ASSIGNMENT-13.2

NAME	:	BOMMAKANTI SRINITHI
ROLL NO	:	2403A51413
COURSE	:	AI ASSISTED CODING
BATCH	:	01 (AIML)

TASK1:

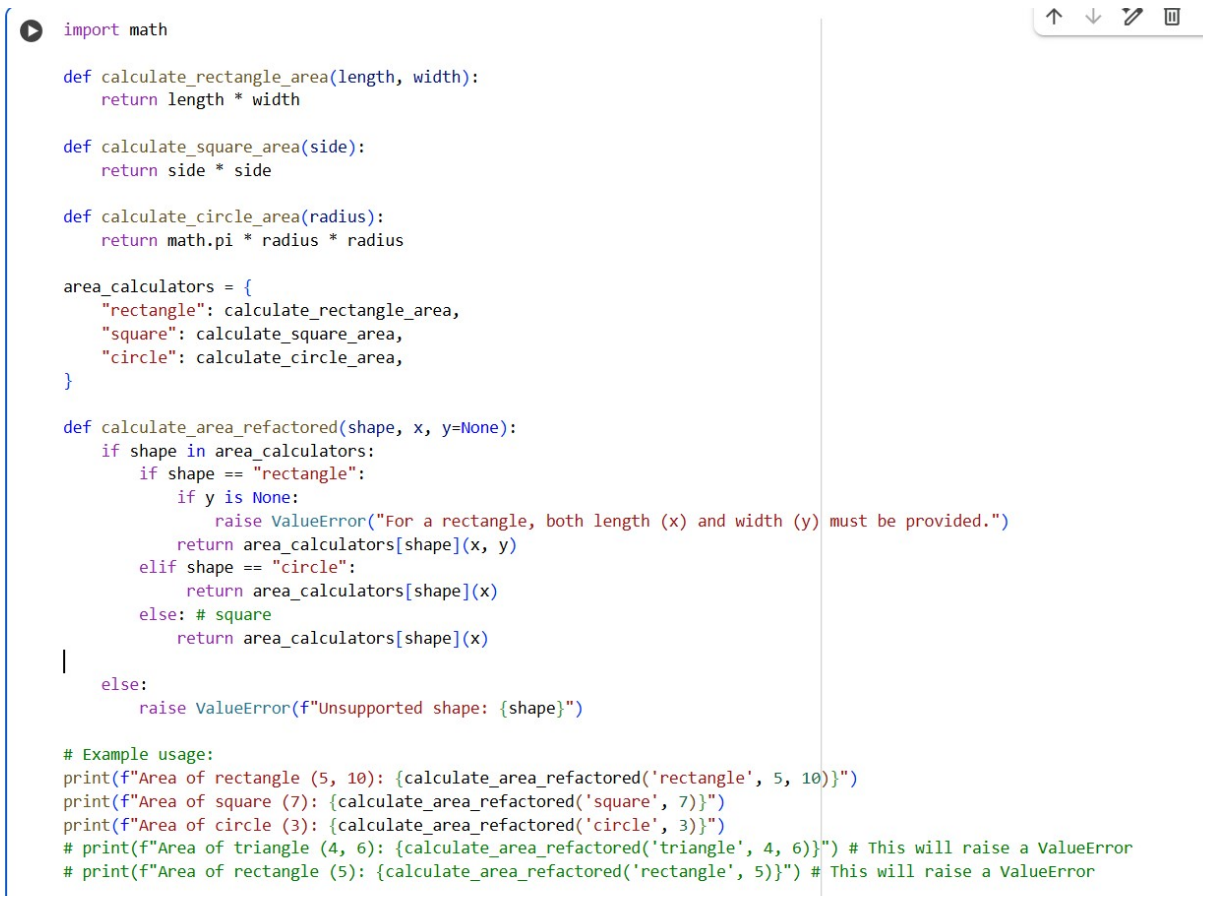
PROMPT :

Refactor this code to remove repetition and use a dictionary-based approach or separate functions.

```
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
```

CODE :



```python
import math

def calculate_rectangle_area(length, width):
    return length * width

def calculate_square_area(side):
    return side * side

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

area_calculators = {
    "rectangle": calculate_rectangle_area,
    "square": calculate_square_area,
    "circle": calculate_circle_area,
}

def calculate_area_refactored(shape, x, y=None):
    if shape in area_calculators:
        if shape == "rectangle":
            if y is None:
                raise ValueError("For a rectangle, both length (x) and width (y) must be provided.")
            return area_calculators[shape](x, y)
        elif shape == "circle":
             return area_calculators[shape](x)
        else: # square
            return area_calculators[shape](x)

    else:
        raise ValueError(f"Unsupported shape: {shape}")

# Example usage:
print(f"Area of rectangle (5, 10): {calculate_area_refactored('rectangle', 5, 10)}")
print(f"Area of square (7): {calculate_area_refactored('square', 7)}")
print(f"Area of circle (3): {calculate_area_refactored('circle', 3)}")
# print(f"Area of triangle (4, 6): {calculate_area_refactored('triangle', 4, 6)}") # This will raise a ValueError
# print(f"Area of rectangle (5): {calculate_area_refactored('rectangle', 5)}") # This will raise a ValueError
```

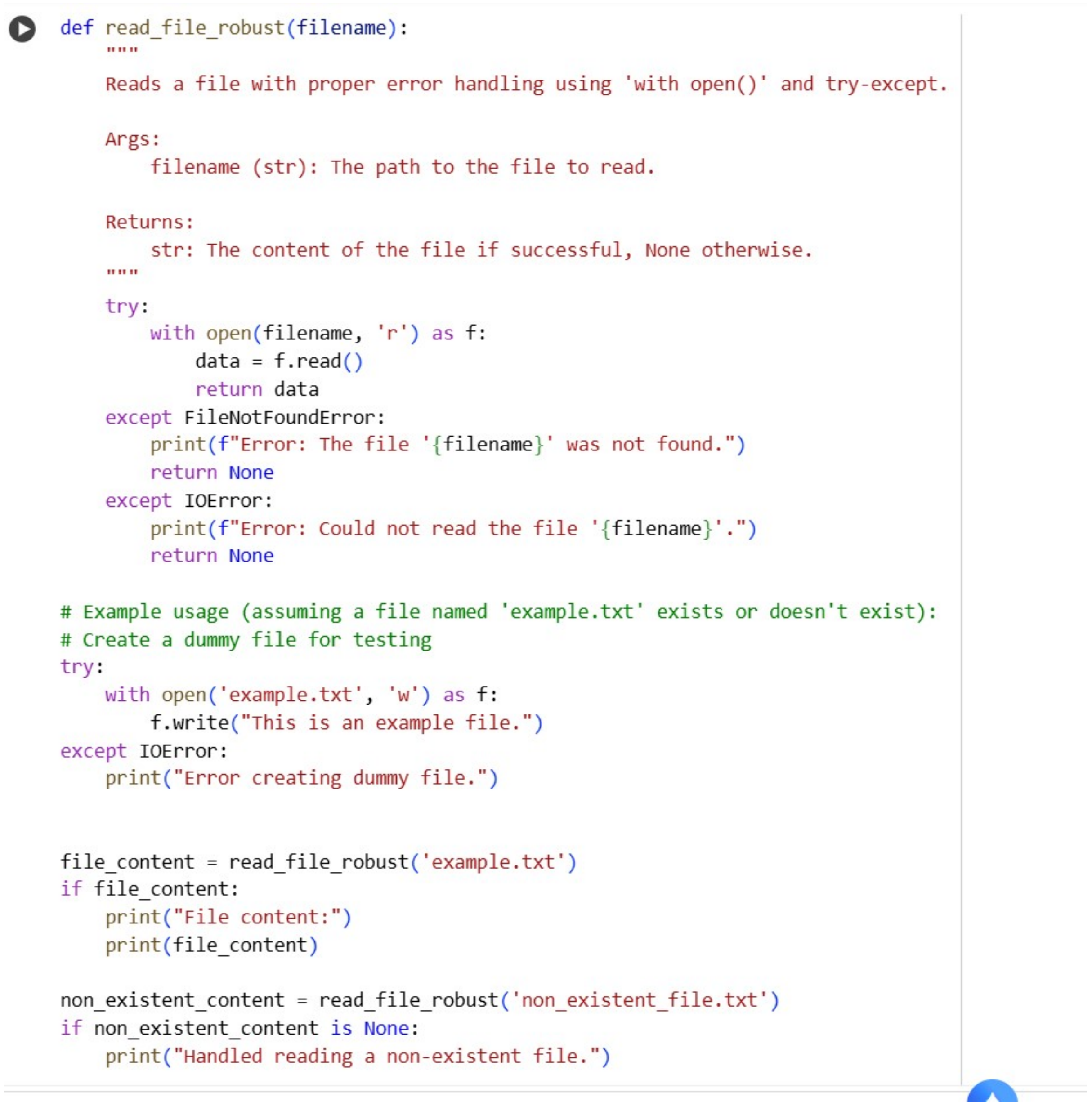
TASK-2 :

PROMPT :

Refactor this code with proper error handling using 'with open()' and try-except.

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

CODE :



```python
def read_file_robust(filename):
    """
    Reads a file with proper error handling using 'with open()' and try-except.

    Args:
        filename (str): The path to the file to read.

    Returns:
        str: The content of the file if successful, None otherwise.
    """
    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 IOError:
        print(f"Error: Could not read the file '{filename}'.")
        return None

# Example usage (assuming a file named 'example.txt' exists or doesn't exist):
# Create a dummy file for testing
try:
    with open('example.txt', 'w') as f:
        f.write("This is an example file.")
except IOError:
    print("Error creating dummy file.")


file_content = read_file_robust('example.txt')
if file_content:
    print("File content:")
    print(file_content)

non_existent_content = read_file_robust('non_existent_file.txt')
if non_existent_content is None:
    print("Handled reading a non-existent file.")
```

TASK-3:

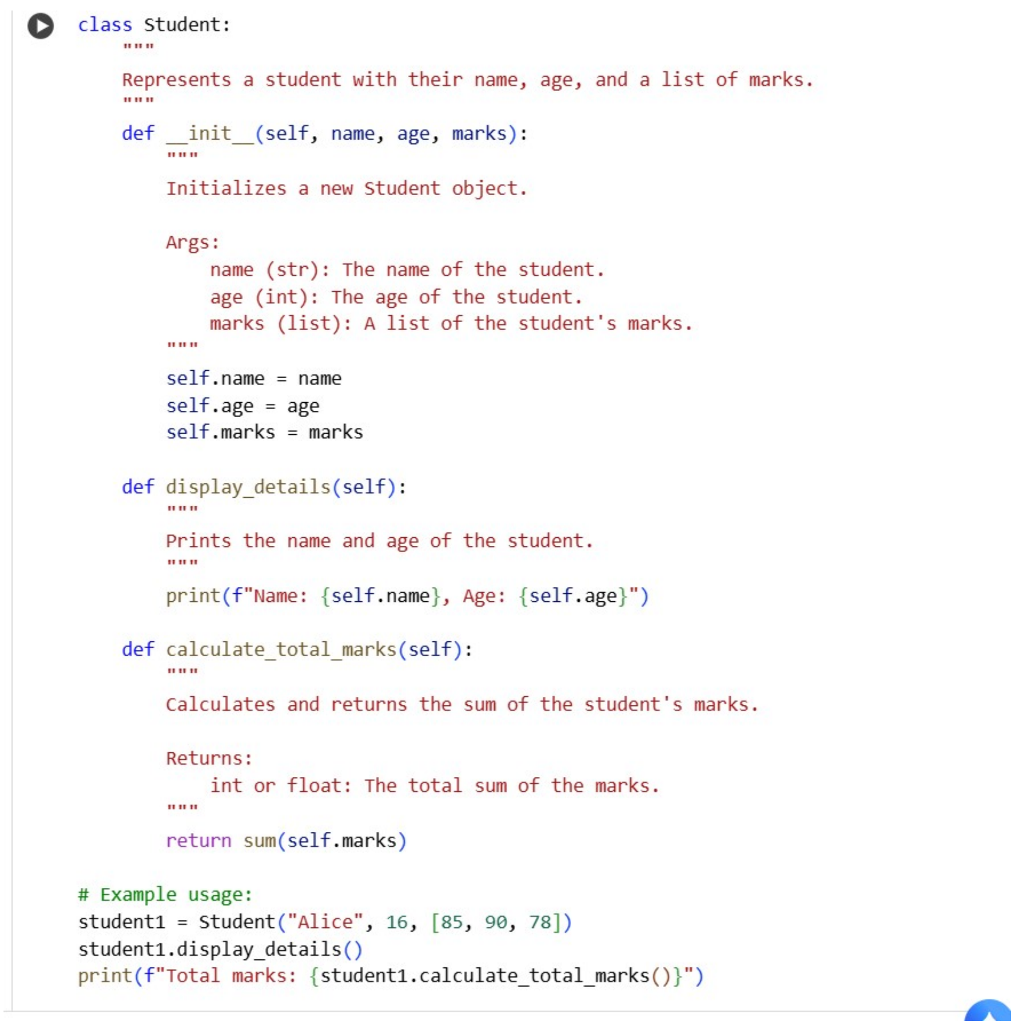
PROMPT:

Refactor this class to improve naming, readability, and modularity.

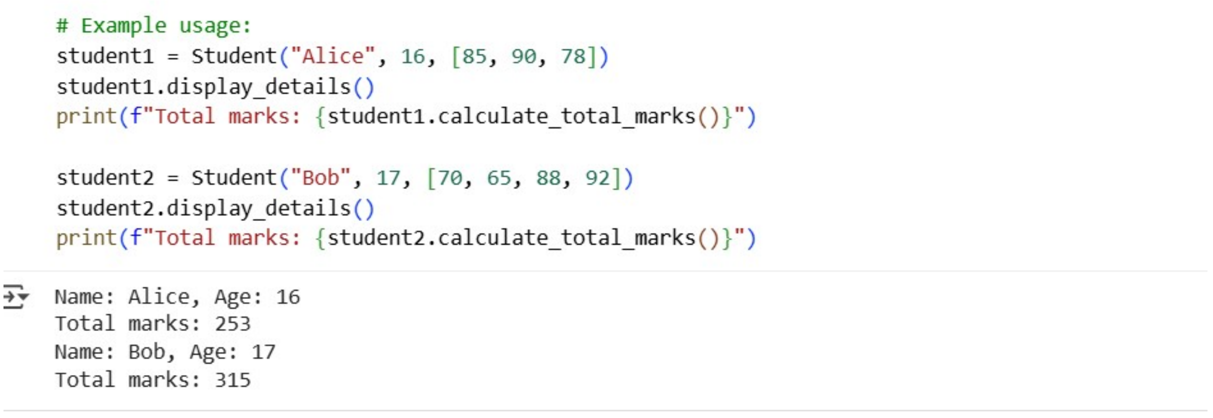
Add docstrings and make marks a list.

```
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
```

CODE:



```python
class Student:
    """
    Represents a student with their name, age, and a list of marks.
    """
    def __init__(self, name, age, marks):
        """
        Initializes a new Student object.

        Args:
            name (str): The name of the student.
            age (int): The age of the student.
            marks (list): A list of the student's marks.
        """
        self.name = name
        self.age = age
        self.marks = marks

    def display_details(self):
        """
        Prints the name and age of the student.
        """
        print(f"Name: {self.name}, Age: {self.age}")

    def calculate_total_marks(self):
        """
        Calculates and returns the sum of the student's marks.

        Returns:
            int or float: The total sum of the marks.
        """
        return sum(self.marks)

# Example usage:
student1 = Student("Alice", 16, [85, 90, 78])
student1.display_details()
print(f"Total marks: {student1.calculate_total_marks()}")
```



```python
# Example usage:
student1 = Student("Alice", 16, [85, 90, 78])
student1.display_details()
print(f"Total marks: {student1.calculate_total_marks()}")

student2 = Student("Bob", 17, [70, 65, 88, 92])
student2.display_details()
print(f"Total marks: {student2.calculate_total_marks()}")
```

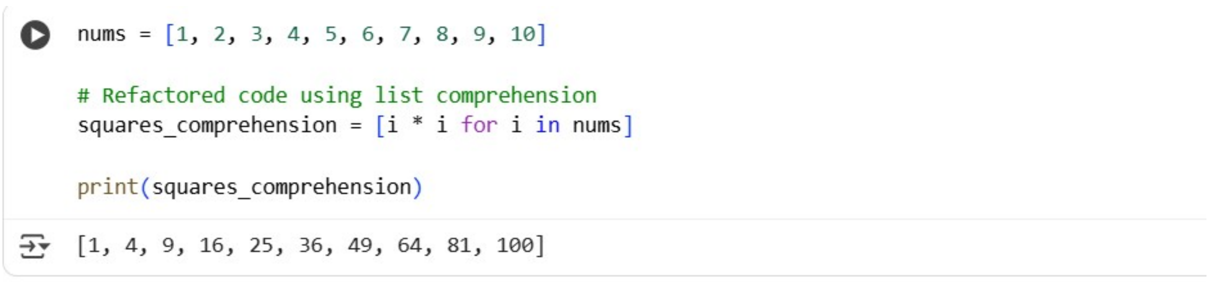
```
Name: Alice, Age: 16
Total marks: 253
Name: Bob, Age: 17
Total marks: 315
```

TASK-4 :

PROMPT :

# Refactor this code using list comprehension for better efficiency. nums = [1,2,3,4,5,6,7,8,9,10] squares = [] for i in nums: squares.append(i * i).

CODE :



```
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Refactored code using list comprehension
squares_comprehension = [i * i for i in nums]

print(squares_comprehension)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```