Data Structure and Algorithm

Laboratory Activity No. 5

Implementation of Arrays

|  |  |
| --- | --- |
| *Submitted by:* | *Instructor:* |
| Calica, Ljay L. | Engr. Maria Rizette H. Sayo |

08, 16, 2025

# Objectives

Introduction

Array, in general, refers to an orderly arrangement of data elements. Array is a type of data structure that stores data elements in adjacent locations. Array is considered as linear data structure that stores elements of same data types. Hence, it is also called as a linear homogenous data structure.

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

* Writing algorithms using Array data structure
* Writing a python program that can implement Array data structure

# Methods

* Write a Python program to create an array of 10 integers and display the array items. Access individual elements through indexes and compute for the sum.
* Write a Python program to append a new item to the end of the array. Original array: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
* Write a Python program to insert a new item before the second element in an existing array. Original array: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
* Write a Python program to reverse the order of the items in the array. Original array: numbers = [5, 4, 3, 2, 1]
* Write a Python program to get the length of the array. Original array: numbers = [5, 4, 3, 2, 1]

# Results

In this arrays, there are linear data structures that store elements sequentially in memory. In this lab, I implemented key array operations in Python including creation, insertion, appending, reversal, and length calculation. These exercises demonstrated arrays' efficient indexing access while highlighting their fixed-size nature and insertion costs. The hands-on practice revealed why arrays remain fundamental for memory-efficient data storage and fast element retrieval in programming.

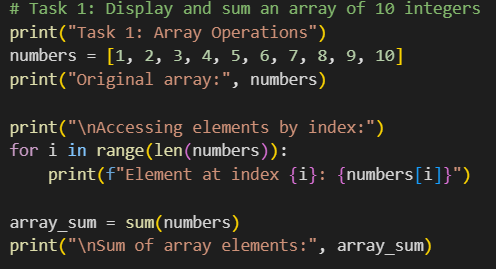


Figure 1 Screenshot of SourceCode

**Figure 1** demonstrates a Python program that creates and manipulates an array of 10 integers. The implementation first initializes the array with values, then systematically displays each element by accessing them through their indices. Finally, it calculates and outputs the sum of all array elements, showcasing the fundamental operations of array creation, element access, and mathematical computation that form the basis of array manipulation in programming.

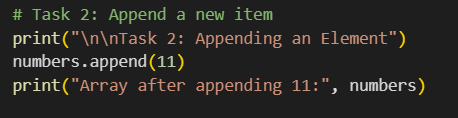


Figure 2 Screenshot of SourceCode

Figure 2. Array expansion using Python's append() operation. The figure demonstrates dynamic array modification by adding a new element (11) to the end of the original 10-element array [1,2,3,...,10], showcasing how Python lists enable efficient amortized time complexity for append operations while maintaining sequential storage.

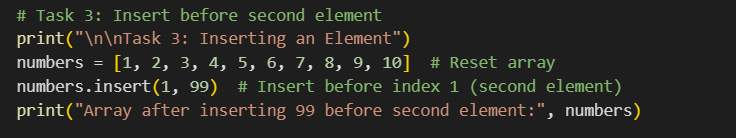


Figure 3 Screenshot of SourceCode

Figure 3. Array insertion operation demonstrating element addition at a specific index position. The program modifies the original array [1,2,3,...,10] by inserting value 99 before the second element (index 1), illustrating Python's list.insert() method and its time complexity due to subsequent element shifting. This contrasts with append operations by showing mid-array modification capabilities.

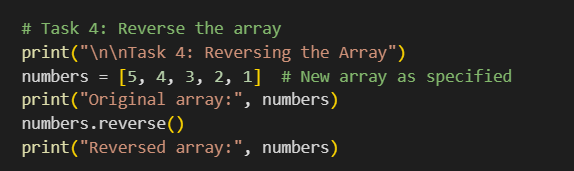


Figure 4 Screenshot of SourceCode

Figure 4. Array reversal operation demonstrating element reordering in Python. The program takes the original array [5,4,3,2,1] and applies the reverse() method to transform it to [1,2,3,4,5], showcasing an in-place modification with time complexity. This operation efficiently reorganizes elements by swapping positions from both ends toward the center.

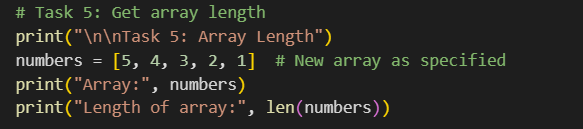


Figure 5 Screenshot of SourceCode

Figure 5. Array length determination using Python's len() function. The program calculates and displays the size of the array [5,4,3,2,1], demonstrating the O(1) time complexity operation that provides immediate access to an array's element count through Python's built-in length tracking mechanism.

# Conclusion

This laboratory activity demonstrated the fundamental operations of array manipulation in Python, including creation, element access, summation, item insertion, reversal, and length calculation. Through these practical exercises, we observed how arrays provide efficient data storage and retrieval while maintaining consistent performance for key operations. The hands-on implementation reinforced why arrays remain essential in programming—offering direct element access, predictable memory usage, and versatile modification capabilities that form the foundation for more complex data structures.

**References**

[1] Python Software Foundation. (2023). Data structures in Python (Version 3.12).

<https://docs.python.org/3/tutorial/datastructures.html>

[2] Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2019). Data structures and algorithms in Python (1st ed.). Wiley.

<https://doi.org/10.1002/9781118476734>