

ASSIGNMENT_13.1

A.Vivek Sai

HT.NO:2303A51710

Batch-11

Task Description #1 (Refactoring – Removing Code Duplication)

- Task: Use AI to refactor a given Python script that contains multiple repeated code blocks.

- Instructions:

- o Prompt AI to identify duplicate logic and replace it with functions or classes.

- o Ensure the refactored code maintains the same output.

- o Add docstrings to all functions.

- Sample Legacy Code:

Legacy script with repeated logic

```
print("Area of Rectangle:", 5 * 10)
```

```
print("Perimeter of Rectangle:", 2 * (5 + 10))
```

```
print("Area of Rectangle:", 7 * 12)
```

```
print("Perimeter of Rectangle:", 2 * (7 + 12))
```

```
print("Area of Rectangle:", 10 * 15)
```

```
print("Perimeter of Rectangle:", 2 * (10 + 15))
```

- Expected Output:

- o Refactored code with a reusable function and no duplication.

CODE:

```
def calculate_rectangle_properties(length, width):  
    """Calculates and prints the area and perimeter of a rectangle.  
  
    Args:  
        length (int or float): The length of the rectangle.  
        width (int or float): The width of the rectangle.  
    """  
    area = length * width  
    perimeter = 2 * (length + width)  
    print(f"Area of Rectangle: {area}")  
    print(f"Perimeter of Rectangle: {perimeter}")  
  
    # Original calculations:  
    # print("Area of Rectangle:", 5 * 10)  
    # print("Perimeter of Rectangle:", 2 * (5 + 10))  
    # print("Area of Rectangle:", 7 * 12)  
    # print("Perimeter of Rectangle:", 2 * (7 + 12))  
    # print("Area of Rectangle:", 10 * 15)  
    # print("Perimeter of Rectangle:", 2 * (10 + 15))  
  
    # Refactored calls:  
    calculate_rectangle_properties(5, 10)  
    calculate_rectangle_properties(7, 12)  
    calculate_rectangle_properties(10, 15)
```

OUTPUT:

```
Area of Rectangle: 50  
Perimeter of Rectangle: 30  
Area of Rectangle: 84  
Perimeter of Rectangle: 38  
Area of Rectangle: 150  
Perimeter of Rectangle: 50
```

Task Description #2 (Refactoring – Optimizing Loops and Conditionals)

- Task: Use AI to analyze a Python script with nested loops and complex conditionals.
- Instructions:
 - o Ask AI to suggest algorithmic improvements (e.g., replace nested loops with set lookups or

comprehensions).

- o Implement changes while keeping logic intact.

- o Compare execution time before and after refactoring.

- Sample Legacy Code:

Legacy inefficient code

```
names = ["Alice", "Bob", "Charlie", "David"]
```

```
search_names = ["Charlie", "Eve", "Bob"]
```

```
for s in search_names:
```

```
    found = False
```

```
    for n in names:
```

```
        if s == n:
```

```
            found = True
```

```
    if found:
```

```
        print(f"{s} is in the list")
```

```
    else:
```

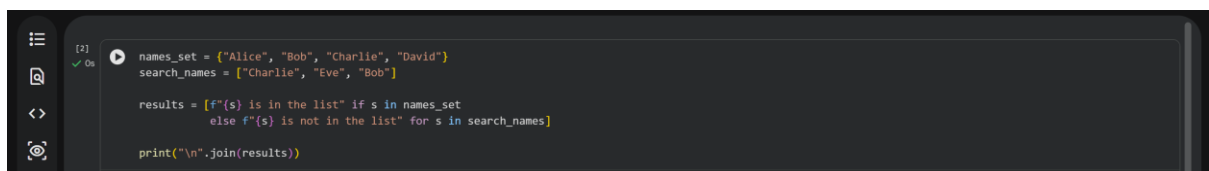
```
        print(f"{s} is not in the list")
```

- Expected Output:

- o Optimized code using set lookups with performance

comparison table.

CODE:

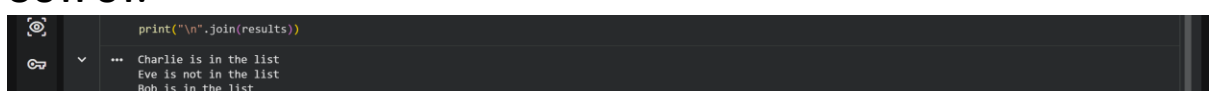


```
[2]  names_set = {"Alice", "Bob", "Charlie", "David"}
      search_names = ["Charlie", "Eve", "Bob"]

      results = [f"{s} is in the list" if s in names_set
                  else f"{s} is not in the list" for s in search_names]

      print("\n".join(results))
```

OUTPUT:



```
print("\n".join(results))

... Charlie is in the list
Eve is not in the list
Bob is in the list
```

Task Description #3 (Refactoring – Extracting Reusable

Functions)

- Task: Use AI to refactor a legacy script where multiple calculations are embedded directly inside the main code block.

- Instructions:

- o Identify repeated or related logic and extract it into reusable functions.

- o Ensure the refactored code is modular, easy to read, and documented with docstrings.

- Sample Legacy Code:

Legacy script with inline repeated logic

```
price = 250
```

```
tax = price * 0.18
```

```
total = price + tax
```

```
print("Total Price:", total)
```

```
price = 500
```

```
tax = price * 0.18
```

```
total = price + tax
```

```
print("Total Price:", total)
```

- Expected Output:

- o Code with a function `calculate_total(price)` that can be reused for multiple price inputs.

CODE:

```
[5] 0s
def calculate_total(price):
    """
    Calculates the total price including an 18% tax.

    Args:
        price (float or int): The base price of the item.

    Returns:
        float: The total price after adding tax.
    """
    tax_rate = 0.18
    tax = price * tax_rate
    total = price + tax
    return total

# Demonstrate the refactored function with sample prices

# First price calculation
price1 = 250
total1 = calculate_total(price1)
print(f"Original Price: {price1}, Total Price (with 18% tax): {total1:.2f}")

# Second price calculation
price2 = 500
total2 = calculate_total(price2)
print(f"Original Price: {price2}, Total Price (with 18% tax): {total2:.2f}")
```

OUTPUT:

```
print(f"Original Price: {price2}, Total Price (with 18% tax): {total2:.2f}")
... Original Price: 250, Total Price (with 18% tax): 295.00
    Original Price: 500, Total Price (with 18% tax): 590.00
```

Task Description #4 (Refactoring – Replacing Hardcoded Values with Constants)

- Task: Use AI to identify and replace all hardcoded “magic numbers” in the code with named constants.

- Instructions:

- o Create constants at the top of the file.

- o Replace all hardcoded occurrences in calculations with these constants.

- o Ensure the code remains functional and is easier to maintain.

- Sample Legacy Code:

Legacy script with hardcoded values

```
print("Area of Circle:", 3.14159 * (7 ** 2))
```

```
print("Circumference of Circle:", 2 * 3.14159 * 7)
```

- Expected Output:

- o Code with constants like $\text{PI} = 3.14159$ and RADIUS

= 7 used in calculations

CODE:

```
[4] ✓ On
# Constants
PI = 3.14159
RADIUS = 7

# Refactored calculations using constants
area_of_circle = PI * (RADIUS ** 2)
circumference_of_circle = 2 * PI * RADIUS

print(f"Area of Circle: {area_of_circle}")
print(f"Circumference of Circle: {circumference_of_circle}")
```

OUTPUT:

```
print(f" Circumference of Circle: {circumference_of_circle} ")
Area of Circle: 153.93791
Circumference of Circle: 43.98226
```

Task Description #5 (Refactoring – Improving Variable Naming and Readability)

- Task: Use AI to improve readability by renaming unclear variables and adding inline comments.

- Instructions:

- o Replace vague names with meaningful ones.

- o Add comments where logic is not obvious.

- o Keep functionality exactly the same.

- Sample Legacy Code:

Legacy script with poor variable names

a = 10

b = 20

c = a * b / 2

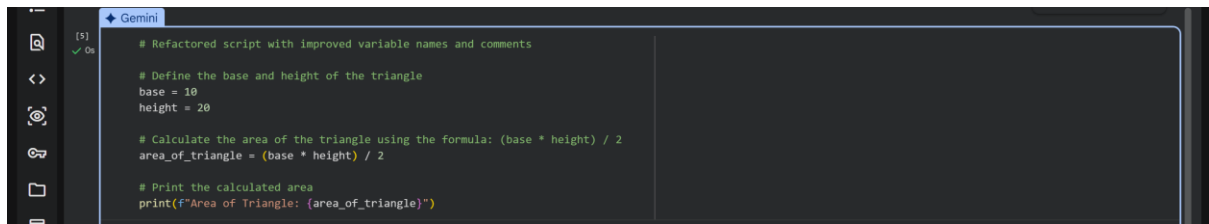
print(c)

- Expected Output:

- o Code with descriptive variable names like base,

- height, area_of_triangle, plus explanatory comments

CODE:




```
# Refactored script with improved variable names and comments

# Define the base and height of the triangle
base = 10
height = 20

# Calculate the area of the triangle using the formula: (base * height) / 2
area_of_triangle = (base * height) / 2

# Print the calculated area
print(f"Area of Triangle: {area_of_triangle}")
```

OUTPUT:



```
Area of Triangle: 100.0
```

Task Description #6 (Refactoring – Removing Redundant Conditional Logic)

- Task: Use AI to refactor a Python script that contains repeated if–else logic for grading students.

- Instructions:

- o Ask AI to identify redundant conditional checks.

- o Replace them with a reusable function.

- o Ensure grading logic remains unchanged.

- Code:

```
marks = 85
```

```
if marks >= 90:
```

```
print("Grade A")
```

```
elif marks >= 75:
```

```
print("Grade B")
```

```
else:
```

```
print("Grade C")
```

```
marks = 72
```

```
if marks >= 90:
```

```
print("Grade A")
```

```
elif marks >= 75:
```

```
print("Grade B")
```

```
else:
```

```
print("Grade C")
```

- Expected Output:

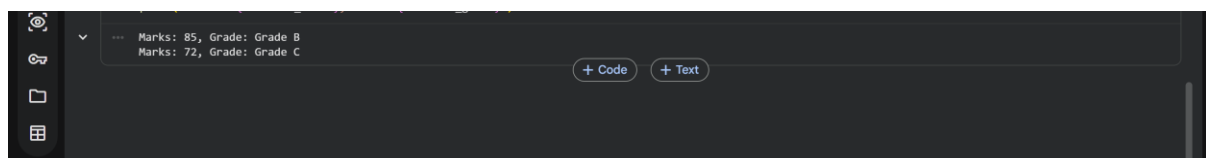
- o A reusable function like `calculate_grade(marks)` with clean logic and docstring.

CODE:



```
def calculate_grade(marks):  
    """  
    Calculates the grade for a given score based on predefined criteria.  
  
    Args:  
        marks (int): The student's score.  
  
    Returns:  
        str: The corresponding grade (A, B, or C).  
    """  
    if marks >= 90:  
        return "Grade A"  
    elif marks >= 75:  
        return "Grade B"  
    else:  
        return "Grade C"  
  
    # Demonstrate the refactored function with sample marks  
  
    # First student's marks  
    student1_marks = 85  
    student1_grade = calculate_grade(student1_marks)  
    print(f"Marks: {student1_marks}, Grade: {student1_grade}")  
  
    # Second student's marks  
    student2_marks = 72  
    student2_grade = calculate_grade(student2_marks)  
    print(f"Marks: {student2_marks}, Grade: {student2_grade}")
```

OUTPUT:



```
Marks: 85, Grade: Grade B  
Marks: 72, Grade: Grade C
```

Task Description #7 (Refactoring – Converting Procedural Code to Functions)

- Task: Use AI to refactor procedural input–processing logic into functions.

- Instructions:

- o Identify input, processing, and output sections.

- o Convert each into a separate function.

- o Improve code readability without changing behavior.

- Sample Legacy Code:

```
num = int(input("Enter number: "))
```

```
square = num * num
```

```
print("Square:", square)
```

- Expected Output:

- o Modular code using functions like `get_input()`, `calculate_square()`, and `display_result()`.

CODE:

```
[x] 155 def get_input():  
    """  
    Prompts the user to enter a number and returns it as an integer.  
  
    Returns:  
    int: The number entered by the user.  
    """  
    try:  
        num_str = input("Enter number: ")  
        num = int(num_str)  
        return num  
    except ValueError:  
        print("Invalid input. Please enter an integer.")  
        return get_input() # Recursively ask for input until valid  
  
def calculate_square(number):  
    """  
    Calculates the square of a given number.  
  
    Args:  
        number (int or float): The number to be squared.  
  
    Returns:  
    int or float: The square of the number.  
    """  
    return number * number  
  
def display_result(original_number, square_value):  
    """  
    Displays the original number and its calculated square.  
    """
```

```
[x] 155 Displays the original number and its calculated square.  
  
    Args:  
        original_number (int or float): The number entered by the user.  
        square_value (int or float): The square of the original number.  
    """  
    print(f"Square of {original_number}: {square_value}")  
  
# Main execution flow using the refactored functions  
input_number = get_input()  
squared_value = calculate_square(input_number)  
display_result(input_number, squared_value)
```

OUTPUT:

```
display_result(input_number, squared_value)  
... Enter number: AKRAM  
Invalid input. Please enter an integer.  
Enter number: 6300846174  
Square of 6300846174: 39700662508410438276
```

Task Description #8 (Refactoring – Optimizing List Processing)

- Task: Use AI to refactor inefficient list processing logic.

- Instructions:

- o Ask AI to replace manual loops with list comprehensions or built-in

functions.

o Ensure output remains identical.

- Sample Legacy Code:

```
nums = [1, 2, 3, 4, 5]
```

```
squares = []
```

```
for n in nums:
```

```
    squares.append(n * n)
```

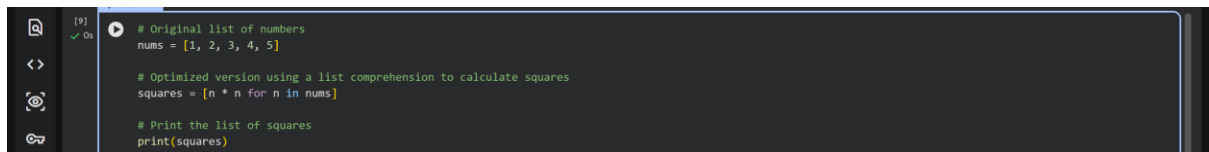
```
print(squares)
```

- Expected Output:

o Optimized version using list comprehension with improved

readability

CODE:

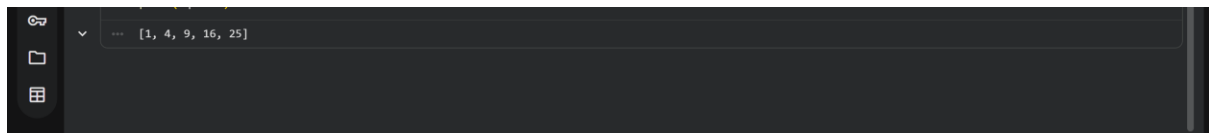


```
[9] ✓ On
# Original list of numbers
nums = [1, 2, 3, 4, 5]

# Optimized version using a list comprehension to calculate squares
squares = [n * n for n in nums]

# Print the list of squares
print(squares)
```

OUTPUT:



```
1, 4, 9, 16, 25
[1, 4, 9, 16, 25]
```