

Multi Paradigm Programming – Shop Report

Introduction:

A programming paradigm is how an approach implemented to solve a problem. There are several programming paradigms. The aim of this project was to build on the functionality of the Shop programme from the lectures. This was to be completed using two paradigms, in two languages - Objected Orientated Programming (OOP) using Python and Procedural Programming using both Python and C. This report will compare the implementations of both paradigms.

Project Objective:

The programme is a simulation of a shop and processed customer orders. The following objectives should be recreated in C and Python procedurally and Python Object Orientated.

- Display shop stock and current cash.
- Reading and processing a customer order from a CSV file.
- Processing a live customer order through user input.
- Throw an error if an order cannot be completed due to stock shortage or insufficient funds.
- Update shop cash and stock as an order is processed.
- Identical “user experience” for each implementation.

Procedural Programming (shop.py and shop.c)

A programming paradigm which operates in a top-down approach. It works by calling procedures, which are the logical steps the computer should follow when conditions are met. These procedures break the programme into smaller chunks to solve the overall problem. It can be thought of in the way a recipe is written in a linear order. Procedures can call other procedures; they may be called at anytime during the programme execution.

Object-Oriented Programming (opp_shop.py)

A programming paradigm which operates in a bottom-up approach. It breaks down the problem by using objects. An object is a data structure which contains its own attributes and procedures. Object-oriented programming aims to mimic real-world behaviour. Objects are an instance of a class; a class can have many objects. We can think of a class as a skeleton, then each object derived from this skeleton can be fleshed out differently

Comparison

Similarities

1. In both POP and OOP implementations there is a way to group related variables. In oop_shop.py (OOP) we use classes and in the POP approach we use dataclasses (*shop.py*) and structs (*shop.c*), which act as the class equivalent. These containers allow for the grouping of different variable types. OOP classes act differently which we will see later in the differences, but the grouping of variables is similar.

Python OOP:

```
# Product class
class Product:
    def __init__(self, name, price=0):
        self.name = name.lower()
        self.price = price
```

Python POP:

```
@dataclass
class Product:
    name: str
    price: float = 0.0
```

C POP:

```
// Product struct
struct Product {
    char* name;
    double price;
};
```

2. Both paradigms are modular i.e., they breakdown the problem into smaller manageable blocks to solve the overall problem. This is achieved in OOP through using objects and in POP this is accomplished with procedures. For example, in each implementation there is a code which solves the unit cost of a product in a customer's shopping list, which is then used later to calculate the overall total order cost. It's best practice that these blocks solve a specific task. This code referencing is achieved through chaining, as seen in the below example and effects the flow of the programme.

Python OOP:

```
e: # Otherwise calculate unit cost (quantity x price)
cost = item.quantity * customerItem.price
```

Python POP:

```
# Get the unit cost of the product (quantity x price)
cost += item.quantity * customerItem.product.price
```

C POP:

```
// Variable to track unit cost for each item
double cost = c.shoppingList[i].quantity * c.shoppingList[i].product.price;
```

3. All three implementations use the display menu to call methods and procedures. What is executed is based on the user's choice. The main difference in the implementation is the in OOP implementation the menu is a method under the *Shop* class. Depending on the menu choice a customer object is created and passed to process their order, whereas in the POP programmes each of the separate procedures must be called.

4. In all implementations I endeavoured to avoid a live customer order not being processed due to case sensitivity. In both Python programmes the `.lower()` method was used for Product names, for the C programme `strcasecmp()` was used. The C implementation worked better as strings are compared irrespective of case whereas `.lower()` meant the Product names had to be transformed into lowercase.

Python POP:

```
if item.product.name.lower() == customerItem.product.name.lower():
```

C POP:

```
// strcmp removes any case sensitivity  
if (strcmp(s->stock[j].product.name, customerProduct) == 0)
```

5. Each implantation alerts a customer that their order is incomplete order due to insufficient funds or their order quantity is greater than the shop stock. This prevents the programme from crashing.

Differences

1. POP employs a top-down approach in which the processes must be systematic. The data and the procedures are separate. On the other hand, OOP takes a bottom-up approach where the data takes precedence of the order.
2. Another difference I found is that OOP allows for Inheritance, this allows other classes to inherit features of another and extend its functionality. This is not available in POP. The ability to inherited makes the OOP code more reusable than POP. For example, when implementing the live customer order in OOP (*oop_shop.py*), the *Live(Customer)* class is the child of the parent *Customer* class. This means it has access to the *Customer* class attributes and methods, this prevents code from having to be rewritten and makes it less error prone in the process order.

Python OOP inheritance:

```
# Live class (subclass of Customer)  
class Live(Customer):  
    def __init__(self):
```

3. In addition to inheritance, in OOP there is the concept of polymorphism. In *oop_shop.py* the child *Live(Customer)* class in allows the parent *Customer* class to extend its functionality so it can handle a live customer order and a CSV customer order. In the POP implementations this had to be written as separate procedures.
4. In the POP programmes global variables could be set, this is not possible in OOP, as the variables can only be accessed through the class member. This posed a problem for me

while trying to show the *Live(Customer)* class the shop stock list as it belongs to the *Shop* class.

5. In the similarities, the grouping of variables was mentioned but how these “containers” communicate is different. In the POP implementation, the variables in the dataclasses (*shop.py*) and structs (*shop.c*) communicate by passing data as parameters to procedures. This is different to how OOP classes operate. The classes in *oop_shop.py* each contain their own methods and attributes. They communicate by passing messages from the object to another object as the methods are from within. For example, the *ProductStock* class in *oop_shop.py* (OOP) houses its own methods whereas in both POP implementations these procedures had to be coded separately entities to the

```
# ProductStock class
class ProductStock:
    def __init__(self, product, quantity):
        self.product = product
        self.quantity = quantity

    # Method to get product name
    def productName(self):
        return self.product.name

    # Method to get product price
    def productPrice(self):
        return self.product.price

    # Method to get cost (price x quantity)
    def cost(self):
        return self.productPrice() * self.quantity

    # Method to get product quantity
    def productQuantity(self):
        return self.quantity

    # Method to update shop quantity when an item is sold
    def updateQuantity(self, customerQuantity):
        self.quantity -= customerQuantity

    # Method to get the product
    def getProduct(self):
        return self
```

dataclass/struct. For example, in *shop.py* (POP) there is a procedure to access a product’s price for a separate *printCustomer* procedure. The classes make OOP code easier to read and navigate as everything is under Class, whereas in POP procedures can be scattered throughout. For example, in *shop.c* I have all the print procedures in one section of the code, with the main functions further down. *The use of dataclasses in *shop.py* was restricted to act as a struct only for this project.

6. Another implementation difference I found between OOP and POP implementations is the print functionality. As mentioned above OOP classes contain their own methods, including the print method. This is done by a *__repr__* method which represent the class object as a string . Whereas in the POP implementations the print function is a separate procedure. I found it difficult to recreate the POP (*shop.py* and *shop.c*) user experience in OOP (*oop_shop.py*).

Python OOP:

```
# Returns customers shopping list with unit and total cost
def __repr__(self):
    print("#####\n")
    print(f'Customer: {self.name} \nBudget: {self.budget:.2f}')
    print("#####\n")
    print("\n Product\t\tPrice\tQty\tUnit Total")
    print(f"_____ \n")
```

Python POP:

```
# Function to print customer
def printCustomer(c,s):
    total = 0 # Variable to track customer order total

    print("#####\n")
    print(f" Customer: {c.name}\n Budget: {c.budget:.2f}\n")
    print("#####\n")
    print("\n Product\t\tPrice\tQty\tUnit Total")
    print("_____ \n")
```

C POP:

```
// Function to print customer
void printCustomer(struct Customer c) {
    double total = 0.0; // Variable to track customer order total

    printf("#####\n");
    printf("\nCustomer: %s\nBudget: %.2f", c.name, c.budget);
    printf("\n#####");

    // Print the shopping list.
    printf("\n\n Product\t\tPrice\tQty\tUnit Total\n");
    printf("_____ \n");
```

7. There was a difference between implementing the reading of the shop stock and customer CSV files in Python and C. In the Python programmes the `csv` module could be imported to read the CSV file, whereas in the low-level C programme, a procedure had to be created (*int my_getline*) within the programme to read the files.


Observations

- Python and C both support POP but C does not support OOP.
- POP follows a systematic approach where it is executed in order. OOP is more dynamic in that sense as it allows for simultaneous execution.
- OOP provides more functionality than POP such as inheritance and polymorphism.
- POP is better suited to solving less complex problems. OOP is better suited to solving real world problems.

Conclusion:

- Prior to this project I had not been exposed to OOP before. I can see the importance and many advantages of OOP, especially when tackling real world problems, as they are difficult to replicate using POP, yet I found it to be the most difficult to implement and get a grasp of. I will continue to develop my understanding of this programming paradigm after the project.
- Furthermore, before this module I had not used C before. I found it tricky to understand the syntax, order of operations and that a procedure can return a single datatype only. The difference was magnified when I predominately use a high-level language like Python.
- Regarding the user experience, I found it easier to replicate the same experience in POP Python and C, than recreate it in OOP Python.
- I found both paradigms to be reusable in terms of reducing the need for code duplication.
- Although I found the POP implementation and process easier to follow, OOP makes it easier to modify a programme, as you can create new objects from existing ones. Whereas in POP the entire programme may need to be changed for a simple modification, I found this related most to my C implementation. This makes OOP better suited to more complex problems and easier to scale.

References

1. Carr D. Multi-Paradigm Programming [Internet]. 2022. Available from: <https://vlegalwaymayo.atu.ie/course/view.php?id=5604>
2. How to explain object-oriented programming to kids [Internet]. FunTech Blog. 2020 [cited 2022 Dec 7]. Available from: <https://funtech.co.uk/latest/explain-object-oriented-programming-to-kids>
3. Bhatia S. Procedural Programming [definition] [Internet]. Hackr.io. [cited 2022 Dec 3]. Available from: <https://hackr.io/blog/procedural-programming>
4. Differences between procedural and object oriented programming [Internet]. GeeksforGeeks. 2019 [cited 2022 Dec 7]. Available from: <https://www.geeksforgeeks.org/differences-between-procedural-and-object-oriented-programming/>
5. Dev.w3.org. [cited 2022 Nov 25]. Available from: <https://dev.w3.org/libwww/Library/src/vms/getline.c>
6. C Language: printf function (Formatted Write) [Internet]. Techonthenet.com. [cited 2022 Nov 17]. Available from: https://www.techonthenet.com/c_language/standard_library_functions/stdio_h/printf.php
7. Code B. C return statement  BACK [Internet]. Youtube; 2021 [cited 2022 Nov 17]. Available from: <https://www.youtube.com/watch?v=HuKYb1yN7Ik>
8. C programming/C reference/nonstandard/strcasecmp [Internet]. Wikibooks.org. [cited 2022 Nov 23]. Available from: https://en.wikibooks.org/wiki/C_Programming/C_Reference/nonstandard/strcasecmp
9. Schafer C. Python OOP Tutorial 1: Classes and Instances [Internet]. Youtube; 2016 [cited 2022 Nov 24]. Available from: <https://www.youtube.com/watch?v=ZDa-Z5JzLYM>
10. Python Simplified. Python Classes and Objects - OOP for Beginners [Internet]. Youtube; 2021 [cited 2022 Nov 24]. Available from: <https://www.youtube.com/watch?v=f0TrMH9s-VE>