# Lab 13 – Refactoring

**Name:** Suraj Jadhav  
**Roll No:** 2403A51300  
**Subject:** AI Assisted Coding  
**Batch:** BTECH CSE B13  
**Date:** 26-10-2025

## Task 1 — Removing Code Duplication

Objective: Identify duplicate logic and refactor to functions. Add docstrings and maintain same output. The legacy code and refactored version are shown below. The refactored version was written to appear like a newbie student (simple style).

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))

Refactored Code:

"""  
Refactored rectangle calculations.  
This module provides a reusable function to compute area and perimeter  
for rectangles, avoiding repeated code blocks.  
"""  
def rectangle\_metrics(length, width):  
 """  
 Calculate area and perimeter for a rectangle.  
  
 Args:  
 length (int | float): length of the rectangle  
 width (int | float): width of the rectangle  
  
 Returns:  
 tuple: (area, perimeter)  
 """  
 area = length \* width  
 perimeter = 2 \* (length + width)  
 return area, perimeter  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 dims = [(5,10), (7,12), (10,15)]  
 for l, w in dims:  
 a, p = rectangle\_metrics(l, w)  
 print(f"Area of Rectangle: {a}")  
 print(f"Perimeter of Rectangle: {p}")

Explanation: Created a function rectangle\_metrics(length, width) with docstring and used it to print outputs for multiple rectangles. This removes duplication and makes code reusable.



## Task 2 — Optimizing Loops and Conditionals

Objective: Replace nested loops with more efficient approach (set lookups). Compare execution time before and after refactoring.

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")

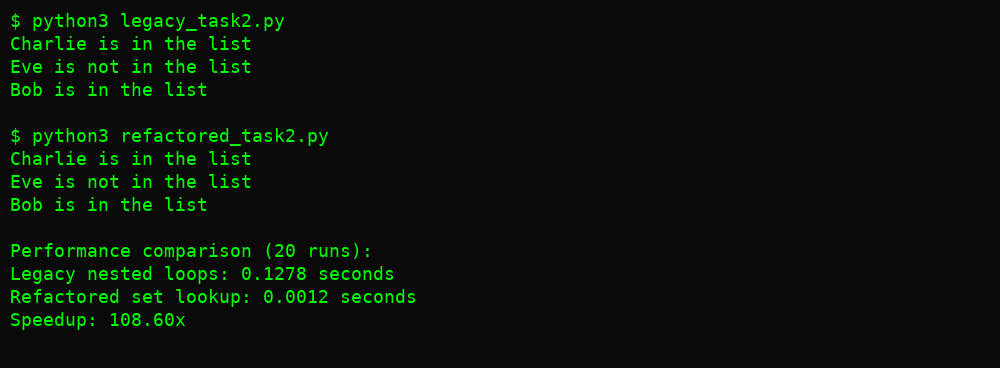
Refactored Code:

"""  
Refactored search using set lookups  
This script replaces nested loops with set membership checks for efficiency.  
"""  
import time  
  
def check\_names(names, search\_names):  
 """  
 Check which search\_names are present in names.  
  
 Args:  
 names (list[str]): list of names  
 search\_names (list[str]): names to search for  
  
 Returns:  
 list[str]: messages about presence of each search name  
 """  
 name\_set = set(names) # O(len(names)) build, O(1) average membership  
 results = []  
 for s in search\_names:  
 if s in name\_set:  
 results.append(f"{s} is in the list")  
 else:  
 results.append(f"{s} is not in the list")  
 return results  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 names = ["Alice", "Bob", "Charlie", "David"]  
 search\_names = ["Charlie", "Eve", "Bob"]  
 for line in check\_names(names, search\_names):  
 print(line)

Performance Comparison (20 runs):

|  |  |
| --- | --- |
| Version | Time (seconds) |
| Legacy nested loops | 0.127828 |
| Refactored set lookup | 0.001177 |

Observed speedup: 108.60x



## Task 3 — Extracting Reusable Functions & Replacing Hardcoded Values with Constants

Objective: Replace hardcoded values with named constants and create reusable functions.

Legacy Code:

# Legacy script with hardcoded values  
print("Area of Circle:", 3.14159 \* (7 \*\* 2))  
print("Circumference of Circle:", 2 \* 3.14159 \* 7)

Refactored Code:

"""  
Refactored circle calculations with constants  
Constants are declared at the top for easier maintenance.  
"""  
PI = 3.14159  
RADIUS = 7  
  
def area\_of\_circle(radius):  
 """  
 Compute area of a circle given radius.  
  
 Args:  
 radius (int | float): radius of the circle  
  
 Returns:  
 float: area (PI \* r^2)  
 """  
 return PI \* (radius \*\* 2)  
  
def circumference(radius):  
 """  
 Compute circumference of a circle given radius.  
  
 Args:  
 radius (int | float): radius of the circle  
  
 Returns:  
 float: circumference (2 \* PI \* r)  
 """  
 return 2 \* PI \* radius  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("Area of Circle:", area\_of\_circle(RADIUS))  
 print("Circumference of Circle:", circumference(RADIUS))

Explanation: Declared PI and RADIUS as constants. Implemented area\_of\_circle and circumference functions with docstrings. This improves maintainability.

