ASSIGNMENT-13.3

Name: D. Sravika Reddy H-T No: 2403a510d0

Batch No: 05 Course: AI Assisted Coding

✓ Task Description #1 – Remove Repetition

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

Python Code

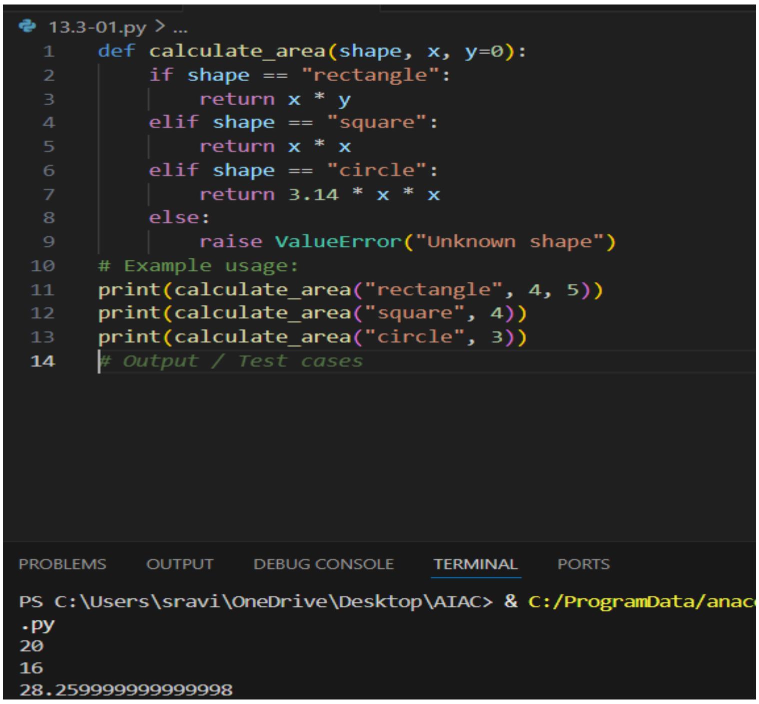
```
def calculate_area(shape, x, y=0):
if shape == "rectangle":
return x * y
elif shape == "square":
return x * x
elif shape == "circle":
return 3.14 * x * x
```

Expected Output

- Refactored version with dictionary-based dispatch or separate functions.
- Cleaner and modular design

✓ #Prompt: Refactor the following redundant code to remove repetition. Use a dictionary-based dispatch or separate functions for each shape to make the design cleaner and more modular.

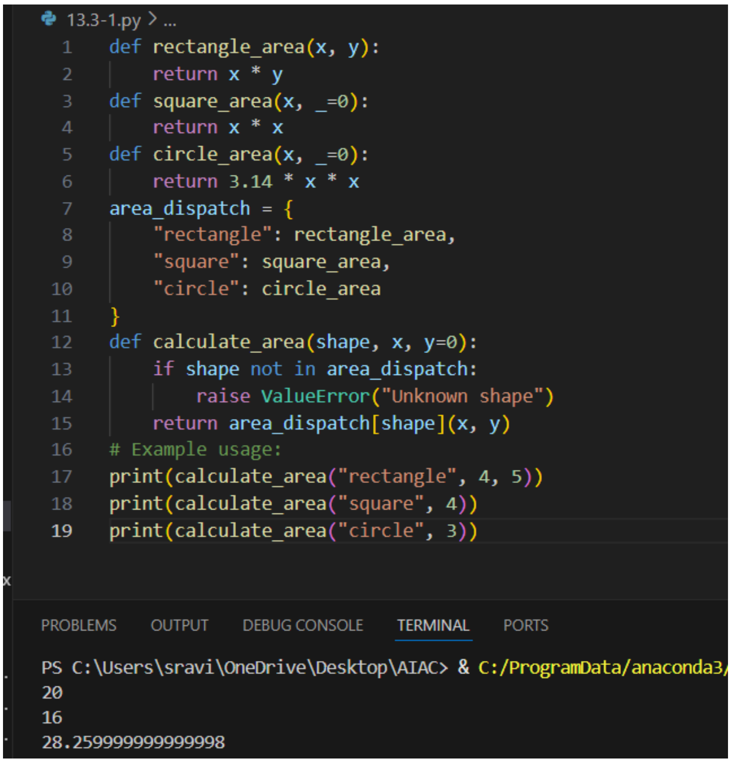
✓ Code & Output:



```python
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
    else:
        raise ValueError("Unknown shape")
# Example usage:
print(calculate_area("rectangle", 4, 5))
print(calculate_area("square", 4))
print(calculate_area("circle", 3))
# Output / Test cases
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anac
.py
20
16
28.259999999999998
```

✓ Refactored Code & Output:



```python
def rectangle_area(x, y):
    return x * y
def square_area(x, _=0):
    return x * x
def circle_area(x, _=0):
    return 3.14 * x * x
area_dispatch = {
    "rectangle": rectangle_area,
    "square": square_area,
    "circle": circle_area
}
def calculate_area(shape, x, y=0):
    if shape not in area_dispatch:
        raise ValueError("Unknown shape")
    return area_dispatch[shape](x, y)
# Example usage:
print(calculate_area("rectangle", 4, 5))
print(calculate_area("square", 4))
print(calculate_area("circle", 3))
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/
20
16
28.259999999999998
```

✓ Explanation for the Original Code:

The function calculate_area computes the area for three different shapes based on the shape argument:

- If the shape is "rectangle", it multiplies x and y to get the area.
- If the shape is "square", it returns x * x (area of a square).
- If the shape is "circle", it returns 3.14 * x * x (area of a circle with radius x).
- If the shape is not recognized, it raises a ValueError.

Example usage:

- calculate_area("rectangle", 4, 5) returns 20
- calculate_area("square", 4) returns 16
- calculate_area("circle", 3) returns 28.26

✓ Explanation for the Refactored Code (from previous answer):

The refactored version separates the area calculation for each shape into its own function and uses a dictionary (area_dispatch) to map shape names to these functions. The main calculate_area function looks up the appropriate function and calls it.

Benefits of the refactored approach:

- **Modularity:** Each shape's logic is isolated, making the code easier to maintain and extend.
- **Readability:** The main function is cleaner and easier to understand.
- **Extensibility:** Adding new shapes only requires defining a new function and updating the dictionary.

Both versions correctly calculate areas, but the refactored version is more maintainable and scalable for larger applications.

✓ Task Description #2 – Error Handling in Legacy Code

Task: Legacy function without proper error handling

Python Code

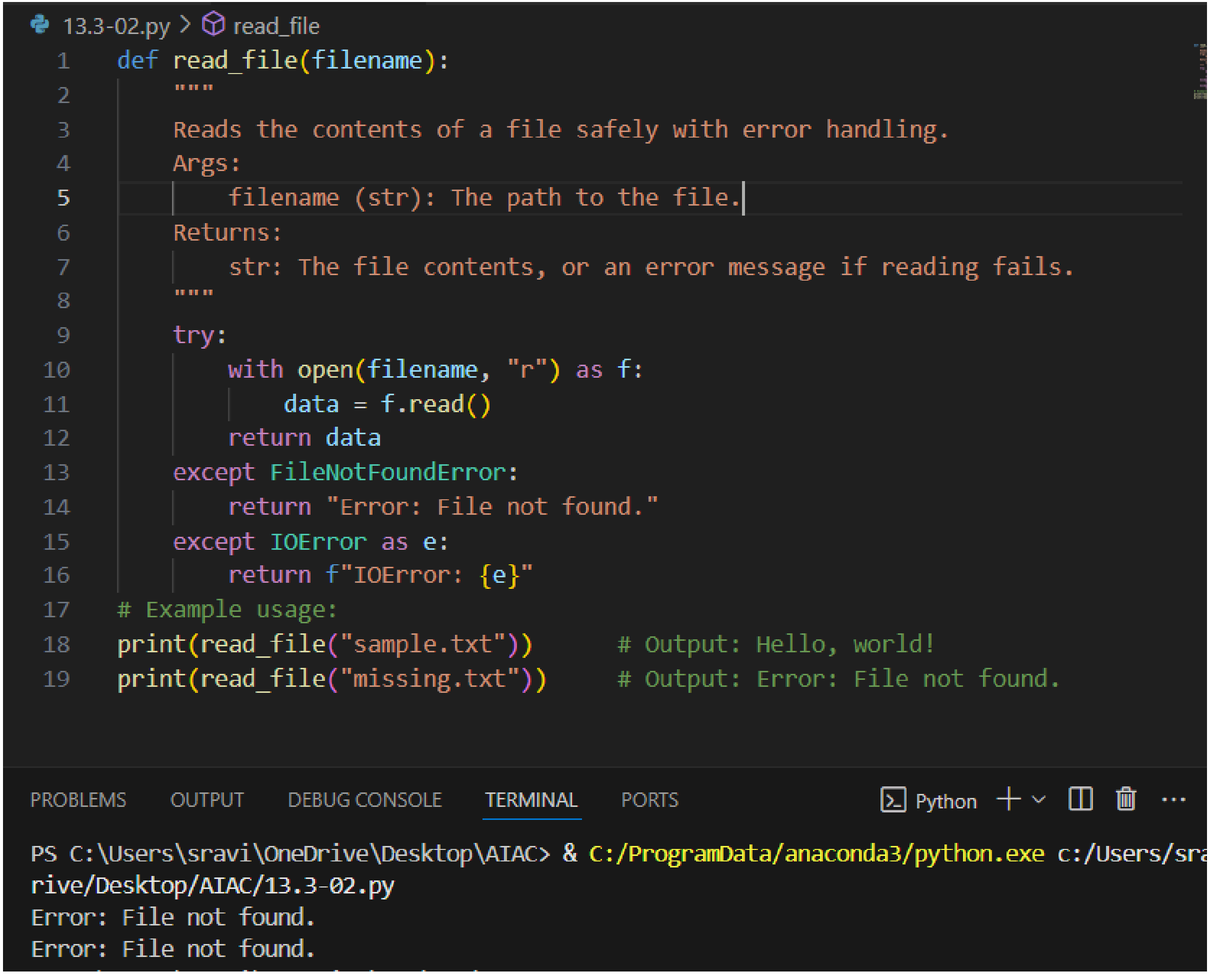
```
def read_file(filename):
f = open(filename, "r")
data = f.read()
f.close()
return data
```

Expected Output:

AI refactors with with open() and try-except:

✓ #Prompt: Refactor the following legacy function to include proper error handling and resource management using with open() and a try-except block.

✓ Code & Output:



```python
def read_file(filename):
    """
    Reads the contents of a file safely with error handling.
    Args:
        filename (str): The path to the file.
    Returns:
        str: The file contents, or an error message if reading fails.
    """
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        return "Error: File not found."
    except IOError as e:
        return f"IOError: {e}"
# Example usage:
print(read_file("sample.txt"))      # Output: Hello, world!
print(read_file("missing.txt"))     # Output: Error: File not found.
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sra
rive/Desktop/AIAC/13.3-02.py
Error: File not found.
Error: File not found.
```

✓ Explanation:

- Uses with open() for automatic file closing, even if an error occurs.
- Handles FileNotFoundError and general IOError with user-friendly messages.
- Returns the file contents if successful, or an error message otherwise.
- This approach is safer and more robust than the legacy code

✓ Task Description #3 – Complex Refactoring

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

Python Code

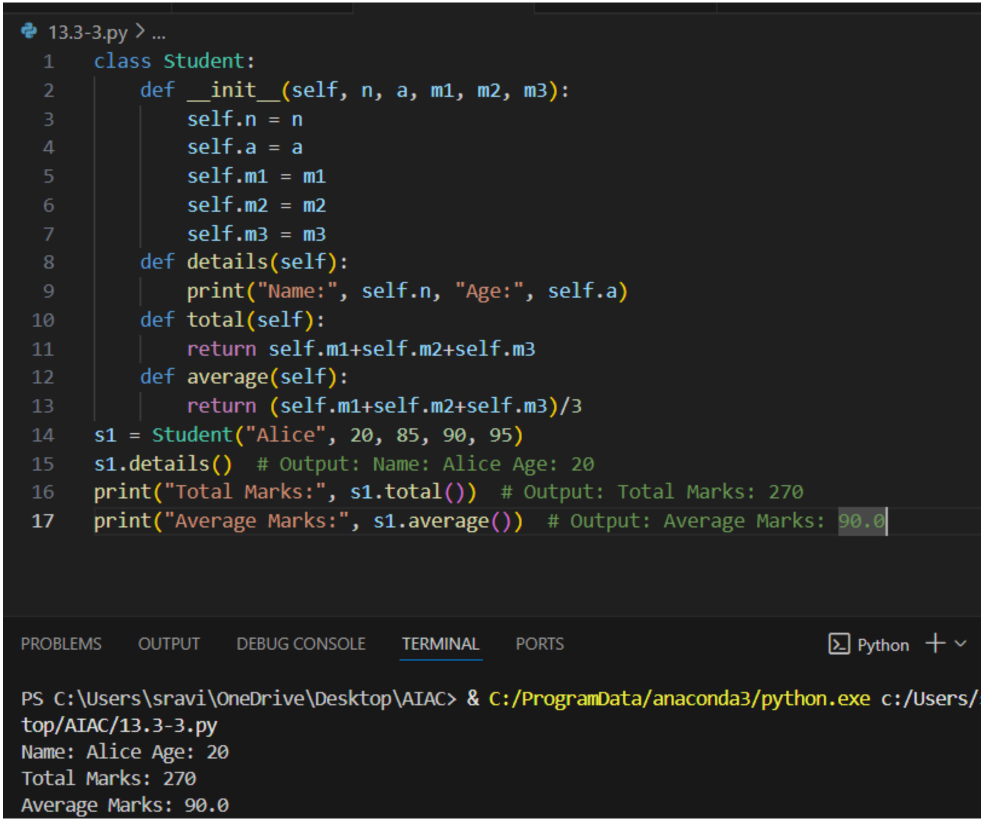
```
class Student:
def __init__(self, n, a, m1, m2, m3):
self.n = n
self.a = a
self.m1 = m1
self.m2 = m2
self.m3 = m3
def details(self):
print("Name:", self.n, "Age:", self.a)
def total(self):
return self.m1+self.m2+self.m3
```

Expected Output:

- AI improves naming (name, age, marks).
- Adds docstrings.
- Improves print readability.
- Possibly uses sum(self.marks) if marks stored in a list.

✓ #Prompt: Refactor the following legacy Student class to improve readability and modularity. Use descriptive variable names, add docstrings, improve print formatting, and consider storing marks in a list to simplify calculations.

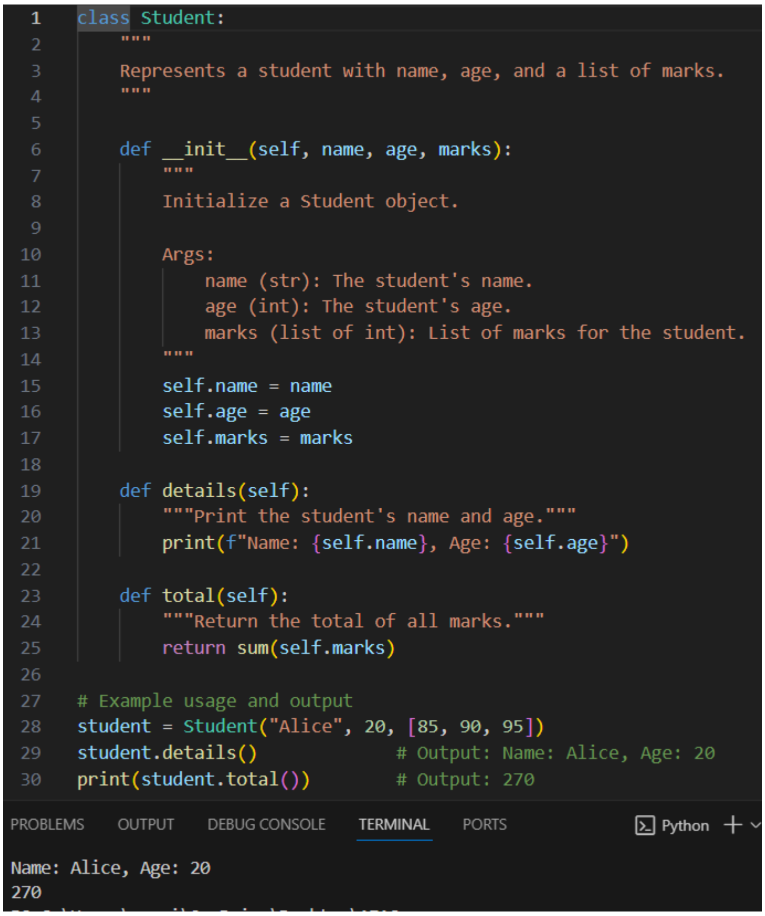
✓ Code & Output:



```python
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
    def average(self):
        return (self.m1+self.m2+self.m3)/3
s1 = Student("Alice", 20, 85, 90, 95)
s1.details()  # Output: Name: Alice Age: 20
print("Total Marks:", s1.total())  # Output: Total Marks: 270
print("Average Marks:", s1.average())  # Output: Average Marks: 90.0
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/
top/AIAC/13.3-3.py
Name: Alice Age: 20
Total Marks: 270
Average Marks: 90.0
```

✓ **Refactored Code & Output:**



```python
class Student:
    """
    Represents a student with name, age, and a list of marks.
    """

    def __init__(self, name, age, marks):
        """
        Initialize a Student object.

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

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

    def total(self):
        """Return the total of all marks."""
        return sum(self.marks)

# Example usage and output
student = Student("Alice", 20, [85, 90, 95])
student.details()             # Output: Name: Alice, Age: 20
print(student.total())        # Output: 270
```

```
Name: Alice, Age: 20
270
```

Explanation:

- Variable names are now descriptive (name, age, marks).
- Marks are stored as a list, allowing easy calculation with sum(self.marks).
- Docstrings are added for the class and methods.
- The details method uses f-strings for clearer output.
- The code is now more modular, readable, and easier to maintain or extend.

Task Description #4 – Inefficient Loop Refactoring

Task: Refactor this inefficient loop with AI help

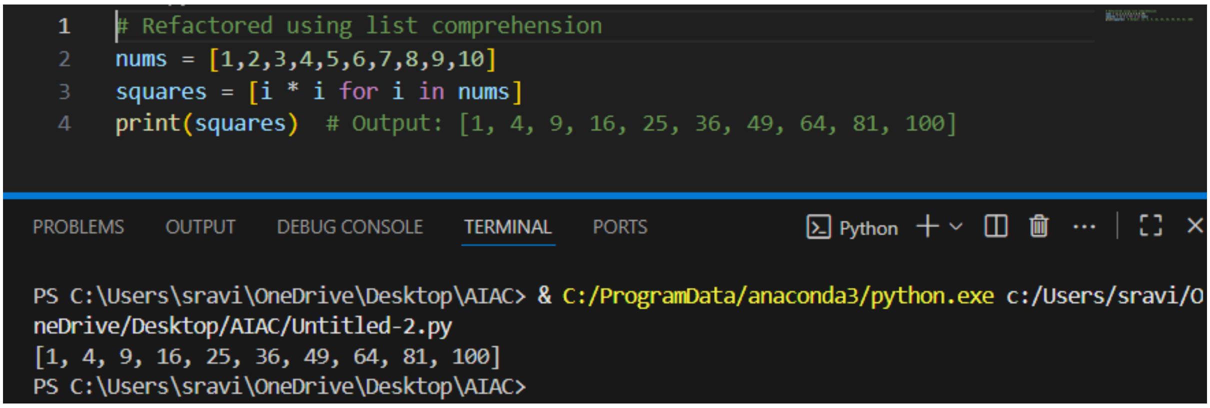
Python Code

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

Expected Output: AI suggested a list comprehension

#Prompt: Refactor this inefficient loop with AI help

Code & Output:



```
# Refactored using list comprehension
nums = [1,2,3,4,5,6,7,8,9,10]
squares = [i * i for i in nums]
print(squares)  # Output: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
PS C:\Users\sravi\OneDrive\Desktop\AIAC> & C:/ProgramData/anaconda3/python.exe c:/Users/sravi/O
neDrive/Desktop/AIAC/Untitled-2.py
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\sravi\OneDrive\Desktop\AIAC>
```

Explanation:

- The original loop is replaced with a list comprehension, which is more concise and efficient in Python.
- This approach creates the squares list in a single line, improving readability and performance for large lists.