UNIVERSITY OF CALOOCAN CITY
COMPUTER ENGINEERING DEPARTMENT

Data Structure and Algorithm

Laboratory Activity No. 4

Arrays

Submitted by:
Adoracion, Jerick Dave D.

Instructor:
Engr. Maria Rizette H. Sayo

August 16, 2025

I. 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
- Solve programming problems using dynamic memory allocation, arrays and pointers

II. Methods

Jenna's Grocery

Jenna's Grocery List		
Apple	PHP 10	x7
Banana	PHP 10	x8
Broccoli	PHP 60	x12
Lettuce	PHP 50	x10

Jenna wants to buy the following fruits and vegetables for her daily consumption. However, she needs to distinguish between fruit and vegetable, as well as calculate the sum of prices that she has to pay in total.

Problem 1: Create a class for the fruit and the vegetable classes. Each class must have a constructor, deconstructor, copy constructor and copy assignment operator. They must also have all relevant attributes (such as name, price and quantity) and functions (such as calculate sum) as presented in the problem description above.

Problem 2: Create an array GroceryList in the driver code that will contain all items in Jenna's Grocery List. You must then access each saved instance and display all details about the items.

Problem 3: Create a function TotalSum that will calculate the sum of all objects listed in Jenna's Grocery List.

Problem 4: Delete the Lettuce from Jenna's GroceryList list and de-allocate the memory assigned.

III. Results

```python
class GroceryItem:
    def __init__(self, name="", price=0.0, quantity=0):
        self.name = name
        self.price = price
        self.quantity = quantity

    def calculate_sum(self):
        return self.price * self.quantity

    def display(self):
        print(f"Item: {self.name} | its Price is: {self.price} | its Quantity is: {self.quantity} | its Total is: {self.calculate_sum()}")

    def get_name(self):
        return self.name


class Fruit(GroceryItem):
    def __init__(self, name, price, quantity):
        super().__init__(name, price, quantity)


class Vegetable(GroceryItem):
    def __init__(self, name, price, quantity):
        super().__init__(name, price, quantity)
```

Figure 1.1: Source Code

```python
def total_sum(grocery_list):
    return sum(item.calculate_sum() for item in grocery_list if item is not None)


def main():
    # Create grocery list
    GroceryList = [
        Fruit("Apple", 10.0, 5),
        Fruit("Banana", 8.5, 6),
        Vegetable("Carrot", 5.0, 10),
        Vegetable("Lettuce", 15.0, 2),
        Fruit("Orange", 12.0, 4)
    ]

    print("=== Jenna's Grocery List ===")
    for item in GroceryList:
        item.display()

    print(f"\nTotal cost: {total_sum(GroceryList)}")
```

Figure 1.2: Source Code

As I was going through this code, I realized that it's a pretty solid example of how object-oriented programming actually makes things easier to manage. The base class GroceryItem feels like the foundation, it sets up the important details like the name, price, and quantity, and it even has methods for calculating the total and displaying everything neatly. What stood out to me is how the Fruit and Vegetable classes don't really add anything new, but by inheriting from GroceryItem using super(), they show how inheritance saves us from repeating code. I think this

could be really useful if later we wanted fruits or vegetables to have unique features, because the structure is already in place.

The total_sum function also caught my attention because it demonstrates how we can process a whole list of objects with just one line of code using a generator expression. That's something I probably wouldn't have thought of right away, but it makes the program feel efficient. When everything is tied together in main, it honestly feels like a simple version of a grocery checkout system such as listing items, showing prices, and giving a final total. Seeing it run makes the concepts of classes, inheritance, and methods feel less abstract, because I can connect it to something I do in real life, like buying groceries. Overall, working through this gave me a better appreciation for how OOP keeps code organized and makes it easier to expand later.

```
    # Delete Lettuce on the list
    for i in range(len(GroceryList)):
        if GroceryList[i] is not None and GroceryList[i].get_name() == "Lettuce":
            print("\nLettuce deleted from list.")
            GroceryList[i] = None

    print(f"\nTotal cost after removing Lettuce: {total_sum(GroceryList)}")


if __name__ == "__main__":
    main()
```

Figure 1.3: Source Code

Adding the delete feature helped me see how lists can be manipulated in Python when working with objects. Instead of removing the item completely, the code replaces it with None and makes sure the total_sum function skips those empty spots. I think this approach works fine for a simple program, but if I were to expand this project, I might want to actually remove the item from the list using del or remove() so that I don't end up with a bunch of None values. Still, this example gave me a good understanding of how deletion and recalculations can be handled in a program that mimics a real-world task like grocery shopping.

```
=== Jenna's Grocery List ===
Item: Apple | its Price is: 10.0 | its Quantity is: 5 | its Total is: 50.0
Item: Banana | its Price is: 8.5 | its Quantity is: 6 | its Total is: 51.0
Item: Carrot | its Price is: 5.0 | its Quantity is: 10 | its Total is: 50.0
Item: Lettuce | its Price is: 15.0 | its Quantity is: 2 | its Total is: 30.0
Item: Orange | its Price is: 12.0 | its Quantity is: 4 | its Total is: 48.0

Total cost: 229.0

Lettuce deleted from list.

Total cost after removing Lettuce: 199.0
```

Figure 2: Output

The program first shows all grocery items with their individual totals and the grand total of 229.0. Then it deletes Lettuce, confirms the deletion, and recalculates the total, which drops to 199.0 because Lettuce's 30.0 is no longer included.

IV. Conclusion

In conclusion, this program demonstrates how object-oriented programming can be applied to a simple real-world scenario like managing a grocery list. By using classes and inheritance, the code stays organized while making it easy to categorize items such as fruits and vegetables. The ability to calculate totals, display item details, and update the list by removing items shows how flexible and practical the design is. When Lettuce is deleted, the recalculated total reflects the updated list, which highlights how functions like total_sum can adapt to changes automatically. Overall, this example not only reinforces the concepts of classes, inheritance, and list manipulation but also shows how these programming ideas can solve everyday problems in a clear and efficient way.

References

[1] Co Arthur O.. "University of Caloocan City Computer Engineering Department Honor Code," UCC-CpE Departmental Policies, 2020.

[2] D. Amos, *Object-Oriented Programming (OOP) in Python – Real Python*, Real Python, Dec. 15, 2024. Available online. Real Python

A comprehensive tutorial on defining classes, using __init__, methods, and leveraging inheritance via super().

[3] "Python Inheritance (With Examples) – Programiz," Programiz. Available online. Programiz

A tutorial emphasizing inheritance for code reuse and cleaner subclass definitions in Python.

[4] "Inheritance in Python," GeeksforGeeks, Jul. 24, 2025. Available online. GeeksforGeeks

An article detailing how subclasses inherit attributes and methods from base classes, promoting scalability and extensibility.