# Multi-Paradigm Programming Shop Assignment Report

Angela Carpenter G00XXXXXX

This report aims to compare the solutions achieved using the procedural approach and the object oriented approach. It looks at the key differences between the two different approaches and how this applied to this Shop simulation assignment. It also looks at the similarities between the two approaches. The Shop project was to be completed in both a procedural programming style as well as in an object oriented manner. The procedural style was written in both C and Python. The second approach was using the object oriented approach. The shop was implemented using the object oriented approach in Python. Python is a multi-paradigm programming langauge.

Programming is a method of communication between an end user and a computer and a programming paradigm refers to the style of thinking about and approaching the programming problem. Whatever paradigm you use will affect how you write and structure the code. It also makes you think about the programming problems in very different ways. Computer programs store data in variables which are mapped to locations in memory and the content of these memory locations at any point in time is known as the program's state. This state will affect the behaviour of the program. For this shop assignment, the programming problem was to build a shop program with functionality to store and update the shops starting stock and cash, deal with customer orders both using a file reading functionality and a live interactive approach, update the shops stock and cash as products were sold for cash. Programming paradigms do not refer to any language in particular. The procedural version of the code was written in both Python and C.

*Procedural programming is a programming paradigm, derived from structured programming, based upon the concept of the procedure call. Procedures, also known as routines, subroutines, or functions, simply contain a series of computational steps to be carried out*. Wikipedia[1] Procedural programming is based on the idea of calling procedures or functions to carry out some computational steps. It tends to follow a top down approach where you basically write a list of instructions to tell the computer what to do step by step. During the program's execution any procedure can be called at any time even by itself or by another procedure. In imperative programming the program describes exactly the steps the computer must take to change the state, procedural programming then groups such instructions into procedures / methods / functions to avoid repetition and to provide structure to the program. The solution to the overall programming problem is broken down into smaller problems that can be solved using functions or procedures. These solutions are then combined together to solve the overall problem or task.

In the procedural approach you consider the logical steps that would be involved and what data structures might be needed. For the shop simulation, data structures are needed to hold the cash and the name and quantity of each product in the shop as well as to hold data about the customers who interact with the shop. The customer comes in with a shopping list and some cash. If the products on the customer's shopping list match those on the shop's stock list and the customer has enough cash to pay then the program must deduct the correct amount from the customer, deduct the stock from the shop and increase the shops cash amount. Data structures are defined to hold the data such as the shops starting cash and stock as well as the customers cash. The shop must be able to deal with more than one customer. It must also be able to make decisions on the correct actions to take. Functions or procedures are written to perform the actions required of each separate task, for example to determine what products the shop has, how many items of each product does it have, how much cash it has, how to check a customers ability to pay and then how to actual process a sale. A function or procedure is a collection of instructions that perform a specific task. They specialise in a task and they prevent duplication of code. Functions can be called as needed throughout a program and while you might write the program in a linear manner, these functions can be called as needed thus turning a linear program into a non-linear program. Functions are also used to communicate with the user the current state in the program.

*Object-oriented programming (OOP) is a programming paradigm based on the concept of “objects”, which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods.* Wikipedia[2]

An object has two characteristics - attributes and behaviour. An object contains code to perform some operation on its data fields or attributes. These operations are usually known as methods. While procedural programming follows a top down approach, object oriented programming follows a bottom up approach. Objected oriented programs are designed using the concept of objects that interact with the real world such as the `Shop` *object*, the `Customer` *object*, the `Product` "object" etc.

The object oriented programming paradigm is quite a different style of thinking about and solving a programming problem from the procedural paradigm which was evident from the start when the classes are being defined. Object Oriented Programming (OOP) is about encapsulating both data and behaviour into objects. A program that uses OOP will use a number of objects that know how to perform certain actions. A *class* in object oriented programming is like a template or a blueprint. Each instance of the class is called an object and it is defined by the template of the class. While they use the same template, each instance of an object can store different state. In OOP the functions to perform actions and to update the state exist within the class itself. In procedural programming the functions to update the state of a variable or data structure exist outside of the data structure. Classes in the object oriented paradigm are more than data containers as they also contain functionality.

The procedural version of C and Python follow the exact same flow. Procedures or functions or sub-routines are used to act on data structures. Data structures are composite data types or just ways of grouping variables together to create a representation of something but they do not contain any functions or methods. This is one of the main ways in which the procedural approach differes from the object oriented approach. While the C programming language does not support object oriented programming, Python does. The dataclasses used for the Python procedural code were not to be used as classes for this exercise but instead as data containers to mirror structs in C. The `structs` in C are just data containers, the `dataclasses` used in the Python procedural version were used as data containers only for the purpose of this assignment.

Any modifications to the state of the data containers in the procedural versions are performed by methods outside of the data structures themselves. The structs in C have no actual way of holding any functionality. While they do hold state, they cannot change the state in any way. This is performed by procedures or functions that exist outside of the data containers themselves. Functions or procedures can be passed in arguments and it is up to the programmer to ensure that sensible values are passed in and that the state is updated in a way that doesn't have unexpected consequences. Functions are independent of the data structures but they can change the state of the data structures that are passed in as parameters. The output of one function can be the input of another function.

In the OOP approach, as the objects have functionality written into them you have to think about what function the object performs as you are writing it. In procedural programming you just define what the data containers hold. So a product will have a name and a price, a product stock will have a quantity as well as a product, a shop will have cash and stock. A customer has cash and a shopping list. In the procedural paradigm, data structures are used as ways of grouping individual pieces of data together or variables to create a representation of something but they do not contain any functionality at all. These data structures can be combined to make more complex data structures. In this shop simulation, a `Product` has both a name and a price, a `ProductStock` structure contains a product and a quantity, a `Shop` contains `ProductStock` containers as well as cash. These data containers do not hold any functionality whatsoever. Objects in OOP on the other hand have both state and functionality. State is some information about the entity and functionality is what the object can perform.

A programming paradigm such as object oriented programming tries to control state information and limit access to it. The state goes alongside the functionality in a container known as a class which allows for the code to be split into more logical functions. In the object oriented version, the product has the same basic data structure but it also has a way of printing a current version of itself using a `repr` representation or printing method. The `ProductStock` class has methods that can return the name of a product or the unit price of a product. The `Shop` was not just a container for the stock and cash variables, it also contained all the methods needed to perform the functionality of the shop and to change the state of the shop.

State affects the behaviour of the program. In some programming paradigms such as Object oriented programming state is intrinsic. However it can be undesirable in other paradigms as the more state there is the more unpredictable the program can be as the state has an effect on the programs behaviour. This is especially a problem in larger programs if state gets updated in unexpected and unintended ways. In both procedural and object oriented versions of the shop, the state of the shop determined the behaviour of the program. A customers order is successfully processed only if the shop has the stock and the customer has enough cash at the time the function or method is called. The cash and stock inventory is then updated.

With the OOP version, the shop class is responsible for updating its own state. The customer object is passed into the method. In the procedural version a separate function is called that takes both the shop and customer data structures as arguments, applies the logic and then makes the updates to the state. (In the project requirements the csv files were not to be updated and therefore we can assume that a customer csv file will only be read once). The customer cash was updated within the function call so after each item is processed the customer cash balance is updated. In both the procedural and OOP programs, whether an order is processed is dependent on the state of the shops stock at that time. Any previous customer orders may have reduced the stock and therefore a later customer order might only be partially filled or not filled at all. The current state of the shop determines the outcome. I didn't encounter any problems with state getting updated in unexpected ways in the procedural programs but this was only a small program dealing with a single customer at a time and a small stock. It might be an issue if there were simultaneous processing of orders I imagine.

Procedural programming breaks a task down step by step into a collection of variables and /or data strucures and separately the functions or procedures that contain a sequence of instructions to follow. Each instruction is carried out in order in a systematic way so that a computer can understand exactly what it has to do. The first function defined in the procedural program was a procedure to create a shop data container. This included the steps to read in values from a csv file and from these values create a representation of the shop with cash and stock. Another function was defined to create a representation of a customer read from data in a customer csv file. This same function could be called repeatedly on different csv files to create representations of different customers. In my programs neither of these two csv reading functions took in any arguments as they shop had only one csv file. The function to read a customer csv file could alternatively have been defined with an argument to take the name of the file entered elsewhere in the program but I decided to keep the main program and menu as simple as possible.

In the OOP version the definition of the class contains all the information to do with the object. This includes how an object is created and the data that it holds. The code to read in the shop stock csv files is part of the `Shop` class. The code to read in the customer csv files is part of the `Customer` class. The code to create product stock is part of the `ProductStock` class. In the procedural version the code to create individual product stock from both the customer and stock csv files is done by passing the different pieces of information into a `Product` container and combining products and prices to create `ProductStock` data containers. This part of the code was very similar to the OOP code except it takes place within the shop or customer object.

In the procedural code, several printing functions were defined which were called in various places throughout the program to print the current state of a variable or data structure such as the shop cash, stock, customer shopping list. In order to print out something, these functions had to be passed in something. For example passing in a customer as a parameter to the print customer method would print the current state of that customer including their cash and their shopping list of products and quantities required. A function to print the current shop state takes in the shop as a parameter and prints the current state of the shops stock and cash. A function to print out the product name and price was called by other print functions that printed out the shop details and the customer details. This demonstrates how problems can be broken down into smaller problems. Repetition is avoided as the functions or procedures can be called when needed even by other functions or procedures. The functions to print customer and shop details are called several times themselves in response to user interaction. The shop print function will always print out the current state of the shop, both the cash and stock. Whenever a sale is processed the cash is updated and the stock is updated.

In OOP every object has a method to print a representation of the object. The `Product` class had a `repr` method to print the product's name and price. The `ProductStock` class has a print method to print out the product stock which reuses the functionality from the product class. The `Customer` class has its own method to print out the customers cash and shopping list containing product stock. If a printing method is not defined for a class it automatically inherits the print method from its parent class or higher up the tree.

In both OOP and procedural paradigms some duplication of code was avoided by reusing the functionality for printing product and product stock. For example printing the shop's stock uses the functionality for printing the product as does printing the customer shopping list. In procedural program the function to print the products is repeatedly called with the different product items passed in as parameters. In the OOP approach the shop and customer are passed in `ProductStock` class instances and therefore the methods to print the products are also passed in and don't need to be explicitly called in the customer and shop printing or `repr` methods. This makes for more concise code overall.

There are some similarities between OOP methods and procedures or functions in procedural programming. One of the main differences is that a method belongs to a specific object.

The classes and objects in OOP are used to create models based on the real world environment. It is considered easier to modify and maintain existing code as new objects can be created that can inherit characteristics from existing objects.

In the procedural programming paradigm, the relationships or communication between data are defined by calling functions or procedures with arguments during the program execution. The data structures themselves do not contain any functions or methods. Any modifications to the state of the data structures is done by methods outside of the class. Communication in OOP is achieved by message passing from one object to another. This is an important feature in object oriented programming and is key to how the object oriented programs are built. Methods are called on an object with the receiver object being the object the method is called on. The message is sent to the receiver object to perform the method with the given input. For example when processing a customer order from a csv file, a customer object `c` is first created by calling the `Customer` class. The method to calculate the cost is then called on the customer object with the shop stock being used as an input, for example `c.calculate_costs(self.stock)`. The same functionality is achieved in the procedural version by first creating a customer using the function to read in a customer csv file, then calling another functions which takes in a customer and a shop data structure.

Message passing and method chaining in OOP makes it easier to write smaller methods that perform a simple single task. The procedural code has one big function to check if the shop stocks an item, check the customers cash balance, checks the actual quantity available in stock and then to actually update the shop stock and customer cash. This makes it more difficult to amend.

The procedural version of the code was quite straightforward with Python. The C version followed the same flow. Besides the syntatic differences between Python and C, there are other differences in the way the code is written. C is far more low level. The code blocks in Python did not necessarily have to be in a particular order for the code to run. In C however this is very important. In C, the code in the `main` method is the first to be executed. This `main` method goes at the end of the script. There is more flexibility in Python as the code can be moved around without affecting how the code runs. It is recommended to keep the script component at the end of the file as this is good programming practice. In C absolutely every variable needs to be declared before it can be used. Memory allocation must be considered. The return type from a function has to be defined. There are no strings and therefore you need to work with arrays of characters and dynamically allocate enough memory.

In the object oriented code all the code relating to a particular class is kept together. The definition of a class contains all information to do with that object, including how it is created and the data that it holds. The `__init__` method in Python is the first method defined for each class and is known as the constructor method. The constructor method creates new instances of a class. The classes for Product and ProductStock were first defined. The customer class was then defined with all its associated methods and then the shop class. This made it easier to navigate the code in OOP. In object oriented programming it is sometimes recommended to keep the classes in different files. This is necessary in languages such as Java. Some of the classes for this program were very small. The Customer class defined how a customer was instantiated from reading in from a customer csv file. A customer object is created by reading in the csv file containing the shopping list and the customers budget. It also had its own methods for calculating the costs to the customer. The class definition also defines how the objects data are updated by methods.

There are several other concepts that are unique to the object oriented programming paradigm. Inheritance is the primary relationship that exists between classes in OOP. When a class inherits from another class it gets all the functionality of that class. This promotes reusability of code. Additional functionality can be added though to the subclass. In the OOP shop the classes for `Product` and `ProductStock` were first defined. `Product` class was given a `repr` representation method which `ProductStock` uses whenever it needs to return a representation of a piece of `ProductStock`. The `Shop` object inherited from the `ProductStock` object which in turn inherited from the `Product` class. This allowed access to the `Product` class and it's methods for printing and for calculating costs and stock control. The `ProductStock` class has several methods that allow you to access its attributes to get product name, prices. It also has methods for calculating its own cost using its own price and quantity. The `ProductStock` class in the OOP version is an example of encapsulation. It is responsible for calculating costs as it knows the price and can be used by both the items on the shopping list of the Customer class and also the shop stock items in the Shop class. The ProductStock calculates its own cost and returns it back to whoever called the method. This encapsulates the calculation of the cost inside the class so that the other classes don't need to work out the cost themselves.

There are some similarities to the procedural and object oriented approaches to writing the shop code. The same type of functionality as in the procedural approach was used for reading in from the stock and customer csv files using the `csv` module and iterating through the lines of the csv file, extracting the pieces of data.

Calculating the cost to the customer in the procedural approach is achieved by passing a customer and the shop into the print customer function. Both approaches uses loops to iterate through the customer shopping list comparing to the shop stock list and getting the price of the product. Then if else statements to check if the quantity desired by the customer is in stock and if so the total cost is compared to the customer budget to see if they can afford the purchase.

In both versions of the code, the cost to the customer is calculated separately to the actual order being processed. This is just to print the customer order and what it would cost them and replicate how an order comes into a shop and is reviewed before it is actually processed. In both procedural and object oriented approaches, there is no change to the shop or customer state until the order is processed.

A separate method or function in both approaches is used to actually process the order and update the state of the shop cash, the customer cash and the shop stock. In the procedural approach, the current shop state and the customer are passed into an order processing function. There is similarity again in the way the functions actually iterate through the customers shopping list, checking if the item exists, if there is sufficient stock to meet the order and checking the customers cash balance. In the OOP version though, several smaller methods exist to perform smaller tasks such as checking the stock and updating the cash which are called by the order processing method. The order processing methods belongs to the shop class in the object oriented approach. The shop is in charge of processing the order and updating the state of the shop cash and stock. The customer object is passed into the shop object as input to the process order method.

In both versions the Shop is created from a `main` method. In the procedural version a function is called to create and stock a shop from the csv file. In the object oriented approach, a shop object is instantiated or constructed from reading in the data from the stock csv file. The shop is only created once in each program when the stock csv file is read in. Once a sale has been processed, the shop's state is updated. The main method is kept as clean as possible in both approaches but it is much more concise in the OOP version. In the OOP code, the main method the Shop object is instantiated using the `stock.csv` file. Then the Shop's display_menu method is called and from there all other methods are called. The main method then is really a chain of method calls. The main method in the procedural versions is really just the interface for the user. It calls a separate function to display a user menu and from there all the functions are called. In the object oriented approach, the `Shop` class has a method to display a user menu. A shop would normally own the interface to the customers as such. Once a shop object has been created, the shops display menu method can be called to print the display options. `self` then in the display menu is referring to the Shop object. In the procedural approach, the display menu is a function that is called by the main method, while the user options are part of the main method. The OOP approach here more closely reflects real world objects.

The `Live` customer option in the procedural version is dealt with by a function to read in the customer name, budget and the item they wish to buy from the user keyboard input. The `Customer` data containers tied the three items together. A `ProductStock` is then created from the product and the quantity and a shopping list is a list of product stock. The function returns a customer data container with their name, budget and shopping list. Once this customer data container is created it is passed into another function to print out the customer order and another function to actually process the order in the same way that batch customers are dealt with.

The `Live` customer option in the OOP version is dealt with using a sub-class of the `Customer` class. This `Live` customer class extends from the `Customer` class and inherits the behaviours and functions of the `Customer` class but takes the customer name, product and quantity desired from the user input. Once the live customer object has been created it works in the same way as a customer instance creating from initialising a customer object from reading the data from the csv file. It has inherited the behaviours of the `Customer` class. A `super()` function can be used inside the `__init__()` method of the child class to run the `__init__()` method of the parent class inside the child class. However the only instance attribute that are the same is the customer shopping list. The customer name and budget are read from keyboard input instead of from a csv file. The child class `Live` modified the behaviour of the `Customer` class and extended the behaviour of the parent `Customer` class by being able to deal with a live customer. All of the `Customer` methods were then available to the `Live` customer class. The `Live` customer class can have unique attributes and behaviours that represents its differences from the parent class.

Initially I found the procedural python the most straightforward of the three versions of code as it is the style of coding that I have been used to writing. This was my first time using the Python dataclasses which was made somewhat confusing by restricting their use to being simply data containers. The procedural code was much simpler to write in Python than in C. This was my first time using C and I found it very time consuming and cumbersome. As C is much more low-level you have to declare every single variable before you can use it, memory allocation must be considered, strings have to be constructed out of characters. This made the program quite long. Everything had to be in the correct order as it was important that any procedure was defined before it was called. This meant a lot of scrolling up and down the screen. While you don't need to have the same order in a python version of the program, it makes the program easier to read.

The object oriented approach is easier to maintain once it is written. I did find it easier to amend a method later, for example when breaking up the order processing method into smaller methods that had a specific responsibility. The functions for processing a customer's order in the procedural code was quite long and care had to be taken to make sure the code was properly indented in Python or closed correctly using `{}` in C.

It is recommended that each class has its own file. The `Shop` class was the biggest class while the `Product`, `ProductStock` and `Customer` classes were not so big. I kept them in the same file to follow the sample submission files for the project. The classes actually made it easier to find things, particularly when compared to the procedural C version.

There are several concepts that are unique to OOP such as polymorphism, encapsulation and inheritance which allow for the shorter code to be written as code can be reused. Inheritance is a way of creating a new class that uses details of an existing class without modifying the existing class. A Live customer class was created that is a derived class or child class of the Customer class. The existing Customer class is the base class or parent class. Polymorphism is an ability in object oriented programming to use a common interface for multiple forms / data types. With polymorphism, different forms of the same object can be created, for example a live customer and a csv customer could be defined as a subclass of a Customer class where the child object would take after the parent object but can have additional functions or properties. The child class inherits from the parent class but can have additional functionalities. The parent class could be defined for all customers and child classes can be used for different subcategories of customer under the customer class template. Unique attributes and behaviours can be added to a child class to represent the differences from the parent class.

I defined the `Live` customer as a separate class to the `Customer` class where the `Customer` class is the parent class and the `Live` customer inherits the functions of the parent `Customer` class. The child class `Live` modified the behaviour of the `Customer` class and extended the behaviour of the parent `Customer` class by being able to deal with a live customer. All of the `Customer` methods were then available to the `Live` customer class. The `Live` customer class can have unique attributes and behaviours that represents its differences from the parent class such as being able to take user input from somewhere other than a csv file. The customer's details are taken from the user input instead of from the csv file for the parent `Customer` class. The `Live` customer has the inherited behaviours of the parent class but can have behaviour unique to subclasses.

Encapsulation provides for access control on a variable so that values cannot be changed from outside of the class. Methods to modify the state of a variable has customised logic to avoid unsuitable values being applied. Private attributes can be denoted by using an underscore _ or __ as the prefix. Private methods were not required for this project. They could be used perhaps if you were setting new prices for products.

### Summary of key differences and similarities of the two approaches for this project.

- Procedural programming follows a top down approach whereas object oriented programming follows a bottom up approach that is built around the object.
- Procedural programming uses a step by step approach to break down a task into a collection of variables and routines or procedures through a sequence of instructions. Each instruction is carried out in order in a systematic way so that a computer can understand what to do.
- In both approaches, the code was written using reusable blocks of code using functions / procedures in the procedural approach and methods in the OOP approach. Functions and methods are called throughout the program. The concept of OOP in Python focuses on creating reusable code.
- The class methods are part of the same entity as the data structure. They both belong to an object. In the procedural approach the data structures have to be passed into functions.
- In procedural programming, the program is divided into smaller parts called functions whereas in object oriented programming the program is divided into smaller parts called objects which are instances of classes.
- Functions / procedure specialise in a task and prevent duplication of writing code. Methods in OOP prevent duplication of code. Inheritance in OOP further prevents duplication of code by extending an existing class.
- In the procedural programming paradigm, functions are independent of the data structures but data can be passed into functions as arguments - the programmer has to make sure that the values passed in make sense and therefore that the state is updated in a way that doesn't have unexpected consequences.
- In OOP, the object is the focus of the program. The methods are part of the same entity as the data structure. Methods and data attributes belong to an object. The definition of a class contains all information to do with that object. All of the updates to the shop state were taken care of by the `Shop` class itself using the Shop class order processing methods.
- Object oriented programming can be used to model more complex things as simple stuctures that are easier to reproduce. Reusable objects can be imported into other programs.

- It is easier to add new data and functions in OOP than in the procedural approach.
- There is no real way to hide data in procedural programming so it is less secure than OOP which does provide data hiding which makes it more secure. Abstraction in OOP allows data hiding. This was not required in this project.
- Encapsulation provides for access control on a variable so that values cannot be changed from outside of the class. A class method can have customised logic built into it so that unsuitable values cannot be applied. Encapsulation makes the data more secure.
- Communication in the procedural approach uses functions which are called as required during the program execution. The output of one function can be the input of another function.
- Communication in OOP is through message passing from one object to another. Methods can be called on a receiver object. The message is sent to the receiver object to perform the method with the given input. Method chaining allows smaller methods to perform simple dedicated tasks.
- Object oriented programming is usually less complex due to the modular nature and new objects can be created from existing ones. Objects can be reused across a program.
- It is easier to modify an OOP program than a procedural program, particularly a C program where the whole program might need to be revised to faciliate a small change in functionality.
- As a class in OOP contain everything relevent to the class, it can be easier to debug the code if something goes wrong.
- Both paradigms allow for code duplication to be avoided, using classes in the object oriented paradigm and through the use of passing arguments into functions in the procedural paradigm.
- In both procedural and the object oriented approaches, the state affects the behaviour of the program.
- Printing in part of an object in OOP. Each class has a representation method. In the procedural programs, printing is a seperate function.
- The OOP code is easier to navigate as everything related to a class is contained together whereas in the procedural versions and particularly the C version, the code can be located throughout the program.

## References

- [1] Website: Procedural Programming Wiki at https://en.wikipedia.org
- [2] Website: Object Oriented Programming Wikie at https://en.wikipedia.org
- Website: The differences between procedural and object-oriented programming at https://www.geeksforgeeks.org
- Website: The differences between procedural and object-oriented programming at https://www.tutorialspoint.com
- Website: What is Object Oriented Programming? OOP Explained in Depth at https://www.educative.io
- Website: Python Object Oriented Programming Tutorial at https://www.programiz.com
- Website: What is Polymorphism in OOPs programming? at https://www.edureka.co
- Website: Python's Instance, Class, and Static Methods Demystified at https://realpython.com
- Website: Check if File Exists in C at https://www.zentut.com
- Website: How to prepend to a string in C at https://www.linuxquestions.org