**AI ASSISTED CODING**

Name: Cholluri Srikruthi

Ht no: 2403A510H0

Lab-Assignment: 13.3

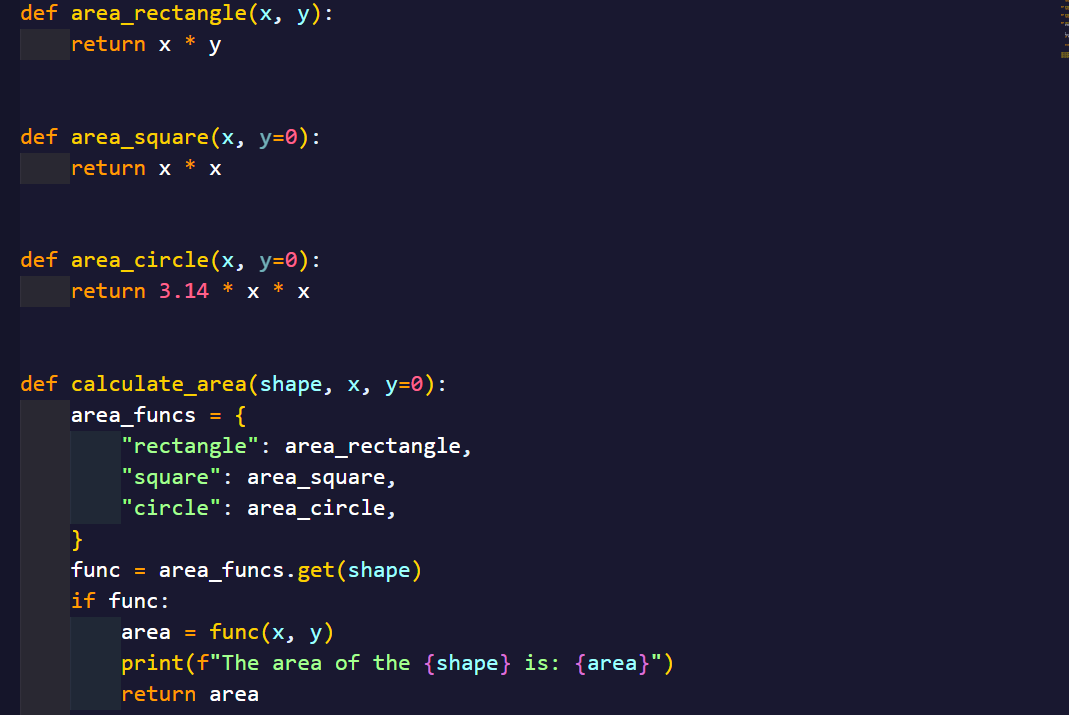
Batch: 06

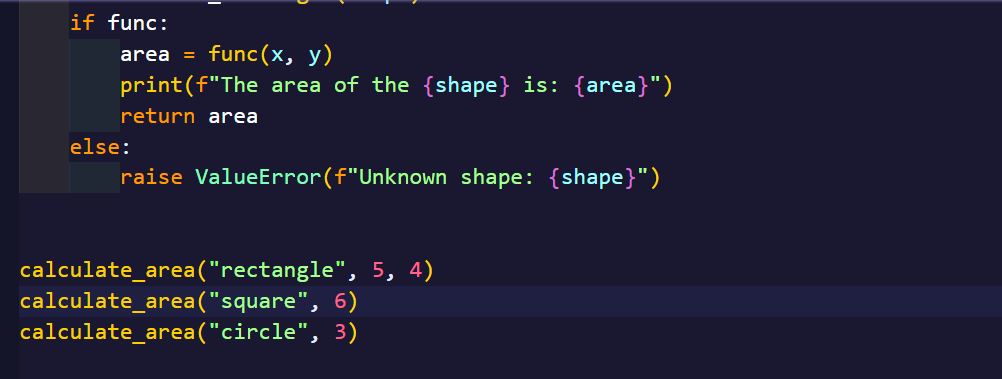
**Task-1:**

**Prompt:**

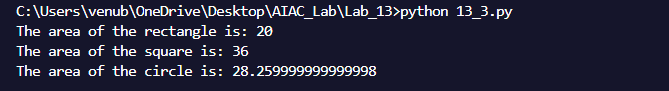
Refactor the Python function to use a dictionary-based dispatch or separate modular functions instead of repeated if-elif blocks. The goal is to improve readability and reduce redundancy.

**Code:**

****

****

**Output:**

****

**Observation:**

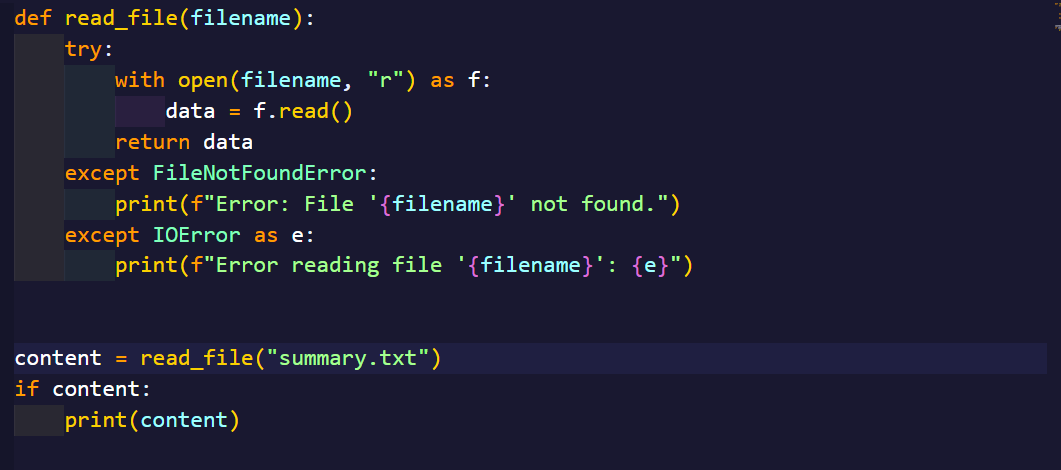
The original calculate\_area function used repetitive if-elif statements, making it harder to maintain. The refactored version improves this by using a dictionary to map shapes to their respective functions, resulting in cleaner, more modular, and scalable code. This approach makes it easier to add new shapes without modifying the core logic.

**Task-2:**

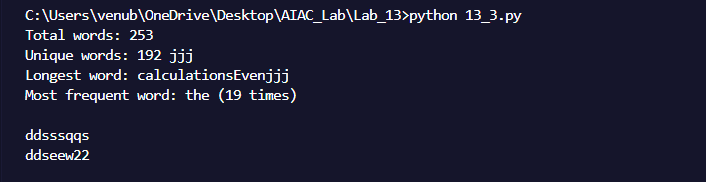
**Prompt:**

Refactor the function to use Python with open() context manager and add proper error handling using try-except blocks. Ensure the function handles cases like missing files or read errors gracefully.

**Code:**

****

**Output:**

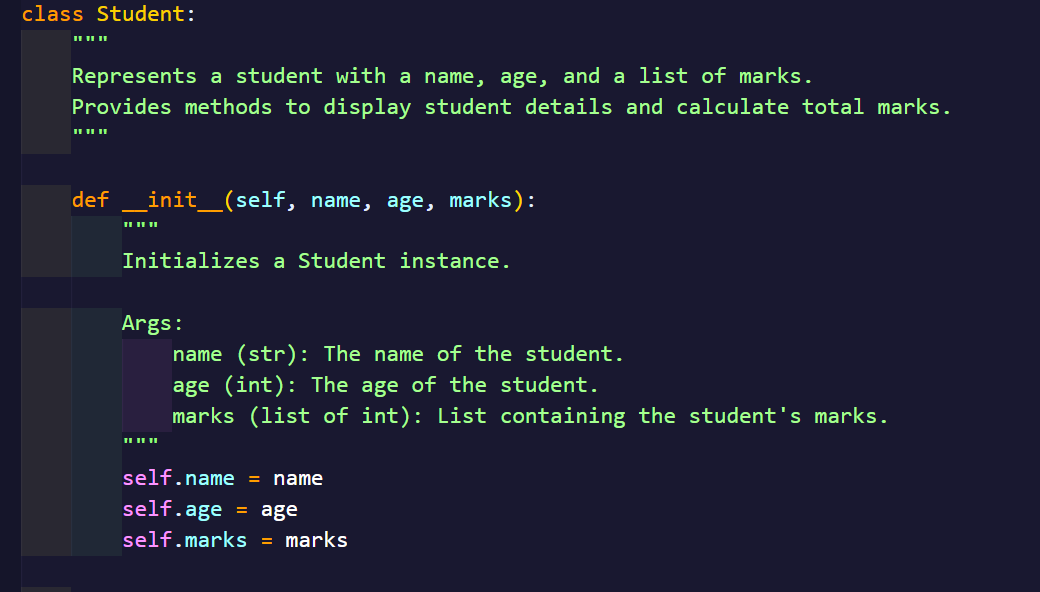
****

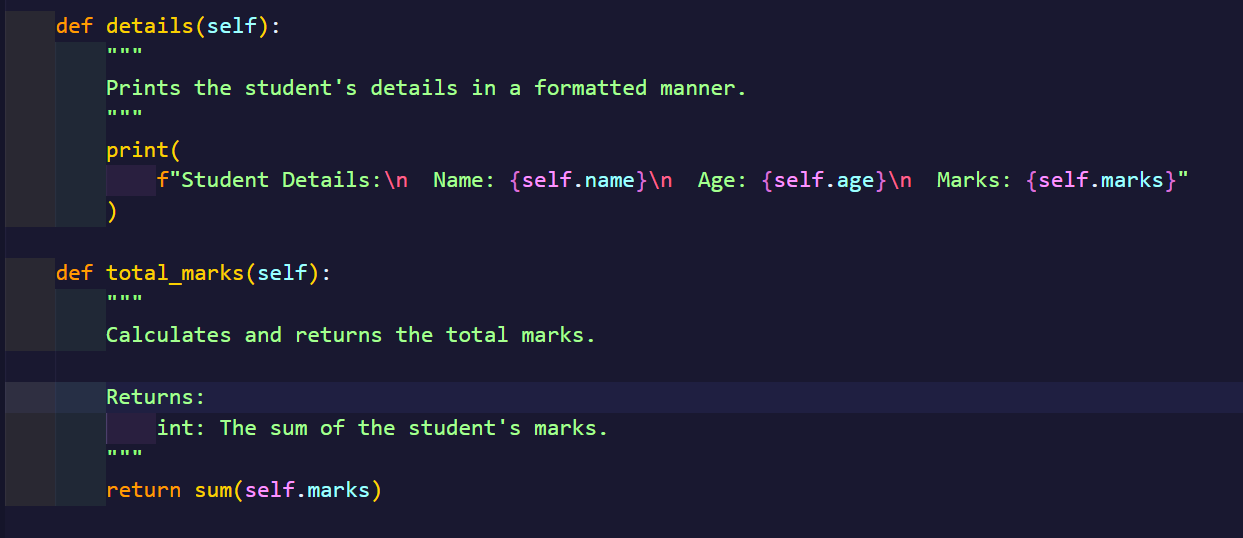
**Task-3:**

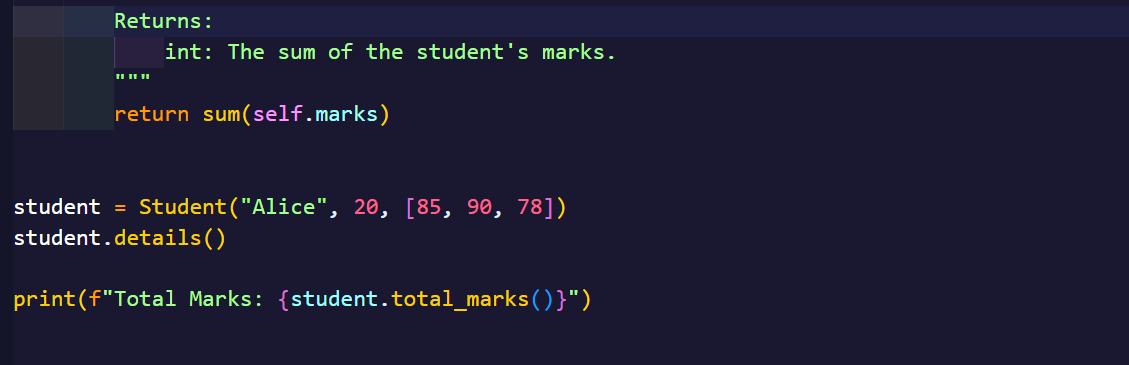
**Prompt:**

Refactor the Student class to improve readability and modularity. Rename variables to more descriptive names, store marks in a list instead of separate attributes, add appropriate docstrings for the class and methods, improve the print output formatting in the details method, and use Python built-in functions like sum() where applicable.

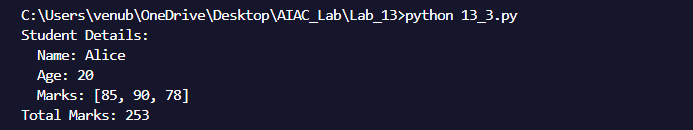
**Code:**

****

****

****

**Output:**

****

**Observation:**

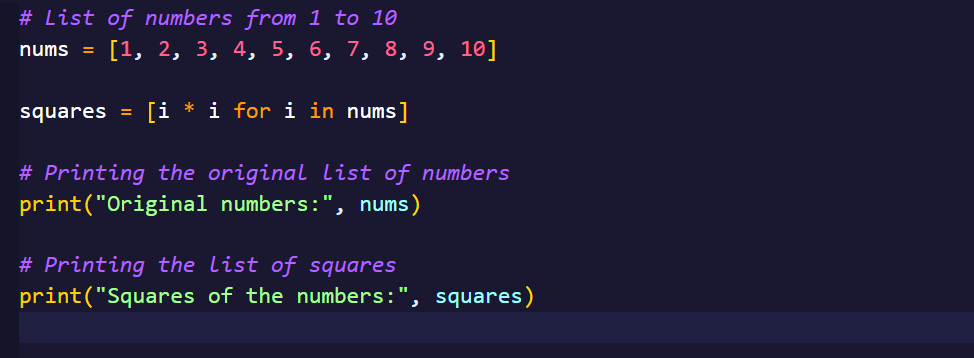
The refactored Student class uses clear names and stores marks in a list for flexibility. Docstrings improve understanding, and the details method prints info neatly. Using sum() simplifies total marks calculation. Overall, the code is cleaner and easier to maintain.

**Task-4:**

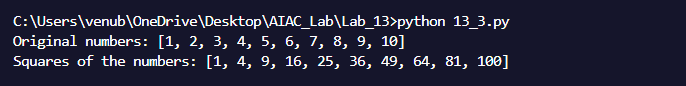
**Prompt:**

Refactor the code that uses a loop to create a list of squares, making it more efficient and Pythonic by using list comprehensions or other suitable techniques.

**Code:**

****

**Output:**

****

**Observation:**

The initial approach using a loop and appending each square to a list is functional but somewhat inefficient and verbose. By switching to a list comprehension, the code becomes more elegant and compact, capturing the entire transformation in one clear expression. The use of print statements to display both the original numbers and their squares adds clarity and helps with debugging. Overall, this refactoring results in cleaner, faster, and more maintainable code that aligns well with Pythonic coding standards.