Adam Kelly

S2026818

Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Case | Description | Expected Outcome | Actual Outcome | Pass/Fail | Comments |
| 1 | Get the current price of each food | Tomato 232  Potato 65  Rice 46  Butter 670  Flour 59  Oil 2195  Chicken 670  Beef 1212  Apple 224  Milk 86 | Tomato 232  Potato 65  Rice 46  Butter 670  Flour 59  Oil 2195  Chicken 670  Beef 1212  Apple 224  Milk 86 | Pass | Test case passed as the correct current price is collected |
| 2 | Get the highest and lowest price for each food | Apple high 281 low 200  Butter high 797 low 660  Potato high 137 low 55  Flour high 66 low 41  Rice high 79 low 43  Milk high 117 low 71  Tomato 316 low 159  Oil high 2484 low 1823  Beef high 1299 low 927  Chicken high 881 low 602 | Apple high 281 low 200  Butter high 797 low 660  Potato high 137 low 55  Flour high 66 low 41  Rice high 79 low 43  Milk high 117 low 71  Tomato 316 low 159  Oil high 2484 low 1823  Beef high 1299 low 927  Chicken high 881 low 602 | Pass | Test case passed as the correct highest and lowest values were found for each food |
| 3 | Get the median price for each food | Apple 225  Butter 737  Potato 94  Flour 58  Rice 63  Milk 92  Tomato 221  Oil 2159  Beef 1154  Chicken 726 | Apple 225  Butter 737  Potato 94  Flour 58  Rice 63  Milk 92  Tomato 221  Oil 2159  Beef 1154  Chicken 726 | Pass | Test case passed as the correct median value was found for each food item |
| 4 | Get the food that has risen in price the most over the past 6 months | Apple | Apple | Pass | Test case passed as the item that has risen the most over the past 6 months is Apple |
| 5.1 | Compare the average price for two foods in put APPLE POTATO | APPLE average 235.2  POTATO average 95.5 | APPLE average 235.2  POTATO average 95.5 | Pass | Test case passed as correct average is found and displayed |
| 5.2 | Input RICE COKE | RICE 62.5  COKE no data | RICE 62.5  COKE NaN | pass | Test case passed as the inputs should display a none value for COKE as it is not in the dataset |
| 5.3 | Input Chicken  Beef | Chicken 738.3  Beef 1140.8 | Chicken NaN  Beef Nan | fail | Test case failed as actual outcome is not the expected one. To solve refactor code to have input validation to either Make sure input is all caps or convert to all caps. |
| 6.1 | Create a food basket using a food name and amount in kg to calculate total cost. Input TOMATO 1 RICE 1 | 278 | 278 | pass | Test case passed as total is correct |
| 6.2 | APPLE 1 POTATO 2  MILK 1 | 440 | 440 | pass | Test case passed as total is correct |
| 7 | Exit the application | Application closes successfully | Application closes successfully | pass | Test case pass as application is closed. |

Evaluation

Functional Thinking

The application uses functional principles throughout. One example of this is the use of higher-order functions. Higher order functions were used in the actionMap and the handleOne, handleTwo etc. The handle functions were used to handle the user input from the menu and map it to the actionMap function. Foreach, foldLeft and filter functions were also used in the solution, these are also higher order functions.

Data transformation is another principle of functional programming. In the solution map and foldLeft functions are used to transform the data. For example, in the currentPrice method to convert a sequence to a map.

Error handling is also functional. For example, the try catch block in the readFile function. This is used to read data in form the file and if the file can’t be found an exception is then thrown to the user.

Another key principle of functional programming is pattern matching. This is used in the menu to get an option selected by the user to run a function in the code. For example, the user typing 1 would call the handleOne function which in return would get the current price of all the food items.

Functional thinking was also used by making the data immutable. Immutable data is data in a database that cannot be modified or deleted. This was achieved in the solution by using the val keyword to declare immutable variables.

Functional Programming style

The following are examples of functional programming techniques that were used in the solution.

One technique that was used in the solution is Immutability. This was achieved through the use of the val keyword to declare variables. Immutability means that the data cannot be changed or deleted. Where if variables were declared using var they would be mutable and therefore be able to be changed.

Another technique that was used was pattern matching. Pattern matching is the process of matching values to other values in order to run functions. It is used in this solution in the menu function. It is used to get input from the user to run functions. In the solution it is mostly used in the handle1 – 7 functions. For example a user input of 3 will call the handle3 function.

The solution also uses pure functions. Pure functions are functions that don’t have side effects, meaning that the output is solely determined by the input. For example in the solution the functions calculateBasketValue, calculateAverage and calculateMedian are pure functions.

Another technique that is used in the solution is functional error handling. This is used in the readFile function when a try and catch block is used to handle the errors that could occur when reading in data from a file.

Another technique that was used in this solution was Data Transformation. Data transformation is the process of transforming data. For example, in the calculateBasketValue function the function foldLeft is used to calculate the total value of the basket based on the food and the amount. This is an example of data transformation.

Comparison of Functional and Imperative Style

Compared to an imperative approach the functional approach used made it easier to manipulate and transform the data. Using functions to transform the data made it simpler than an imperative approach. I felt it was easier to use the data to produce outputs from the input data than it would be using an imperative approach. Having the data be immutable and not change makes using that data more understandable.

Another aspect of functional programming that I found to be useful was the use of recursion. In an imperative language such as java especially if you have nested loops code can often become cluttered and messy looking making it quite challenging to understand and read. But with using recursion in a functional approach it felt following the flow of the data clearer. Using functions calls and recursion to control how the data is processed throughout a function was useful.

Another aspect of functional programming that I thought contributed to the ease of development was the use of pure functions. Pure functions are functions that the output is only affected by the input. This made understanding what was happening to the data easier in my opinion. Whereas in an imperative approach functions are often used to change the state of variables and control global variables and dependant on how many variables and what variables are being changed can become quite confusing at times.

If I were to approach this assignment again using a programming language that wasn’t scala I would choose Java. I would choose java as I have the most experience with it and have made food basket applications in the past. I would like to compare the differences in approaches after completing the solution in java as it would be interesting to see how the same processes can be achieved using an imperative approach compare to a functional one.

In terms of the developing the application using java and an imperative approach. I would still have the same menu structure. Classes would be used instead of objects. Store the database info in global variables so it can be accessed by each class.