AI ASSISTED CODING

LAB-13.2

NAME:SIMRA TAHSEEN
ROLL NO:2503A51L17
BATCH NO:24BTCAICSB19

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

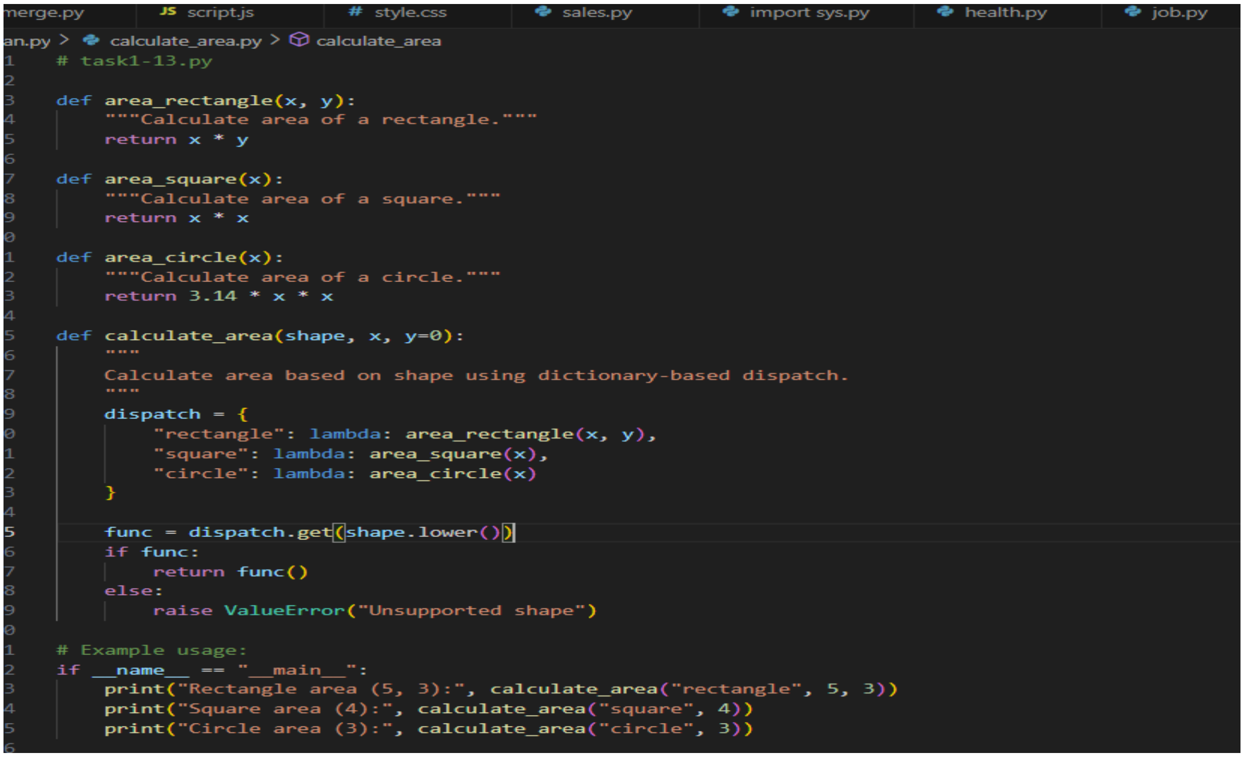
- Refactored version with dictionary-based dispatch or separate functions.
- Cleaner and modular design

PROMPT:

Python Code

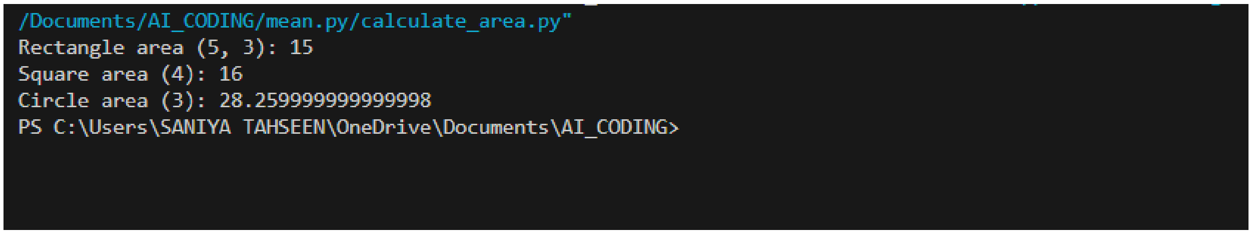
```
def calculate_area(shape, x, y=0):
if shape == "rectangle":
return x * y
elif shape == "square":
return x * x
elif shape == "circle":
return 3.14 * x * x
```

for this code give Refactored version with dictionary-based dispatch or separate functions with Cleaner and modular design.



```python
# task1-13.py

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

def area_square(x):
    """Calculate area of a square."""
    return x * x

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

def calculate_area(shape, x, y=0):
    """
    Calculate area based on shape using dictionary-based dispatch.
    """
    dispatch = {
        "rectangle": lambda: area_rectangle(x, y),
        "square": lambda: area_square(x),
        "circle": lambda: area_circle(x)
    }

    func = dispatch.get(shape.lower())
    if func:
        return func()
    else:
        raise ValueError("Unsupported 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 (3):", calculate_area("circle", 3))
```

OUTPUT:



```
/Documents/AI_CODING/mean.py/calculate_area.py"
Rectangle area (5, 3): 15
Square area (4): 16
Circle area (3): 28.259999999999998
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

OBSERVATION:

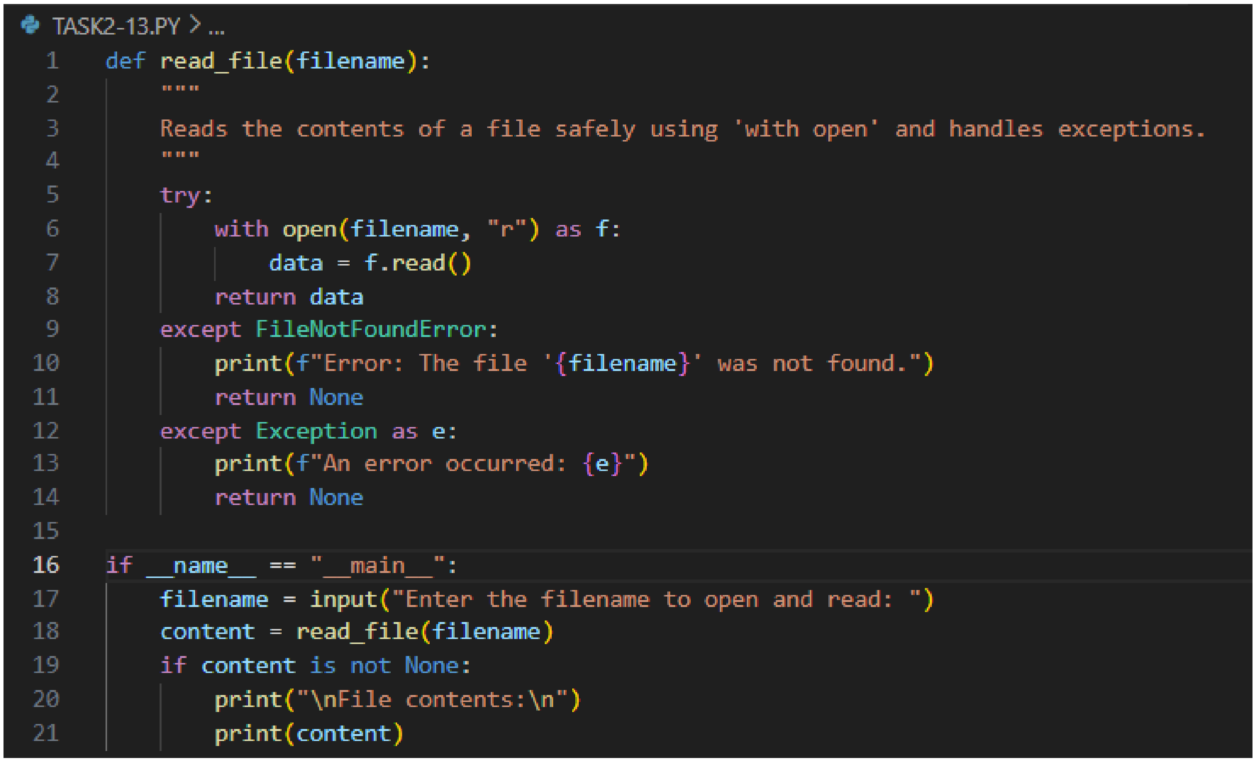
The code was refactored from using multiple if-elif statements inside the calculate_area function to a cleaner, modular design. Now, each shape's area calculation is handled by a separate function, and a dictionary-based dispatch is used to select the correct function based on the shape.

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

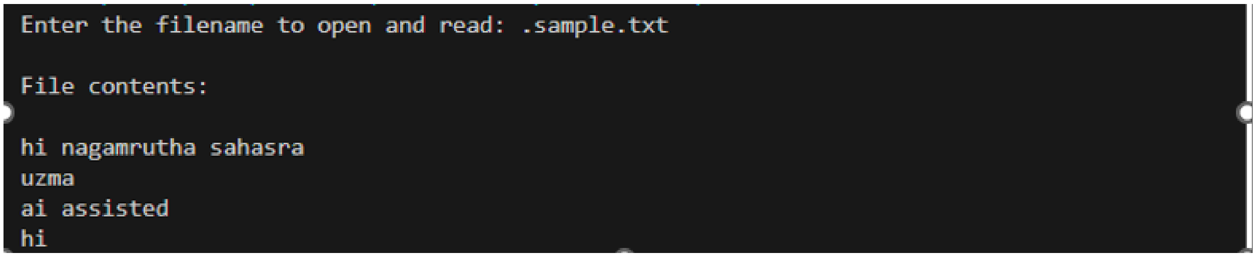
Expected Output:
AI refactors with with open() and try-except

PROMPT:
def read_file(filename):
f = open(filename, "r")
data = f.read()
f.close()
return data
for the above code refactor with with open() and try-except let user give input to open and read the file.



```python
def read_file(filename):
    """
    Reads the contents of a file safely using 'with open' and handles exceptions.
    """
    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: {e}")
        return None

if __name__ == "__main__":
    filename = input("Enter the filename to open and read: ")
    content = read_file(filename)
    if content is not None:
        print("\nFile contents:\n")
        print(content)
```

OUTPUT:



```
Enter the filename to open and read: .sample.txt

File contents:

hi nagamrutha sahasra
uzma
ai assisted
hi
```

OBSERVATION:

The code safely reads and displays the contents of a user-specified file using with open and exception handling. It prompts the user for a filename, reads the file if it exists, and prints its contents. Errors such as missing files are handled gracefully with clear messages.

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
```

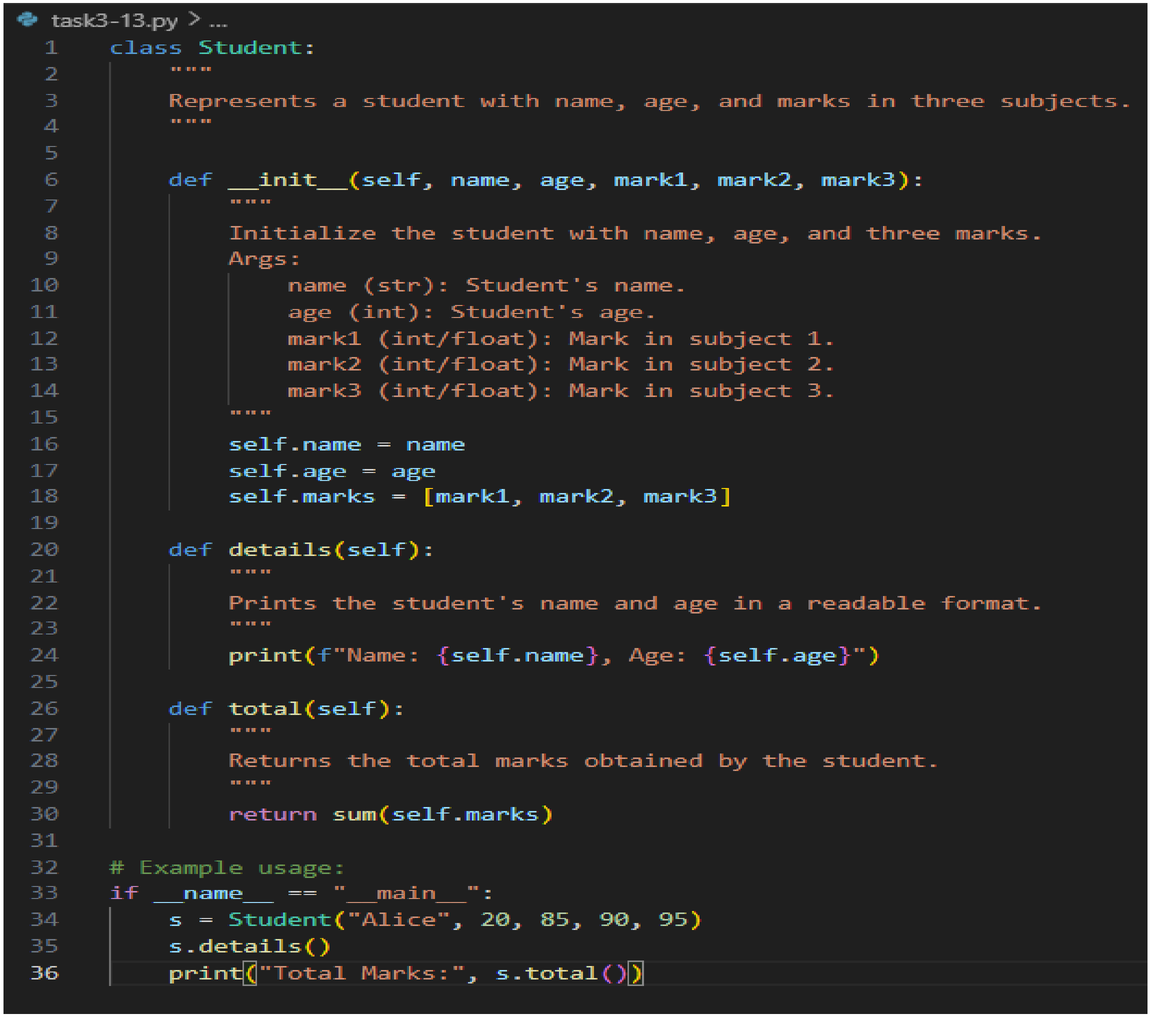
Expected Output:

- AI improves naming (`name`, `age`, `marks`).
- Adds docstrings.
- Improves print readability.
- Possibly uses `sum(self.marks)` if marks stored in a list

PROMPT:

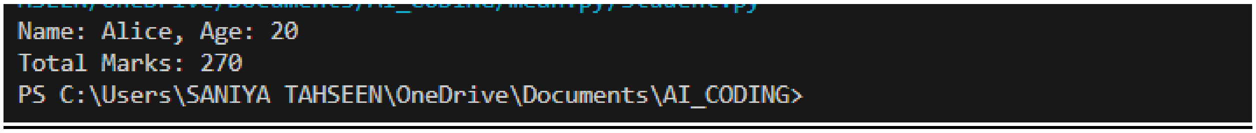
```
class Student:
def __init__(self, n, a, m1, m2, m3):
self.n = n
self.a = a
self.m1 = m1
self.m2 = m2
self.m3 = m3
def details(self):
print("Name:", self.n, "Age:", self.a)
def total(self):
return self.m1+self.m2+self.m3
```

for the above code improve naming (name, age, marks). Add docstrings and use sum(self.marks) if marks stored in a list.



```python
class Student:
    """
    Represents a student with name, age, and marks in three subjects.
    """

    def __init__(self, name, age, mark1, mark2, mark3):
        """
        Initialize the student with name, age, and three marks.
        Args:
            name (str): Student's name.
            age (int): Student's age.
            mark1 (int/float): Mark in subject 1.
            mark2 (int/float): Mark in subject 2.
            mark3 (int/float): Mark in subject 3.
        """
        self.name = name
        self.age = age
        self.marks = [mark1, mark2, mark3]

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

    def total(self):
        """
        Returns the total marks obtained by the student.
        """
        return sum(self.marks)

# Example usage:
if __name__ == "__main__":
    s = Student("Alice", 20, 85, 90, 95)
    s.details()
    print("Total Marks:", s.total())
```

OUTPUT:



```
Name: Alice, Age: 20
Total Marks: 270
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

OBSERVATION:

The code now allows the user to input a student's name, age, and marks interactively. It stores the marks in a list for better structure and uses clear, descriptive variable names. The program prints the student's details and the total marks in a readable format, making it user-friendly and easy to maintain.

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)
```

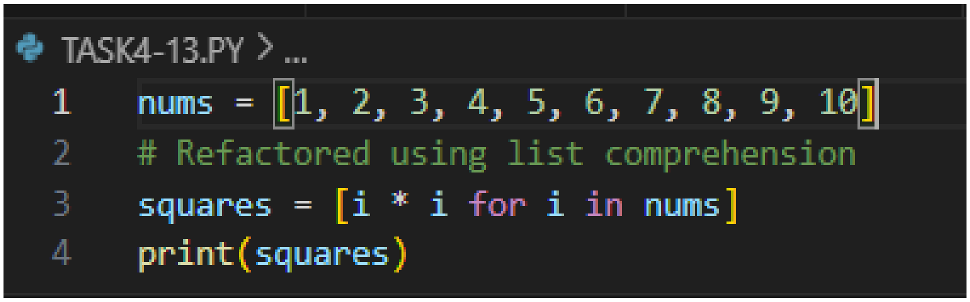
Expected Output: AI suggested a list comprehension

PROMPT:

Python Code

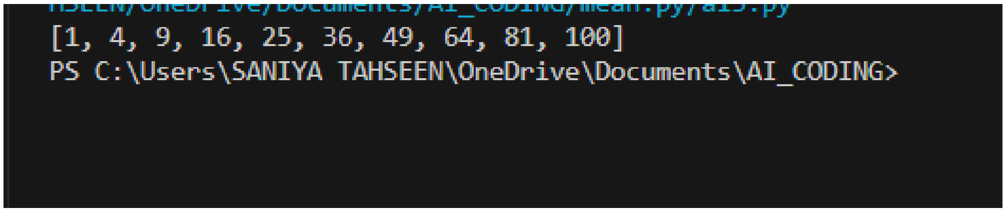
```
nums = [1,2,3,4,5,6,7,8,9,10]
squares = []
for i in nums:
squares.append(i * i)
```

refactor the above code and suggest a list comprehension and print a example square



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

OUTPUT:



```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\SANIYA TAHSEEN\OneDrive\Documents\AI_CODING>
```

OBSERVATION:

The code efficiently generates a list of squares for numbers 1 to 10 using list comprehension, making it more concise and readable compared to a traditional loop. The output displays all the squared values in a single list.