Evaluation & Retrospective

Provides a detailed discussion of how the final product could be improved and documents any bugs and/or limitations. /5
Completes a detailed evaluation of the development process and suggests future impacts. /3

Improvements:
The entire program could utilise a GUI, and be ran as an .exe. I did try to create an .exe yet it wasn't functioning properly, i believe its cause of the multiple files i used, and them being in different folders complicating the directory.
A new screen that shows the total sales of the day would be great, and useful. It would also be good if all the GUI was in the same window as opposed to creating a new window for each screen. And of course, if I were to link an EFTPOS machine to the program, it would be the total package.

Bugs:
In terms of bugs, one thing i noticed is when I close a window, it doesn't always actually close, sometimes it requires me to close it twice before it stops reappearing. Also in some situations clicking exit or typing its corrospodning number doesn't always work as expected, and sometimes resulting in the program quitting entirely. Also when the program isn't run within VS Code the receipts function where it opens the folder to where the sale receipts are saved does not work properly, another reason the .exe failed. When on MacOS, the text and button are the same colour for the GUI, thus you can't actually read when your clicking, yet it does show up when you click it due to an "Animation".

Evaluation:
The project began with an investigation phase to outline the application's requirements and create a plan. This included understanding how to handle different types of items and their prices, as well as the needs of the Community Store for efficiency and ease of use. The design phase involved writing pseudocode to outline the application's core logic and the cashier's interaction with the application, including specifying how to handle different categories of items and prices, calculating totals, and providing receipts. During the development phase, the application was built using Python with a focus on modularity and best programming practices, such as input validation, meaningful variable names, and exception handling. The software's functionality included calculating total costs based on item categories and quantities, generating receipts, and saving purchase records to a text file. User acceptance testing (UAT) was performed with the assistance of peers to verify that the application met user expectations and the requirements of the brief. Feedback from this phase helped prioritize improvements and highlighted potential areas for enhancement.

The application successfully met the needs of the Community Store by streamlining purchase processing and providing an easy to use system for cashiers. The modular approach and use of meaningful names and comments in the code contributed to a clean, readable, and maintainable codebase. Challenges included ensuring the application was user-friendly for users and handling various input and exception

scenarios. Additionally, integrating a GUI framework added complexity but also improved usability.

Storing data such as purchase records in a text file is a good start, but transitioning to a database system would enable better data management, faster querying, and more complex reporting.