainable, and user-oriented file reading functionality.

Task Description #3 – Complex Refactoring

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

Python Code

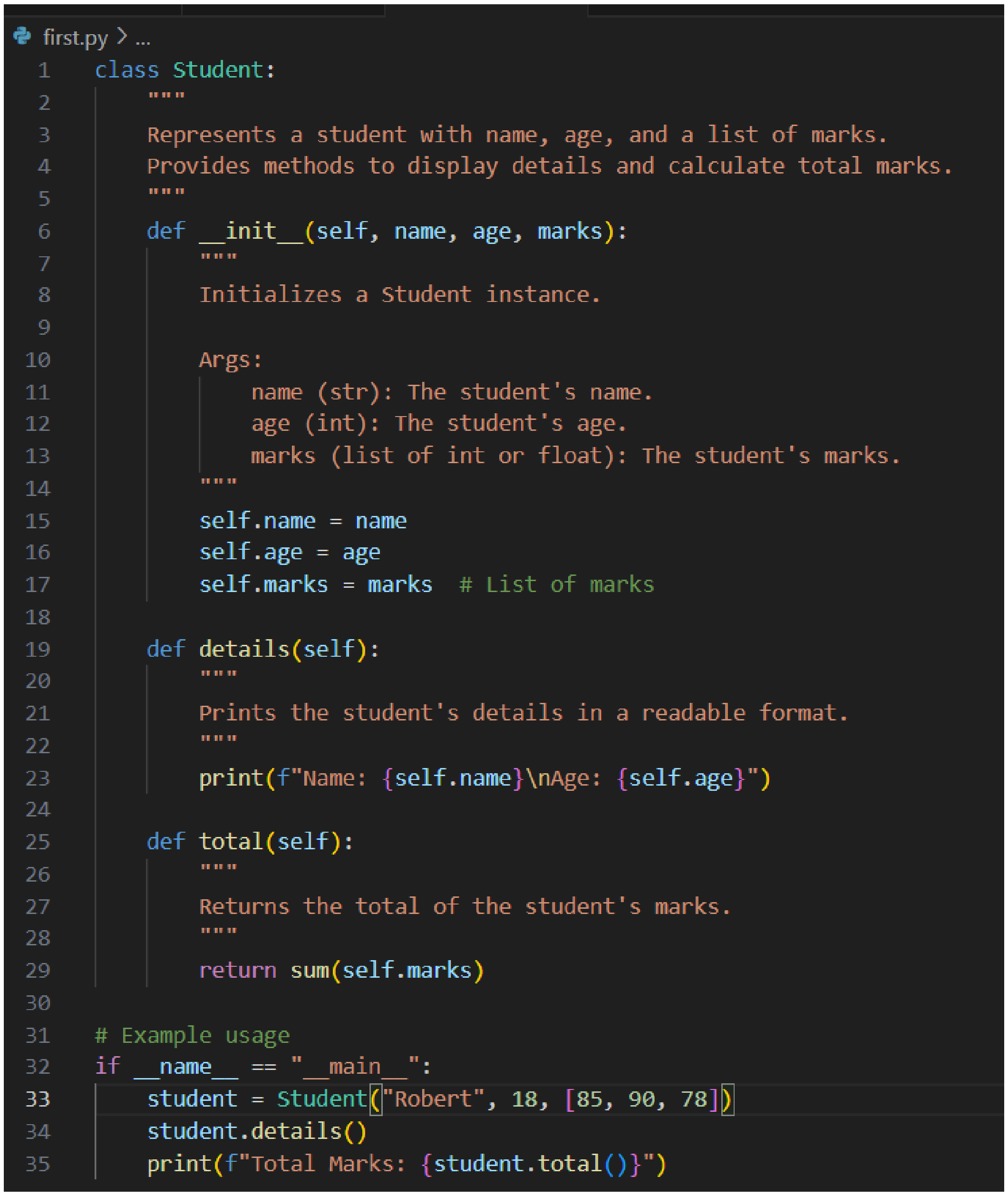
```
class Student:
def __init__(self, n, a, m1, m2, m3):
self.n = n
self.a = a
self.m1 = m1
self.m2 = m2
self.m3 = m3
def details(self):
```

```
print("Name:", self.n, "Age:", self.a)
def total(self):
return self.m1+self.m2+self.m3
Output :
• AI improves naming (name, age, marks).
• Adds docstrings.
• Improves print readability.
• Possibly uses sum(self.marks) if marks stored in a list.
```

#Prompt :

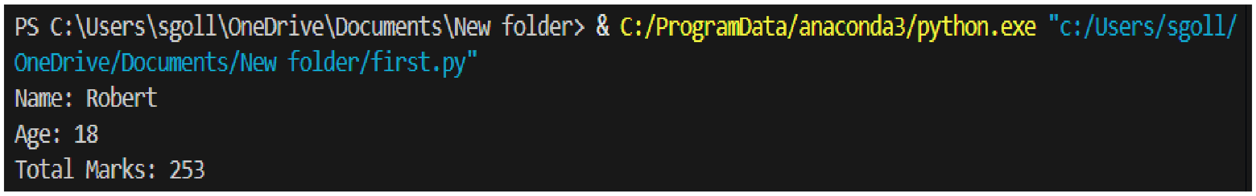
Write a python code to provide this legacy class to AI for readability and modularity improvements

#Code :


first.py > ...

```python
class Student:
    """
    Represents a student with name, age, and a list of marks.
    Provides methods to display details and calculate total marks.
    """
    def __init__(self, name, age, marks):
        """
        Initializes a Student instance.

        Args:
            name (str): The student's name.
            age (int): The student's age.
            marks (list of int or float): The student's marks.
        """
        self.name = name
        self.age = age
        self.marks = marks  # List of marks

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

    def total(self):
        """
        Returns the total of the student's marks.
        """
        return sum(self.marks)

# Example usage
if __name__ == "__main__":
    student = Student("Robert", 18, [85, 90, 78])
    student.details()
    print(f"Total Marks: {student.total()}")
```

#Output :



```
PS C:\Users\sgoll\OneDrive\Documents\New folder> & C:/ProgramData/anaconda3/python.exe "c:/Users/sgoll/
OneDrive/Documents/New folder/first.py"
Name: Robert
Age: 18
Total Marks: 253
```

#Observation :

The refactored Student class demonstrates improved readability and modularity by using clear attribute names (name, age, marks) and storing marks as a list, which allows for flexible handling of any number of subjects. The inclusion of descriptive docstrings for the class and its methods enhances code documentation and understanding. The details method provides a clean, formatted output of student information, while the total method leverages Python’s built-in sum() function for concise and efficient calculation of total marks. Overall, the design is clean, extensible, and easy to maintain, making it well-suited for real-world applications involving student data management.

Task Description #4 – Inefficient Loop Refactoring

Task: Refactor this inefficient loop with AI help

Python Code

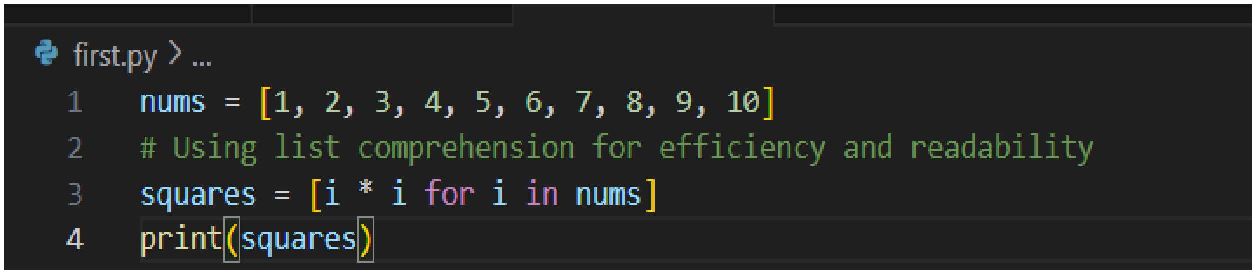
```
nums = [1,2,3,4,5,6,7,8,9,10]
squares = []
for i in nums:
squares.append(i * i)
```

Output: AI suggested a list comprehension

#Prompt :

Write a python code to refactor this inefficient loop with AI help

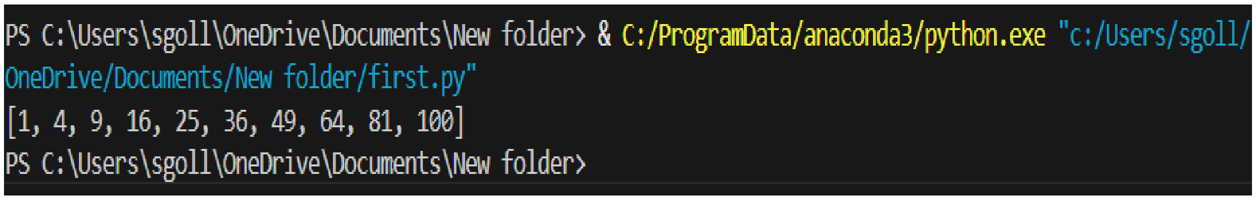
#Code :



first.py > ...

```python
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Using list comprehension for efficiency and readability
squares = [i * i for i in nums]
print(squares)
```

#Output :



```
PS C:\Users\sgoll\OneDrive\Documents\New folder> & C:/ProgramData/anaconda3/python.exe "c:/Users/sgoll/
OneDrive/Documents/New folder/first.py"
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\sgoll\OneDrive\Documents\New folder>
```

#Observation :

The refactored code efficiently generates a list of squares from the original list of numbers using a list comprehension. This approach is more concise and readable compared to the traditional for-loop with an append operation. List comprehensions in Python are not only syntactically cleaner but also generally faster for such simple transformations. As a result, the code is easier to maintain and clearly communicates its intent, making it a preferred choice for similar tasks involving list transformations.