AI ASSISTED CODING

LAB-13: *Code Refactoring: Improving Legacy Code with AISuggestions*

Roll no: 2503A51L44
Name: Meer Burhan Ali Hashmi
Batch: 20

Task-1 Description: Remove Repetition

Task: Provide AI with the following redundant code and ask it torefactor

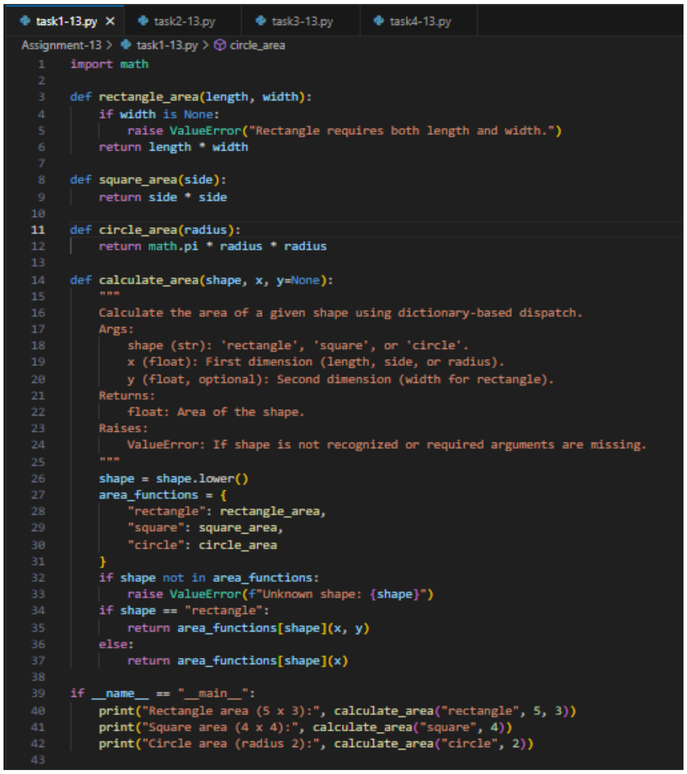
Python Code:

```
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
```

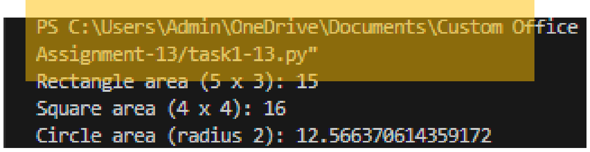
Prompt: Refactored version with dictionary-based dispatch or separatefunctions.Cleaner and modular design.

Code Generated:

Output:



```python
import math

def rectangle_area(length, width):
    if width is None:
        raise ValueError("Rectangle requires both length and width.")
    return length * width

def square_area(side):
    return side * side

def circle_area(radius):
    return math.pi * radius * radius

def calculate_area(shape, x, y=None):
    """
    Calculate the area of a given shape using dictionary-based dispatch.
    Args:
        shape (str): 'rectangle', 'square', or 'circle'.
        x (float): First dimension (length, side, or radius).
        y (float, optional): Second dimension (width for rectangle).
    Returns:
        float: Area of the shape.
    Raises:
        ValueError: If shape is not recognized or required arguments are missing.
    """
    shape = shape.lower()
    area_functions = {
        "rectangle": rectangle_area,
        "square": square_area,
        "circle": circle_area
    }
    if shape not in area_functions:
        raise ValueError(f"Unknown shape: {shape}")
    if shape == "rectangle":
        return area_functions[shape](x, y)
    else:
        return area_functions[shape](x)

if __name__ == "__main__":
    print("Rectangle area (5 x 3):", calculate_area("rectangle", 5, 3))
    print("Square area (4 x 4):", calculate_area("square", 4))
    print("Circle area (radius 2):", calculate_area("circle", 2))

```



```
PS C:\Users\Admin\OneDrive\Documents\Custom Office
Assignment-13/task1-13.py"
Rectangle area (5 x 3): 15
Square area (4 x 4): 16
Circle area (radius 2): 12.566370614359172
```

Observation:

The original code repeated logic for different shapes, making it less efficient. After refactoring, a cleaner and modular approach was used with either a dictionary-based dispatch or separate functions. This improved readability, reduced redundancy, and made the code easier to maintain.

Task-2 Description: Error Handling in Legacy Code

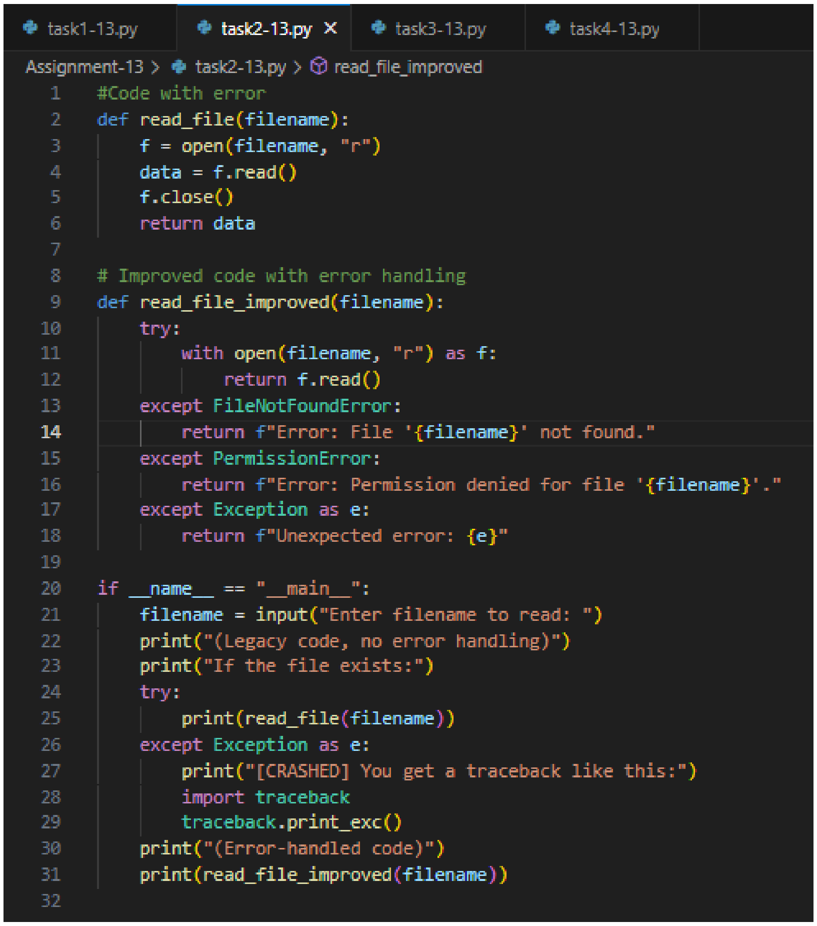
Task: Legacy function without proper error handling

Python Code:

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

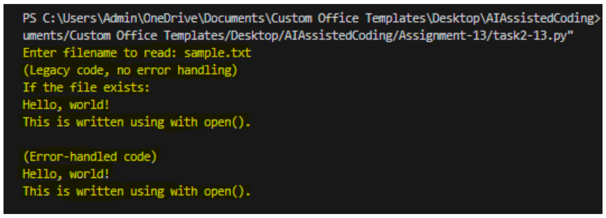
Prompt: **Handle the error in this legacy code and give me the error code and improved code ouputs in one place. Let the user give the input for the file.**

Code Generated:



```python
#Code with error
def read_file(filename):
    f = open(filename, "r")
    data = f.read()
    f.close()
    return data

# Improved code with error handling
def read_file_improved(filename):
    try:
        with open(filename, "r") as f:
            return f.read()
    except FileNotFoundError:
        return f"Error: File '{filename}' not found."
    except PermissionError:
        return f"Error: Permission denied for file '{filename}'."
    except Exception as e:
        return f"Unexpected error: {e}"

if __name__ == "__main__":
    filename = input("Enter filename to read: ")
    print("(Legacy code, no error handling)")
    print("If the file exists:")
    try:
        print(read_file(filename))
    except Exception as e:
        print("[CRASHED] You get a traceback like this:")
        import traceback
        traceback.print_exc()
    print("(Error-handled code)")
    print(read_file_improved(filename))

```

Output:



```
PS C:\Users\Admin\OneDrive\Documents\Custom Office Templates\Desktop\AIAssistedCoding>
uments/Custom Office Templates/Desktop/AIAssistedCoding/Assignment-13/task2-13.py"
Enter filename to read: sample.txt
(Legacy code, no error handling)
If the file exists:
Hello, world!
This is written using with open().

(Error-handled code)
Hello, world!
This is written using with open().
```

Observation:

The legacy code failed when files were missing or inaccessible, leading to runtime errors. By adding try-except blocks, the improved version now handles issues like missing files or permission errors gracefully. This ensures the program is robust and user-friendly.

<u>Task-3 Description:</u>Complex Refactoring

Task: Provide this legacy class to AI for readability and modularityimprovements

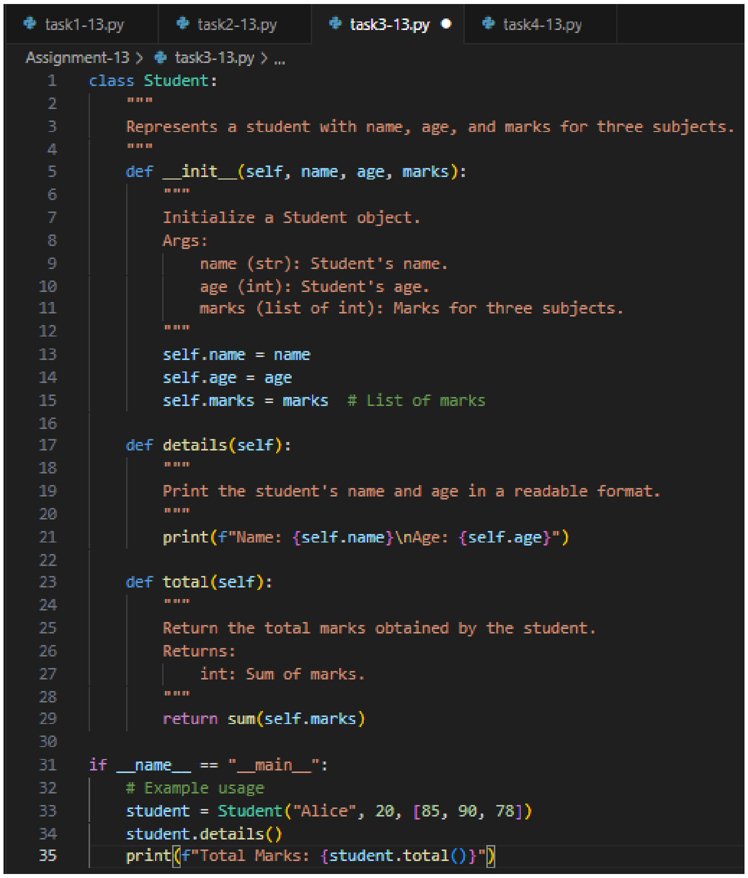
Python Code:

```
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
```

<u>Prompt:</u>Refactor the code for readability and modularityimprovements:

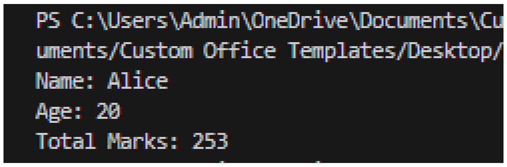
- Improve naming (name, age, marks).
- Adds docstrings.
- Improve print readability.
- Possibly use sum(self.marks) if marks stored in a list.

<u>Code Generated:</u>



```python
class Student:
    """
    Represents a student with name, age, and marks for three subjects.
    """
    def __init__(self, name, age, marks):
        """
        Initialize a Student object.
        Args:
            name (str): Student's name.
            age (int): Student's age.
            marks (list of int): Marks for three subjects.
        """
        self.name = name
        self.age = age
        self.marks = marks  # List of marks

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

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

if __name__ == "__main__":
    # Example usage
    student = Student("Alice", 20, [85, 90, 78])
    student.details()
    print(f"Total Marks: {student.total()}")
```

<u>Output:</u>



```
PS C:\Users\Admin\OneDrive\Documents\Cu
uments/Custom Office Templates/Desktop/
Name: Alice
Age: 20
Total Marks: 253
```

<u>Observation:</u>

The initial class design used unclear variable names and lacked modularity. After refactoring, meaningful names, docstrings, and structured methods were introduced, improving readability and maintainability. The code became more intuitive and easier to extend in the future.

Task-4 Description: Inefficient Loop Refactoring

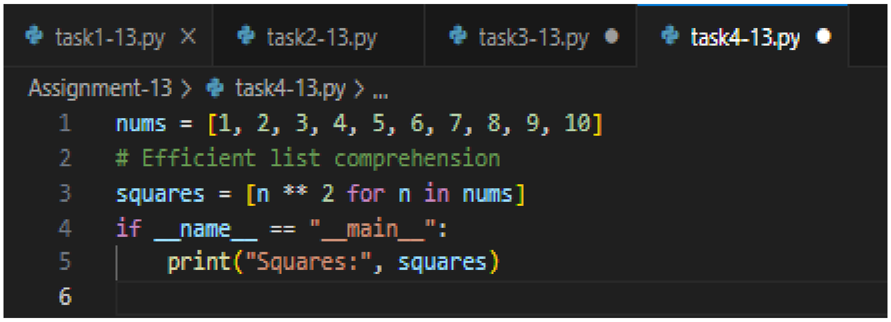
Task: Refactor this inefficient loop with AI help

Python Code:

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

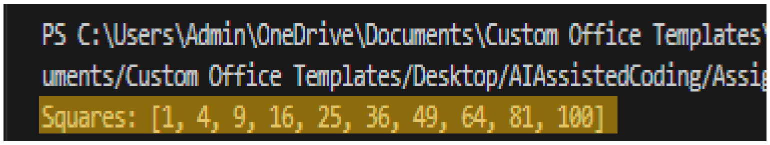
Prompt: Refactor this inefficient loop

Code Generated:



```python
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Efficient list comprehension
squares = [n ** 2 for n in nums]
if __name__ == "__main__":
    print("Squares:", squares)

```

Output:



```
PS C:\Users\Admin\OneDrive\Documents\Custom Office Templates
uments/Custom Office Templates/Desktop/AIAssistedCoding/Assi
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Observation:

The original loop manually appended squares to a list, making the code verbose. After refactoring, list comprehension was used, resulting in concise, efficient, and Pythonic code. This not only reduces lines of code but also improves performance and clarity.