# C950 Task Directions and Rubric Summary

Alternative task directions and FAQ (NHP2)

Author: Dr. Jim Ashe
version NHP1 (older version)

## Scenario

The Western Governors University Parcel Service (WGUPS) needs to determine the best route and delivery distribution for their Daily Local Deliveries. The Salt Lake City DLD route has three trucks, two drivers, and an average of 40 packages to deliver each day; each package has specific criteria and delivery requirements.

Your task is to write code that determines and presents a solution delivering all 40 packages, listed in the attached "WGUPS Package File," on time, according to their criteria while reducing the total number of miles traveled by the trucks. The "Salt Lake City Downtown Map" provides each address's location, and the "WGUPS Distance Table" provides the distance between each address (note: mileage on the distance files may not match distances on the map).

The supervisor (user) needs the means to check the status of any given package at any given time using package IDs. The report should also include the delivery times, which packages are at the hub, and en route. The intent is to use this program for this specific location and use the same program in different cities as WGUPS expands its business. As such, you will need to include detailed comments following the industry-standards to make your code easy to read and justifying the decisions you made while writing your program.

## Project Summary

You will write a program in Python, which does the following:
- Can store the package information into a hash table.
- Uses a *self-adjusting* heuristic algorithm to find a solution that delivers all packages using under 140 miles and according to the provided requirements (e.g., delivery deadlines, addresses, number of trucks, special notes, etc.).
    > Note: Non-self-adjusting methods are also allowed, but there must be at least one part that uses a self-adjusting algorithm (identified in Part A) with one scalable element (discussed in Part B4).
- Allows the 'user' to check the status (at the hub, en route, or delivery time) of any package at any given time.

Using code comments and a separate document, complete the following:
- Provide code comments for every significant portion of code that explains the logic and time-complexity.
- Provide documentation on algorithms used, alternative methods to, efficiency, and scalability of the methods used to find a solution.
- Provide documentation on the data structure(s) describing their use, alternative methods, efficiency, and scalability of data processes.

**Technical Requirements and Resources:**

- Evaluators need to successfully run your code using only submitted files and the most recent version of Python. Submitted code may use anything from Python's standard library, including the built-in data structures (e.g., lists, tuples, sets, and dictionaries), except for the hash table where only the use of dictionaries is prohibited.
- The project must use the package, and distance data provided in files WGUPS Distance Table.xlsx and WGUPS Package Table.xlsx found in 'View Task>Task Overview>Supporting Documents' on the C950 COS page. You may make minor changes (such as removing headers) and convert these documents into '.csv.' files. *Note: there is one  minor differences between the addresses in the package and distance files:*

  > ***Distance file:***
  > 5383 S 900 East #104 (84117)
  > ***Package file:***
  > 5383 South 900 East #104

- Code downloaded from the internet or acquired from another student or any other source may not be submitted and will result in this assessment's automatic failure.
- Submitted code may use anything from Python's standard library, including the built-in data structures (e.g., lists, tuples, sets, and dictionaries). The only exception is the hash-table, where the use of dictionaries is prohibited (a dictionary is a hash-table).

**Live and Recorded Webinars:**

- Find live webinars for all BSCS courses on the **Computer Science Webinar Calendar**.
- All C950 Recorded Webinars
- C950 project overview recorded webinar (42 minutes)
- C950 project overview (15 minutes)
- Getting started with the C950 code
- C950 - Webinar-1 - Let's Go Hashing - Recording (30 min)
- C950 - Webinar-2 - Getting Greedy, who moved my data - Recording (14 min)
- C950 - Webinar-3 - How to Dijkstra - Recording (20 min)
- C950 - Webinar-4 - Python Modules - Recording (10 min)

**Recommendations:**

- We recommend using the PyCharm IDE. You can simply zip and upload your project folder when you submit your project.
- Organize your document so that it specifically addresses the rubric requirements. For example:



Typically 2-3 sentences suffice per section. This way makes it easier to write, grade, and fix if necessary. Mention everything in the rubric requirements, even if it's redundant or superfluous. The coding/annotations sections of the directions are only suggestions.

- Avoid using the provided Sample Algorithm Overview.docx as a template for your submitted document. It gives a pseudocode example for one way (but not the only way) to satisfy Part B1 (see below).
- The supporting data files, WGUPS Distance Table.xlsx and WGUPS Package Table.xlsx, are best viewed in MS Excel. However, there are many alternatives for those without Excel. LibreOffice is freely available and runs in Windows, Mac, and Linux.
- Watch this C950 webinar: Getting Started on the project
- Make an appointment with a CI, send us an email, and read the course chatter.

**Assumptions**
- Two drivers and three trucks are available. So no more than two trucks can be away from the hub at the same time.
- The trucks move at a constant speed of 18 miles per hour.
- Trucks can carry a maximum of 16 packages.
- Trucks can leave the hub no sooner than 8:00 a.m.
- Trucks can be loaded only at the hub.
- You only need to account for the time spent driving. You can ignore the time spent on all other activities, such as loading trucks and dropping off packages.
- The wrong delivery address for package #9, Third District Juvenile Court, will be corrected at 10:20 a.m. The correct address is "410 S State St., Salt Lake City, UT 84111". You may assume that WGUPS knows the address is correct and when the correction will be available.
- Packages #13, #14, #15. #16, #19, and #20 must go out for delivery on the same truck.
- Packages #3, #18, #36, and #38 may only be delivered by truck 2.
- #6, #25, #28, #32 cannot leave the hub before 9:05 a.m.

**Comments on evaluations:**
- Course Instructors are available to answer specific questions about your project or submission. However, we do not evaluate the performance assessments. If you did your best to meet the requirements, i*t is usually best to submit the task and let the evaluator tell you which parts need correction*. You can then work with your course instructor and use the evaluator's comments to fix only those sections.
- Evaluators grade, using the rubric as a checklist. Submissions minimally meeting specific requirements of the rubric typically pass.
- Evaluators are required to respond using the following format:
    - Acknowledge anything written attempting to address the rubric section (even if incorrect or irrelevant).
    - State what is missing or incorrect addressing the rubric section.
    - Automatically mark sections as "Not Evident," i.e., red, if a dependent section does not pass. For example, if Part A is missing, Parts I1-I3A may be marked "red" automatically.
- If an evaluator writes "insufficient details…" (or something similar), the entry is typically *close* to being accepted -they just need more.

# Rubric Requirements Summary

**Note:** The official rubric is the final say on what is required to pass; evaluators use it as a checklist. The sections below paraphrase or restate the official rubric for better readability and alignment with general evaluation practices. Submissions minimally meeting specific requirements of the rubric typically pass. You can find the official rubric on your C950 COS page.

## A: ALGORITHM SELECTION

Identify by name the self-adjusting algorithm used to create a program to deliver the packages and meet all requirements specified in the scenario.

*You must identify the self-adjusting part(s) of your algorithm by name. Though you may have written your algorithm using a variety of approaches, you should be able to connect or generalize part of your algorithm to a commonly recognizable method, e.g., a greedy algorithm, farthest neighbor algorithm, etc.*

*What is self-adjusting? Any code which adjusts dynamically according to input, i.e., you give the code package info, and the code makes decisions. For example, if you manually choose ten packages and then your code decides how to deliver those packages,*

- *The code choosing the packages is not self-adjusting.*
- *The code deciding how to deliver the packages is adjusting.*

*The identified (also called "chosen" in Part I1 of the rubric and "core" in Part I of the directions) algorithm must be self-adjusting. Using non-self-adjusting methods is acceptable, but at least one part must use a self-adjusting algorithm.*

---

## B1: LOGIC COMMENTS

The submission accurately explains the algorithm's logic using pseudocode.

*Provide an outline of how your code finds a solution. You can do this in your document or code comments, though we recommend the former.*

*There is no formal way to write pseudocode (or it would not be pseudocode). Following the example found in the sample algorithm document is one way to fulfill this requirement. Still, you certainly need a step-by-step explanation (via pseudocode or bullet points) outlining how your code finds a solution.*

---

## B2: DEVELOPMENT ENVIRONMENT

The submission accurately describes the software and hardware used to create the Python application.

*You should include brief descriptions of the programming language, IDE, operating systems, hardware, etc., used to write and run your application.*

---

## B3: SPACE-TIME AND BIG-O

The (code) submission accurately describes the space-time complexity for each major block of code and the entire program using Big-O notation.

*Include this in code comments. "Major block" of codes and "accuracy" is not defined. Ask yourself, "if I was looking at this code for the first time, what would I want to be described? How much detail would I need?"*

*How accurately does the time complexity description need to be? Big-O classifications are sets of functions defined by an upper bound. e.g., a function that is in O(1) is also in O(n). However, it is reasonable for evaluators to expect better accuracy in obvious cases, e.g., if it's constant -write O(1). For more complicated situations, less accuracy should be acceptable.*

*If you are uncertain about whether comments or accuracy is sufficient -submit it. At worst, the evaluator will ask for more.*

---

## B4: ADAPTABILITY

The submission application accurately explains the application's capability to scale and adapt to an increasing number of packages.

*The application will have at least two scalable elements, the self-adjusting algorithm from [Part A](#) and the self-adjusting data structure (hash-table) from [Part D](#). The discussion can also include shortcomings, e.g., "X will not scale well because…"*

---

## B5: SOFTWARE EFFICIENCY AND MAINTAINABILITY

The discussion addresses how the software is efficient and easy to maintain.

*By "efficiency," they mean the Big-O time complexity of your entire program. Your program must run in polynomial time or better to be efficient.*

*For "maintainability," describe how future developers can easily understand, repair, and enhance your code, i.e., -[software maintainability](#)—for example, code structure, comments, compartmentalization, etc.*

---

# B6: SELF-ADJUSTING DATA STRUCTURES

The discussion addresses the strengths and weaknesses of the self-adjusting data structures (including the hash-table).

Include all self-adjusting (non-builtin) data structures in the discussion. However, you are only required to have one such data structure -the hash-table from Part D. The rubric specifies "strength-s" and "weaknesses." So include at least two for each.

---

# C: ORIGINAL CODE

The code is original and runs without errors or warnings.

*The code should run using a standard installation of the latest version of Python 3. The evaluators should not need to download any libraries, e.g., pandas. Most submissions run through an IDE; evaluators will use PyCharm. So if you do something different, say running it through the console, consider including special instructions.*

---

# C1: IDENTIFICATION INFORMATION

The initial comment within a file named "main.py" includes your first name, last name, and student ID.

*Include these comments in a "main" file. Easy to satisfy, but easy to forget!  Also, as this section requires a "main.py" file, it makes sense to have this file be your application's entry point. Though it is not required, if you do something different, provide specific instructions on how to run your application.*

---

# C2: PROCESS AND FLOW COMMENTS

Include comments within the code adequately explaining the process and flow of the program.

*Explain the intent and decisions of each "major" block of code, i.e., the "why, what, and how." The comments should improve readability. Provide a little more detail for any process that is unusual or complicated.*

*This can be done alongside comments satisfying Part B3. Similarly, which parts require explanation and what qualifies as "adequately" is highly subjective. Ask yourself, "if I was looking at this code for the first time, what would I want to be described? How much detail would I need?"*

---

# D: DATA STRUCTURE

The submission identifies a self-adjusting data structure that can store the package information and perform well with Part A's algorithm.

*As with [Part A](#), self-adjusting means any code which adjusts dynamically according to input. This can include adjusting to size (lists are mutable in Python) or searches. For example, a hash table that can adapt to more packages without rewriting the code would be self-adjusting. This data structure must be the same hash-table in [Part E](#) and [Part F](#). Nothing else regarding the complexity of your hash table is required, e.g., it can be a 1-1 mapping, does not have to handle collisions, have chaining, etc.*

*The official task directions include a note:*
> "Note: Use only appropriate built-in data structures, except dictionaries."

*Submitted code may use anything from [Python's standard library](#), including the built-in data structures (e.g., lists, tuples, sets, and dictionaries). The only exception is the hash-table, where the use of dictionaries is prohibited (a dictionary is a hash-table).*

*Per parts E and F, the hash table must have the following:*
- *E: an insertion function that includes as input all a package's info (see below).*
- *F: a look-up function that uses the package's ID as input and returns the corresponding package's information (see below).*

*The ability to store and retrieve package info (via the package's ID) is the only requirement. The information can be stored in an object and include additional parameters, e.g., special notes, time the package left the hub, etc.*

*The insert function (Part E) and look-up function (Part F) must respectively store and retrieve the following information:*
- *package ID number*
- *delivery address*
- *delivery deadline*
- *delivery city*
- *delivery zip code*
- *package weight*
- *delivery status (at the hub, en route, or delivery time)*

---

# D1: EXPLANATION OF DATA STRUCTURE

The submission accurately explains how the data structure (hash-table) uses package IDs to store and retrieve package information.

*Provide an explanation that describes the hash table's logic, i.e., how it stores and retrieves package information. You should include a description of why your hash-table retrieves information accurately and more efficiently than a simple linear search.*

# E: HASH TABLE

The hash table has an insertion function that stores all of the given components (listed in Part D) using the package ID as the key.

# F: LOOK-UP FUNCTION

The provided hash table should include a look-up function that can use a package's ID to retrieve all of the same package's components from the hash table (listed in Part D).

*The package ID must be the key, and the components from Part D must be the values. There are no other specifications, and you can choose how the values within the hash-table are stored. For example, the components could be in an object or a different data structure of your choosing.*

# G: INTERFACE

Provide an intuitive interface for the user to view the status and information (as listed in Part D)  of any package at any time and the total mileage traveled by all trucks.  The delivery status should report the package as "at the hub," "en route," or "delivered at _____." Delivery status must include the time.

*This can be done through a simple console interface (it might be preferable). The evaluators need an easy way to check that all packages were delivered according to their constraints using under 140 miles. "Intuitive" means that evaluators can do this through the interface without reading your code.*

*To check package statuses,  you should be able to enter a time and check all packages' statuses* (delivered at X, at the hub, or en route). For example, in the console, it might look like:

```
Enter a time after 08:00:00 (in military time, i.e., 0900) to check package
status or 'X' to exit:
0950
        Package ID: 1, Address:  195 W Oakland Ave, ….Delivered at 08:46:20
        Package ID: 2, Address:  2530 S 500 E At Hub, …. At the hub.
        .
        .
        .
```

# G1-G3: 1st, 2nd, and 3rd status checks.

*Provide screenshots showing the information (as listed in [Part D](#)) and statuses (see [Part G](#)) at a time between:*
- *8:35 a.m. and 9:25 a.m.*
- *9:35 a.m. and 10:25 a.m.*
- *12:03 p.m. and 1:12 p.m.*

Provide one screenshot within each of the time intervals above. If everything does not fit on a single screenshot, you can use multiple images; label them, so it's clear which time interval they cover, e.g., "G1-a.jpg, G1-b.jpg,..." The screenshots can be included anywhere in your submission, e.g., the document, separately, in the project folder -but make them easy to find.

---

# H: SCREENSHOTS OF CODE EXECUTION

Provide a screenshot or screenshots showing the total delivery mileage and successful completion of the code free from runtime errors or warnings.

*Provide a screenshot or screenshots so that the evaluator can check the mileage and that your code ran successfully to completion. The screenshot(s) should include a view of the console output, the project files, etc. The screenshot(s) can be added anywhere in your submission, e.g., the document, separately, in the project folder -but make it easy to find.*

---

# I1: STRENGTHS OF THE CHOSEN ALGORITHM

Identify at least two specific strengths of the algorithm from [Part A](#) relevant to finding a solution.

*Strengths can include anything, e.g., efficiency, simplicity, adaptability, etc. You also need to justify that these are strengths relative to the project.*

---

# I2: VERIFICATION OF ALGORITHM

Verify that the algorithm used in the application meets all the requirements by:
- Provide the total combined miles traveled by all trucks. It must be less than 140.
- State that all packages were delivered on time.
- State that all packages were delivered according to their delivery specifications.
- Describe how all the above points are verifiable through the user interface.

*By "requirements," they mean requirements of the solution -not requirements of the algorithm listed in Part A. Evaluators should be able to verify the mileage and deliveries via the user interface. So this section*

*may seem a little odd as it equates verification with "stating," but that is what they ask you to do.*

# I3: OTHER POSSIBLE ALGORITHMS

The submission identifies two algorithms different from the one provided in Part A that could meet the scenario's requirements.

*The two alternative algorithms only need to be different from the algorithm identified in Part A; they do not need to be equitable or better in completely-known. Furthermore, these two algorithms could apply to any portion of the application. The problem of finding a delivery route is known as the "Traveling Salesperson Problem" or TSP. An old and well-known problem, there are many, many approaches to this problem.*

# I3A: ALGORITHM DIFFERENCES

*The description includes attributes of each algorithm identified in Part I3 and how the identified attributes compare to the algorithm's attributes from Part A.*

Compare the two alternative algorithms to the algorithm identified in Part A. Attributes, and the comparison can include almost anything, e.g., time-complexity, advantages, disadvantages, etc. The rubric writes "attribute-s." So you should list at least two attributes per algorithm list in Part I3.

# J: DIFFERENT APPROACH

*The description includes at least one aspect of the process that the candidate would do differently and includes how the candidate would modify the process.*

# K1: VERIFICATION OF DATA STRUCTURE

Verify that the data structure used in the application meets all the requirements by:
- Provide the total combined miles traveled by all trucks. It must be less than 140.
- State that all packages were delivered on time.
- State that all packages were delivered according to their delivery specifications.
- State that an "efficient" hash-table (Part D) with a look-up function (Part F) is present.
- State that the "reporting" (package statuses and information) can be verified through the user interface and that all information is accurate (Part G).

*This section looks to have been cut and pasted from Part I2. It is confusing that you are asked to re-verify mileage and delivery statuses, particularly as these items are not relevant to the hash table. However, these items are specified in the rubric. When the rubric is redundant -be redundant.*

## K1A: EFFICIENCY

The discussion accurately explains how adding packages directly affects the time needed to complete the look-up                                                                                                    function.

*The "look-up function" refers to the hash table's look-up function identified in Part D. Describe how adding more packages affects the time it takes to retrieve package information. The effect could be nil, as in the case of a direct (1-1) mapping. Whatever the case, provide a brief explanation.*

## K1B: OVERHEAD

The discussion accurately explains how adding packages directly affects the data structure space usage.

## K1C: IMPLICATIONS

The discussion accurately explains adding trucks or cities would affect look-up time and space usage.

*Look-up and space usage are both referring to the hash-table. Depending on your code, additional cities or trucks may not affect hash-table performance. In which case, you should explain why.*

## K2: OTHER DATA STRUCTURES

The submission identifies two data structures other than the one used in [Part D]() to meet the requirements in the scenario.

*Identify two alternative data structures and justify why they could have been used as your hash-table. The alternatives can include modifications of the hash table listed in [Part D]().*

## K2A: DATA STRUCTURES DIFFERENCES

The submission accurately describes attributes of both data structures identified in [part K2]() and compares these to [Part D]()'s data structure attributes.

*Attributes and the comparison can include almost anything, e.g., mappings, structure, advantages, disadvantages, etc. The rubric writes "attribute-s." So you should list at least two attributes per data structure from* part K2*.*

---

# L: SOURCES

The submission includes in-text citations for properly quoted sources, paraphrased, or summarized and a reference list that accurately identifies the author, date, title, and source location as available.

*You must follow (APA standards). Every reference listed must have a matching in-text citation; sources not warranting an in-text citation should be excluded. If you used no sources, include a sources section in your documentation stating "no outside sources were used." You should include all code in code comments. Contact the* writing center *for questions or help with this portion.*

---

# M: PROFESSIONAL COMMUNICATION

The content reflects an attention to detail, is organized, and focuses on the main ideas as prescribed in the task or chosen by the candidate. Terminology is pertinent, is used correctly, and effectively conveys the intended meaning. Mechanics, usage, and grammar promote accurate interpretation and understanding.

*Don't overlook this! Minor grammar errors are one of the most common reasons for rejected submissions. The submitted document should be grammatically correct and easy to read. Make use of one of the freely available* grammar *checkers or the* writing center*.* **The evaluators use grammarly.com.**

*For any task returned for Professional Communication, a document with marked errors will be linked at the top of the evaluator comment page -not in section M.*
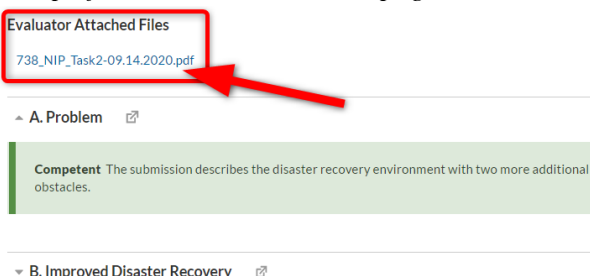


*Though the comments will advise that all errors may not be marked, correcting only the marked errors has always been sufficient to pass.*

---

# C950 Performance Assessment FAQ

**Do I have to use Python?**
Yes. The project requires that you use a version compatible with the latest version of Python 3.

**Am I required to use a specific IDE?**
You can use whatever IDE you like, but we recommend the free community version of PyCharm.

**Do all packages have to be delivered on time?**
Yes. Packages must be delivered on or before their delivery time. Packages labeled "EOD" must be delivered before the "End Of Day." "EOD" is not specified, but it's reasonable to assume that it's 5:00 pm. However, solutions almost always complete before 3:00 pm.

**Why are there three trucks?**
Only two drivers are available, and trucks have unlimited gas and load instantly. Perhaps, a third truck makes instantaneous load times realistic (truck three can be loaded while the others are out). However, it is not required to use truck 3 (or truck 1).

**What is the maximum total mileage allowed for trucks?**
Less than 140 miles. This is what the official task directions mean by "optimal." Both heuristics and optimizing algorithms are essential to get a good solution.

**Does the solution have to be optimal?**
The delivery problem is a variation of the traveling salesperson problem (TSP) known as NP-Hard. Even if you found the optimal solution, we couldn't verify it without checking every possible route (and we don't have enough time for that!). Furthermore, the task requires your algorithm to be "efficient," meaning it must run in polynomial time. If you find a polynomial-time algorithm for solving the TSP, please let us know as this would answer the P versus NP problem -a millennium problem with a one million dollar prize.

Therefore, you should use an algorithm that finds a solution and sufficient optimization. For this, many approaches will result in far less than 140 miles.

**Do I need to use any algorithms not covered in the learning materials?**
No, but you certainly can. However, a program using only the covered algorithms (plus some heuristics) can find a solution.

**Can I use external third-party libraries?**
No. Only submit code that you've written yourself or from the standard built-in Python library. Evaluators will need to run your code on their machines without installing additional resources.

**Can I use the built-in Python dictionary construct for my hash table?**
No. Your hash table can use lists, sets, or tuples. You can use dictionaries elsewhere in your project.

**Does my program have to parse the excel data straight from excel?**

No. You can export the data from excel to text or CSV file. Include these files with your submitted project. You can clean up the file manually, making it easier to import, e.g., removing headers.

**The Excel mileage matrix has the lower half filled in, so it only has mileage entries in one direction, for example, from point A to point B, but not from point B to point A. Does that mean a truck has to find an alternate route?**

No. The distance table is bi-directional, i.e., the distance from A to B equals B to A. Furthermore, the distance table is a fully connected graph, i.e., it provides a direct path between every address. So using a graph data structure is not necessary.

However, the triangle inequality does not hold. Meaning the provided distance from A to B mileage might not always be the shortest path! For example, from the distance table: hub-->Taylorsville is 11.0, but hub--> Valley Regional--> Taylorsville is 6.4+0.6 =7 miles. You can optimize the distance table via a pathfinding algorithm, such as Dijkstra's. However, this step is not necessary to find a solution under 140 miles.

**Does my program need to have an interactive external user interface?**

Yes. See Part G above. The 'user' needs to check the status of any package at any time conveniently. For example, the user should look up package #19 at 10:43 am and check the info and status. Having the user provide a time and printing the information and statuses of all the packages will meet this requirement. We recommend a simple command-line interface.

**Does the user need to be able to update the package's address?**

There is no requirement for this, and the code should be able to complete the delivery simulation using only the provided data.

**How do we handle the package with the wrong address (#9)?**

The wrong delivery address for package #9, Third District Juvenile Court, will be corrected at 10:20 a.m. The correct address is "410 S State St., Salt Lake City, UT 84111". There is no specification for handling this situation other than package #9 cannot be delivered to the correct address before 10:20 a.m. It is acceptable to assume WGUPS knows the address is incorrect and when it will be corrected. Hence effectively treating package #9 like a delayed package.

**How do we handle packages with the special notes:**
- Package #14: "Must be delivered with 15, 19."
- Package #16: "Must be delivered with 13, 19."
- Package #20:  "Must be delivered with 13, 15."
- Packages #13, #14, #15. #16, #19, and #20 must go out for delivery on the same truck simultaneously, e.g., those packages all leave the hub at 9:30 on truck 2.

**Can I change the data files?**

Yes, you can make non-meaningful changes to [WGUPS Distance Table.xlsx](#) and [WGUPS Package Table.xlsx](#). Don't change the distances! But you may remove headers, change address formats, e.g., 'S' to 'South', and convert the .xlsx files into a '.csv' file (necessary for importing the data).

*Note: there is one  minor differences between the addresses in the package and distance files:*

**Distance file:**
5383 S 900 East #104 (84117)
**Package file:**
5383 South 900 East #104