SR UNIVERSITY

AI ASSISTED CODING

LAB-13.2: Code Refactoring: Improving Legacy Code with AI Suggestion

Name :B.HASINI

Roll No:2503A51L13

Batch: 25BTCAICSB19

TASK #1:

Prompt used:

- Refactor this Python function to remove repetition using a dictionary or helper functions, keeping the same outputs. Add inline comments and a few test cases with expected results.

Before Refactoring code:

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
import math

def calculate_area(shape, *dims):
    formulas = {
        "rectangle": lambda l, w: l * w,
        "square": lambda s: s * s,
        "circle": lambda r: math.pi * r * r
    }
    try:
        return formulas[shape.lower()](*dims)
    except KeyError:
        raise ValueError(f"Unknown shape: {shape}")
    except TypeError:
        raise ValueError(f"Invalid number of arguments for {shape}")

print(calculate_area("rectangle", 5, 10))  # 50
print(calculate_area("square", 4))         # 16
print(calculate_area("circle", 3))         # 28.274333882308138
```

Observations:

1. **Flexibility with *dimensions:**The function can take a variable number of arguments depending on the shape.

2. **Improved Accuracy :**Uses math.pi instead of 3.14 for more precise circle calculations.

3. **Error Handling:**If the shape is unknown, the function raises a clear error message (ValueError).

TASK #2:

Prompt used:

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

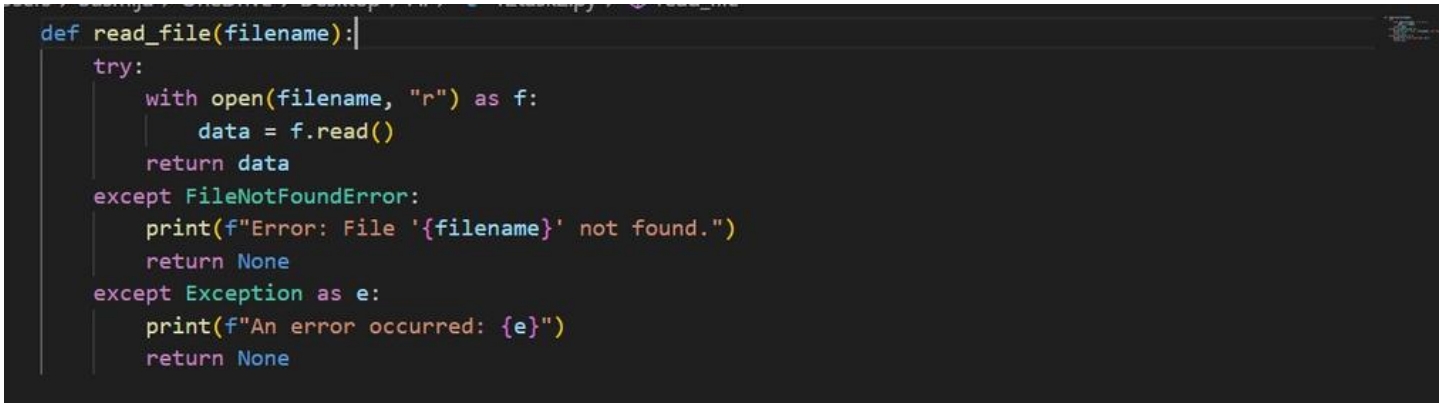
Before Refactoring code:

Python Code

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

f.close()
return data
```

After Refactoring Code Generated:



```
def read_file(filename):
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        print(f"Error: File '{filename}' not found.")
        return None
    except Exception as e:
        print(f"An error occurred: {e}")
        return None
```

Observations:

- Old code lacks error handling and manual file closing.
- New code uses with open() for auto file closing.
- Added try-except for handling errors safely.
- More readable, efficient, and reliable

TASK #3:

Prompt used:

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

Before Refactoring code:

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
```

After Refactoring Code Generated:

```python
class Student:
    def __init__(self, name, age, marks):
        self.name, self.age, self.marks = name, age, marks

    def details(self):
        return f"Name: {self.name}, Age: {self.age}"

    def total(self):
        return sum(self.marks)

    def average(self):
        return sum(self.marks) / len(self.marks)

# Example usage:
s1 = Student("Alice", 20, [85, 90, 78])
print(s1.details())           # Output: Name: Alice, Age: 20
print("Total:", s1.total())   # Output: Total: 253
print("Average:", s1.average()) # Output: Average: 84.33333333333333
```

Observations:

- Variables renamed to name, age, and marks instead of cryptic n, a, m1, etc.
- Method names are descriptive: show_details() and total_marks() clearly convey intent
- Class and methods have docstrings that explain purpose, parameters, and return values.
- This improves readability and maintainability of the users.

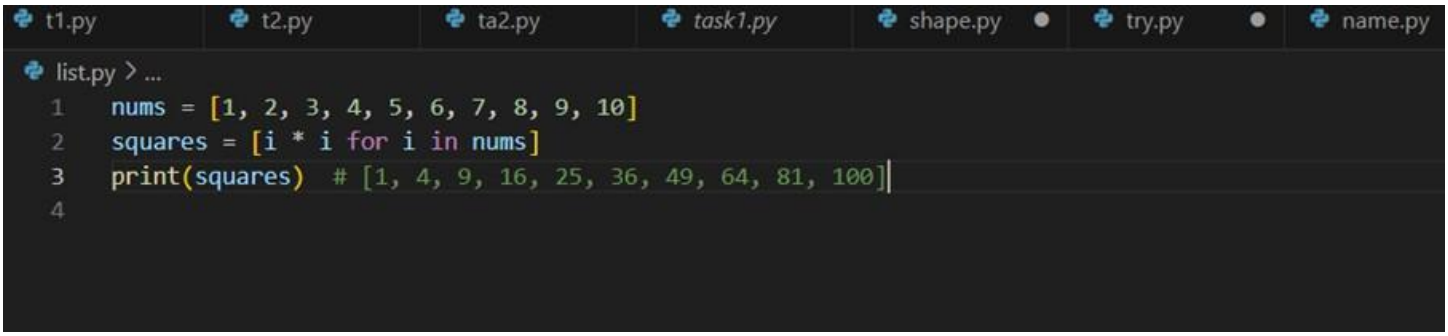
Task #4:

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:



Observation:

- The code was refactored from a loop with .append() to a list comprehension.
- This makes it more concise, readable, and Pythonic.
- It also improves performance since list comprehensions are optimized internally.
- The intent (generate squares of numbers) is now expressed in a single clear line.