MULTI PARADIGM PROGRAMMING – SHOP ASSIGNMENT

Keith Ryan (G00387816)

Assignment Overview

The main aim of this assignment is to demonstrate the same program that simulates the behaviour of a shop but approached 3 different ways. Across the 3 programs the behaviour should be the same but necessarily the implementation will differ.

The intent behind this approach is to show the differences and similarities across the programming languages and paradigms used.

This report will give a brief overview of each approach and then analyse the similarities and differences across the different approaches, finally concluding with some observations of the different approaches and my expectations.

The programs were written using C and Python and the approaches are:

- Procedural Programming using C
- Procedural Programming using Python
- Object Oriented Programming using Python

Shop in C (Procedural Programming)

The first (and most challenging) aspect of this assignment was to create a shop using the C programming language. Structs are used to represent the data for the shop and customers, these encapsulate the types of data to track from product level through to the customer and shop. Populating data into the structs is done using functions to take either an input from csv files or via the user entering details based on prompts. On running the code, the user is presented with a set of print statements prompting them to select what they would like to do by entering in a value in the terminal, this then controls what functions get called from elsewhere in the code, such as printing the details of the shop or loading in a customer and attempting to fulfil their order.

The main complications with this approach are down to how C handles memory and declares variables, this is a significant departure from the other approaches which are dynamically typed (Python variables can freely change type and size in memory). Where this exhibits itself as a problem is typified in reading the csv's for the customers and the shop, when reading the csv, you don't know how large a word there will be for the product name, so the product name's variable needs to dynamically get allocated memory to it.

Further adding to this complexity is the use of pointers, where when reading a product name out of the csv a set of pointers is first pointed to an allocated location of memory, then one of these pointer variables is used to point at the text being read in from the csv, this is then copied out to the other pointer variables allocated memory. Which coming from other higher-level programming languages is a confusing process as typically you don't worry about pointers, memory allocation, etc.

Shop in Python (Procedural Programming)

After creating the shop in C, it was recreated using a similar approach with the Python Programming Language. This is structurally similar to the C approach, and all of the functions used are replicated. Instead of structs the python approach uses dataclasses, which functionally serve the same purpose; a way to structure what data we want to represent the shop, customers, it's products etc.

Unlike the approach using C, memory allocation and pointers are not a concern, though we still do need to worry about type conversion as data read in from csv or command line needs to be converted from string/character to it's appropriate data type like int or float.

Shop in Python (Object-Oriented Programming)

For the last iteration, the shop was recreated again in Python but this time following an object-oriented programming paradigm. This approach replaced the dataclasses with attributes in the classes but now the classes themselves also contain the functions for loading in data and for interacting with the other classes, e.g., a customer class being passed to a shop classes fulfill_order function which updates the shop classes data.

Much of the code was repurposed from the original python implementation, with the functions having their input arguments changed to include "self" and having them embedded in classes which alter the data within their class as opposed to being external functions to the classes having their state changed as was done in the procedural approach.

Comparing the approaches

To compare the approaches, it isn't practical or sensible to compare each implementation to one another, instead the "Shop in C" will be compared against the "Shop in Python" to show the similarities and differences between implementations across different languages but following the same programming paradigm.

Comparing Object-Oriented and Procedural programming will be done with the python implementations.

Procedural Programming language comparison

Similarities:

The main similarities between the approaches in C and Python are in how the functions are structured and how the functions get called, as well as their uses. Both start with include/import statements to pull in external packages that they need for extra functionality such as for string manipulation, handling files etc. Functions in both languages need to be declared before they can be called, so for any functions that will be called within other functions, these must be declared first, this is also why the import/include statements are the first few lines of code.

To replicate the structs from C, dataclasses were used, these function very similarly but in python the data type is more a suggestion as they can be changed (due to C being a statically typed and python being a dynamically typed language).

Both implementations have a main function at their end which acts as the default entry point for executing the rest of the code, this is where the code for prompting the user is also contained, which based on what is selected, the relevant functions are then called.

Finally for similarities, both change their variables for the respective structs/dataclasses in-place, where for example the fulfill_order function updates the shop without returning it. This means the struct/dataclass does not need to be copied, updated and then replace the original and this is most clearly shown by the type for fulfill_order in shop.c, as it is void it returns nothing, but yet the shop struct is updated.

Differences

The differences between the C and Python Procedural Programming approach comes from fundamental differences in how the languages work and the syntactic differences.

Every function present in shop.c is also present in shop.py, however, these functions differ in that the C language requires the functions to be given a type such as void if function returns nothing or int if the function is supposed to return an integer value. In python this is not the case, functions have no associated datatype as python is a dynamically typed language, this means a variable can be reassigned to represent a different data type with no issue.

C requires the use of memory allocation, declaring type and the use of pointers for variables which Python does not rely on and this results in the biggest differences between the two's implementations. The C approach requires a lot more variables and lines of code to do the same thing as in Python, an example of this is in how the shop or customer is read in from a csv.

In the python approach, reading in the csv's to populate the dataclasses is done by reading the file in to a variable, looping through it row by row and assigning the values on the current line to variables directly, these variables are directly used to populate data in the dataclasses.

The C approach instead needs pointers to the file, the line, and the values to be grabbed. For the variables that will be storing character arrays, these need their memory to be dynamically allocated first, the character array data needs to be pointed to with one variable first and then using strcpy() to copy the contents to another variable which is what then gets assigned as the value in the struct. Skipping this strcpy() step leads to confusing results, if removed from the stocking of shop for the product name, every product in the shop will be named after the last product loaded in as every item is pointing to the same location in memory.

Procedural Programming and Object-Oriented Paradigm comparison

Similarities

Syntactically the two Python approaches are the same, the functions have been mostly reused with minimal changes to the input arguments and have instead been embedded to their associated class e.g. createLiveCustomerOrder() function from the Procedural programming implementation was replaced with the Customer classes function create_live_mode() as this function is specific to a customer and so belongs in its class. For this example the only change made (aside from function name) was to remove the return in the object oriented approach as the function affects the object that called it as opposed to returning the customer object in the procedural approach.

Data types used across the two are the same, though the dataclasses have been replaced by classes.

Differences

With the Object-oriented approach classes are the significant difference, dataclasses have each been replaced with classes which also encapsulate the functionality that the objects should have, they no longer merely represent the structure for the data but how that data is generated and updated also.

By using classes in the object-oriented approach, it means we can make use of the __repr__ functions to provide structured print functionality for each of the classes, this means the functions like printProduct() and printShop() from the procedural approach can be replaced by the __repr__ of the respective product and shop classes.

The object-oriented approach also gives a smarter way of accessing product information such as name and price, as with this approach, the ProductStock class can have functions within for retrieving this information, which makes sense as you are able to associate the stock of a product more tightly with its name and price.

Conclusion

Even ignoring the differerent in-built and importable functions across different languages within the same programming paradigm, the differences in how the languages behave requires significant consideration for how they interact with memory, declare variables and their usage of data types. While the general structure and logical flow of the programs can be largely reused, there still will be big differences in functionality between the two languages which needs to be considered, and in the case of converting a program from C to python, will require especial attention around the movement of data through memory. My main takeaway from the experience of remaking a program from C to Python is that it would have been far more challenging to convert a program from Python into C due to the added complexities.

Within the same language and comparing programming paradigms the differences were not as significant as I expected. Much of the code could be readapted, requiring minimal changes, just inclusion of the “self” argument and in referencing variables. The object-oriented approach gives a more structured and tighter association between objects, the data they contain and the functionality they can perform, in the object-oriented approach these were all contained within the classes, in the procedural approach it was functions external to dataclasses which modified them in-place. Object-oriented is desirable behaviour in the case of something like a shop as we always want it to have specific functionality for accessing its data and interacting with other objects like customers, rather than relying on a set of disparate functions be called in the correct sequence. The uses of classes here also give the option to hide behaviour we don’t want tampered with, while not explored in this project, name mangling might be useful to provide some additional internal functionality to classes which we don’t want accessed outside of them.

In summary, there are advantages to both approaches, though given the nature of this project an object-oriented approach seems to suit best, however for a more linear program or simpler program a procedural programming approach might be more desirable.