

SR UNIVERSITY

AI ASSIST CODING

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

NAME:MD NASRIN

BATCH:19

2503a51l23

TASK #1:

Prompt used:

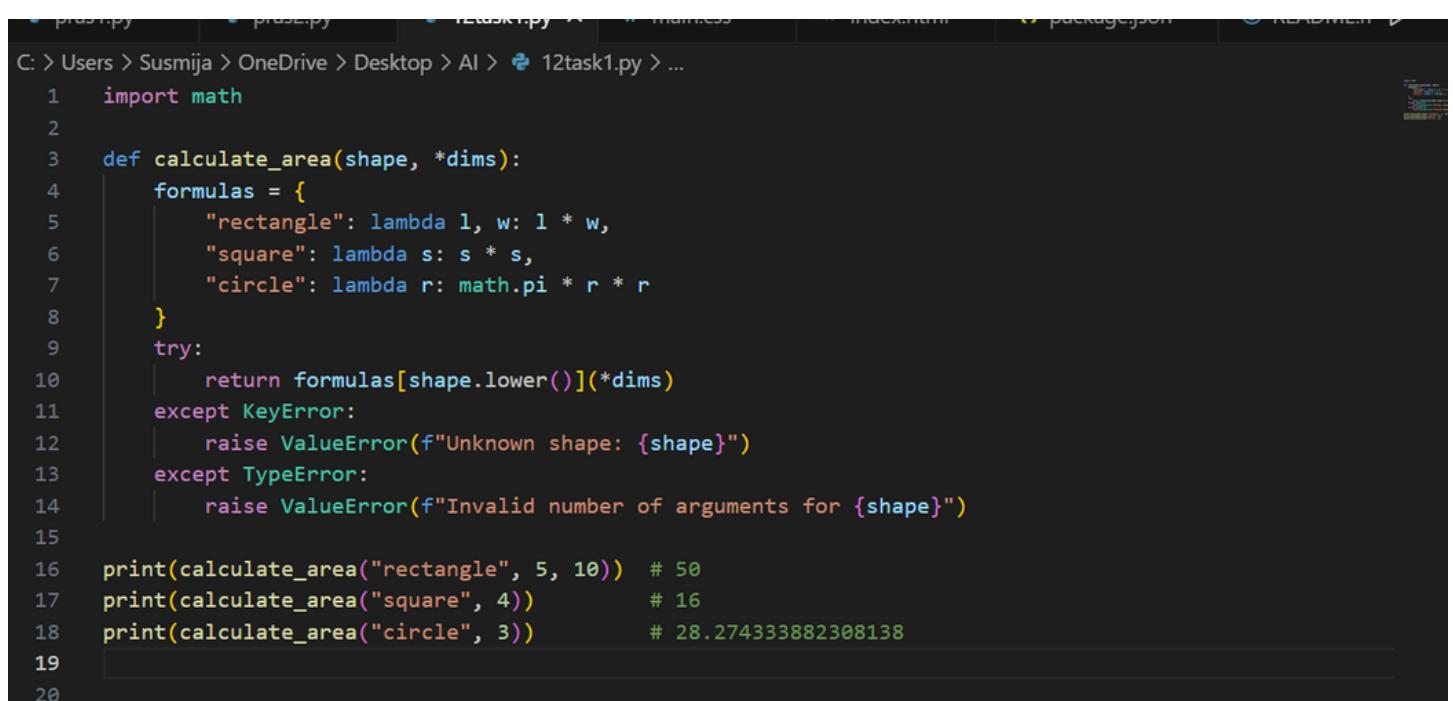
- Refactor this Python function to remove repetition using a dictionary or helper functions, keeping the same outputs. Add inline comments and a few test cases with expected results.

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:



The screenshot shows a code editor with a dark theme. The file being edited is '12task1.py'. The code has been refactored into a more modular form:

```
C: > Users > Susmija > OneDrive > Desktop > AI > 12task1.py > ...
1  import math
2
3  def calculate_area(shape, *dims):
4      formulas = {
5          "rectangle": lambda l, w: l * w,
6          "square": lambda s: s * s,
7          "circle": lambda r: math.pi * r * r
8      }
9      try:
10         return formulas[shape.lower()](*dims)
11     except KeyError:
12         raise ValueError(f"Unknown shape: {shape}")
13     except TypeError:
14         raise ValueError(f"Invalid number of arguments for {shape}")
15
16 print(calculate_area("rectangle", 5, 10)) # 50
17 print(calculate_area("square", 4))         # 16
18 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:



```
Users > Susmija > OneDrive > Desktop > AI > 12task2.py > read_file
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:

```
> Users > Susmija > OneDrive > Desktop > AI > 12task3.py > ...  
1  class Student:  
2      def __init__(self, name, age, marks):  
3          self.name, self.age, self.marks = name, age, marks  
4  
5      def details(self):  
6          return f"Name: {self.name}, Age: {self.age}"  
7  
8      def total(self):  
9          return sum(self.marks)  
10  
11     def average(self):  
12         return sum(self.marks) / len(self.marks)  
13  
14 # Example usage:  
15 s1 = Student("Alice", 20, [85, 90, 78])  
16 print(s1.details())          # Output: Name: Alice, Age: 20  
17 print("Total:", s1.total())  # Output: Total: 253  
18 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.

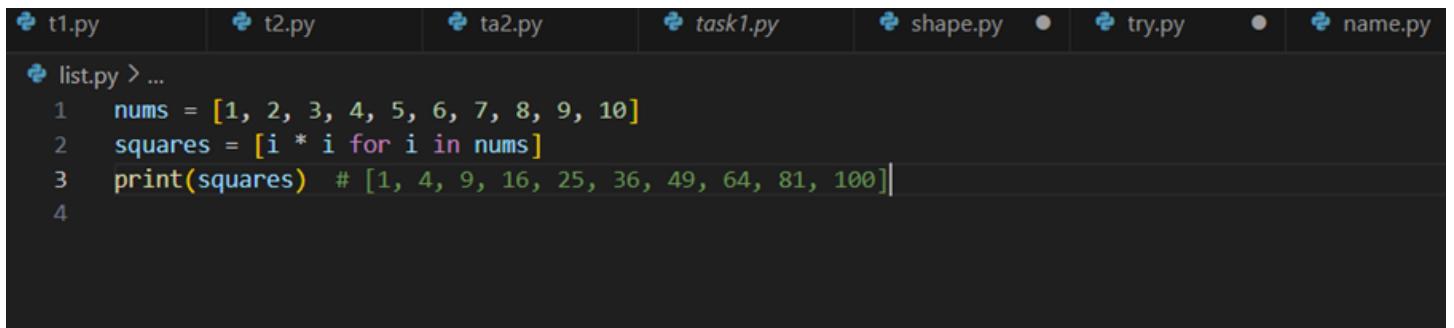
Task #4:

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:



A screenshot of a code editor showing a file named 'list.py'. The code contains the following Python code:

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

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.