AI ASSISTED CODING

ASSIGNMENT-13.3

Name: G Hasini

Roll no.: 2503A51L41

Batch: 24BTCAICSB20

Lab 13 – Code Refactoring: Improving Legacy Code with AI Suggestions

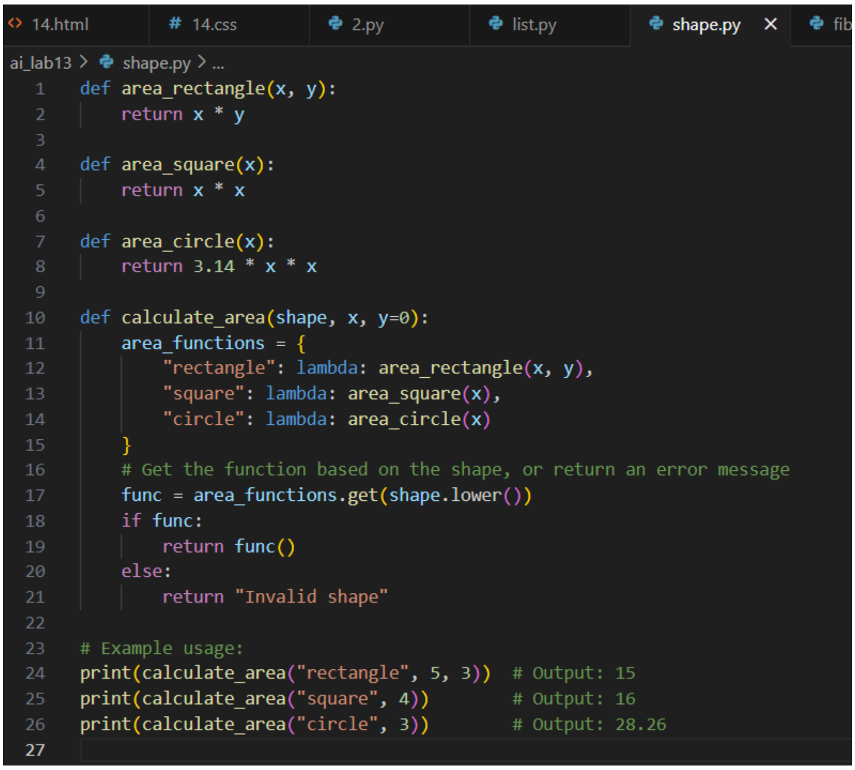
Task #1

Prompt: Provide AI with the following redundant code and ask it to refactor.

Python Code

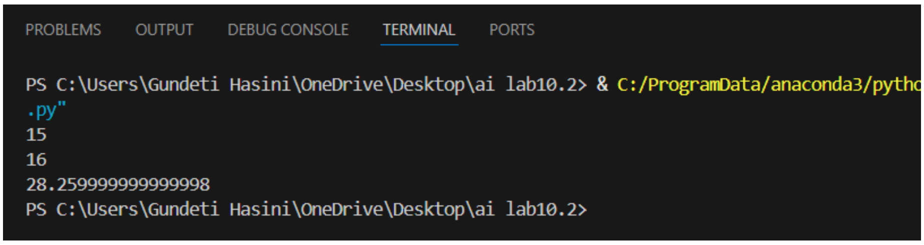
```
def calculate_area(shape, x, y=0):
    if shape == "rectangle":
        return x * y
    elif shape == "square":
        return x * x
    elif shape == "circle":
        return 3.14 * x * x
```

After refactoring, **Code generated:**



```python
def area_rectangle(x, y):
    return x * y

def area_square(x):
    return x * x

def area_circle(x):
    return 3.14 * x * x

def calculate_area(shape, x, y=0):
    area_functions = {
        "rectangle": lambda: area_rectangle(x, y),
        "square": lambda: area_square(x),
        "circle": lambda: area_circle(x)
    }
    # Get the function based on the shape, or return an error message
    func = area_functions.get(shape.lower())
    if func:
        return func()
    else:
        return "Invalid shape"

# Example usage:
print(calculate_area("rectangle", 5, 3))  # Output: 15
print(calculate_area("square", 4))        # Output: 16
print(calculate_area("circle", 3))        # Output: 28.26
```

Output:



```
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> & C:/ProgramData/anaconda3/pytho
.py"
15
16
28.259999999999998
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2>
```

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.

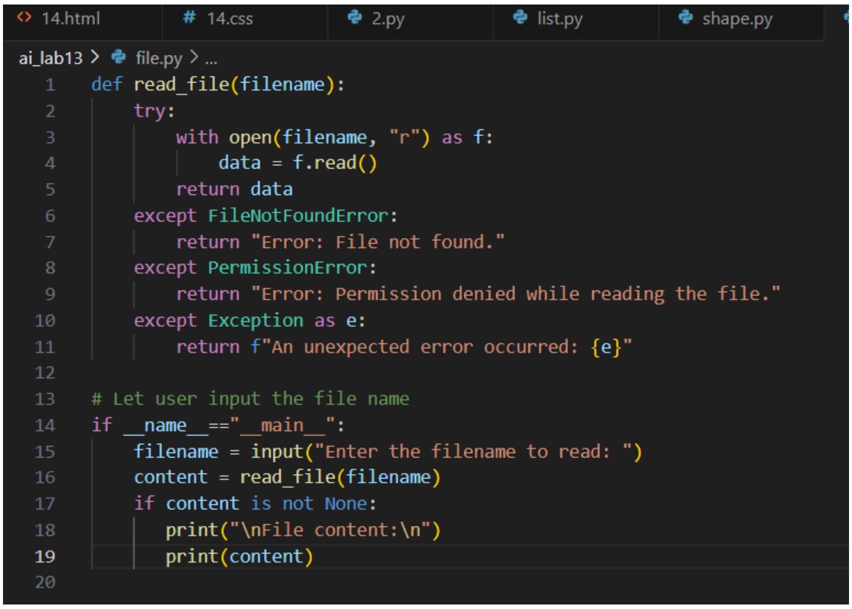
Task #2 :

Prompt: Provide AI with the following redundant code and ask it to refactor. with open() and try-except:

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

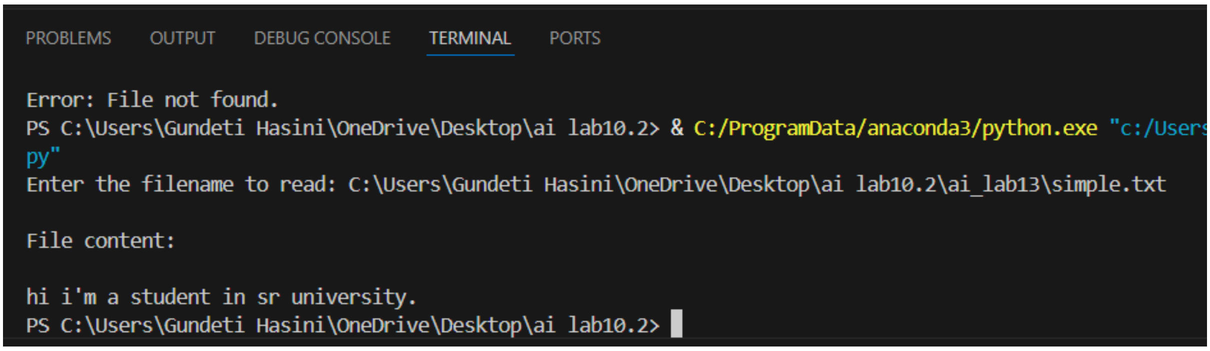
f.close()
return data

After refactoring **code generated:**



```python
def read_file(filename):
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        return "Error: File not found."
    except PermissionError:
        return "Error: Permission denied while reading the file."
    except Exception as e:
        return f"An unexpected error occurred: {e}"

# Let user input the file name
if __name__=="__main__":
    filename = input("Enter the filename to read: ")
    content = read_file(filename)
    if content is not None:
       print("\nFile content:\n")
       print(content)
```

Output:



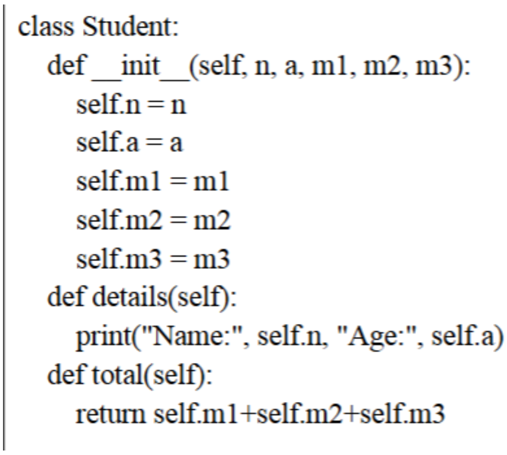
```
Error: File not found.
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> & C:/ProgramData/anaconda3/python.exe "c:/User
py"
Enter the filename to read: C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2\ai_lab13\simple.txt

File content:

hi i'm a student in sr university.
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2>
```

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 #3

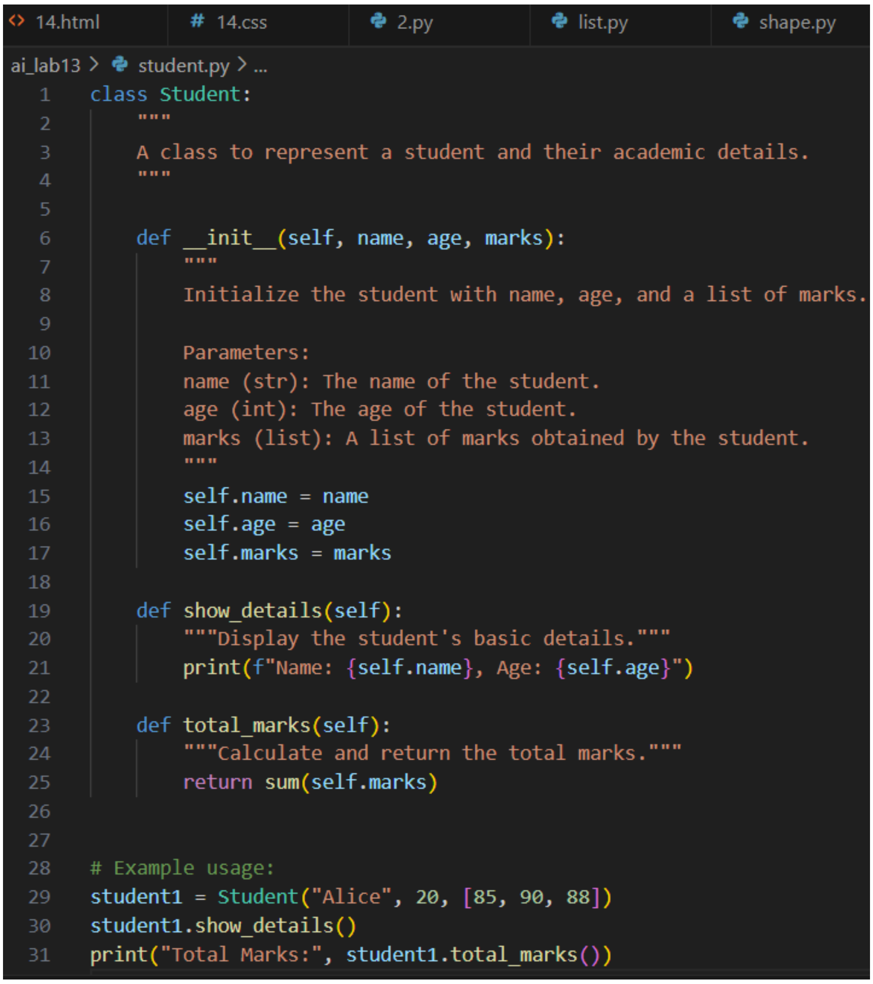
Prompt: Provide this legacy class to AI for readability and modularity improvements.



```
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
```

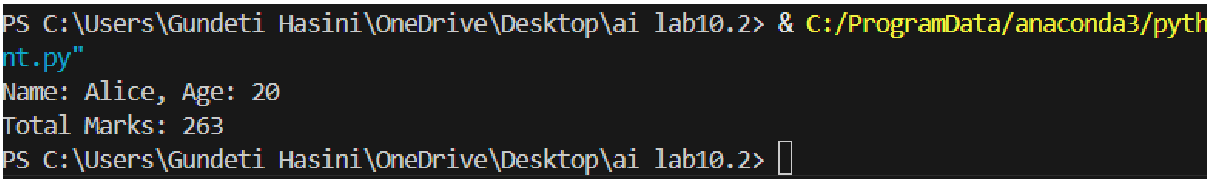
After refactoring,

Code generated:



```python
class Student:
    """
    A class to represent a student and their academic details.
    """

    def __init__(self, name, age, marks):
        """
        Initialize the student with name, age, and a list of marks.

        Parameters:
        name (str): The name of the student.
        age (int): The age of the student.
        marks (list): A list of marks obtained by the student.
        """
        self.name = name
        self.age = age
        self.marks = marks

    def show_details(self):
        """Display the student's basic details."""
        print(f"Name: {self.name}, Age: {self.age}")

    def total_marks(self):
        """Calculate and return the total marks."""
        return sum(self.marks)


# Example usage:
student1 = Student("Alice", 20, [85, 90, 88])
student1.show_details()
print("Total Marks:", student1.total_marks())
```

Output:



```
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> & C:/ProgramData/anaconda3/pyth
nt.py"
Name: Alice, Age: 20
Total Marks: 263
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> 
```

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 #4:

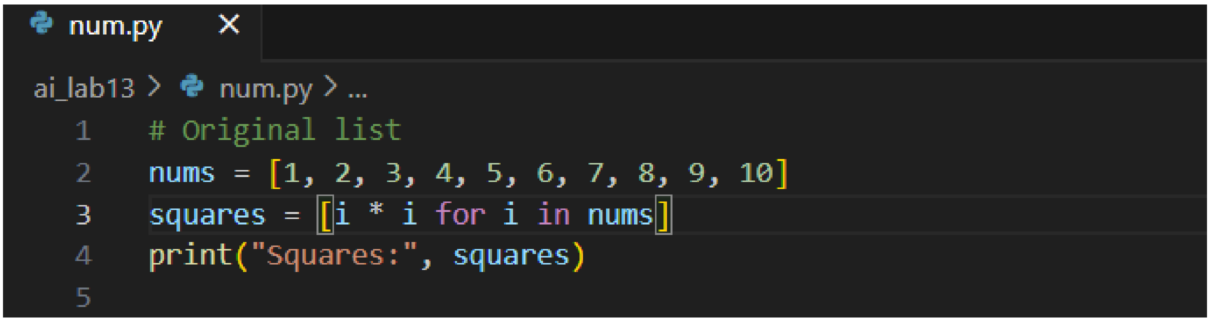
Prompt: Refactor this inefficient loop with AI help.

Python Code

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

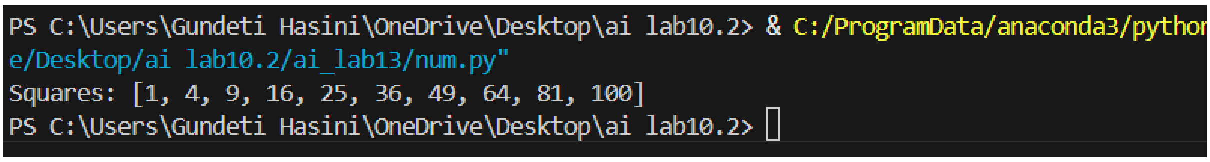
After refactoring,

Code generated:



```python
# Original list
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = [i * i for i in nums]
print("Squares:", squares)

```

Output:



```
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> & C:/ProgramData/anaconda3/python
e/Desktop/ai lab10.2/ai_lab13/num.py"
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\Gundeti Hasini\OneDrive\Desktop\ai lab10.2> 
```

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.