Shop Assignment Report

Maria Claudia Morazzoni - G00382878

This report aims to inform comparisons of the solutions achieved using the approach procedure and the object-oriented approach. But for the logical argument to make sense, I will need to briefly present both paradigms.

S.no.	On the basis of	Procedural Programming	Object-oriented programming
1.	Definition	It is a programming language that is derived from structure programming and based upon the concept of calling procedures. It follows a step-by-step approach in order to break down a task into a set of variables and routines via a sequence of instructions.	Object-oriented programming is a computer programming design philosophy or methodology that organizes/ models software design around data or objects rather than functions and logic.
2.	Security	It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
3.	Approach	It follows a top-down approach.	It follows a bottom-up approach.
4.	Data movement	In procedural programming, data moves freely within the system from one function to another.	In OOP, objects can move and communicate with each other via member functions.
5.	Orientation	It is structure/procedure-oriented.	It is object-oriented.
6.	Access modifiers	There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
7.	Inheritance	Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
8.	Code reusability	There is no code reusability present in procedural programming.	It offers code reusability by using the feature of inheritance.
9.	Overloading	Overloading is not possible in procedural programming.	In OOP, there is a concept of function overloading and operator overloading.
10.	Importance	It gives importance to functions over data.	It gives importance to data over functions.
11.	Virtual class	In procedural programming, there are no virtual classes.	In OOP, there is an appearance of virtual classes in inheritance.
12.	Complex problems	It is not appropriate for complex problems.	It is appropriate for complex problems.
13.	Data hiding	There is not any proper way for data hiding.	There is a possibility of data hiding.
14.	Program division	In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
15.	Examples	Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

Comparing solutions achieved using the procedural approach and object oriented approach:

1) Procedural programming:
   a) Procedural programming is intuitive in the sense that it is very similar to how you would expect a program to work. In this way I built my code providing step-by-step and top-down instructions.
   b) Right after I started my reasoning thinking about the first thing that should appear on the terminal as the initial requested information. The def cash() function expresses such a thought.

```
#*****Global variables*****
stock = []
stockCash = []
productName = []
productPrice = []
productQuantity = []
customerName = []
customerBudget = []
customerProduct = []
customerQuantity = []
shoppingList = []
operatorOnlineList = []
customerTestName = []
customerTestBudget = []

def cash ():
    fp = open("stock.csv","r") #*****FILE pointer which is opening the file
stock.csv as in read mode*****
    first = fp.readline()
    split_line = first.split(",")
    cash = float(split_line[0])
    #print(cash)
    stockCash.append(cash)
    #print(stockCash)
    return stockCash
```

2) Object Oriented Programming
   a) Object-oriented programming is an approach to problem solving in which all calculations are performed using objects. So I built my code first by creating classes and objects and then I could give properties to those objects so that they could do certain things..

```
class Product:
    """*****for storing product*****"""

    def __init__(self , name , price):
      self.name = name
      self.price = price

class ProductStock:
    """*****for storing ProductStock*****"""
```

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

b) In the code below we see the concepts of data in modular units, polymorphic way of executing the code and the partial encapsulation of this code and functionality.

c) A class (Customer) encapsulates a set of properties (function _init_) and behavior (class functions changeShoppingList) that can be used to instantiate an object of specific values, that is, in the initiateShopping function c = Customer(name, budget) . This is typically used to model real world objects.

```
class Shop:
    """*****for storing Shop details*****"""
    stock = []

    @classmethod
    def changeStock(cls , newStock):
        cls.stock.append(newStock)


class Customer:
    """*****for storing Customer Details*****"""

    shoppingList = []
    def __init__(self , name , budget):
        self.name = name
        self.budget = budget

    @classmethod
    def changeShoppingList(cls , item):
        cls.shoppingList.append(item)
```

```
def initiateShopping(customer , shop):
    index = 0
    totalCost = 0.0

    fp = open(customer,"r")
    if(fp == None):
        exit()     #*****if file is empty ...*exit****
    first = fp.readline()
    split_line = first.split(",")
    name = split_line[0]
    budget = float(split_line[1])
    c = Customer(name , budget)
        print("-------------\n")
    print("CUSTONMER ORDER- CSV FILE\n")
    print("Customer Name:" , c.name ,"\nCustomer Budget:" ,c.budget ,"\n")
```

```
    for line in fp:
        split_line = line.split(",")   #*****spliting the line by ","*****
        pName = split_line[0]
        if(split_line[1] != ''):
            quantity = int(split_line[1])    #*****converting to float*****
        else:
            quantity = 0

        index = checkStock(pName,shop)

        if(index != -1):
            if(shop.stock[index].quantity >= quantity):
                cStock = ProductStock(shop.stock[index].product , quantity)
                c.changeShoppingList(cStock)
                shop.stock[index].quantity -= quantity
                cost = quantity * shop.stock[index].product.price
                totalCost += cost

            else:
                print("-------------\n")
                print("We have only",shop.stock[index].quantity , pName)
        else:
            print("-------------\n")
            print(pName," not available\n")

    printCustomer(c)
    shop.cash += totalCost
    print("-------------\n")
    print("The total cost to " , c.name ," will be " ,totalCost ,"\n")
    return shop
```

Predominance of paradigms in the language that built my code.

C is a procedural programming language. It was initially developed by Dennis Ritchie in the year 1972. It was primarily developed as a system programming language for writing an operating system. Key features of the C language include low-level memory access, a simple set of keywords, and a clean styling.

```
int main()
{
    printf("-------------\n");
    printf("MENU\n");
    printf("-------------\n");
    printf("\n1 - order by csv file");
    printf("\n2 - place your order in a live mode");
    int choice = -1;

    while (choice != 0){
        fflush(stdin);
        printf("\nChoose your number (1 or 2): ");
        scanf("%d", &choice);
```

```
        if (choice == 1){
            struct Shop shop = createAndStockShop();
            printShop(shop);
            shop = initiateShopping(shop);
            printShop(shop);
            shop = initiateShoppingTest(shop);
            printShop(shop);
        } else if (choice == 2){
            struct Shop shop = createAndStockShop();
            printShop(shop);
            operatorOnline(shop);
        }
    }
    printf("Press 1 or 2");
    return choice;
```

Python is not a complete OOP language as it does not allow for strong encapsulation. That's because its creator, Guido van Rossum, wanted to keep things simple and that meant not hiding data in the strictest sense of the term. Python, however, supports all the basic features of the OOP language.

```
def menu():
    print("-------------\n")
    print("MENU")
    print("-------------\n")
    print("1 - order by csv file")
    print("2 - place your order in a live mode")

    while True:
        try:
            option = int(input("Choose your number (1 or 2): "))

            if (option == 1):
                shop = createAndStockShop()
                shop = initiateShopping('customer.csv', shop)
                printShop(shop)
                shop = initiateShoppingtest('customertest.csv', shop)
            elif (option == 2):
                shop = createAndStockShop()
                shop = operatorOnline(shop)
        except Exception as e:
            if e is ValueError or e is TypeError:
                continue
            else:
                break

    return option
```

```
#****Here  calling of function and real programming take place****#
```

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

Which one to choose, then?

Procedural code makes it easier to add new functions without changing existing data structures, OOP code, on the other hand, makes it easier to add new classes without changing existing functions.

Therefore, we can conclude that when we want to add new data types instead of new functions, for these cases the OOP approach is appropriate. Whereas, when we want to add new functions as opposed to data types, procedural code is more appropriate.

Other aspects to consider include performance vs maintainability. If you intend to have a specialized program that has no changing requirements and is designed for high performance, your efforts are best served by building with procedural program concepts.

References

1) https://medium.com/swlh/procedural-vs-object-oriented-coding-style-a25b0a78f01b
2) https://levelup.gitconnected.com/functional-object-oriented-procedural-programming-644feda5bcfc
3) https://ichi.pro/pt/estilo-de-codificacao-orientado-a-objetos-e-procedural-178511901421595