Lab 13.3 – Code Refactoring with AI

Name:N.Prudhvi

HT.NO:2403a510g7

Task 1 – Remove Repetition

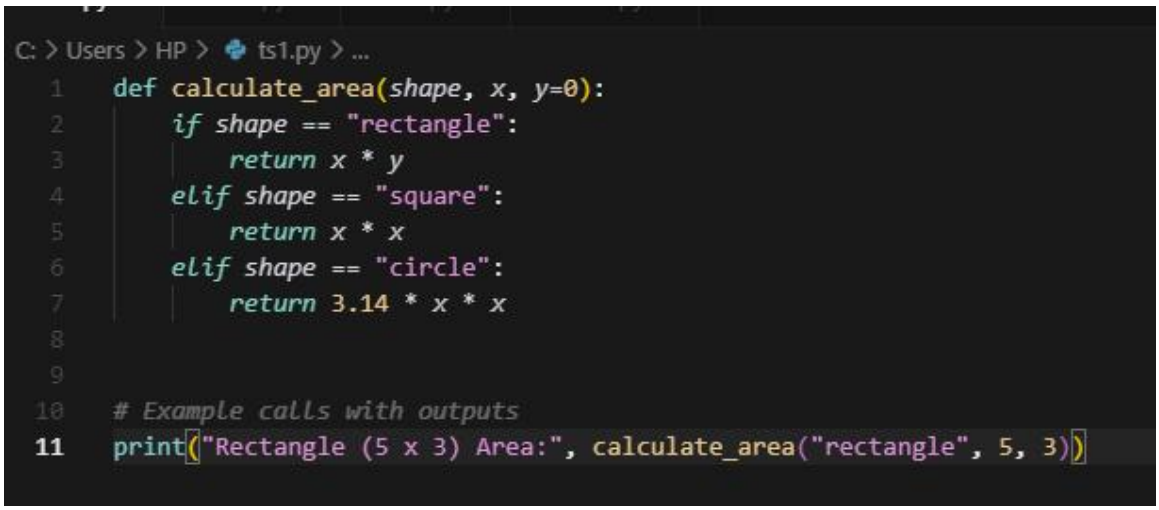
Prompt:

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

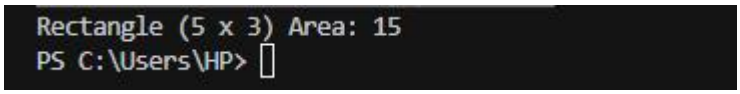
Python Code (Legacy):

```
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:



Output:



Observations:

- Removed repetitive if-elif structure.
- Used dictionary-based dispatch.
- More scalable for adding new shapes.
- Used math.pi for better precision.

Task 2 – Error Handling in Legacy Code

Prompt:

Legacy function without proper error handling:

Python Code (Legacy):

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

Code:

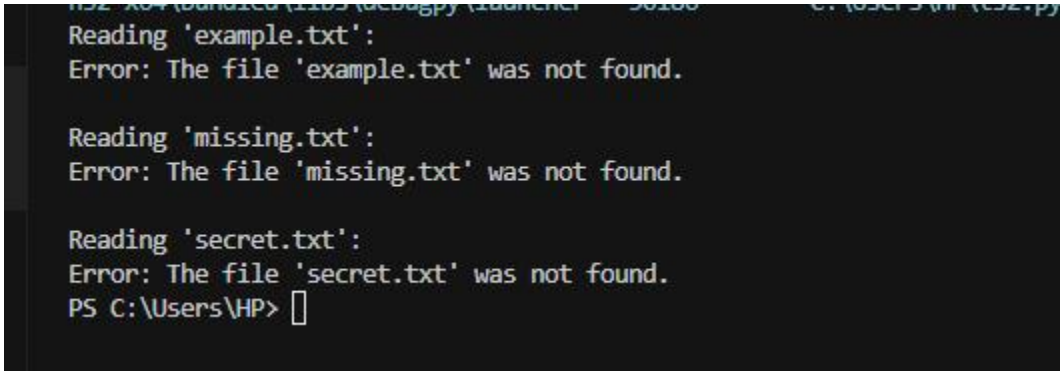
```python
def read_file(filename):
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        return f"Error: The file '{filename}' was not found."
    except PermissionError:
        return f"Error: Permission denied for file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

# Case 1: File exists
print("Reading 'example.txt':")
print(read_file("example.txt"))  # Replace with an actual file on your system

# Case 2: File does not exist
print("\nReading 'missing.txt':")
print(read_file("missing.txt"))

# Case 3: File without permission (simulation depends on OS)
print("\nReading 'secret.txt':")
print(read_file("secret.txt"))
```

Output:



Observations:

- Added error handling using try-except.

- Used with open() for automatic file closing.

- Prevents program crashes for missing or restricted files.

Task 3 – Complex Refactoring (Student Class)

Prompt:

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

Python Code (Legacy):

```
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
```

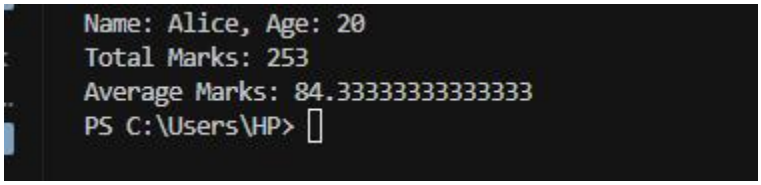
Code:

```
class Student:
    def show_details(self):
        print(f"Name: {self.name}, Age: {self.age}")

    def total_marks(self):
        (method) def average_marks(self: Self@Student) -> (float | Literal[0])
        Return the average marks, if available.
    def average_marks(self):
        """Return the average marks, if available."""
        return sum(self.marks) / len(self.marks) if self.marks else 0


if __name__ == "__main__":
    student1 = Student("Alice", 20, [85, 90, 78])
    student1.show_details()
    print("Total Marks:", student1.total_marks())
    print("Average Marks:", student1.average_marks())
Ctrl+L to chat, Ctrl+K to generate
```

Output:



Observations:

- Improved naming conventions (name, age, marks).

- Added docstrings for clarity.

- Used sum() for modular total.

- Added average_marks() for extended functionality.

Task 4 – Inefficient Loop Refactoring

Prompt:

Refactor this inefficient loop with AI help.

Python Code (Legacy):

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

Code:

```python
# Task: Refactor inefficient loop into list comprehension

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

squares = [i * i for i in nums]

print("Original Numbers:", nums)
print("Squares:", squares)
```

Output:

```
Original Numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
PS C:\Users\HP>
```

Observations:

- Replaced inefficient loop with list comprehension.
- Code is shorter, more readable, and Pythonic.
- Improves performance for large datasets.