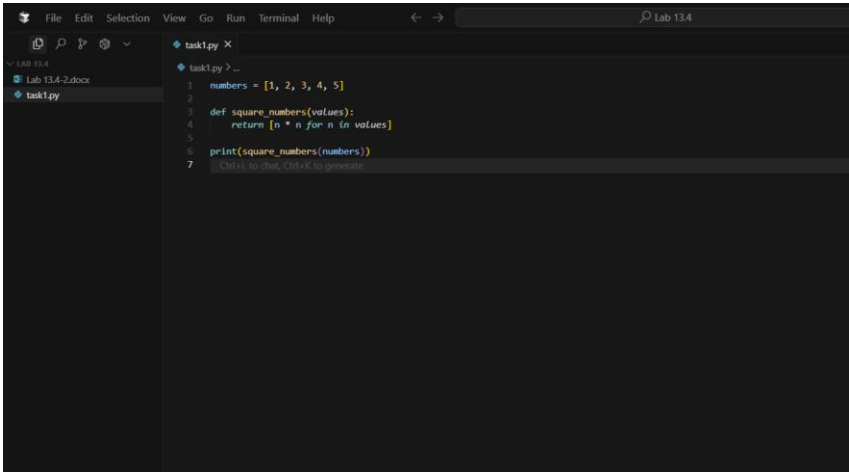
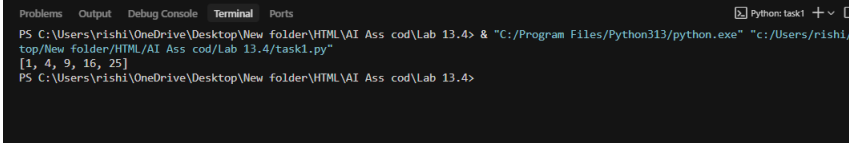


SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Course Coordinator Name		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator)	
		Dr. T. Sampath Kumar	
		Dr. Pramoda Patro	
		Dr. Brij Kishor Tiwari	
		Dr.J.Ravichander	
		Dr. Mohammand Ali Shaik	
		Dr. Anirodh Kumar	
		Mr. S.Naresh Kumar	
		Dr. RAJESH VELPULA	
		Mr. Kundhan Kumar	
		Ms. Ch.Rajitha	
		Mr. M Prakash	
		Mr. B.Raju	
		Intern 1 (Dharma teja)	
		Intern 2 (Sai Prasad)	
		Intern 3 (Sowmya)	
		NS_2 ( Mounika)	
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	II/I	Regulation	R24
Date and Day of Assignment	Week7 - Thursday	Time(s)	
Duration	2 Hours	Applicable to Batches	
AssignmentNumber:13.1(Present assignment number)/24(Total number of assignments)			
Q.No.	Question		Expected Time to complete
1	<b>Lab 13: Code Refactoring – Improving Legacy Code with AI Suggestions</b> <b>Lab Objectives:</b> <ul style="list-style-type: none"> <li>Identify code smells and inefficiencies in legacy Python scripts.</li> <li>Use AI-assisted coding tools to <b>refactor</b> for readability,</li> </ul>		Week7 - Thursday

	<p>maintainability, and performance.</p> <ul style="list-style-type: none"> <li>• Apply <b>modern Python best practices</b> while ensuring output correctness.</li> </ul>	
	<p><b>Task 1</b></p> <ul style="list-style-type: none"> <li>• <b>Task:</b> Refactor repeated loops into a cleaner, more Pythonic approach.</li> </ul> <p><b>Instructions:</b></p> <ul style="list-style-type: none"> <li>• Analyze the legacy code.</li> <li>• Identify the part that uses loops to compute values.</li> <li>• Refactor using <b>list comprehensions</b> or helper functions while keeping the output the same.</li> </ul> <p><b>Legacy Code:</b></p> <pre> numbers = [1, 2, 3, 4, 5] squares = [] for n in numbers:     squares.append(n ** 2) print(squares) </pre> <p><b>Expected Output:</b></p> <pre>[1, 4, 9, 16, 25]</pre> <p><b>CODE:</b></p>  <pre> task1.py X task1.py &gt; ... 1 numbers = [1, 2, 3, 4, 5] 2 3 def square_numbers(values): 4     return [n * n for n in values] 5 6 print(square_numbers(numbers)) 7 </pre> <p><b>Output</b></p>  <pre> Python: task1 + PS C:\Users\rishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4&gt; &amp; "C:/Program Files/Python313/python.exe" "c:/Users/rishi/top/New folder/HTML/AI Ass cod/Lab 13.4/task1.py" [1, 4, 9, 16, 25] PS C:\Users\rishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4&gt; </pre>	

## Task 2

**Task:** Simplify string concatenation.

**Instructions:**

- Review the loop that builds a sentence using +=.
- Refactor using " ".join() to improve efficiency and readability.

**Legacy Code:**

```
words = ["AI", "helps", "in", "refactoring", "code"]
```

```
sentence = ""
```

```
for word in words:
```

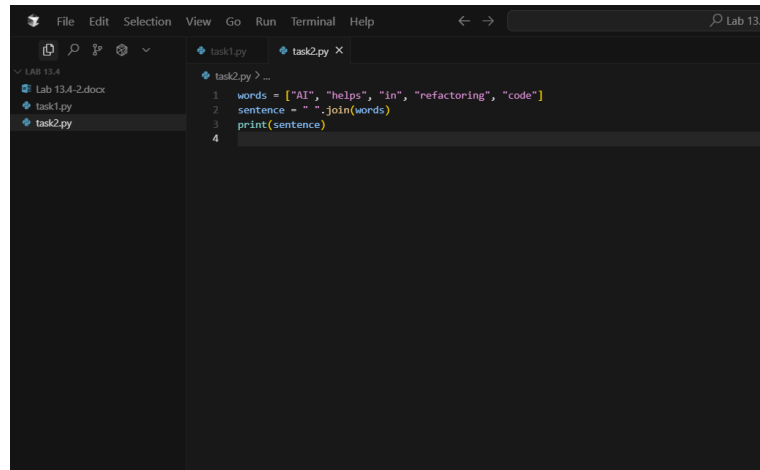
```
    sentence += word + " "
```

```
print(sentence.strip())
```

**Expected Output:**

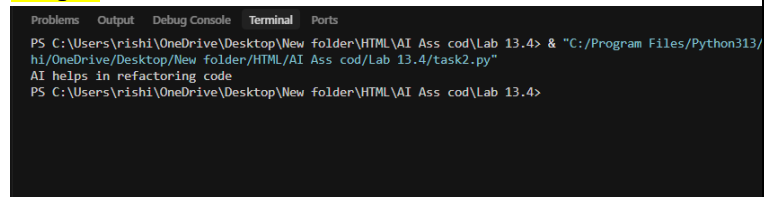
AI helps in refactoring code

- **Code**

A screenshot of a code editor window with a dark theme. The editor shows two files: task1.py and task2.py. task2.py is open and contains the following code:

```
1 words = ["AI", "helps", "in", "refactoring", "code"]
2 sentence = " ".join(words)
3 print(sentence)
4
```

**Output**

A screenshot of a terminal window with a dark theme. The terminal shows the command prompt and the output of the Python script:

```
PS C:\Users\rishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4> & "C:/Program Files/Python313/hi/OneDrive/Desktop/New folder/HTML/AI Ass cod/Lab 13.4/task2.py"
AI helps in refactoring code
PS C:\Users\rishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4>
```

## Task 3

**Task:** 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.

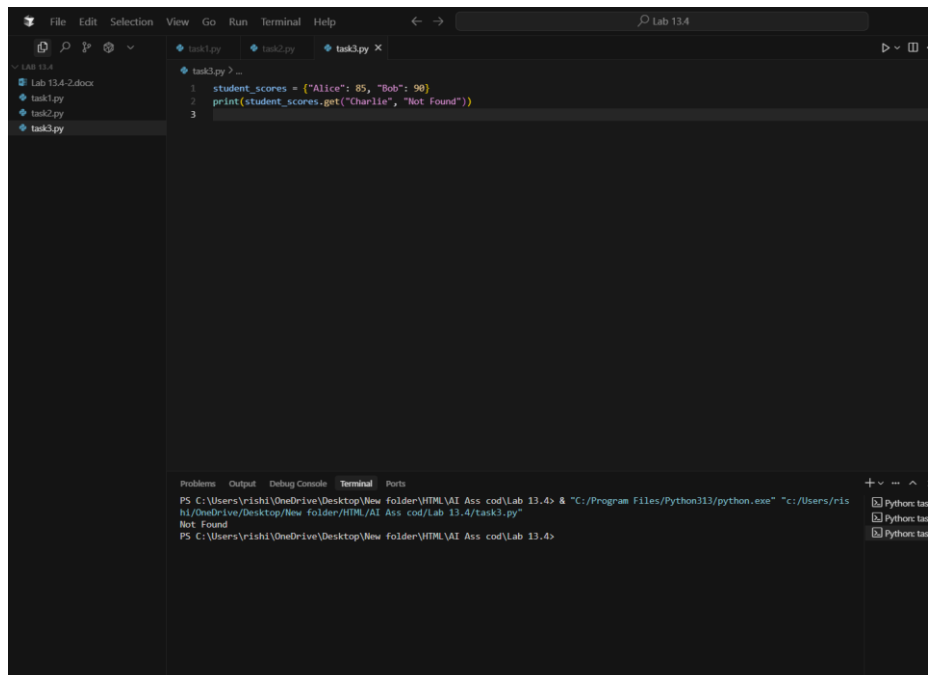
### Legacy Code:

```
student_scores = {"Alice": 85, "Bob": 90}
if "Charlie" in student_scores:
    print(student_scores["Charlie"])
else:
    print("Not Found")
```

### Expected Output:

Not Found

### Code within the output

A screenshot of a Python IDE window titled 'Lab 13.4'. The editor shows a file named 'task3.py' with the following code:

```
1 student_scores = {"Alice": 85, "Bob": 90}
2 print(student_scores.get("Charlie", "Not Found"))
3
```

The IDE has a sidebar on the left with a file explorer showing 'Lab 13.4-2.docx', 'task1.py', 'task2.py', and 'task3.py'. At the bottom, there is a 'Terminal' pane showing the command prompt output:

```
PS C:\Users\irishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4> & "C:/Program Files/Python313/python.exe" "C:/Users/irishi/OneDrive/Desktop/New folder/HTML/AI Ass cod/Lab 13.4/task3.py"
Not Found
PS C:\Users\irishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4>
```

### Task 4

**Task:** Refactor repetitive if-else blocks.

### Instructions:

- Examine multiple if-elif statements for operations.
- Refactor using **dictionary mapping** to make the code scalable and clean.

### Legacy Code:

```
operation = "multiply"
a, b = 5, 3
```

```
if operation == "add":
    result = a + b
```

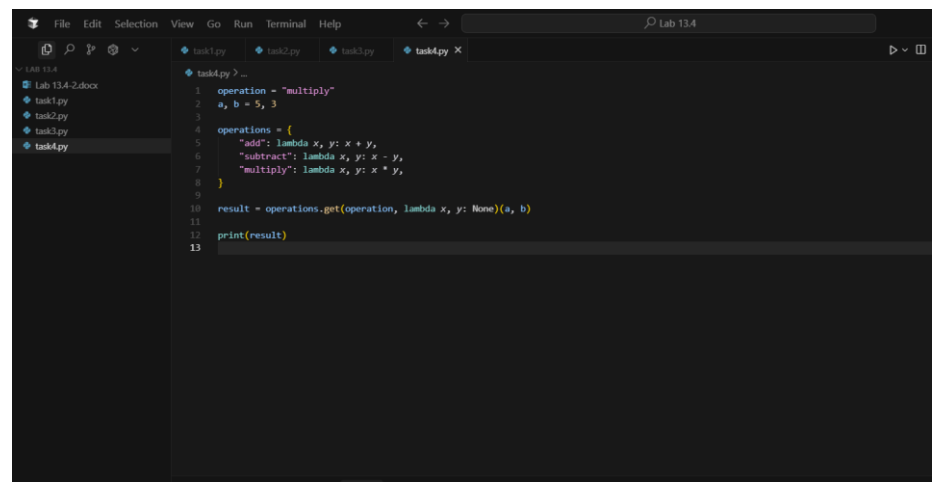
```
elif operation == "subtract":
    result = a - b
elif operation == "multiply":
    result = a * b
else:
    result = None
```

```
print(result)
```

**Expected Output:**

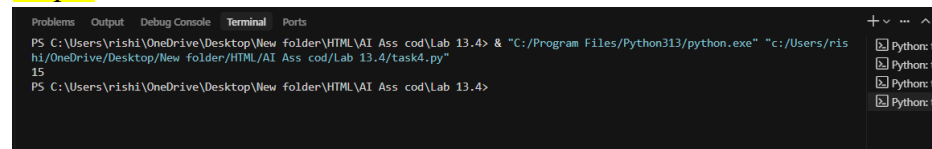
15

**code**



```
1 operation = "multiply"
2 a, b = 5, 3
3
4 operations = {
5     "add": lambda x, y: x + y,
6     "subtract": lambda x, y: x - y,
7     "multiply": lambda x, y: x * y,
8 }
9
10 result = operations.get(operation, lambda x, y: None)(a, b)
11
12 print(result)
13
```

**output**



```
PS C:\Users\rishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4> & "C:/Program Files/Python313/python.exe" "c:/Users/rishi/OneDrive/Desktop/New folder/HTML/AI Ass cod/Lab 13.4/task4.py"
15
PS C:\Users\rishi\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4>
```

## Task 5

**Task:** Optimize nested loops for searching.

**Instructions:**

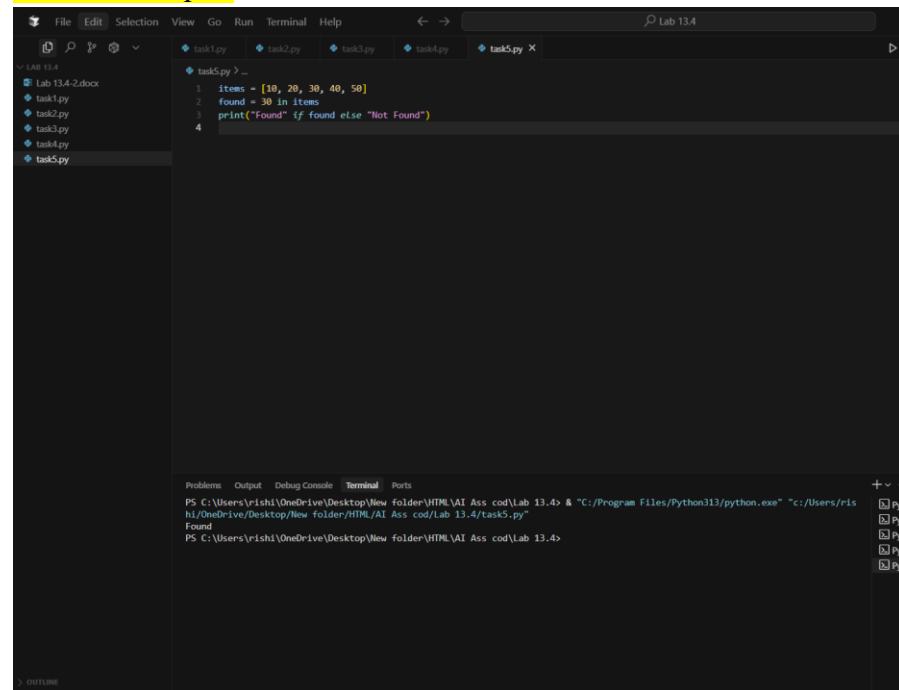
- Identify the nested loop used to find an element.
- Refactor using Python's in keyword or other efficient search techniques.

**Legacy Code:**

```
items = [10, 20, 30, 40, 50]
found = False
for i in items:
    if i == 30:
        found = True
        break
print("Found" if found else "Not Found")
```

**Expected Output:**  
Found

#### Code and output:



The screenshot shows a code editor with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The left sidebar shows a file explorer with a folder named 'Lab 13.4' containing files 'task1.py', 'task2.py', 'task3.py', 'task4.py', and 'task5.py'. The main editor area displays the following Python code in 'task5.py':

```
1 items = [10, 20, 30, 40, 50]
2 found = 30 in items
3 print("Found" if found else "Not Found")
4
```

Below the code editor, there is a 'Terminal' tab showing the execution output:

```
PS C:\Users\rish1\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4> & "C:/Program Files/Python313/python.exe" "c:/Users/rish1/OneDrive/Desktop/New folder/HTML/AI Ass cod/Lab 13.4/task5.py"
Found
PS C:\Users\rish1\OneDrive\Desktop\New folder\HTML\AI Ass cod\Lab 13.4>
```