**NHP2**

A. The 2-opt Algorithm was used for my solution.

B1.

2opt(route\_to\_sort)

repeat until no improvements are made:

best\_distance = calculate\_total\_distance(route\_to\_sort)

for (i=0; i < length\_of(route\_to\_sort) - 1; i++):

for (k = i + 1; k < length\_of(route\_to\_sort); k++):

new\_route = 2optswap(route\_to\_sort, i, k)

new\_distance = calculate\_total\_distance(new\_route)

If (new\_distance < best\_distance):

route\_to\_sort = new\_route

best\_distance = new\_distance

B2. PyCharm 2021.3.3 (Community Edition) was used to create the Python Application.

Device:

Device name Bruh

Processor Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz 2.30 GHz

Installed RAM 16.0 GB (15.9 GB usable)

Device ID 03085A50-838F-4308-9372-8D5B17BB6B05

Product ID 00325-95957-46688-AAOEM

System type 64-bit operating system, x64-based processor

Pen and touch No pen or touch input is available for this display

Windows:

Edition Windows 10 Home

Version 21H2

Installed on ‎9/‎28/‎2020

OS build 19044.1620

Experience Windows Feature Experience Pack 120.2212.4170.0

B3. For the purposes of this assessment, we will take the major segments of this program to be the plan\_truck\_routes, two\_opt, and two\_opt\_loops methods of the Route class, and the Truck and PackageHash classes and their methods.

two\_opt: time complexity is O(N^2) space complexity is O(N).

two\_opt\_loops: time complexity is O(N^2) space complexity is O(N).

plan\_truck\_routes: time complexity is O(N^2) space complexity is O(N).

Truck.add\_package\_to\_truck\_route: time complexity is O(c) space complexity is O(k).

Truck.load\_truck\_and\_start\_driving: time complexity is O(N) space complexity is O(N).

Truck.get\_mileage: time complexity is O(N) space complexity is O(N).

Truck.get\_return\_time: time complexity is O(N) space complexity is O(N).

PackageHash.h2: time complexity is O(c) space complexity is O(k).

PackageHash.insert: time complexity is O(c) space complexity is O(N).

PackageHash.lookup: time complexity is O(c) space complexity is O(N).

PackageHash.remove:time complexity is O(c) space complexity is O(N).

PackageHash.resize: time complexity is O(N) space complexity is O(N).

PackageHash.\_\_str\_\_: time complexity is O(N) space complexity is O(N).

The entire program has a time complexity of O(N^2) and a space complexity of O(N).

B4. Most of the code written for the program is scalable to a growing number of packages. In the current plan\_truck\_routes function, many decisions are made by hand, not programmatically. If code was rewritten to make decisions without code intervention, the solution would be fully scalable for almost any number of packages.

B5. The software is efficient and easy to maintain because it uses an appropriate amount of abstraction in its many functions. Parts of the code can easily be changed or updated without affecting the entire program.

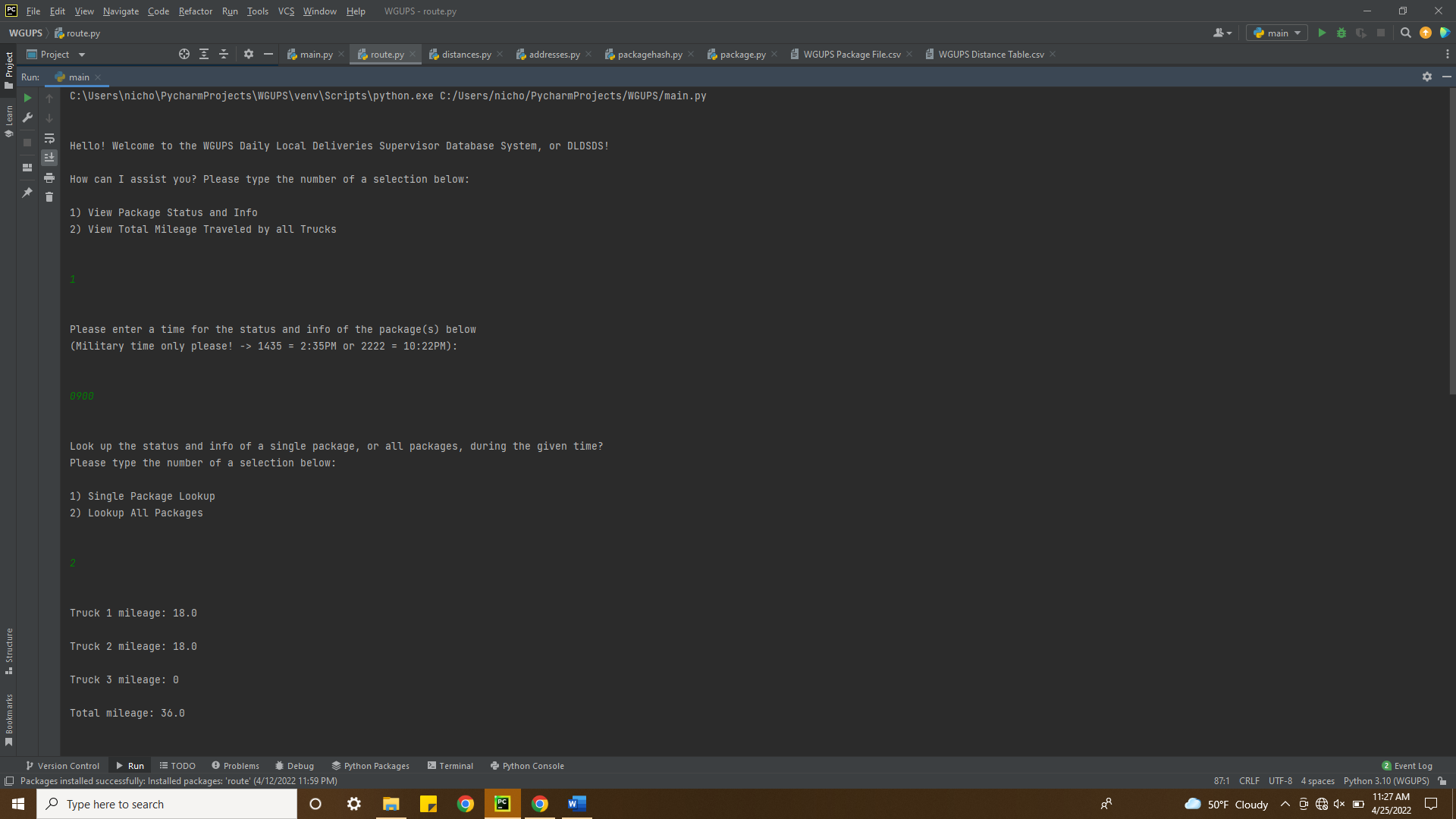
B6. The obvious strengths of the hash table (PackageHash class) created and used in the solution are its ability to perform lookups in constant time (O(c)) and iterate in nearly O(n). A current weakness of the PackageHash hash table is its inability to efficiently resize, requiring quite the time and space overhead to do so, quite frequently as well.

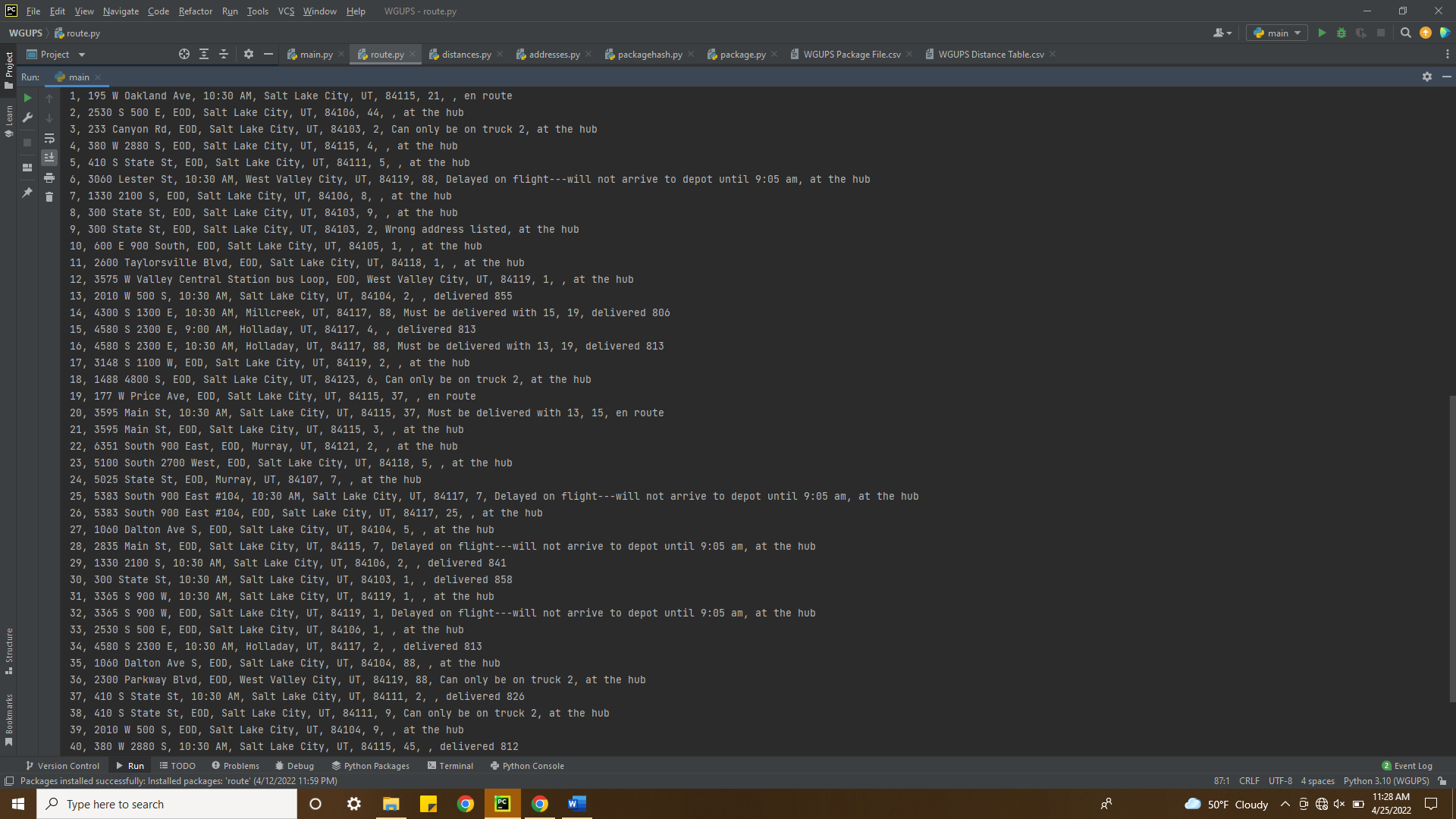
D. A double-hashing, self-resizing hash table is used to store package data.

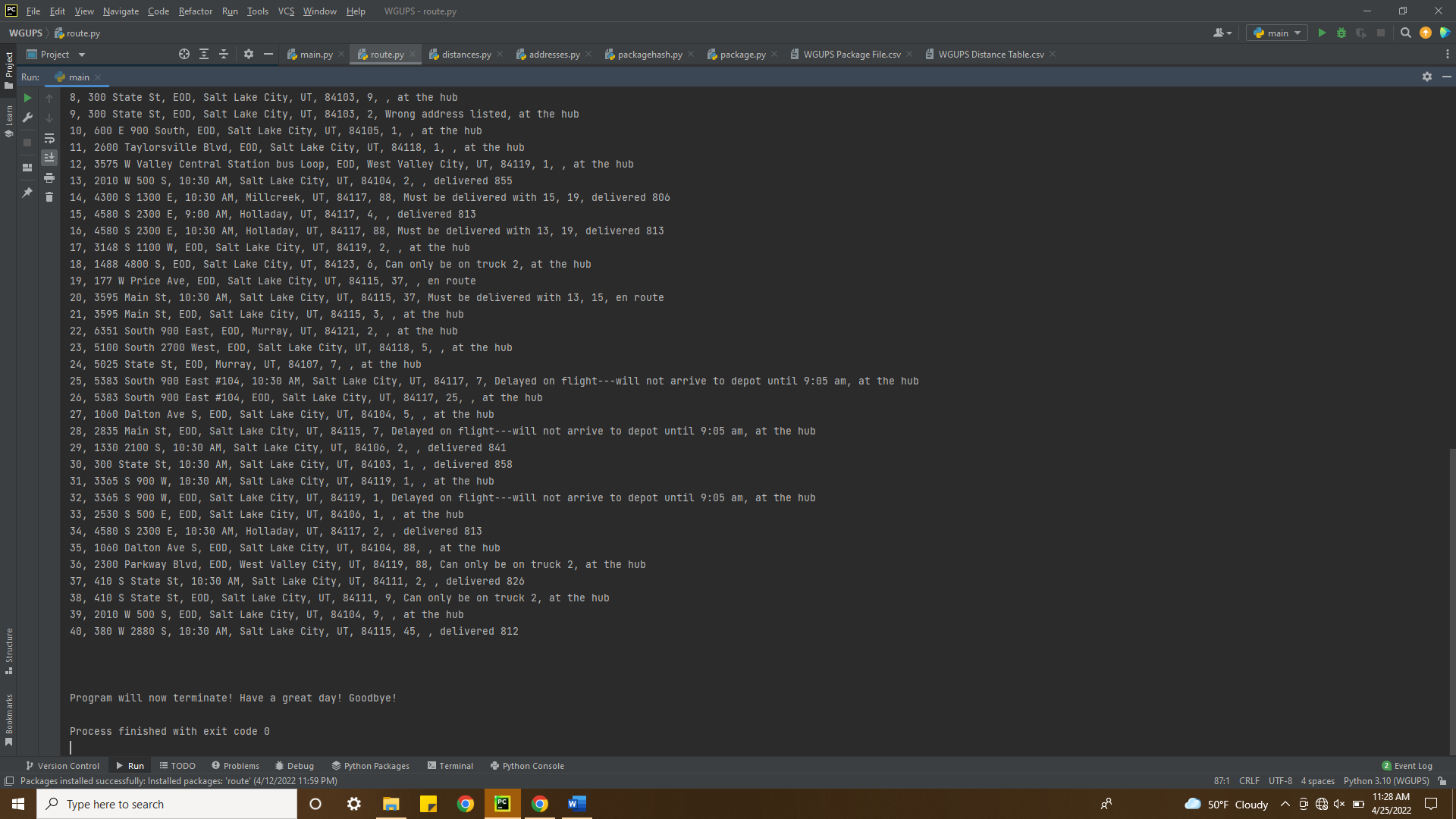
D1. The PackageHash class is able to store any number of packages and their data, using the unique package ID of each as a key. It is able to perform lookups using unique package IDs.

G1.

9:00AM

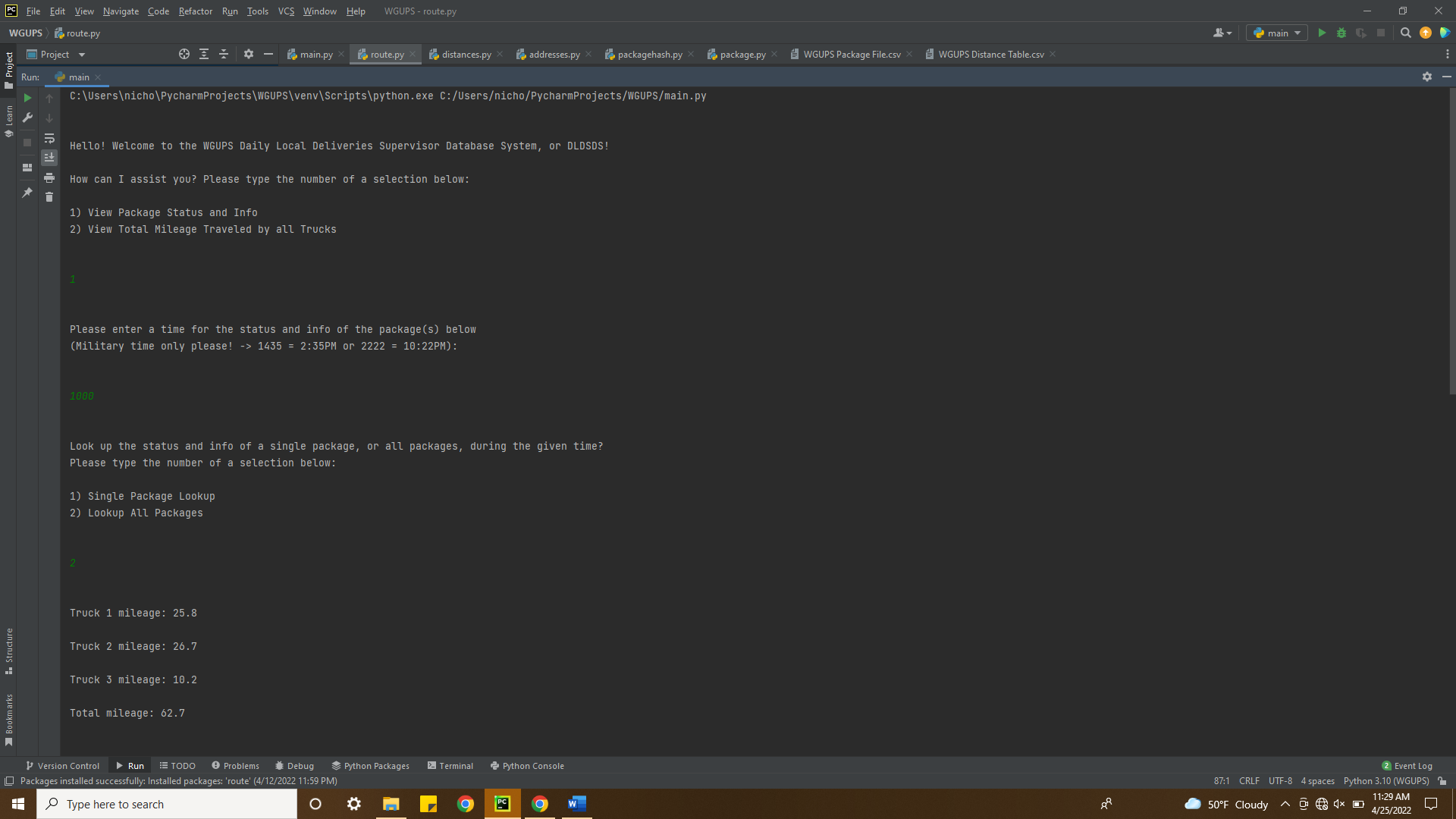


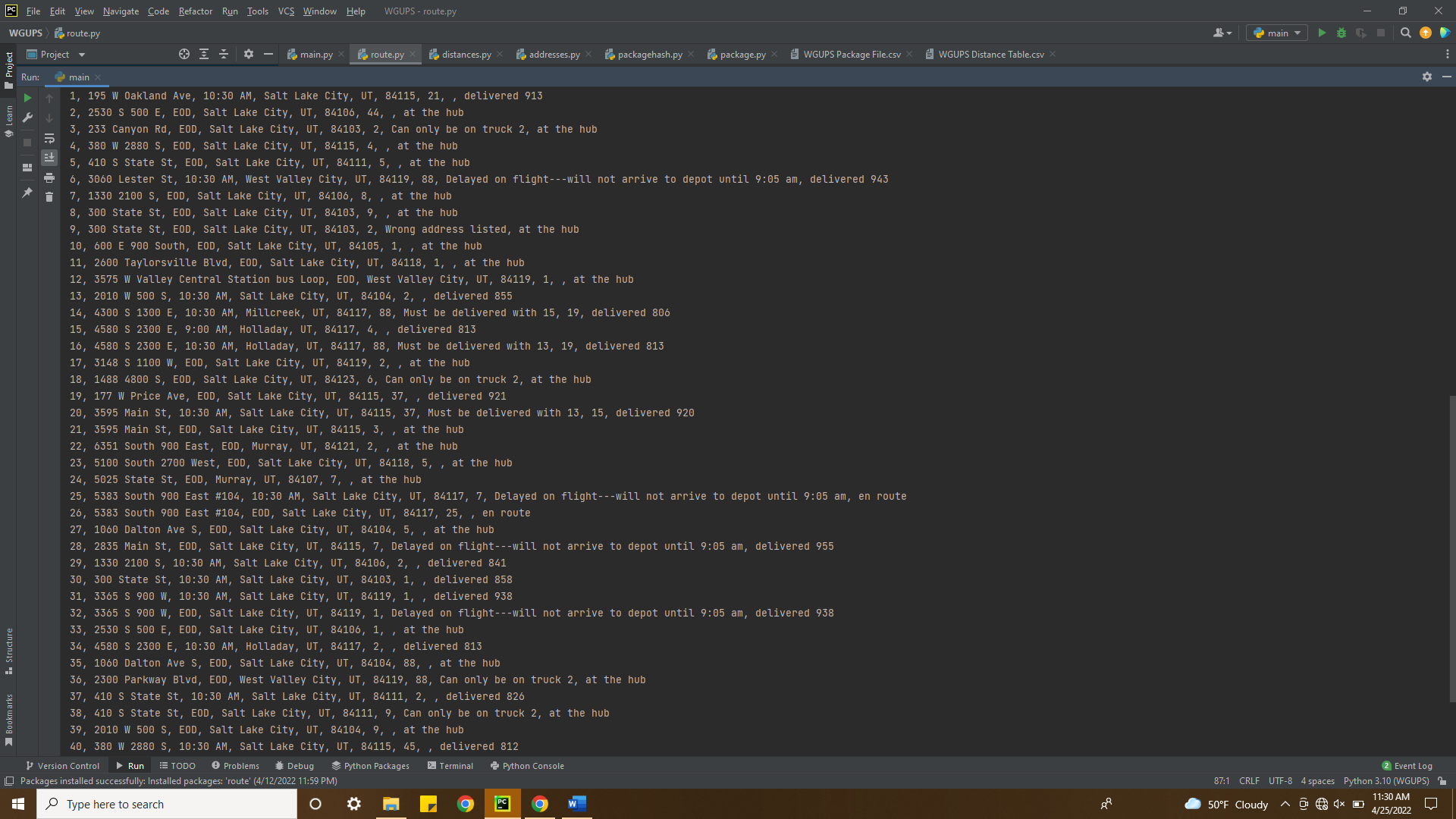


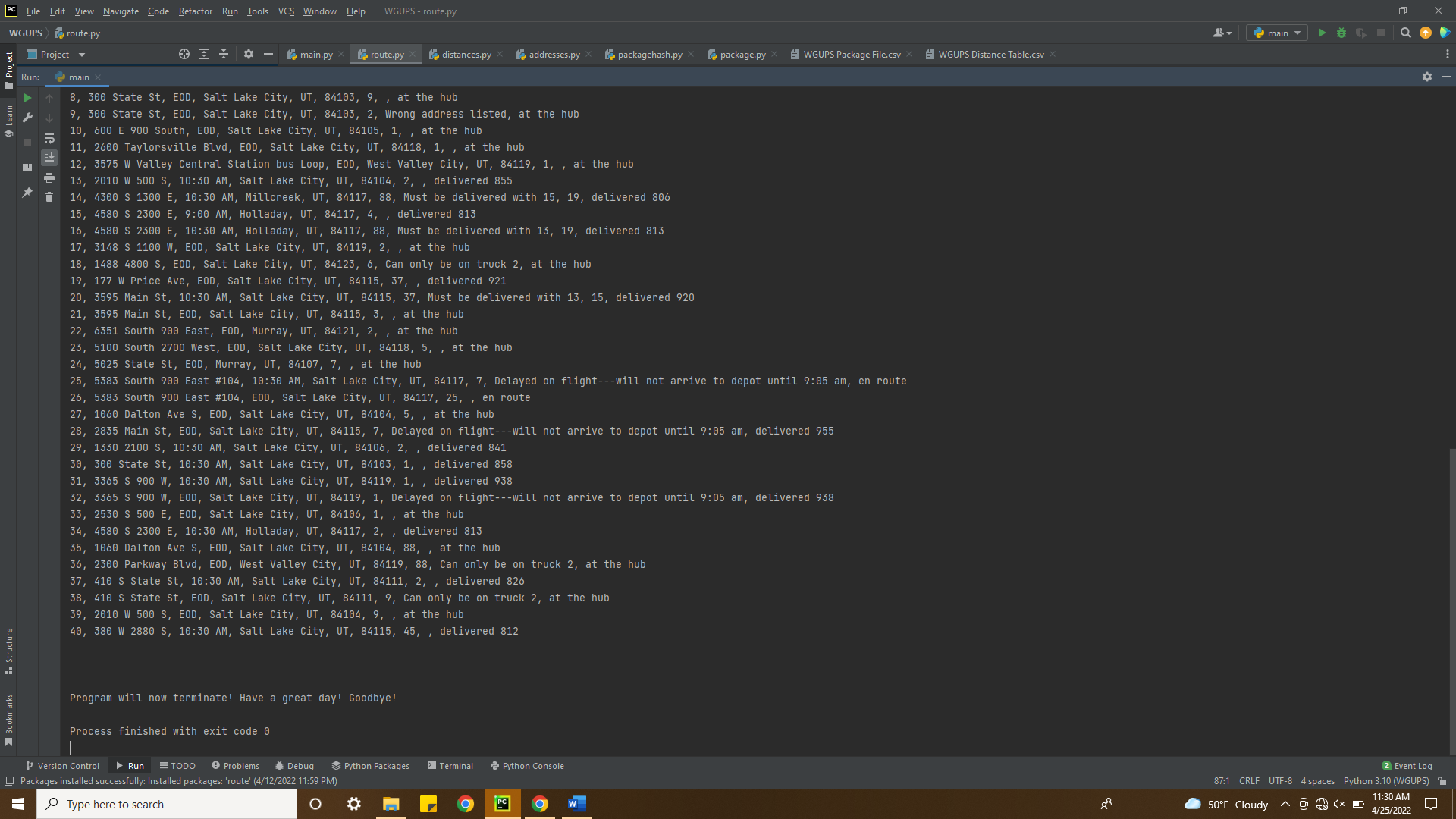


G2.

10:00AM

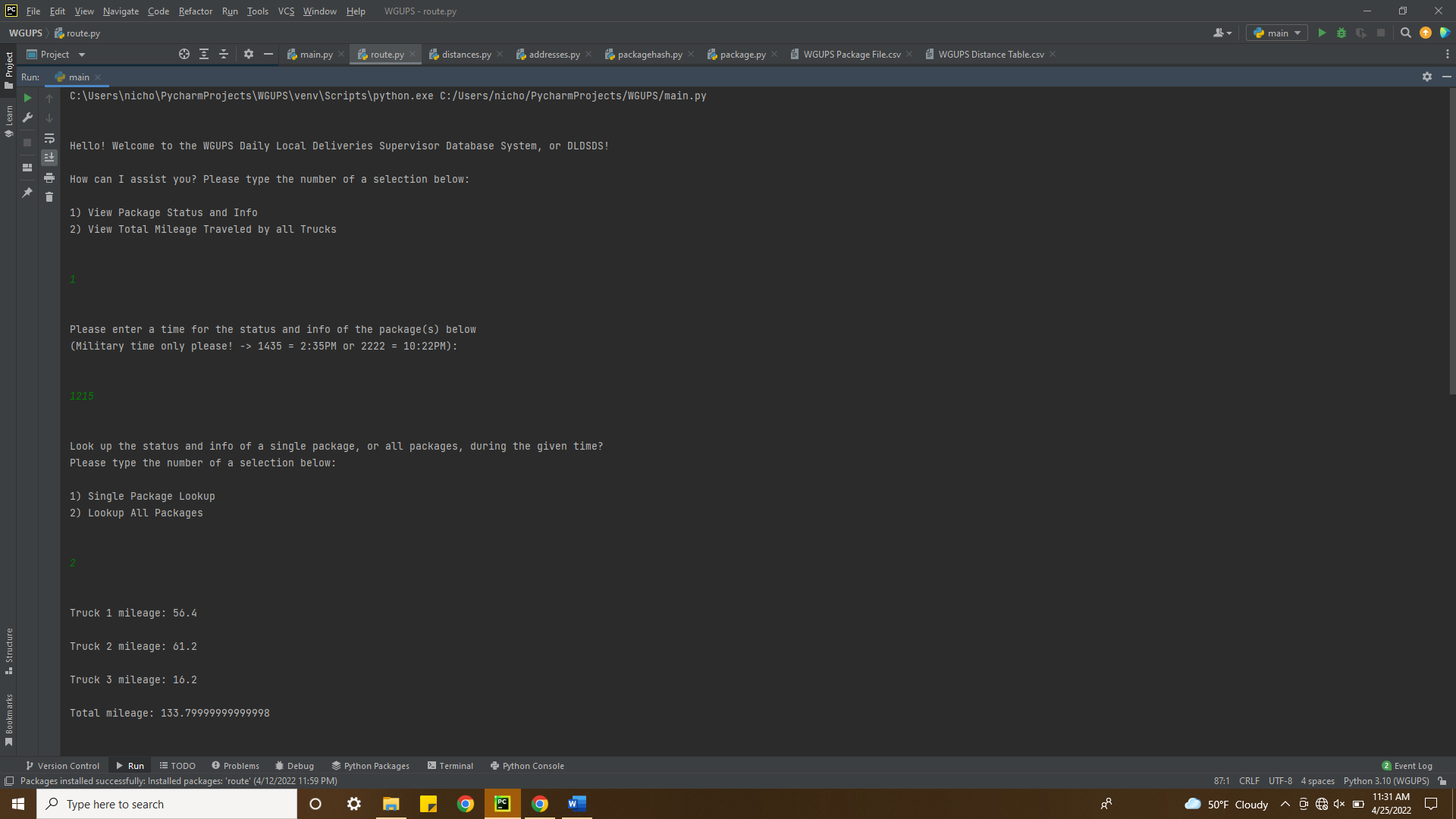


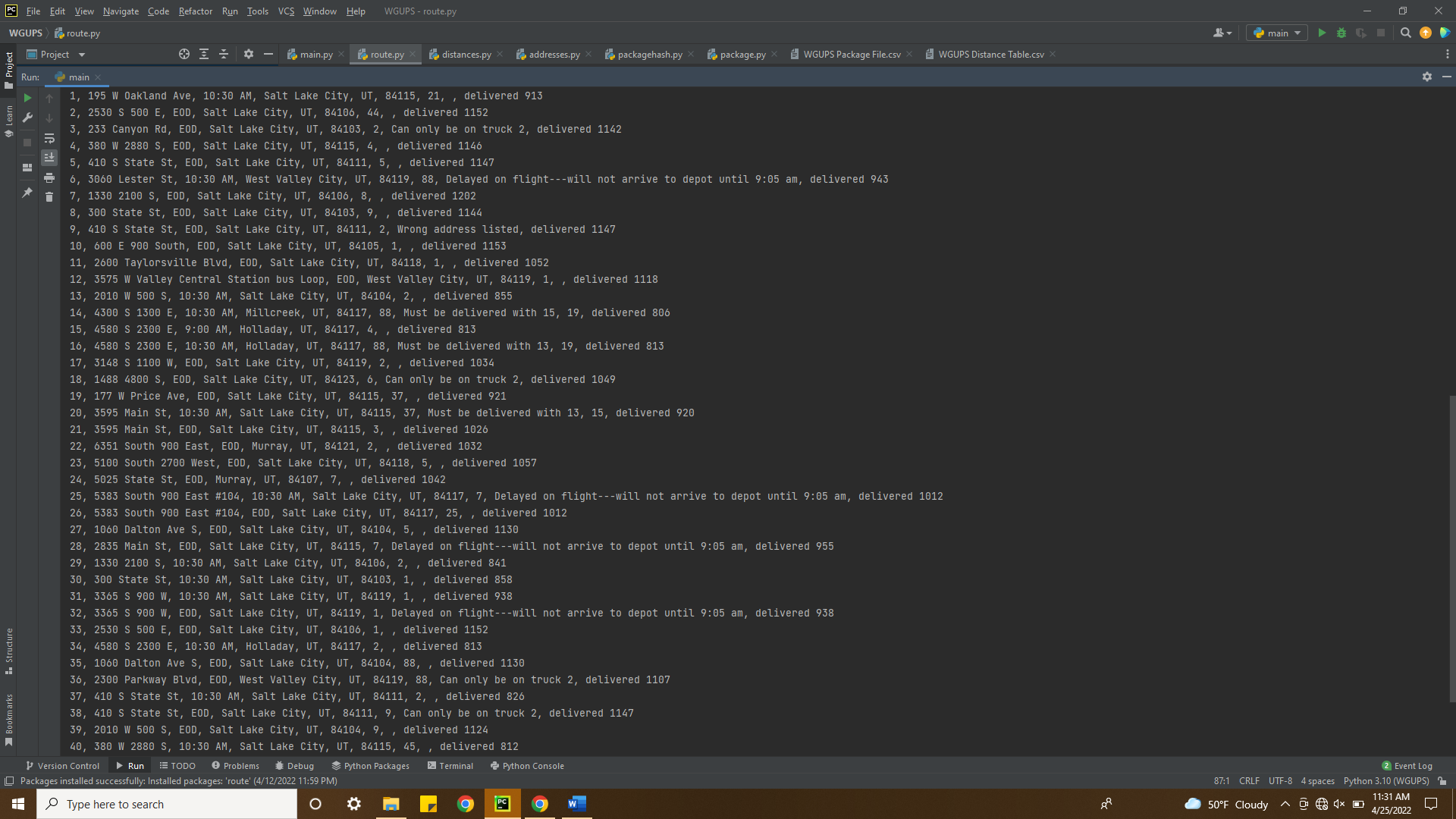


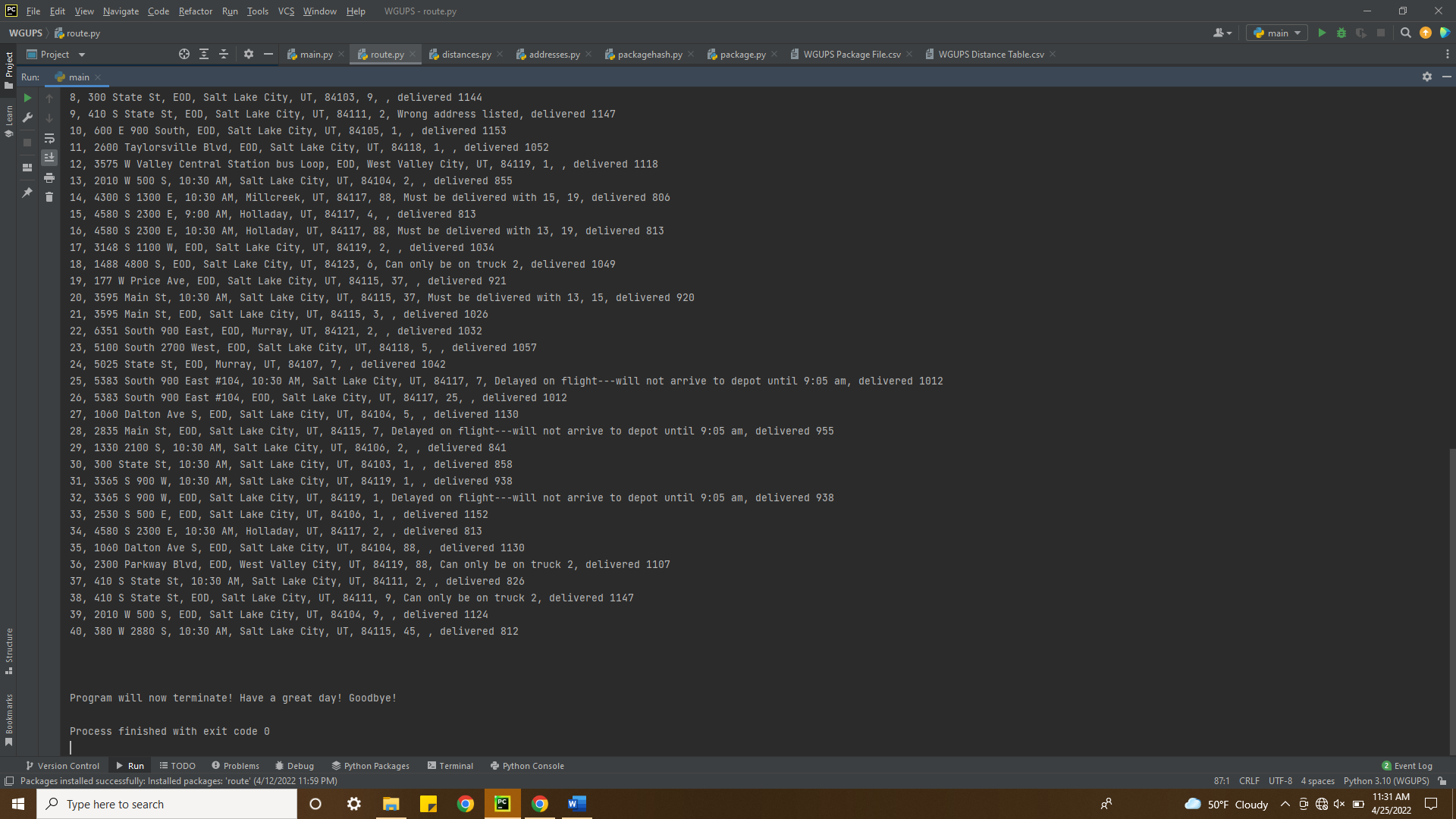


G3.

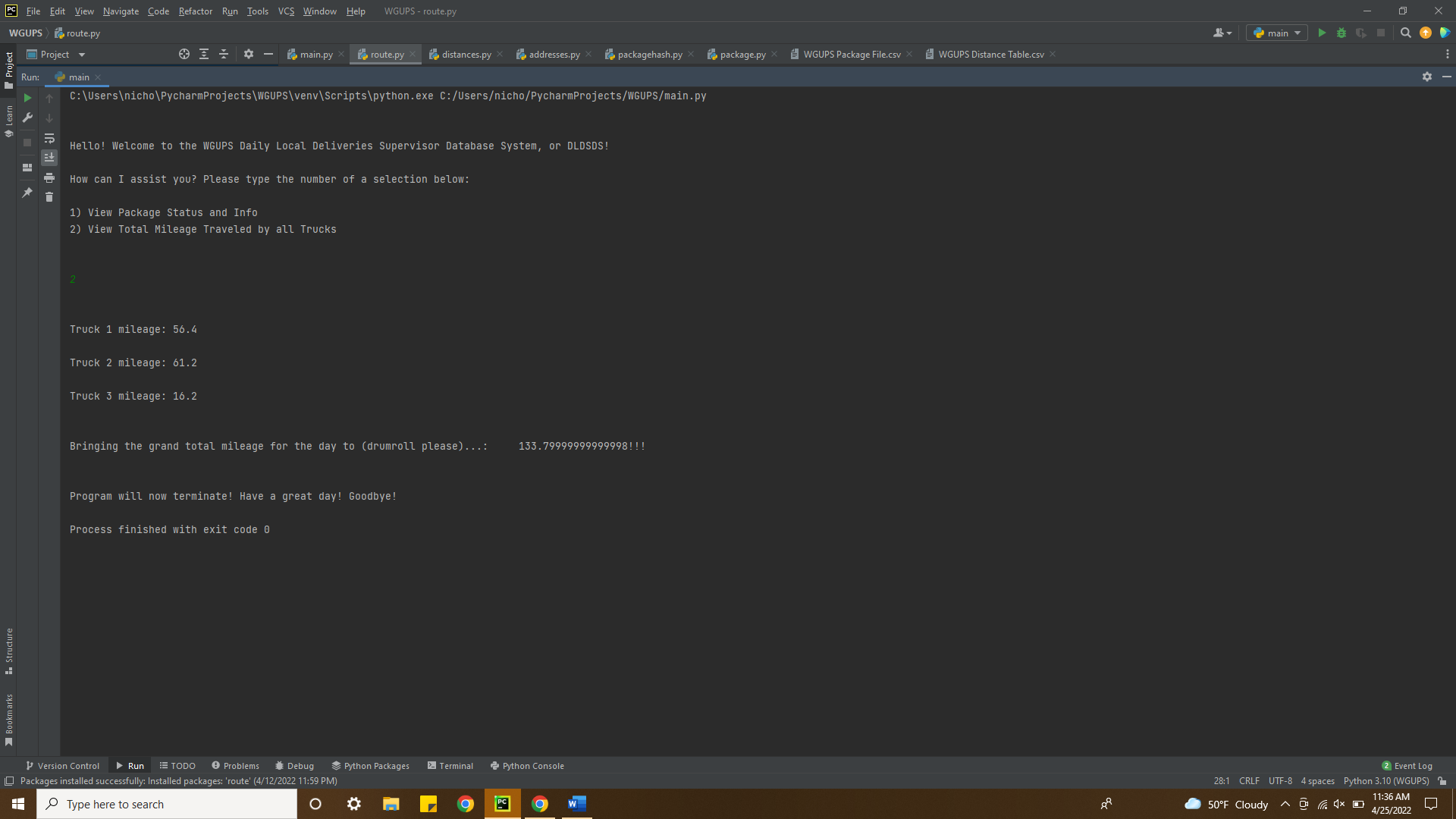
12:15PM







H.



I1. One strength of the 2-opt heuristic algorithm implemented in the program is that it is easy to implement and easy to understand. Another strength of the 2-opt algorithm is that it is a relatively strong performer and only has a performance cost of O(n^2).

I2. The algorithm indeed helps meet all of the requirements of the scenario and all packages are delivered on time and below a total of 140 miles as shown in the screenshots above.

I3. One algorithm that would meet the requirements of this scenario is the 3-opt heuristic algorithm. Another is the Christofides algorithm.

I3a:

3-opt: 3-opt is a more complex and slower algorithm that is similar to 2-opt. It produces more optimal solutions than 2-opt, but each execution of 3-opt is O(n^3).

Christofides: Christofides algorithm uses a minimum spanning tree. It guarantees that a solution will be found that is within 150% of the optimal solution, however its time complexity is O(n^2 \* log(n))

J. If I were to do this project again, I would like to ensure scalability by writing the program to algorithmically / programmatically choose how to load packages based on the passed in package data .csv file of any size.

K1. The PackageHash hash table indeed meets all of the data storage and retrieval requirements in this scenario.

K1a. The time needed to complete a look-up does not change based on the number of packages to be delivered. It is ensured to always be O(c) / O(1). This is because a key value will immediately map to the location of the package contained in the hash map table. If there is a collision, a few more comparisons will need to be made, but this is unlikely.

K1b. The space usage will grow with the number of packages stored in the PackageHash hash table. Since the hash table will only ever be at most %70, our space complexity is O(1.429N). Although this implementation uses more than O(N) space, it is more effective at avoiding collisions.

K1c. Within the current implementation, changes to the number of trucks or cities would not affect the look-up time or space complexity of the PackageHash hash table.

K2. 2 other data structures that could be used in the solution would be an Array (Python List) or a Binary Tree.

K2a.

Array: An Array would be very similar to the underlying data structure behind the hash table, however look-ups would jump from O(1) to O(N), having to iterate over the entire array.

Binary tree: A Binary Tree would show improvement in look-up time over an array, but would still be worse than a hash table, with a look-up complexity of O(logN).