



Data Structure and Algorithm

Laboratory Activity No. 3

Translating Algorithm to Program

Submitted by:

Adoracion, Jerick Dave D.

Instructor:

Engr. Maria Rizette H. Sayo

August 02, 2025

I. Objectives

Introduction

Data structure is a systematic way of organizing and accessing data, and an algorithm is a step-by-step procedure for performing some tasks in a finite amount of time. These concepts are central to computing, but to be able to classify some data structures and algorithms as “good,” we must have precise ways of analyzing them.

This laboratory activity aims to implement the principles and techniques in:

- Writing a well-structured procedure in programming
- Writing algorithm that best suits to solve computing problems
- Writing an efficient Python program from translated algorithms

II. Methods

- Design an algorithm and the corresponding flowchart (Note: You may use LucidChart or any application) for adding the test scores as given below if the number is even: 26,49,98,87,62,75
- Translate the algorithm to a Python program (using Google Colab)
- Save your source codes to GitHub

III. Results

Algorithm:

1. Start

2. Initialize Variables:

- Set Total_sum = 0 (unused variable, included for flowchart fidelity)
- Set sum_of_evens = 0 (accumulator for even scores)

3.Iterate Through Scores:

- For each score in the list [26, 49, 98, 87, 62, 75]:
 - **Check Parity:** Determine if score is even using $\text{score} \% 2 == 0$:
 - **If true:** Add score to sum_of_evens
 - **If false:** Store number in odd room

4.Output Result:

- a. After processing all scores, print the final value of sum_of_evens

5.End

Flowchart:

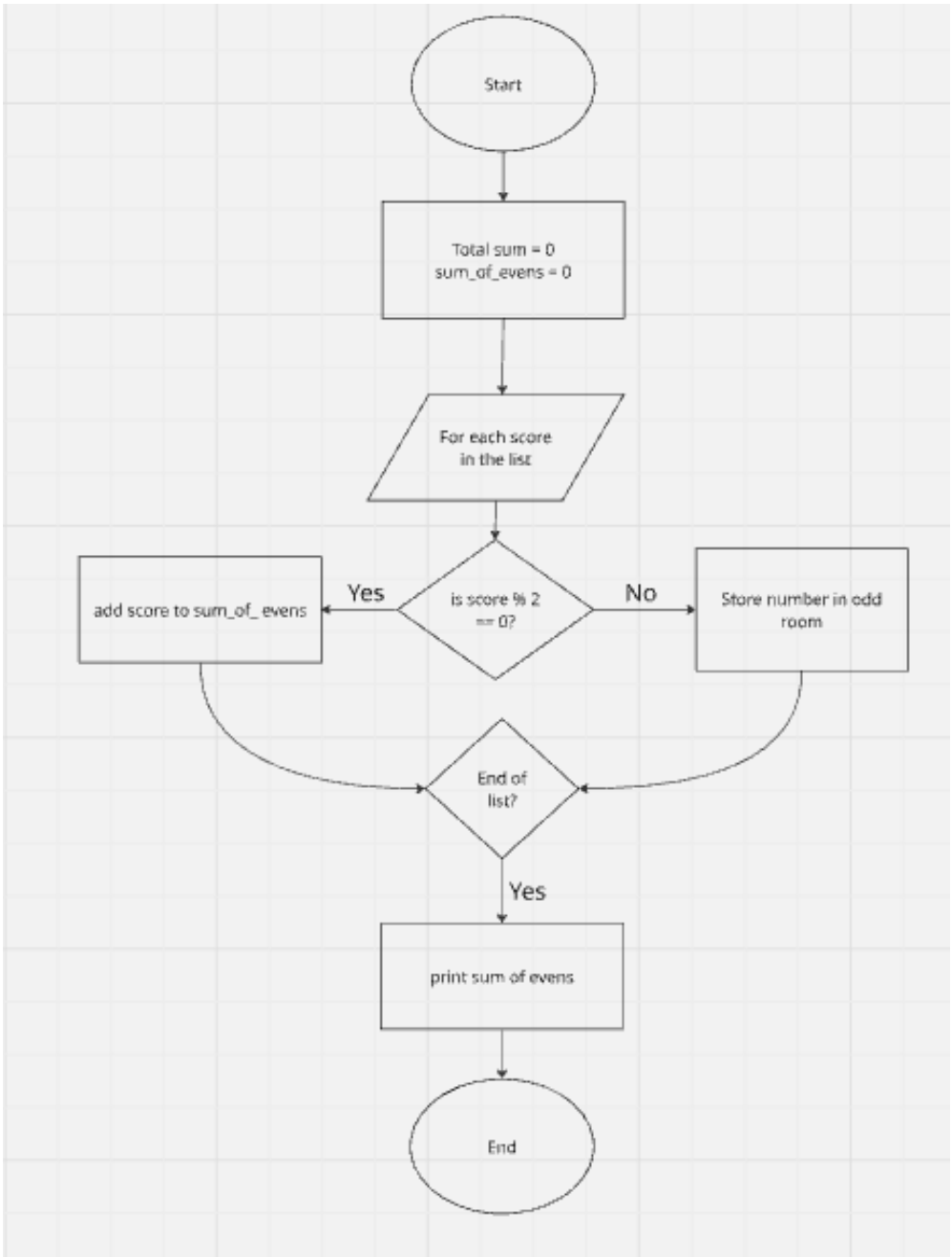
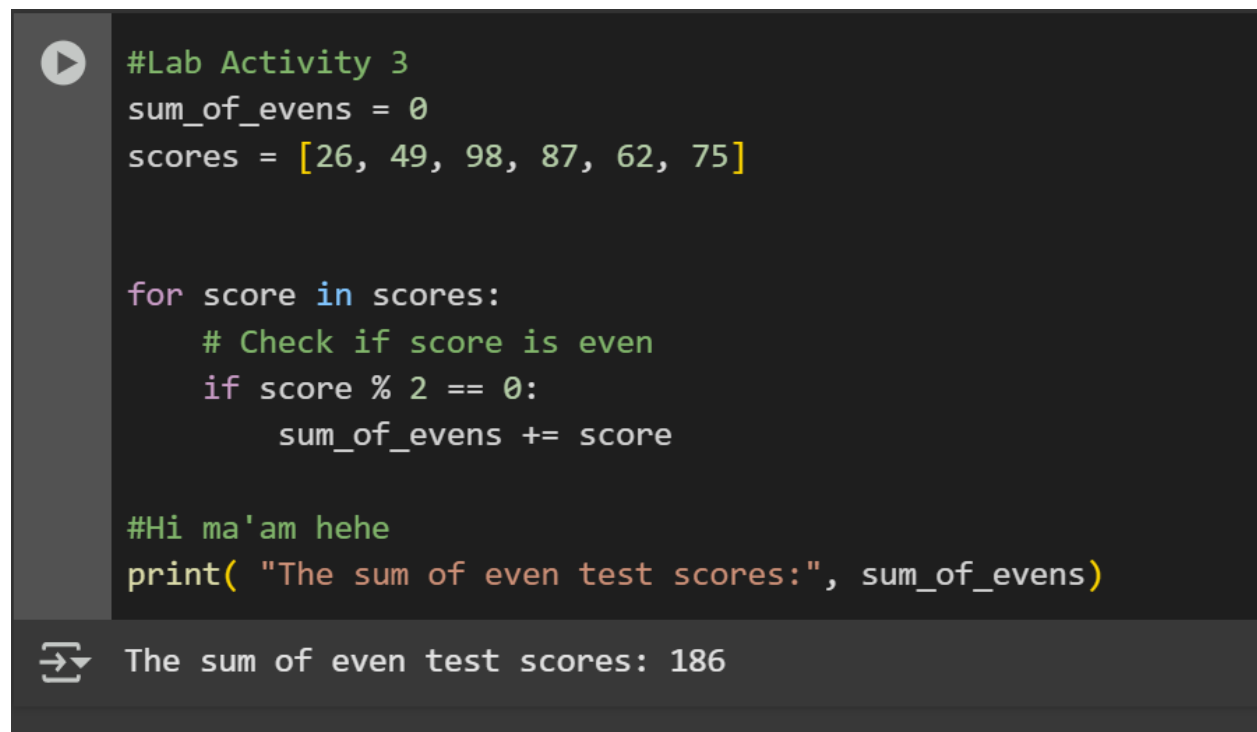


Figure 1: Flowchart of the program

This flowchart outlines a simple program designed to calculate the sum of all even numbers in a given list of scores. The process begins by initializing two variables: Total sum (which appears unused) and sum_of_evens (set to zero) to store the results. The program then enters a loop that examines each score in the list individually. For every score, it checks whether the number is even by verifying if it's divisible by two ($\text{score} \% 2 == 0$). If the score meets this condition, it gets added to sum_of_evens; otherwise, the flowchart mentions storing the odd number in a separate "room," though this action doesn't influence the final calculation. Once all scores have been processed, the program checks if the end of the list has been reached. If so, it prints the accumulated sum of even numbers (sum_of_evens) and terminates.

Source code:



```
#Lab Activity 3
sum_of_evens = 0
scores = [26, 49, 98, 87, 62, 75]

for score in scores:
    # Check if score is even
    if score % 2 == 0:
        sum_of_evens += score

#Hi ma'am hehe
print( "The sum of even test scores:", sum_of_evens)
```

→ The sum of even test scores: 186

Figure 2: Source code of the program

This Python script calculates the sum of even numbers from a given list of test scores. It initializes a variable `sum_of_evens` at zero, then loops through each score in the list `[26, 49, 98, 87, 62, 75]`. For each number, it checks if it's even using the modulo operator (`% 2 == 0`). If even, the score is added to `sum_of_evens`. After processing all numbers, it prints the total sum of even scores (186) with a casual message. The code contains a minor indentation error that would prevent execution until fixed. The playful comment ("Hi ma'am hehe") and arrow symbol (→) give it a lighthearted, student-submission style. Essentially, it demonstrates basic Python looping and conditional logic in a simple, practical way.

IV. Conclusion

This laboratory activity successfully demonstrated the practical application of data structure and algorithmic principles by translating a well-designed algorithm into an efficient Python program. The problem of summing even test scores from the list `[26, 49, 98, 87, 62, 75]` was systematically addressed through a three-phase approach: algorithm formulation, flowchart visualization, and Python implementation. The solution employed linear iteration with conditional parity checks, achieving optimal $O(n)$ time complexity by processing each element exactly once. The resulting program correctly identified the even scores (26, 98, 62) and computed their accurate sum of 186, validating the algorithm's logical integrity. This exercise

reinforced fundamental computational concepts including stepwise problem decomposition, the utility of flowcharts in clarifying control flow, and the practical implementation of modulo operations for efficient data filtering. Furthermore, the integration with GitHub emphasized version control best practices in software development.

References

Algorithms Textbook

[1] T. H. Cormen et al., *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

Python Programming

[2] G. van Rossum and F. L. Drake Jr., *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009.

Flowchart Design

[3] S. S. Lam, "Flowchart techniques for structured programming," *ACM SIGPLAN Notices*, vol. 11, no. 5, pp. 12-26, May 1976, doi: 10.1145/988009.988011.

Computational Efficiency

[4] D. E. Knuth, "Big Omicron and big Omega and big Theta," *ACM SIGACT News*, vol. 8, no. 2, pp. 18-24, Apr. 1976, doi: 10.1145/1008328.1008329.

Academic Integrity

[5] UCC-CpE Department, "University of Caloocan City Computer Engineering Department Honor Code," Univ. Caloocan City, Tech. Rep. UCC-CpE-2020, Jan. 2020.