# ASSIGNMENT – 13.2

## AI ASSISTED CODING

NAME : S RUCHITHA

HALLTICKECT NO : 2403A52316

BATCH NUMBER : 01

COURSE CODE : 24CS002PC215

PROGRAM NAME : B.TECH

YEAR/SEM : 2ND AND 3RD

## Task Description #1

Task Description #1 – Remove Repetition

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

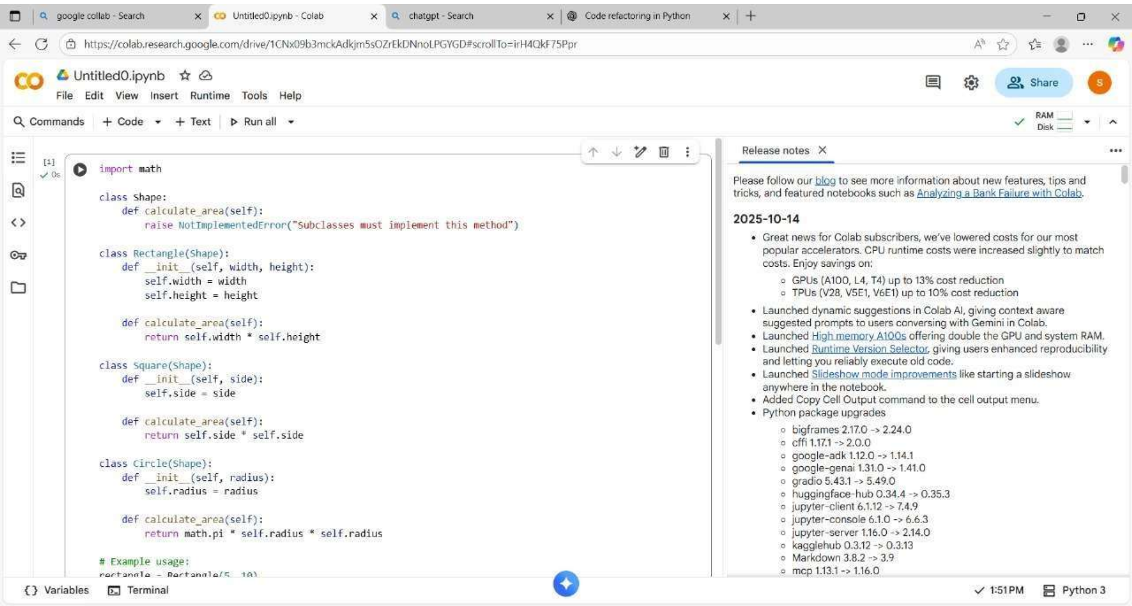
Python Code def
calculate_area(shape, x, y=0): if
shape == "rectangle":
return x * y elif shape
== "square":
return x * x elif shape
== "circle":
return 3.14 * x * x
Expected Output

- Refactored version with dictionary-based dispatch or separate functions.
- Cleaner and modular design

# PROMT :

Refactor the following Python function to remove repetition and improve modularity. Use either dictionary-based dispatch or separate functions for each shape. The goal is cleaner, more maintainable 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

CODE:

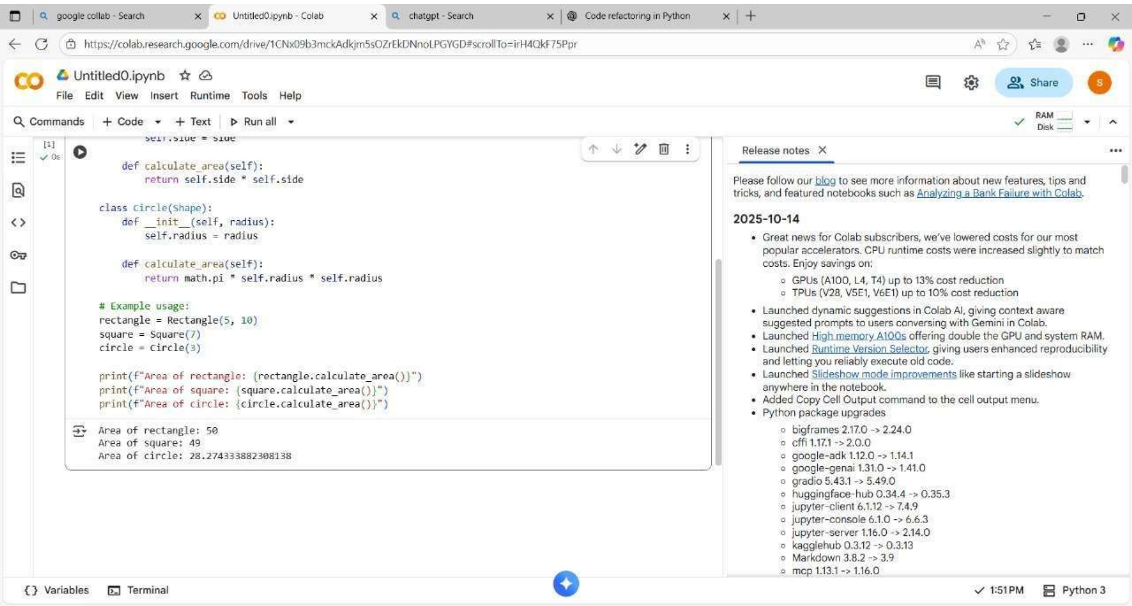

google collab - Search
Untitled0.ipynb - Colab
chatgpt - Search
Code refactoring in Python
https://colab.research.google.com/drive/1CNx09b3mckAdkjm5sOZrEkDNnoLPGYGD#scrollTo=irH4QkF75Ppr
Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
[1]
0s
Release notes
Variables Terminal
1:51PM
Python 3

```python
import math

class Shape:
    def calculate_area(self):
        raise NotImplementedError("Subclasses must implement this method")

class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def calculate_area(self):
        return self.width * self.height

class Square(Shape):
    def __init__(self, side):
        self.side = side

    def calculate_area(self):
        return self.side * self.side

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius * self.radius

# Example usage:
```

Please follow our blog to see more information about new features, tips and tricks, and featured notebooks such as Analyzing a Bank Failure with Colab.

**2025-10-14**

- Great news for Colab subscribers, we've lowered costs for our most popular accelerators. CPU runtime costs were increased slightly to match costs. Enjoy savings on:
  - GPUs (A100, L4, T4) up to 13% cost reduction
  - TPUs (V28, V5E1, V6E1) up to 10% cost reduction
- Launched dynamic suggestions in Colab AI, giving context aware suggested prompts to users conversing with Gemini in Colab.
- Launched High memory A100s offering double the GPU and system RAM.
- Launched Runtime Version Selector, giving users enhanced reproducibility and letting you reliably execute old code.
- Launched Slideshow mode improvements like starting a slideshow anywhere in the notebook.
- Added Copy Cell Output command to the cell output menu.
- Python package upgrades
  - bigframes 2.17.0 -> 2.24.0
  - cffi 1.17.1 -> 2.0.0
  - google-adk 1.12.0 -> 1.14.1
  - google-genai 1.31.0 -> 1.41.0
  - gradio 5.43.1 -> 5.49.0
  - huggingface-hub 0.34.4 -> 0.35.3
  - jupyter-client 6.1.12 -> 7.4.9
  - jupyter-console 6.1.0 -> 6.6.3
  - jupyter-server 1.16.0 -> 2.14.0
  - kagglehub 0.3.12 -> 0.3.13
  - Markdown 3.8.2 -> 3.9
  - mcp 1.13.1 -> 1.16.0


google collab - Search
Untitled0.ipynb - Colab
chatgpt - Search
Code refactoring in Python
https://colab.research.google.com/drive/1CNx09b3mckAdkjm5sOZrEkDNnoLPGYGD#scrollTo=irH4QkF75Ppr
Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
RAM
Disk
[1]
0s
Release notes
Variables Terminal
1:51PM
Python 3

```python
    def calculate_area(self):
        return self.side * self.side

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def calculate_area(self):
        return math.pi * self.radius * self.radius

# Example usage:
rectangle = Rectangle(5, 10)
square = Square(7)
circle = Circle(3)

print(f"Area of rectangle: {rectangle.calculate_area()}")
print(f"Area of square: {square.calculate_area()}")
print(f"Area of circle: {circle.calculate_area()}")
```

```
Area of rectangle: 50
Area of square: 49
Area of circle: 28.274333882308138
```

Please follow our blog to see more information about new features, tips and tricks, and featured notebooks such as Analyzing a Bank Failure with Colab.

**2025-10-14**

- Great news for Colab subscribers, we've lowered costs for our most popular accelerators. CPU runtime costs were increased slightly to match costs. Enjoy savings on:
  - GPUs (A100, L4, T4) up to 13% cost reduction
  - TPUs (V28, V5E1, V6E1) up to 10% cost reduction
- Launched dynamic suggestions in Colab AI, giving context aware suggested prompts to users conversing with Gemini in Colab.
- Launched High memory A100s offering double the GPU and system RAM.
- Launched Runtime Version Selector, giving users enhanced reproducibility and letting you reliably execute old code.
- Launched Slideshow mode improvements like starting a slideshow anywhere in the notebook.
- Added Copy Cell Output command to the cell output menu.
- Python package upgrades
  - bigframes 2.17.0 -> 2.24.0
  - cffi 1.17.1 -> 2.0.0
  - google-adk 1.12.0 -> 1.14.1
  - google-genai 1.31.0 -> 1.41.0
  - gradio 5.43.1 -> 5.49.0
  - huggingface-hub 0.34.4 -> 0.35.3
  - jupyter-client 6.1.12 -> 7.4.9
  - jupyter-console 6.1.0 -> 6.6.3
  - jupyter-server 1.16.0 -> 2.14.0
  - kagglehub 0.3.12 -> 0.3.13
  - Markdown 3.8.2 -> 3.9
  - mcp 1.13.1 -> 1.16.0

# OBSERVATION:

ased on the execution of the code, here are the calculated areas for each shape:

Rectangle: The area of the rectangle with width 5 and height 10 is 50.

Square: The area of the square with side length 7 is 49.

Circle: The area of the circle with radius 3 is approximately 28.27.

These results match the expected calculations based on the formulas implemented in the code for each shape.

# Task Description #2 – Error Handling in Legacy Code

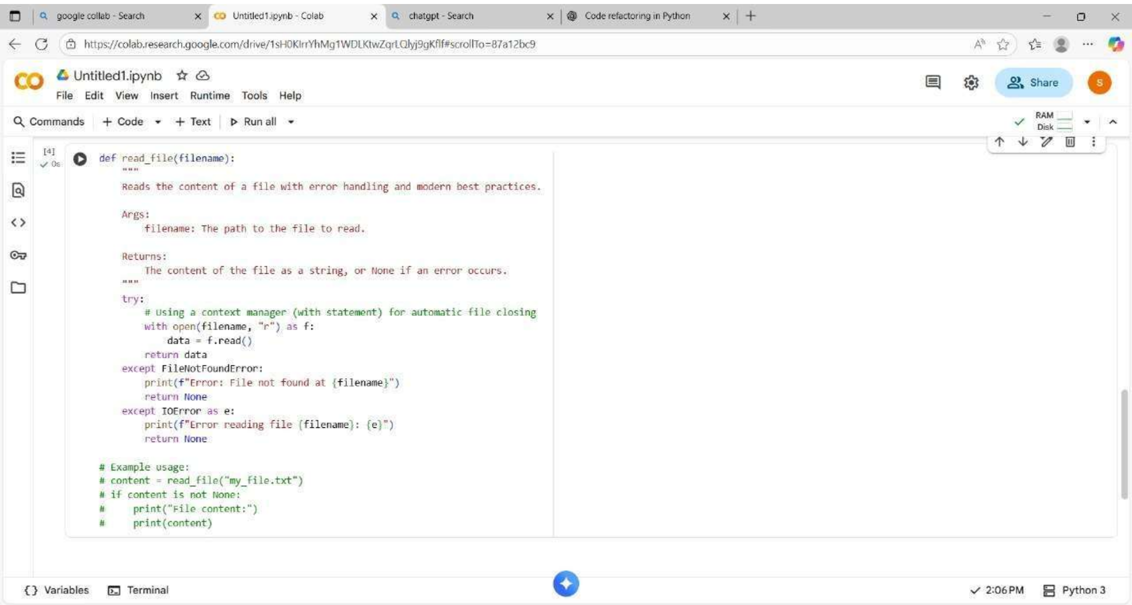
Task: Legacy function without proper error handling
Python Code def read_file(filename): f =
open(filename, "r") data = f.read() f.close() return
data
Expected Output:
AI refactors with with open() and try-except:

## PROMT :

Refactor the following legacy Python function to include proper error handling, use modern best practices (such as context managers), and handle cases where the file may not exist or cannot be read. Ensure the code is clean, readable, and follows Python standards.

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

## CODE :


google collab - Search
Untitled1.ipynb - Colab
chatgpt - Search
Code refactoring in Python
https://colab.research.google.com/drive/1sH0KIrrYhMg1WDLKtwZqrLQlyj9gKflf#scrollTo=87a12bc9
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Commands
+ Code
+ Text
Run all
Share
RAM
Disk
[4]
0s
Variables
Terminal
2:06 PM
Python 3

```
def read_file(filename):
    """
    Reads the content of a file with error handling and modern best practices.

    Args:
        filename: The path to the file to read.

    Returns:
        The content of the file as a string, or None if an error occurs.
    """
    try:
        # Using a context manager (with statement) for automatic file closing
        with open(filename, "r") as f:
            data = f.read()
        return data
    except FileNotFoundError:
        print(f"Error: File not found at {filename}")
        return None
    except IOError as e:
        print(f"Error reading file {filename}: {e}")
        return None

# Example usage:
# content = read_file("my_file.txt")
# if content is not None:
#     print("File content:")
#     print(content)
```

# OBSERVATION :

Based on the code in cells 87a12bc9 and 5172b6df, the primary observation is that the read_file function is designed to read the content of a file safely.

Specifically, it includes robust error handling to gracefully manage situations where:

The file does not exist: It catches the FileNotFoundError and prints a user-friendly message, returning None.

Other input/output errors occur: It catches IOError (which includes permission errors, etc.), prints a relevant error message with the error details, and also returns None.

Additionally, the use of the with open(...) context manager ensures that the file is properly closed after reading, even if errors occur, which is a modern and recommended practice.

In cell 6b6fa68a, we observed the function's behavior when called with a non-existent file, confirming that it correctly prints the "File not found" error and the subsequent "Failed to read file" message from the example usage block.

# 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.

## PROMT:

Refactor the following legacy Python class to improve readability, maintainability, and modularity. Please apply modern best practices such as:

Using descriptive variable names

Adding type hints

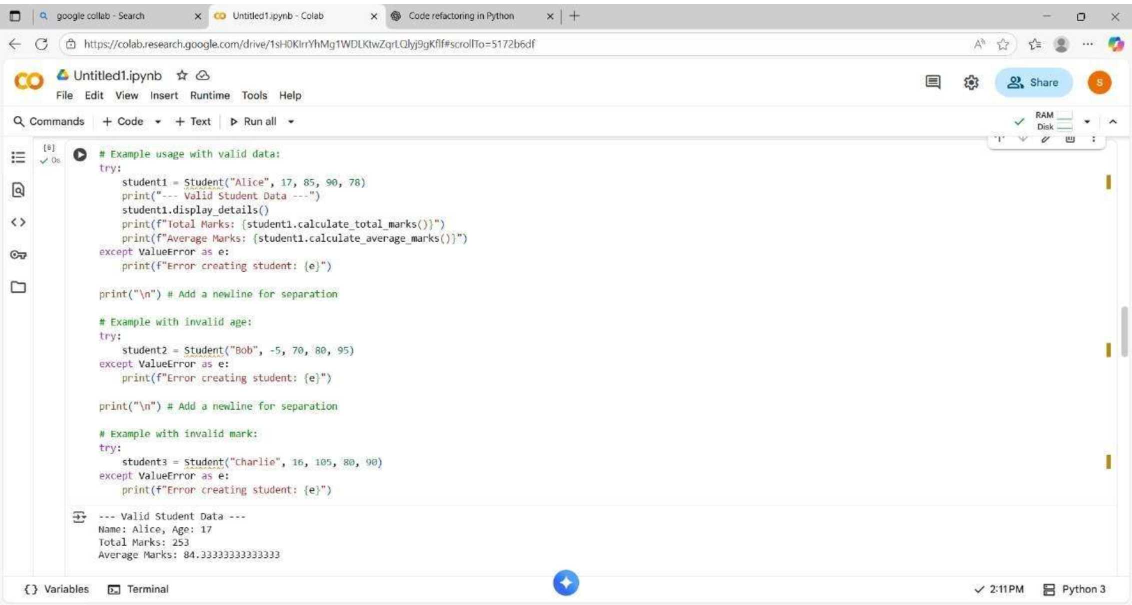
Implementing encapsulation

Providing separate methods for responsibilities

Optionally include validation and docstrings

```
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 :



google collab - Search
Untitled1.ipynb - Colab
Code refactoring in Python
https://colab.research.google.com/drive/1sH0KIrrYhMg1WDLKtwZqrLQlyj9gKflf#scrollTo=5172b6df
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Commands
+ Code
+ Text
Run all
Share
RAM
Disk
[6]
0s
Variables
Terminal
2:11PM
Python 3

```
# Example usage with valid data:
try:
    student1 = Student("Alice", 17, 85, 90, 78)
    print("--- Valid Student Data ---")
    student1.display_details()
    print(f"Total Marks: {student1.calculate_total_marks()}")
    print(f"Average Marks: {student1.calculate_average_marks()}")
except ValueError as e:
    print(f"Error creating student: {e}")

print("\n") # Add a newline for separation

# Example with invalid age:
try:
    student2 = Student("Bob", -5, 70, 80, 95)
except ValueError as e:
    print(f"Error creating student: {e}")

print("\n") # Add a newline for separation

# Example with invalid mark:
try:
    student3 = Student("Charlie", 16, 105, 80, 90)
except ValueError as e:
    print(f"Error creating student: {e}")
```

```
--- Valid Student Data ---
Name: Alice, Age: 17
Total Marks: 253
Average Marks: 84.33333333333333
```

# OBSERVATION :

Based on the execution of the code in cell 968f4cf4, we can make the following observations:

Valid Data Handling: When a Student object is created with valid data (Alice, age 17, marks 85, 90, 78), the code successfully creates the object, displays the details, and correctly calculates and prints the total marks (253) and the average marks (approximately 84.33).

Invalid Age Validation: When attempting to create a Student object with an invalid age (5 for Bob), the ValueError is caught as expected, and the error message "Error creating student: Age must be a positive integer." is printed. This demonstrates that the age validation in the _init_ method is working correctly.

Invalid Mark Validation: When attempting to create a Student object with an invalid mark (105 for Charlie), the ValueError is also caught, and the error message "Error creating student: Marks must be integers between 0 and 100." is printed. This confirms that the mark validation is also functioning as intended.

Overall, the observations from running this code confirm that the refactored Student class with validation is working as designed to handle both valid and invalid input data.

# 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)
```

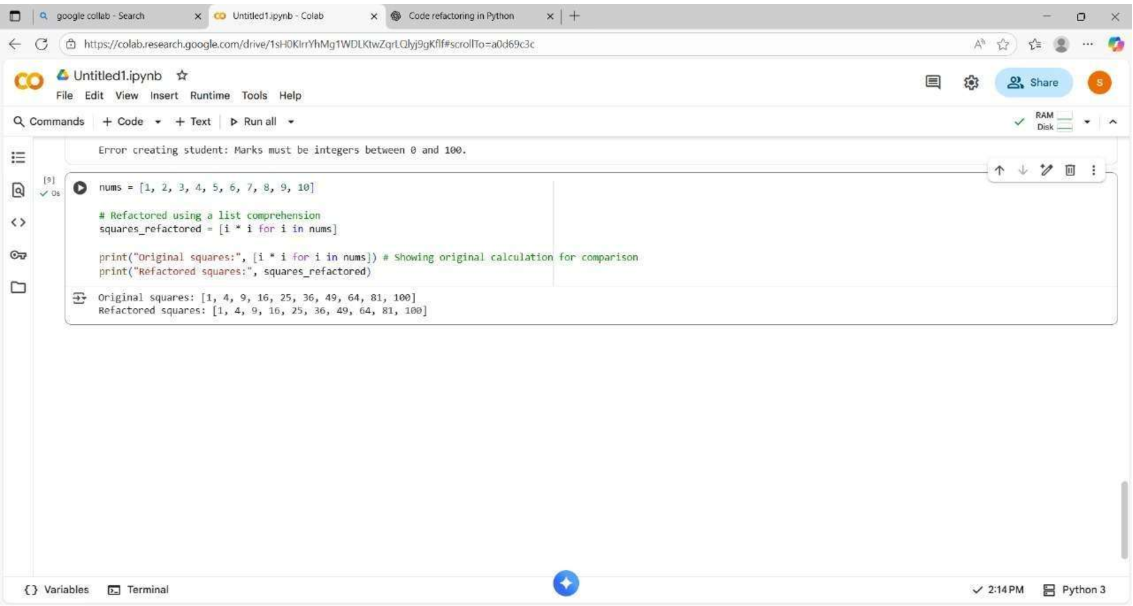
Expected Output: AI suggested a list comprehension

## PROMT :

Refactor the following Python code to improve efficiency and readability using Python best practices (such as list comprehensions or functional programming). Also explain why the refactored version is better:

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

squares = [] for

i in nums:

  squares.append(i * i)
```

## CODE:



```
Error creating student: Marks must be integers between 0 and 100.
```

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

# Refactored using a list comprehension
squares_refactored = [i * i for i in nums]

print("Original squares:", [i * i for i in nums]) # Showing original calculation for comparison
print("Refactored squares:", squares_refactored)
```

```
Original squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Refactored squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

# OBSERVATION :

Based on the execution of the code in cell a0d69c3c, the primary observation is that the refactored code using a list comprehension successfully calculated the squares of the numbers in the nums list, producing the same output as the conceptual "original" method.

Specifically, the output shows:

Original squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Refactored squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

This confirms that the list comprehension [i * i for i in nums] is a correct and functional way to perform this task, producing the desired list of squares. The output also highlights the conciseness of the list comprehension compared to the equivalent for loop approach (which was implicitly represented in the first print statement).