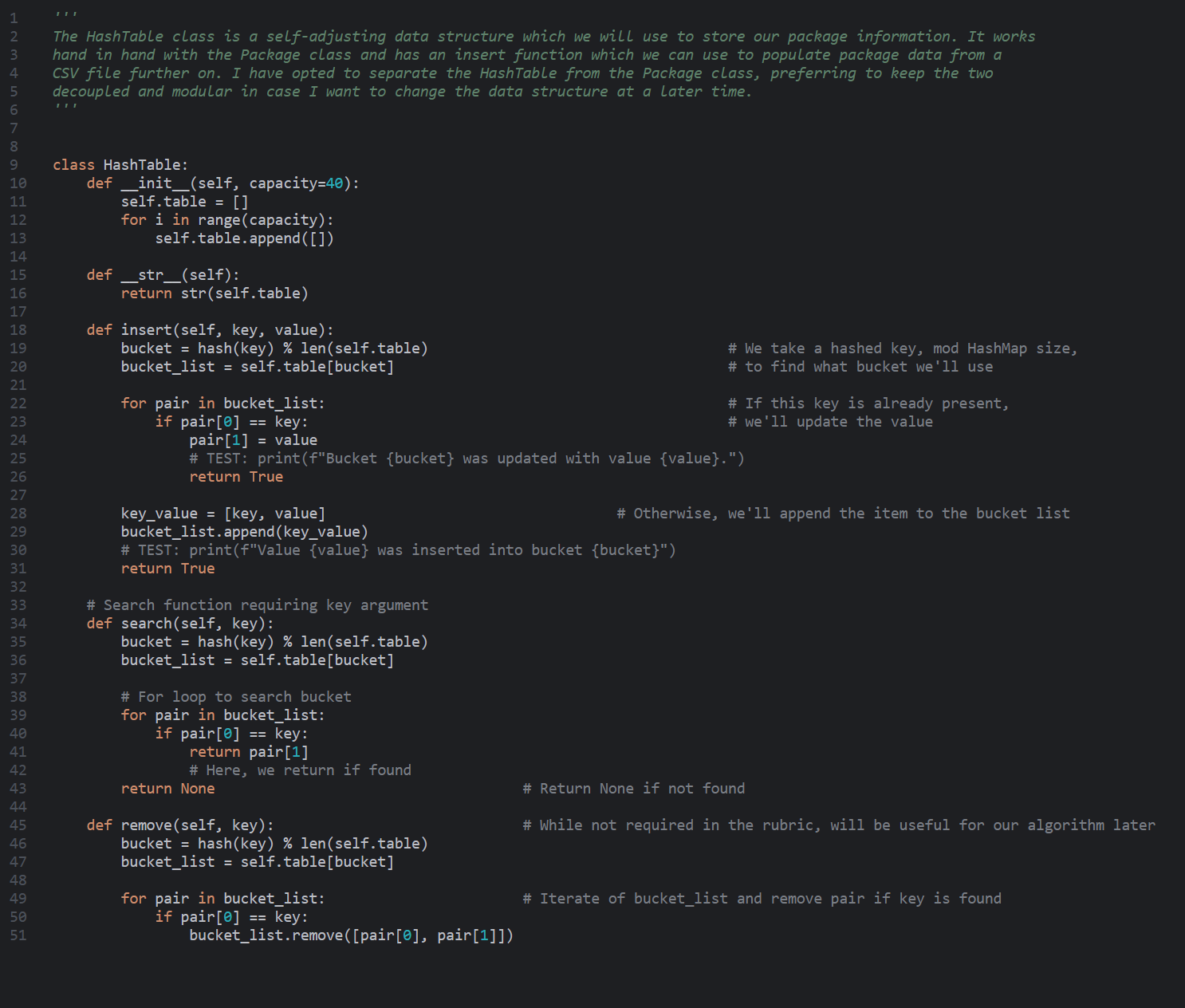
**DATA STRUCTURES AND ALGORITHMS II**

**Task 2: Program Implementation**

**James Revello (SID: 010649181)**

# Section A: Hash Table

The Chaining Hash Table utilized in our code has an insertion function which takes the Package ID as input. It is completely error free and inserts all data components correctly.



# Section B: Look Up Function

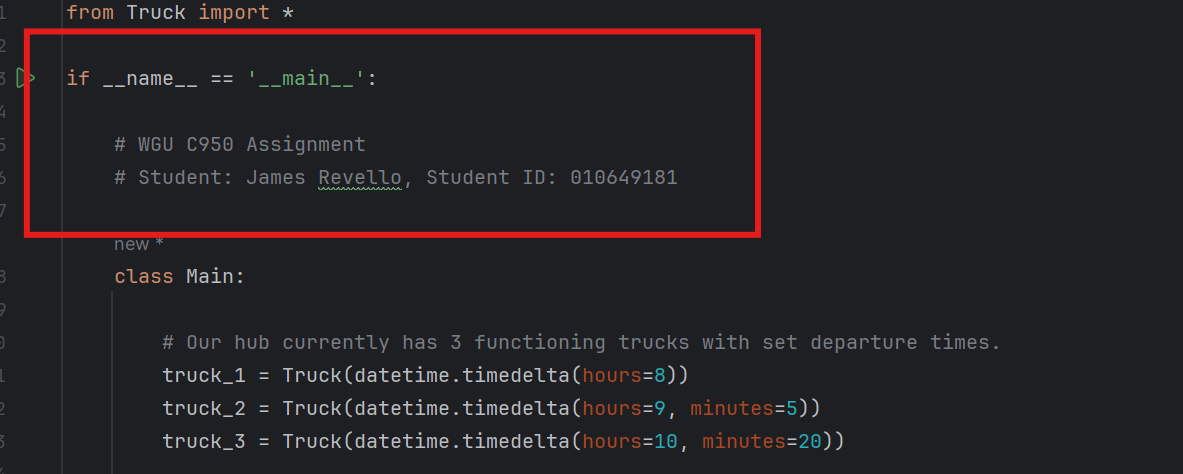
The Look Up Function--seen here as “Search()”—compiles without error, takes the Package ID as the only parameter, and returns all components. (See Screen Capture in Section A)

# Section C: Original Code

The code is entirely my own, and all learning sources used to understand the material are cited in Section I. There are no runtime errors or warnings. All packages get delivered within the constraints provided in the rubric.

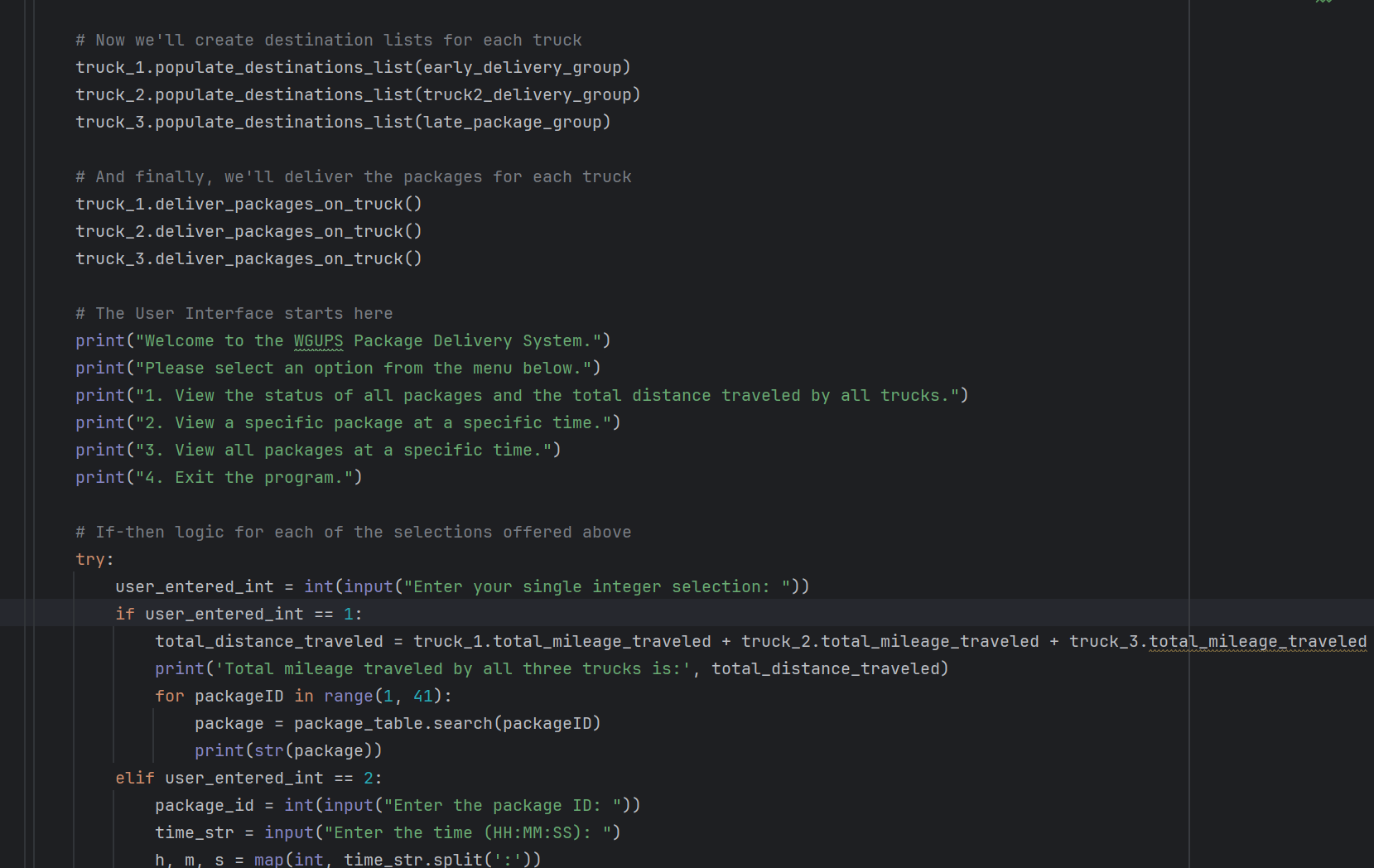
# Section C1: Identification Information

My Main function has the required identifying information



# Section C2: Process and Flow Comments

The code has process comments such as explaining the how and why of various functions and programming decisions. It also has flow comments, namely in the Truck and Main functions, where it explains the logic step by step.



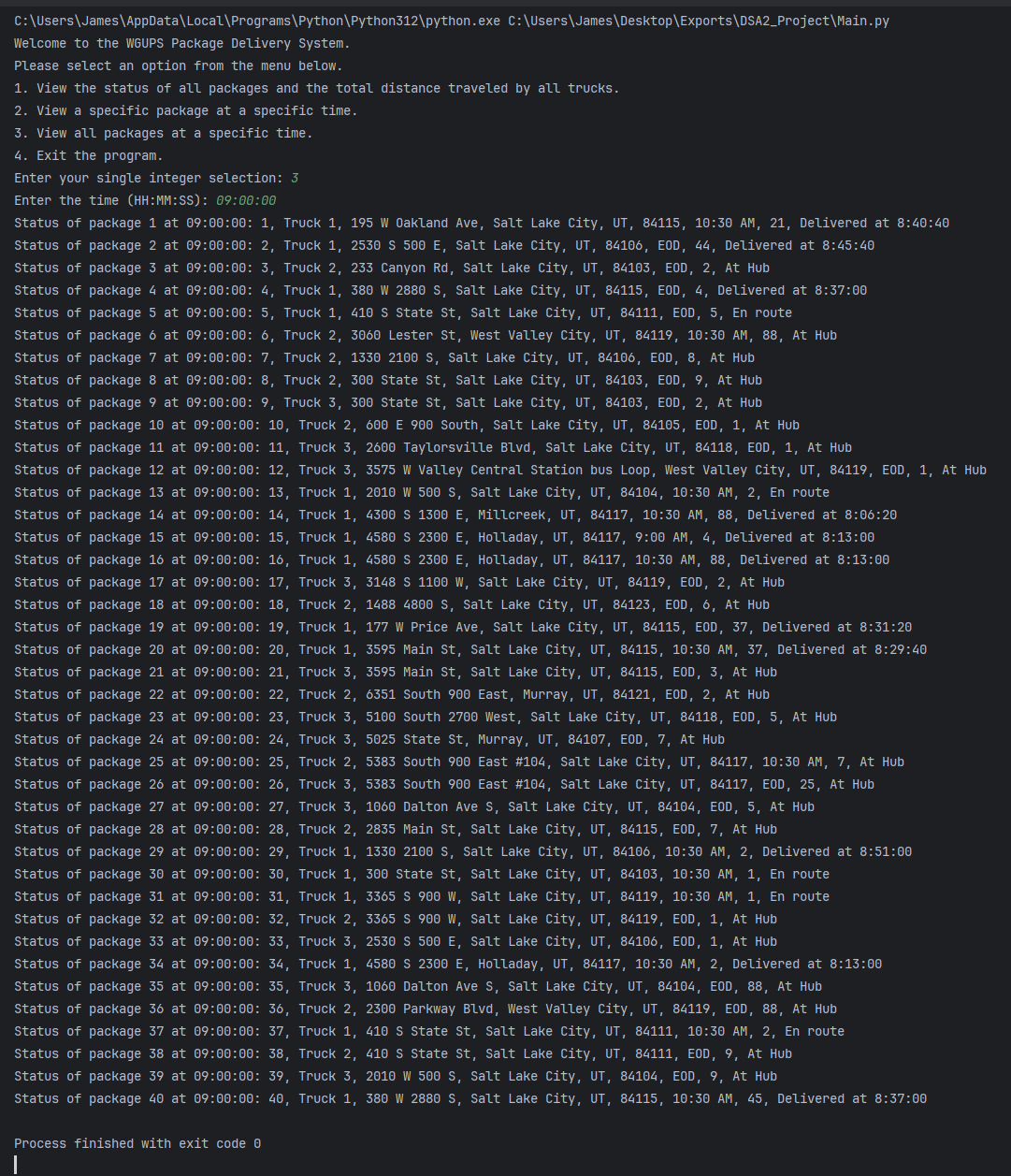
# Section D: User Interface

The code provides a user interface which allows a user to view delivery status (with timestamp) and check total mileage traveled by all trucks combined.

# 

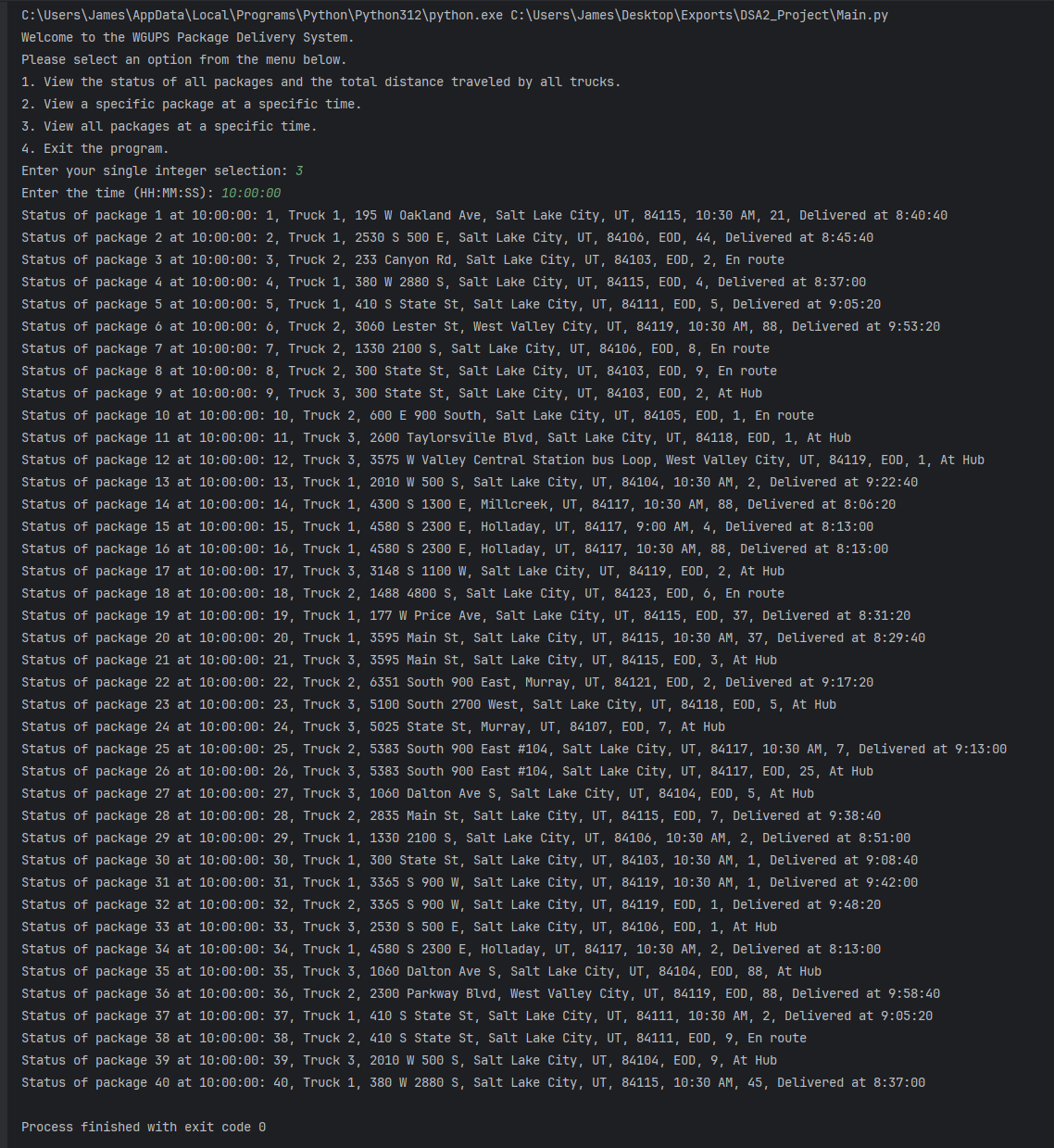
# Section D1: First Status Check

The screenshot below shows all packages on each truck, capturing the status of each one between 8:35 a.m. and 9:25 a.m.



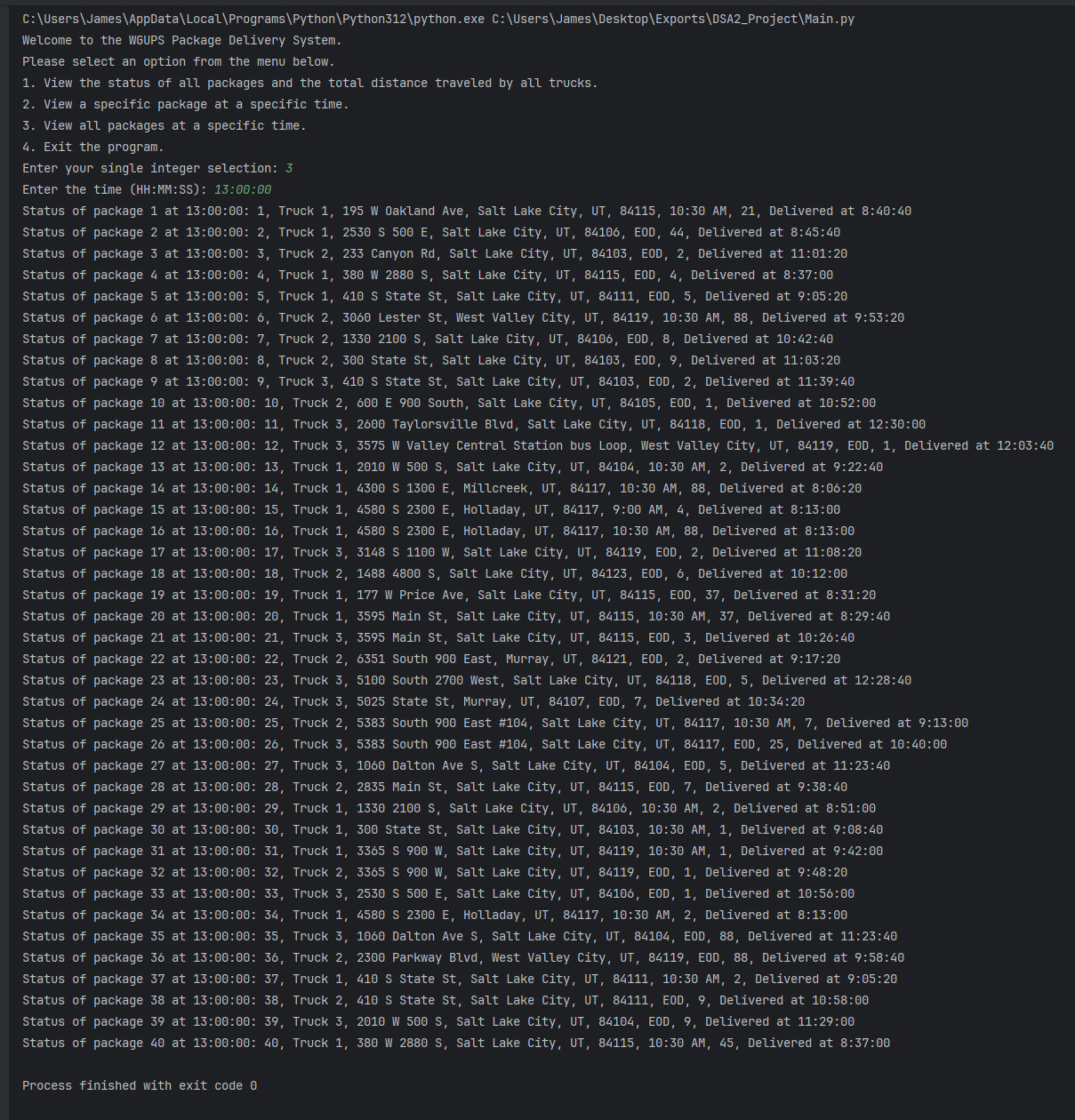
# Section D2: Second Status Check

The screenshot below shows all packages on each truck, capturing the status of each one between 9:35 a.m. and 10:25 a.m.



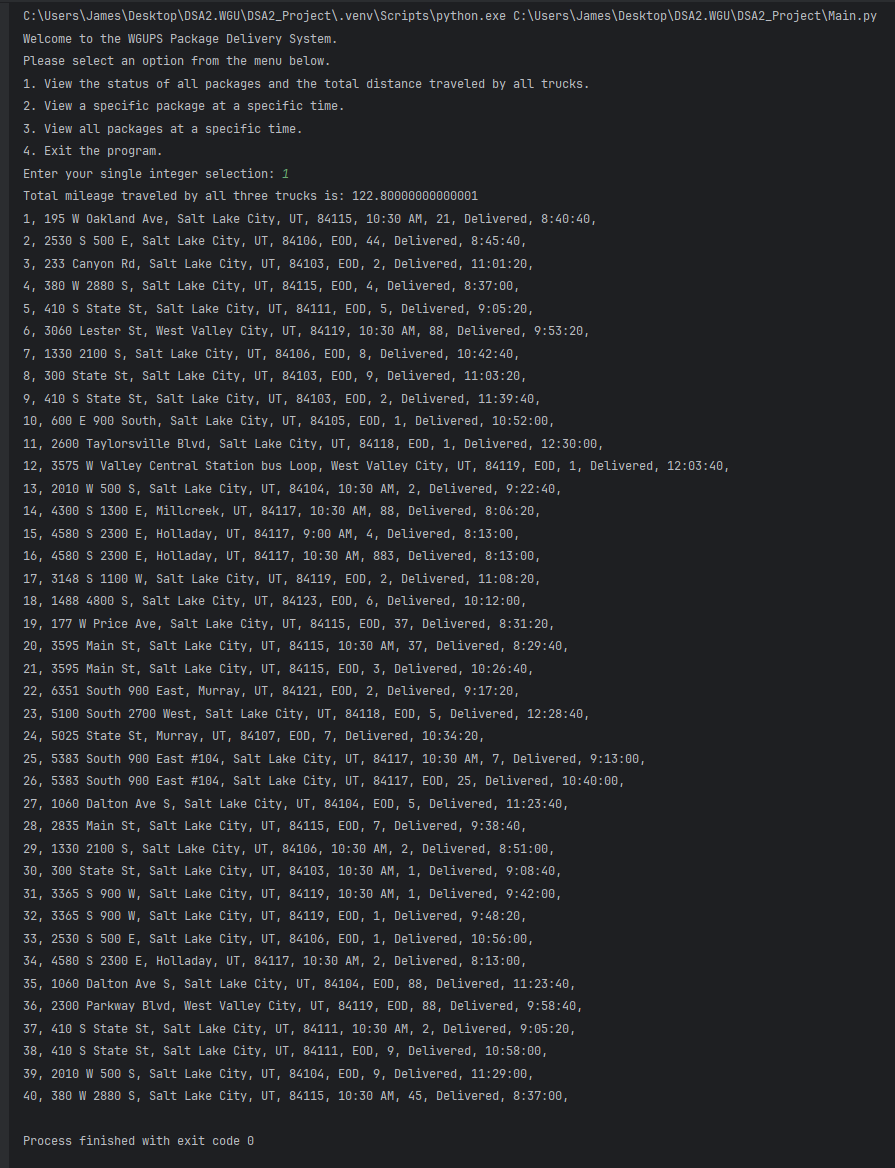
# Section D3: Third Status Check

The screenshot below shows all packages on each truck, capturing the status of each one between 12:03 p.m. and 1:12 p.m.



Section E: Screenshot of Code Execution

This screenshot shows the code runs without any errors (finished with exit code 0) and shows the total mileage traveled by all three trucks is 122.8



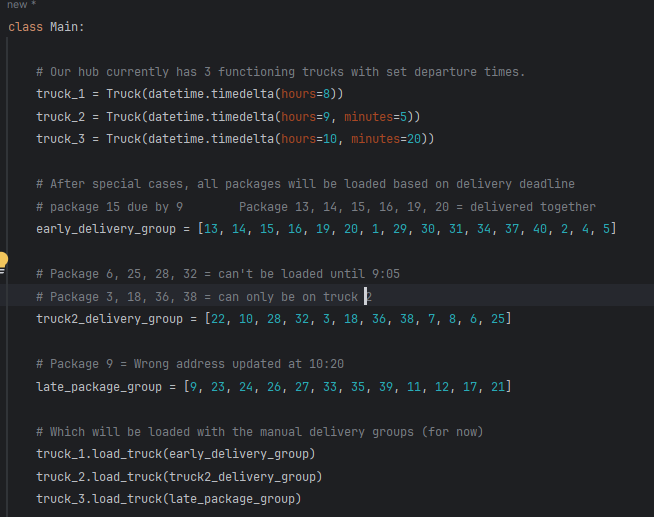
# Section F1: Strengths of Chosen Algorithm

The Nearest Neighbor Algorithm I opted to use for this project has two main strengths: it is simple to explain, and it can handle increasing inputs with a runtime of O(n^2). The former is extremely important because the business plans to roll out the program to other hubs. By using a simple algorithm, the code can be easily understood without in-depth knowledge of graphs, vertices, and other heuristic models. Despite being easy to understand, the algorithm is by no means slow. Since it only presents a worst-case space-time complexity of O(n^2) it can handle increasing inputs without too much difficulty. Beyond these obvious strengths, the algorithm is self-adjusting and will be easy to scale and maintain moving forward.

# Section F2: Verification of Algorithm

All requirements for the project for met. As seen in the screenshot in Section E and the screenshot below:

1. No trucks leave before 0800 Hours
2. All three trucks delivered packages in 122.8 miles (within the 140 miles required)
3. All packages were delivered by their deadline
4. All package constraints were met by their loading logic
   1. Packages 13, 14, 15, 16, 19, and 20 were delivered together
   2. Packages 6, 25, 28, 32 were loaded after 0905 Hours
   3. Packages 3, 18, 36, 38 were all loaded on Truck 2
   4. Package 9 was not delivered until after 1020 Hours and the address was not changed until that time.



# Section F3 and F3A: Other Possible Algorithms and Their Differences

Instead of my Nearest Neighbor algorithm, I could have utilized Dijkstra’s Algorithm, or 2-Opt. Both are significantly different than Nearest Neighbor, but nevertheless meet all requirements for heuristically delivering packages with the constraints provided in our problem. Dijkstra’s algorithm finds the shortest path from one node to *all* other nodes on a weighted graph. In this way, Dijkstra’s always finds the shortest path without fail, whereas Nearest Neighbor simply finds only the immediate nextshortest node. This makes Dijkstra’s algorithm more resource intensive, but also more precise in finding the shortest/quickest path. 2-Opt, on the other hand, takes one initial path or “tour”, then iteratively swaps various edges along its route and tests them until it finds the shortest possible route. In both cases, the Nearest Neighbor algorithm is much simpler, merely iterating over the next shortest edge instead of iterating over all possible edges or over tours with swapped edges.

# Section G: Different Approach

I think the first thing I would do differently if I tackled this project again would be to automatically load my packages. With such a small package size of 40, it was more efficient to manually load the packages based on the constraints provided. However, as the project scales we will have more packages across more cities. I would implement a simple if-then chain to check the constraints of given packages and load them appropriately. I could even *load* the packages using the nearest neighbor algorithm so they are placed from back to front based on destination times from furthest to closest. In this way, the offloading of the trucks would be more efficient.

# Section H: Verification of Data Structure

As you can see in the screenshot (Section A), the data structure utilized in my solution meets all requirements of the scenario: it take Package ID as key input, has a search function, and even has a remove function should one wish to remove data within the structure. It is written from scratch without utilizing any remote Python Libraries. All packages get delivered to the appropriate addresses while meeting all project requirements. (See Screenshot in Section A)

# Section H1: Other Data Structures

Instead of my Hash Table, I could have used a Linked List or Graph of Nodes for my data structure to contain the packages.

# Section H1A: Data Structure Differences

By using a Linked List instead of a Hash Table, I would be able to add packages with O(1) time instead of O(n) as package sizes become prohibitively large. And since the packages are a linked list, we could simply use the linkage order as the delivery order for the various packages (almost creating a form of first-in-first-out queue for delivery)

By using a Graph of Nodes instead of a Hash Table, I would have a way of storing the distance data of edges between the various nodes and the nodes representing addresses. This would help calculate my delivery route as soon as the packages were entered on the data structure.

# Section I: Sources Cited

Tepe, C., Dr (2020, November 17). C950 - Data Structures and Algorithms 2 Course Webinar [Video Webinar]. <https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=f08d7871-d57a-496e-a6a1-ac7601308c71>

Western Governor's University (2021, July 28). C950 WGUPS Project Implementation Steps - Example - Nearest Neighbor. WGU Course Resources. <https://srm--c.vf.force.com/apex/CourseArticle?id=kA03x000001DbBGCA0>