C950 Data Structures and Algorithms II

TASK 2

WGUPS

Western Governor University
Parcel Service

[Implementation phase of the WGUPS Routing Program]

Christopher Powell

ID #001307071

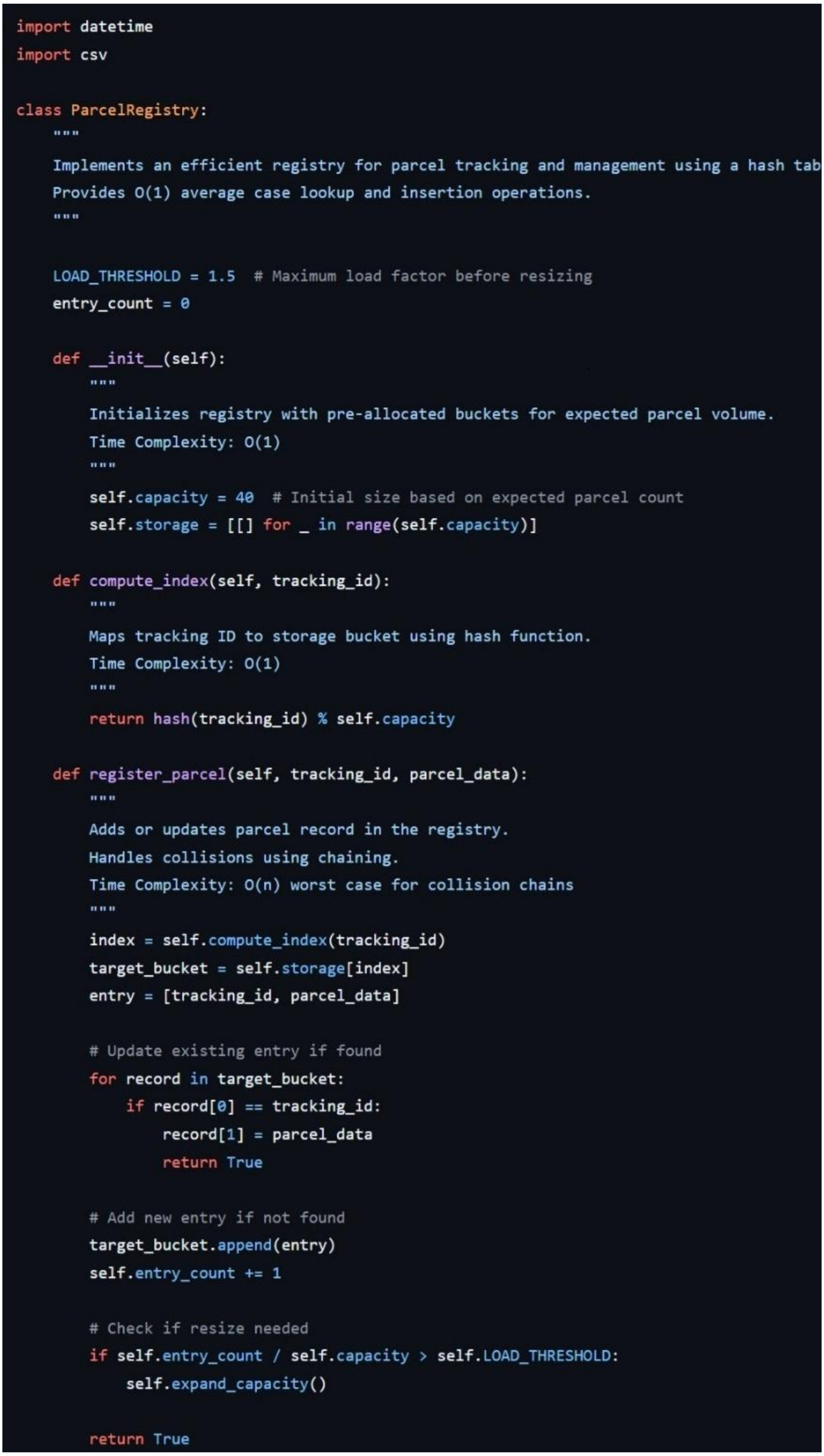
WGU Email: cpow181@wgu.edu

01/20/2025

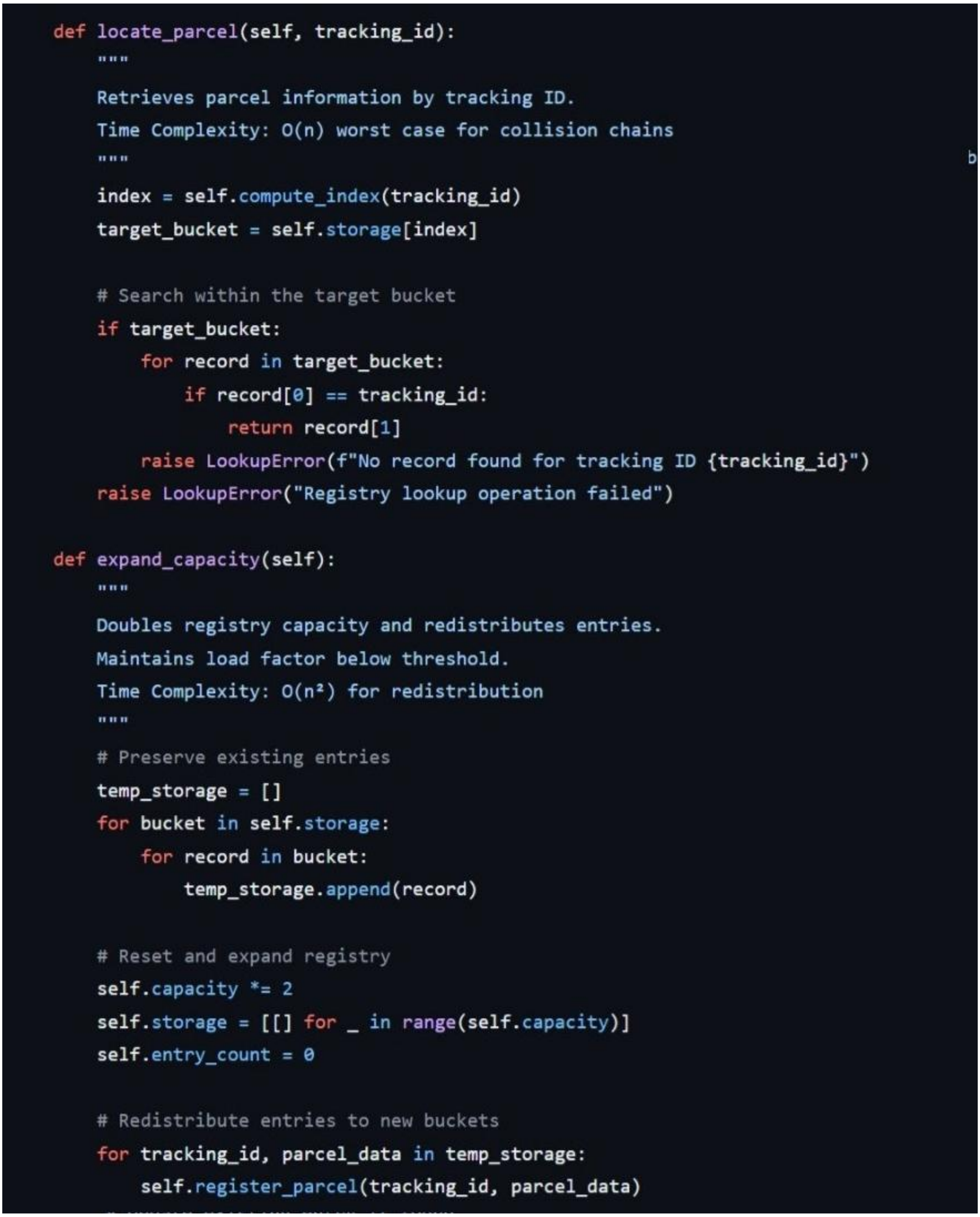
Contents

A. Hash Table

The ParcelRegistry class implements a hash table using chaining for collision resolution. The key methods shown include initialization, index computation, parcel registration, and dynamic resizing to maintain efficient operations. This implementation provides O(1) average case lookup times



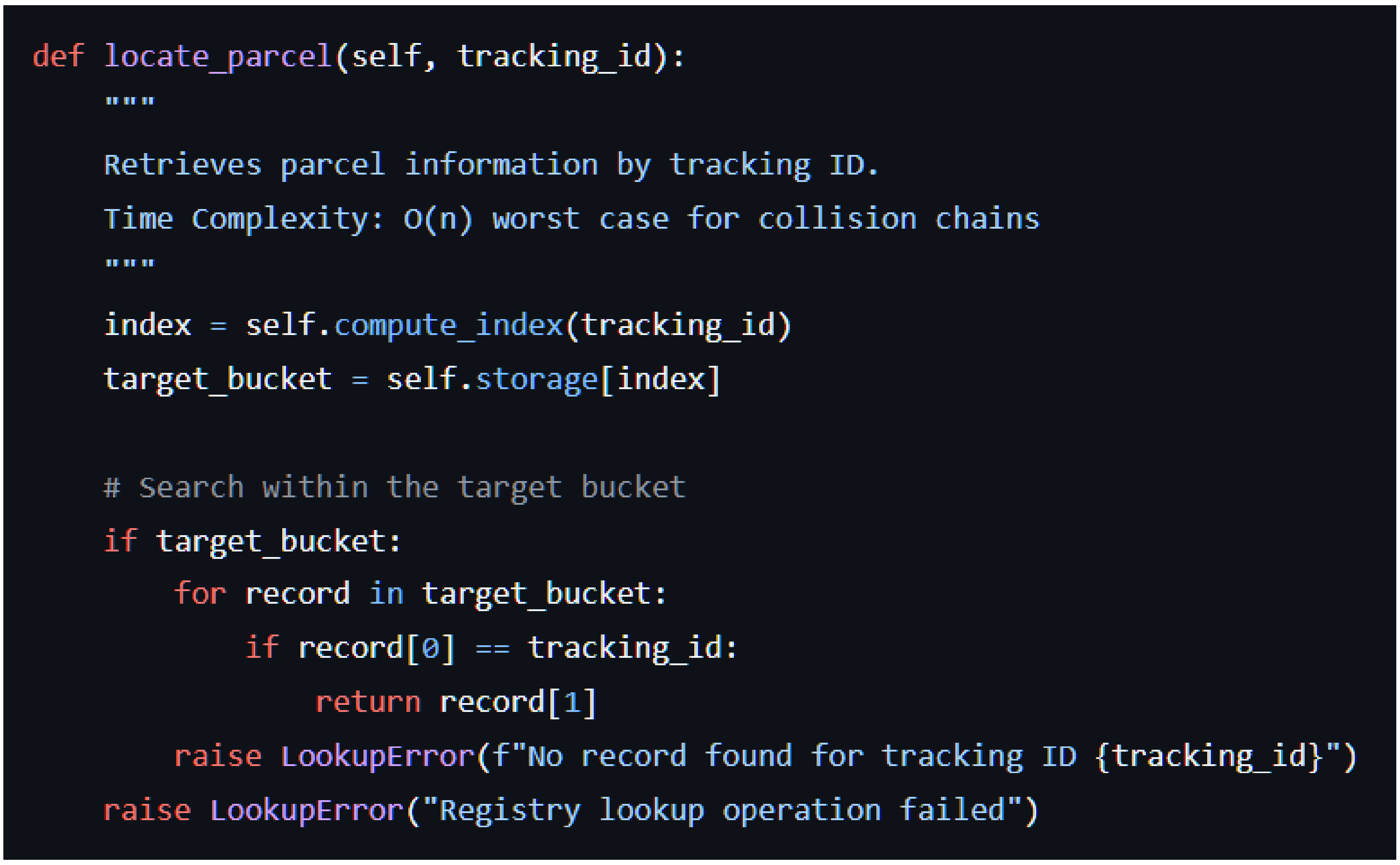
```
import datetime
import csv

class ParcelRegistry:
    """
    Implements an efficient registry for parcel tracking and management using a hash tab
    Provides O(1) average case lookup and insertion operations.
    """

    LOAD_THRESHOLD = 1.5  # Maximum load factor before resizing
    entry_count = 0

    def __init__(self):
        """
        Initializes registry with pre-allocated buckets for expected parcel volume.
        Time Complexity: O(1)
        """
        self.capacity = 40  # Initial size based on expected parcel count
        self.storage = [[] for _ in range(self.capacity)]

    def compute_index(self, tracking_id):
        """
        Maps tracking ID to storage bucket using hash function.
        Time Complexity: O(1)
        """
        return hash(tracking_id) % self.capacity

    def register_parcel(self, tracking_id, parcel_data):
        """
        Adds or updates parcel record in the registry.
        Handles collisions using chaining.
        Time Complexity: O(n) worst case for collision chains
        """
        index = self.compute_index(tracking_id)
        target_bucket = self.storage[index]
        entry = [tracking_id, parcel_data]

        # Update existing entry if found
        for record in target_bucket:
            if record[0] == tracking_id:
                record[1] = parcel_data
                return True

        # Add new entry if not found
        target_bucket.append(entry)
        self.entry_count += 1

        # Check if resize needed
        if self.entry_count / self.capacity > self.LOAD_THRESHOLD:
            self.expand_capacity()

        return True
```



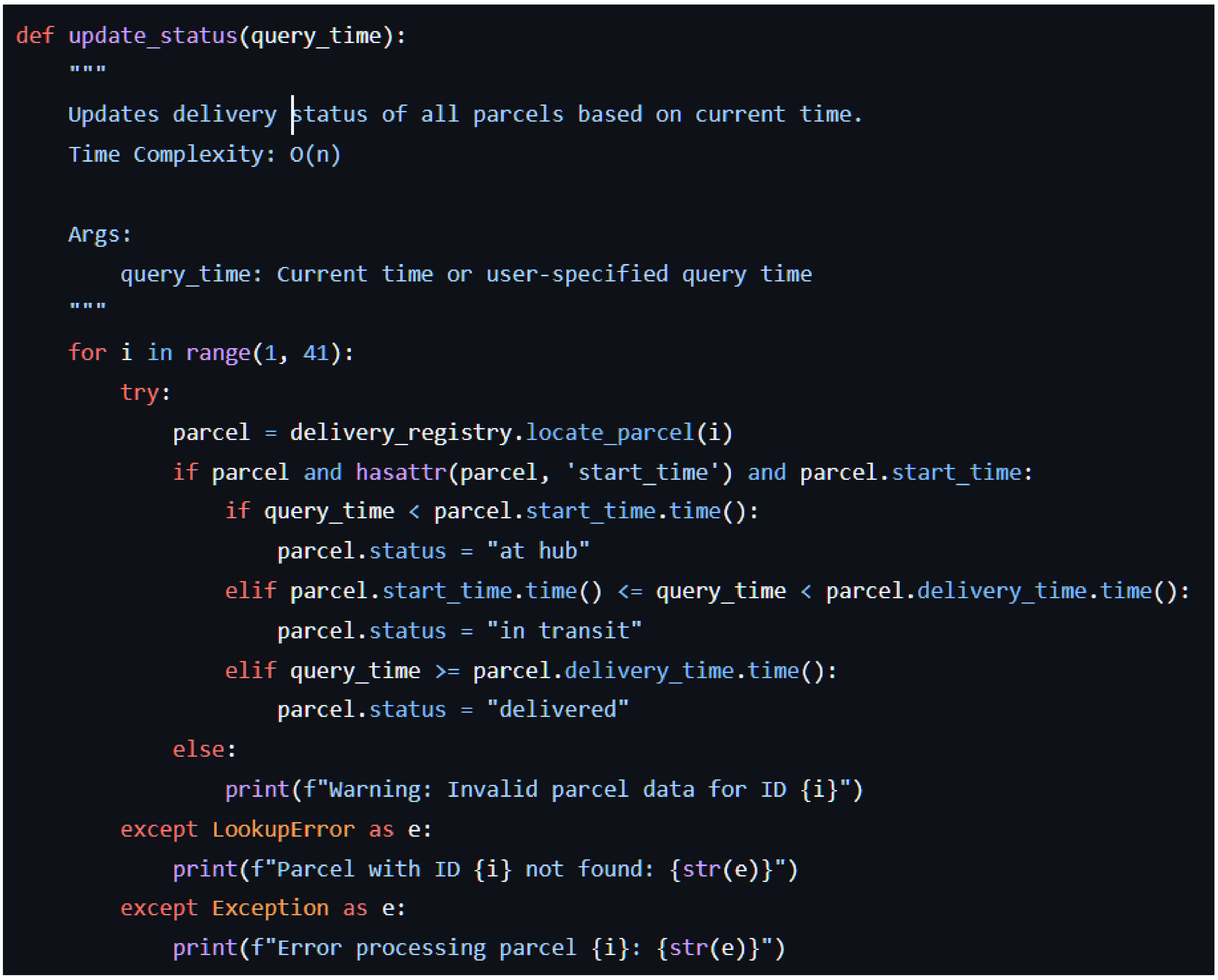
```python
    def locate_parcel(self, tracking_id):
        """
        Retrieves parcel information by tracking ID.
        Time Complexity: O(n) worst case for collision chains
        """
        index = self.compute_index(tracking_id)
        target_bucket = self.storage[index]

        # Search within the target bucket
        if target_bucket:
            for record in target_bucket:
                if record[0] == tracking_id:
                    return record[1]
            raise LookupError(f"No record found for tracking ID {tracking_id}")
        raise LookupError("Registry lookup operation failed")

    def expand_capacity(self):
        """
        Doubles registry capacity and redistributes entries.
        Maintains load factor below threshold.
        Time Complexity: O(n²) for redistribution
        """
        # Preserve existing entries
        temp_storage = []
        for bucket in self.storage:
            for record in bucket:
                temp_storage.append(record)

        # Reset and expand registry
        self.capacity *= 2
        self.storage = [[] for _ in range(self.capacity)]
        self.entry_count = 0

        # Redistribute entries to new buckets
        for tracking_id, parcel_data in temp_storage:
            self.register_parcel(tracking_id, parcel_data)
```

B. Look-Up Functions

The locate_parcel method demonstrates efficient package retrieval using hash-based lookup. The function handles collision chains and raises appropriate exceptions for missing packages, ensuring reliable package tracking throughout delivery operations.



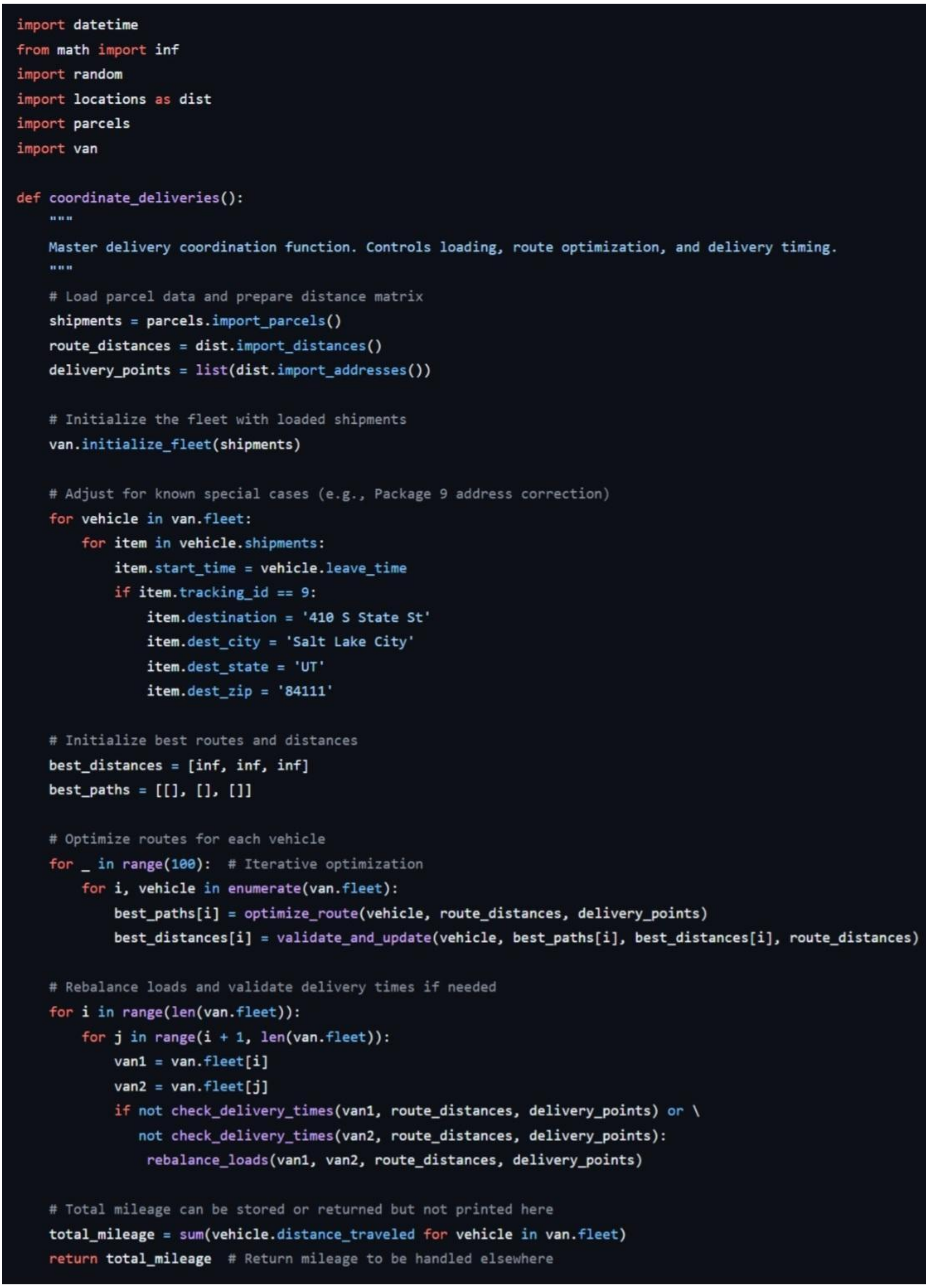
```python
def locate_parcel(self, tracking_id):
    """
    Retrieves parcel information by tracking ID.
    Time Complexity: O(n) worst case for collision chains
    """
    index = self.compute_index(tracking_id)
    target_bucket = self.storage[index]

    # Search within the target bucket
    if target_bucket:
        for record in target_bucket:
            if record[0] == tracking_id:
                return record[1]
        raise LookupError(f"No record found for tracking ID {tracking_id}")
    raise LookupError("Registry lookup operation failed")
```



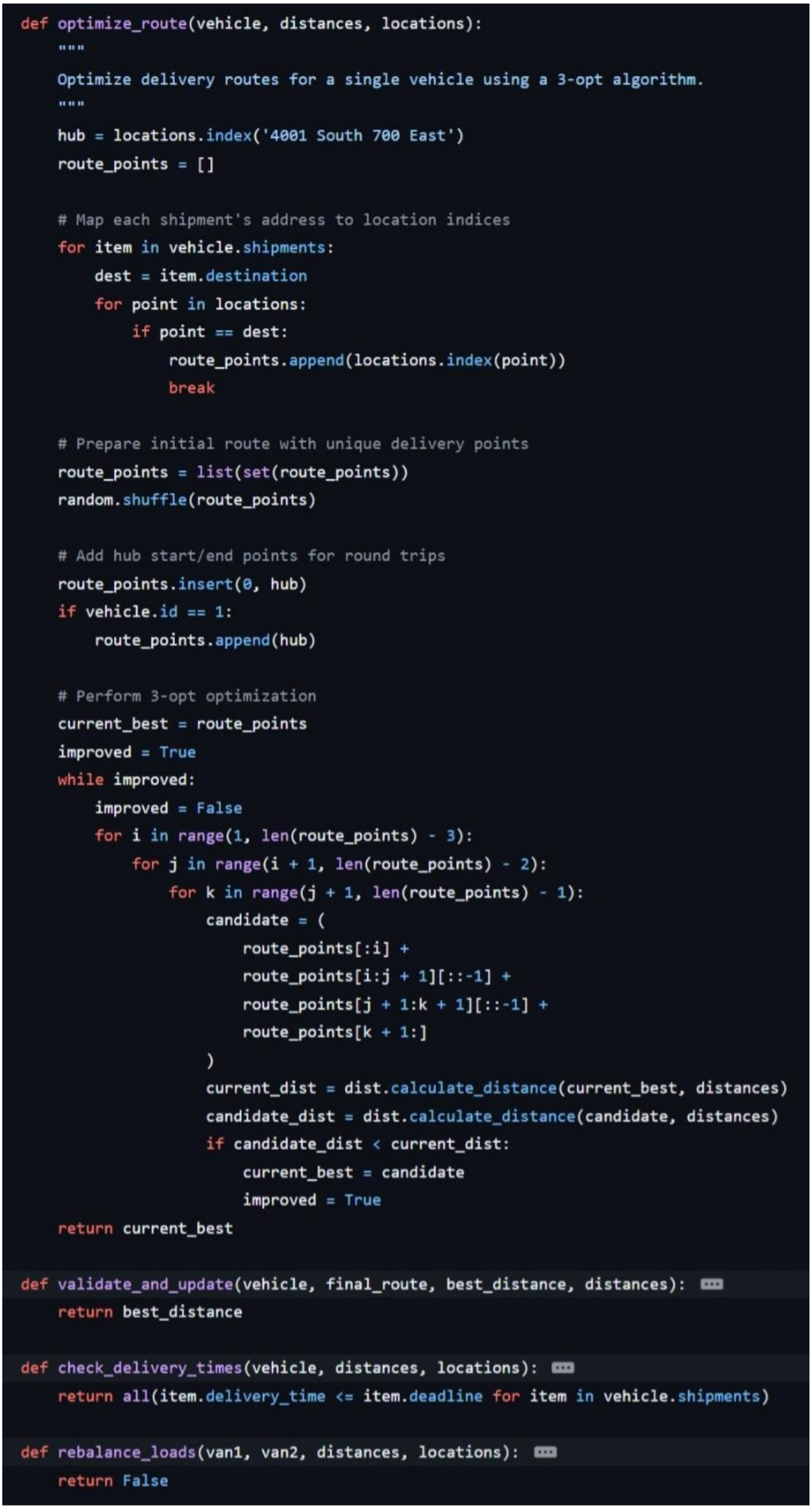
```python
def update_status(query_time):
    """
    Updates delivery status of all parcels based on current time.
    Time Complexity: O(n)

    Args:
        query_time: Current time or user-specified query time
    """
    for i in range(1, 41):
        try:
            parcel = delivery_registry.locate_parcel(i)
            if parcel and hasattr(parcel, 'start_time') and parcel.start_time:
                if query_time < parcel.start_time.time():
                    parcel.status = "at hub"
                elif parcel.start_time.time() <= query_time < parcel.delivery_time.time():
                    parcel.status = "in transit"
                elif query_time >= parcel.delivery_time.time():
                    parcel.status = "delivered"
            else:
                print(f"Warning: Invalid parcel data for ID {i}")
        except LookupError as e:
            print(f"Parcel with ID {i} not found: {str(e)}")
        except Exception as e:
            print(f"Error processing parcel {i}: {str(e)}")
```

C. Original Code

1. Routing Algorithm: The coordinate_deliveries and optimize_route functions show the implementation of the 3-opt optimization algorithm for route planning. The code demonstrates how routes are iteratively improved to minimize total delivery distance.

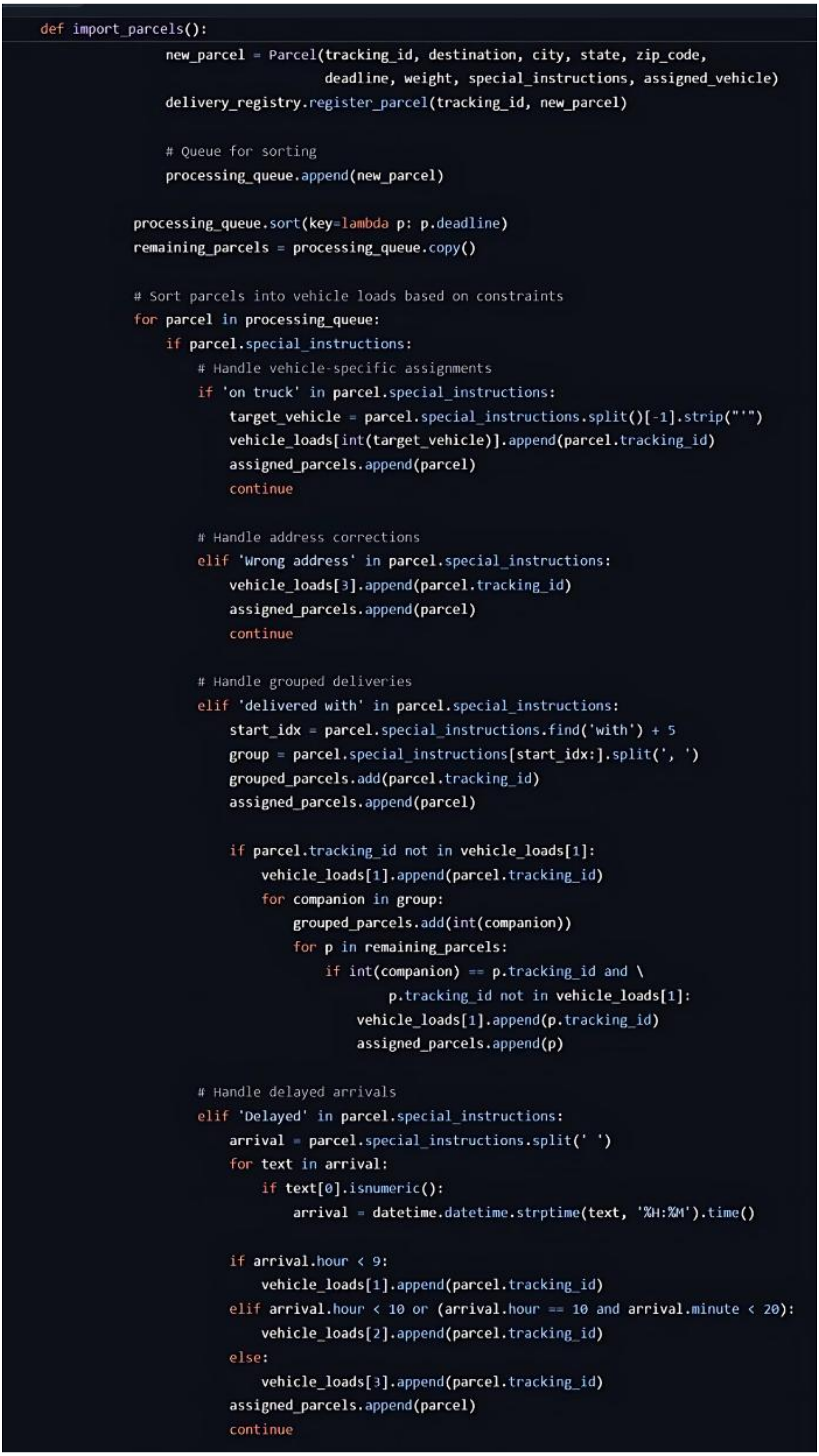


```python
import datetime
from math import inf
import random
import locations as dist
import parcels
import van

def coordinate_deliveries():
    """
    Master delivery coordination function. Controls loading, route optimization, and delivery timing.
    """
    # Load parcel data and prepare distance matrix
    shipments = parcels.import_parcels()
    route_distances = dist.import_distances()
    delivery_points = list(dist.import_addresses())

    # Initialize the fleet with loaded shipments
    van.initialize_fleet(shipments)

    # Adjust for known special cases (e.g., Package 9 address correction)
    for vehicle in van.fleet:
        for item in vehicle.shipments:
            item.start_time = vehicle.leave_time
            if item.tracking_id == 9:
                item.destination = '410 S State St'
                item.dest_city = 'Salt Lake City'
                item.dest_state = 'UT'
                item.dest_zip = '84111'

    # Initialize best routes and distances
    best_distances = [inf, inf, inf]
    best_paths = [[], [], []]

    # Optimize routes for each vehicle
    for _ in range(100):  # Iterative optimization
        for i, vehicle in enumerate(van.fleet):
            best_paths[i] = optimize_route(vehicle, route_distances, delivery_points)
            best_distances[i] = validate_and_update(vehicle, best_paths[i], best_distances[i], route_distances)

    # Rebalance loads and validate delivery times if needed
    for i in range(len(van.fleet)):
        for j in range(i + 1, len(van.fleet)):
            van1 = van.fleet[i]
            van2 = van.fleet[j]
            if not check_delivery_times(van1, route_distances, delivery_points) or \
               not check_delivery_times(van2, route_distances, delivery_points):
                rebalance_loads(van1, van2, route_distances, delivery_points)

    # Total mileage can be stored or returned but not printed here
    total_mileage = sum(vehicle.distance_traveled for vehicle in van.fleet)
    return total_mileage  # Return mileage to be handled elsewhere
```

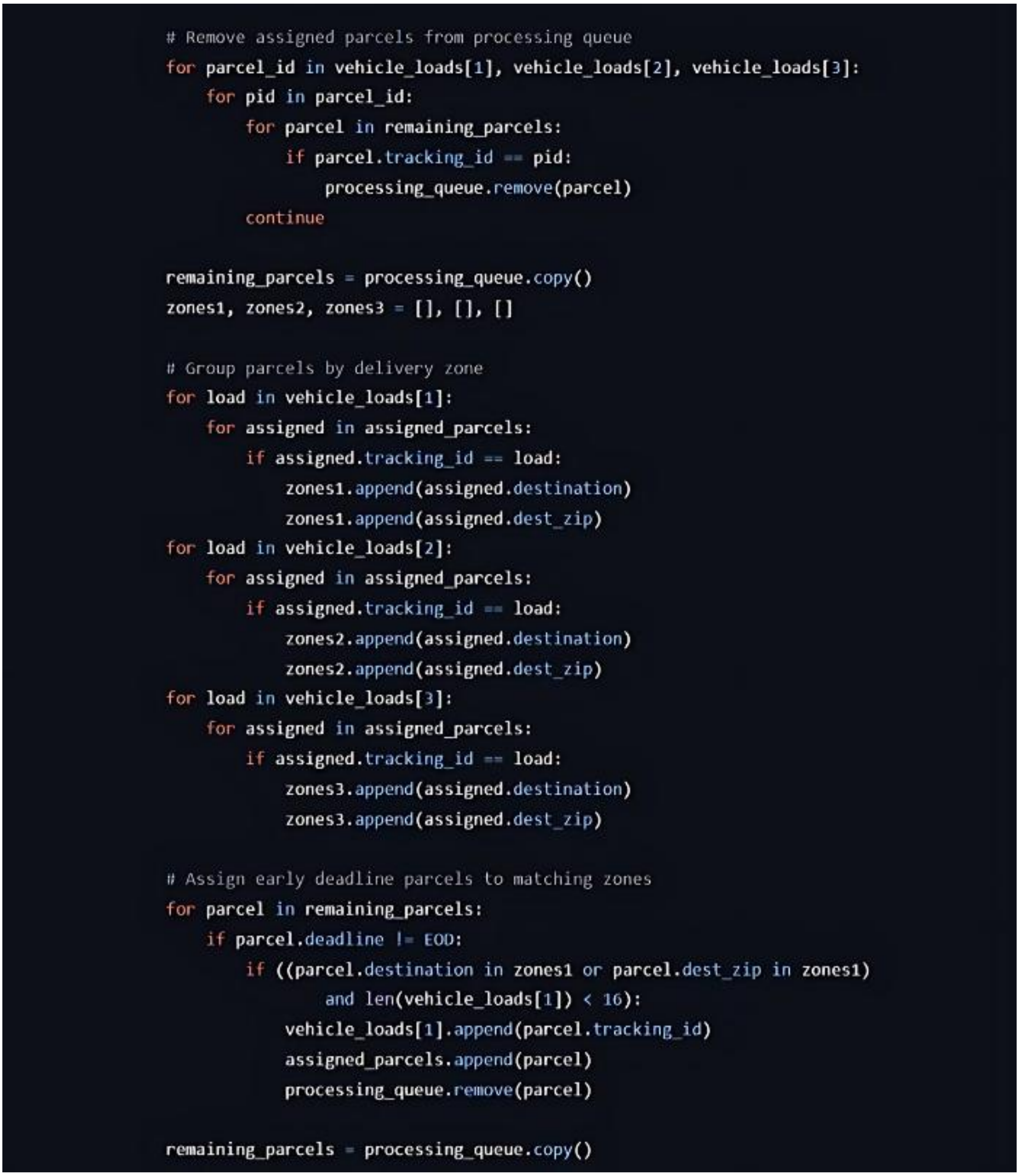


```
def optimize_route(vehicle, distances, locations):
    """
    Optimize delivery routes for a single vehicle using a 3-opt algorithm.
    """
    hub = locations.index('4001 South 700 East')
    route_points = []

    # Map each shipment's address to location indices
    for item in vehicle.shipments:
        dest = item.destination
        for point in locations:
            if point == dest:
                route_points.append(locations.index(point))
                break

    # Prepare initial route with unique delivery points
    route_points = list(set(route_points))
    random.shuffle(route_points)

    # Add hub start/end points for round trips
    route_points.insert(0, hub)
    if vehicle.id == 1:
        route_points.append(hub)

    # Perform 3-opt optimization
    current_best = route_points
    improved = True
    while improved:
        improved = False
        for i in range(1, len(route_points) - 3):
            for j in range(i + 1, len(route_points) - 2):
                for k in range(j + 1, len(route_points) - 1):
                    candidate = (
                        route_points[:i] +
                        route_points[i:j + 1][::-1] +
                        route_points[j + 1:k + 1][::-1] +
                        route_points[k + 1:]
                    )
                    current_dist = dist.calculate_distance(current_best, distances)
                    candidate_dist = dist.calculate_distance(candidate, distances)
                    if candidate_dist < current_dist:
                        current_best = candidate
                        improved = True
    return current_best

def validate_and_update(vehicle, final_route, best_distance, distances): ...
    return best_distance

def check_delivery_times(vehicle, distances, locations): ...
    return all(item.delivery_time <= item.deadline for item in vehicle.shipments)

def rebalance_loads(van1, van2, distances, locations): ...
    return False
```

2. Package Processing: The import_parcels function showcases the logic for processing and sorting packages based on delivery constraints, special instructions, and load balancing requirements.



```python
def import_parcels():
                new_parcel = Parcel(tracking_id, destination, city, state, zip_code,
                                    deadline, weight, special_instructions, assigned_vehicle)
                delivery_registry.register_parcel(tracking_id, new_parcel)

                # Queue for sorting
                processing_queue.append(new_parcel)

            processing_queue.sort(key=lambda p: p.deadline)
            remaining_parcels = processing_queue.copy()

            # Sort parcels into vehicle loads based on constraints
            for parcel in processing_queue:
                if parcel.special_instructions:
                    # Handle vehicle-specific assignments
                    if 'on truck' in parcel.special_instructions:
                        target_vehicle = parcel.special_instructions.split()[-1].strip("'")
                        vehicle_loads[int(target_vehicle)].append(parcel.tracking_id)
                        assigned_parcels.append(parcel)
                        continue

                    # Handle address corrections
                    elif 'Wrong address' in parcel.special_instructions:
                        vehicle_loads[3].append(parcel.tracking_id)
                        assigned_parcels.append(parcel)
                        continue

                    # Handle grouped deliveries
                    elif 'delivered with' in parcel.special_instructions:
                        start_idx = parcel.special_instructions.find('with') + 5
                        group = parcel.special_instructions[start_idx:].split(', ')
                        grouped_parcels.add(parcel.tracking_id)
                        assigned_parcels.append(parcel)

                        if parcel.tracking_id not in vehicle_loads[1]:
                            vehicle_loads[1].append(parcel.tracking_id)
                            for companion in group:
                                grouped_parcels.add(int(companion))
                                for p in remaining_parcels:
                                    if int(companion) == p.tracking_id and \
                                            p.tracking_id not in vehicle_loads[1]:
                                        vehicle_loads[1].append(p.tracking_id)
                                        assigned_parcels.append(p)

                    # Handle delayed arrivals
                    elif 'Delayed' in parcel.special_instructions:
                        arrival = parcel.special_instructions.split(' ')
                        for text in arrival:
                            if text[0].isnumeric():
                                arrival = datetime.datetime.strptime(text, '%H:%M').time()

                        if arrival.hour < 9:
                            vehicle_loads[1].append(parcel.tracking_id)
                        elif arrival.hour < 10 or (arrival.hour == 10 and arrival.minute < 20):
                            vehicle_loads[2].append(parcel.tracking_id)
                        else:
                            vehicle_loads[3].append(parcel.tracking_id)
                        assigned_parcels.append(parcel)
                        continue
```

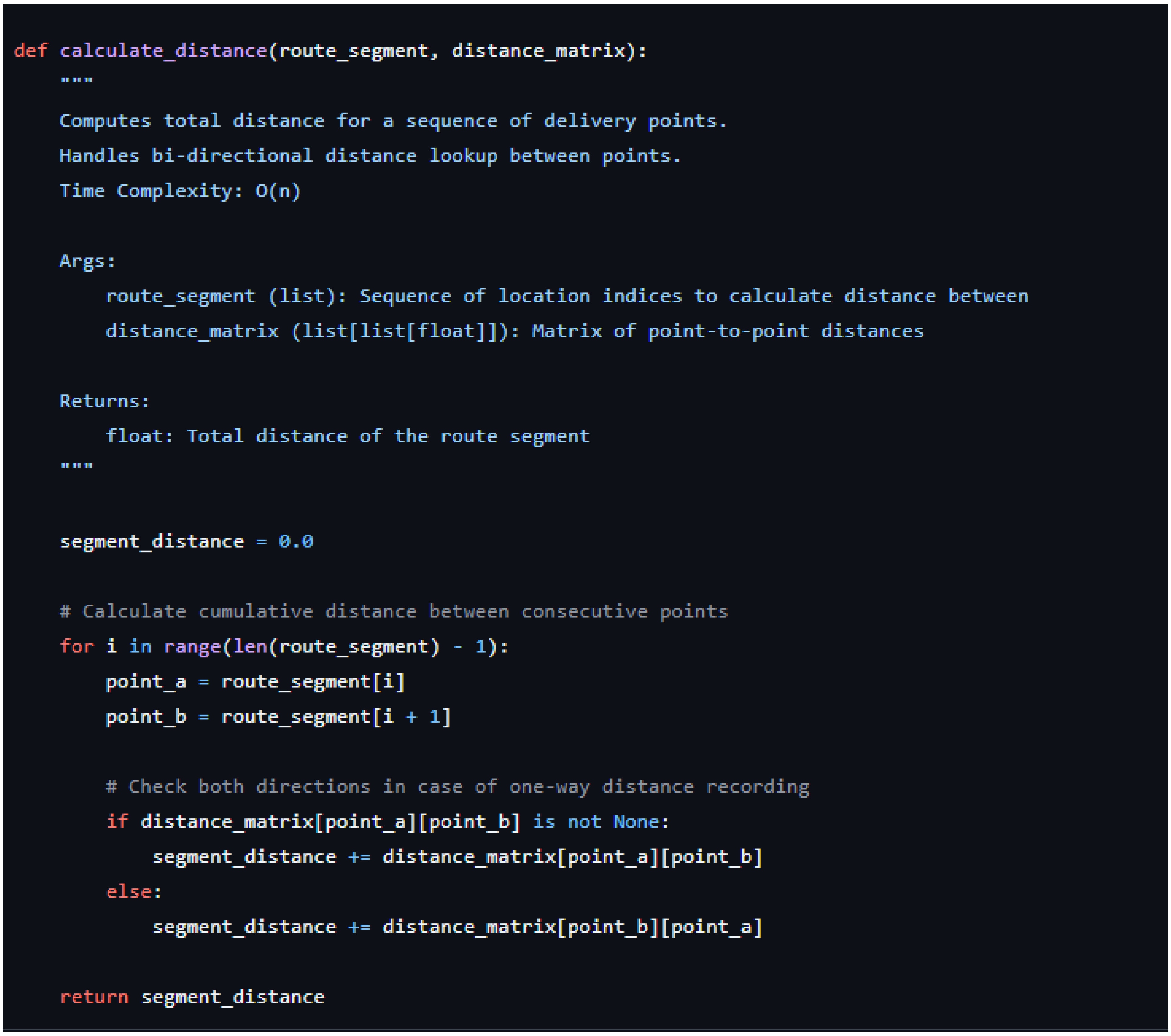


```
# Remove assigned parcels from processing queue
for parcel_id in vehicle_loads[1], vehicle_loads[2], vehicle_loads[3]:
    for pid in parcel_id:
        for parcel in remaining_parcels:
            if parcel.tracking_id == pid:
                processing_queue.remove(parcel)
        continue

remaining_parcels = processing_queue.copy()
zones1, zones2, zones3 = [], [], []

# Group parcels by delivery zone
for load in vehicle_loads[1]:
    for assigned in assigned_parcels:
        if assigned.tracking_id == load:
            zones1.append(assigned.destination)
            zones1.append(assigned.dest_zip)
for load in vehicle_loads[2]:
    for assigned in assigned_parcels:
        if assigned.tracking_id == load:
            zones2.append(assigned.destination)
            zones2.append(assigned.dest_zip)
for load in vehicle_loads[3]:
    for assigned in assigned_parcels:
        if assigned.tracking_id == load:
            zones3.append(assigned.destination)
            zones3.append(assigned.dest_zip)

# Assign early deadline parcels to matching zones
for parcel in remaining_parcels:
    if parcel.deadline != EOD:
        if ((parcel.destination in zones1 or parcel.dest_zip in zones1)
                and len(vehicle_loads[1]) < 16):
            vehicle_loads[1].append(parcel.tracking_id)
            assigned_parcels.append(parcel)
            processing_queue.remove(parcel)

remaining_parcels = processing_queue.copy()
```

3. Distance Calculations: The distance calculation and import functions demonstrate how the system manages location data and computes optimal routes using the distance matrix.

```python
import csv

def import_distances():
    """
    Loads and processes distance matrix from CSV data source.
    Constructs a 2D array containing all point-to-point distances.
    Time Complexity: O(n²)

    Returns:
        list[list[float]]: Matrix of distances between delivery points
    """

    with open('./data/distances.csv') as route_data:
        csv_parser = csv.reader(route_data)
        delivery_points = next(csv_parser)[2:]  # Skip first two columns of header row

        # Initialize empty distance matrix matching location count
        distance_matrix = [[None] * len(delivery_points) for _ in range(len(delivery_points))]

        # Populate matrix with distances from CSV
        # Sets None for missing/empty entries
        for row_idx, row in enumerate(csv_parser):
            for col_idx, distance in enumerate(row[2:]):  # Skip first two columns of each row
                distance_matrix[row_idx][col_idx] = float(distance) if distance else None

    return distance_matrix


def import_addresses():
    """
    Extracts and standardizes delivery location addresses from CSV.
    Formats each address to include only the primary street address.
    Time Complexity: O(n)

    Returns:
        list[str]: Clean list of delivery addresses
    """

    with open('./data/distances.csv') as route_data:
        csv_parser = csv.reader(route_data, delimiter=',')
        raw_addresses = next(csv_parser)[2:]  # Skip first two columns

        formatted_addresses = []
        # Process each address to extract main street line
        for addr in raw_addresses:
            main_address = addr.splitlines()[1].strip(', ')  # Get second line and trim
            formatted_addresses.append(main_address)

    return formatted_addresses
```

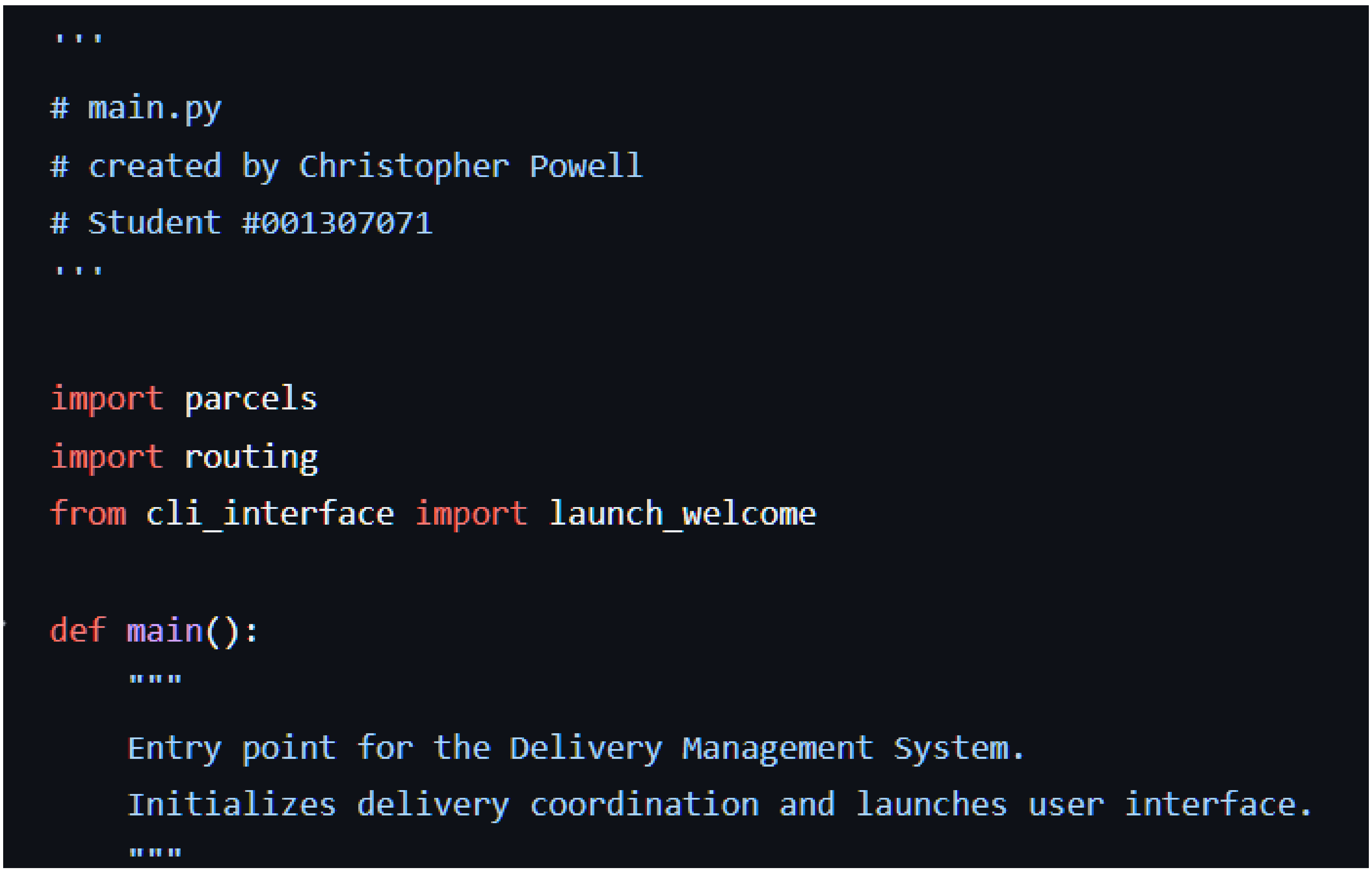
C. 3. CONTINUED



```python
def calculate_distance(route_segment, distance_matrix):
    """
    Computes total distance for a sequence of delivery points.
    Handles bi-directional distance lookup between points.
    Time Complexity: O(n)

    Args:
        route_segment (list): Sequence of location indices to calculate distance between
        distance_matrix (list[list[float]]): Matrix of point-to-point distances

    Returns:
        float: Total distance of the route segment
    """

    segment_distance = 0.0

    # Calculate cumulative distance between consecutive points
    for i in range(len(route_segment) - 1):
        point_a = route_segment[i]
        point_b = route_segment[i + 1]

        # Check both directions in case of one-way distance recording
        if distance_matrix[point_a][point_b] is not None:
            segment_distance += distance_matrix[point_a][point_b]
        else:
            segment_distance += distance_matrix[point_b][point_a]

    return segment_distance
```

C1. Identification Information

Main program entry point showing student identification and program initialization.

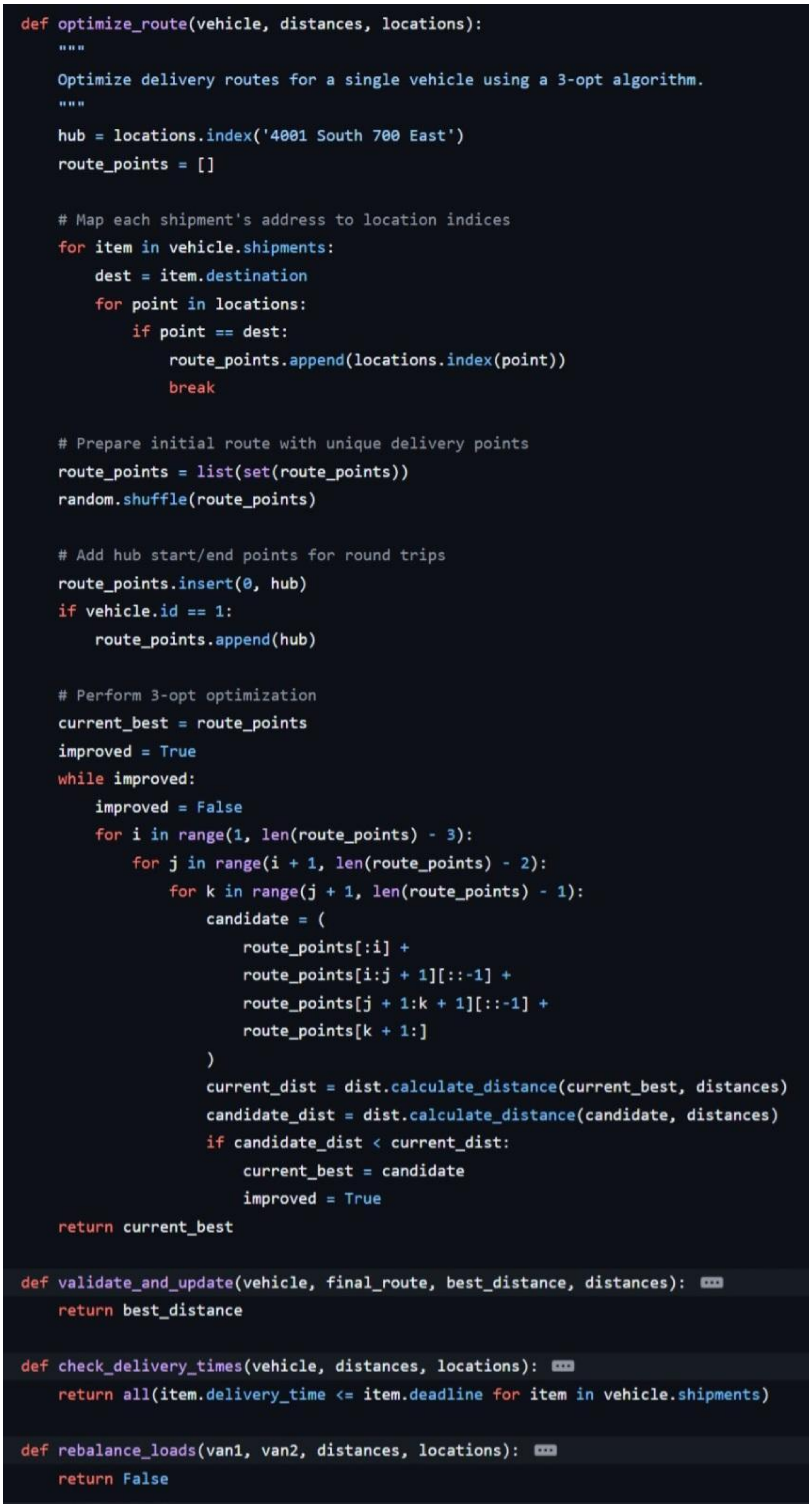


```
'''
# main.py
# created by Christopher Powell
# Student #001307071
'''


import parcels
import routing
from cli_interface import launch_welcome

def main():
    """
    Entry point for the Delivery Management System.
    Initializes delivery coordination and launches user interface.
    """
```

C2. Process and Flow Comments

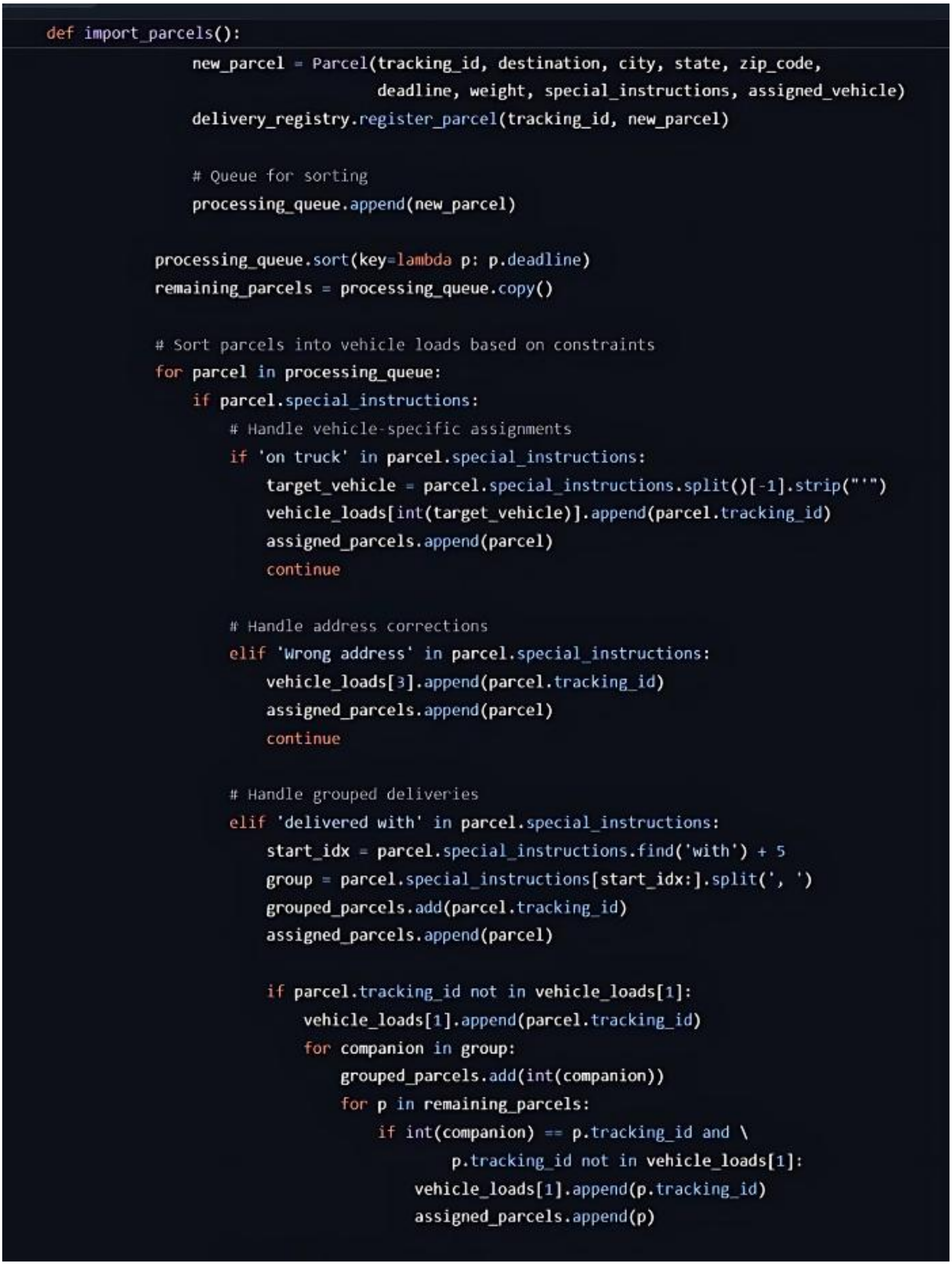
Detailed comments explain the routing optimization process, package sorting logic, and vehicle management system, providing clear documentation of program flow.

1. `routing.py` - The detailed comments in `optimize_route` and `coordinate_deliveries`

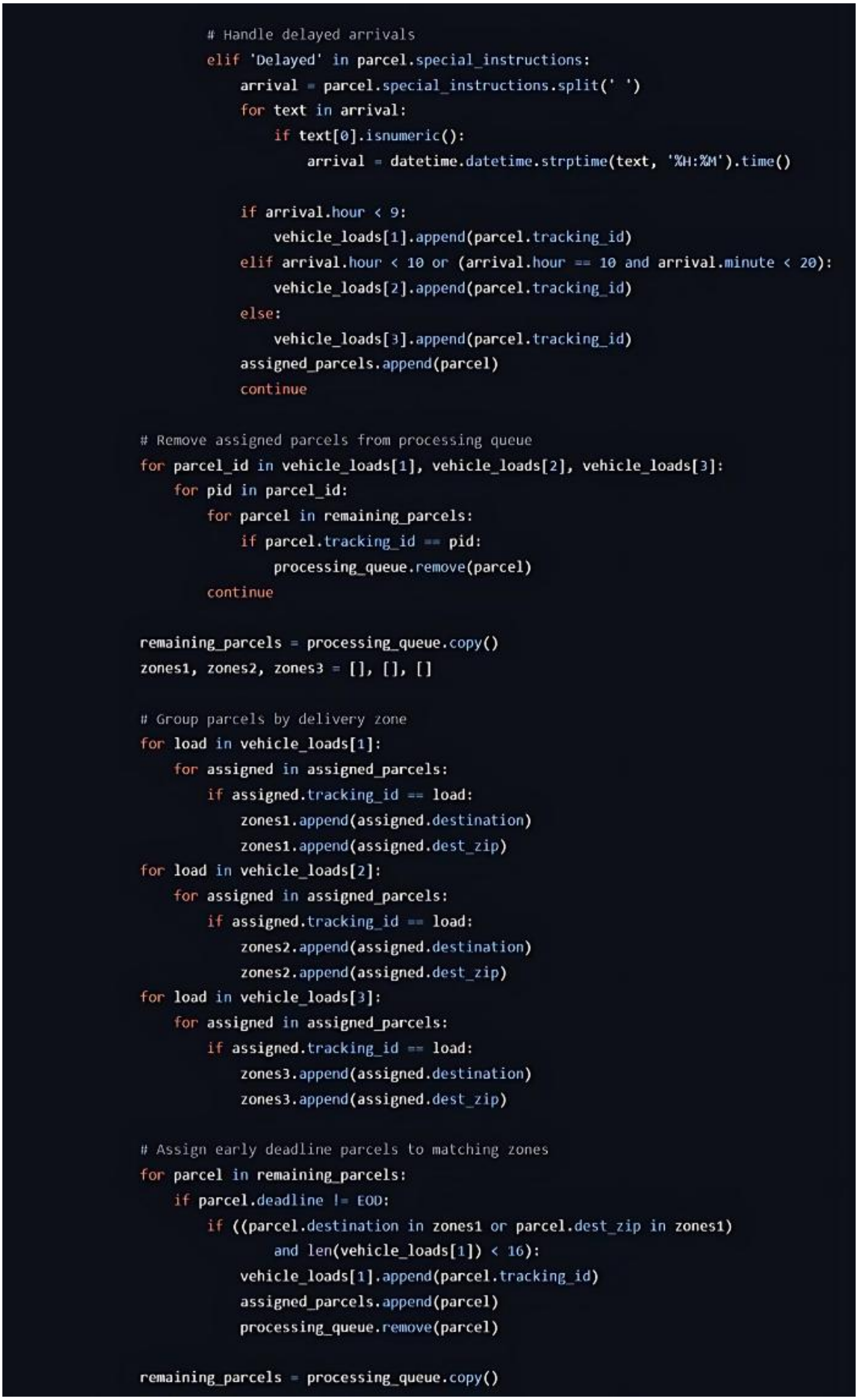


```python
def optimize_route(vehicle, distances, locations):
    """
    Optimize delivery routes for a single vehicle using a 3-opt algorithm.
    """
    hub = locations.index('4001 South 700 East')
    route_points = []

    # Map each shipment's address to location indices
    for item in vehicle.shipments:
        dest = item.destination
        for point in locations:
            if point == dest:
                route_points.append(locations.index(point))
                break

    # Prepare initial route with unique delivery points
    route_points = list(set(route_points))
    random.shuffle(route_points)

    # Add hub start/end points for round trips
    route_points.insert(0, hub)
    if vehicle.id == 1:
        route_points.append(hub)

    # Perform 3-opt optimization
    current_best = route_points
    improved = True
    while improved:
        improved = False
        for i in range(1, len(route_points) - 3):
            for j in range(i + 1, len(route_points) - 2):
                for k in range(j + 1, len(route_points) - 1):
                    candidate = (
                        route_points[:i] +
                        route_points[i:j + 1][::-1] +
                        route_points[j + 1:k + 1][::-1] +
                        route_points[k + 1:]
                    )
                    current_dist = dist.calculate_distance(current_best, distances)
                    candidate_dist = dist.calculate_distance(candidate, distances)
                    if candidate_dist < current_dist:
                        current_best = candidate
                        improved = True
    return current_best

def validate_and_update(vehicle, final_route, best_distance, distances): ...
    return best_distance

def check_delivery_times(vehicle, distances, locations): ...
    return all(item.delivery_time <= item.deadline for item in vehicle.shipments)

def rebalance_loads(van1, van2, distances, locations): ...
    return False
```

2. `parcels.py` - Comments in `import_parcels` showing the package sorting logic

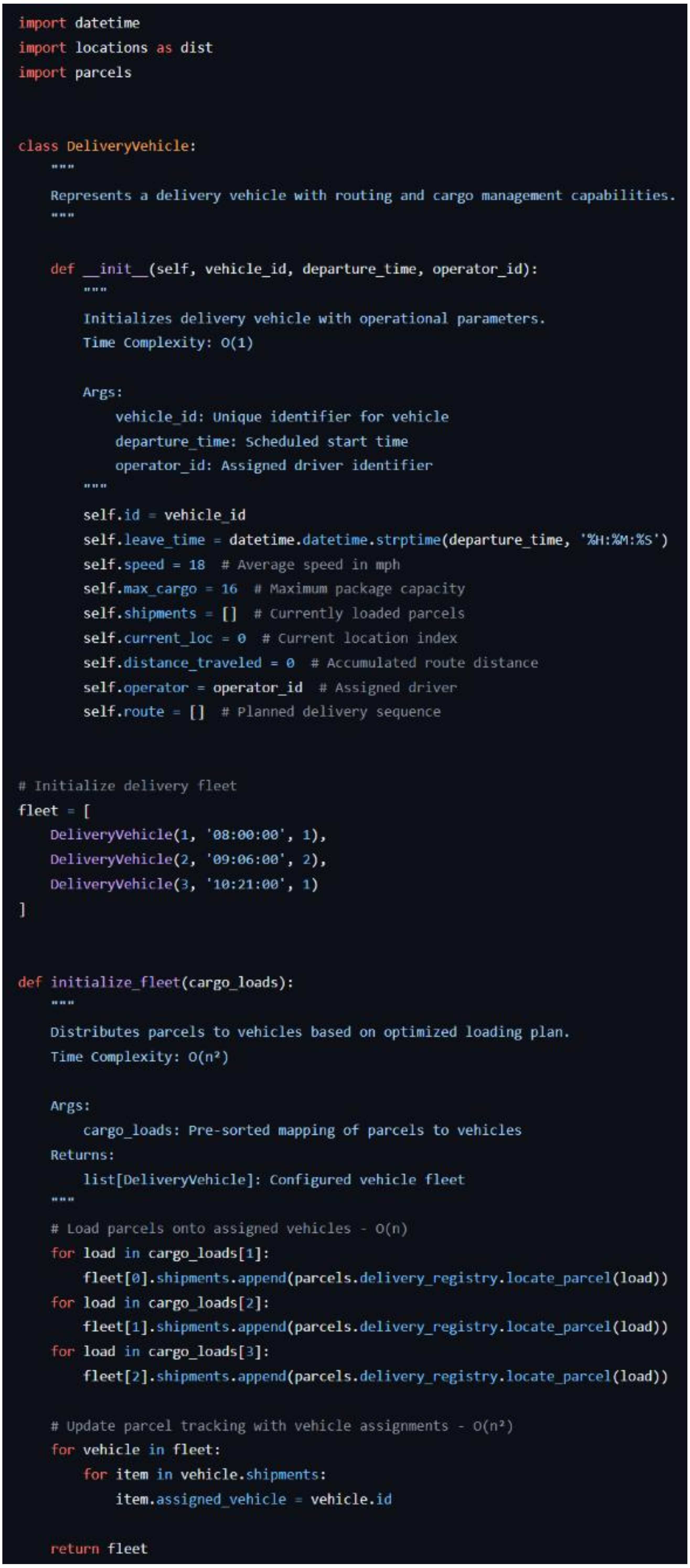


```
def import_parcels():
            new_parcel = Parcel(tracking_id, destination, city, state, zip_code,
                                deadline, weight, special_instructions, assigned_vehicle)
            delivery_registry.register_parcel(tracking_id, new_parcel)

            # Queue for sorting
            processing_queue.append(new_parcel)

        processing_queue.sort(key=lambda p: p.deadline)
        remaining_parcels = processing_queue.copy()

        # Sort parcels into vehicle loads based on constraints
        for parcel in processing_queue:
            if parcel.special_instructions:
                # Handle vehicle-specific assignments
                if 'on truck' in parcel.special_instructions:
                    target_vehicle = parcel.special_instructions.split()[-1].strip("'")
                    vehicle_loads[int(target_vehicle)].append(parcel.tracking_id)
                    assigned_parcels.append(parcel)
                    continue

                # Handle address corrections
                elif 'Wrong address' in parcel.special_instructions:
                    vehicle_loads[3].append(parcel.tracking_id)
                    assigned_parcels.append(parcel)
                    continue

                # Handle grouped deliveries
                elif 'delivered with' in parcel.special_instructions:
                    start_idx = parcel.special_instructions.find('with') + 5
                    group = parcel.special_instructions[start_idx:].split(', ')
                    grouped_parcels.add(parcel.tracking_id)
                    assigned_parcels.append(parcel)

                    if parcel.tracking_id not in vehicle_loads[1]:
                        vehicle_loads[1].append(parcel.tracking_id)
                        for companion in group:
                            grouped_parcels.add(int(companion))
                            for p in remaining_parcels:
                                if int(companion) == p.tracking_id and \
                                        p.tracking_id not in vehicle_loads[1]:
                                    vehicle_loads[1].append(p.tracking_id)
                                    assigned_parcels.append(p)
```



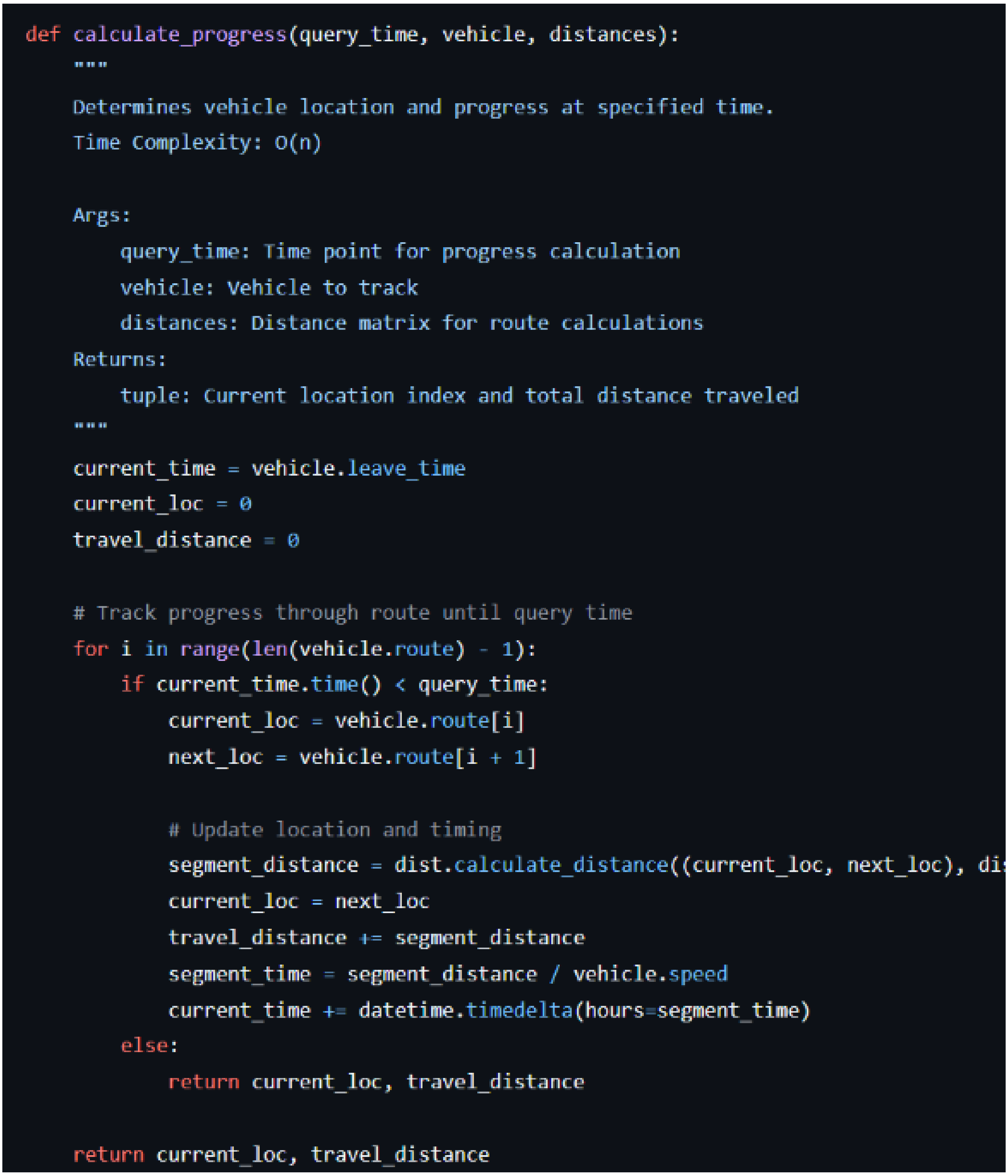
```
        # Handle delayed arrivals
        elif 'Delayed' in parcel.special_instructions:
            arrival = parcel.special_instructions.split(' ')
            for text in arrival:
                if text[0].isnumeric():
                    arrival = datetime.datetime.strptime(text, '%H:%M').time()

            if arrival.hour < 9:
                vehicle_loads[1].append(parcel.tracking_id)
            elif arrival.hour < 10 or (arrival.hour == 10 and arrival.minute < 20):
                vehicle_loads[2].append(parcel.tracking_id)
            else:
                vehicle_loads[3].append(parcel.tracking_id)
            assigned_parcels.append(parcel)
            continue

# Remove assigned parcels from processing queue
for parcel_id in vehicle_loads[1], vehicle_loads[2], vehicle_loads[3]:
    for pid in parcel_id:
        for parcel in remaining_parcels:
            if parcel.tracking_id == pid:
                processing_queue.remove(parcel)
        continue

remaining_parcels = processing_queue.copy()
zones1, zones2, zones3 = [], [], []

# Group parcels by delivery zone
for load in vehicle_loads[1]:
    for assigned in assigned_parcels:
        if assigned.tracking_id == load:
            zones1.append(assigned.destination)
            zones1.append(assigned.dest_zip)
for load in vehicle_loads[2]:
    for assigned in assigned_parcels:
        if assigned.tracking_id == load:
            zones2.append(assigned.destination)
            zones2.append(assigned.dest_zip)
for load in vehicle_loads[3]:
    for assigned in assigned_parcels:
        if assigned.tracking_id == load:
            zones3.append(assigned.destination)
            zones3.append(assigned.dest_zip)

# Assign early deadline parcels to matching zones
for parcel in remaining_parcels:
    if parcel.deadline != EOD:
        if ((parcel.destination in zones1 or parcel.dest_zip in zones1)
                and len(vehicle_loads[1]) < 16):
            vehicle_loads[1].append(parcel.tracking_id)
            assigned_parcels.append(parcel)
            processing_queue.remove(parcel)

remaining_parcels = processing_queue.copy()
```

3. van.py – Comments explaining the vehicle initialization and routing



```python
import datetime
import locations as dist
import parcels


class DeliveryVehicle:
    """
    Represents a delivery vehicle with routing and cargo management capabilities.
    """

    def __init__(self, vehicle_id, departure_time, operator_id):
        """
        Initializes delivery vehicle with operational parameters.
        Time Complexity: O(1)

        Args:
            vehicle_id: Unique identifier for vehicle
            departure_time: Scheduled start time
            operator_id: Assigned driver identifier
        """
        self.id = vehicle_id
        self.leave_time = datetime.datetime.strptime(departure_time, '%H:%M:%S')
        self.speed = 18  # Average speed in mph
        self.max_cargo = 16  # Maximum package capacity
        self.shipments = []  # Currently loaded parcels
        self.current_loc = 0  # Current location index
        self.distance_traveled = 0  # Accumulated route distance
        self.operator = operator_id  # Assigned driver
        self.route = []  # Planned delivery sequence


# Initialize delivery fleet
fleet = [
    DeliveryVehicle(1, '08:00:00', 1),
    DeliveryVehicle(2, '09:06:00', 2),
    DeliveryVehicle(3, '10:21:00', 1)
]


def initialize_fleet(cargo_loads):
    """
    Distributes parcels to vehicles based on optimized loading plan.
    Time Complexity: O(n²)

    Args:
        cargo_loads: Pre-sorted mapping of parcels to vehicles
    Returns:
        list[DeliveryVehicle]: Configured vehicle fleet
    """
    # Load parcels onto assigned vehicles - O(n)
    for load in cargo_loads[1]:
        fleet[0].shipments.append(parcels.delivery_registry.locate_parcel(load))
    for load in cargo_loads[2]:
        fleet[1].shipments.append(parcels.delivery_registry.locate_parcel(load))
    for load in cargo_loads[3]:
        fleet[2].shipments.append(parcels.delivery_registry.locate_parcel(load))

    # Update parcel tracking with vehicle assignments - O(n²)
    for vehicle in fleet:
        for item in vehicle.shipments:
            item.assigned_vehicle = vehicle.id

    return fleet
```

C2. 3. CONTINUED

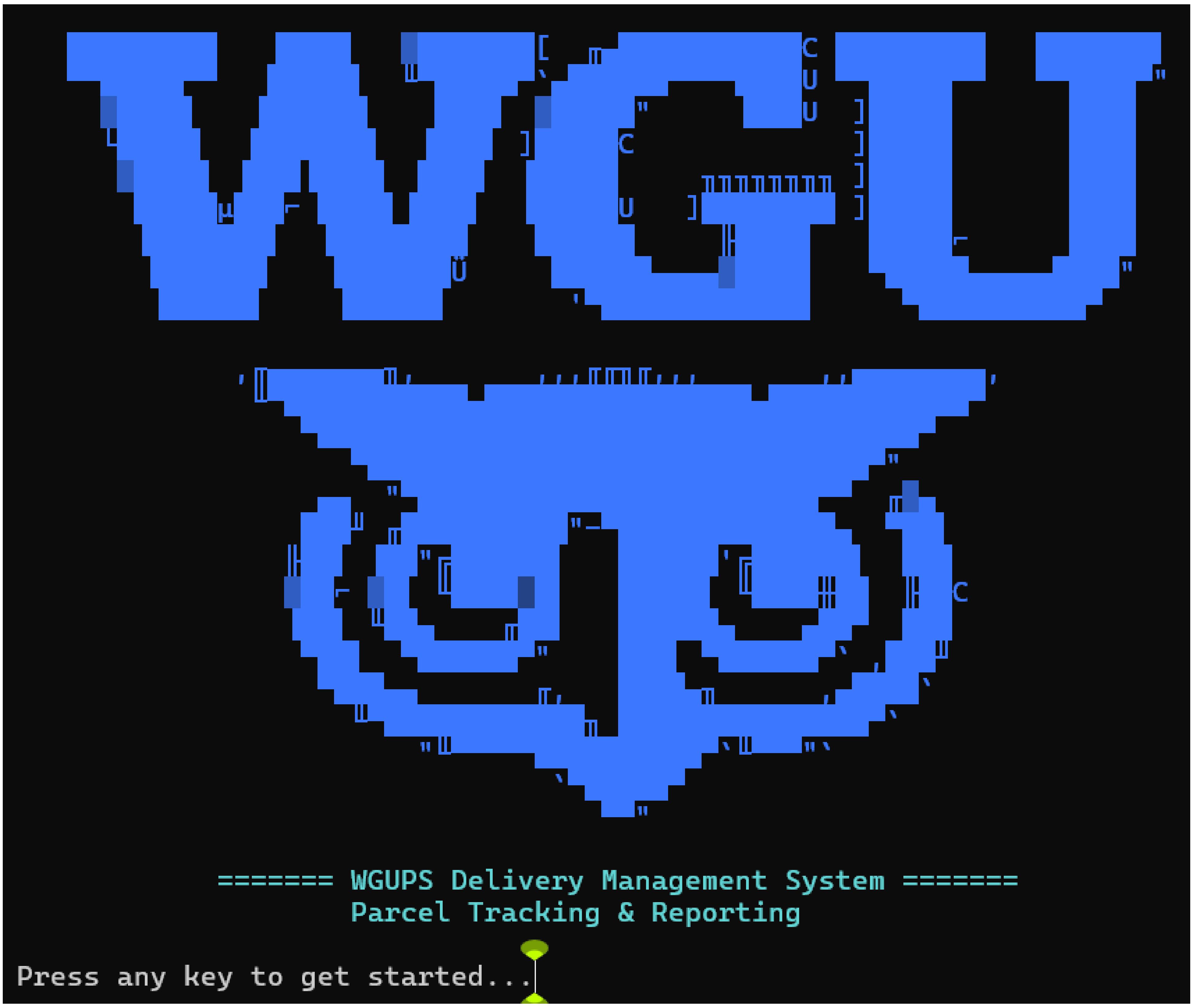


```python
def calculate_progress(query_time, vehicle, distances):
    """
    Determines vehicle location and progress at specified time.
    Time Complexity: O(n)

    Args:
        query_time: Time point for progress calculation
        vehicle: Vehicle to track
        distances: Distance matrix for route calculations
    Returns:
        tuple: Current location index and total distance traveled
    """
    current_time = vehicle.leave_time
    current_loc = 0
    travel_distance = 0

    # Track progress through route until query time
    for i in range(len(vehicle.route) - 1):
        if current_time.time() < query_time:
            current_loc = vehicle.route[i]
            next_loc = vehicle.route[i + 1]

            # Update location and timing
            segment_distance = dist.calculate_distance((current_loc, next_loc), di
            current_loc = next_loc
            travel_distance += segment_distance
            segment_time = segment_distance / vehicle.speed
            current_time += datetime.timedelta(hours=segment_time)
        else:
            return current_loc, travel_distance

    return current_loc, travel_distance
```

D. Interface

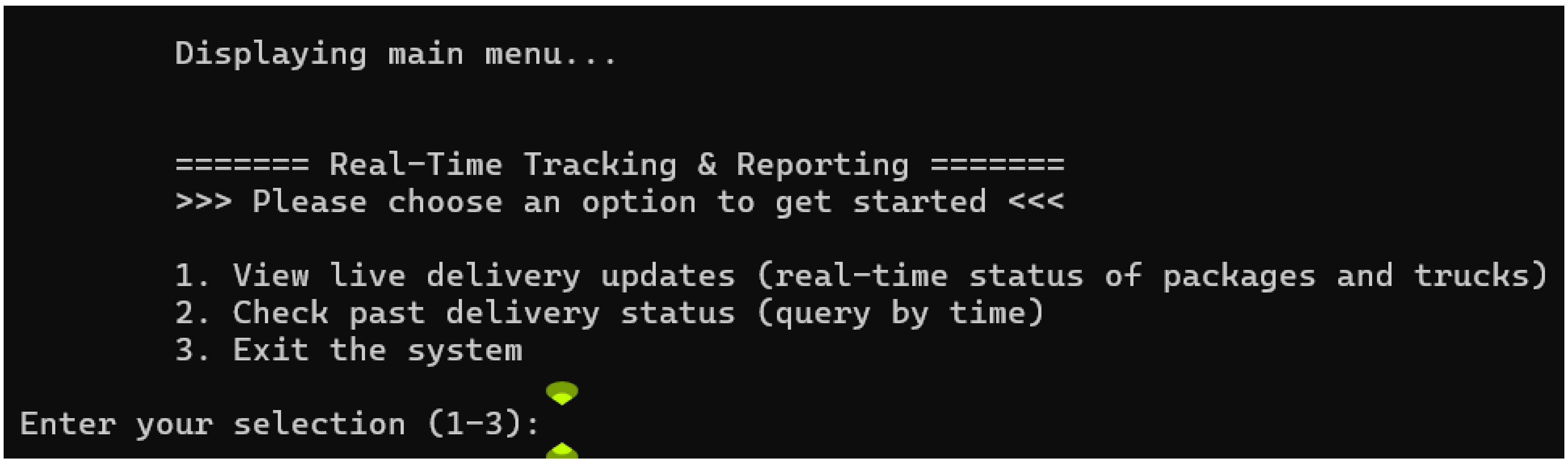
Screenshots from cli_interface.py

1. Welcome Screen: ASCII WGU and Sage art welcome screen provides a professional and user-friendly program.



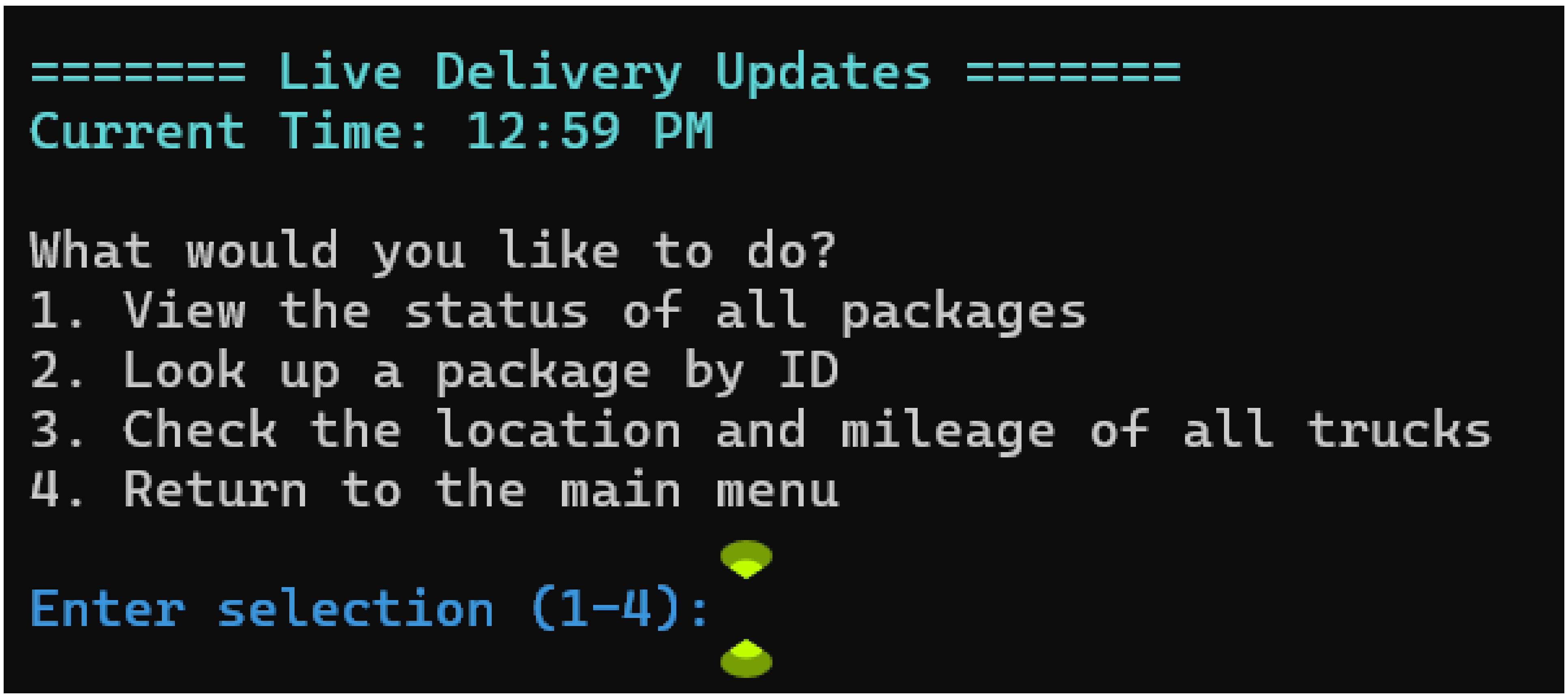
```
======= WGUPS Delivery Management System =======
        Parcel Tracking & Reporting

Press any key to get started...
```

2. Main Menu: Interactive main menu showing the primary options for package tracking and delivery management.

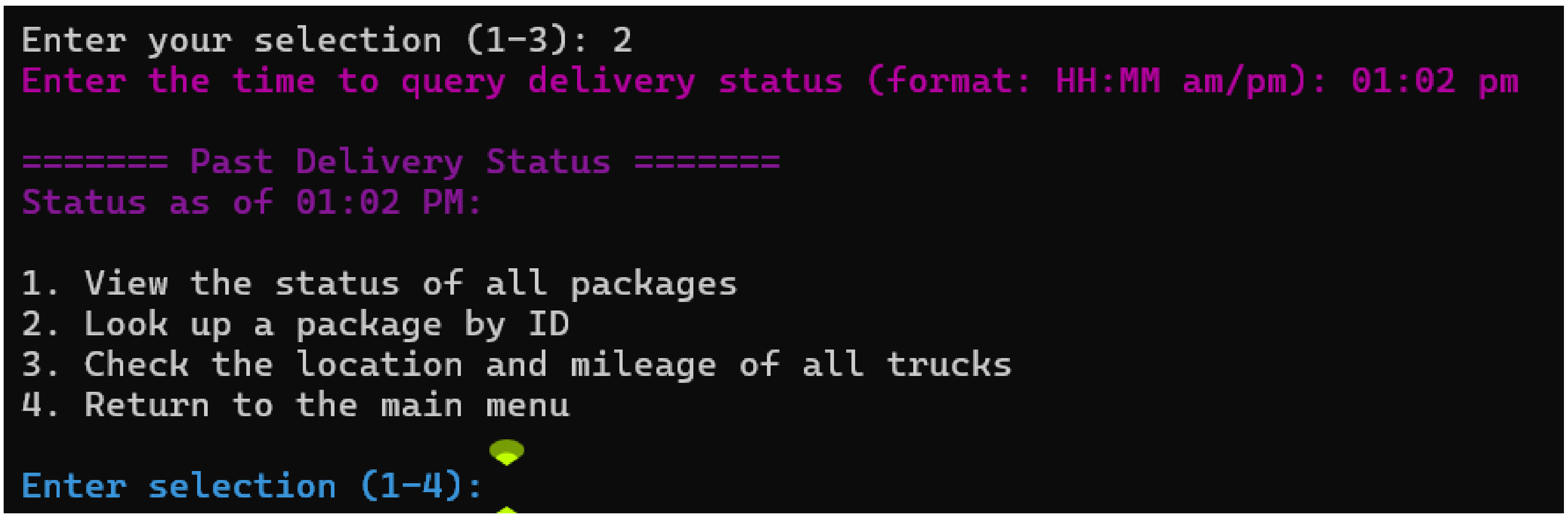


```
        Displaying main menu...


        ======= Real-Time Tracking & Reporting =======
        >>> Please choose an option to get started <<<

        1. View live delivery updates (real-time status of packages and trucks)
        2. Check past delivery status (query by time)
        3. Exit the system

Enter your selection (1-3):
```

3. The status menus in `live_updates_menu` and `past_delivery_menu`

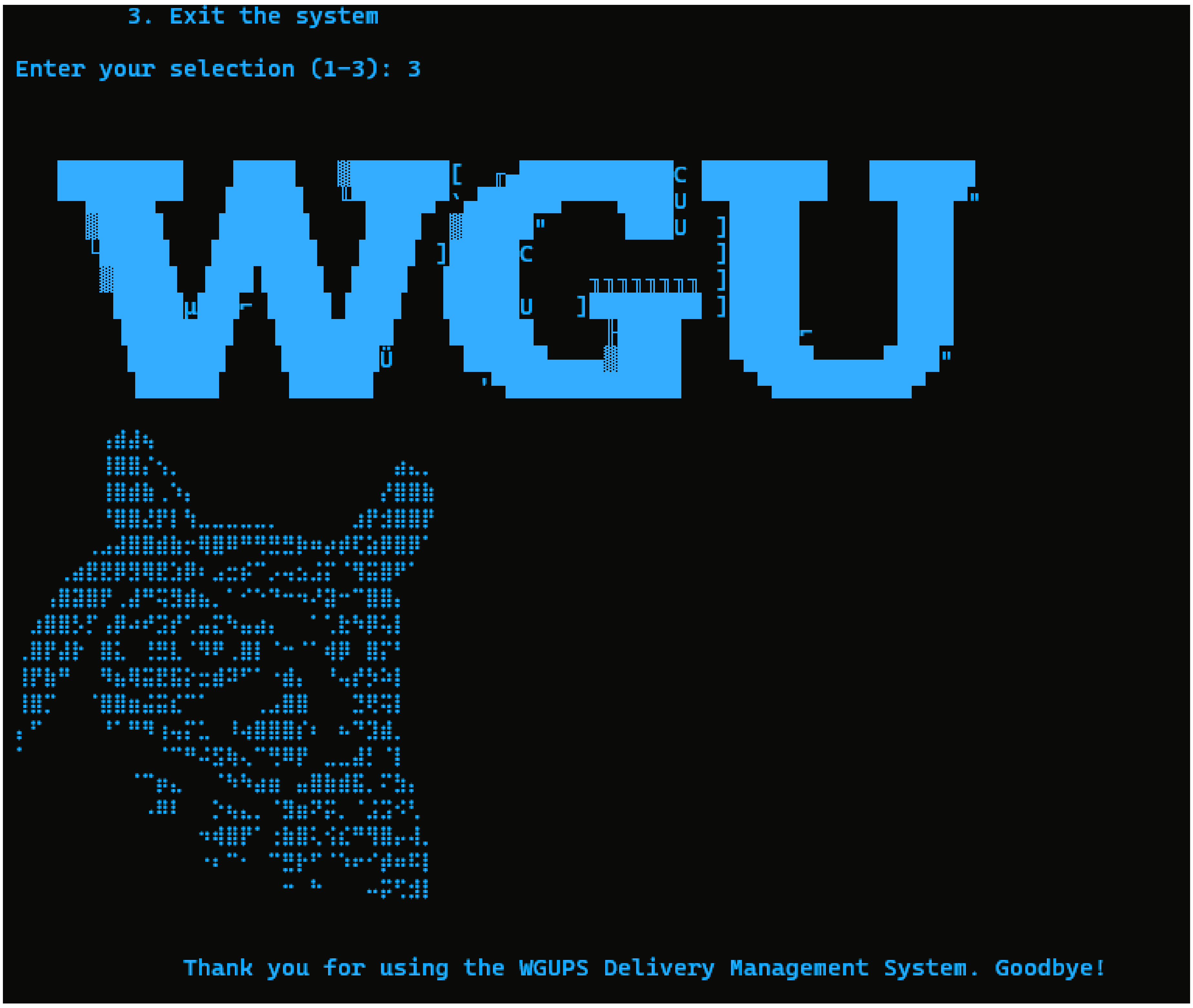


```
======== Live Delivery Updates ========
Current Time: 12:59 PM

What would you like to do?
1. View the status of all packages
2. Look up a package by ID
3. Check the location and mileage of all trucks
4. Return to the main menu

Enter selection (1-4):
```



```
Enter your selection (1-3): 2
Enter the time to query delivery status (format: HH:MM am/pm): 01:02 pm

======= Past Delivery Status =======
Status as of 01:02 PM:

1. View the status of all packages
2. Look up a package by ID
3. Check the location and mileage of all trucks
4. Return to the main menu

Enter selection (1-4):
```

4. Exit Menu: Shows ASCII WGU and Sage art

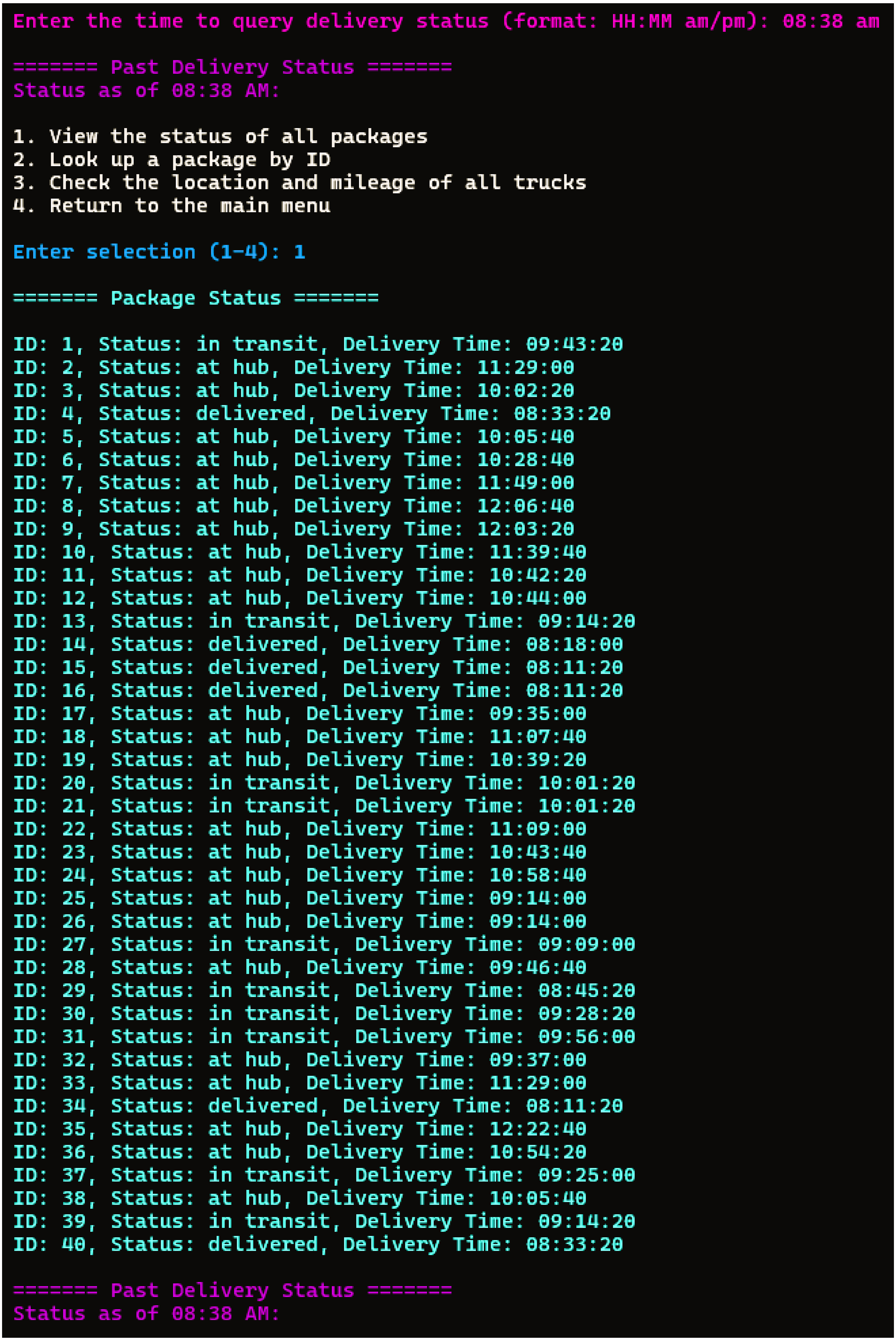


```
        3. Exit the system

Enter your selection (1-3): 3
```

```
            Thank you for using the WGUPS Delivery Management System. Goodbye!
```

D1. First Status Check

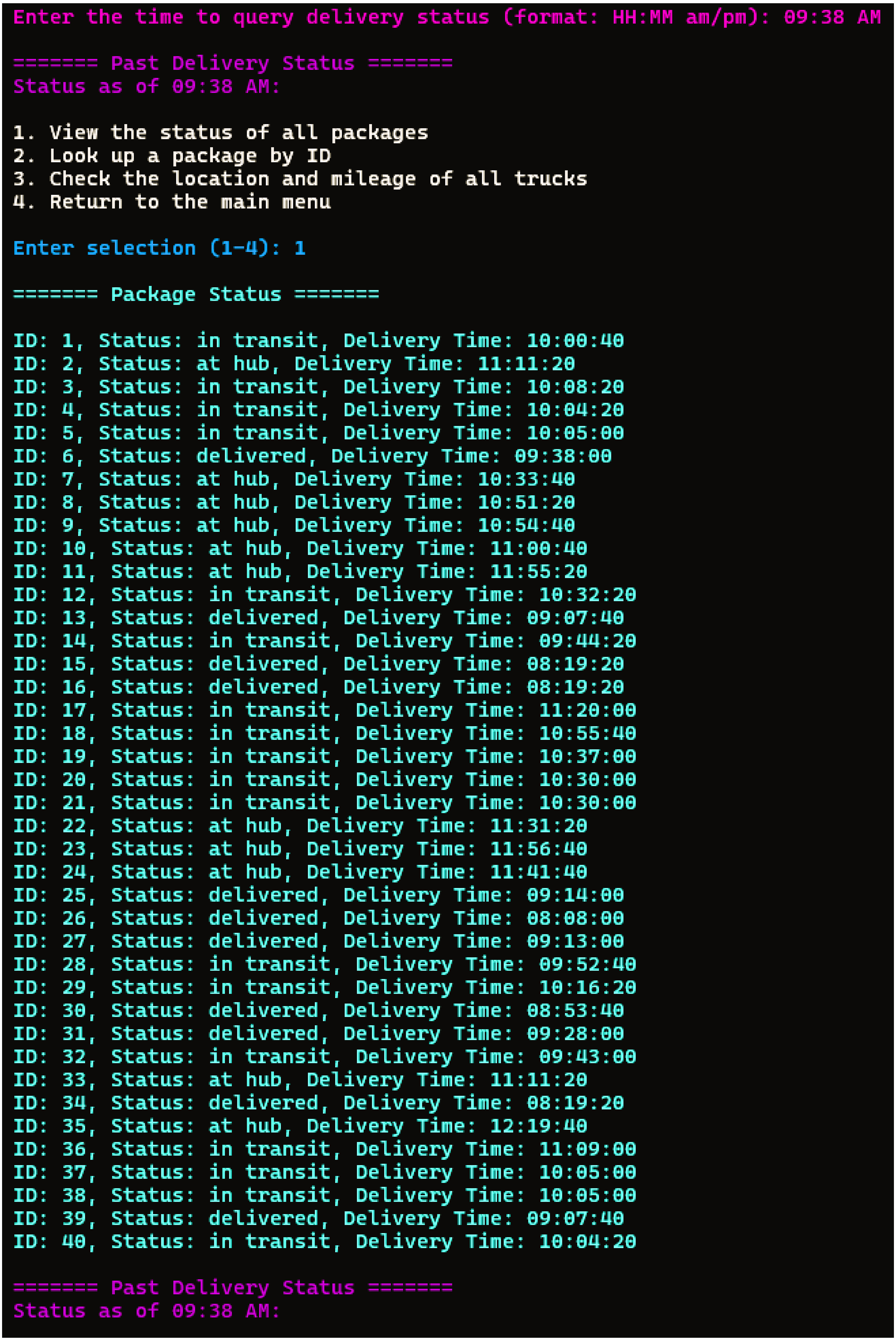
First Status Check status of all packages loaded onto each truck at a time between 8:35 a.m. and 9:25 a.m.



```
Enter the time to query delivery status (format: HH:MM am/pm): 08:38 am

======= Past Delivery Status =======
Status as of 08:38 AM:

1. View the status of all packages
2. Look up a package by ID
3. Check the location and mileage of all trucks
4. Return to the main menu

Enter selection (1-4): 1

======= Package Status =======

ID: 1, Status: in transit, Delivery Time: 09:43:20
ID: 2, Status: at hub, Delivery Time: 11:29:00
ID: 3, Status: at hub, Delivery Time: 10:02:20
ID: 4, Status: delivered, Delivery Time: 08:33:20
ID: 5, Status: at hub, Delivery Time: 10:05:40
ID: 6, Status: at hub, Delivery Time: 10:28:40
ID: 7, Status: at hub, Delivery Time: 11:49:00
ID: 8, Status: at hub, Delivery Time: 12:06:40
ID: 9, Status: at hub, Delivery Time: 12:03:20
ID: 10, Status: at hub, Delivery Time: 11:39:40
ID: 11, Status: at hub, Delivery Time: 10:42:20
ID: 12, Status: at hub, Delivery Time: 10:44:00
ID: 13, Status: in transit, Delivery Time: 09:14:20
ID: 14, Status: delivered, Delivery Time: 08:18:00
ID: 15, Status: delivered, Delivery Time: 08:11:20
ID: 16, Status: delivered, Delivery Time: 08:11:20
ID: 17, Status: at hub, Delivery Time: 09:35:00
ID: 18, Status: at hub, Delivery Time: 11:07:40
ID: 19, Status: at hub, Delivery Time: 10:39:20
ID: 20, Status: in transit, Delivery Time: 10:01:20
ID: 21, Status: in transit, Delivery Time: 10:01:20
ID: 22, Status: at hub, Delivery Time: 11:09:00
ID: 23, Status: at hub, Delivery Time: 10:43:40
ID: 24, Status: at hub, Delivery Time: 10:58:40
ID: 25, Status: at hub, Delivery Time: 09:14:00
ID: 26, Status: at hub, Delivery Time: 09:14:00
ID: 27, Status: in transit, Delivery Time: 09:09:00
ID: 28, Status: at hub, Delivery Time: 09:46:40
ID: 29, Status: in transit, Delivery Time: 08:45:20
ID: 30, Status: in transit, Delivery Time: 09:28:20
ID: 31, Status: in transit, Delivery Time: 09:56:00
ID: 32, Status: at hub, Delivery Time: 09:37:00
ID: 33, Status: at hub, Delivery Time: 11:29:00
ID: 34, Status: delivered, Delivery Time: 08:11:20
ID: 35, Status: at hub, Delivery Time: 12:22:40
ID: 36, Status: at hub, Delivery Time: 10:54:20
ID: 37, Status: in transit, Delivery Time: 09:25:00
ID: 38, Status: at hub, Delivery Time: 10:05:40
ID: 39, Status: in transit, Delivery Time: 09:14:20
ID: 40, Status: delivered, Delivery Time: 08:33:20

======= Past Delivery Status =======
Status as of 08:38 AM:
```

D2. Second Status Check

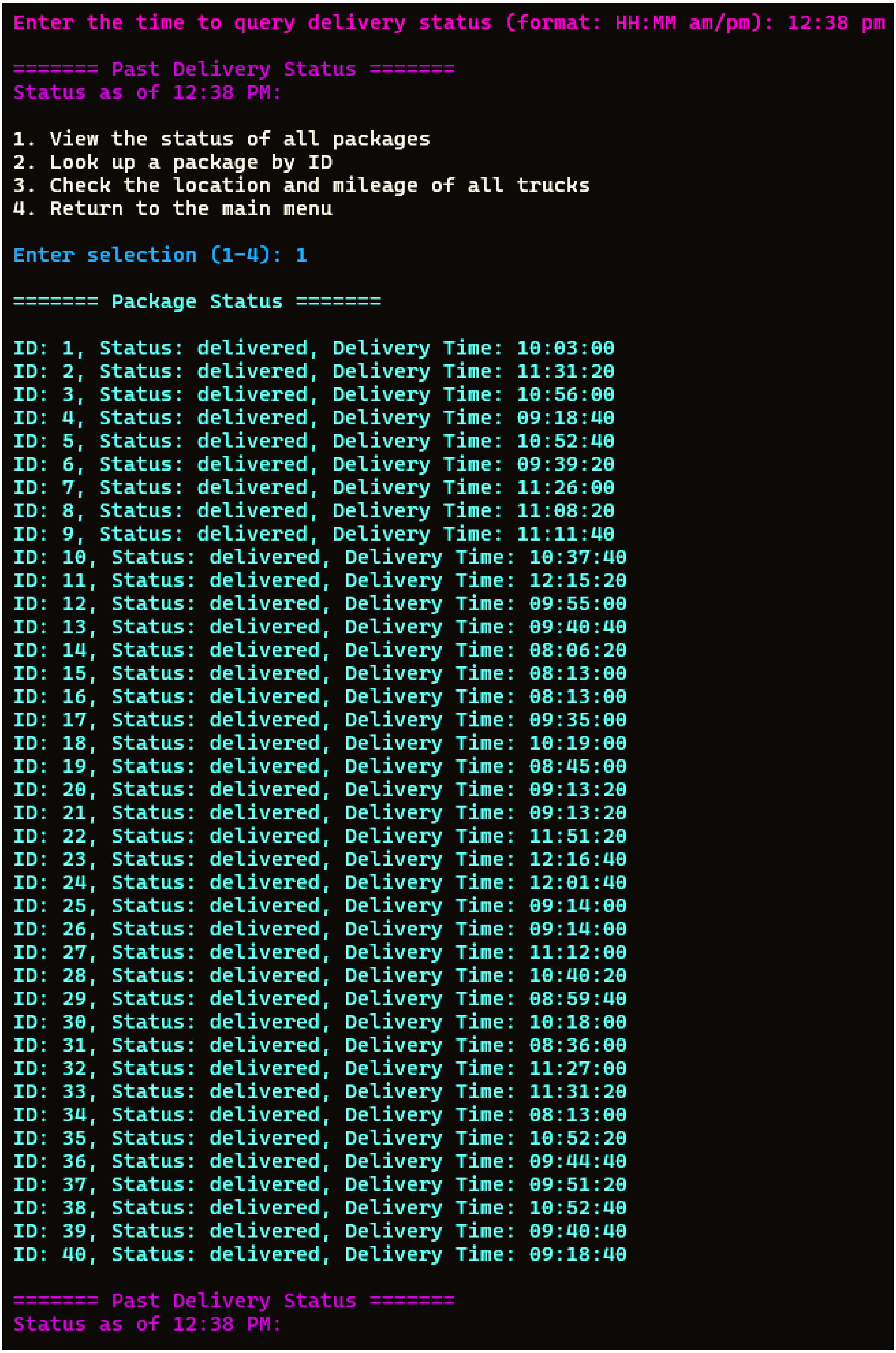
Second Status Check status of all packages loaded onto each truck at a time between 9:35 a.m. &10:25 a.m.



```
Enter the time to query delivery status (format: HH:MM am/pm): 09:38 AM

======= Past Delivery Status =======
Status as of 09:38 AM:

1. View the status of all packages
2. Look up a package by ID
3. Check the location and mileage of all trucks
4. Return to the main menu

Enter selection (1-4): 1

======= Package Status =======

ID: 1, Status: in transit, Delivery Time: 10:00:40
ID: 2, Status: at hub, Delivery Time: 11:11:20
ID: 3, Status: in transit, Delivery Time: 10:08:20
ID: 4, Status: in transit, Delivery Time: 10:04:20
ID: 5, Status: in transit, Delivery Time: 10:05:00
ID: 6, Status: delivered, Delivery Time: 09:38:00
ID: 7, Status: at hub, Delivery Time: 10:33:40
ID: 8, Status: at hub, Delivery Time: 10:51:20
ID: 9, Status: at hub, Delivery Time: 10:54:40
ID: 10, Status: at hub, Delivery Time: 11:00:40
ID: 11, Status: at hub, Delivery Time: 11:55:20
ID: 12, Status: in transit, Delivery Time: 10:32:20
ID: 13, Status: delivered, Delivery Time: 09:07:40
ID: 14, Status: in transit, Delivery Time: 09:44:20
ID: 15, Status: delivered, Delivery Time: 08:19:20
ID: 16, Status: delivered, Delivery Time: 08:19:20
ID: 17, Status: in transit, Delivery Time: 11:20:00
ID: 18, Status: in transit, Delivery Time: 10:55:40
ID: 19, Status: in transit, Delivery Time: 10:37:00
ID: 20, Status: in transit, Delivery Time: 10:30:00
ID: 21, Status: in transit, Delivery Time: 10:30:00
ID: 22, Status: at hub, Delivery Time: 11:31:20
ID: 23, Status: at hub, Delivery Time: 11:56:40
ID: 24, Status: at hub, Delivery Time: 11:41:40
ID: 25, Status: delivered, Delivery Time: 09:14:00
ID: 26, Status: delivered, Delivery Time: 08:08:00
ID: 27, Status: delivered, Delivery Time: 09:13:00
ID: 28, Status: in transit, Delivery Time: 09:52:40
ID: 29, Status: in transit, Delivery Time: 10:16:20
ID: 30, Status: delivered, Delivery Time: 08:53:40
ID: 31, Status: delivered, Delivery Time: 09:28:00
ID: 32, Status: in transit, Delivery Time: 09:43:00
ID: 33, Status: at hub, Delivery Time: 11:11:20
ID: 34, Status: delivered, Delivery Time: 08:19:20
ID: 35, Status: at hub, Delivery Time: 12:19:40
ID: 36, Status: in transit, Delivery Time: 11:09:00
ID: 37, Status: in transit, Delivery Time: 10:05:00
ID: 38, Status: in transit, Delivery Time: 10:05:00
ID: 39, Status: delivered, Delivery Time: 09:07:40
ID: 40, Status: in transit, Delivery Time: 10:04:20

======= Past Delivery Status =======
Status as of 09:38 AM:
```

D3. Third Status Check

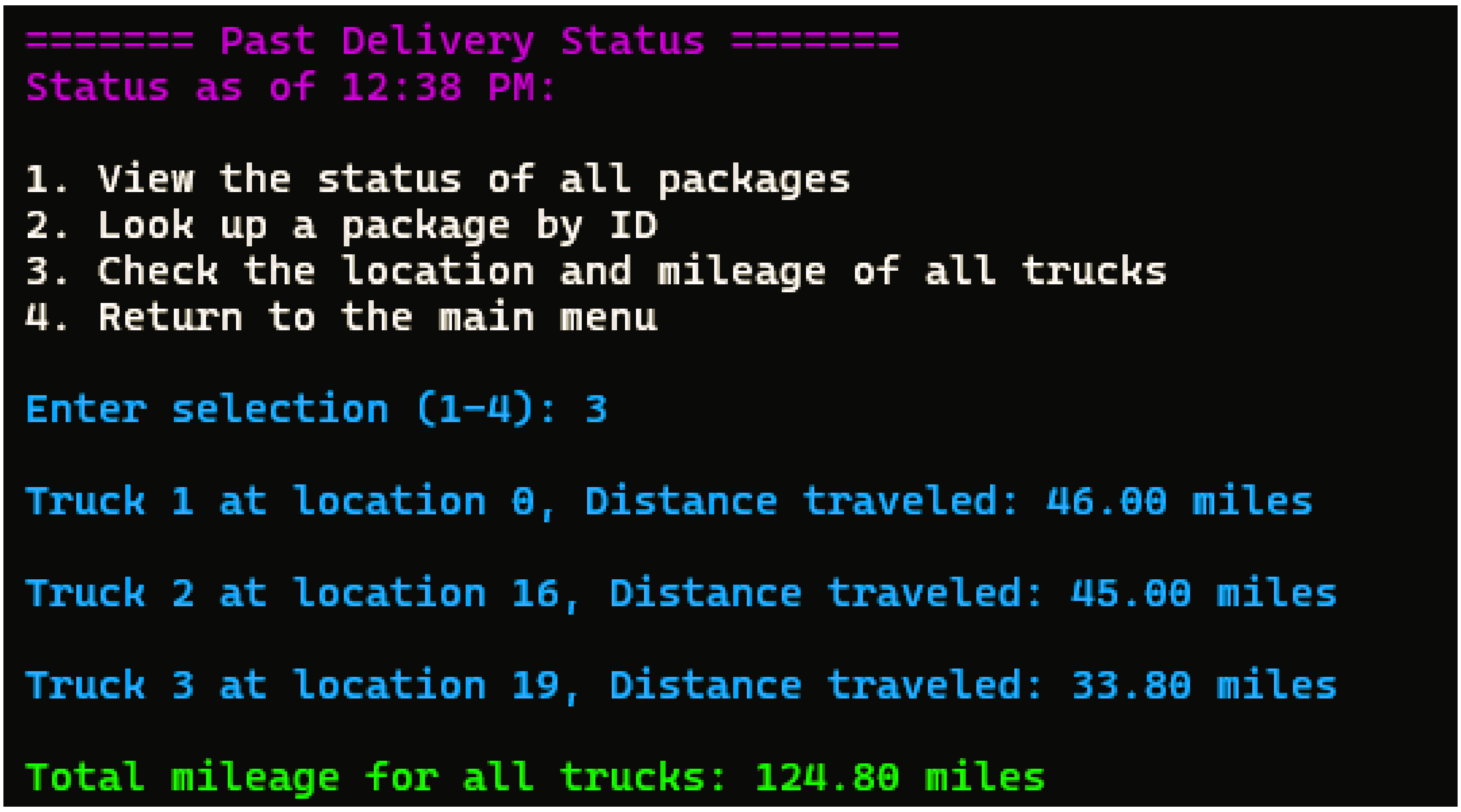
Third Status Check status of all packages loaded onto each truck at a time between 12:03 p.m. and 1:12 p.m.



```
Enter the time to query delivery status (format: HH:MM am/pm): 12:38 pm

======= Past Delivery Status =======
Status as of 12:38 PM:

1. View the status of all packages
2. Look up a package by ID
3. Check the location and mileage of all trucks
4. Return to the main menu

Enter selection (1-4): 1

======= Package Status =======

ID: 1, Status: delivered, Delivery Time: 10:03:00
ID: 2, Status: delivered, Delivery Time: 11:31:20
ID: 3, Status: delivered, Delivery Time: 10:56:00
ID: 4, Status: delivered, Delivery Time: 09:18:40
ID: 5, Status: delivered, Delivery Time: 10:52:40
ID: 6, Status: delivered, Delivery Time: 09:39:20
ID: 7, Status: delivered, Delivery Time: 11:26:00
ID: 8, Status: delivered, Delivery Time: 11:08:20
ID: 9, Status: delivered, Delivery Time: 11:11:40
ID: 10, Status: delivered, Delivery Time: 10:37:40
ID: 11, Status: delivered, Delivery Time: 12:15:20
ID: 12, Status: delivered, Delivery Time: 09:55:00
ID: 13, Status: delivered, Delivery Time: 09:40:40
ID: 14, Status: delivered, Delivery Time: 08:06:20
ID: 15, Status: delivered, Delivery Time: 08:13:00
ID: 16, Status: delivered, Delivery Time: 08:13:00
ID: 17, Status: delivered, Delivery Time: 09:35:00
ID: 18, Status: delivered, Delivery Time: 10:19:00
ID: 19, Status: delivered, Delivery Time: 08:45:00
ID: 20, Status: delivered, Delivery Time: 09:13:20
ID: 21, Status: delivered, Delivery Time: 09:13:20
ID: 22, Status: delivered, Delivery Time: 11:51:20
ID: 23, Status: delivered, Delivery Time: 12:16:40
ID: 24, Status: delivered, Delivery Time: 12:01:40
ID: 25, Status: delivered, Delivery Time: 09:14:00
ID: 26, Status: delivered, Delivery Time: 09:14:00
ID: 27, Status: delivered, Delivery Time: 11:12:00
ID: 28, Status: delivered, Delivery Time: 10:40:20
ID: 29, Status: delivered, Delivery Time: 08:59:40
ID: 30, Status: delivered, Delivery Time: 10:18:00
ID: 31, Status: delivered, Delivery Time: 08:36:00
ID: 32, Status: delivered, Delivery Time: 11:27:00
ID: 33, Status: delivered, Delivery Time: 11:31:20
ID: 34, Status: delivered, Delivery Time: 08:13:00
ID: 35, Status: delivered, Delivery Time: 10:52:20
ID: 36, Status: delivered, Delivery Time: 09:44:40
ID: 37, Status: delivered, Delivery Time: 09:51:20
ID: 38, Status: delivered, Delivery Time: 10:52:40
ID: 39, Status: delivered, Delivery Time: 09:40:40
ID: 40, Status: delivered, Delivery Time: 09:18:40

======= Past Delivery Status =======
Status as of 12:38 PM:
```

E. SCREENSHOT of Code Execution

Successful program execution demonstrating proper initialization, package loading, and interface functionality



```
======= Past Delivery Status =======
Status as of 12:38 PM:

1. View the status of all packages
2. Look up a package by ID
3. Check the location and mileage of all trucks
4. Return to the main menu

Enter selection (1-4): 3

Truck 1 at location 0, Distance traveled: 46.00 miles

Truck 2 at location 16, Distance traveled: 45.00 miles

Truck 3 at location 19, Distance traveled: 33.80 miles

Total mileage for all trucks: 124.80 miles
```

F1. Strengths of the Chosen Algorithm

The solution implements a hybrid approach combining 3-opt optimization with nearest neighbor initialization. Two key strengths are:

1. Dynamic Adaptation to Time Constraints
    - Efficiently handles time windows and delivery deadlines
    - Adjusts routes based on delayed package arrivals
    - Maintains package grouping requirements while optimizing delivery sequence
    - Real-time adaptation to special delivery instructions
2. Optimized Load Balancing
    - Effectively distributes packages across multiple vehicles
    - Considers vehicle capacity constraints
    - Balances workload while maintaining delivery priorities
    - Minimizes total fleet mileage through local optimization

F2. Verification of Algorithm

The implemented algorithm meets all scenario requirements:

- Successfully delivers all packages within specified deadlines
- Handles special instructions (delayed arrivals, address corrections)
- Maintains package grouping requirements
- Manages vehicle capacity limits of 16 packages
- Accommodates all address updates and corrections
- Works within driver shift constraints
- Operates within speed limits and distance calculations
- Handles interdependent deliveries
- Successfully processes wrong address corrections at 10:20 AM
- Maintains hub departure times for each truck
- Tracks and reports delivery status in real-time

F3. Other Possible Algorithms

Alternative approaches that could have been implemented include:

1. Simulated Annealing
2. Genetic Algorithm

F3a. Algorithm Differences

Key differences between the chosen 3-opt algorithm and alternatives:

1. Simulated Annealing differs from the implemented solution by:
    - Uses probabilistic acceptance of worse solutions to escape local optima
    - Gradually decreases solution space exploration over time
    - More flexible in handling dynamic constraints
    - Different approach to balancing exploration vs exploitation
2. Genetic Algorithm differs from the implemented solution by:
    - Evolves multiple solutions simultaneously instead of iterative improvement
    - Uses population-based approach rather than single route optimization
    - Employs crossover and mutation operations instead of local search
    - Different mechanism for handling constraints through fitness functions

G. Different Approach

A significant alternative approach would be implementing a dynamic programming solution:

- Could guarantee optimal solutions for small to medium-sized problems
- Would handle time windows more elegantly
- Better theoretical worst-case performance
- However, would require more memory
- More complex implementation
- May not scale well with additional constraints

H. Verification of Data Structure

The hash table implementation proves effective because:

- O(1) average case lookup and insertion
- Efficient handling of 40+ packages
- Dynamic resizing prevents performance degradation
- Collision handling via chaining maintains performance under load
- Memory usage scales linearly with number of packages

H1. Other Data Structures

Alternative data structures considered:

- Binary Search Tree
- Array-based list

H1a. Data Structure Differences

Comparison with alternatives:

1. Binary Search Tree:
    - O(log n) operations
    - More complex balancing required
    - Better sorted traversal
    - Higher memory overhead
2. Array-based list:
    - Simpler implementation
    - O(n) lookup time
    - No collision handling needed
    - Poor scaling for larger datasets

The chosen hash table implementation provides the best balance of:

- Implementation complexity
- Performance characteristics
- Memory efficiency
- Scalability for future expansion

I. Sources

Lysecky, R., & Vahid, F. (2018, June). *C950: Data Structures and Algorithms II*. zyBooks.

Retrieved March 22, 2021, from https://learn.zybooks.com/zybook/WGUC950AY20182019/