**School of Computer Science and Artificial Intelligence**

**Lab Assignment # 13.4**

**Program : B. Tech (CSE)**

**Specialization : CSE**

**Course Title : AI ASSISTED CODING**

**Course Code : 24CS101PC214**

**Semester : III**

**Academic Session : 2025-2026**

**Name of Student : E.KARTHIK PATEL**

**Enrollment No. : 2403A51416**

**Batch No. : 16**

**Date :22/10/2025**

**TASK1:**

Refactor repeated loops into a cleaner, more Pythonic approach.

Analyze the legacy code.

 Identify the part that uses loops to compute values.

 Refactor using list comprehensions or helper functions while

keeping the output the same.

**CODE AND OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**EXPLANATION:**

The code initializes a list called numbers with integers from 1 to 5. It then creates an empty list called squares. A for loop iterates through each number in the numbers list. Inside the loop, it calculates the square of the current number (n \*\* 2) and appends it to the squares list. After the loop finishes, the print(squares) statement displays the contents of the squares list, which contains the squares of the original numbers, resulting in the output [1, 4, 9, 16, 25].

**TASK 2:**

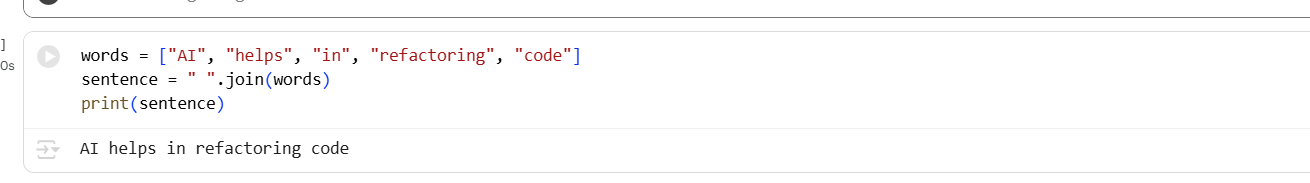
**CODE AND OUTPUT:**

Simplify string concatenation.

Instructions:

 Review the loop that builds a sentence using +=.

 Refactor using &quot; &quot;.join() to improve efficiency and readability.

****

**EXPLANATION:**

Simplify string concatenation. Instructions: • Review the loop that builds a sentence using +=. • Refactor using &quot; &quot;.join() to improve efficiency and readability. Legacy Code: words = [&quot;AI&quot;, &quot;helps&quot;, &quot;in&quot;, &quot;refactoring&quot;, &quot;code&quot;] sentence = &quot;&quot; for word in words: sentence += word + &quot; &quot; print(sentence.strip())

Okay, I can help with that! Here is the refactored code using " ".join(), which is a more efficient and Pythonic way to concatenate strings from a list.

**TASK 3:**

**CODE AND OUTPUT:**

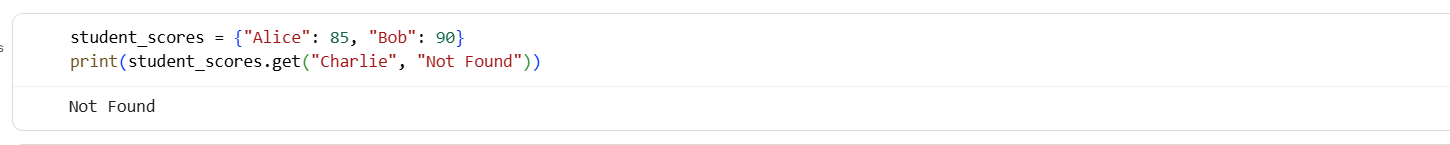
Replace manual dictionary lookup with a safer method.

Instructions:

 Check how the code accesses dictionary keys.

 Use .get() or another Pythonic approach to handle missing keys

gracefully.

****

**EXPLANATION:**

This code snippet initializes a dictionary called student\_scores with two key-value pairs: "Alice" with a score of 85 and "Bob" with a score of 90. It then uses the .get() method to retrieve the score for the key "Charlie". The .get() method is safer than direct dictionary access ([]) because it allows you to provide a default value (in this case, "Not Found") to be returned if the key is not found in the dictionary, preventing a KeyError. Since "Charlie" is not in the student\_scores dictionary, the code prints "Not Found".

**TASK 4:**

Refactor repetitive if-else blocks.

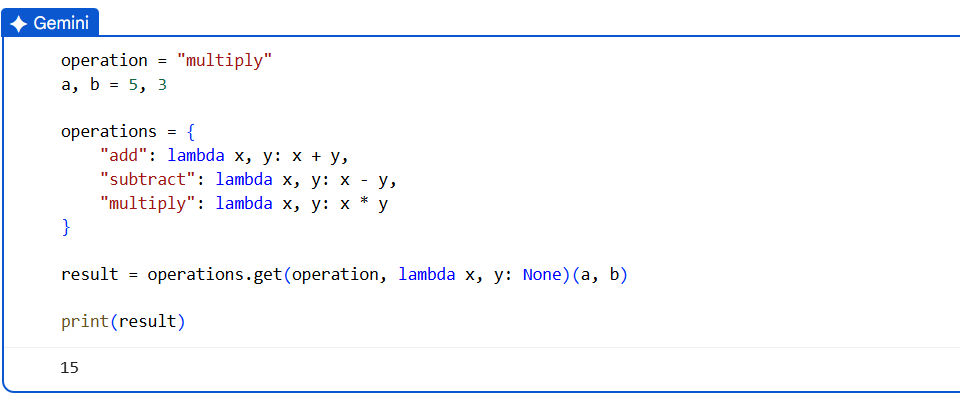
Instructions:

 Examine multiple if-elif statements for operations.

 Refactor using dictionary mapping to make the code scalable

and clean.

**CODE AND OUTPUT:**

****

**EXPLANATION:**

This code snippet demonstrates how to use a dictionary to map operation names to their corresponding functions, which is a more Pythonic approach than using multiple if-elif statements. It initializes a string variable operation to "multiply" and two integer variables a and b. A dictionary called operations is created where keys are strings representing operation names ("add", "subtract", "multiply") and values are lambda functions that perform the actual mathematical operations. The code then uses the .get() method on the operations dictionary with the operation string as the key.

**TASK 5 :**

**CODE AND OUTPUT:**

Optimize nested loops for searching.

Instructions:

 Identify the nested loop used to find an element.

 Refactor using Python’s in keyword or other efficient search

techniques.

**A white rectangular object with blue lines

AI-generated content may be incorrect.**

**EXPLANATION:**

This code snippet initializes a list called items with several integer values. It then uses the in keyword within an if statement to check for the presence of the integer 30 directly within the items list. The in keyword is a very efficient way to perform membership testing in Python lists. If the value 30 is found in the items list, the condition of the if statement evaluates to true, and the code inside the if block, print("Found"), is executed. If 30 is not found in the list, the condition is false, and the code inside the else block, print("Not Found"), is executed. In this specific case, since 30 is present in the list, the output will be "Found".