

**GROUP ASSIGNMENT**

**CT109-3-1-DGTIN**

|  |  |
| --- | --- |
| **STUDENT NAMES & TP:** | Zhassulan Niyazaly TP071009  Musaev Beniamin TP071185  Khegai Olga TP069365 |
| **INTAKE CODE:** | APD1F2309CS(CYB) |
| **MODULE NAME:** | PYTHON PROGRAMMING |
| **LECTURER NAME:** | WONG KENG TUCK |
| **HAND-OUT DATE:** | WEEK 3 |
| **HAND-IN DATE:** | WEEK 12 |

Table of Contents

[1.Introduction 3](#_Toc154941954)

[Design of the program(Pseudocode) 3](#_Toc154941955)

[2. Program source code and explanation 22](#_Toc154941956)

[3. Screenshots of sample input/output and explanation 95](#_Toc154941957)

[4.Conclusion 115](#_Toc154941958)

[5. References 116](#_Toc154941959)

# 1.Introduction

Title of code: Parcel delivery service

The Parcel delivery service program is a sophisticated, user-friendly system designed to streamline the process of calculating and billing the cost of parcel deliveries. Users can log in as an administrator or operator, with functionalities tailored to each role. Administrators have the ability to manage operators, ensuring smooth system operation. They can add and remove operators as needed, granting or revoking access to various system features. This level of control helps maintain security and accountability within the program. Operators have access to a range of tools designed to facilitate efficient parcel cost calculation and management. One of the key features is the algorithm that considers factors such as weight, size, and distance to accurately determine the cost of each parcel. This ensures fair billing and helps prevent overcharging or undercharging customers. Additionally, operators can effectively manage the customer base by adding new customers and removing outdated or inactive ones. This allows for a dynamic and up-to-date record of customers, ensuring accurate billing and effective communication. Overall, the Parcel delivery service program simplifies the parcel billing process, enhances user experience, and improves business operations. Its user-friendly interface and comprehensive functionalities make it a valuable tool for any parcel delivery service.

# 2.Design of the program(Pseudocode)

START

# Initialize user data

user\_dict = create\_userlist()

admin\_list = user\_dict["admins"]

operator\_list = user\_dict["operators"]

# Initialize pricetable data

pricetable = create\_pricetable()

# Initialize zones

zones = ["ZoneA", "ZoneB", "ZoneC", "ZoneD", "ZoneE"]

# File paths

customer\_file\_name = "customers.txt"

parcel\_file\_name = "parcels.txt"

bills\_file\_name = "bills.txt"

parcel\_number\_file\_path = "parcel\_number.txt"

# Initialize customer data and IDs

customers = []

last\_customer\_id = 10000000

last\_parcel\_num = 1

# Initialize price table for different zones and weights

price\_table = {

"A": {"1": 8.00, "3": 16.00},

"B": {"1": 9.00, "3": 18.00},

"C": {"1": 10.00, "3": 20.00},

"D": {"1": 11.00, "3": 22.00},

"E": {"1": 12.00, "3": 24.00},

}

# Function to create a bill file if it doesn't exist

create\_bill\_file()

# Function to create a parcels file if it doesn't exist

create\_parcels\_file()

# Function to load data

function load\_data():

load\_customer\_data()

load\_parcel\_data()

# Function to save data

function save\_data():

save\_customer\_data()

save\_parcel\_data()

# Additional utility functions

function add\_user():

# Input: User details (username, password, role)

# Output: None

# Process: Adds a new user to the system

username = input("Enter username: ")

password = input("Enter password: ")

role = input("Enter role (admin/operator): ")

new\_user = create\_user(username, password, role)

user\_list = get\_user\_list()

user\_list.append(new\_user)

save\_user\_list(user\_list)

function assign\_admin():

# Input: User ID

# Output: None

# Process: Assigns admin status to a user

user\_id = input("Enter user ID to assign admin status: ")

user\_list = get\_user\_list()

for user in user\_list:

if user["id"] == user\_id:

user["role"] = "admin"

save\_user\_list(user\_list)

print(f"Admin status assigned to user {user\_id}.")

break

else:

print("User not found.")

function remove\_admin():

# Input: User ID

# Output: None

# Process: Removes admin status from a user

user\_id = input("Enter user ID to remove admin status: ")

user\_list = get\_user\_list()

for user in user\_list:

if user["id"] == user\_id and user["role"] == "admin":

user["role"] = "operator"

save\_user\_list(user\_list)

print(f"Admin status removed from user {user\_id}.")

break

else:

print("User not found or not an admin.")

function del\_user():

# Input: User ID

# Output: None

# Process: Deletes a user from the system

user\_id = input("Enter user ID to delete: ")

user\_list = get\_user\_list()

for i, user in enumerate(user\_list):

if user["id"] == user\_id:

del user\_list[i]

save\_user\_list(user\_list)

print(f"User {user\_id} deleted.")

break

else:

print("User not found.")

function view\_list():

# Input: None

# Output: User list

# Process: Displays a list of users

user\_list = get\_user\_list()

if user\_list:

print("User List:")

for user in user\_list:

print(f"ID: {user['id']}, Username: {user['username']}, Role: {user['role']}")

else:

print("No users found.")

function add\_price\_over3kg():

# Input: Zone, price

# Output: None

# Process: Adds a price for parcels over 3kg to the pricetable

zone = input("Enter destination zone (A, B, C, D, E): ")

price = float(input("Enter price for parcels over 3kg: "))

pricetable[zone]["3"] = price

save\_pricetable(pricetable)

print(f"Price for zone {zone} updated to {price}.")

function modify\_price\_over3kg():

# Input: Zone, new price

# Output: None

# Process: Modifies the price of parcels over 3kg in the pricetable

zone = input("Enter destination zone (A, B, C, D, E): ")

current\_price = pricetable[zone]["3"]

new\_price = float(input(f"Current price for zone {zone}: {current\_price}. Enter new price: "))

pricetable[zone]["3"] = new\_price

save\_pricetable(pricetable)

print(f"Price for zone {zone} updated to {new\_price}.")

function delete\_price\_over3kg():

# Input: Zone

# Output: None

# Process: Deletes the price of parcels over 3kg in the pricetable

zone = input("Enter destination zone (A, B, C, D, E): ")

if zone in pricetable:

del pricetable[zone]["3"]

save\_pricetable(pricetable)

print(f"Price for zone {zone} deleted.")

else:

print(f"No price found for zone {zone}.")

function check\_price():

# Input: Weight, destination zone

# Output: Parcel price

# Process: Checks the price of a parcel based on weight and destination zone

weight = float(input("Enter parcel weight (below 3kg): "))

zone = input("Enter destination zone (A, B, C, D, E): ")

if weight <= 3 and zone in pricetable:

price = pricetable[zone].get(str(weight), None)

if price is not None:

print(f"The price for a {weight}kg parcel to destination {zone} is RM{price}.")

else:

print("Invalid weight.")

else:

print("Invalid weight or destination zone.")

function view\_all\_prices():

# Input: None

# Output: None

# Process: Displays all prices in the pricetable

print("Price Table:")

for zone, weights in pricetable.items():

for weight, price in weights.items():

print(f"Zone: {zone}, Weight: {weight}kg, Price: RM{price}.")

# Beginning of the program

welcome()

function admin\_menu():

# Input: None

# Output: None

# Process: Displays the admin menu and handles admin actions

while True:

display\_menu\_options()

choice = get\_user\_input()

if choice == "a":

add\_user()

elif choice == "b":

assign\_admin()

elif choice == "c":

remove\_admin()

elif choice == "d":

del\_user()

elif choice == "e":

view\_list()

elif choice == "f":

add\_price\_over3kg()

elif choice == "g":

modify\_price\_over3kg()

elif choice == "h":

delete\_price\_over3kg()

elif choice == "i":

check\_price()

elif choice == "j":

view\_all\_prices()

elif choice == "k":

welcome()

elif choice == "l":

save\_and\_exit()

else:

display\_invalid\_choice\_message()

function operator\_code():

# Input: None

# Output: None

# Process: Handles operator actions

load\_data()

while True:

display\_operator\_menu\_options()

operator\_choice = get\_operator\_input()

if operator\_choice == "a":

add\_customer()

elif operator\_choice == "b":

modify\_cus\_details()

elif operator\_choice == "c":

calculate\_parcel\_price()

elif operator\_choice == "d":

parcels\_received()

elif operator\_choice == "e":

operator\_menu\_exit()

else:

display\_invalid\_operator\_choice\_message()

function load\_data():

# Input: None

# Output: None

# Process: Loads customer and parcel data

load\_customer\_data()

load\_parcel\_data()

function save\_data():

# Input: None

# Output: None

# Process: Saves customer and parcel data

save\_customer\_data()

save\_parcel\_data()

function add\_customer():

# Input: Customer details (name, address, telephone)

# Output: None

# Process: Adds a new customer to the system

name = input("Enter customer name: ")

address = input("Enter customer address: ")

telephone = input("Enter customer telephone: ")

customer\_id = add\_customer(name, address, telephone)

if customer\_id is not None:

print(f"Customer added successfully. Customer ID: {customer\_id}")

function modify\_cus\_details():

# Input: Customer ID, new address, new telephone number

# Output: None

# Process: Modifies customer data

customer\_id = int(input("Please enter customer id: "))

cus\_address = input("Please enter customer's new address: ")

phone\_num\_number = input("Please enter customer's new phone number: ")

if modify\_cus\_details(customer\_id, cus\_address, phone\_num\_number):

print("Customer details modified successfully.")

else:

print("Customer not found.")

function calculate\_parcel\_price():

# Input: Parcel weight, destination zone

# Output: None

# Process: Calculates the price for a parcel and displays it

weight = input("Enter parcel weight (below 3kg): ")

destination = input("Enter destination (A, B, C, D, E): ").upper()

price = calculate\_parcel\_price(weight, destination)

if price is not None:

print(f"The price for a {weight}kg parcel to destination {destination} is RM{price}.")

function parcels\_received():

# Input: Date received, destination zone

# Output: None

# Process: Displays parcels received on a specific date to a destination zone

date = input("Enter date received (DD/MM/YYYY): ")

destination = input("Enter destination (zone): ")

parcel\_list = parcels\_received(date, destination)

if not parcel\_list:

# Output: No parcels received message

output(f"No parcels received on {date} to destination {destination}.")

else:

# Output: Parcels received header

output(f"Parcels received on {date} to destination {destination}:")

# Loop through each parcel in the parcel list

for parcel in parcel\_list:

# Output: Parcel details

output(f"Parcel Number: {parcel['parcel\_number']}")

output(f"Receiver Name: {parcel['receiver\_name']}")

output(f"Weight: {parcel['weight']}")

output(f"Price: {parcel['price']}")

function operator\_menu\_exit():

# Input: None

# Output: None

# Process: Exits the operator menu and returns to the main menu

print("Going back to welcome page.")

save\_data()

welcome()

function generate\_user\_id():

# Input: None

# Output: New user ID

# Process: Generates a new user ID based on the last used ID

global last\_user\_id

last\_user\_id += 1

return last\_user\_id

# Operator Menu Functions

function delete\_parcel\_from\_bill(consignment\_number, parcel\_number):

# Input: Consignment number, parcel number

# Output: None

# Process: Deletes a parcel from a bill

bill = get\_bill\_by\_consignment(consignment\_number)

if bill is not None:

parcels = bill["customers"]["parcels"]

for i, parcel in enumerate(parcels):

if parcel["parcel\_number"] == parcel\_number:

del parcels[i]

save\_data()

print(f"Parcel {parcel\_number} deleted from consignment {consignment\_number}.")

return

print(f"Parcel {parcel\_number} not found in consignment {consignment\_number}.")

else:

print(f"Consignment {consignment\_number} not found.")

# Function to view details of a bill for a consignment number

function view\_bill(consignment\_number):

# Input: Consignment number

# Output: None

# Process: Displays details of a bill for a consignment number

bill = get\_bill\_by\_consignment(consignment\_number)

if bill is not None:

# Output: Display bill details

print(“A bill of the consignment:")

print('') # Empty line for separation

print(f"BillCreatedDate: {bill['date']}")

print(f"ConsignmentNumber: {bill['consignment\_number']}")

printt(f"FinalPrice: {bill['final\_price']}")

printt("Customer\_information:")

printt(f"Customer ID: {bill['customers']['id']}")

print(f"Name: {bill['customers']['name']}")

ptint(f"Address: {bill['customers']['address']}")

print(f"Telephone: {bill['customers']['telephone']}")

output("Parcel\_information:")

for parcel in bill['customers']['parcels']:

output(f"Parcel Number: {parcel['parcel\_number']}")

output(f"Receiver Name: {parcel['receiver\_name']}")

output(f"Receiver Address: {parcel['receiver\_address']}")

output(f"Receiver Telephone: {parcel['receiver\_telephone']}")

output(f"Destination: {parcel['destination']}")

output(f"Weight: {parcel['weight']}")

output(f"Price: {parcel['price']}")

output(f"Date Received: {parcel['date\_received']}")

output('') # Empty line for separation

else:

# Output: Display not found message

output(f"Consignment {consignment\_number} not found.")

# Function to view bills and total amount charged to a customer

function view\_cus\_bill(customer\_name):

# Input: Customer name

# Output: None

# Process: Displays a list of bills and total amount charged to a customer

count = 1

cus\_total = 0

match = False

for bill in get\_bills():

if bill["customers"]["name"].lower() == customer\_name.lower():

match = True

if count == 1:

# Output: Display header for customer bill

print("Bill of the customer:")

count = count - 1

# Output: Display bill details

print(f"BillCreatedDate: {bill['date']}")

print(f"ConsignmentNumber: {bill['consignment\_number']}")

print(f"FinalPrice: {bill['final\_price']}")

print("Customer\_information:")

print(f"Customer ID: {bill['customers']['id']}")

print(f”Name: {bill['customers']['name']}")

print(f"Address: {bill['customers']['address']}")

print(f"Telephone: {bill['customers']['telephone']}")

print("Parcel\_information:")

for parcel in bill['customers']['parcels']:

print(f"Parcel Number: {parcel['parcel\_number']}")

print(f"Receiver Name: {parcel['receiver\_name']}")

print(f"Receiver Address: {parcel['receiver\_address']}")

print(f"Receiver Telephone: {parcel['receiver\_telephone']}")

print(f"Destination: {parcel['destination']}")

print(f"Weight: {parcel['weight']}")

print(f"Price: {parcel['price']}")

print(f"Date Received: {parcel['date\_received']}")

print('') # Empty line for separation

cus\_total += bill["final\_price"]

if match:

# Output: Display total amount charged to the customer

print(f"Total amount charged to {customer\_name}: {cus\_total}")

else:

# Output: Display no bills found message

output(f"No bills found for the customer {customer\_name}.")

if match:

print("Total amount of the customer:", "RM", cus\_total)

else:

print("\nNo bills found for the customer.\n")

# Call the operator\_menu function to start the program

operator\_menu()

function get\_bills():

# Input: None

# Output: List of bills

# Process: Retrieves the list of bills

# Implementation details depend on how your data is structured

function get\_bill\_by\_consignment(consignment\_number):

# Input: Consignment number

# Output: Bill details or None if not found

# Process: Retrieves a bill based on the consignment number

# Implementation details depend on how your data is structured

# Other Operator Menu Functions

function display\_operator\_menu\_options():

# Input: None

# Output: None

# Process: Displays the operator menu options

print("\nOperator Menu:")

print("a. Add Customer")

print("b. Modify Customer Details")

print("c. Calculate Parcel Price")

print("d. Parcels Received")

print("e. Operator Menu Exit")

# Operator Menu Flow

function operator\_menu():

# Input: None

# Output: None

# Process: Displays the operator menu and handles operator actions

load\_data()

while True:

display\_operator\_menu\_options()

operator\_choice = get\_operator\_input()

if operator\_choice == "a":

add\_customer()

elif operator\_choice == "b":

modify\_cus\_details()

elif operator\_choice == "c":

calculate\_parcel\_price()

elif operator\_choice == "d":

parcels\_received()

elif operator\_choice == "e":

operator\_menu\_exit()

else:

display\_invalid\_operator\_choice\_message()

# Function to welcome users

function welcome():

# Initialize user variable

user = "False"

# Output: Welcome message

output("===================================")

output("Hello, my friend!")

output("Welcome to our own Shopee:)")

output("===================================")

# Input: Ask if the user has an account

ask = input("Do you have an account? (yes/no): ").lower()

# Check if the user has an account

if ask == "yes":

# Input: Enter login and password

login = input("Enter your login: ").lower()

password = input("Enter your password: ").lower()

# Loop through admin list

for item in admin\_list:

# Check if login and password match admin credentials

if login == item[0] and password == item[1]:

# Update user variable

user = "True"

# Call admin code function

admin\_code()

# Loop through operator list

for oper in operator\_list:

# Check if login and password match operator credentials

if login == oper[0] and password == oper[1]:

# Update user variable

user = "True"

# Call operator code function

operator\_code()

# Check if user was not found

if user == "False":

# Output: User not found message

output("User was not found! Try again.")

# Recursive call to welcome function

welcome()

# If the user doesn't have an account

elif ask == "no":

# Output: Cannot join the system message

output("Oops... Then you can't join the system!")

# Call bye function

bye()

# Exit the system

sys.exit()

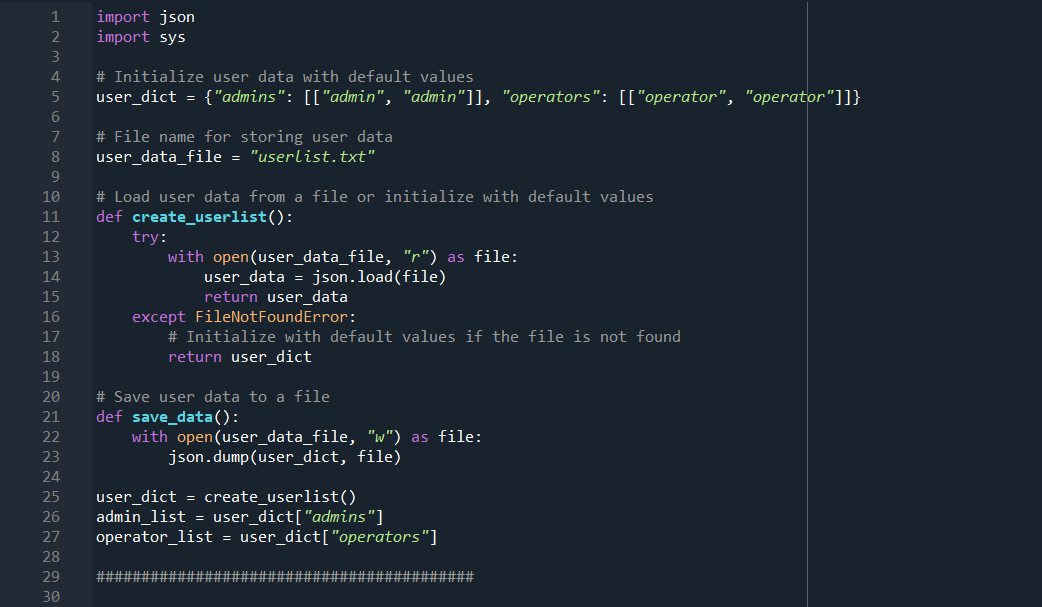
# program starts with a welcome function

welcome()

END

# 3. Program source code and explanation

**Managing user data and pricetable**



*Figure 1: Code snippet of managing user data*

The provided Python script serves as a user data management system, specifically handling user credentials for administrators and operators. The data is stored in JSON format, and the script offers functionality to load and save user data to a designated file.

Variables used in the code snippet in Figure 1:

|  |  |
| --- | --- |
| Variables | Description |
| user\_dict | A dictionary containing user data with default values. It includes lists of administrators and operators. |
| user\_data\_file | A string representing the filename for storing user data in JSON format. Default is "userlist.txt". |
| admin\_list | A list of administrator credentials stored in user\_dict. Each entry is a list of username and password. |
| operator\_list | A list of operator credentials stored in user\_dict. Each entry is a list of username and password. |

Code snippet justification in Figure 1:

|  |  |
| --- | --- |
| Line number | Justification |
| 1-2 | Import necessary modules (json and sys). |
| 5 | Default user data with administrator and operator credentials. |
| 8 | Filename for storing user data in JSON format. |
| 11-18 | create\_userlist function definition. |
| 12-16 | Attempts to open the file and load user data. If the file is not found, the user data is initialized with default values. |
| 21-24 | Definition of the save\_data function. |
| 22-23 | Writes the current user data to the specified file in JSON format. |
| 26 | Invokes the create\_userlist function to initialize or load user data. |
| 27-28 | Extracts administrator and operator lists from user\_dict. |

Functions:

* create\_userlist():

Loads user data from a file (user\_data\_file) using JSON.

If the file is not found, initializes with default values (user\_dict).

* save\_data():

Saves the current user data (user\_dict) to the file specified by u ser\_data\_file.



*Figure 2: Code snippet of price table management*

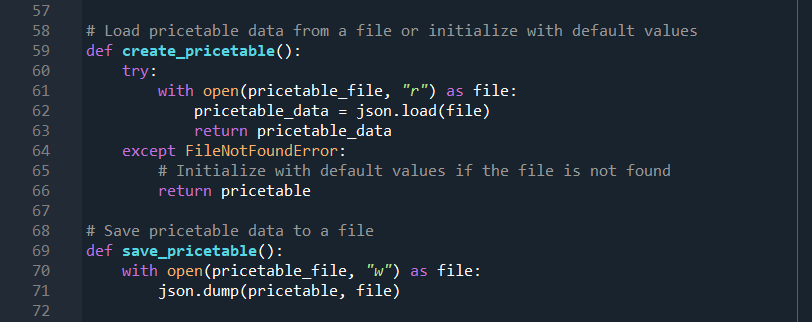
This script provides a framework for storing and accessing pricing information, facilitating easy retrieval based on weight categories and delivery zones. The hierarchical structure of the pricetable dictionary simplifies the organization of pricing data.

Variables used in the code snippet in Figure 2:

|  |  |
| --- | --- |
| Variables | Description |
| pricetable\_file | A string representing the filename for storing pricing data. The default is set to "pricetable.txt". |
| pricetable | A nested dictionary representing the pricing structure. It includes categories for different weight ranges and corresponding prices for various delivery zones. |

Code snippet justification in Figure 2:

|  |  |
| --- | --- |
| Line number | Justification |
| 33 | pricetable\_file: Variable storing the filename for pricing data, facilitating easy identification of the data storage file. |
| 36-55 | pricetable Dictionary Initialization: Definition of the pricetable dictionary with default pricing values. The nested structure allows for easy organization and retrieval of pricing data. |
| 49-54 | Adds a placeholder for the "over3kg" category with zero cost for all zones. Adds a placeholder for the "over3kg" category with zero cost for all zones. |
| 55 | Closes the pricetable definition. |

*Figure 3: Code snippet of price table handling*

The following code snippet is designed for managing price table data. It provides functions to load and save the price table, either from a file in JSON format or by initializing with default values.

Variables used in the code snippet in Figure 3:

|  |  |
| --- | --- |
| Variables | Description |
| pricetable\_file | A string representing the filename for storing the price table data in JSON format. |
| pricetable | Default price table data, initialized with default values. |

Code snippet justification in Figure 2:

|  |  |
| --- | --- |
| Line number | Justification |
| 59-64 | Definition of the create\_pricetable function. The function handles loading or initializing price table data. |
| 69-71 | Definition of the save\_pricetable function. The function handles saving the current price table data to a file. |

Functions:

* create\_pricetable():

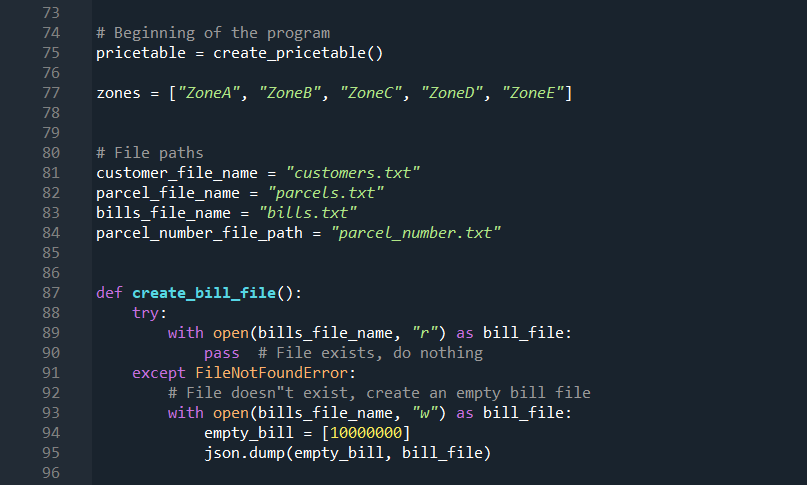
Attempts to load the price table data from a file (pricetable\_file) using JSON.

If the file is not found, initializes the price table with default values (pricetable).

* save\_pricetable():

Writes the current price table data (pricetable) to the file specified by pricetable\_file in JSON format.

**Parcel management**

*Figure 4: Code snippet of initializing program and creating bill file*

The following code snippet represents the beginning of a program. It initializes a price table and sets up file paths for customer information, parcel details, bills, and a file to store parcel numbers. Additionally, it contains a function to create a bill file if it doesn't already exist.

Variables used in the code snippet in Figure 4:

|  |  |
| --- | --- |
| Variables | Description |
| pricetable | Holds the price table, likely a data structure storing pricing information. Its creation is dependent on a create\_pricetable function. |
| zones | List of strings representing different zones. |
| customer\_file\_name | String indicating the filename for customer information. Default is "customers.txt". |
| parcel\_file\_name | String indicating the filename for parcel information. Default is "parcels.txt". |
| bills\_file\_name | String indicating the filename for storing bills. Default is "bills.txt". |
| parcel\_number\_file\_path | String indicating the file path for storing parcel numbers. Default is "parcel\_number.txt". |

Code snippet justification in Figure 4:

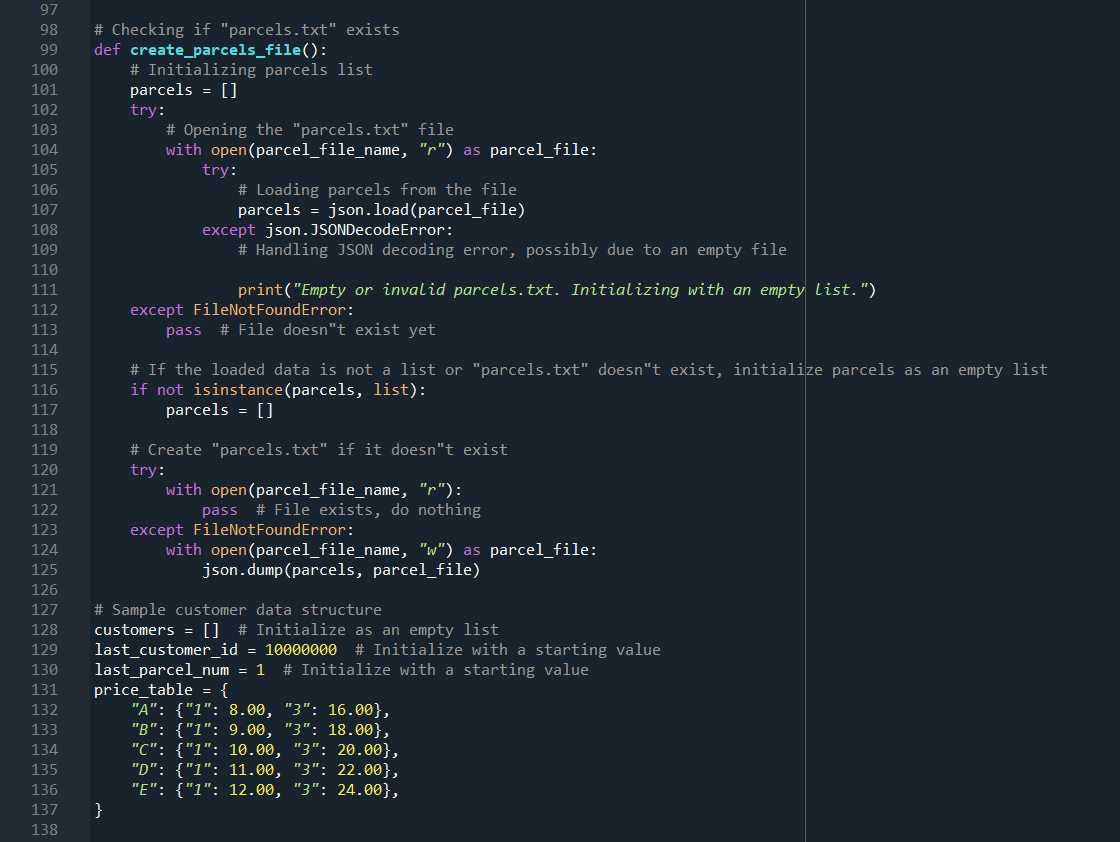
|  |  |
| --- | --- |
| Line number | Justification |
| 75 | Initializes the pricetable using create\_pricetable function. |
| 77 | List of strings representing different zones. |
| 81-84 | File paths for customer data, parcel data, bills, and parcel numbers. |
| 87-95 | Definition of the create\_bill\_file function. |
| 88-95 | Tries to open the bill file. If not found, creates an empty bill file and initializes it with a default bill number. |

Functions:

* create\_bill\_file():

Checks if the bill file already exists.

If not found, creates an empty bill file and initializes it with a default bill number (e.g., [10000000]).

*Figure 5: Code snippet of parcel management*

This code snippet manages parcels, specifically loading and initializing data from a file named "parcels.txt". It also sets up a structure for customer data and defines a price table based on parcel categories.

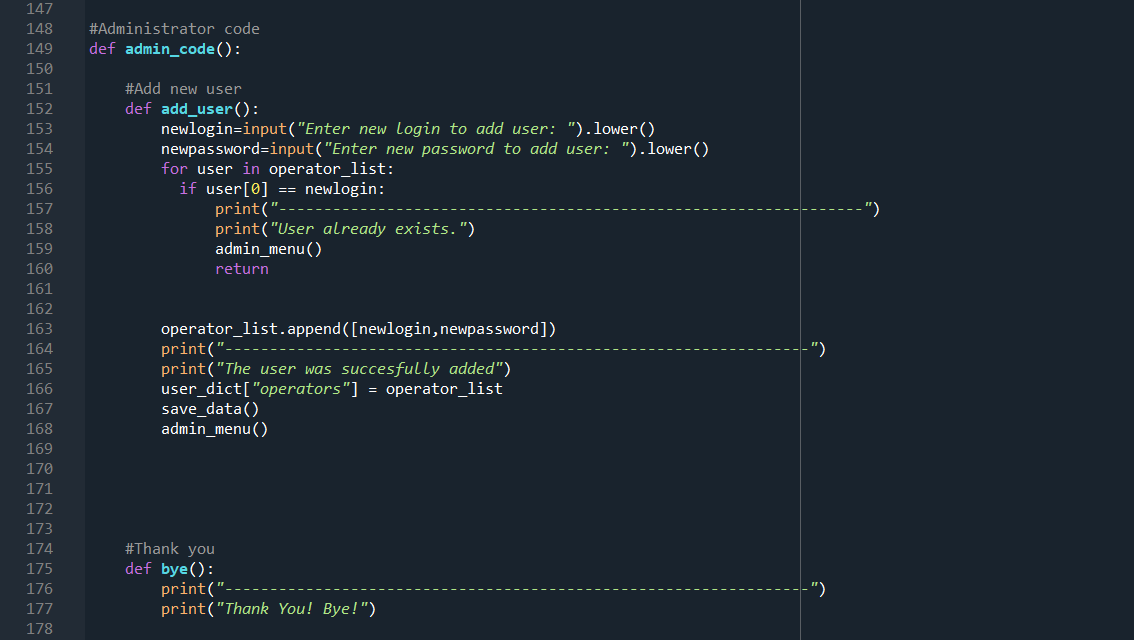
Variables used in the code snippet in Figure 5:

|  |  |
| --- | --- |
| Variables | Description |
| parcels | A list to store parcel data, loaded from "parcels.txt" or initialized as an empty list. |
| parcel\_file\_name | The filename for storing parcel data in JSON format, set to "parcels.txt". |
| customers | A list to store customer data, initialized as an empty list. |
| last\_customer\_id | An integer representing the last assigned customer ID, initialized with a starting value of 10,000,000. |
| last\_parcel\_num | An integer representing the last assigned parcel number, initialized with a starting value of 1. |
| price\_table | A nested dictionary representing the price table for different parcel categories and quantities. |

Code snippet justification in Figure 5:

|  |  |
| --- | --- |
| Line number | Justification |
| 99-125 | Definition of the create\_parcels\_file function. |
| 102-113 | Tries to open "parcels.txt" and load parcel data. If the file is not found or empty, initializes with an empty list. Handles JSON decoding errors. |
| 116-117 | If the loaded data is not a list or "parcels.txt" doesn't exist, initializes parcels as an empty list. |
| 120-125 | Attempts to open "parcels.txt" to check its existence. If it doesn't exist, creates the file and writes the default parcels list in JSON format. |
| 128 | Initializes the customers list as an empty list. |
| 129 | Initializes the last\_customer\_id with a starting value. |
| 130 | Initializes the last\_parcel\_num with a starting value. |
| 131-137 | Initializes the price\_table with predefined values for different parcel categories and quantities. |

**Administrator**

*Figure 6: Code snippet of administrator functions*

This code snippet defines functions within an administrator menu, allowing the addition of new users and providing a farewell message.

Variables used in the code snippet in Figure 6:

|  |  |
| --- | --- |
| Variables | Description |
| newlogin | Input variable capturing the new login for adding a user. |
| newpassword | Input variable capturing the new password for adding a user. |

Code snippet justification in Figure 6:

|  |  |
| --- | --- |
| Line number | Justification |
| 149 | Definition of the admin\_code function, which contains all functions for admin part |
| 152-168 | Definition of the add\_user function within admin\_code. Allows the administrator to add new users to the system. |
| 153-154 | Prompting the administrator for a new login and password. |
| 155-160 | Checking if the user already exists among operators and providing appropriate feedback. |
| 163-168 | Adding the new user to the list of operators, updating the user dictionary, saving the data, and returning to the administrator menu. |
| 175-177 | Definition of the bye function. Prints a farewell message. |

Functions:

* add\_user():

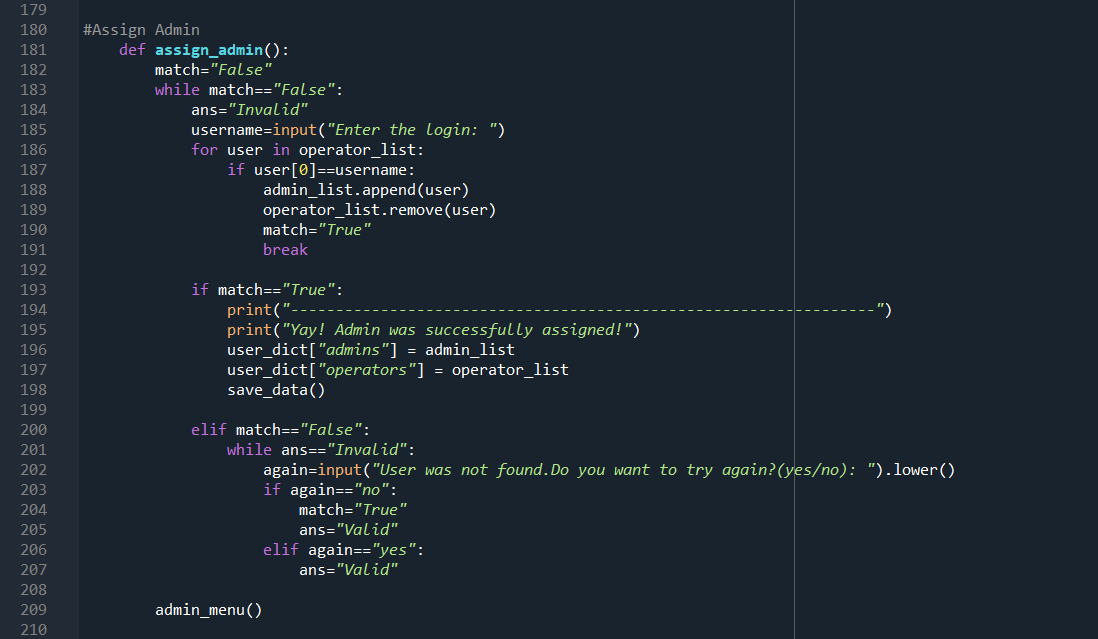
Allows the administrator to input a new login and password for adding a user to the system.

Checks if the user already exists among operators and prints a message if so.

Appends the new user to the list of operators, updates the user dictionary, saves the data, and returns to the administrator menu.

* bye():

Prints a farewell message when called.

*Figure 7: Code snippet of administrator assigning functions*

This code snippet defines a function, assign\_admin(), which facilitates the assignment of an operator to the administrator role. The user is prompted to enter a login, and if a matching operator is found, it is moved to the admin list. The user is given the option to try again in case of an invalid input. Upon successful assignment, the updated user data is saved, and the admin menu is invoked.

Variables used in the code snippet in Figure 7:

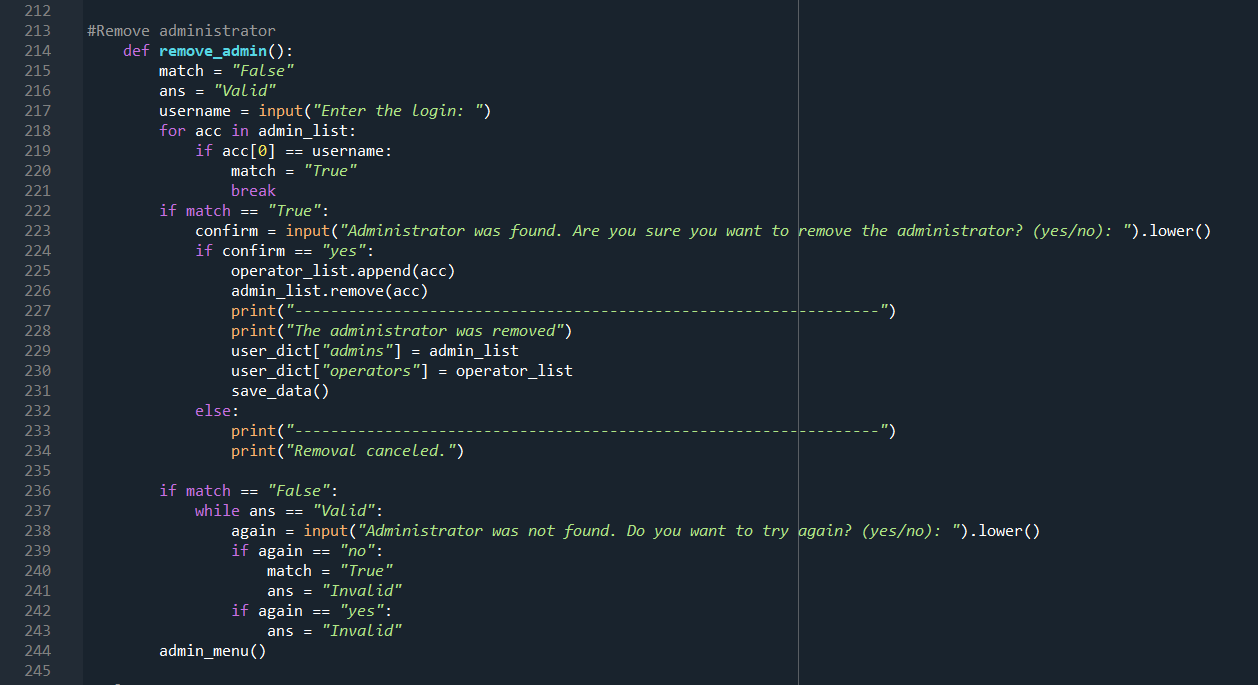
|  |  |
| --- | --- |
| Variables | Description |
| match | Input variable capturing the new login for a flag indicating whether a matching operator has been found. Initialized as "False" and set to "True" upon a successful match. dding a user. |
| ans | A flag representing the validity of the user's input. Initialized as "Invalid" and set to "Valid" when the user decides whether to try again. |
| username | User-input login to search for in the operator list. |

Code snippet justification in Figure 7:

|  |  |
| --- | --- |
| Line number | Justification |
| 181-207 | Function definition for assigning an operator to the admin role. |
| 182 | Initializing the match flag as "False". |
| 183-185 | While loop to continue until a matching operator is found. |
| 186-191 | Prompting the user for a login and checking for a match in the operator list. If found, the operator is moved to the admin list. |
| 193-198 | Displaying success message, updating user data, saving to the file, and breaking out of the loop. |
| 200-207 | If no match is found, the user is prompted to try again or exit. |
| 209 | Invoking the admin menu after successful admin assignment. |

Functionality:

* The user is prompted to enter a login (username).
* The script checks if the entered login matches any operator in the operator\_list.
* If a match is found, the corresponding operator is moved from operator\_list to admin\_list.
* User data is updated, saved to the file, and an acknowledgment is displayed.
* If no match is found, the user is given the option to try again or exit.

*Figure 8: Code snippet of administrator removal functions*

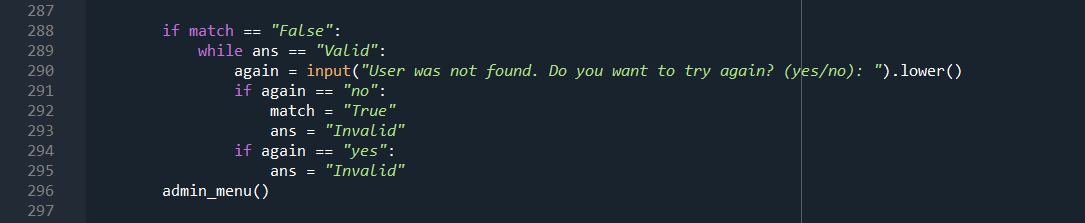
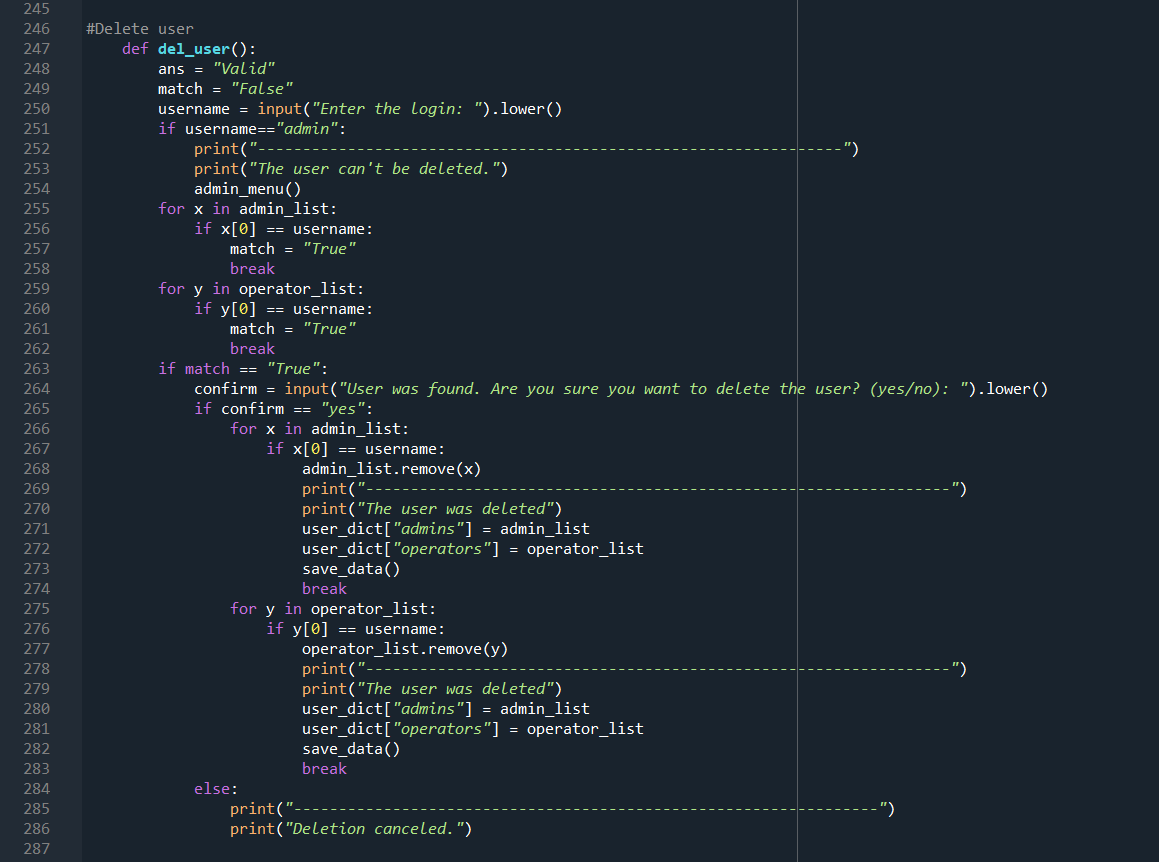
The provided Python code snippet defines a function, remove\_admin(), responsible for removing an administrator from the user data. It prompts the user for a username, checks if the provided username corresponds to an existing administrator, and if confirmed, removes the administrator. If the username is not found, the user has the option to retry or cancel the removal.

Variables used in the code snippet in Figure 8:

|  |  |
| --- | --- |
| Variables | Description |
| match | A flag indicating whether a matching administrator was found ("True") or not ("False"). |
| ans | A flag determining the validity of the user's input during retry prompts ("Valid" or "Invalid"). |
| username | The username entered by the user for administrator removal. |
| admin\_list | List of administrator credentials stored in user\_dict. |
| operator\_list | List of operator credentials stored in user\_dict. |
| confirm | User input to confirm the removal of the identified administrator. |
| again | User input during the retry prompt for attempting removal again. |

Code snippet justification in Figure 8:

|  |  |
| --- | --- |
| Line number | Justification |
| 215-217 | Initialization of flags and user input for the username. |
| 218-221 | Iterates through the list of administrators to check for a match with the entered username. |
| 222-234 | If a match is found, prompts the user for confirmation to remove the administrator. If confirmed, updates the lists, prints a removal message, updates user\_dict, and saves the data. If not confirmed, cancellation message is printed. |
| 236-243 | If no match is found, provides an option to the user to retry or cancel the removal process. The loop ensures valid input for retry decisions. |
| 244 | Calls admin\_menu() to return to the administrator menu after removal or cancellation. |

*Figure 9: Code snippet of user deletion functions*

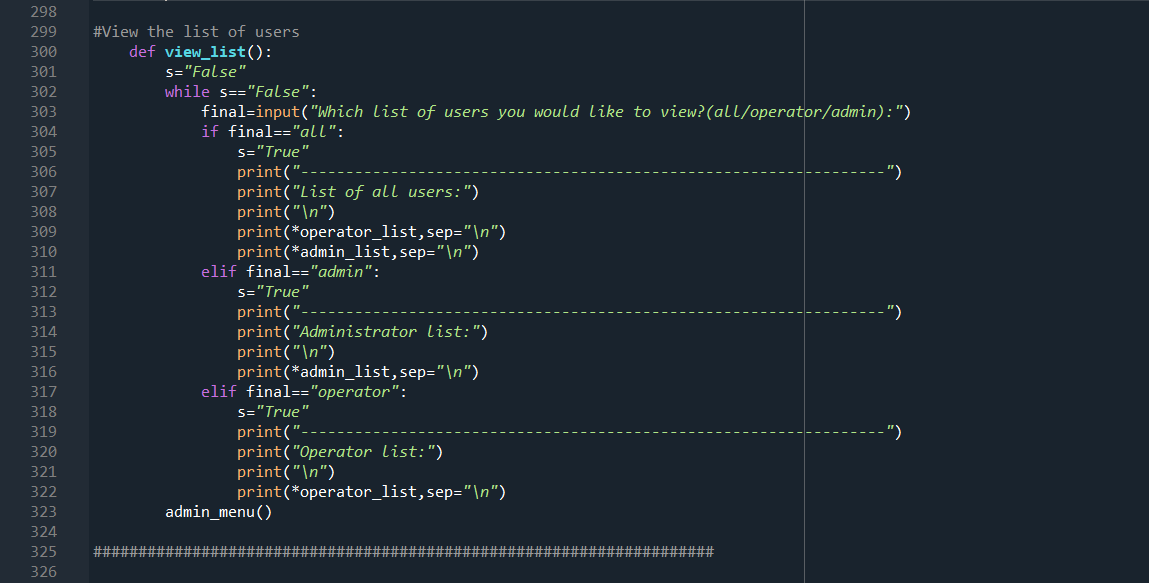
This code snippet offers a flexible and interactive mechanism for removing users, providing confirmation prompts and retry options while handling different user roles.

Variables used in the code snippet in Figure 9:

|  |  |
| --- | --- |
| Variables | Description |
| match | A flag indicating whether a matching administrator was found ("True") or not ("False"). |
| ans | A flag determining the validity of the user's input during retry prompts ("Valid" or "Invalid"). |
| username | The username entered by the user for administrator removal. |
| admin\_list | List of administrator credentials stored in user\_dict. |
| operator\_list | List of operator credentials stored in user\_dict. |
| confirm | User input to confirm the deletion of the identified user. |

Code snippet justification in Figure 9:

|  |  |
| --- | --- |
| Line number | Justification |
| 248-254 | Initialization of flags and user input for the username. If the username is "admin," it cannot be deleted, and the function returns to the admin menu. |
| 255-258 | Loop to check if the entered username matches an administrator's username. |
| 259-262 | Loop to check if the entered username matches an operator's username. |
| 263-286 | If a match is found, prompts the user for confirmation to delete the user. If confirmed, updates the lists, prints a deletion message, updates user\_dict, and saves the data. If not confirmed, cancellation message is printed. |
| 288-295 | If no match is found, provides an option to the user to retry or cancel the deletion process. The loop ensures valid input for retry decisions. |
| 296 | Calls admin\_menu() to return to the administrator menu after deletion or cancellation. |

*Figure 10: Code snippet of user viewing list function*

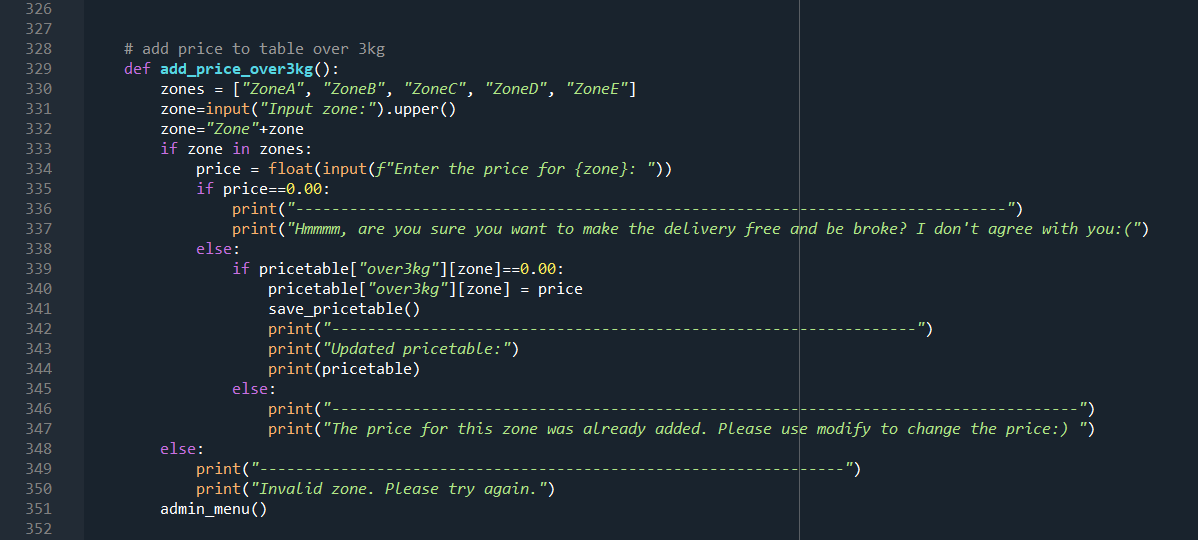
The provided code snippet defines a function, view\_list(), which allows the user to view lists of users based on specified categories. The function prompts the user to choose from options like viewing all users, only administrators, or only operators. It then displays the selected list and returns to the administrator menu.

Variables used in the code snippet in Figure 10:

|  |  |
| --- | --- |
| Variables | Description |
| s | A flag indicating whether the user has made a valid selection ("True") or not ("False"). |
| final | User input specifying the category of user list to be viewed (e.g., "all", "admin", "operator"). |

Code snippet justification in Figure 10:

|  |  |
| --- | --- |
| Line number | Justification |
| 302-303 | While loop to ensure a valid user selection. Prompts the user to choose from options like viewing all users, only administrators, or only operators. |
| 304-322 | Conditional blocks for handling user selections. If "all" is chosen, it prints the list of all users (operators and administrators). If "admin" is chosen, it prints only the administrator list. If "operator" is chosen, it prints only the operator list. The loop continues until a valid selection is made. |
| 323 | Calls admin\_menu() to return to the administrator menu after viewing the selected user list. |

*Figure 11: Code snippet of adding price over 3kg function*

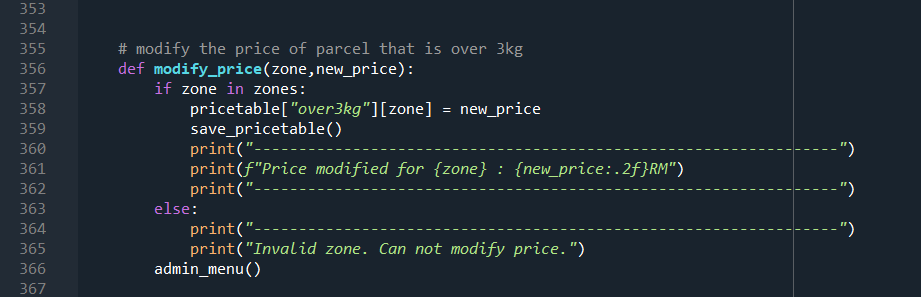
The provided Python code snippet defines a function, add\_price\_over3kg(), which enables the user (presumably an administrator) to add or update prices for delivery zones specifically for items over 3 kilograms. The function prompts the user for a delivery zone, checks its validity, and allows the input of a new price. It provides appropriate feedback based on the entered values and updates the price table accordingly.

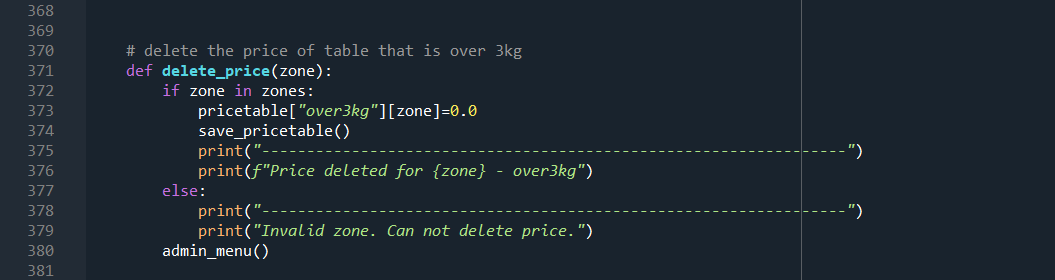
Variables used in the code snippet in Figure 11:

|  |  |
| --- | --- |
| Variables | Description |
| zones | A list containing the valid delivery zones. |
| zone | User input specifying the delivery zone for which the price is to be added or updated. |
| price | User input representing the new price for the specified delivery zone. |
| pricetable | A dictionary storing delivery prices, specifically for items over 3 kilograms. |

Code snippet justification in Figure 11:

|  |  |
| --- | --- |
| Line number | Justification |
| 329 | Definition of valid delivery zones (zones). |
| 330-332 | User input for the delivery zone, converting it to uppercase and appending "Zone" to match the format in zones. |
| 333-334 | Checks if the entered zone is valid. If so, prompts the user for a new price and provides appropriate feedback based on the entered value. |
| 335-337 | If the price for the specified zone is zero, prints a message discouraging free delivery. |
| 339-350 | Updates the price in the pricetable dictionary and saves the updated table. Prints the updated pricetable. If the price for the zone was already added, a message is displayed, recommending the use of the modify function. |
| 351 | Calls admin\_menu() to return to the administrator menu. |

*Figure 12: Code snippet of modifying price over 3kg function*

*Figure 13: Code snippet of deleting price over 3kg function*

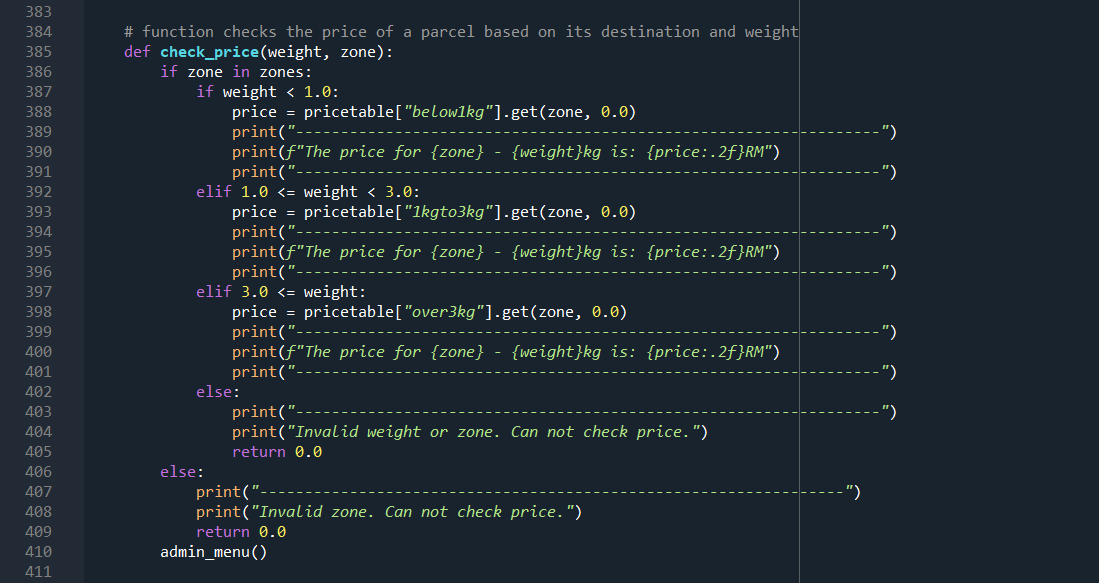
The provided Python code snippet includes two functions, modify\_price(zone, new\_price) and delete\_price(zone). These functions are designed to modify and delete prices for delivery zones, specifically for items over 3 kilograms. The modify\_price() function takes a delivery zone and a new price as inputs, modifies the price in the pricetable dictionary, saves the updated table, and prints a confirmation message. The delete\_price() function takes a delivery zone as input, sets the price for the specified zone to zero in the pricetable dictionary, saves the updated table, and prints a deletion confirmation message.

Variables used in the code snippet in Figure 12, 13:

|  |  |
| --- | --- |
| Variables | Description |
| zones | A list containing the valid delivery zones. |
| zone | Delivery zone for which the price is to be modified or deleted. |
| new\_price | User input representing the new price for the specified delivery zone. |
| pricetable | A dictionary storing delivery prices, specifically for items over 3 kilograms. |

Code snippet justification in Figure 12, 13:

|  |  |
| --- | --- |
| Line number | Justification |
| 356-365 | modify\_price() function definition. Modifies the price in the pricetable dictionary, saves the updated table, and prints a confirmation message. |
| 371-379 | delete\_price() function definition. Sets the price for the specified zone to zero in the pricetable dictionary, saves the updated table, and prints a deletion confirmation message. |
| 366, 380 | Call to admin\_menu() to return to the administrator menu after modifying or deleting the price. |

*Figure 14: Code snippet of checking price function*

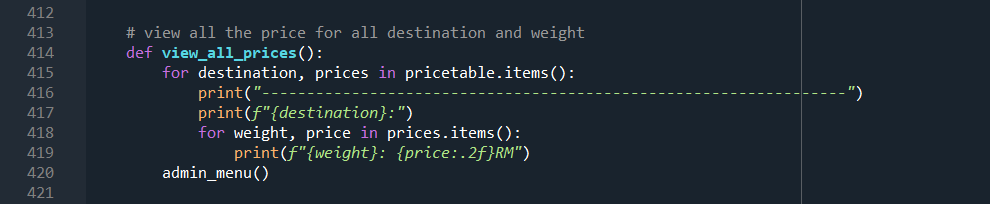
The provided Python code snippet defines a function, check\_price(weight, zone), designed to check and display the price for a given weight and delivery zone. The function validates the provided zone, checks the weight range, retrieves the corresponding price from the pricetable, and prints the result. If the weight or zone is invalid, an appropriate message is displayed.

Variables used in the code snippet in Figure 14:

|  |  |
| --- | --- |
| Variables | Description |
| zones | A list containing the valid delivery zones. |
| zone | Delivery zone for which the price is to be checked. |
| price | The retrieved price based on the weight and zone. |
| pricetable | Dictionary storing prices for items of different weight ranges, organized by delivery zones. |
| weight | Weight of the item for which the price is to be checked. |

Code snippet justification in Figure 14:

|  |  |
| --- | --- |
| Line number | Justification |
| 385-401 | Checks if the entered zone is valid. If valid, it determines the weight range and retrieves the corresponding price from the pricetable. The result is printed. |
| 402-409 | If the weight or zone is invalid, prints an appropriate message. |
| 410 | Calls admin\_menu() to return to the administrator menu after checking the price. |

*Figure 15: Code snippet of viewing all prices price function*

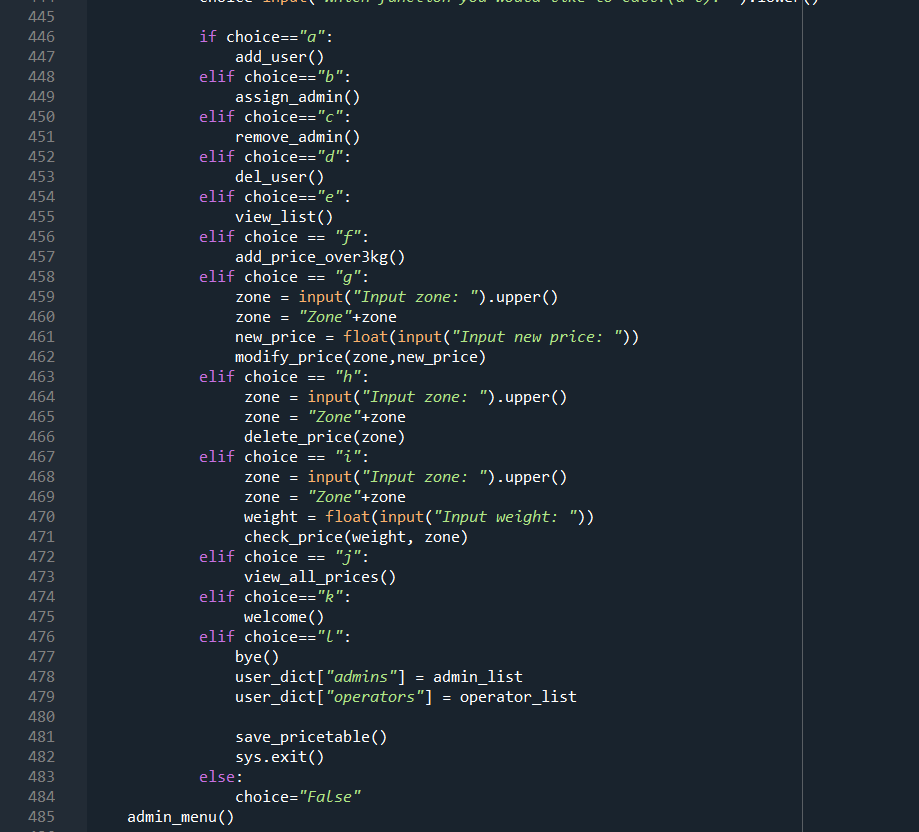
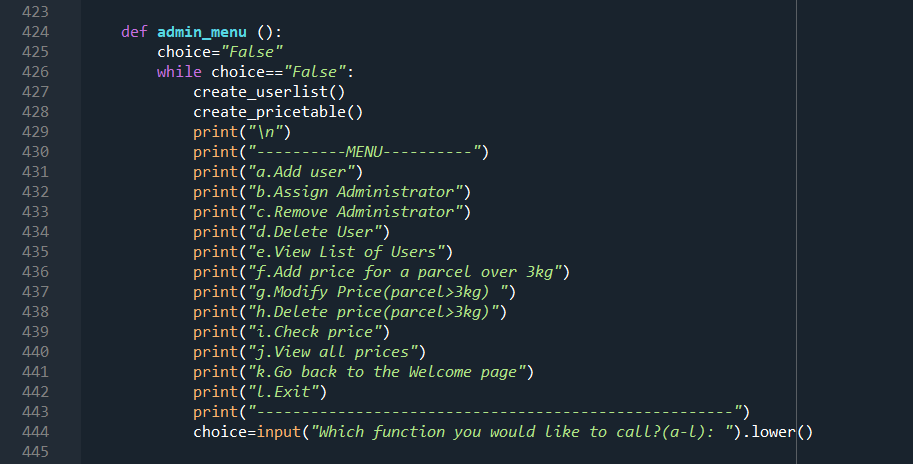
The provided Python code snippet defines a function, view\_all\_prices(), which allows the user (presumably an administrator) to view all prices stored in the pricetable dictionary. The function iterates through the dictionary, displaying the destination (e.g., "over3kg") and corresponding prices for different weights and delivery zones. After displaying the prices, it returns to the administrator menu.

Variables used in the code snippet in Figure 15:

|  |  |
| --- | --- |
| Variables | Description |
| destination | Key representing the destination category in pricetable (e.g., "over3kg"). |
| prices | Sub-dictionary containing prices for different weights in the specified destination. |
| price | Price associated with the specified weight in the prices sub-dictionary. |
| pricetable | Dictionary storing prices for different weights and delivery zones. |
| weight | Key representing the weight category in prices. |

Code snippet justification in Figure 15:

|  |  |
| --- | --- |
| Line number | Justification |
| 414-419 | Iterates through the pricetable dictionary, printing each destination category along with the associated prices for different weights and delivery zones. |
| 420 | Calls admin\_menu() to return to the administrator menu after displaying all prices. |



*Figure 16: Code snippet of administrator menu*

The provided code snippet defines the main administrator menu, encapsulated in the function admin\_menu(). This menu serves as the central control hub for various administrative actions, offering a list of options to perform functions such as user management, administrator assignment/removal, and price management for parcels over 3 kilograms. The menu provides a loop for continuous user interaction until an exit command is issued.

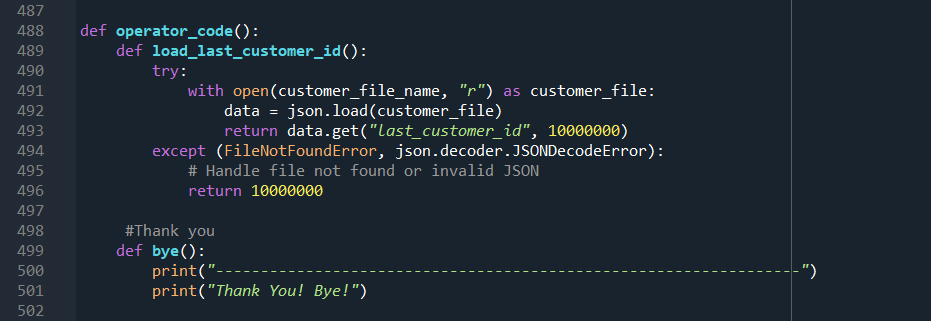
Variables used in the code snippet in Figure 16:

|  |  |
| --- | --- |
| Variables | Description |
| choice | User input indicating the chosen function in the menu. |
| user\_dict | Dictionary storing user data, including administrators and operators. |
| admin\_list | List of administrator credentials stored in user\_dict. |
| operator\_list | List of operator credentials stored in user\_dict. |

Code snippet justification in Figure 16:

|  |  |
| --- | --- |
| Line number | Justification |
| 424-443 | Definition of the admin\_menu() function, which serves as the main administrator menu. The menu provides options for various administrative functions, each linked to a specific function defined in the script. The user is prompted to make a choice, and the corresponding function is called based on the input. The menu continues to loop until an exit command is given. |
| 444 | Calls welcome() function to return to the welcome page. |
| 446-473 | Calls various functions based on user input, such as add\_user(), assign\_admin(), remove\_admin(), del\_user(), view\_list(), add\_price\_over3kg(), modify\_price(), delete\_price(), check\_price(), view\_all\_prices(). Each function corresponds to a specific action within the administrator menu. |
| 474-482 | Conditional block to handle the exit option (l). Prints a goodbye message, updates user data and price table, and exits the script using sys.exit(). |
| 483-485 | Continues the loop if an invalid choice is made. |

**Operator**

*Figure 17: Code snippet of load last customer id*

Variables used in the code snippet in Figure 17:

|  |  |
| --- | --- |
| Variables | Description |
| customer\_file\_name | The name of the file containing customer data |

Code snippet justification in Figure 17:

|  |  |
| --- | --- |
| Line number | Justification |
| 488 | Definition of the operator\_code function, which contains all functions for operator part |
| 489-496 | Definition of the load data function which loads the last recorded customer ID from a file named "customer.txt |
| 499-501 | Definition of bye() function. This function prints a farewell message. |

*Figure 18: Code snippet of load data function*

Variables used in the code snippet in Figure 18:

|  |  |
| --- | --- |
| Variables | Description |
| customers | A global variable to store customer data. |
| last\_customer\_id: | A global variable to store the last customer ID. |
| last\_parcel\_num | A global variable to store the last parcel number. |
| parcels | A global variable to store parcel data. |
| customer\_file\_name | The file name for customer data. |
| parcel\_file\_name | The file name for parcel data. |

Code snippet justification in Figure 18:

|  |  |
| --- | --- |
| Line number | Justification |
| 505 | This line defines the load\_data function, which is responsible for loading customer and parcel data from files. |
| 507-508 | These lines attempt to open the customer data file (customer\_file\_name) for reading. |
| 509-514 | This block tries to load the data from the customer file as JSON. If the file is corrupt or doesn't exist, it sets data to a dictionary with empty customer list and default last\_customer\_id. |
| 515-516 | These lines check if the loaded data is a dictionary with a "customers" key. If so, it extracts the customer list and last\_customer\_id from the data. |
| 517-518 | If the data is invalid or missing the "customers" key, this block initializes an empty customer list. |
| 519 | This line sets the last\_customer\_id to a default value if it wasn't present in the loaded data. |
| 520-524 | This block handles the case where the customer data file doesn't exist. It creates an empty customer list and default last\_customer\_id, then calls save\_data to create the file with initial data. |
| 525-526 | These lines attempt to open the parcel data file (parcel\_file\_name) for reading. |
| 527-529 | This block tries to load the data from the parcel file as JSON. If the file is corrupt or doesn't exist, it sets data to a list with default values for last\_parcel\_num and parcels. |
| 531-532 | These lines check if the loaded data is a list with two elements. If so, it extracts the values for last\_parcel\_num and parcels. |
| 533-534 | If the data is invalid or missing elements, this block sets last\_parcel\_num to a default value. |
| 535-537 | This block handles the case where the parcel data file doesn't exist. It initializes an empty parcels list and calls save\_data to create the file with initial data. |
| 540-546 | This block is the fallback case if any of the data files are missing. It initializes empty customer and parcel lists, sets default values for IDs, and then calls save\_data to create the files with initial data. |

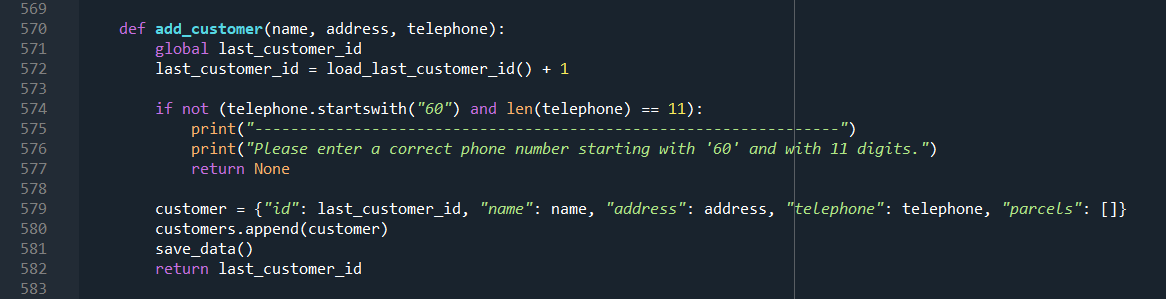
*Figure 19: Code snippet of saving data function*

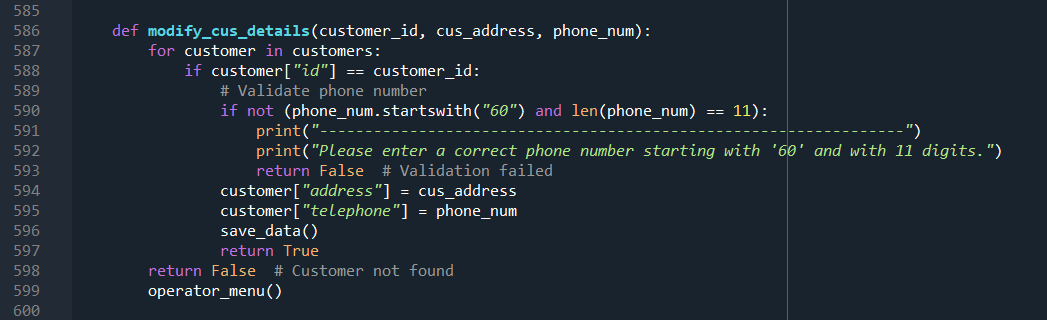
Variables used in the code snippet in Figure 19:

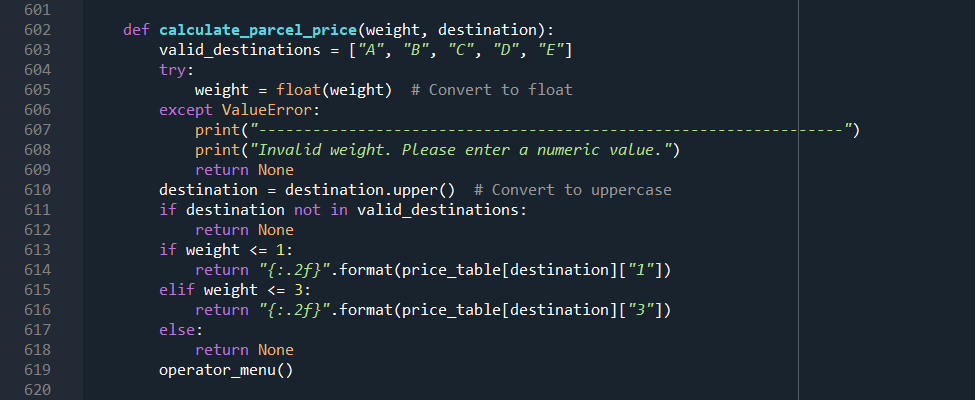
|  |  |
| --- | --- |
| Variables | Description |
| customer\_file\_name | File name for storing customer data. |
| parcel\_file\_name | File name for storing parcel data. |
| last\_customer\_id | Last used customer ID. |
| customers | List of customer data. |
| last\_parcel\_num | Last used parcel number. |
| parcels | List of parcel data. |

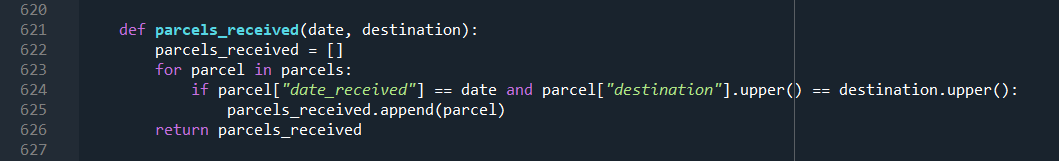
Code snippet justification in Figure 19:

|  |  |
| --- | --- |
| Line number | Justification |
| 548-553 | Employs a try block to attempt opening the database connection, ensuring proper resource cleanup with nested with statements in case of errors. |
| 555-558 | Writes the loaded customer data (including the list of customers) to the database table named customers. |
| 559-560 | Writes the loaded parcel data (including the list of parcels) to the database table named parcels. |
| 561-562 | Catches potential database errors within the except block, logging them for future investigation and troubleshooting. |
| 564-566 | Closes the database connection if it was successfully opened in the try block. |

*Figure 20: Code snippet of adding customer function*

*Figure 21: Code snippet of modifying customer details*

*Figure 22: Code snippet of calculating parcel price*

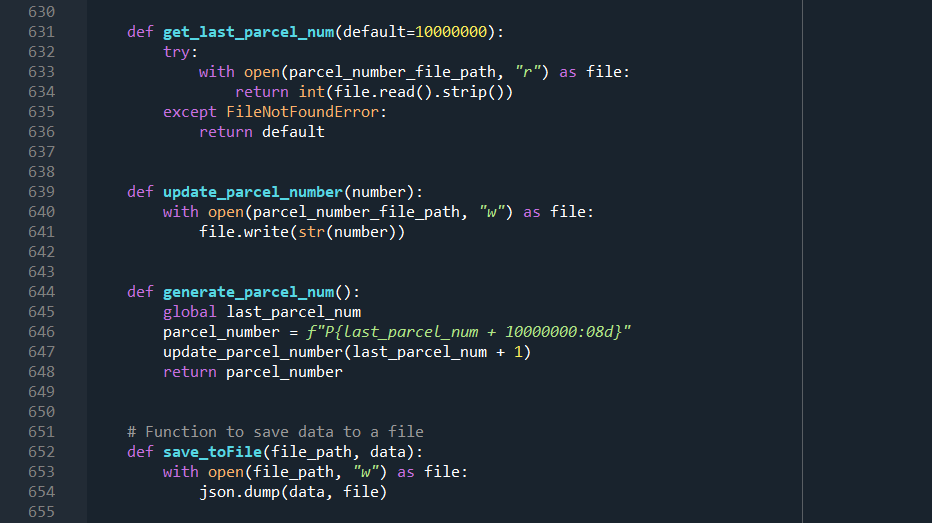
*Figure 23: Code snippet of received parcel's function*

Variables used in the code snippet in Figure 20-23:

|  |  |
| --- | --- |
| Variables | Description |
| last\_customer\_id | The last used customer ID. |
| customers | List of customer data. |
| last\_parcel\_num | Last used parcel number. |
| parcels | List of parcel data. |
| price\_table | Dictionary storing parcel prices based on weight and destination. |
| date\_received | Date when a parcel is received. |
| valid\_destinations | List of valid destination codes (A, B, C, D, E). |

Code snippet justification in Figure 20-23:

|  |  |
| --- | --- |
| Line number | Justification |
| 570-582 | Definition of the add\_customer() function. Adds a new customer with the provided details, validates the telephone number, and saves the updated data. Returns the ID of the newly added customer. |
| 586-599 | Definition of the modify\_cus\_details() function. Modifies the address and phone number of a customer identified by their ID. Validates the phone number format and saves the updated data. Returns a boolean indicating the success or failure of the modification. |
| 602-619 | Definition of the calculate\_parcel\_price() function. Calculates the price of a parcel based on its weight and destination. Validates the weight as a numeric value and the destination as a valid code. Returns the calculated price or None if validation fails. |
| 621-626 | Definition of the parcels\_received() function. Retrieves a list of parcels received on a specific date and destination by comparing with existing parcels in the system. Returns the list of matching parcels. |

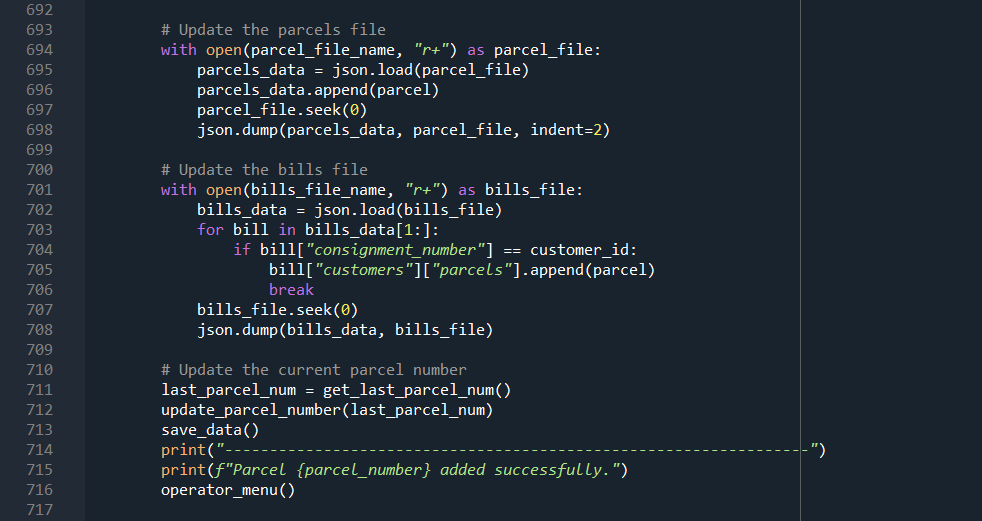
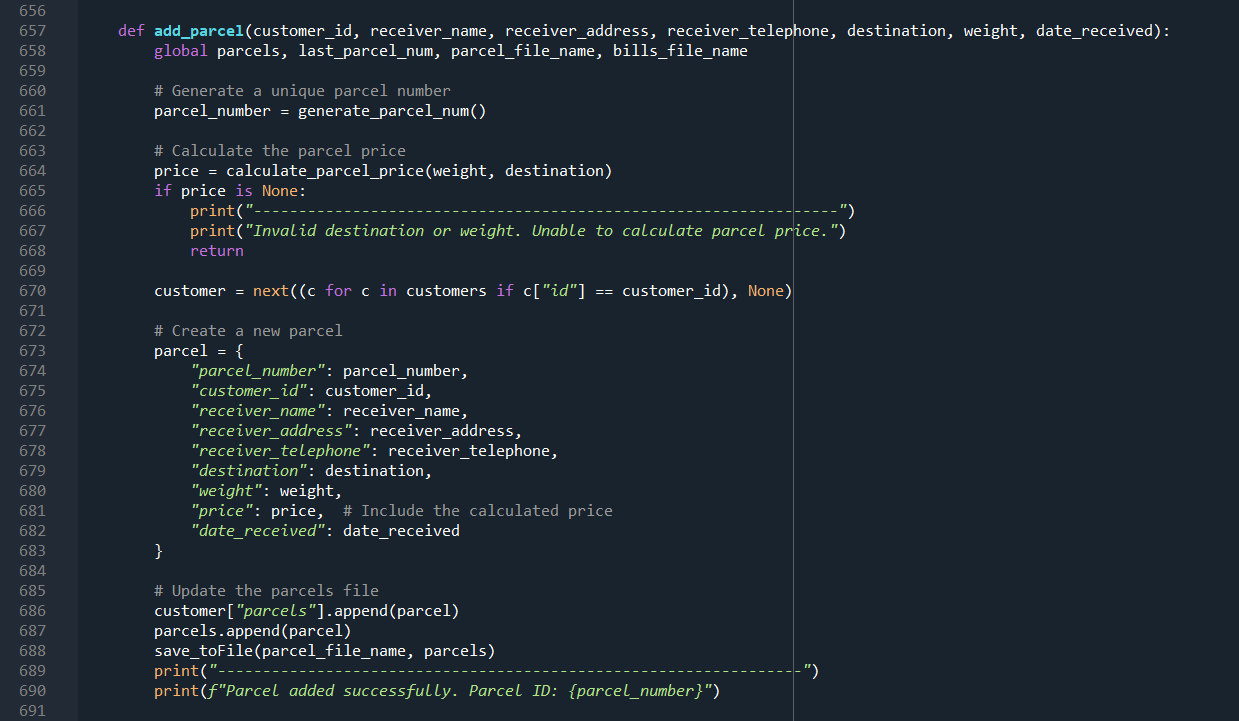
*Figure 24: Code snippet of parcel number functions*

Variables used in the code snippet in Figure 24:

|  |  |
| --- | --- |
| Variables | Description |
| last\_parcel\_num | Last used parcel number. |
| parcel\_number\_file\_path | File path for storing the last used parcel number. |

Code snippet justification in Figure 24:

|  |  |
| --- | --- |
| Line number | Justification |
| 631-636 | Definition of the get\_last\_parcel\_num() function. Retrieves the last used parcel number from a file. If the file is not found, returns a default value (10000000). |
| 639-641 | Definition of the update\_parcel\_number() function. Updates the last used parcel number in the file. |
| 644-648 | Definition of the generate\_parcel\_num() function. Generates a new parcel number based on the last used parcel number, updates the last used parcel number in the file, and returns the generated parcel number. |
| 652-654 | Definition of the save\_toFile() function. Saves data to a specified file using the JSON format. |

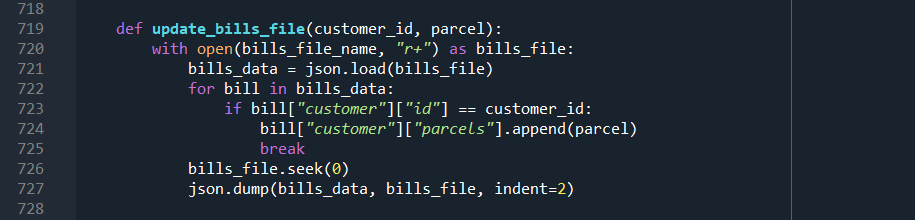
*Figure 25: Code snippet of parcel addition function*

Variables used in the code snippet in Figure 25:

|  |  |
| --- | --- |
| Variables | Description |
| parcels | List of parcel data. |
| last\_parcel\_num | Last used parcel number. |
| parcel\_file\_name | File name for storing parcel data. |
| bills\_file\_name | File name for storing bills data. |
| customers | List of customer data. |
| customer\_id | Customer ID associated with the parcel. |
| receiver\_name | Name of the parcel receiver. |
| receiver\_address | Address of the parcel receiver. |
| receiver\_telephone | Telephone number of the parcel receiver. |
| destination | Destination code for the parcel. |
| weight | Weight of the parcel. |
| date\_received | Date when the parcel is received. |

Code snippet justification in Figure 25:

|  |  |
| --- | --- |
| Line number | Justification |
| 657-690 | Definition of the add\_parcel() function. Generates a unique parcel number, calculates the parcel price, retrieves customer information, creates a new parcel, and updates the parcels file. |
| 693-715 | Updates the parcels file, bills file, and current parcel number. Saves the data and prints success messages. |
| 694-697 | Reads the existing data from the parcels and bills files, appends the new parcel information, and updates the files. |
| 711-713 | Updates the current parcel number and saves the data. |
| 715 | Prints a success message and navigates back to the operator menu. |

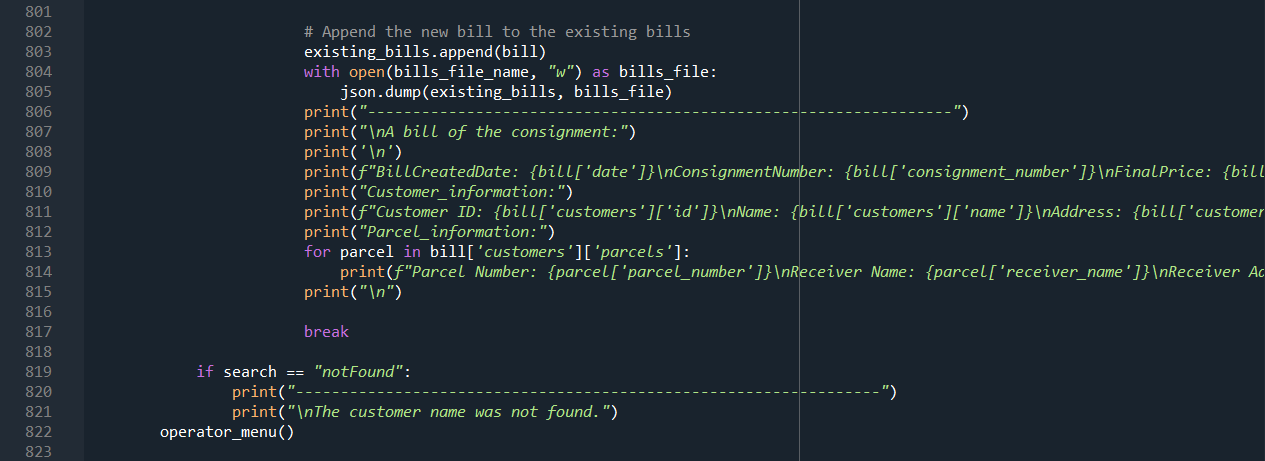
*Figure 26: Code snippet of bills file update function*

Variables used in the code snippet in Figure 26:

|  |  |
| --- | --- |
| Variables | Description |
| bills\_file\_name | File name for storing bills data. |
| customer\_id | Customer ID associated with the parcel. |
| parcel | Information of the newly added parcel. |

Code snippet justification in Figure 26:

|  |  |
| --- | --- |
| Line number | Justification |
| 719-727 | Definition of the update\_bills\_file() function. Opens the bills file, reads the existing data, locates the customer based on the customer ID, appends the parcel information, and writes the updated data back to the file. |
| 726 | Seeks to the beginning of the bills file. |
| 727 | Writes the updated bills data back to the file with indentation. |

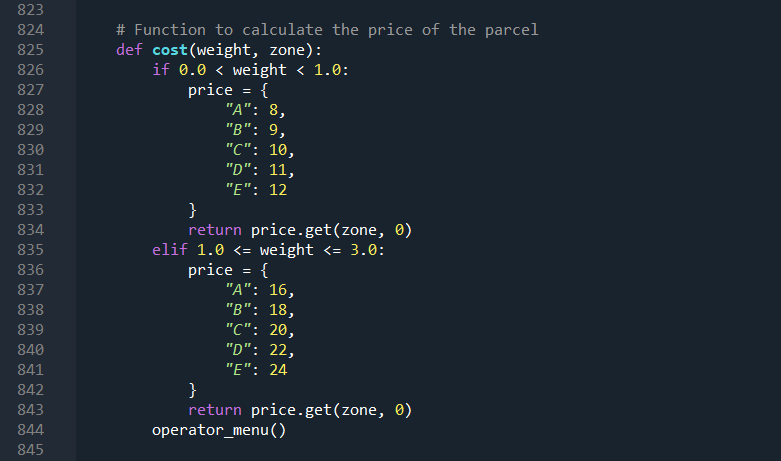
*Figure 27: Code snippet of bill creation function*

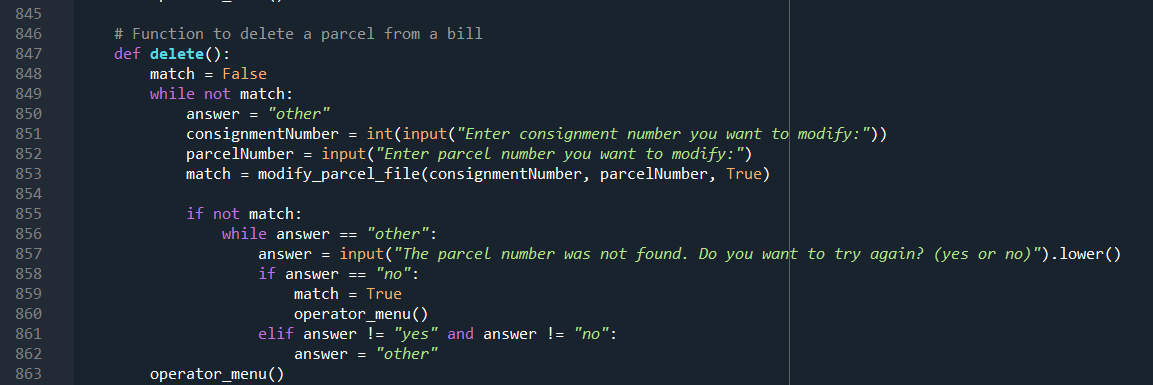
Variables used in the code snippet in Figure 27:

|  |  |
| --- | --- |
| Variables | Description |
| cus\_name | Customer name for whom the bill is created. |
| date | Today's date for the bill creation. |
| con\_num | Consignment number for the bill. |
| final\_Price | Final price of the bill. |
| match | Flag indicating the validity of weight and zone information. |
| parcel | Information about a specific parcel. |
| price | Calculated price for a parcel based on weight and zone. |
| existing\_bills | List containing existing bills data. |
| bill | Information about the created bill. |

Code snippet justification in Figure 27:

|  |  |
| --- | --- |
| Line number | Justification |
| 732-735 | Definition of the **create\_bill()** function. Prompts the user to enter the customer's name and loads customer data from the file. |
| 737-781 | Iterates through each customer and their parcels. Calculates the price for each parcel, updates customer and parcel files, and handles invalid weight or zone information. |
| 783-818 | Calculates the final price with an 8% tax, creates a bill entry, appends it to the existing bills data, prints the details of the created bill, and writes the updated bills data back to the file. |
| 820-822 | Prints an error message if the customer name is not found and navigates back to the operator menu. |

*Figure 28: Code snippet of cost function*

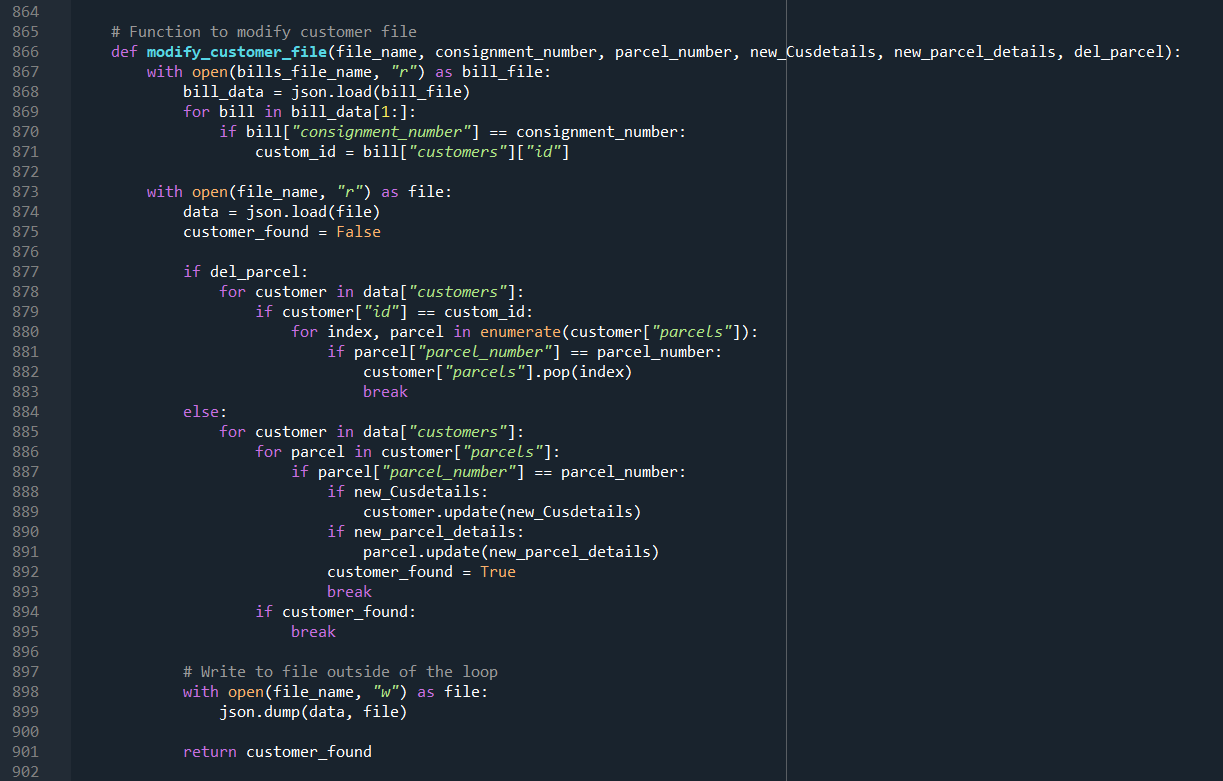
*Figure 29: Code snippet of delete function*

Variables used in the code snippet in Figure 28, 29:

|  |  |
| --- | --- |
| Variables | Description |
| weight | Weight of the parcel for cost calculation. |
| zone | Destination zone of the parcel for cost calculation. |
| price | Calculated price for shipping the parcel. |
| match | Boolean indicating whether a match was found. |
| answer | User input to determine whether to try again. |
| consignmentNumber | Consignment number of the bill to modify. |
| parcelNumber | Parcel number to modify or delete. |

Code snippet justification in Figure 28, 29:

|  |  |
| --- | --- |
| Line number | Justification |
| 825-854 | Definition of the **cost(weight, zone)** function. Calculates the cost of shipping a parcel based on weight and destination zone, using predefined price tables. |
| 856-862 | Definition of the **delete()** function. Prompts the user to enter consignment and parcel numbers, calls the **modify\_parcel\_file()** function to delete the specified parcel, and handles cases where the entered parcel number is not found. |

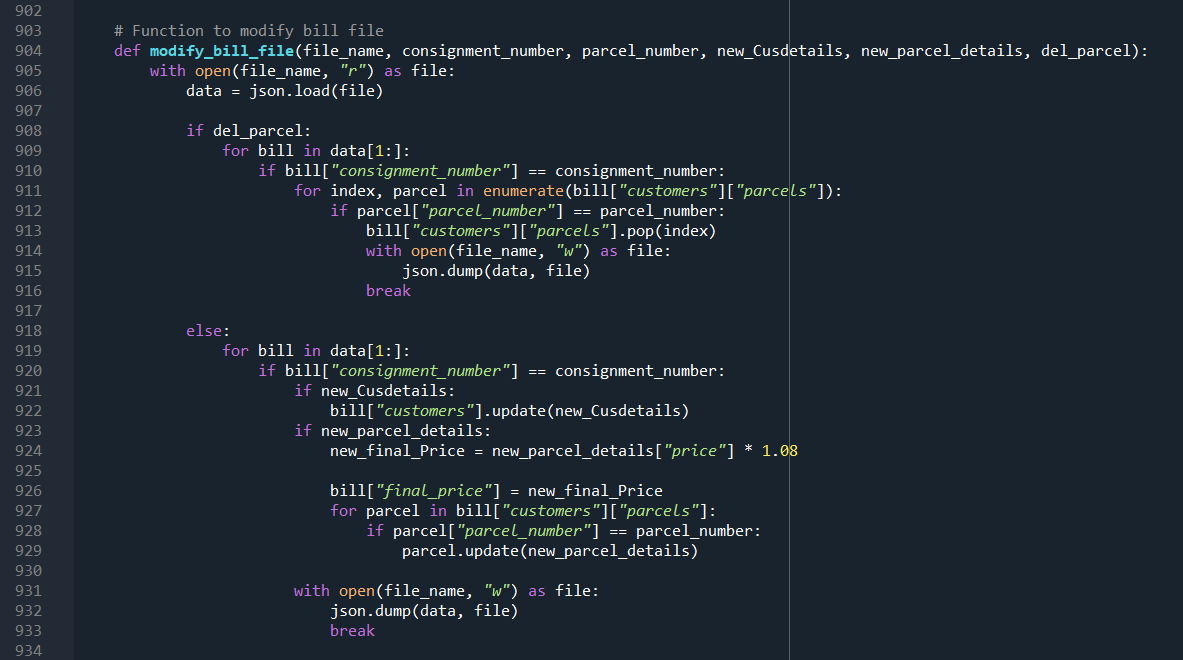
*Figure 30: Code snippet of modifying customer file*

Variables used in the code snippet in Figure 30:

|  |  |
| --- | --- |
| Variables | Description |
| file\_name | Name of the file to modify (customer or parcel file). |
| consignment\_number | Consignment number used to identify the customer. |
| parcel\_number | Parcel number used to identify the specific parcel. |
| new\_Cusdetails | New customer details to be updated. |
| new\_parcel\_details | New parcel details to be updated. |
| del\_parcel | Boolean flag indicating whether to delete a parcel. |
| bill\_data | Data extracted from the bills file. |
| custom\_id | Customer ID associated with the consignment number. |
| data | Customer data read from the customer file. |
| customer\_found | Boolean indicating whether the customer was found. |

Code snippet justification in Figure 30:

|  |  |
| --- | --- |
| Line number | Justification |
| 866-869 | Definition of the modify\_customer\_file() function. Takes input parameters for file name, consignment number, parcel number, new customer details, new parcel details, and a flag to delete a parcel. |
| 871-877 | Reads the bills file to extract the customer ID associated with the given consignment number. |
| 879-896 | Reads the customer file, iterates through customer data, and modifies the customer or parcel details based on the provided information. |
| 898-900 | Writes the updated data back to the customer file and returns a boolean indicating whether the customer was found. |

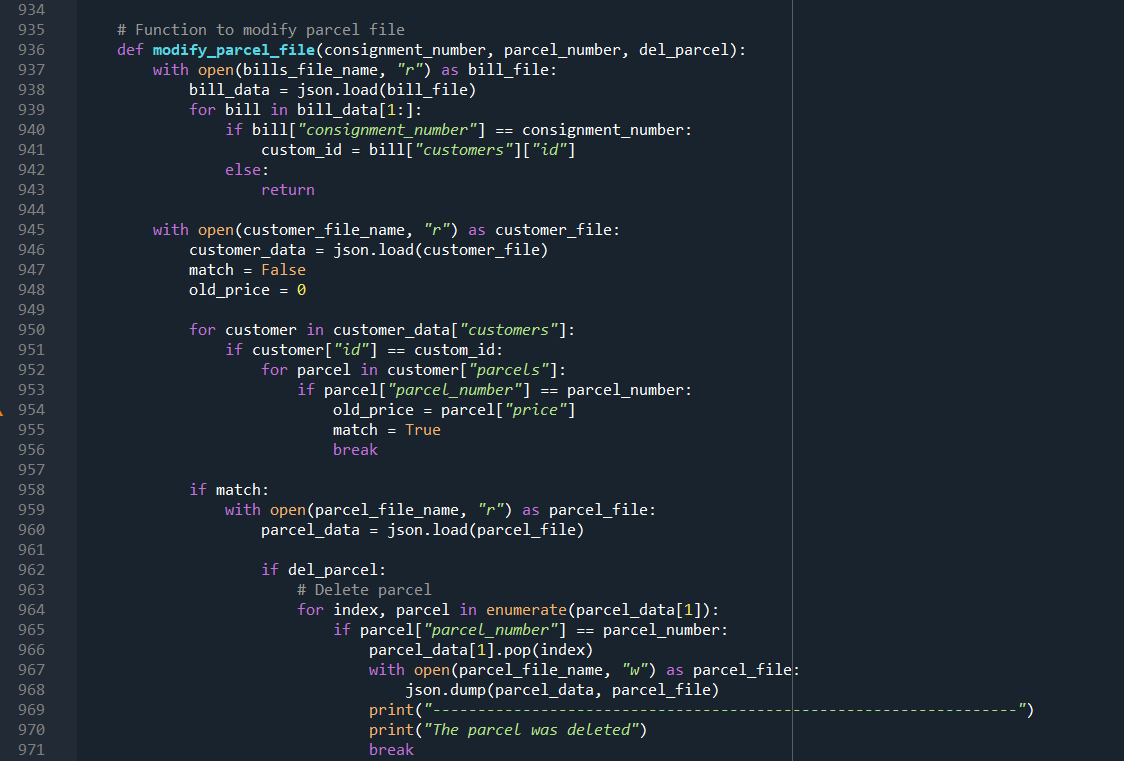
*Figure 31: Code snippet of*

Variables used in the code snippet in Figure 31:

|  |  |
| --- | --- |
| Variables | Description |
| file\_name | A string variable representing the name of the file to be modified. |
| consignment\_number | An integer representing the consignment number of the bill to be modified. |
| parcel\_number | A string representing the parcel number within the consignment to be modified. |
| new\_Cusdetails | A dictionary containing new customer details to be updated in the bill. |
| new\_parcel\_details | A dictionary containing new parcel details to be updated in the bill. |
| del\_parcel | A boolean indicating whether to delete the specified parcel from the bill. |
| file | A file object representing the opened file in read mode. |
| data | A dictionary representing the data loaded from the file to be modified. |
| bill | A dictionary representing a specific bill within the data. |
| index | An integer used to iterate over the parcels within the bill. |
| parcel | A dictionary representing a specific parcel within the bill. |
| new\_final\_Price | A float representing the new final price of the bill after modifications. |

Code snippet justification in Figure 31:

|  |  |
| --- | --- |
| Line number | Justification |
| 904-914 | Definition of the modify\_bill\_file() function. Takes input parameters for the file name, consignment number, parcel number, new customer details, new parcel details, and a flag to delete the parcel. |
| 916-930 | Depending on user input, either deletes the parcel from the bill or updates the customer and parcel details, including the final price. |
| 932-933 | Writes the modified billing information back to the file. |

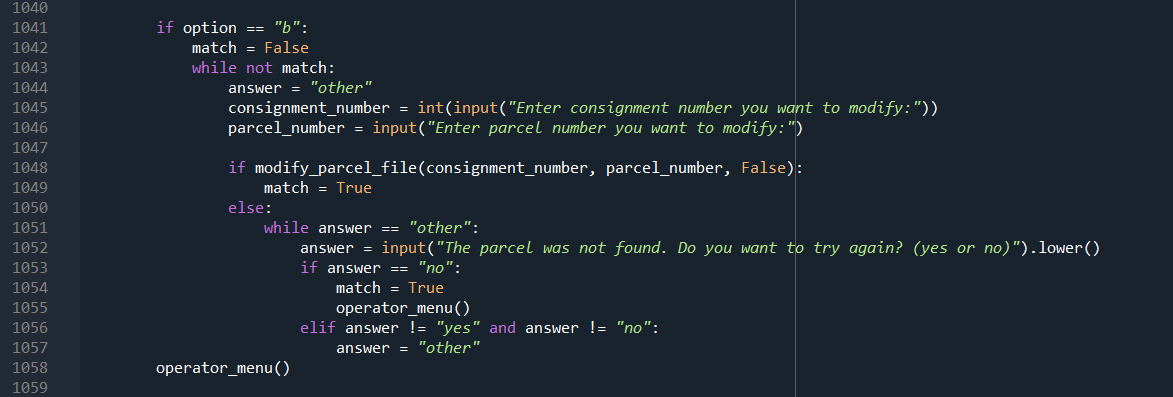
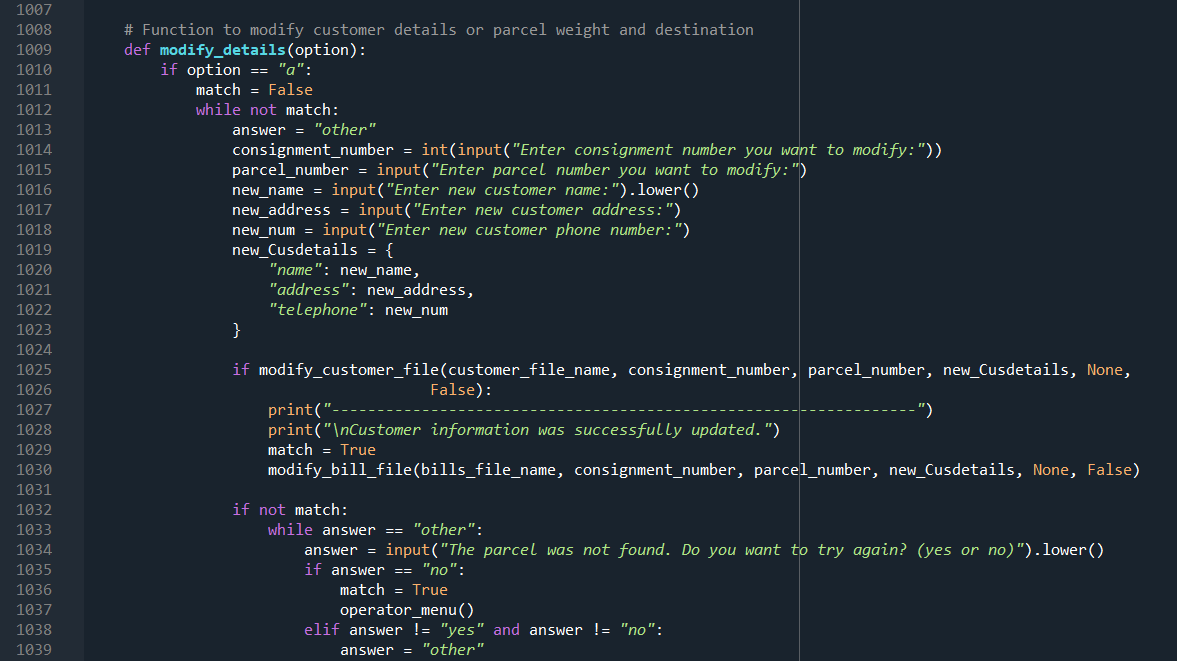
*Figure 32: Code snippet of modifying parcel file*

Variables used in the code snippet in Figure 32:

|  |  |
| --- | --- |
| Variables | Description |
| consignment\_number | An integer representing the consignment number of the parcel to be modified. |
| parcel\_number | A string representing the parcel number to be modified. |
| del\_parcel | A boolean indicating whether to delete the specified parcel. |
| bill\_file | A string representing the name of the file storing bill information. |
| customer\_file | A string representing the name of the file storing customer information. |
| parcel\_file | A string representing the name of the file storing parcel information. |
| bill\_data | A dictionary representing the data loaded from the bill file. |
| customer\_data | A dictionary representing the data loaded from the customer file. |
| parcel\_data | A dictionary representing the data loaded from the parcel file. |
| custom\_id | An integer representing the customer ID associated with the specified parcel. |
| match | A boolean indicating whether a match was found for the specified parcel. |
| old\_price | A float representing the original price of the specified parcel. |

Code snippet justification in Figure 32:

|  |  |
| --- | --- |
| Line number | Justification |
| 936-938 | Definition of the modify\_parcel\_file() function. Takes input parameters for consignment number, parcel number, and a flag to delete the parcel. |
| 940-946 | Reads the bills file to extract the customer ID associated with the given consignment number. |
| 948-972 | Reads the customer file and iterates through customer data to find the specified parcel and retrieve its original price. |
| 974-988 | Depending on user input, either deletes the parcel or prompts for new zone and weight information to update the parcel. |
| 990-994 | Updates the customer and bill files with the modified parcel information. |
| 996-1002 | Writes the updated parcel data back to the parcel file. |
| 1004-1006 | Returns a boolean indicating whether the specified parcel was found and modified. |

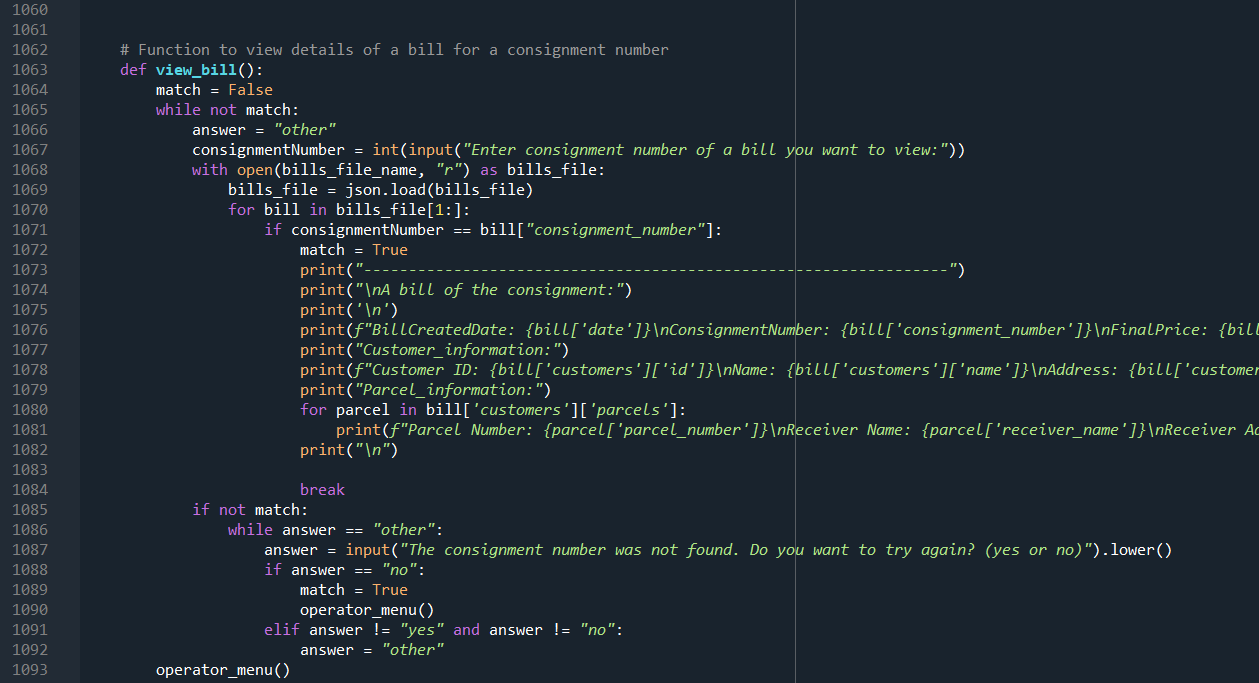
*Figure 33: Code snippet of modifying details*

Variables used in the code snippet in Figure 33:

|  |  |
| --- | --- |
| Variables | Description |
| option | A string variable representing the user's choice ('a' or 'b') for modifying details. |
| match | A boolean variable initialized to False. It is used to check whether a consignment or parcel matches any existing records. If it finds a match, it is set to True. |
| answer | A string variable initialized to "other". It is used in the loop to determine if the consignment or parcel was not found and if the user wants to try again. |
| consignment\_number | An integer variable used to store the consignment number input by the user. |
| parcel\_number | A string variable used to store the parcel number input by the user. |
| new\_name | A string variable used to store the new customer name input by the user. |
| new\_address | A string variable used to store the new customer address input by the user. |
| new\_num | A string variable used to store the new customer phone number input by the user. |
| new\_Cusdetails | A dictionary storing the new customer details. It includes keys for "name," "address," and "telephone." |
| customer\_file\_name | A variable referencing the name of the file storing customer information. It's assumed to be defined elsewhere in the program. |
| bills\_file\_name | A variable referencing the name of the file storing bill information. It's assumed to be defined elsewhere in the program. |
| operator\_menu() | A function that is called at the end of the modify\_details() function to return to the operator menu. |

Code snippet justification in Figure 33:

|  |  |
| --- | --- |
| Line number | Justification |
| 1009-1065 | Definition of the modify\_details() function. Takes the user's option as input and allows the operator to either modify customer details or delete a parcel. |
| 1011-1063 | Option "a" - Modifies customer details. Prompts the user for consignment number, parcel number, and new customer details. Calls modify\_customer\_file() to update data and prints success messages. |
| 1065 | Option "b" - Deletes a parcel. Prompts the user for consignment number and parcel number. Calls modify\_parcel\_file() to delete the parcel and prints success messages. |
| 1067-1069 | Returns to the operator menu. |

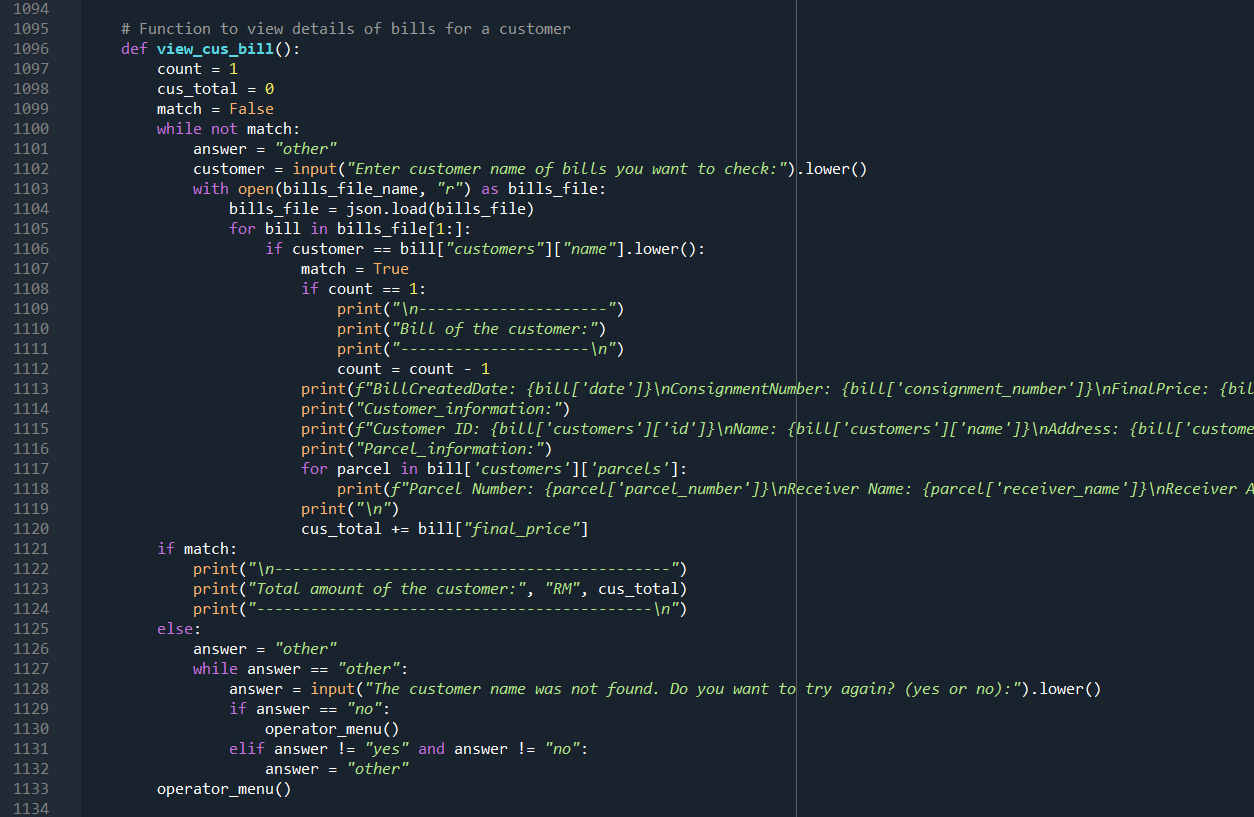
*Figure 34: Code snippet of view bill function*

Variables used in the code snippet in Figure 34:

|  |  |
| --- | --- |
| Variables | Description |
| match | A boolean variable initialized to False. It is used to check whether a consignment number matches any existing bill. If it finds a match, it is set to True. |
| answer | A string variable initialized to "other". It is used in the first loop to determine if the consignment number was not found and if the user wants to try again. |
| consignmentNumber | An integer variable used to store the consignment number input by the user. |
| bills\_file\_name | A variable referencing the name of the file storing bill information. It's assumed to be defined elsewhere in the program. |

Code snippet justification in Figure 34:

|  |  |
| --- | --- |
| Line number | Justification |
| 1063-1093 | Definition of the view\_bill() function. Takes user input for the consignment number and iterates through the bills file to find and display details of the corresponding bill. Handles cases where the consignment number is not found and prompts the user to try again or exit. |

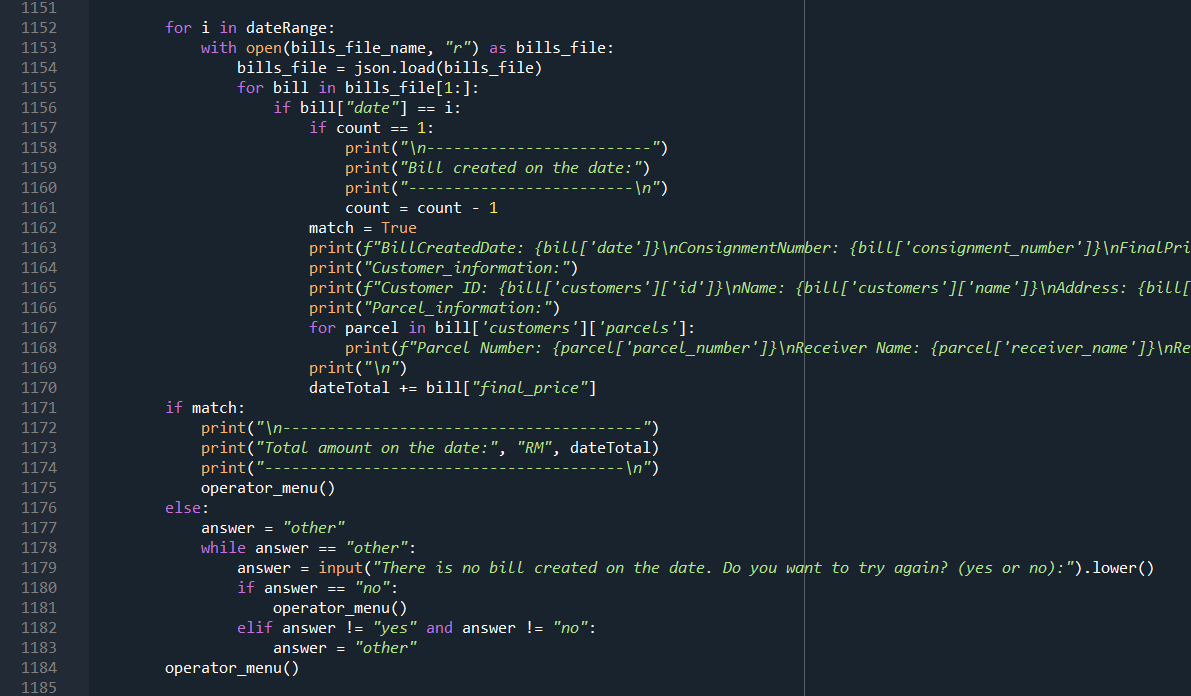
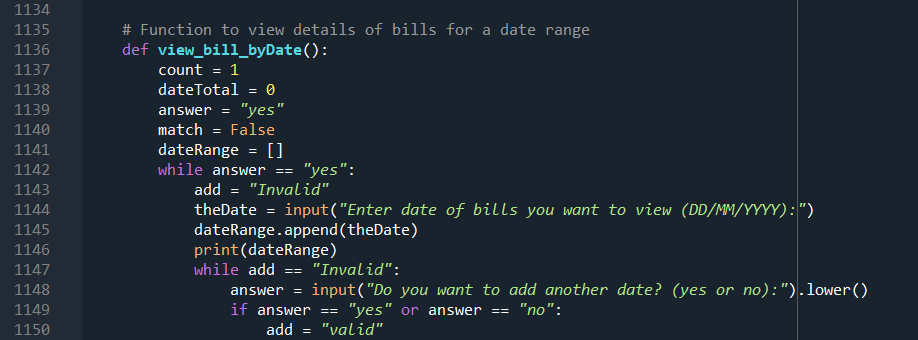
*Figure 35: Code snippet of view customer bill function*

Variables used in the code snippet in Figure 35:

|  |  |  |
| --- | --- | --- |
| Variables | Description | |
| match | | A boolean variable initialized to False. It is used to check whether a consignment number matches any existing bill. If it finds a match, it is set to True. |
| answer | | A string variable initialized to "other". It is used in the first loop to determine if the consignment number was not found and if the user wants to try again. |
| consignmentNumber | | An integer variable used to store the consignment number input by the user. |
| bills\_file\_name | | It's referenced but not defined in the provided snippet. Assumed to be defined elsewhere in the program. |
| json | | The JSON module for loading data from a file. |

Code snippet justification in Figure 35:

|  |  |
| --- | --- |
| Line number | Justification |
| 1096-1099 | Definition of the view\_cus\_bill() function. Prompts the user to enter the customer's name for bill checking. |
| 1101-1108 | Reads the bills file and iterates through the data to find bills associated with the specified customer. |
| 1110-1131 | Prints detailed information about each bill, including customer details and parcel information. Calculates the total amount spent by the customer. |
| 1132-1133 | Provides an option to try again if the customer name is not found. |

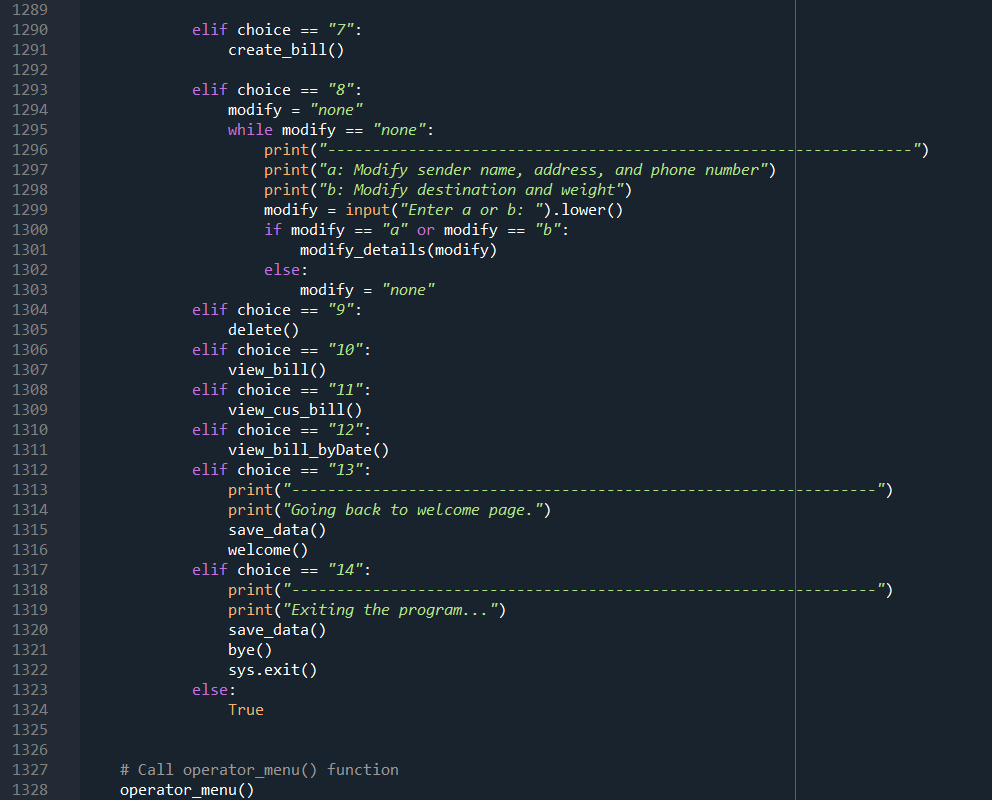
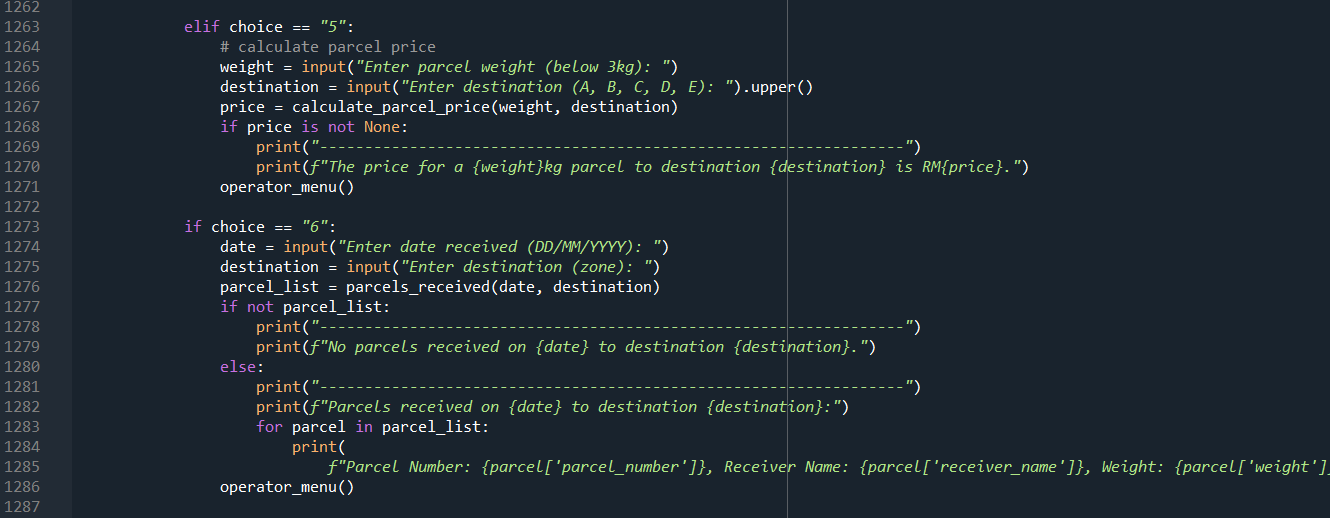
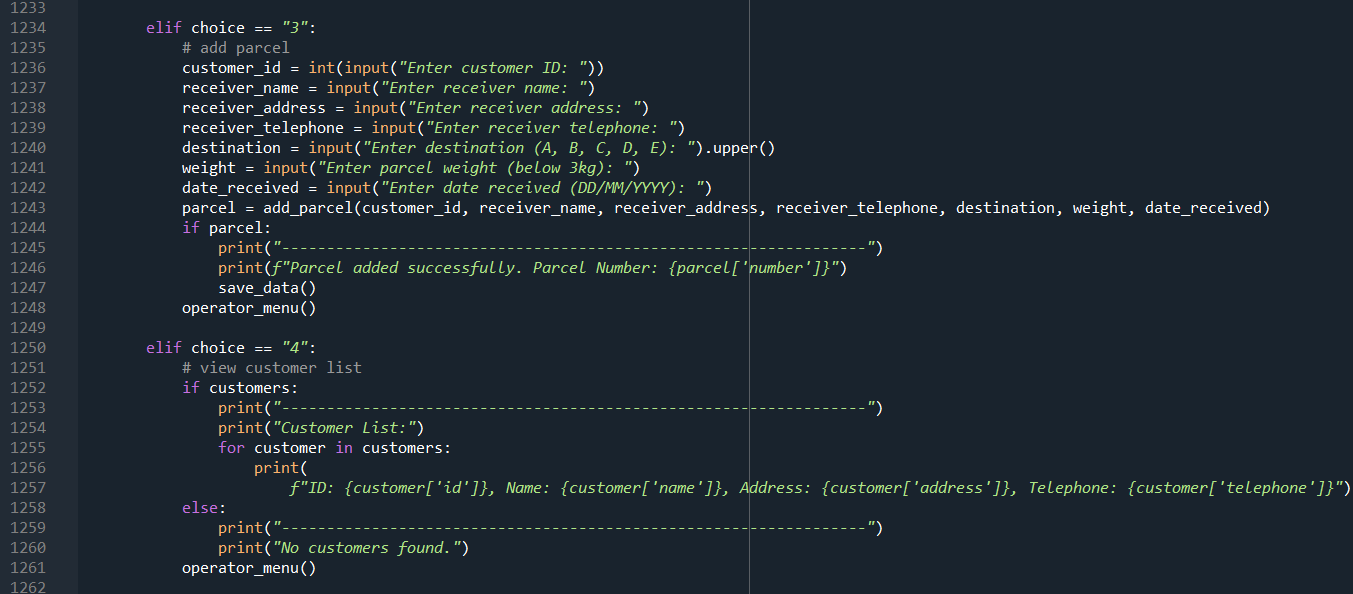
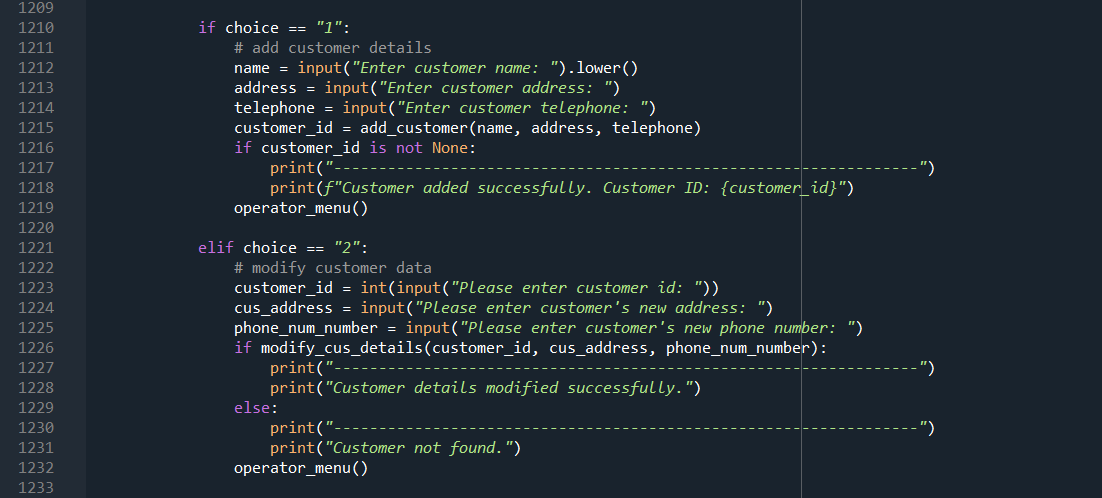
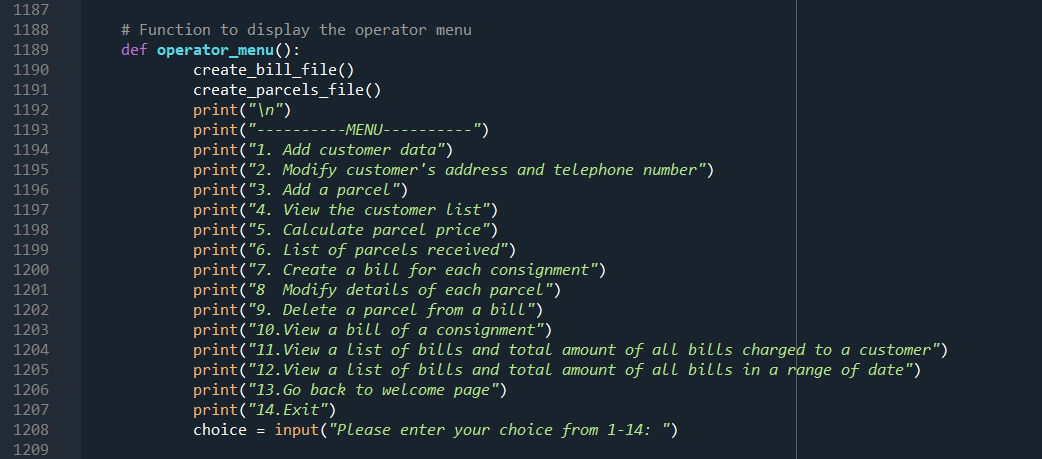
*Figure 36: Code snippet of viewing bill by date*

Variables used in the code snippet in Figure 36:

|  |  |
| --- | --- |
| Variables | Description |
| count | Keeps track of the number of times the user has entered a date. |
| dateTotal | Initialized to 0; its purpose is unclear, as it's not used in the provided code snippet. |
| answer | Initialized to "yes"; used in a loop condition to determine whether the user wants to continue entering dates. Set based on user input ("yes" or "no"). |
| match | Initialized to False; not used in the provided code snippet. Its purpose is unclear. |
| dateRange | Initialized as an empty list; used to store the dates entered by the user. |
| add | Initialized to "Invalid"; used in a nested loop condition. Set to "valid" if the user enters "yes" or "no" when prompted, determining whether to add another date. |
| theDate | Stores the user's input when prompted to enter the date of bills in the format "DD/MM/YYYY." |

Code snippet justification in Figure 36:

|  |  |
| --- | --- |
| Line number | Justification |
| 1136-1184 | Definition of the view\_bill\_byDate() function. The function interacts with the user to input one or more dates and displays detailed information for bills created on those dates. Calculates and presents the total amount for each date. |

*Figure 37: Code snippet of operator menu function*

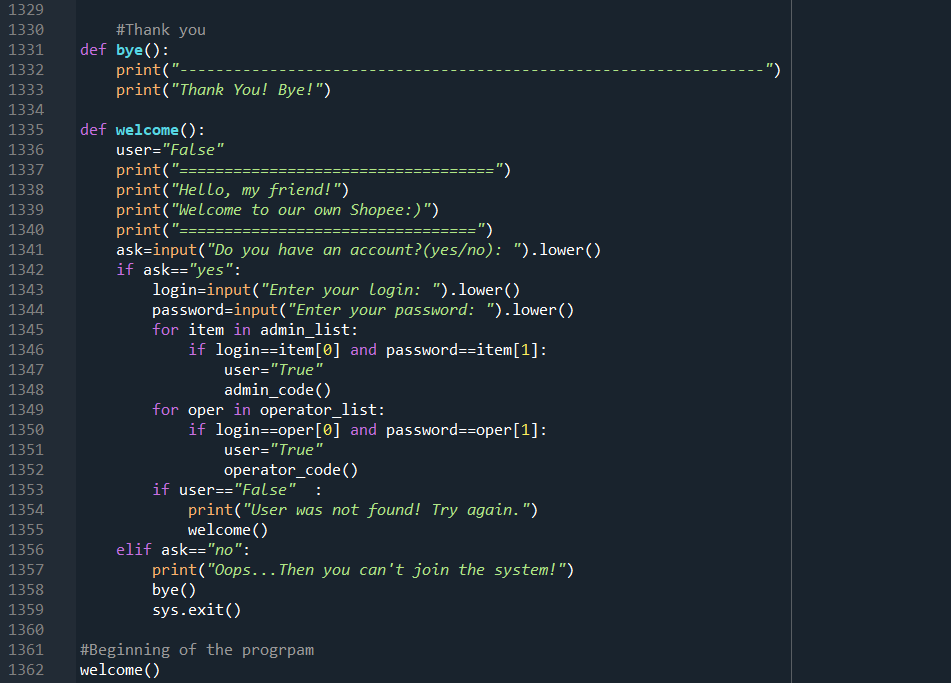
Variables used in the code snippet in Figure 37:

|  |  |  |
| --- | --- | --- |
| Variables | Description | |
| choice | | Stores the user's input, representing the chosen option from the menu (1-14). |
| name | | Used to store the customer's name when adding a new customer. |
| address | | Used to store the customer's address when adding a new customer or modifying customer details. |
| telephone | | Used to store the customer's telephone number when adding a new customer or modifying customer details. |
| customer\_id | | Stores the customer ID when adding or modifying customer details. |
| cus\_address | | Stores the new address when modifying customer details. |
| phone\_num\_number | | Stores the new phone number when modifying customer details. |
| receiver\_name | | Stores the receiver's name when adding a new parcel. |
| receiver\_address | | Stores the receiver's address when adding a new parcel. |
| receiver\_telephone | | Stores the receiver's telephone number when adding a new parcel. |
| destination | | Stores the destination when adding a new parcel or calculating parcel price. |
| weight | | Stores the weight of the parcel when adding a new parcel or calculating parcel price. |
| date\_received | | Stores the date when a parcel is received. |
| parcel | | Stores the information about a newly added parcel. |
| price | | Stores the calculated price of a parcel. |
| date | | Stores the date received when searching for parcels. |
| parcel\_list | | Stores the list of parcels received on a specific date and destination. |
| modify | | Used in the modification loop. Stores whether the modification is for sender details ('a') or destination and weight ('b'). |
| add | | Used in the date addition loop. It is set to "valid" if the user enters "yes" or "no" to add another date. |
| customers | | It's assumed to be a global variable storing information about customers. |
| bills\_file\_name | | It's referenced but not defined in the provided snippet. Assumed to be defined elsewhere in the program. |
| json | | The JSON module for loading data from a file. |

Code snippet justification in Figure 37:

|  |  |
| --- | --- |
| Line number | Justification |
| 1189-1328 | Definition of the operator\_menu() function, which serves as the central hub for operator interactions. This function is designed as an interactive menu loop, allowing operators to choose from a variety of options. Each option is implemented in a dedicated code block to ensure modular and organized code. The menu encompasses essential functionalities such as customer management, parcel handling, price calculation, and bill creation, streamlining the operator's tasks effectively. |

**Main page**

*Figure 38: Code snippet of welcome and authentication function*

The provided Python code snippet contains two functions, welcome() and bye(), which collectively handle the initial user interaction and authentication. The welcome() function prompts the user to indicate whether they have an account. If yes, it requests login credentials and checks them against the stored user data. If the login is successful, it directs the user to either the administrator or operator code functionalities. If the user does not have an account or the login is unsuccessful, appropriate messages are displayed, and the program exits. The bye() function prints a farewell message.

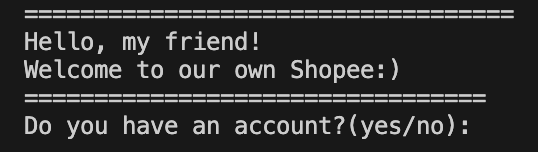
Variables used in the code snippet in Figure 38:

|  |  |
| --- | --- |
| Variables | Description |
| user | A flag indicating whether a valid user is identified ("True") or not ("False"). |
| ask | User input indicating whether they have an account ("yes" or "no"). |
| login | User input for the username during login. |
| password | User input for the password during login. |
| item | An item in the admin\_list during iteration for administrator login. |
| oper | An item in the operator\_list during iteration for operator login. |

Code snippet justification in Figure 38:

|  |  |
| --- | --- |
| Line number | Justification |
| 1335-1359 | Definition of the welcome() function. Prompts the user to indicate if they have an account. If yes, requests login credentials, checks them against stored user data, and directs the user to the appropriate functionality (administrator or operator). If no account or unsuccessful login, appropriate messages are displayed, and the program exits. |
| 1331-1333 | Definition of the bye() function. Prints a farewell message. |

# 4. Screenshots of sample input/output and explanation



Upon launching the program, the primary welcome message will be displayed to the user. This message will invite the user to interact with the program by making a choice based on whether they already have an account or not. To facilitate this interaction, the user is prompted to type either "Yes" if they already have an account, or "No" if they do not. This initial interaction is crucial as it helps the program understand the user's context and provide the most appropriate options moving forward. For instance, typing "Yes" will lead to a login process, while typing "No" will start the registration process. This feature ensures the program is user-friendly and caters to both existing users and new users effectively.

A screenshot of a computer screen

Description automatically generated

Upon initiating the program, the user will be prompted to enter 'login' to sign in to their account. The program is designed with two different roles in mind - Administrator and Operator. The range of functionalities available to the user will vary depending on the role they sign in as, allowing for a customized user experience tailored to their specific needs and responsibilities.

For instance, if the user signs in as an Administrator, they will be granted a higher level of access, which includes overseeing system operation and managing Operator roles. The Administrator role is critical for the smooth running of the program, ensuring all components work in harmony and any potential issues are promptly addressed.

On the other hand, if the user signs in as an Operator, they will be presented with a different set of tools designed to streamline their day-to-day tasks. These may include features for calculating parcel costs, managing their customer base, and other functionality pertinent to their role.

A black screen with white text

Description automatically generated

At the start, if the user write “No”, then the program will shut down. And show the message as shown in the picture above.

A screenshot of a computer program

Description automatically generated

This is the menu bar, a comprehensive tool that displays all the privileges and access rights you possess within the system. It comprises various functionalities, ranging from “a” to “l”, each providing unique capabilities to enhance your user experience.

A screen shot of a computer

Description automatically generated

The first function in our menu bar is "Adding a User." To initiate this process, the user simply needs to click on the corresponding alphabet. Upon doing so, the user will be prompted to enter a login and a password for the new user. Once these details are entered, the system will process the information. After successful verification, a confirmation message will appear, signalling the successful addition of the new user to the system. This message is not only a confirmation but also an assurance that the new user has been integrated into the system.

A black background with white text

Description automatically generated

The second, assign administrator which gives administrator to the user.

A black background with white text

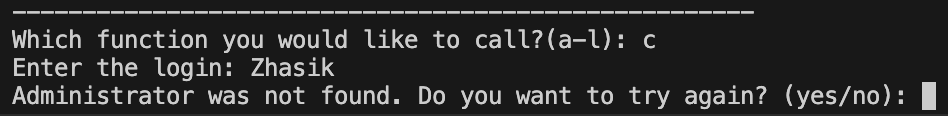
Description automatically generated

This is what happens when you enter login incorrect.

A black screen with white text

Description automatically generated

The third function removes administrator users. Before removing the administrator role, the system will prompt you for confirmation. You will be asked to type "Yes" or "No" to proceed with the removal. This extra step ensures that you do not accidentally remove administrator privileges from a user.



This is what happens when you type login wrong.

A black screen with white text

Description automatically generated

Next is delete user function. As previous functions you should enter login then confirm. If you mistype the deletion process will be cancelled.

A black screen with white text

Description automatically generated  
This is when you confirm properly. Then user will be deleted .

A black background with white text

Description automatically generated

This picture shows what happen when the user haven't found.

A screenshot of a computer program

Description automatically generated

This function shows list of user. You can choose 3 options which are all, operator, admin. In this picture I chose all users and it showed all current users.







A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

The next function allows you to add a price for a parcel that weighs over 3 kilograms. In the following 5 pictures, you can see that the user has the option to manually enter the price.

This feature enables you to specify a custom price for parcels that exceed the weight threshold of 3 kilograms. By manually inputting the price, you have greater flexibility and control over the cost calculation.

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A screen shot of a black and white screen

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

The next function allows you to modify the prices you previously added. In the following 5 pictures, the user has the option to change the prices for each zone. However, please note that the last picture demonstrates what happens when an incorrect zone is entered, resulting in an error message. In this case, if you type an incorrect zone, the system will display an error message indicating that the zone is invalid. It is important to ensure that you enter the correct zone to avoid any issues with the price modification process.

A screenshot of a computer

Description automatically generated

This function shows all price in table format.

A black screen with white text

Description automatically generated

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black background with white text

Description automatically generated

This function allows you to delete the prices that you entered in the previous function. The following five pictures demonstrate how a user can delete all prices from all zones. In the last picture, you can see the result when the user enters incorrect information, triggering an error message.

A black screen with white text

Description automatically generated

This is the price checking function. First, you enter the zone and weight of the parcel. After that, it gives you the price that you will pay. In the first picture, the user entered zone "A" and a weight of 2.9. Then, the price is shown based on the entered information. Next 4 pictures are example of checking function.

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

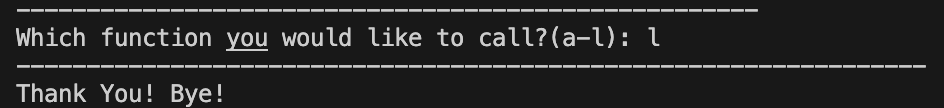
A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

Function: back to Welcome page, it returns to first function that user sigh in at the start.



This is the last function, which is the administrator exit program. The administrator can use this function to exit the program and close all operations.

Operator

A screen shot of a computer screen

Description automatically generated

Upon launching the program, the user is prompted to choose between logging in or registering. Typing "Yes" leads to login, while typing "No" starts the registration process. In the picture above the operator login typed to enter operator role.

A screen shot of a computer

Description automatically generated

The menu bar is an essential component of the program as it displays all the available functions and features. Users frequently interact with this menu bar to access various functionalities.

A screenshot of a computer

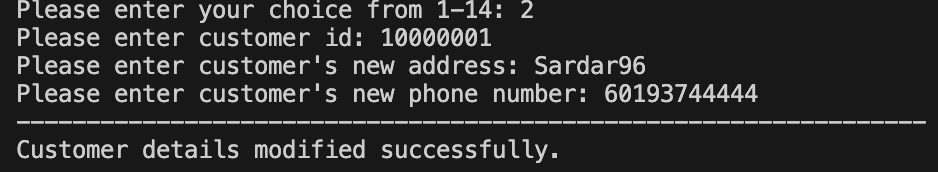
Description automatically generated

The first function is the customer registration feature. It allows users to add customers to the program by entering their name, address, and telephone number. After entering the required information, the system will automatically generate a unique ID for each customer.

A screen shot of a computer

Description automatically generated

As shown in the picture, when a user enters an invalid phone number, an error message will be displayed, reminding the user to enter a correct phone number with 11 digits. This feature prevents any potential issues that may arise from incorrect or incomplete phone numbers and ensures that the system only stores valid and reliable customer contact information.



The second function allows users to update a customer's address and phone number by entering the customer ID. If the entered customer ID is incorrect, the system will display a message stating "customer not found," similar to the example shown below.

A screen shot of a phone number

Description automatically generated

If user enter wrong customer ID, it will print customer not found.

A screenshot of a computer

Description automatically generated

This function is add parcel. It allows users to add customers to the program by entering their name, address, and telephone number. After entering the required information, the system will automatically generate a unique ID for each customer. This feature is essential for keeping track of customer details and facilitating efficient communication and management.

A computer screen with white text

Description automatically generated

If you write one of these inputs wrong then output will be invalid destination or weight.

A black background with white text

Description automatically generated

This is calculation function that calculate based on weight and destination.

A black screen with white text

Description automatically generated

This functionality helps to show list of parcels received by entering date and zone.

A black screen with white text

Description automatically generated

As you can see in this picture if there is no parcel received on specific date, then program will print it out.

A screenshot of a computer error

Description automatically generated

Also, you can modify details of each parcel by adding new customer name, address, phone number. Fist you should choose a or b based on what user want to chance then enter information.

A computer screen shot of a black screen

Description automatically generated

This function create a bill for each consignment. Shows a lot of information such as name, address, telephone, destination, weight, date and so forth.

A screenshot of a computer

Description automatically generated

If you enter one of these data then it will print out parcel was not found .

A screenshot of a computer error

Description automatically generated

This function modify details of each parcel

A screenshot of a computer error message

Description automatically generated

Its delete a parcel from a bill,.

A screenshot of a computer

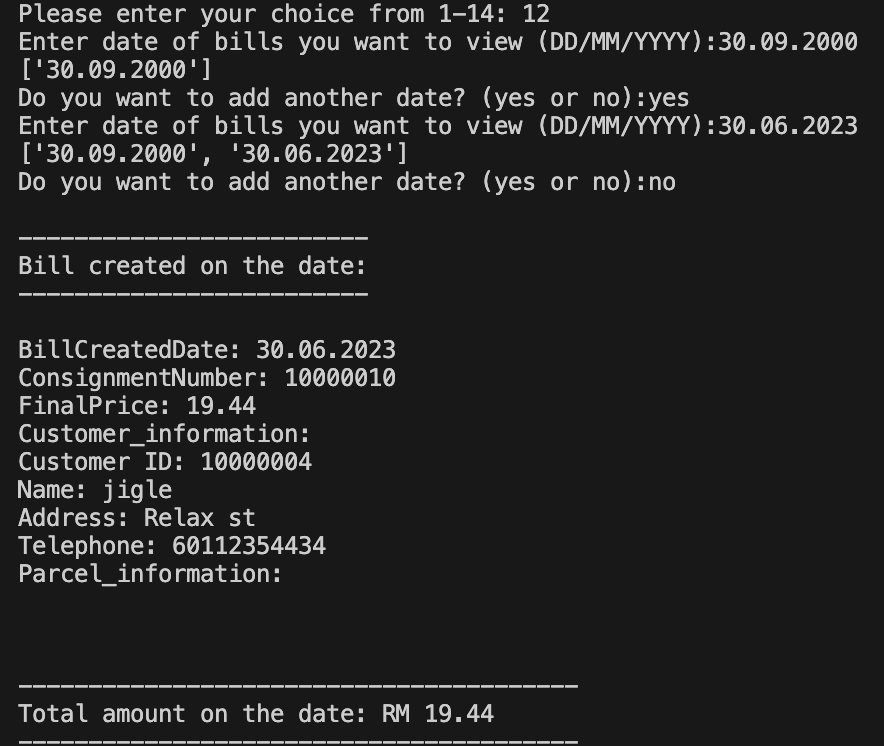
Description automatically generated

This function helps to view a bill of a consignment.

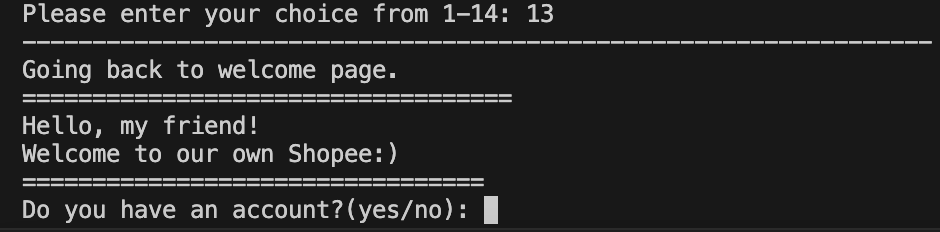
A screenshot of a computer

Description automatically generated

Function helps user to view a list of bills and total amount of all bills charged to a customer.



This function helps user to view a list of bills and total amount of all bills in a range of date.



Back to the Welcome page function - exit operator role and go to login.

A black screen with white text

Description automatically generated

As in administrator role, this function exit the program.

# 5.Conclusion

This program helps to calculate parcels price, and, in this program, we have a bunch of functionalities such as calculating price of parcel, add customers, remove users, view customers, show a list of parcels etc. Also, we have learnt important lessons during coding process. Fist, we should understand the requirements of the program. How it should be, what kind of tools or functions should you use. Second, testing and debugging the program. While we were creating this program, we faced a lot of difficulties. Most difficult part was debugging and fixing errors. We spent a lot of time to identify bugs and tried to fix it. In the future, this program could be improved by adding proper interface for using. Also, adding some features such as tracking, priority delivery, show revenue by date etc. A big thank you for our team members outstanding contributions to our recent coding project. Your creativity, dedication, and positive spirit made our collaboration a success. I appreciate each one of you for the excellent work and look forward to more fruitful collaborations in the future.

# 6. References

W3Schools. (n.d.). Python for Loops.

Retrieved from: <https://www.w3schools.com/python/python_for_loops.asp>

IOFlood. (2023, January 15). Python: Create File. IOFlood Blog.

Retrieved from: <https://ioflood.com/blog/python-create-file/>

freeCodeCamp. (2023, February 10). Python Exit: How to Use an Exit Function in Python to Stop a Program.

Retrieved from: <https://www.freecodecamp.org/news/python-exit-how-to-use-an-exit-function-in-python-to-stop-a-program/>

Simplilearn. (2023, March 5). Enumerate in Python: Understanding the Basics. Retrieved from: <https://www.simplilearn.com/tutorials/python-tutorial/enumerate-in-python#:~:text=Enumerate()%20is%20a%20built,form%20of%20an%20enumerate%20object>.

W3Schools. (2023, April 12). Python JSON.

Retrieved from: <https://www.w3schools.com/python/python_json.asp>

Programiz. (2023, May 20). Python While Loop.

Retrieved from: <https://www.programiz.com/python-programming/while-loop>

Real Python. (2023, June 8). Python Conditional Statements.

Retrieved from: <https://realpython.com/python-conditional-statements/>

Array - efficient arrays of numeric values. Python documentation. (n.d.). https://docs.python.org/3/library/array.html

GeeksforGeeks. (2023, December 4). Python list methods. GeeksforGeeks. <https://www.geeksforgeeks.org/list-methods-python/>

How to append elements in python tuple?. Online Tutorials, Courses, and eBooks Library. (n.d.). https://www.tutorialspoint.com/How-to-append-elements-in-Python-tuple

Workload Matrix

|  |  |  |
| --- | --- | --- |
| Name | TP number | Work done |
| Musaev Beniamin | TP071185 | Group Leader  Admin part-student 1  Operator part- student 4  Worked on documentation together |
| Niyazaly Zhassulan | TP071009 | Operator part-student 3  Worked on documentation together |
| Khegai Olga | TP069365 | Admin part- student 2  Worked on documentation together |