**Introduction**

For Coursework 2 of the Programming 3 module, I developed a food-shopping statistics application using Scala. The aim of this project was to demonstrate an understanding of the methodologies of Functional Programming using a largely Functional language. The application itself worked as expected, however the methods used to develop it remain yet to be evaluated.

**Functional Thinking**

Based on the program code I have written, I believe that the application to a fair extent signifies an understanding of functional principles. Throughout the code, only immutable variables are used in order to prevent value reassignment, something which is ill-advised in Functional Programming. However, one way that I could have improved upon the application of functional principles would be by placing more emphasis on Pure Functions. There are several characteristics which make a function “Pure”, such as: the only data used is the data within the scope of the function, the function is deterministic (when provided with the same input parameters multiple times, the output will always be the same), the function does not accidentally affect variables or processes outside of itself.

In the application I developed, many of the functions do adhere to most of these principles, however all have one shortcoming: the functions read data from the main Map of food items (herein known as MapData) without actually passing the Map into the function. This negates the program’s value as functions are reading data from outside their own scope, therefore making them “impure”. Additionally, this could be considered bad programming practice in general, as it is effectively having the entire application revolve around a global variable. From the lens of a Functional development team, the Food Shop application would likely be seen as not very maintainable, modular, or indeed Pure. As a result, if the application were to be expanded it would almost certainly require a significant amount of refactoring in order to be more aligned with functional methodologies.

**Functional Programming Style**

The Food Shop application makes use of a variety of functional programming techniques. The most ubiquitous would be Recursion. Usage of recursion can be seen all over the source code, from integer input validation to data structure traversal. One of the more complex operations in the application was processing the user’s basket to be displayed to the console. As the basket is originally returned as a tuple, I created a recursive function to iterate through the map and list of quantities found within the Tuple. For this function I could arguably have used a foldLeft() method and had even more concise code. My reasoning for building my own recursive function stems from the fact that using recursion allowed me to visualise every step of the process as every line of code is visible. During testing, this made tracking any problems significantly less challenging as I could follow the flow of data step-by-step. Due to the fact that folding is implemented recursively in Scala, the two possible implementations would likely yield the very same result, however as a novice in functional programming, I found recursion to be very intuitive. Additionally, I made use of recursion for input validation. In functions where user input was required, I implemented a design wherein every input is validated, and if inputs were invalid, the function to get the user input would simply execute again until a valid input was read in.

In order to further demonstrate my understanding of Functional Programming and also experiment with alternative methods, I modified the readFile() method to incorporate a folding method and immutable variables. I used a foldLeft() as I had not found an appropriate place to incorporate this method throughout the program and wished to demonstrate my understanding of how to implement it. My reasoning for modifying the readFile() method in particular was due to the fact that I already had a working (albeit closer to Imperative) implementation of the feature. This allowed me to modify the method in a separate GitHub branch, meaning that if I was unable to successfully pivot the method to a functional implementation, the original code would still be intact. The outcome of this was successful and arguably increased the value of the program as a Functional application greatly as the most essential method now adhered to functional methodologies.

For error handling, I made use of pattern-matching several times. In the function allowing the user to compare the averages of two food items, I made use of pattern matching to ensure the program only displayed the final output if all food items were valid. The method for finding averages returned a tuple wrapped in an Option. Using pattern matching, I was able to have the function show an error message if the user did not enter valid food items (because a None would be returned).

**Comparison of Functional and Imperative Style:**

The use of functional methodologies for this program encouraged me as a developer to come up with more elegant, logical solutions as opposed to overly relying on a large number of libraries and imported functions. I believe this has resulted in a solution which is more modular, maintainable, and easily testable with methods such as dry-running.

In an imperative context, I believe I would approach the problem much differently. Given the choice of language, I would opt for an object-oriented approach using Java. The key differences would be that the Basket functionality would be implemented as a Class with each entry being a Basket Line object consisting of: Item, Quantity, Price, Total Price. This would make for a solution which would be much simpler to implement, but would arguably engage less critical thinking on the part of the developer and take more code to complete as additional class files would have to be created.