PROGRAM : B.TECH/CSE

SPECIALIZATION : AIML

COURSE TITLE : AI ASSISTED CODING

COURSE CODE : 24CS101PC214

SEMESTER : 3RD

NAME OF THE STUDENT : KARNAKANTI GODADEVI

ENROLLMENT NO : 2403A52003

BATCH NO : 01

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
```

Expected Output

• Refactored version with dictionary-based dispatch or separate functions.

• Cleaner and modular design.

Prompt:

Refactor the following redundant Python code to remove repetition and make it cleaner and modular. use a dictionary-based dispatch or separate functions for each shape.

```
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 rectangle_area(x, y):
  return x * y

def square_area(x):
  return x * x

def circle_area(x):
  return math.pi * x * x

area_functions = {
    "rectangle": rectangle_area,
    "square": square_area,
    "circle": circle_area
}

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

  Args:
    shape: The name of the shape (e.g., "rectangle", "square", "circle").
    x: The first dimension (e.g., length for rectangle/square, radius for circle).
    y: The second dimension (only used for rectangle, defaults to 0).

  Returns:
    The area of the shape.
```

```python
    x: The first dimension (e.g., length for rectangle/square, radius for circle).
    y: The second dimension (only used for rectangle, defaults to 0).

  Returns:
    The area of the shape.
  """
  if shape in area_functions:
    if shape == "rectangle":
      return area_functions[shape](x, y)
    else:
      return area_functions[shape](x)
  else:
    raise ValueError("Invalid shape provided.")

# Example usage
print(f"Area of rectangle (4, 5): {calculate_area('rectangle', 4, 5)}")
print(f"Area of square (4): {calculate_area('square', 4)}")
print(f"Area of circle (3): {calculate_area('circle', 3)}")
```

```
Area of rectangle (4, 5): 20
Area of square (4): 16
Area of circle (3): 28.274333882308138
```

Observation:

import math: This line imports the math module, which is needed to access the value of pi (math.pi) for the circle area calculation.

rectangle_area(x, y): This function calculates the area of a rectangle given its length (x) and width (y).

square_area(x): This function calculates the area of a square given its side length (x).

circle_area(x): This function calculates the area of a circle given its radius (x) using the formula πr^2.

area_functions = { ... }: This creates a dictionary called area_functions that maps shape names (strings like "rectangle", "square", "circle") to their corresponding area calculation functions. This is a way to dispatch or call the correct function based on the shape name.

calculate_area(shape, x, y=0): This is the main function that takes the shape name, the first dimension (x), and an optional second dimension (y) as input.

It checks if the provided shape is a key in the area_functions dictionary.

If it's a "rectangle", it calls the rectangle_area function with both x and y.

For other shapes ("square" and "circle"), it calls the corresponding function from the area_functions dictionary with only x.

If the shape is not found in the dictionary, it raises a ValueError.

Example usage: The lines at the end demonstrate how to use the calculate_area function to find the areas of a rectangle, a square, and a circle and print the results.

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
```

Expected Output:

AI refactors with with open() and try-except:

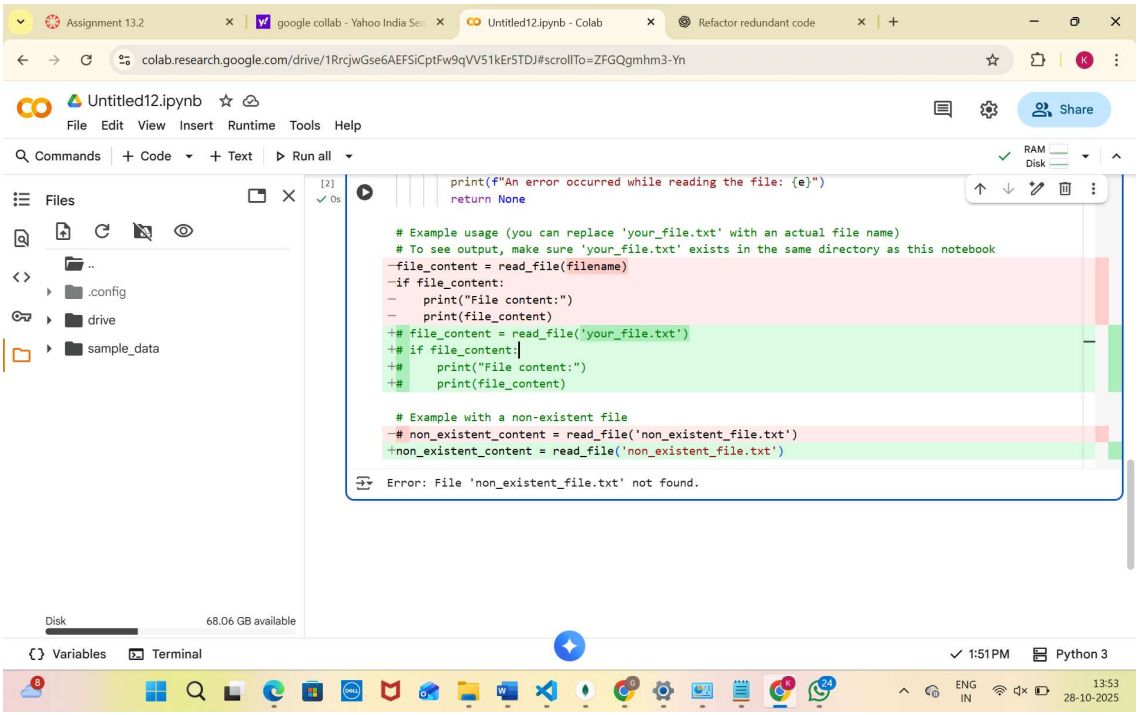
Prompt:

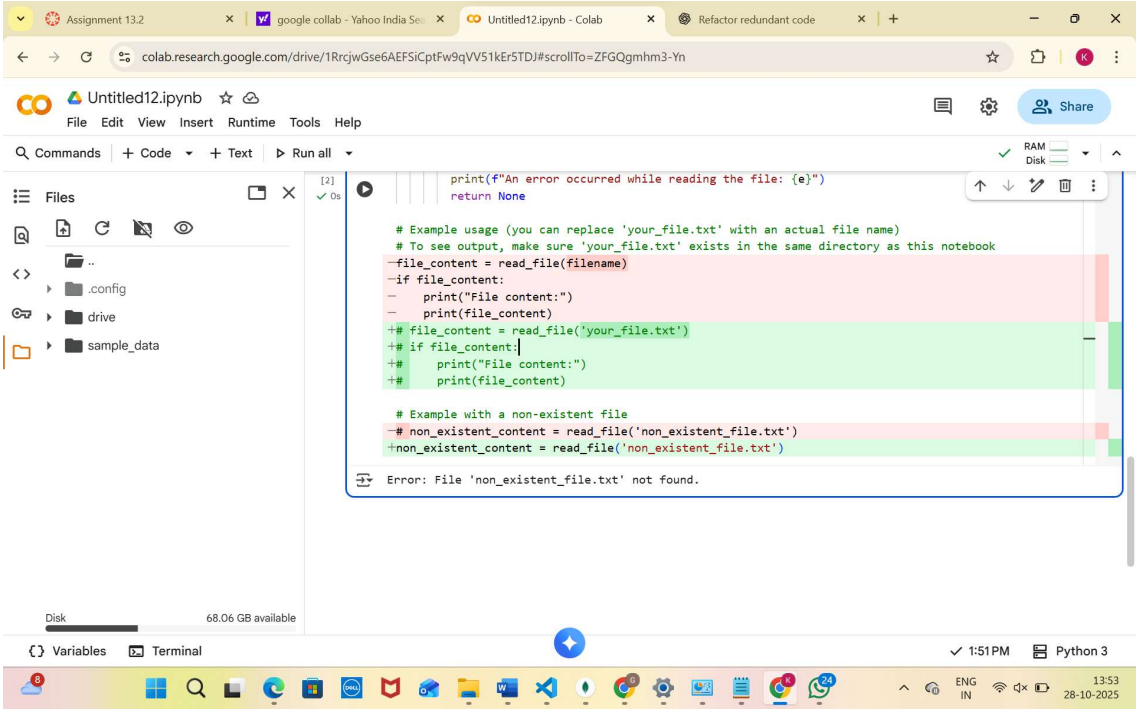
Refactor the following legacy Python code to include proper error handling and use modern best practices such as with open().

```
def read_file(filename):

    f = open(filename, "r")
```

```
data = f.read()

f.close()

return data
```

Code&Output:





Observation:

- def read_file(filename):: This line defines the function read_file that takes one argument, filename, which is the name of the file to be read.
- Docstring: The text within the triple quotes explains what the function does, its arguments, and what it returns.
- try...except block: This block is used for error handling.
    - try:: The code inside this block is attempted to be executed.
    - with open(filename, "r") as f:: This opens the file specified by filename in read mode ("r"). The with statement ensures that the file is automatically closed even if errors occur. The opened file object is assigned to the variable f.
    - data = f.read(): This reads the entire content of the file and stores it in the data variable.
    - return data: If the file is read successfully, the function returns the file content.
    - except FileNotFoundError:: If a FileNotFoundError occurs (meaning the file doesn't exist), the code inside this block is executed. It prints an error message indicating that the file was not found.

    - except Exception as e:: This catches any other potential exceptions that might occur during file reading. It prints a generic error message along with the specific error that occurred.

- return None: If any exception occurs, the function returns None.

The commented-out sections show examples of how to use the read_file function with an existing file and a non-existent file.

Task Description #3 – Complex Refactoring

Task: 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
```

Expected Output:

• AI improves naming (name, age, marks).
• Adds docstrings.
• Improves print readability.
• Possibly uses sum(self.marks) if marks stored in a list

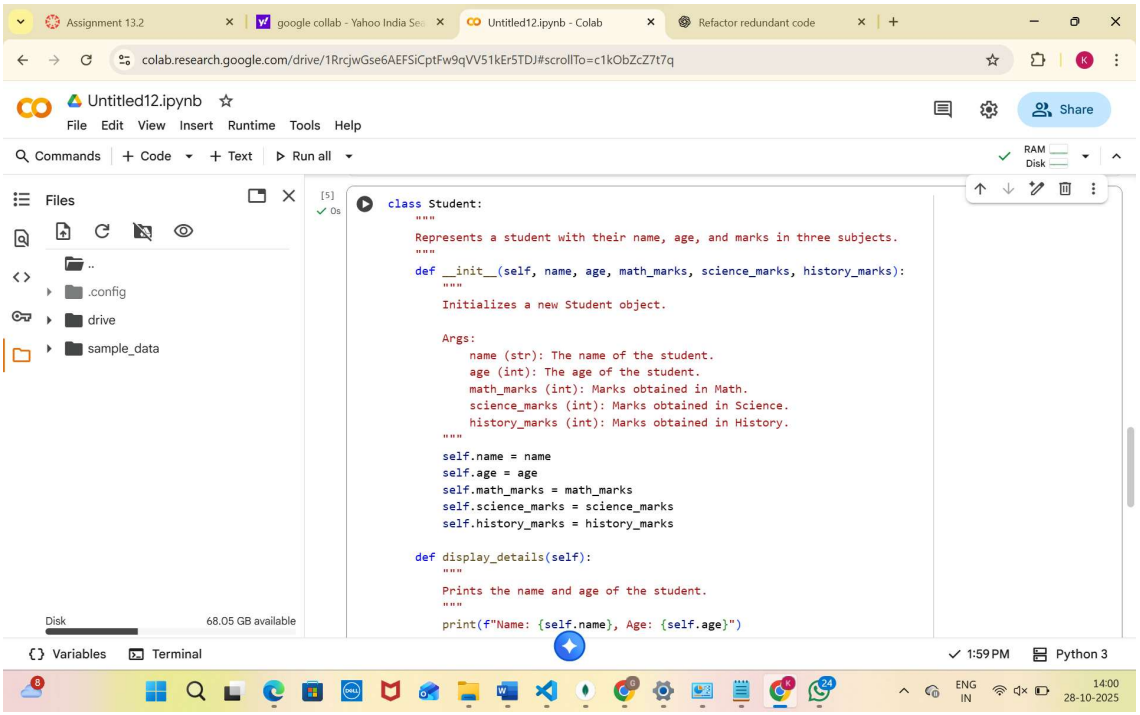
Prompt:

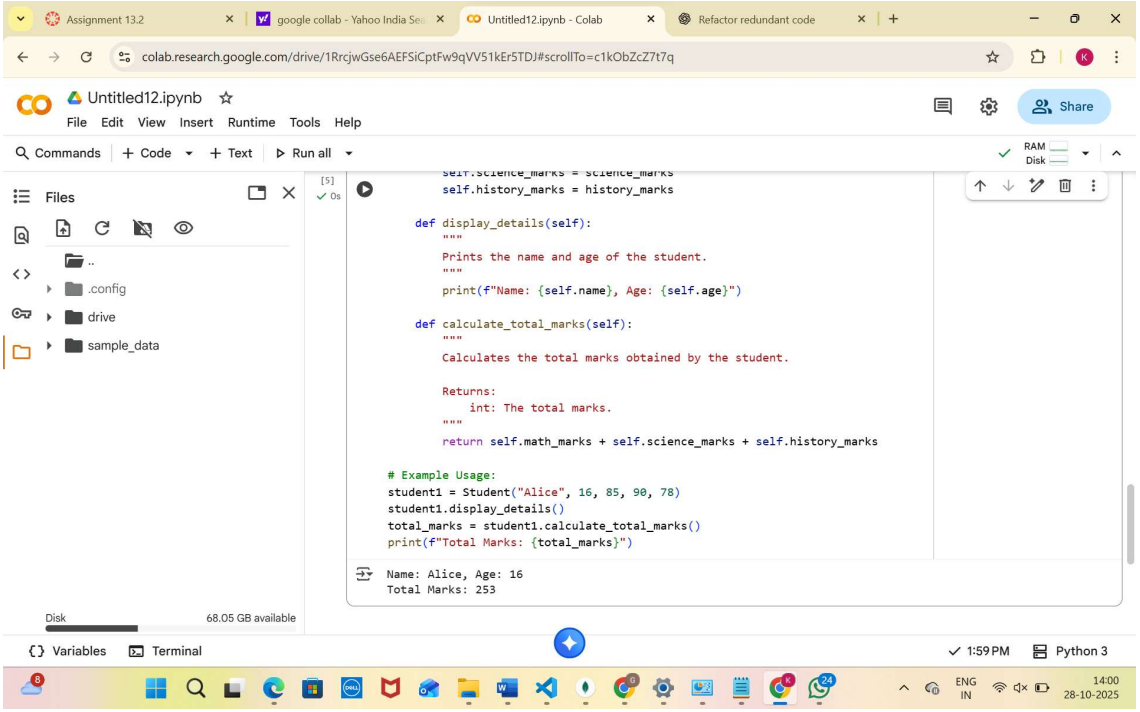
Refactor the following legacy Python class to improve readability, naming, and modularity.

```
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
```

Code &Output:





Observation:

- class Student:: This line defines the beginning of the class.
- __init__(self, name, age, math_marks, science_marks, history_marks): This is the constructor method. It's called when you create a new Student object. It initializes the object's attributes (self.name, self.age, self.math_marks, self.science_marks, self.history_marks) with the values passed as arguments.
- display_details(self): This method prints the student's name and age to the console.
- calculate_total_marks(self): This method calculates the sum of the student's marks in math, science, and history and returns the total.

The commented-out section at the end shows an example of how to create a Student object, display its details, and calculate its total marks.

Task Description #4 – Inefficient Loop Refactoring

Task: 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)

Expected Output: AI suggested a list comprehension

Prompt:

Refactor the following inefficient Python loop to make it more concise and Pythonic.

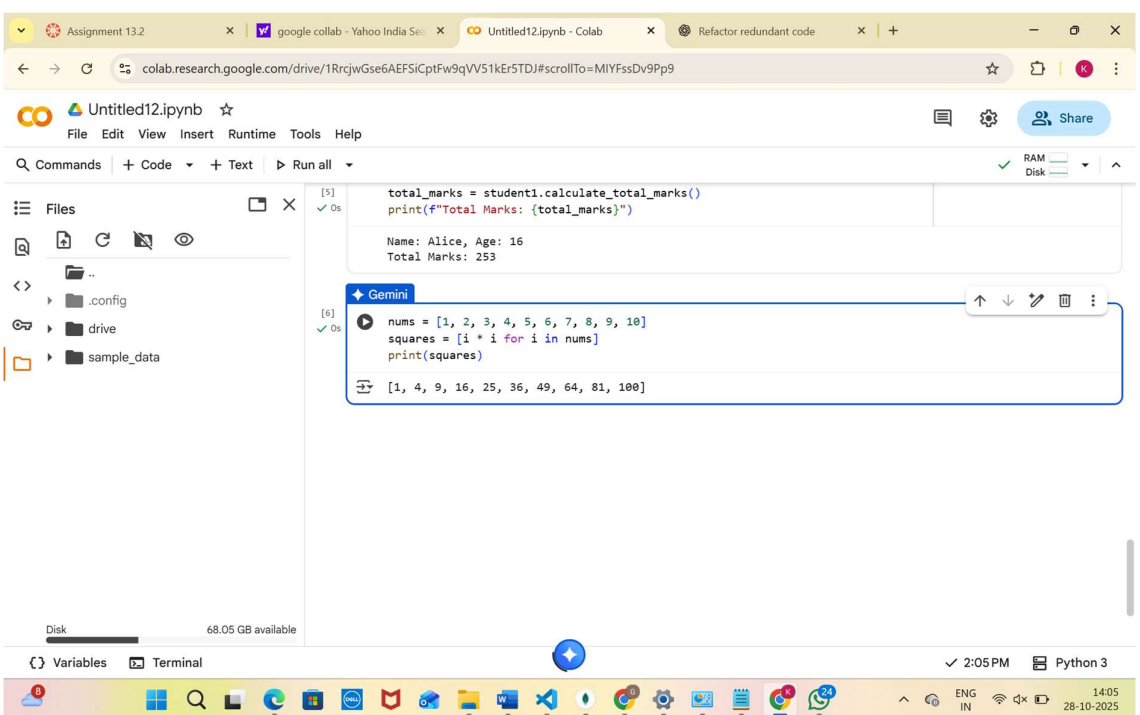
nums = [1,2,3,4,5,6,7,8,9,10]

squares = []

for i in nums:

  squares.append(i * i)

Code&Output:



Observation:

- nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: This line initializes a list named nums containing integers from 1 to 10.
- squares = [i * i for i in nums]: This is a list comprehension. It's a concise way to create lists. It iterates through each element i in the nums list and creates a new list squares where each element is the square of i (i * i).
- print(squares): This line prints the squares list to the console.