AI ASSISTED CODING

ASSIGNMENT 13.2

NAME: ALUWALA SHARANYA

HT NO: 2403A52013

BATCH NO: 02

TASK 1~

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:

Refactor the following Python code to remove redundancy:

```
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:

```python
import math

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

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

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

CODE EXPLANATION:

This Python function calculate_area computes the area of rectangles, squares, or circles. It takes the shape name, a primary dimension x, and an optional second dimension y for rectangles. It uses math.pi for accurate circle area calculations and returns "Invalid shape" for unknown shapes

TASK 2~

Legacy function without proper error handling
Python Code

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

f.close()
return data
```

PROMPT:

Refactor the following Python code to include proper error handling for file operations:

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

f.close()
return data
```

CODE & OUTPUT:

```python
def read_file(filename):
    """Reads content from a file with error handling.

    Args:
        filename: The name of the file to read.

    Returns:
        The content of the file, or an error message if the file is not found.
    """
    try:
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        return f"Error: File '{filename}' not found."
```

CODE EXPLANATION:

This Python function read_file attempts to open and read the content of a file specified by filename. It uses a try...except block to gracefully handle a FileNotFoundError. If the file is found, it returns the file's content; otherwise, it returns an informative error message indicating that the file was 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:

Improve the readability and modularity of the following Python class


```
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 & OUTPUT:

```
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 EXPLANATION:

This Python code defines a Student class.
The __init__ method initializes a student object with a name, age, and three marks. The details method prints the student's name and age, and the total method calculates and returns the sum of the three marks. This class provides a structured way to represent and manage student information.

TASK 4~

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)

PROMPT:

Refactor this inefficient loop in Python:

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

CODE & OUTPUT:

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

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

CODE EXPLANATION:

This Python code initializes a list called nums with integers from 1 to 10. It then uses a list comprehension [i * i for i in nums] to create a new list called squares. This list comprehension efficiently iterates through nums, squares each element, and collects the results into squares. Finally, it prints the squares list.