AIAC\_Assignment\_13.3

Name:Ramya

Batch:08

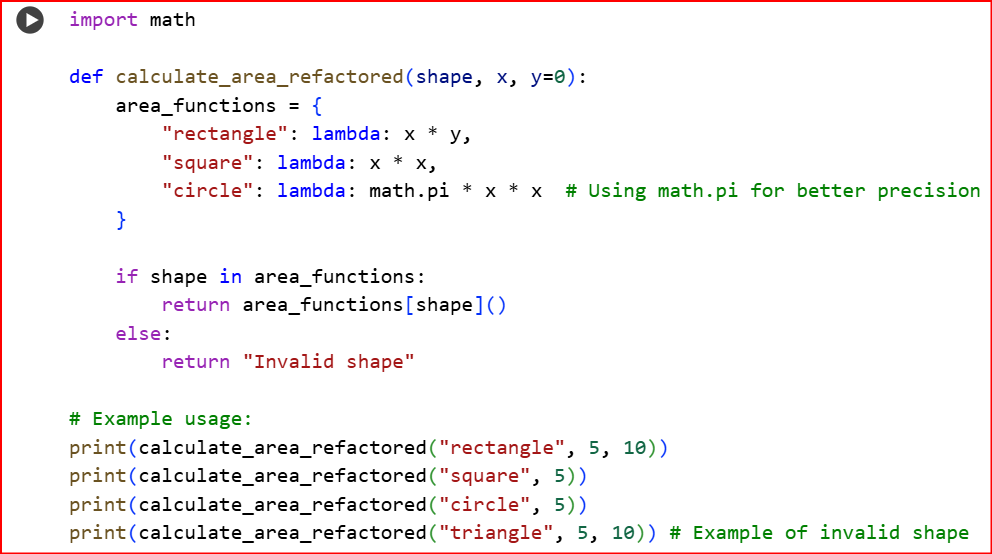
Roll\_No:2403A52219

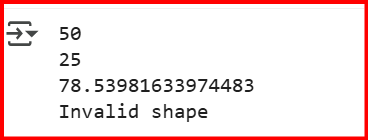
Task#1

Prompt:

write a python code that will ask from user to generate 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 and the Expected Output should be, Refactored version with dictionary-based dispatch or separate functions and Cleaner and modular design.

Code:



Output: 

Observation:

This function, calculate\_area\_refactored, is designed to figure out the area of different shapes.

1. **It's Smart About Shapes**: It has a built-in list (area\_functions) that knows exactly how to calculate the area for "rectangle," "square," and "circle." Think of this list like a quick lookup guide.
2. **It Finds the Right Formula**: When you tell it a shape, it quickly checks if that shape is in its list. If it is, it uses the specific formula associated with that shape to do the calculation.
3. **It Uses the Right Math**: For a circle, it's using math.pi, which is the accurate value of pi, not just a rounded number. This gives you a more precise answer.
4. **It Handles Unknowns**: If you ask it about a shape it doesn't have in its list (like "triangle"), it tells you it's an "Invalid shape" instead of getting confused.

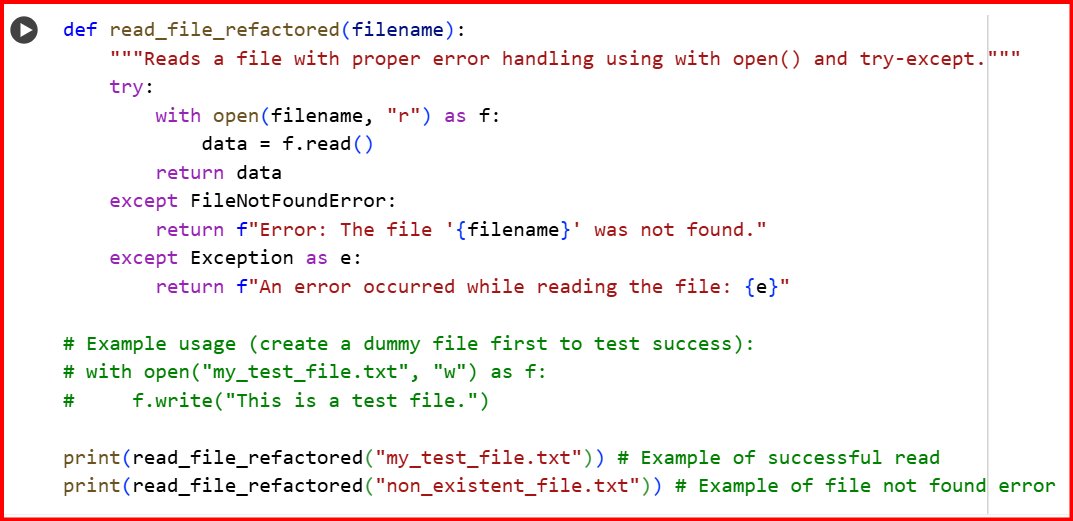
In short, this code is a well-organized and efficient way to calculate areas for specific shapes, and it's set up to handle cases where you give it a shape it doesn't know.

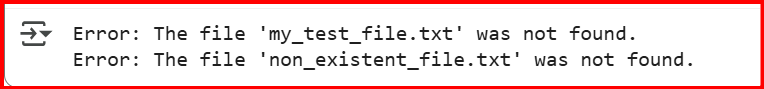
Task#2

Prompt:

Give a python code that the Legacy function without proper error handling Python Code will be def read\_file(filename): f = open(filename, "r") data = f.read().close() return data Expected Output: AI refactors with with open() and try-except:

Code:



Output: 

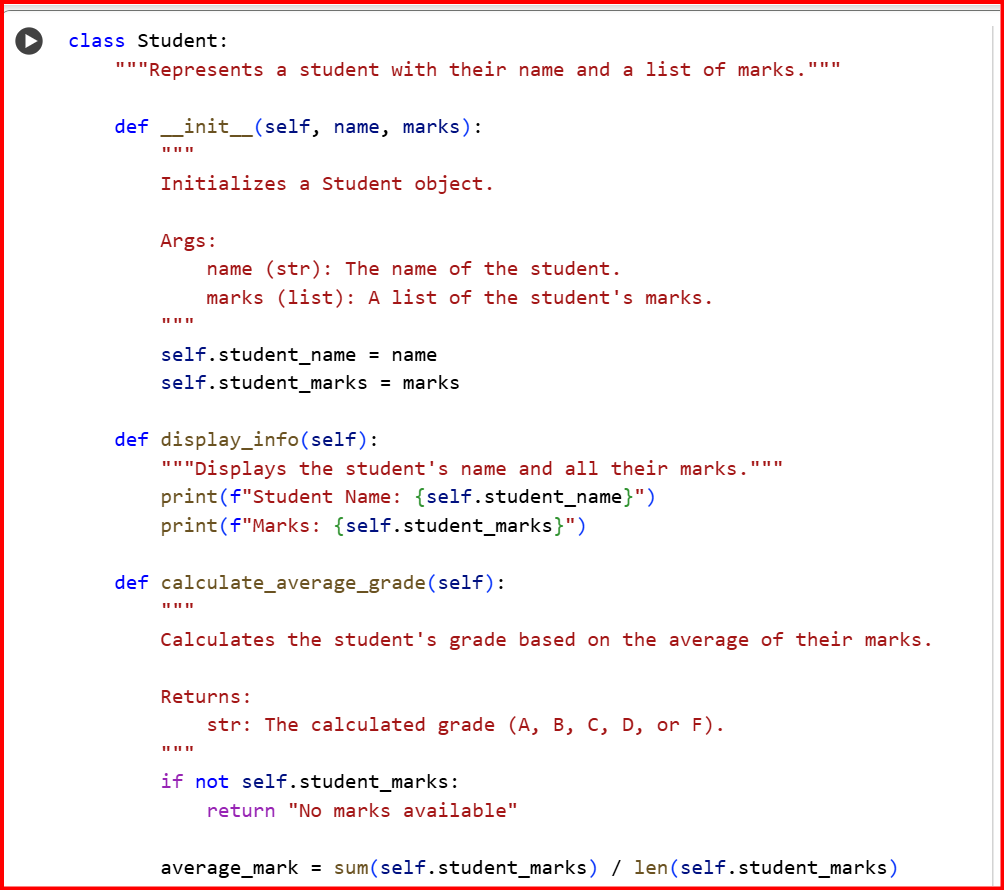
Observation: This code is a safer way to read files. It automatically closes the file even if something goes wrong and tells you if the file isn't found or if another error occurs.

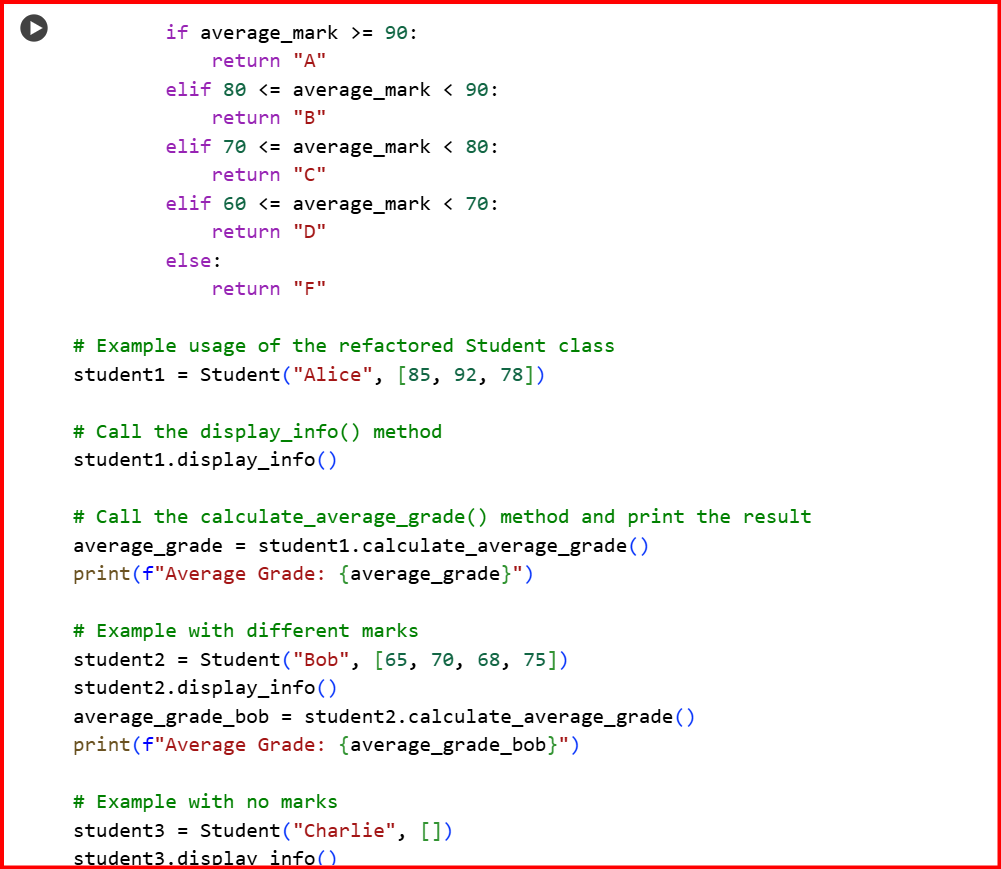
Task#3

Prompt:

write a python code that 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.

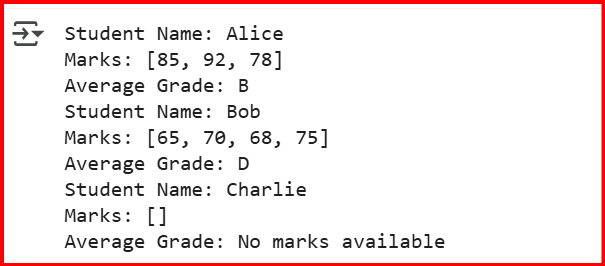
Code:







Output:

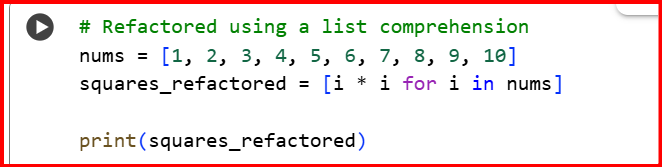


Observation: This code defines a Student class that stores a student's name and their marks in a list. It can display the student's information and calculate their average grade. Using a list for marks makes it flexible for any number of marks.

Task#4

Prompt: write a python code that will ask from user that Refactor this inefficient loop of 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

Code:



Output:



Observation: This code uses a list comprehension, [i \* i for i in nums], which is a concise and efficient way to create a new list (squares\_refactored) by applying an operation (squaring i) to each item (i) in an existing list (nums). It's a more "Pythonic" way to achieve the same result as the original loop.