

AI ASSISTED CODING LAB 13.2

**Lab 13: Code Refactoring: Improving
Legacy Code with AI Suggestions.**

ENROLLMENT NO: 2403A52007

NAME: M.RAKSHITH

COURSE: CSM(AI/ML)

BATCH: 02

DATE: 28-10-2025

Task Description 1 – Remove Repetition

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

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

Used Prompt:

**Refactor the given code,
convert into separate functions and make
it modular and cleaner. Take input from
user.**

```
shape = input("Enter the shape (rectangle, square, or circle): ").lower()

if shape == "rectangle":
    length = float(input("Enter the length of the rectangle: "))
    width = float(input("Enter the width of the rectangle: "))
    rectangle_area = calculate_rectangle_area(length, width)
    print(f"Area of rectangle: {rectangle_area}")
elif shape == "square":
    side = float(input("Enter the side length of the square: "))
    square_area = calculate_square_area(side)
    print(f"Area of square: {square_area}")
elif shape == "circle":
    radius = float(input("Enter the radius of the circle: "))
    circle_area = calculate_circle_area(radius)
    print(f"Area of circle: {circle_area}")
else:
    print("Invalid shape entered.")
```

Enter the shape (rectangle, square, or circle): rectangle
Enter the length of the rectangle: 3
Enter the width of the rectangle: 4
Area of rectangle: 12.0

Task Description 2 – Error Handling in Legacy Code

Task: Legacy function without proper error handling

CODE: def read_file(filename):

f = open(filename, "r")

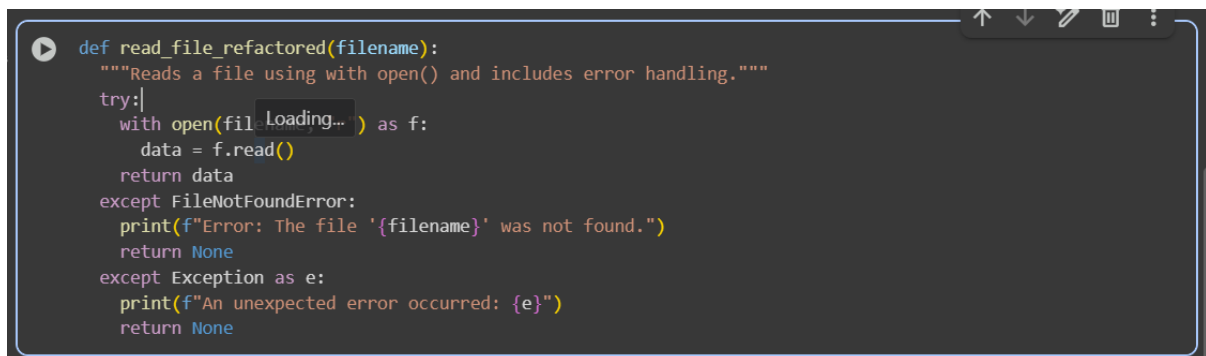
data = f.read()

f.close()

return data

Used Prompt:

**refactor the code with with
open() and try-except.**

A screenshot of a code editor with a dark background. The code is a Python function named `read_file_refactored` that takes a `filename` parameter. It uses a `try-except` block to handle file operations. The `try` block contains a `with open(filename, 'r') as f:` statement, followed by `data = f.read()` and `return data`. The `except FileNotFoundError:` block contains `print(f"Error: The file '{filename}' was not found.")` and `return None`. The `except Exception as e:` block contains `print(f"An unexpected error occurred: {e}")` and `return None`. The code is color-coded: keywords are blue, strings are orange, and comments are green. A play button icon is visible in the top left corner of the editor.

```
def read_file_refactored(filename):  
    """Reads a file using with open() and includes error handling."""  
    try:  
        with open(filename, 'r') as f:  
            data = f.read()  
            return data  
    except FileNotFoundError:  
        print(f"Error: The file '{filename}' was not found.")  
        return None  
    except Exception as e:  
        print(f"An unexpected error occurred: {e}")  
        return None
```

Task Description 3 – Complex Refactoring
Task: Provide this legacy class to AI for
readability and modularity improvements:

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
```

Used Prompt:

in the given code make more readability and cleaner by giving names such as (name, age, marks), add docstrings, improve print readability and use sum(self.marks) if marks are stored in a list and take input from the user

```

class Student:
    """Represents a student with name, age, and marks."""

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

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

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

    def total(self):
        """Calculates and returns the sum of the student's marks."""
        return sum(self.marks)

```

```

# Take input from the user
name = input("Enter student name: ")
age = int(input("Enter student age: "))
marks_str = input("Enter student marks (comma-separated): ")
marks = [int(mark.strip()) for mark in marks_str.split(',')]

# Create a Student object and use its methods
student1 = Student(name, age, marks)
student1.details()
print(f"Total marks: {student1.total()}")

```

```

Enter student name: mustafa
Enter student age: 19
Enter student marks (comma-separated): 88,84,91,95
Name: mustafa, Age: 19
Total marks: 358

```

Task Description #4 – Inefficient Loop

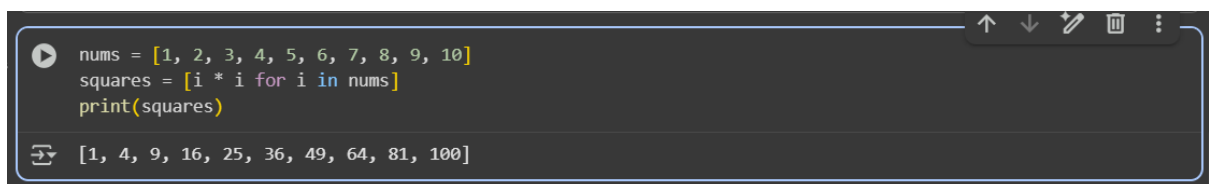
Refactoring

Task: Refactor this inefficient loop with AI help

CODE:

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

Used Prompt: refactor this inefficient loop.



The image shows a code editor window with a dark background. The code is written in Python and is as follows:

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

Below the code, the output of the script is displayed:

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

The code editor has a toolbar at the top right with icons for undo, redo, search, and other editing functions.