AI ASSISTED CODING.

LAB ASSIGNMENT -13.2

Program	: B-TECH
Specialization	: AIML
Name of Student	: DONIKELA LAXMAN
Enrollment-No	: 2403A51283
Batch No	: 01
Date	: 22-10-2025

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
```

PROMPT:

Remove Repetition
refactor the 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
```

GENERATED CODE:

```python
import math

def calculate_area(shape, x, y=0):
    """Calculates the area of different shapes.

    Args:
        shape: The shape type ('rectangle', 'square', or 'circle').
        x: The first dimension (length for rectangle, side for square, radius for circle).
        y: The second dimension for rectangle (width). Defaults to 0.

    Returns:
        The calculated area.
    """
    if shape == "rectangle":
        return x * y
    elif shape == "square":
        return x ** 2
    elif shape == "circle":
        return math.pi * x ** 2
    else:
        return "Invalid shape"

# Get input from the user
shape_input = input("Enter the shape (rectangle, square, or circle): ").lower()
x_input = float(input("Enter the first dimension: "))

if shape_input == "rectangle":
    y_input = float(input("Enter the second dimension (width): "))
    area = calculate_area(shape_input, x_input, y_input)
else:
    area = calculate_area(shape_input, x_input)

if isinstance(area, (int, float)):
    print(f"The area of the {shape_input} is: {area}")
else:
    print(area)
```

OUTPUT:-

```
Enter the shape (rectangle, square, or circle): square
Enter the first dimension: 3
The area of the square is: 9.0
```

TASK DESCRIPTION-2

Error Handling in Legacy Code

Task: Legacy function without proper error handling

Python Code

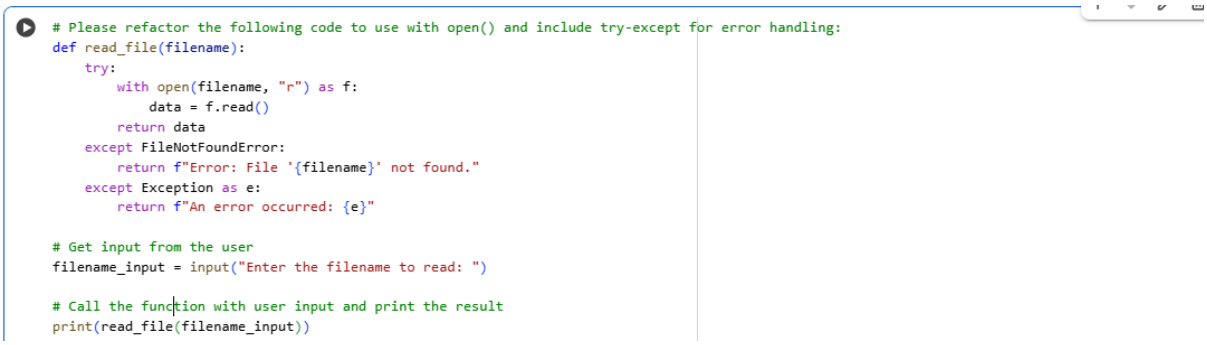
```
def read_file(filename):
f = open(filename, "r")
data = f.read()
```

f.close()
return data

PROMPT:- convert the given Python Code
def read_file(filename):
f = open(filename, "r")
data = f.read()

f.close()
return data Make sure to apply legacy function without proper error handling

GENERATED CODE:-



```
# Please refactor the following code to use with open() and include try-except for error handling:
def read_file(filename):
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        return f"Error: File '{filename}' not found."
    except Exception as e:
        return f"An error occurred: {e}"

# Get input from the user
filename_input = input("Enter the filename to read: ")

# Call the function with user input and print the result
print(read_file(filename_input))
```

OUTPUT:-

```
Enter the filename to read: manoj
Error: File 'manoj' not found.
```

TASK-3

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
```

PROMPT:

Refactor the following legacy Python class to improve readability, maintainability, and modularity 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
```

AI improves naming (name, age, marks).

Adds docstrings.

Improves print readability.

Possibly uses sum(self.marks) if marks stored in a list

GENERATED CODE:-

```python
# Please refactor the following class to improve readability, maintainability, and modularity,
# and incorporate the following:
# - Improve naming (name, age, marks).
# - Add docstrings.
# - Improve print readability in the details method.
# - Possibly use sum(self.marks) if marks are stored in a list.
class Student:
    """
    Represents a student with their name, age, and marks in three subjects.
    """
    def __init__(self, name, age, mark1, mark2, mark3):
        """
        Initializes a new Student object.

        Args:
            name (str): The name of the student.
            age (int): The age of the student.
            mark1 (float): The mark in the first subject.
            mark2 (float): The mark in the second subject.
            mark3 (float): The mark in the third subject.
        """
        self.name = name
        self.age = age
        self.marks = [mark1, mark2, mark3]

    def details(self):
        """
        Prints the details of the student (name and age).
        """
        print(f"Student Name: {self.name}\nStudent Age: {self.age}")

    def total(self):
        """
        Calculates and returns the total marks of the student.

        Returns:
            float: The sum of the student's marks.
        """
        return sum(self.marks)

# Example usage:
student1 = Student("Alice", 16, 85, 90, 78)
student1.details()
print(f"Total Marks: {student1.total()}")
```

OUTPUT:

```
Student Name: Alice
Student Age: 16
Total Marks: 253
```

TASK -4

Refactor this inefficient loop with AI help

PROMPT:

make and rewrite the code Python Code

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

squares = []

for i in nums:

squares.append(i * i) add list comprehension

GENERATED CODE:

```
# Please rewrite the following code using a list comprehension:
nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
squares = [i * i for i in nums]

# You can print the new 'squares' list to verify the result:
print(squares)
```

OUTPUT:-

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

THANK YOU