ASIA PACIFIC UNIVERSITY
TECHNOLOGY & INNOVATION

CT108-3-1-PYP-LAB-7

PROGRAMMING WITH PYTHON

TITLE	**PROGRAMMING WITH PYTHON ASSIGNMENT**
NAME & STUDENT ID	**Chee Kai Jian TP073698**
INTAKE	**March 2023**
LECTURER	**Amardeep Singh A/L Uttam Singh**
DATE OF SUBMISSION	**6th July 2023**

Tables of Content

1.0 Assumptions

The user is assumed to have some level of English literacy, and a functional input device. The user is also assumed to be using Microsoft window and have all the main code, inventory.txt and userdata.txt on their computer. The data inside all the file is assumed to be remain untouched by the navigating user.

1.1 Introduction

To begin the program, the system utilizes a “while True” loop to initiate the log_in () function. This is the first program , shown in Figure 1.0.1. The system will run and receive input from the user. In this case , a prompt regrading the choices will be shown (Figure 1.0.2) and the user will have two choices either type in log in or 1 to proceed and to exit and break the program type in exit or 2.

```
while True:
    print('''
                Welcome to Main Page
            ---------------------------------
            1 - Log In
            2 - Exit
        ''')
    choice = input("Enter Your Choice: ")
    if choice == '1' or choice == "log in".lower():
        log_in()
    elif choice == '2' or choice == 'Exit':
        break
```

Figure 1.0.1

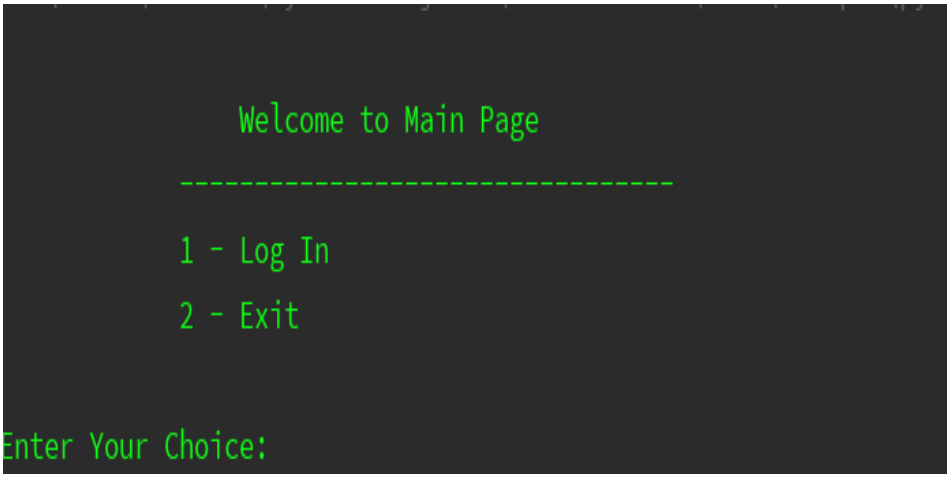

Figure 1.0.2

Once the user enter the first choice to continue the program , the log_in () function will Prompts the user to enter their username, password, and user type. It checks if the entered credentials match any user in the user file and if a match is found, it directs the user to the corresponding functionality based on their identity. If the credentials don't match, the user is prompted to try again. In the Figure 1.0.3 , Figure 1.0.4 and Figure 1.0.5 below shows the interaction with the log_in () function.

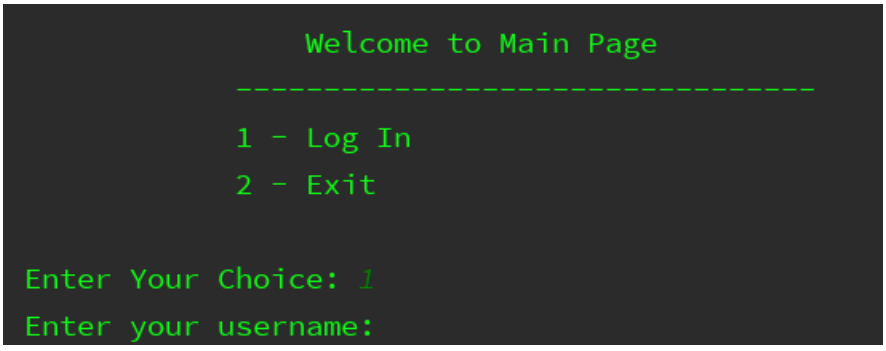

Figure 1.0.3 shows username input

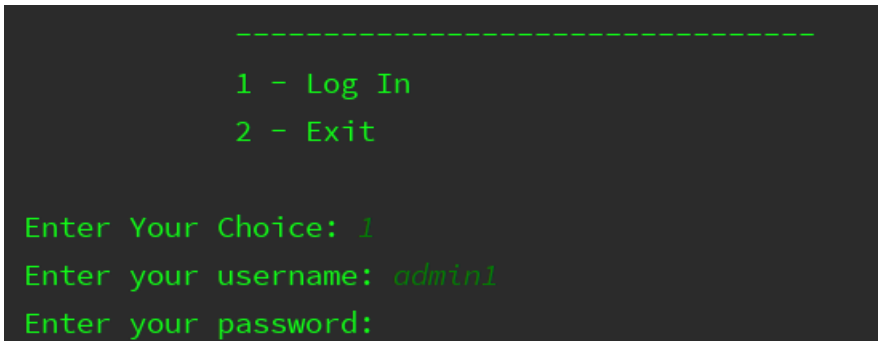

Figure 1.0.4 shows password input

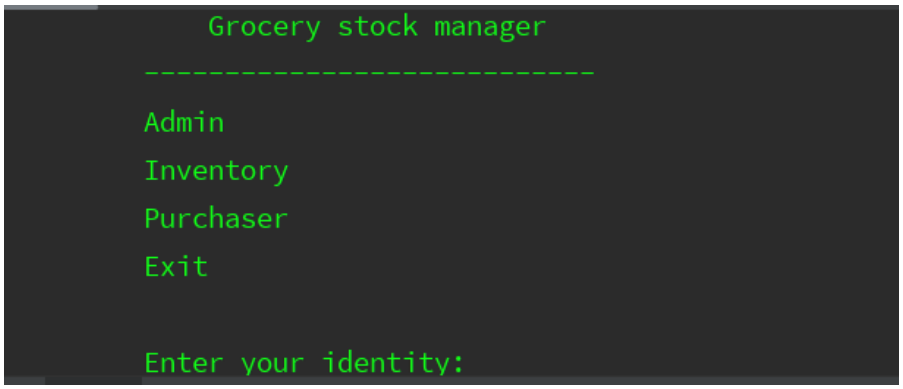

Figure 1.0.5 shows Identity/User type input

Following a successful log in into the system, the program will match the identity input to different user type and with different user type comes with difference in the degree of authorization to alter, change, or view the inventory file itself.

For instance, the user type admin will allow user to view, add new items , delete items , reduce quantity of items, view item under minimum quantity , add new user , delete user and search for item using different method including by item code range, description, price range or category . In the Figure 1.0.6 shows the admin main menu screen giving user the choice to do myriads of action regrading editing the inventory information and user information.

```
def admin(user_type):
    while True:
        print('''
                    Please Enter An Option:
                    ----------------------
                    1 - View Inventory
                    2 - Insert Item
                    3 - Delete Item
                    4 - Stock taking
                    5 - Update item
                    6 - View Replenish List
                    7 - Stock Replenishment
                    8 - Search item
                    9 - Add new user
                   10 - Delete user
                   11 - Exit
                    ''')
```

Figure 1.0.6 shows admin function

The admin function will first take in the input from user regarding the option presented and store the option in the variable {choice} and the program will match the input with the condition of different functions. For example, in Figure 1.0.7, the item_insert function used for adding new item into the inventory needs the choice (user input) to be either insert item or the number beside in the Figure 1.0.6 and to make it easier to access. lower () is utilized to make the user input lowercase to match the lowercase string “insert item”. Once the condition is met, the program will run the functions accordingly.

```
if choice == "insert item".lower() or choice == "2":
    item_insert(user_type)
if choice == "delete item".lower() or choice == "3":
    delete_item(user_type)
if choice == "stock taking".lower() or choice == "4":
    stock_taking(user_type)
if choice == "update item".lower() or choice == "5":
    update_inventory(user_type)
if choice == "View replenish".lower() or choice == "6":
    view_replenish()
if choice == "stock replenishment".lower() or choice == "7":
    stock_replenishment(user_type)
if choice == "search item".lower() or choice == "8":
    search(user_type)
if choice == "add new user".lower() or choice == "9":
    add_new_user(user_type)
if choice == "delete user".lower() or choice == "10":
    delete_user(user_type)
if choice == "exit".lower() or choice == "11":
    log_in()
```

Figure 1.0.7 shows the condition of running different functions.

```
def inventory_checker(user_type):
    while True:
        print('''
        Please Enter An Option:
        ----------------------
        1 - View Inventory
        2 - Stock Taking
        3 - Search item
        4 - Exit
        ''')
        choice = input("Enter your option: ")
        if choice == "view inventory".lower() or choice == "1":
            inventory = read_inventory()
            for item in inventory:
                i = ' '.join(item) + '\n'
                print(i)
        if choice == "stocking taking".lower() or choice == "2":
            stock_taking(user_type)
        if choice == "search item".lower() or choice == "3":
            search(user_type)
        if choice == "exit".lower() or choice == "4":
            log_in()
```

Figure 1.0.8 shows the function for inventory checker menu.

For the inventory_checker function ,it uses a while True loop to keep the program running , this while True loop is comes into use when the user input a number or an option that does not belong to the authority of this particular user type . To display the option provided by this user type, the program will print options and ask the user to input his or her choice . This inventory checker accepts either number besides the option or the option itself, to make the system more efficient , the program will automatically lowercase the users input .

If the case of user choosing to view inventory , the program first will assign the list from read_inventory() function to inventory, secondly, the for loop will go through every item in the inventory list and use join() to turn it into string , ‘ ‘ to make spaces in between and ‘\n’ to go next line. Finally , it will all be assigned to the variable { i } and print .

For the other choices , both stock taking and search item will take user to the function of those option and finally exit takes user back to log_in function (Main menu).

```
def purchaser(user_type):
    while True:
        print('''
        Please Enter an Option
        ----------------------
        1 - View Inventory
        2 - Stock Replenishment
        3 - Search item
        4 - Exit

        ''')
        choice = input("Enter your option: ")
        if choice.lower() == "view inventory" or choice == "1":
            inventory = read_inventory()
            for item in inventory:
                i = ' '.join(item) + '\n'
                print(i)

        if choice.lower() == "stock replenishment" or choice == "2":
            stock_replenishment(user_type)
        if choice.lower() == "search item" or choice == "3":
            search(user_type)
        if choice.lower() == "exit" or choice == "4":
           log_in()
```

Figure 1.0.9 shows the function for purchaser.

In the purchaser use type menu the option view inventory, stock replenishment , search item and exit are presented. Similar to inventory_checker above except for view inventory option ,other option goes to respective function and exit returns user back to log_in function.

-1.2 View Inventory

The first choice within the admin function is the View inventory, this choice allows user to take a look at the inventory information of the system. In following Figure 1.0.8 shows the code of the view inventory program inside the admin function. This program to run the choice must meet the conditions. After that , the program will firstly assign the variable inventory to the return value of the function read_inventory () , the program will use for loop to loop through the item inside inventory and use empty spaces to join item into string and utilize \n to go to line below the existing written line ;afterwards assign the result to the variable { i } and print the variable { i }.Finally the result of this particular program shown in the Figure 1.0.9. For the design of the code , In the Figure 1.2.2 and 1.2.3 shows the flow chart and the pseudocode in Figure 1.2.4.

```
if choice == "view inventory".lower() or choice == "1":
    inventory = read_inventory()
    for item in inventory:
        i = ' '.join(item) + '\n'
        print(i)
```

Figure 1.2.1

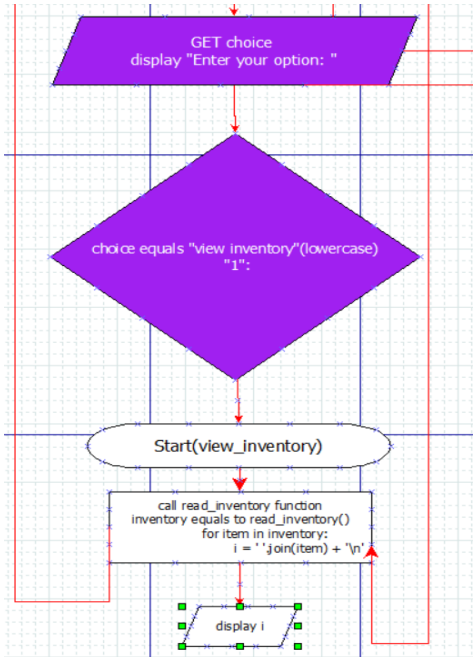

Figure 1.2.2

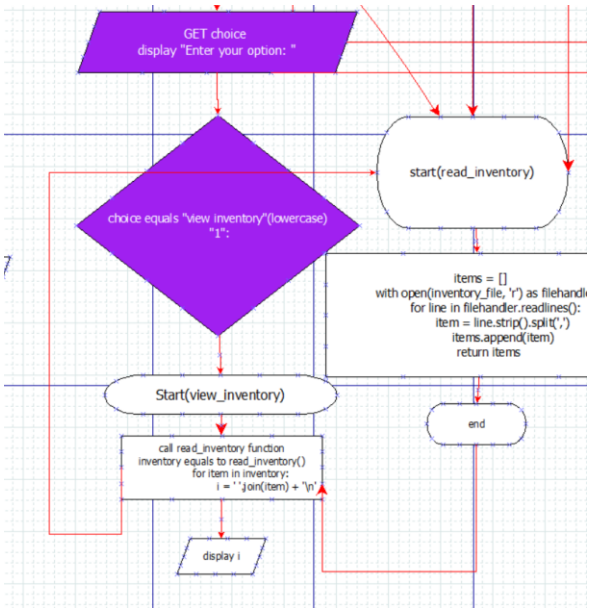

Figure 1.2.3 shows the program calling read inventory function.

```
if user_input equal "1" or "view item(lowercase)":
    assign inventory to the return value of read_inventory()
        for item in inventory:
            i = ' '.join(item) + '\n'
            display i
```

Figure 1.2.4

-1.3 Insert item

Another function of the admin identity is the ability to add / append new items into the inventory file. Once the condition is met the program will call and run the item_insert function. The purpose of this function is to collect data, combine it into a list and append it into the inventory file. This function starts off with assigning an empty list, while at the same time asking the input of the information regarding the new item shown below (Figure 1.3.1). After receiving input from the user, the program takes the variables of the inputs and append them into the empty string call inventory previously assigned and by calling the append_inventory_file(inventory) function to append the information of the new item. Regarding the append_inventory_file function shown in Figure 1.3.2, due to the function characteristic it takes in {inventory} as variable, loops it and appends into the inventory file. This is also the reason for making a empty list called inventory.

After successfully inserting a new item, the system will ask input from user regarding the choice to either continue adding items or going back to the main menu of the user's identity. This particular action is made possible by using the log_in function when getting user's identity and returning it shown in Figure 1.3.3. When it comes to the planning of this function both pseudocode (Figure 1.3.4 and flowchart (Figure 1.3.5.1 and 1.3.5.2)

```
def item_insert(user_type):
    inventory = []
    code = input("Enter item code: ")
    item_name = input("Enter item name: ")
    category = input("Enter category: ")
    unit = input("Enter Unit: ")
    price = input("Enter price: ")
    quantity = input("Enter your quantity: ")
    minimum = input("The minimum amount of the item: ")
    item = [code, item_name, category, unit, str(price), str(quantity), minimum]
    inventory.append(item)
    append_inventory_file(inventory)
    input_1 = input("Do you wish to continue adding item?(Yes Or No):")
    if input_1 == 'yes'.lower():
        return item_insert(user_type)
    if input_1 == 'no'.lower():
        return user_type
    else:
        print("Please try again..")
```

Figure 1.3.1 shows the code for item_insert function.

```
def append_inventory_file(inventory):
    with open(inventory_file, 'a') as filehandler:
        for item in inventory:
            new_data = ','.join(item) + '\n'
            filehandler.write(new_data)
```

Figure 1.3.2 shows append_inventory_file(inventory) Function.

```python
    for user in user_data:
        if user[0] == username and user[1] == password and user[2] == user_type:
            item_found = True
            if user_type == 'admin':
                admin(user_type)
                return user_type
            if user_type == 'inventory':
                inventory_checker(user_type)
                return user_type
            if user_type == 'purchaser':
                purchaser(user_type)
                return user_type
    if not item_found:
        print("Identity doesn't match...")
        return log_in()
```

Figure 1.3.3 shows the log_in function returning user_type.

```
function item_insert(user_type):
    declare code, name, category, unit, price, quantity, minimum as variables
    declare item and inventory as list
    display "Enter item code: "
    GET code
    display "Enter item name: "
    GET name
    display "Enter category: "
    GET category
    display "Enter unit: "
    GET unit
    display "Enter price: "
    GET price
    display "Enter quantity: "
    GET quantity
    display "Enter minimum: "
    GET minimum

    item = [code, item_name, category, unit, str(price), str(quantity), minimum]
    append item into inventory
    append_inventory_file(inventory)

    display "Do you wish to continue adding item?(Yes Or No):"
    GET input_1
    if input_1 == 'yes'(lowercase):
        return item_insert()
    elif input_1 == 'no'(lowercase):
        return to user_type
    else:
        display "Please try again"
```

Figure 1.3.4 shows pseudocode

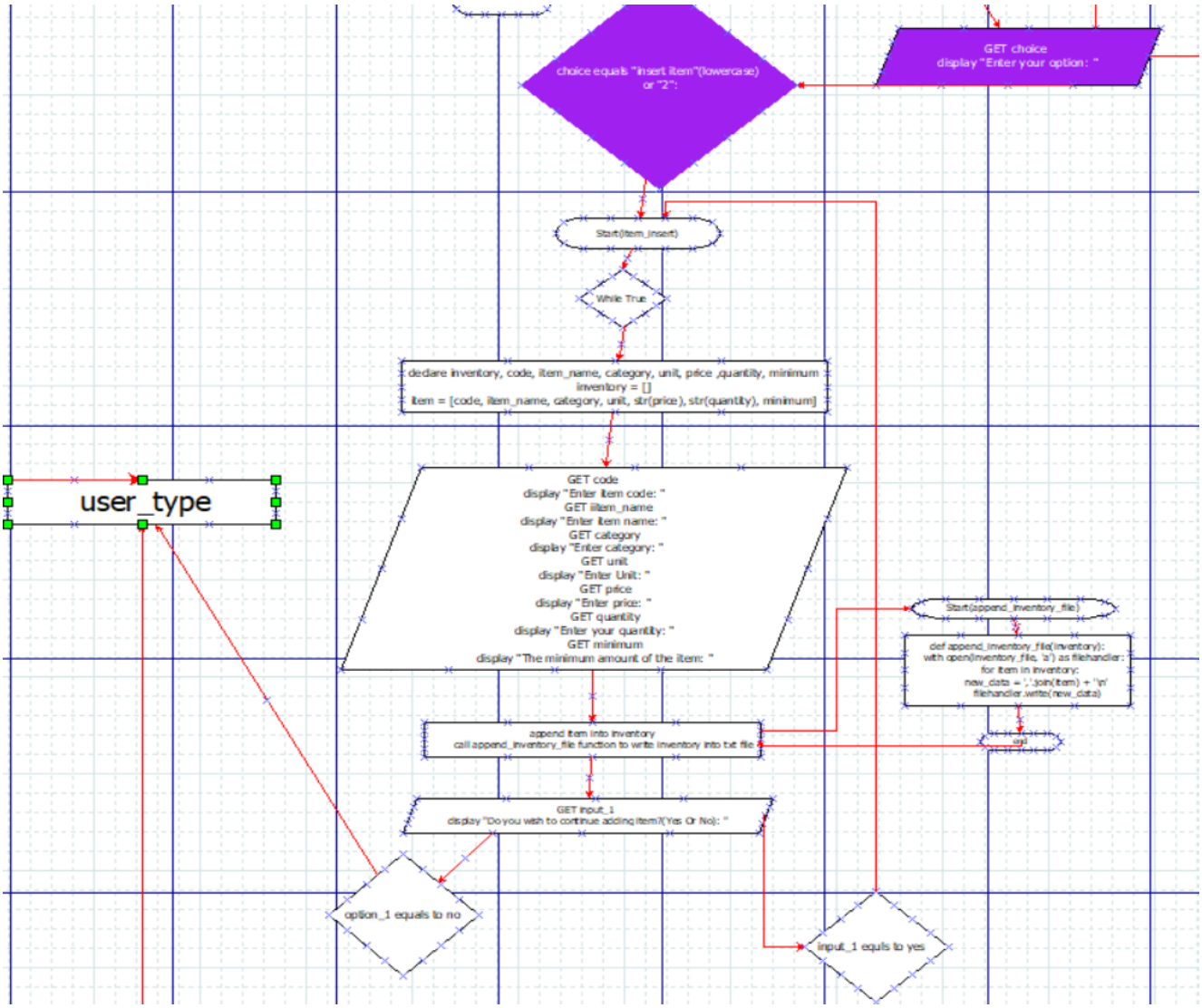

Figure 1.3.5 shows the flow chart for item_insert

- 1.4 Delete item.

For the ability to delete items in the system, it is only available for the authority of the admin. In the figure 1.4.1 below shows the pseudocode, the figure 1.4.2 shows the flow chart and the source code shown in figure 1.4.3. To begin, the read_inventory function is assigned to variable call items, Secondly the delete_item function will ask an input from user regarding the code of the item that the user wishes to delete.

A Boolean value of False is assigned to item_found at the start and as the program runs following lines it will change based on whether the item is found in the inventory.

The program will go through each line of the inventory and match the code with the user's input.

If user does not match , it is append into the a list and the list will be written again into the inventory file ; on the other hand if matched, the specific line will not be written down.
In the end the system will ask the user to choice between continuing or return to the user's identity's menu. This particular action is made possible by using the log_in function when getting user's identity and returning it.

```
function delete_item(user_type):
    declare item_found as boolean
    declare items, code, option_1, option_2 as variable
    declare inventory as list
    item_found = False
    inventory = []
    items = read_inventory

    display "Enter the item code you want to delete: "
    GET code

    for item in items:
        if code found in item:
        item_found = True
        display "Item has been deleted"

        else:
            append item into inventory

    if item_found is True:
        GET option_1
        display "Do you wish to continue ?(Yes Or No):"(lowercase)
        if option_1 == "yes":
            return delete_user(user_type)
        if option_1 == "no":
            return to user_type


    if not item_found:
        display 'Item not found'
        GET option_2
        display "Do you wish to continue ?(Yes Or No):")(lowercase)
        if option_2 == "yes":
            return delete_user(user_type)
        if option_2 == "no":
            return to user_type
```

Figure 1.4.1 shows pseudocode for delete function

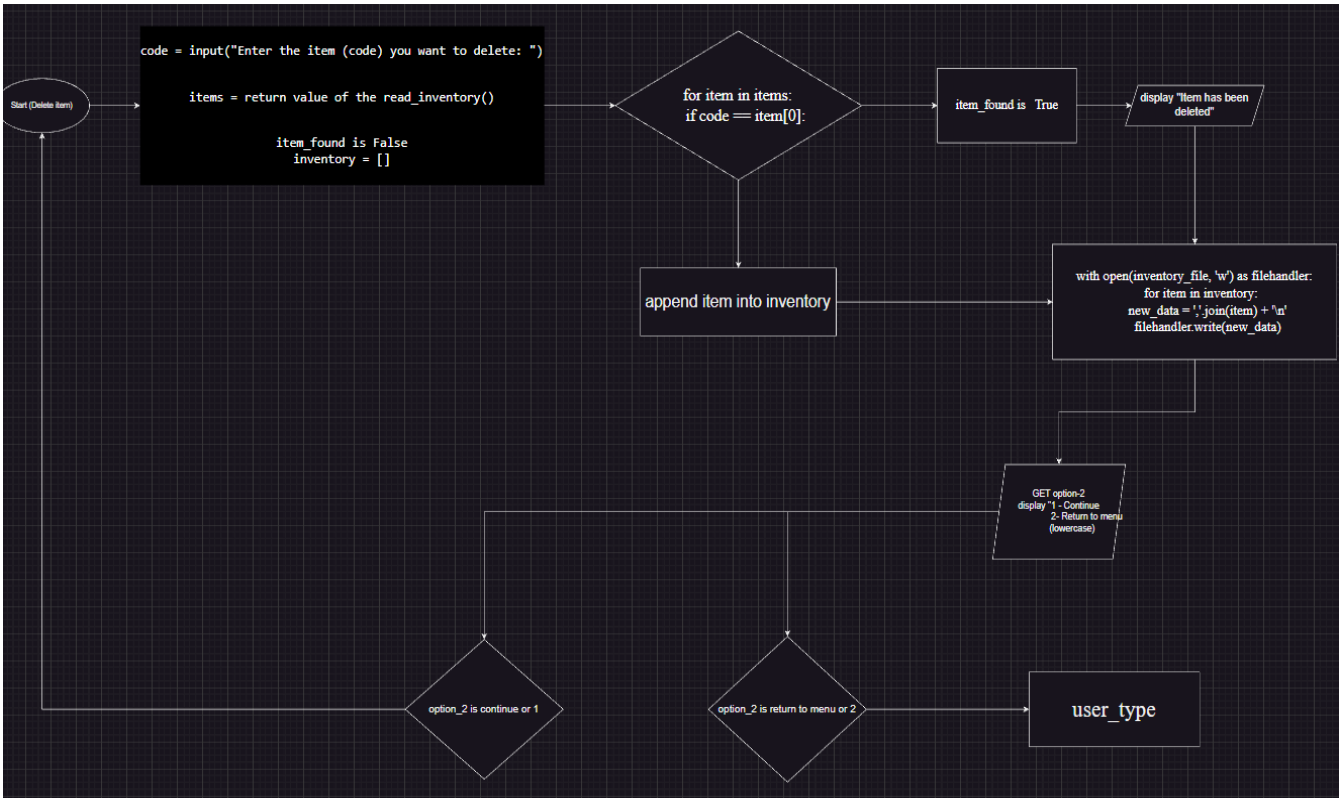

Figure 1.4.2 shows flowchart for delete function.

```
def delete_item(user_type):
    code = input("Enter the item (code) you want to delete: ")
    items = read_inventory()
    item_found = False
    inventory = []
    for item in items:
        if code == item[0]:
            item_found = True
            print("Item has been deleted")
        else:
            inventory.append(item)
    with open(inventory_file, 'w') as filehandler:
        for item in inventory:
            new_data = ','.join(item) + '\n'
            filehandler.write(new_data)
    if item_found is True:
        option_1 = input("Do you wish to continue ?(Yes Or No):").lower()
        if option_1 == "yes":
            return delete_user(user_type)
        if option_1 == "no":
            return user_type
    if not item_found:
        print('Item not found')
        option_2 = input("Do you wish to continue ?(Yes Or No):").lower()
        if option_2 == "yes":
            return delete_user(user_type)
        if option_2 == "no":
            return user_type
```

Figure 1.4.3 shows source code

-1.5 Update Inventory

The code for update_inventory function shown in figure 1.5.1 and figure 1.5.2 is only allowed to be accessed by admins. This function utilizes a for loop to find a match between the code of the item that the user wish to change and if this condition is met the system will ask new information from user to write new information into the inventory file, and similar to the delete function above ,if the code does not match it will append the specific line into a list and write it again into the file. Finally, this function uses the write_inventory_function to insert data. The pseudocode (figure 1 and figure 2) and flow chart is shown at figure 3 and 4 below. In both situations of successful updating of an item's information and item not found, the user would ask to input a choice regarding to continue or go back to user_type input at the log_in function above. This is made possible by having the log_in function returning user_type because the value that a function returns to the caller is generally known as the function's return value. This value can be call upon by imputing the value of the variable assigned it is to, inside the bracket of the function and within the function use return user type.

```
def update_inventory(user_type):
    items = read_inventory()
    inventory = []
    codenotfound = True
    item_found = False
    item_number = input("Enter the item(code) you want to update: ")
    for item in items:
        if item_number == item[0]:
            item_found = True
            new_code = input("Enter new item code: ")
            new_name = input("Enter new item name: ")
            new_category = input("Enter new category: ")
            new_unit = input("Enter new unit: ")
            new_price = input("Enter new price: ")
            new_quantity = input("Enter new quantity: ")
            new_minimum = input("Enter new minimum: ")
            updated_item = [new_code, new_name, new_category, new_unit, new_price, new_quantity, new_minimum]
            inventory.append(updated_item)

            codenotfound = False
            print("Item Updated!")
        else:
            inventory.append(item)

    if not item_found:
        print("Item not found...")
    write_inventory_file(inventory)
```

Figure 1.5.1 shows source code for upper section update inventory function

```
    if not item_found:
        print("Item not found...")
    write_inventory_file(inventory)



    if item_found is True:
        option_1 = input("Do you wish to continue ?(Yes Or No):").lower()
        if option_1 == "yes":
            return update_inventory(user_type)
        if option_1 == "no":
            return user_type

    if not codenotfound:
        option_2 = input("Do you wish to continue ?(Yes Or No):").lower()
        if option_2 == "yes":
            return update_inventory(user_type)
        if option_2 == "no":
            return user_type
```

Figure 1.5.2 shows source code for lower section update inventory function

```
function update_inventory (user_type):
    declare item, items, item_number, new_code, new_name, new_category, new_unit, new_quantity, new_minimum, option_1, option_2 as variable
    declare codenotfound and item_found as boolean
    declare inventory as list
    items = read_inventory()
    inventory = []
    codenotfound = True
    display "Enter the item(code) you want to update: "
    GET item_number

    for item in items:
        item_found = False
        if item_number found in item:
            item_found = True
            display "Enter new item code: "
            GET new_code
            display "Enter new item name: "
            GET new_name
            display "Enter new category: "
            GET new_category
            display "Enter new unit: "
            GET new_unit
            display "Enter new price: "
            GET new_price
            display "Enter new quantity: "
            GET new_quantity
            display "Enter new minimum: "
            GET new_minimum


            item[0] = new_code
            item[1] = new_name
            item[2] = new_category
            item[3] = new_unit
            item[4] = new_price
            item[5] = new_quantity
            item[6] = new_minimum

            append item into inventory
            codenotfound = False
            print("Item Updated!")

        if codenotfound still False
            append item into inventory
```

Figure 1 shows upper section of flow chart for update function.

```
    if codenotfound is True:
        print("Item not found..")

    if item_found is True:
        GET option_1
        Display"Do you wish to continue ?(Yes Or No):"(lowercase)
        if option_1 == "yes":
            return update_inventory(user_type)
        if option_1 == "no":
            return to user_type

    if codenotfound still False:
        GET option_2
        display "Do you wish to continue ?(Yes Or No):"(lowercase)
        if option_2 == "yes":
            return update_inventory(user_type)
        if option_2 == "no":
            return to user_type




    call write_inventory_file(inventory) function to write file
```

Figure 2 shows lower section of flow chart for update function

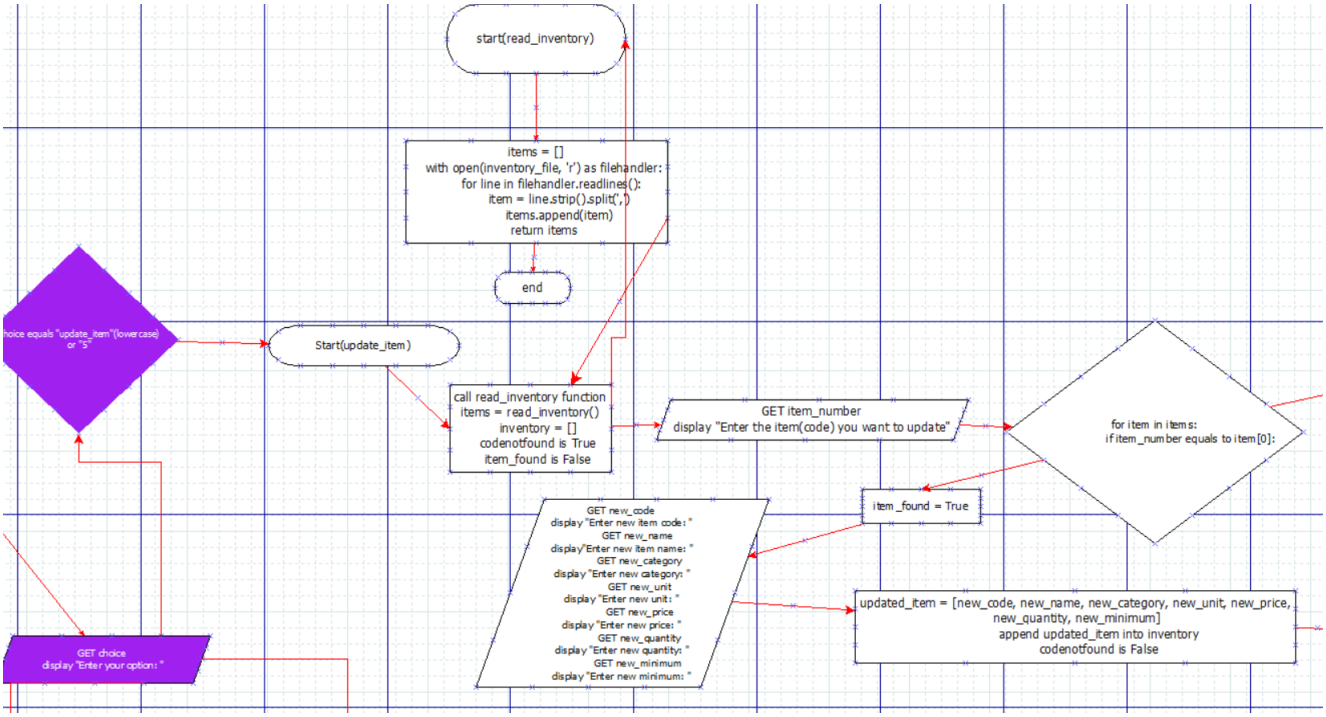

Figure 3 shows left side of flow chart for update function.

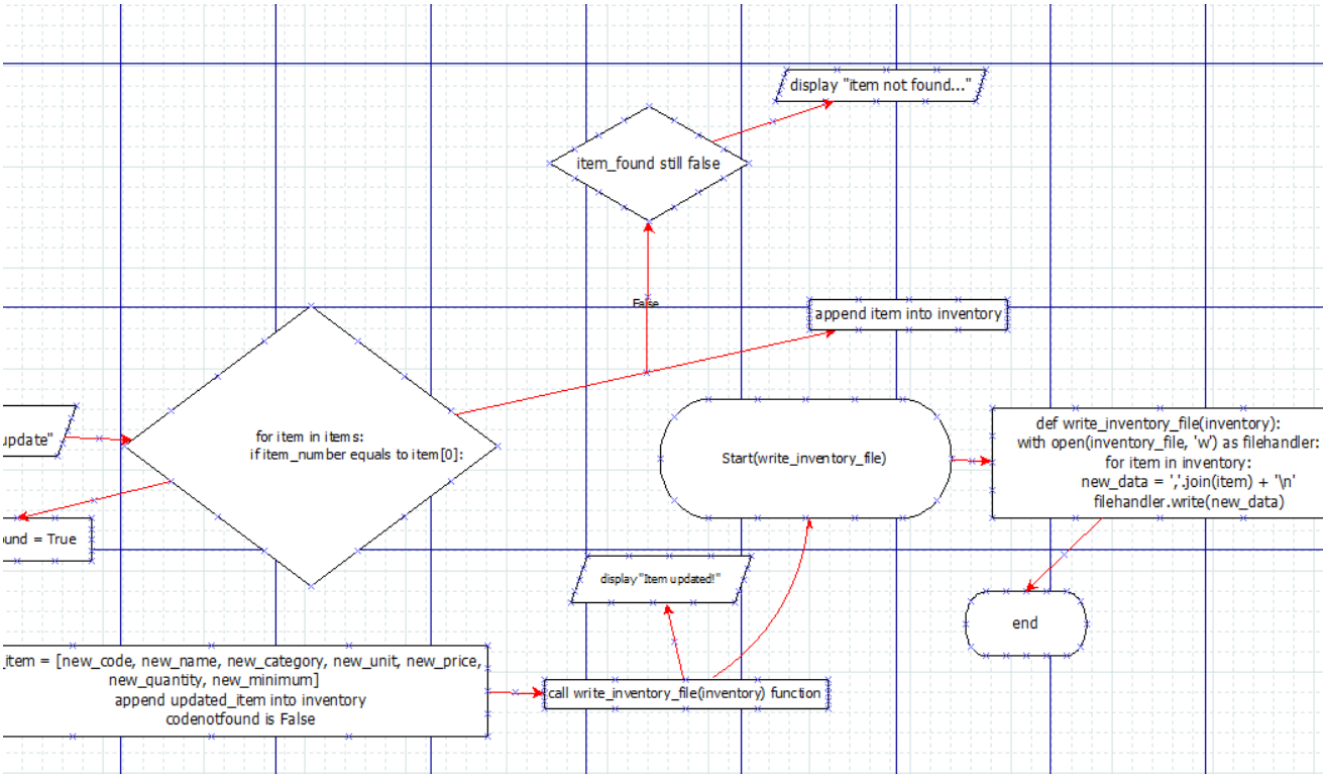

Figure 4 shows right side of flow chart for update function.

-1.6 Stock taking

```
def stock_taking(user_type):
    code = input("Enter the item(code) you want to take: ")
    inventory = read_inventory()

    item_found = False
    for item in inventory:
        if code == item[0]:
            print("The quantity of the item is " + item[5])
            option_1 = input("Confirm or change? :")
            if option_1 == 'confirm'.lower():
                item_found = True
                option_2 = input("Do you wish to continue ?(Yes Or No):").lower()
                if option_2 == "yes":
                    return stock_taking(user_type)
                if option_2 == "no":
                    return user_type

            if option_1 == 'change'.lower():
                item_found = True
                new_quantity = input("Enter your new Quantity: ")
                item[5] = new_quantity
                print("Item's quantity changed successfully")
                option_2 = input("Do you wish to continue ?(Yes Or No):").lower()
                if option_2 == "yes":
                    return stock_taking(user_type)
                if option_2 == "no":
                    return user_type

    if not item_found:
        print("The item doesnt exist...")
        option_1 = input("Do you wish to continue ?(Yes Or No):").lower()
        if option_1 == "yes":
            return stock_taking(user_type)
        if option_1 == "no":
            return user_type

    write_inventory_file(inventory)
```

Figure 1.6.1 shows the stock taking function

In the figure 1.6.1 above shows the stock taking function authorized to both admin and inventory checker. This function can perform stock-taking for the items by keying in the item code. The system will print the quantity available, and the user may confirm or change the quantity accordingly. This program uses an if statement inside a for loop that allows it to look through each line of the inventory file. On the other hand, this program uses a similar method like other function to identity if the item is found through Boolean value. Finally, to write the new quantity into the file, it uses the write_inventory_function. To get a detailed process of stock taking function, the flow chart shown in figure 1.6.3 and figure 1.6.4 provides it and the pseudocode shown in figure 1.6.2 provides better understanding of structure of the function.

Similar to other functions in the program , in situations of confirm , change or failure to find the specific item , the program will present a choice to either continue(return stock_taking(user_type)) or return user-type

```
function stock_taking(user_type):
    declare code, item, option_1,option_2, new_quantity as variables
    declare inventory as list
    declare item_found as boolean
    inventory = read_inventory()
    item_found = False
    display "Enter the item code: "
    GET code
     for item in inventory:
        if code is in item:
            display "The quantity of the item is " + item[5]

            display "Confirm or change? :"
            GET option_1

            if option_1 == 'confirm'.lower():
                item_found = True
                GET option_2
                display "Do you wish to continue ?(Yes Or No):"
                if option_2 == "yes":
                    return stock_taking(user_type)
                if option_2 == "no":
                    return user_type

            if option_1 == 'change'.lower():
                item_found = True
                GET new_quantity
                display "Enter your new Quantity: "
                item[5] = new_quantity
                display "Item's quantity changed successfully"
                GET option_2
                display "Do you wish to continue ?(Yes Or No):"
                if option_2 == "yes":
                    return stock_taking(user_type)
                if option_2 == "no":
                    return user_type

    if item_found still False:
        display "The item doesnt exist..."
        GET option_1
        display "Do you wish to continue ?(Yes Or No):"(lowercase)
        if option_1 == "yes":
            return stock_taking(user_type)
        if option_1 == "no":
            return to user_type
    call write_inventory_file(inventory) function to write file
```

Figure 1.6.2 shows pseudocode for stock taking

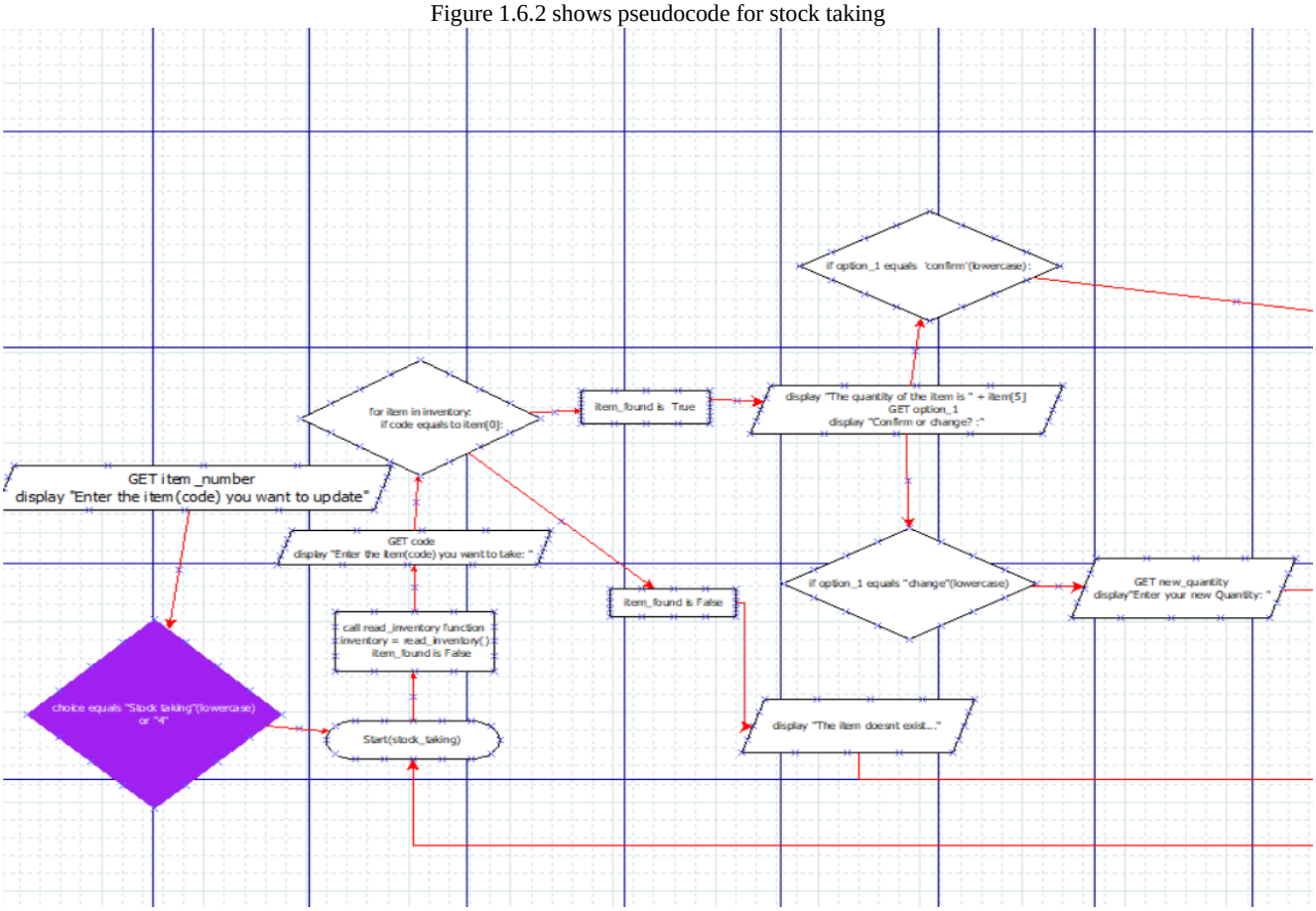

Figure 1.6.3 shows flowchart (left portion)

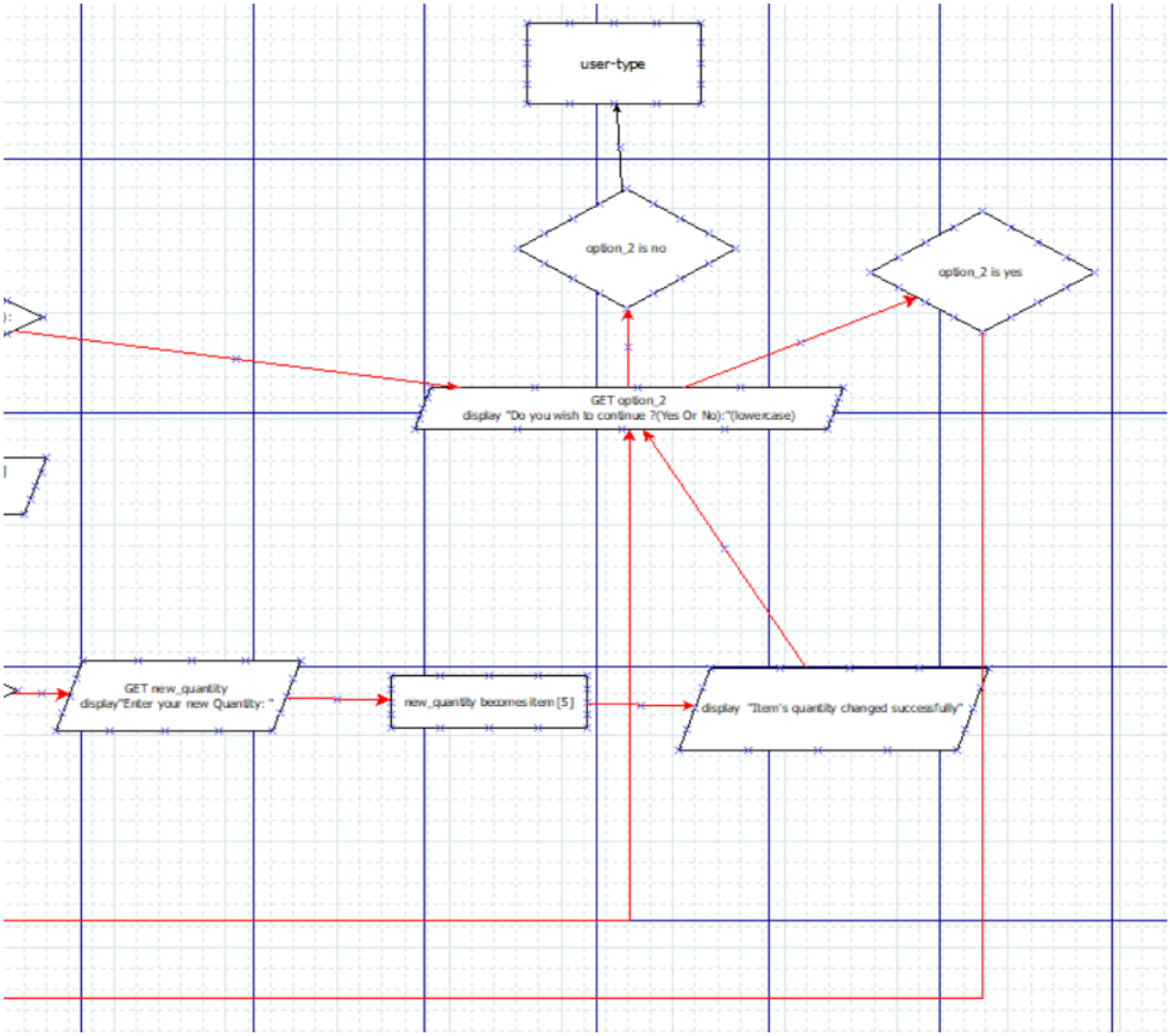

Figure 1.6.4 shows flowchart (Right portion)

-1.7 View Replenish List

The user accessing this function will be able to see a list of things they need to buy again and this function is meant to show the items that have fewer than the required minimum quantity. The Pseudocode of this function shown in figure 1.7.1 give a clear idea on the structure of the codes.

```
function view_replenishment():
    declare inventory as list
    declare item_found as boolean
    declare i, quantity, minimum_stock as variables

    inventory = read_inventory()
    item_found = False

    for i in inventory:
        quantity = int(i[5])
        minimum_stock = int(i[6])
        if minimum_stock > quantity:
            print(i)
            item_found = True
    if not item_found:
        print("All items are above the minimum quantity")
```

Figure 1.7.1

Unlike the majority of the function in the program , due to its characteristics , It is on the short side when it comes to codes length .This program will read inventory using a function and on default assign the Boolean value ‘False ‘ to item_found, this will turn to True if an item is found and in the case of item not found according to the condition , the program will return a string statement confirming items are above minimum quantity. This function can only be accessed by admins. Inside the for loop the program assigns the fifth item of inventory to quantity and turn it into integer, this is also applied on the sixth item in inventory to the integer variable minimum_stock . Afterwards, the program will try to look through the item’s quantity that are under the minimum amount and if the inventory has item under minimum amount, the program will print item and item_found will be assigned the Boolean value true.

This function does not have a continue or go back user type’s main menu option, due to it providing the ability to see whole list.

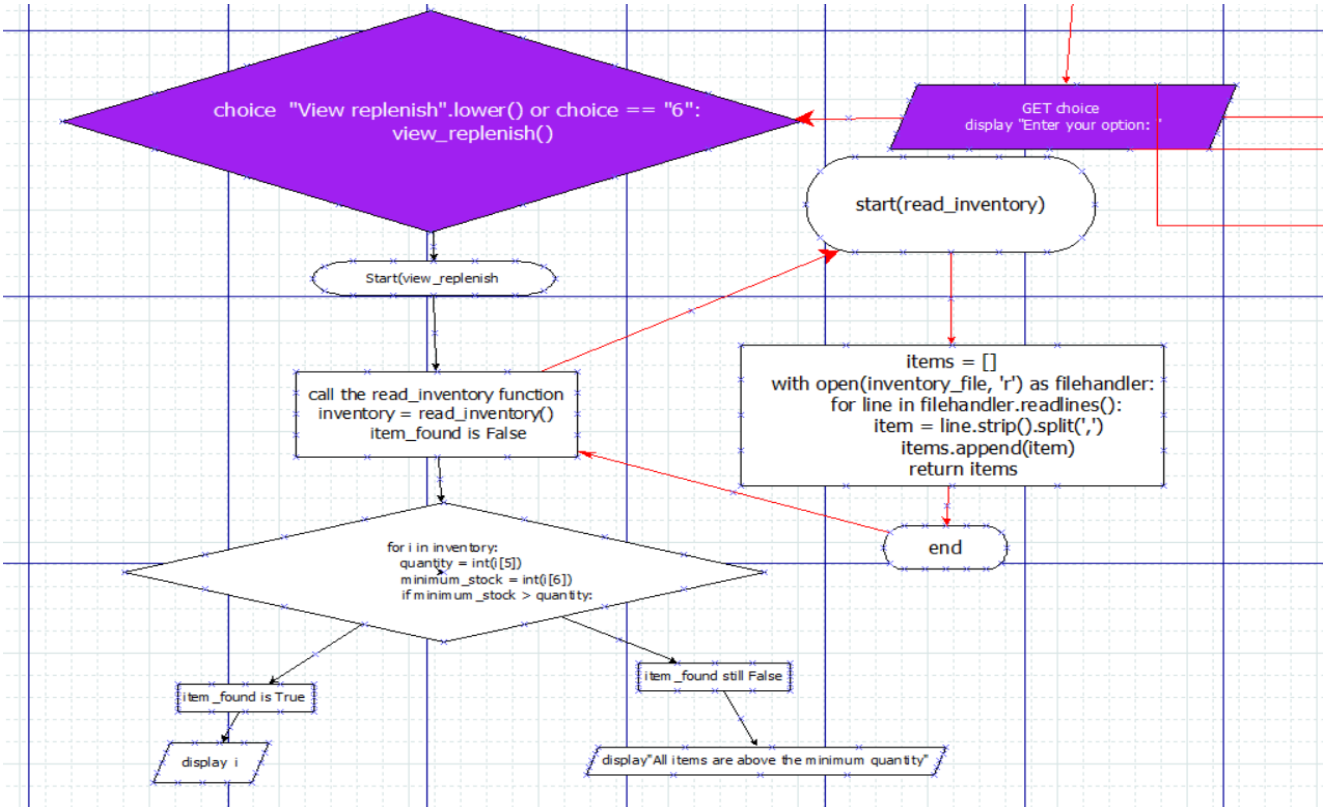

Figure 1.7.2 shows flowchart for view replenish list function.

```
def view_replenish():
    inventory = read_inventory()
    item_found = False

    for i in inventory:
        quantity = int(i[5])
        minimum_stock = int(i[6])
        if minimum_stock > quantity:
            print(i)
            item_found = True
    if not item_found:
        print("All items are above the minimum quantity")
```

Figure 1.7.3 shows source code for view replenish list function

-1.8 Stock Replenishment

```
def stock_replenishment(user_type):
    inventory = read_inventory()
    code = input("Enter code of Item: ")
    item_found = False
    for item in inventory:
        if item[0] == code:
            item_found = True
            print(f"The quantity of this particular item is {item[5]}")
            option_1 = input("Do you want to update the quantity of this item?(yes or no) : ").lower()
            if option_1 == 'yes':
                new_quantity = input("New quantity: ")
                item[5] = new_quantity
                print("Quantity of the item has been updated")
                option_3 = input("Do you wish to continue?(yes or no): ").lower()
                if option_3 == 'yes':
                    return stock_replenishment(user_type)
                if option_3 == 'no':
                    return user_type

            elif option_1 == 'no':
                return user_type

    if not item_found:
        print("Item cannot be found...")
        option_2 = input("""
            1-Continue
            2-Return to menu
            """).lower()
        if option_2 == "continue" or '1':
            return stock_replenishment(user_type)
        if option_2 == "return to menu" or '2':
            return user_type
    write_inventory_file(inventory)
```

Figure 1.8.1 shows stock replenishment function

This function allows the user (purchaser and admin) will key in the item code, the system will show the quantity. Then, the user may add the quantity of the newly purchased items. The stock_replenishment() function begins by retrieving the inventory data. It asks the user to provide the code of the item they want to update. The function then searches the inventory for a matching item based on the entered code. If a match is found, the current quantity of the item is shown, and the user is prompted to update the quantity if desired. Upon updating the quantity, a confirmation message is displayed. The function offers the option to continue or exit. Choosing to continue allows the user to input another item code, while selecting to exit returns control to the user's type main program. If no matching item is found, an error message is shown, and the user is given the choice to continue or return to user's type main menu. Additionally, the function ensures that the updated inventory data is saved back to the original source using the write_inventory_file

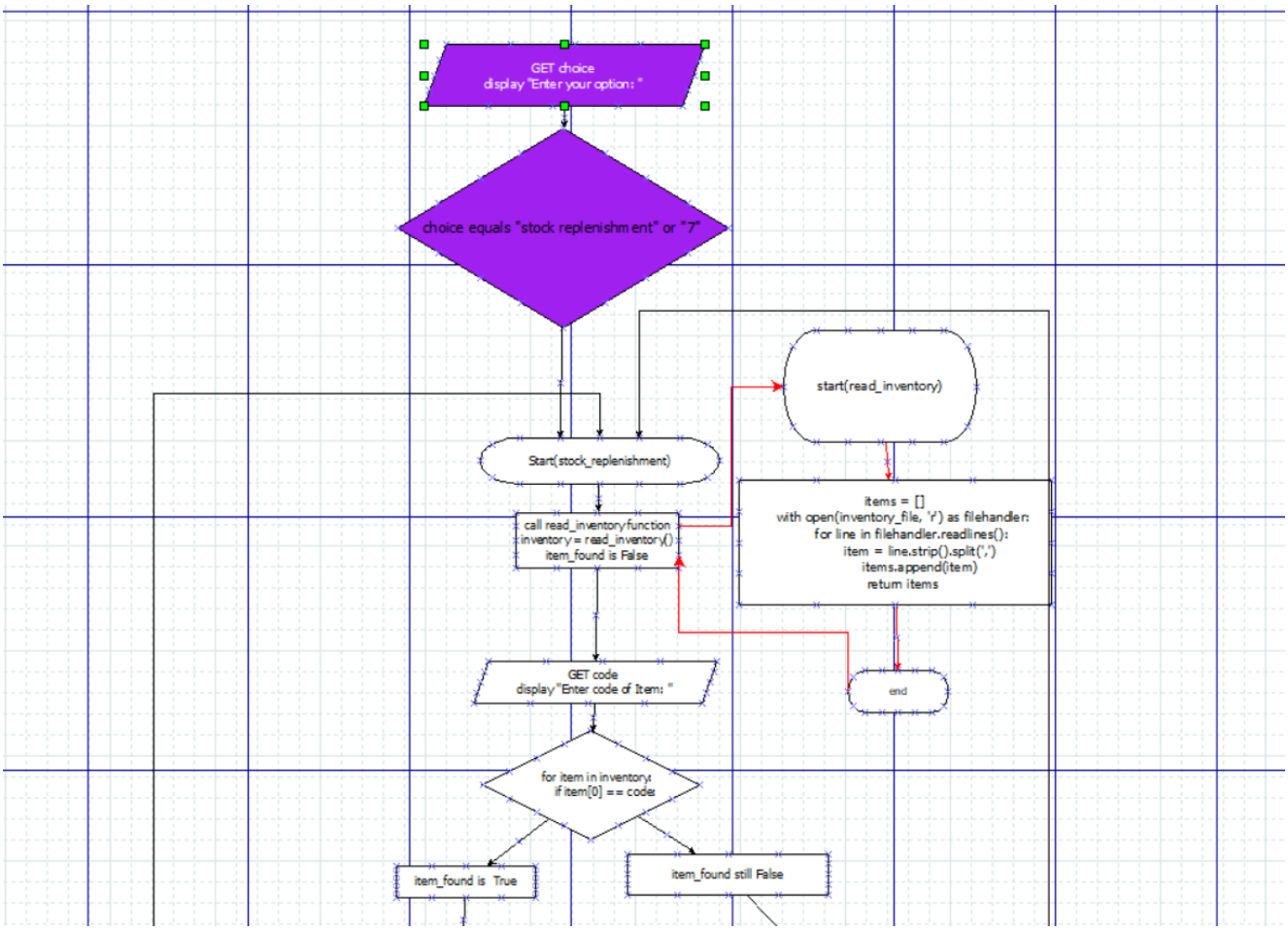

Figure 1.8.2 shows the top half of flowchart for stock replenish function.

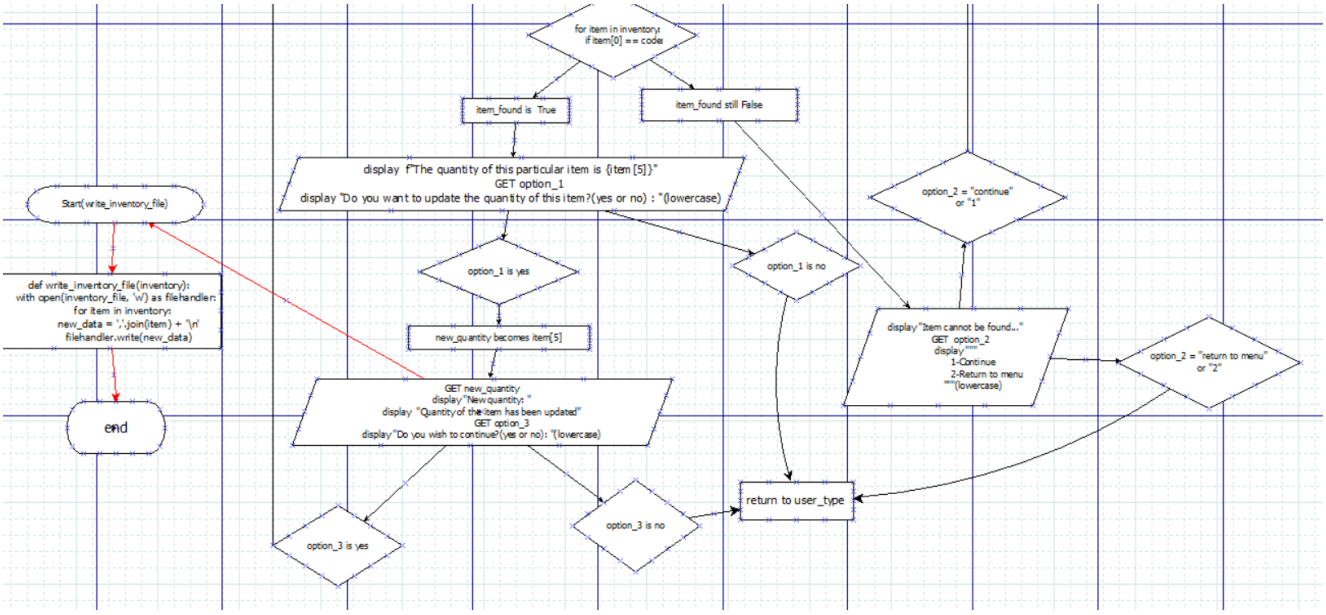

Figure 1.8.3 shows the bottom half of flowchart for stock replenish function.

```
function stock_replenishment(user_type):
    declare item_found as boolean
    declare inventory as list
    declare code, option_1, item, option_3, option_2 as variables

    item_found = False
    inventory = read_inventory()
    display "Enter code of Item: "
    GET code

    for item in inventory:

        if item[0] == code:
            item_found = True
            display f"The quantity of this particular item is {item[5]}"

            display "Do you want to update the quantity of this item?(yes or no) : "(lowercase)
            GET option_1

            if option_1 == 'yes':
                display "New quantity: "
                GET new_quantity
                item[5] = new_quantity
                display "Quantity of the item has been updated"

                display "Do you wish to continue?(yes or no): "(lowercase)
                GET option_3

                if option_3 == 'yes':
                    return stock_replenishment()

                if option_3 == 'no':
                    return log_in

            elif option_1 == 'no':
                return log_in()

    if item_found still False:
        display "Item cannot be found..."
        GET option_2 = input
        display """
                1-Continue
                2-Return to menu
                """(lowercase)

        if option_2 == "continue" or '1':
            return stock_replenishment(user_type)

        if option_2 == "return to menu" or '2':
            return to user_type

    call write_inventory_file(inventory) function to write file
```

Figure 1.8.4 shows pseudocode for stock replenish function.

-1.9 Search Item

The search_function is can be separated into four different part , and each part is separated by the option provided to the user shown below in figure 1.9.1 .The options are description , code range , category and price range .Each option are different method the user can search up items in the inventory .To begin the program , it uses a while True to make sure if the user enter an invalid input the function will keep going and ask the user to input again. The program will convert the users input into lowercase to make it easier for the user .

```
def search(user_type):
    while True:
        inventory = read_inventory()
        option_1 = input('''
            --------------
            1- Description
            2- Code range
            3- Category
            4- Price range
            5- Exit
            --------------
            Enter Method of item Search: ''').lower()
```

Figure 1.9.1

DESCRIPTION

The first part of this function is the search by description portion of the codes shown in figure 1.9.2.By default the program assign an empty list to found_items and Boolean value to item_found .To start of the program the program will ask input from user regarding the description of one's desired item ,using a for loop to search through each line .Once the descriptions matched , the program will append the exact line of the item into the empty list call found_items that the program made on default and the item_found utilized to keep the program updated on the status of the search .Under the item_found's Boolean Value , if item_found is true the program will display the search result . The program display it using for loop go through each item in found_items and using ' , ' to join list into string , '\n' to go next line .

Finally , In both situation of when item is found and display or item not found , the program will ask the user weather he\she would like to continue or go back to menu.

```python
if (option_1 == 'description') or (option_1 == "1"):
    description = input("Enter item description to search: ").lower()
    found_items = []
    item_found = False
    for item in inventory:
        if description.lower() == item[1].lower():
            found_items.append(item)
            item_found = True

    if item_found:
        print("Search Results:")
        for item in found_items:
            i = ','.join(item) + '\n'
            print(i)
            option_2 = input("""
            1-Continue
            2-Return to menu
            """)
            if option_2.lower() == "continue" or option_2 == "1":
                return search(user_type)
            if option_2.lower() == "return to menu" or option_2 == "2":
                return user_type

    if not item_found:
        print("Item not found...")
        option_2 = input("""
                    1-Continue
                    2-Return to menu
                    """)
        if option_2.lower() == "continue" or option_2 == "1":
            return search(user_type)
        if option_2.lower() == "return to menu" or option_2 == "2":
            return user_type
```

Figure 1.9.2 shows source code for search function

```
if (option_1 equals "description" or "1"):
    GET description
    display "Enter item description to search: "(lowercase)
    found_items = []
    item_found is  False

    for item in inventory:
        if description(lowercase) == item[1](lowecase):
            found_items.append(item)
            append item into found_items
            item_found is True

    if item_found:
        display "Search Results:"
        for item in found_items:
            i = ','.join(item) + '\n'
            display i
            GET option_2
            display """
                    1-Continue
                    2-Return to menu
                    """(lowercase)
            if option_2 equals "continue" or '1':
                return to search(user_type)
            if option_2 equals "return to menu" or '2':
                return to user_type

    if not item_found:
        display "Item not found..."
        GET option_2 = display """
                                1-Continue
                                2-Return to menu
                                """(lowercase)
        if option_2 equals "continue" or '1':
            return search(user_type)
        if option_2 equals "return to menu" or '2':
            return user_type
```

Figure 1.9.3 shows pseudocode for description section of search function

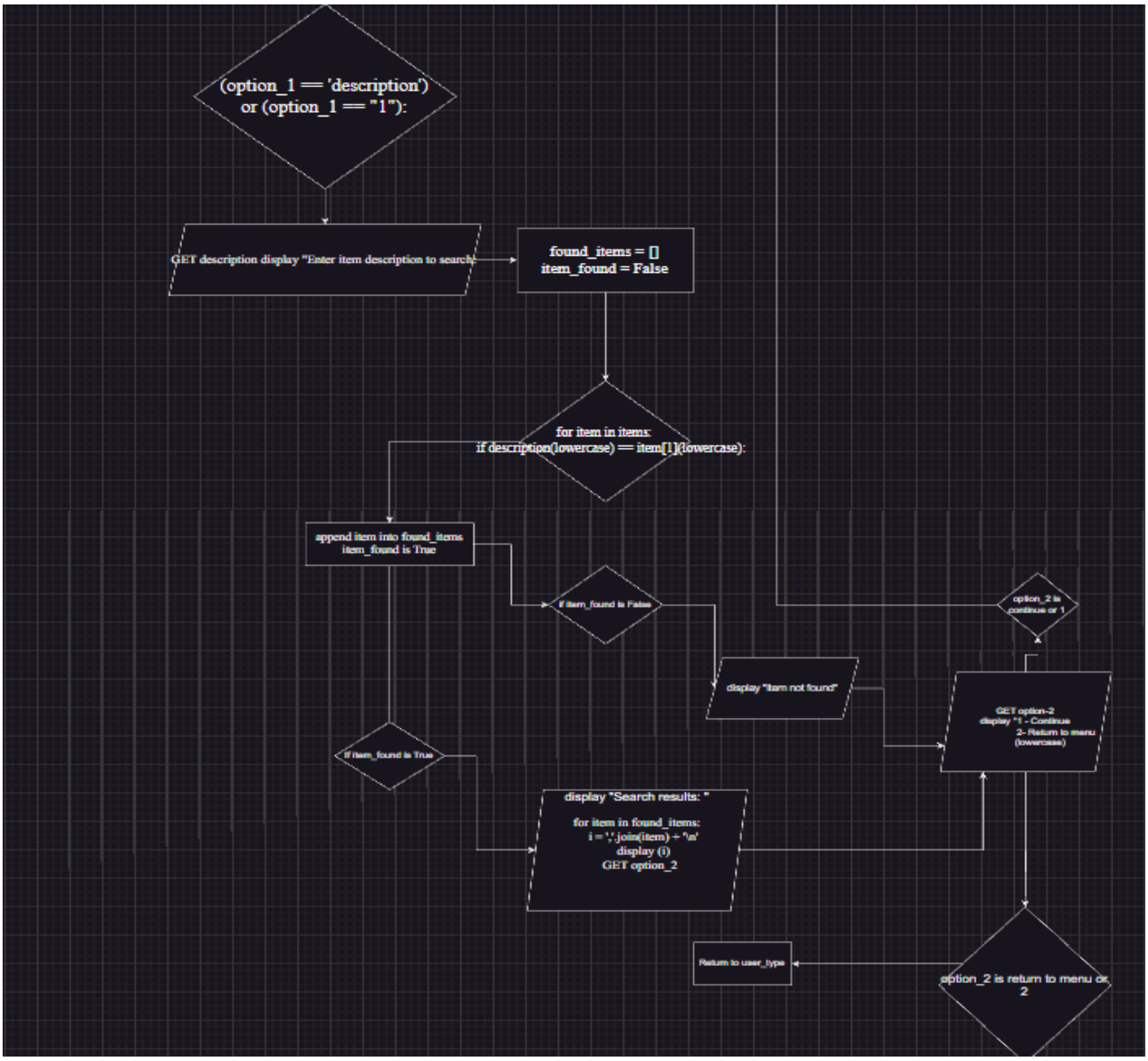

Figure 1.9.4 shows flowchart for description part of search function

CODE RANGE

```
if (option_1 == 'code range') or (option_1 == "2"):
    min_code = int(input("Enter minimum code: "))
    max_code = int(input("Enter maximum code: "))
    item_found = False
    for item in inventory:
        code = int(item[0])
        if min_code <= code <= max_code:
            item_found = True
            print(item)

    if item_found:
        option_3 = input("""
                        1-Continue
                        2-Return to menu
                        """)
        if option_3.lower() == "continue" or option_3 == "1":
            return search(user_type)
        if option_3.lower() == "return to menu" or option_3 == "2":
            return user_type

    if not item_found:
        print("Item not found...")
        option_3 = input("""
                        1-Continue
                        2-Return to menu
                        """)
        if option_3.lower() == "continue" or option_3 == "1":
            return search(user_type)
        if option_3.lower() == "return to menu" or option_3 == "2":
            return user_type
```

Figure 1.9.5 shows source code for code range section of search function

In the figure shown above, this section starts of with program asking input from user regarding minimum code(min_code) and maximum code(max_code) code range for item searches. On default the item_found will be False, next step for the program is using a for loop to find the code in between the desired code range and display it to user. Just like the previous functions and programs, the user will be presented with choice of either continuing or returning to menu.

```
if (option_1 equals 'code range' or "2"):
    GET min_code
    display"Enter minimum code: "
    GET max_code
    display "Enter maximum code: "
    item_found is False
    for item in inventory:
        code = integer(item[0])
        if code is min_code and max_code:
            item_found is True
            display item

    if item_found:
        GET option_2 =
        display"""
                    1-Continue
                    2-Return to menu
                    """(lowercase)
        if option_2 equals to "continue" or '1':
            return search(user_type)
        if option_2 equals to "return to menu" or '2':
            return user_type

    if not item_found:
        display "Item not found..."
        GET option_1 =
        display """
                1-Continue
                2-Return to menu
                """(lowecase)
        if option_1 is equals "continue" or '1':
            return search(user_type)
        if option_1 is equals "return to menu" or '2':
            return user_type
```

Figure 1.9.6 shows pseudocode for code range section of search function

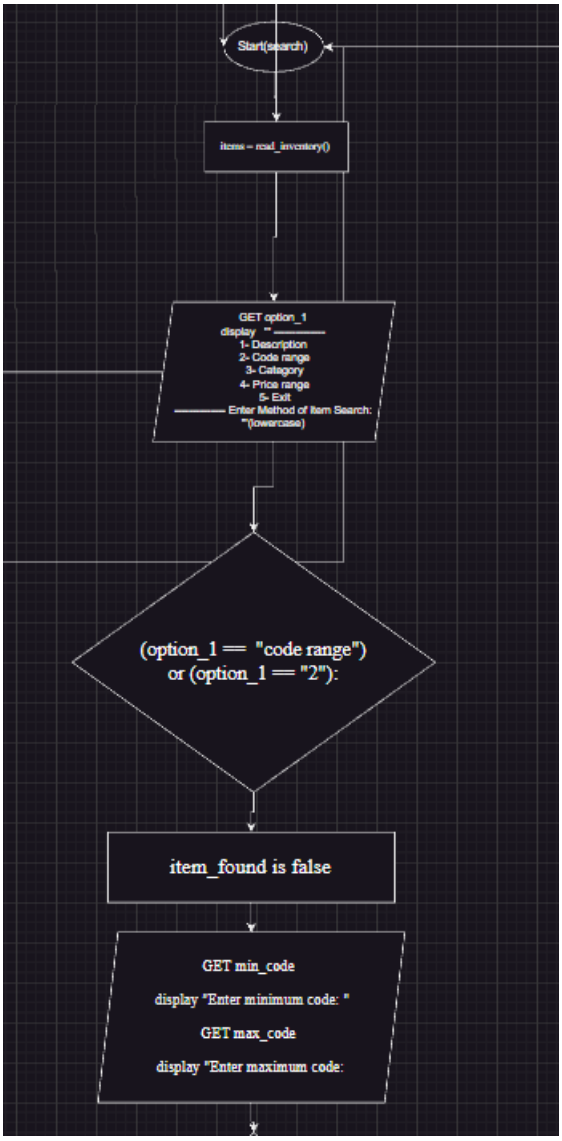

Figure 1.9.7 shows upper portion of search function's flowchart

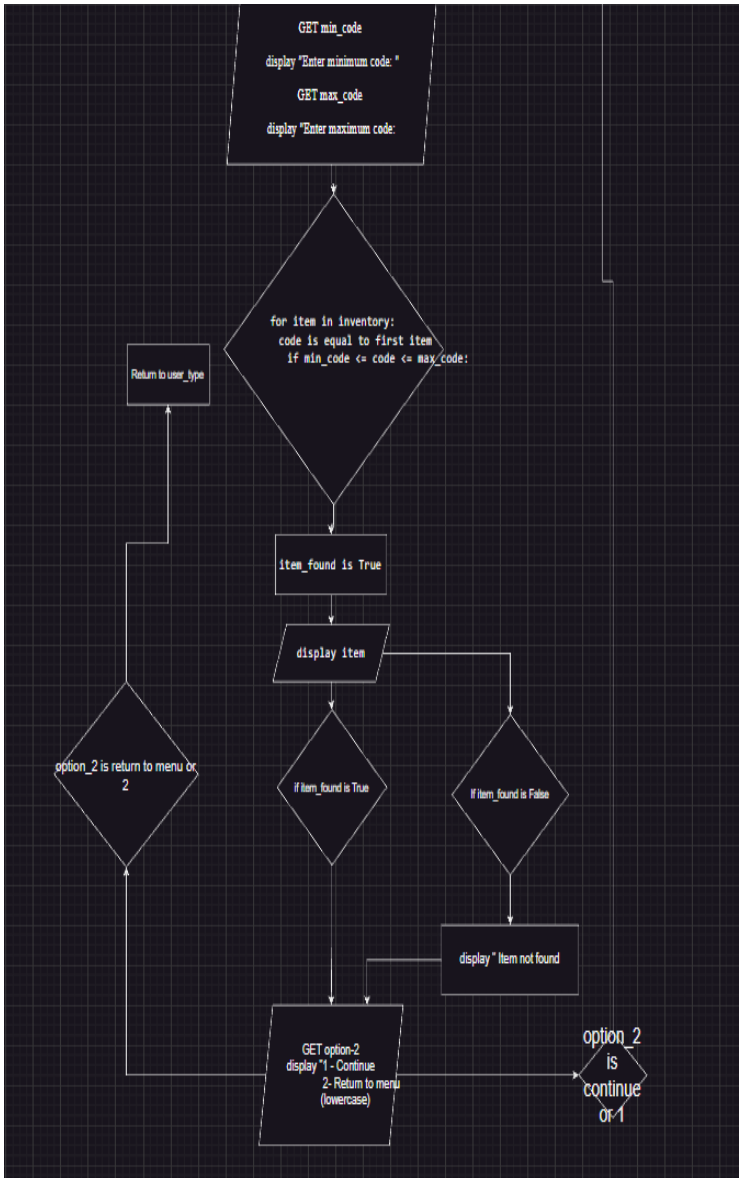

Figure 1.9.8 shows lower portion of search function's flowchart.

CATEGORY

This section of the code allows the user to search item by typing in the description of that particular item. This works the same way as the description section of the code. This program takes in input of the user and matches with existing category and prints it.The program ends user choosing if they want continue or go back to user's menu.

```
if (option_1 == 'category') or (option_1 == "3"):
    category = input("Enter item category to search: ")
    item_found = False
    for item in inventory:
        if category.lower() == item[2].lower():
            item_found = True
            print(item)

    if item_found is True:
        option_2 = input("""
                        1-Continue
                        2-Return to menu
                        """)
        if option_2.lower() == "continue" or option_2 == "1":
            return search(user_type)
        if option_2.lower() == "return to menu" or option_2 == "2":
            return user_type

    if not item_found:
        print("Item not found...")
        option_2 = input("""
                        1-Continue
                        2-Return to menu
                        """)
        if option_2.lower() == "continue" or option_2 == "1":
            return search(user_type)
        if option_2.lower() == "return to menu" or option_2 == "2":
            return user_type
```

Figure 1.9.9 shows the source code for category part of the function

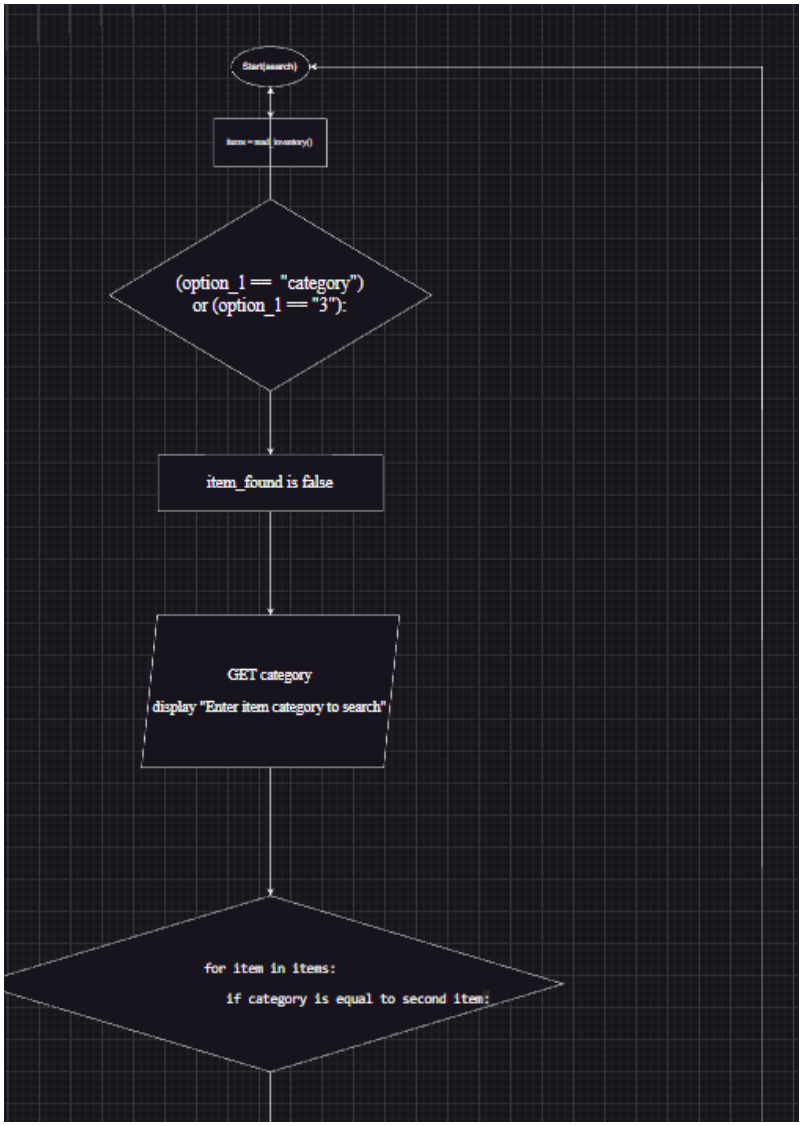

Figure 1.9.10 shows the upper flowchart for the category section.

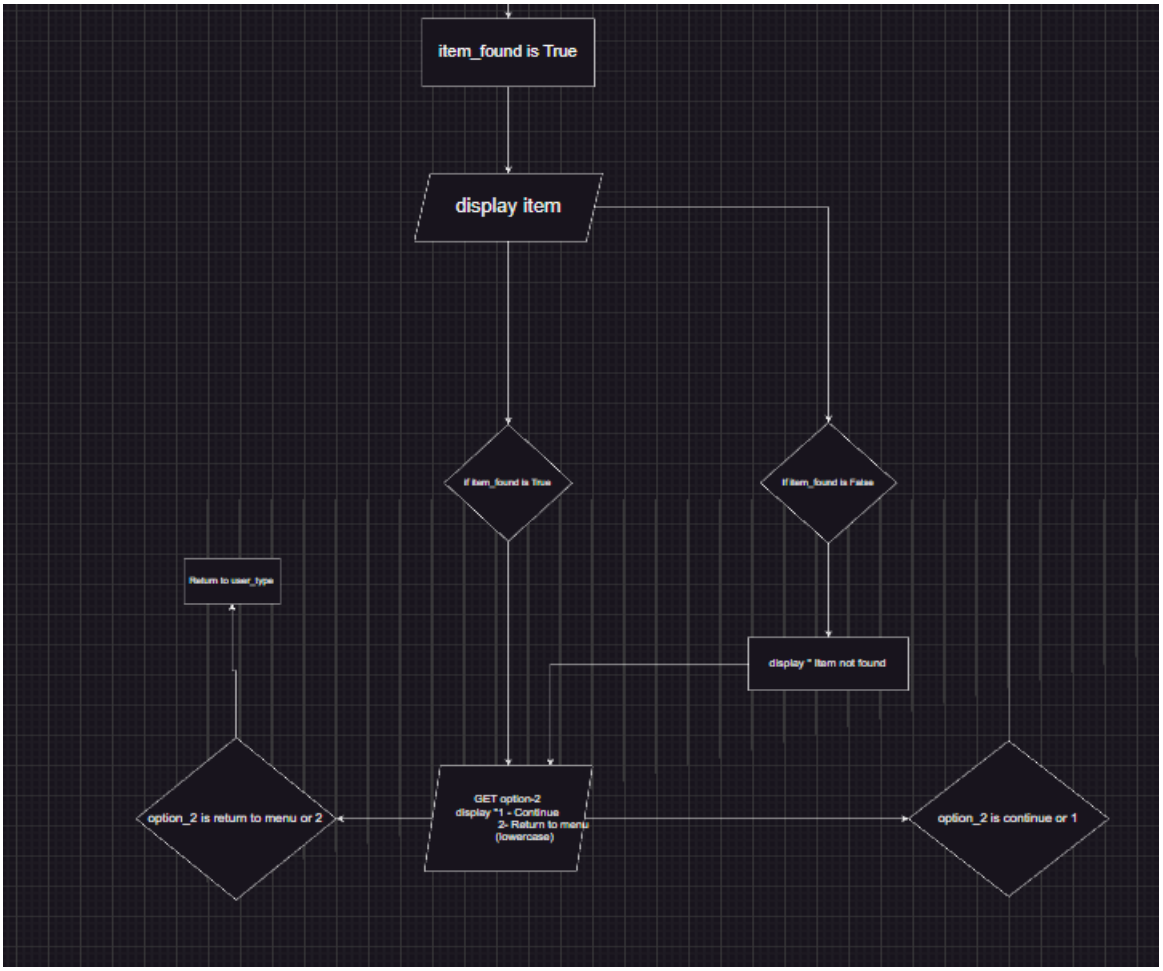

Figure 1.9.11 shows the lower flowchart for the category section

```
if (option_1 equals 'category' or "3"):
    GET category
    display "Enter item category to search: "
    item_found is False
    for item in inventory:
        if category(lowercase) == item[2](lowercase):
            item_found is True
            display item

    if item_found is True:
        GET option_2
        display"""
                1-Continue
                2-Return to menu
                """(lowercase)
        if option_2 equals "continue" or '1':
            return search(user_type)
        if option_2 equals "return to menu" or '2':
            return user_type

    if not item_found:
        print("Item not found...")
        GET option_2 = display"""
                            1-Continue
                            2-Return to menu
                            """(lowercase)
        if option_2 equals "continue" or '1':
            return search(user_type)
        if option_2 equals "return to menu" or '2':
            return user_type
```

Figure 1.9.12 shows pseudocode

- PRICE RANGE

Another method the user can use to search item is by price range shown in figure 1.9.13, for every item there is going to be a price tag recorded in the inventory file .This program takes the price range from the user and display the item in between the prices.

This function in terms of logic of the code is identical to the code range portion of this function. The code starts by checking if the user has selected the option for a "price range" search or entered "4". If they have, it proceeds with the search process. To begin the search, the program asks the user to input the minimum and maximum prices they are interested in. This helps narrow down the search results. Next, the program goes through each item in the inventory and compares its price to the specified range. If an item's price falls within that range, it gets displayed as a search result, and the program keeps track of finding at least one item by setting the item_found variable to True. After the search is complete, the program checks if any items were found. If items were found, it gives the user the choice to continue searching or return to the menu. The program waits for the user's input, and based on their choice, it either starts the search process again or goes back to the main menu.

If no items were found during the search, the program informs the user by displaying the message "Item not found." It then presents the same options to continue or return to the menu. The user can decide whether they want to try searching again or go back to the main menu. Lastly, if the user chooses the return option the program returns to the main menu of that user without performing any search.

```
if (option_1 == "price range") or (option_1) == "4":
    item_found = False
    min_price = int(input("Enter minimum price: "))
    max_price = int(input("Enter maximum price: "))

    for item in inventory:
        if min_price < int(item[0]) and int(item[0]) < max_price:
            print(item)
            item_found = True


    if item_found:
        option_4 = input("""
                        1-Continue
                        2-Return to menu
                        """)
        if option_4.lower() == "continue" or option_4 == "1":
            return search(user_type)
        if option_4.lower() == "return to menu" or option_4 == "2":
            return user_type

    if not item_found:
        print("Item not found...")
        option_4 = input("""
                        1-Continue
                        2-Return to menu
                        """)
        if option_4.lower() == "continue" or option_4 == "1":
            return search(user_type)
        if option_4.lower() == "return to menu" or option_4 == "2":
            return user_type
```

Figure 1.9.13 shows pseudocode for price range part in the search function

```
if (option_1 equals "price range" or  "4"):
    item_found is False
    GET min_price
    display "Enter minimum price: "
    GET max_price
    display "Enter maximum price: "

    for item in inventory:
        if integer(item[0]) is between min_price and max_price
            display item
            item_found is True


    if item_found:
        GET option_2
        display"""
                1-Continue
                2-Return to menu
                """(lowecase)
        if option_2 equals "continue" or '1':
            return search(user_type)
        if option_2 equals "return to menu" or '2':
            return user_type

    if not item_found:
        display "Item not found..."
        GET option_2 =
        display"""
                1-Continue
                2-Return to menu
                """(lowercase)
        if option_2 equals "continue" or '1':
            return search(user_type)
        if option_2 equals "return to menu" or '2':
            return user_type

if option_1 equals "exit" or "5":
    return user_type
```

Figure 1.9.14 shows pseudocode for price range option of search function

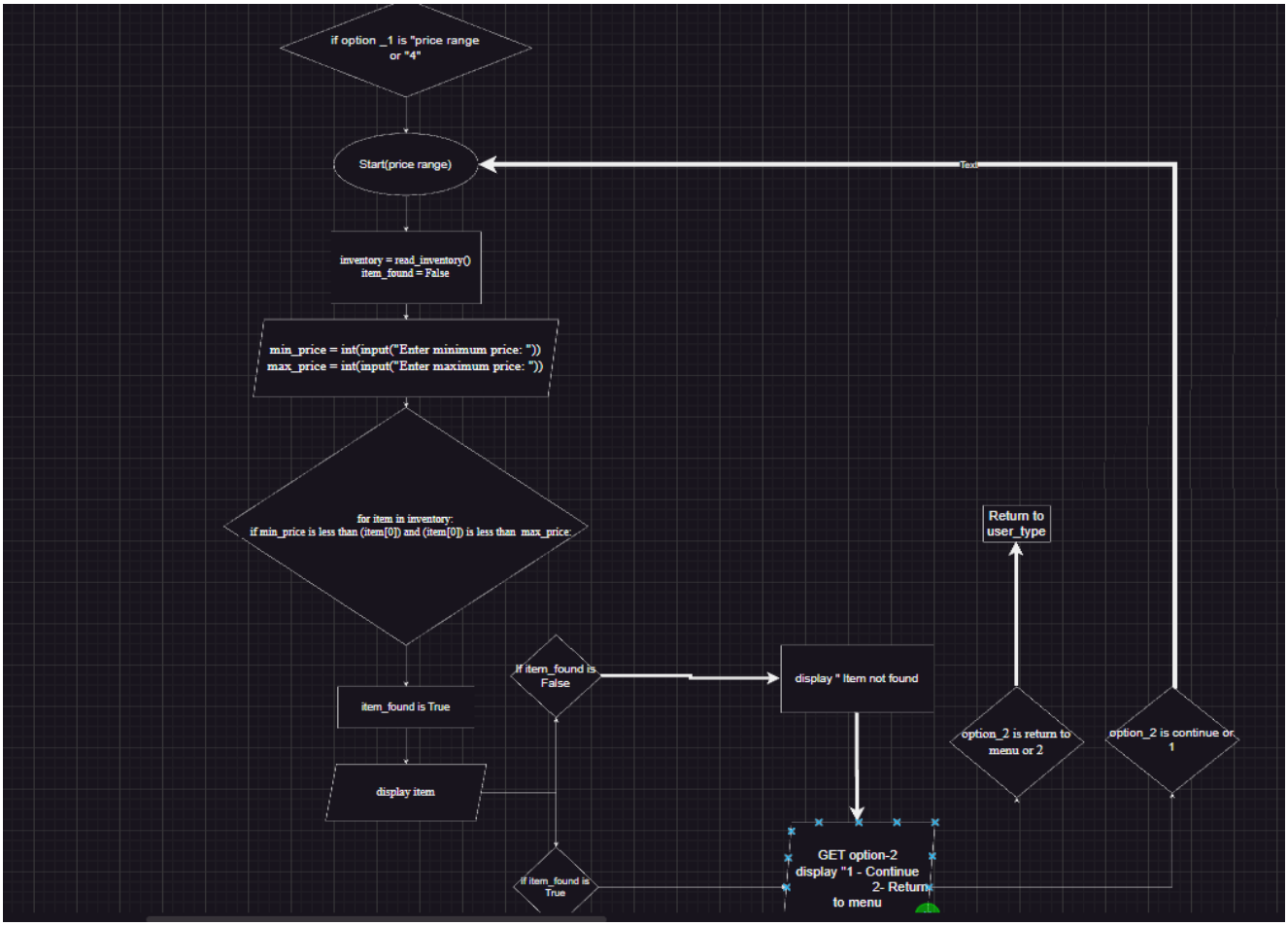

Figure 1.9.15 shows flowchart for price range method of search function

- 1.10 Add New User

Add new user function allows You to add new user to the system with the aid of the add_new_user function. This feature will request username, password, and user from you whilst you run it. Once you've entered user's data, it'll be saved in the user.txt file. You have the option to continue adding new user after each entry or return to main menu. The source code shown in figure 1.10.1, flow chart in figure 1.10.2 and pseudocode in figure 1.10.3.

First, you will be asked to input to enter a username, password, user type. This helps categorize users based on their roles or permissions within the system. The program will assign these info into user_info list and append it by firstly, use ‘,’ to join the list into string and \n to go next line. After that, the program uses .write to append it into the user file

The program ends by asking the user to input choice on continuing or return to user’s menu

```
def add_new_user(user_type):
    while True:
        username = input("Enter Your Username: ")
        password = input("Enter Your Password: ")
        usertype = input("Enter your Identity: ")
        user_info = [username, password, usertype]
        with open(user_file, 'a') as filehandler:
            user_insert = ','.join(user_info) + '\n'
            filehandler.write(user_insert)
        option_1 = input("Would u like to continue(Yes or No)")
        if option_1 == "yes".lower():
            return add_new_user(user_type)
        if option_1 == "no".lower():
            return user_type
```

Figure 1.10.1 shows source code for add_new_user function.

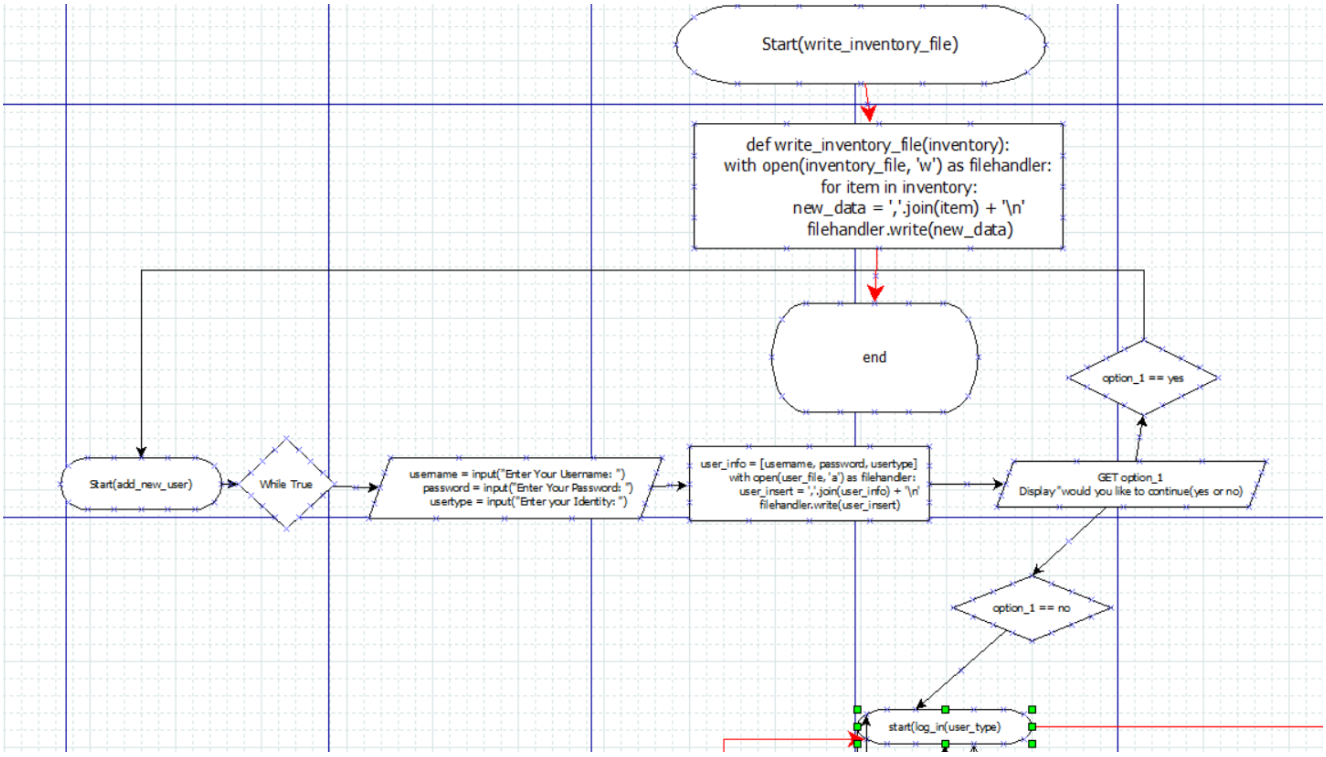

Figure 1.10.2 shows flowchart for add_new_user function

```
def add_new_user(user_type):
    while True:
        GET username
        display "Enter Your Username: "
        GET password
        display "Enter Your Password: "
        GET usertype
        display "Enter your Identity: "
        user_info = [username, password, usertype]
        with open as filehandler:
            user_insert = ','.join(user_info) + '\n'
            write(user_insert) into filehandler
        GET option_1
        display "Would u like to continue(Yes or No)"
        if option_1 equals "yes".lower():
            return add_new_user(user_type)
        if option_1 equals "no".lower():
            return user_type
```

Figure 1.10.3 shows pseudocode for add_new_user function.

-1.11 DELETE USER

```python
def delete_user(user_type):
    userdata = []
    with open(user_file, 'r') as filehandler:
        for data in filehandler.readlines():
            userdata.append(data.strip())

    username = input("Enter the user you want to delete: ")

    userdata = [item for item in userdata if not item.startswith(username)]
    print("User deleted..")

    with open(user_file, 'w') as filehandler:
        filehandler.write('\n'.join(userdata) + '\n')

    option_1 = input("Would you like to continue?(Yes or No): ")
    if option_1 == "yes".lower():
        return delete_user(user_type)
    if option_1 == "no".lower():
        return user_type
```

Figure 1.11.1 shows delete user function.

Delete item function provides the user ability to remove user from the user file. This function starts with empty list assigned to userdata and opening file as filehandler , after that it uses for loop and .readlines() to go through every line and append the data into userdata using .strip() by removing spacing. To continue, the program request an input from user regrading the desired item the user wants to delete . Next, the program code goes through each item in the userdata list and checks if it starts with the provided username. If it finds an item that matches the username, it excludes that item from the updated userdata list and display “User deleted..”. Finally , the program opens the file and use .write and join the list into strings .To end this program , the user can chose to continue or return menu.

```
def delete_user(user_type):
    userdata = []
    open user file as filehandler:
        for data in filehandler.readlines():
            append data into userdata

    GET username
    display "Enter the user you want to delete: "

    userdata = [item  equals  item in userdata if not item.startswith(username)]
    display "User deleted.."

    with open user_file  as filehandler:
        write userdata into user file using filehandler

    GET option_1
    display "Would you like to continue?(Yes or No): "
    if option_1 equals "yes".lower():
        return delete_user(user_type)
    if option_1 equals "no".lower():
        return user_type
```

Figure 1.11.2 shows pseudocode for delete user function.

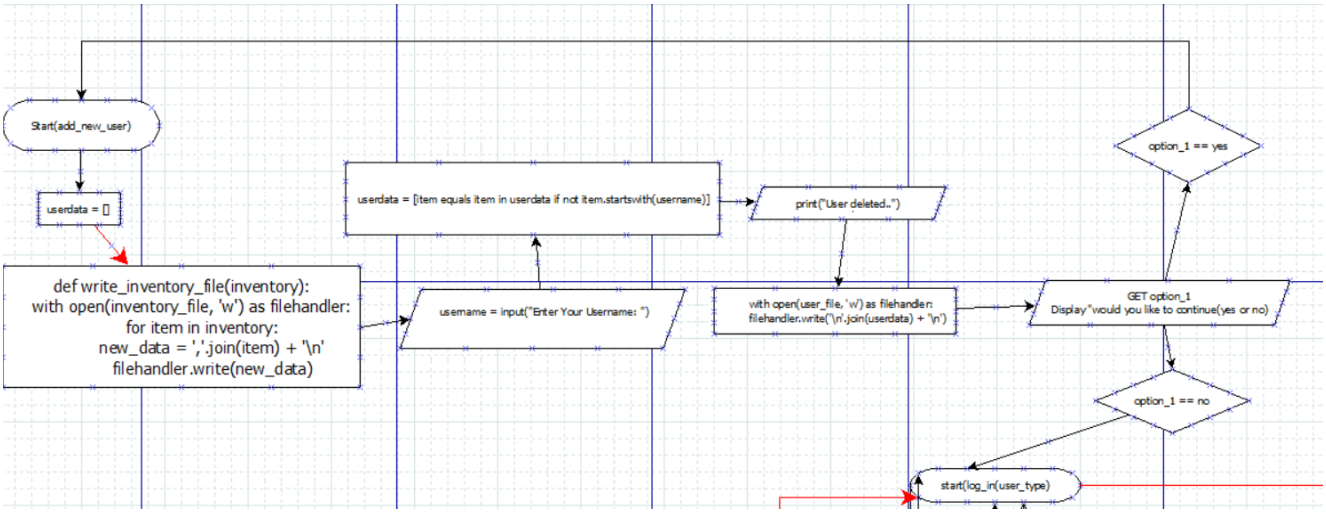

Figure 1.11.3 shows flowchart for delete user function

- 1.12 APPEND INVENTORY

This function is used at item insert function above. This function sole purpose is to take the variable { inventory } and append it into the inventory file. To begin the program open inventory fiule as filehandler. Afterwards, It uses a for loop and for each item inside it uses “,”. join to convert list into string . Finally, it uses the filehandler is utilized to use .write , so that the item is properly added into the inventory.

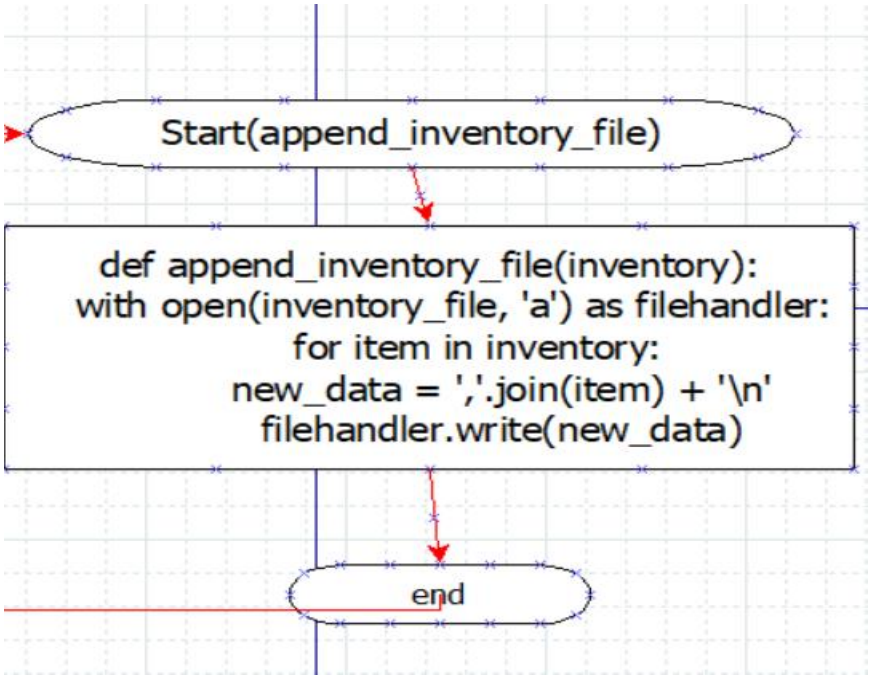

Figure 1.12.1 shows flowchart for append inventory function

```
function append_inventory_file(inventory):
    with open inventory_file as filehandler to append:
        for item in inventory:
            new_data = ','.join(item) + '\n'
            write new_data into filehandler
```

Figure 1.12.2 shows pseudocode for append inventory function

13

- 1.13 WRITE INVENTORY

```
def write_inventory_file(inventory):
    with open(inventory_file, 'w') as filehandler:
        for item in inventory:
            new_data = ','.join(item) + '\n'
            filehandler.write(new_data)
```

Figure 1.13.1 shows write_inventory_file function.

This function is identical to the append inventory function above, this function takes the variable { inventory } from other function and write(replace) the inventory file. This function works by having inventory_file open as filehandler for write, uses for loop and ‘,’.join and assigning it into new_data to finally utilize .write to write it into inventory file.

```
function write_inventory_file(inventory):
    with open inventory_file as filehandler to write:
        for item in inventory:
            new_data = ','.join(item) + '\n'
            write new_data into filehandler
```

Figure 1.13.2 shows pseudocode for write_inventory_function.

-1.14 READ INVENTORY

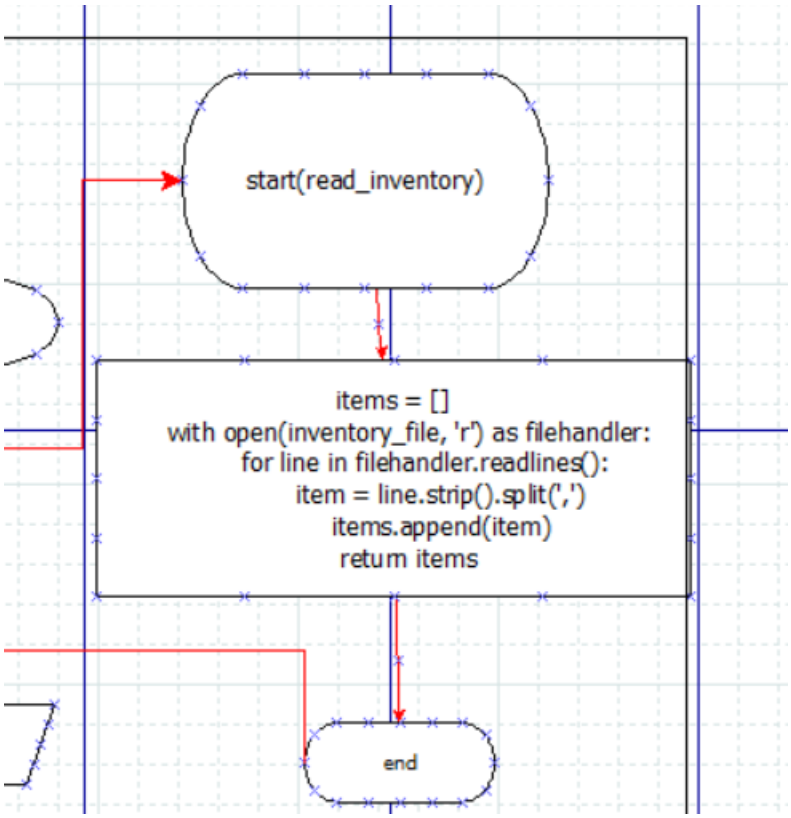

Figure 1.14.1 shows flow chart for read inventory function.

This function provides other functions with the information of the inventory. Firstly, the variable items is assign to an empty list , afterwards the program open inventory_file as filehandler for reading.;it uses for loop, strip and split for managing aesthetics and assigned to item. Finally , it is appended into the empty list above and returned to allows the function to provide its output to the caller.(In this case many of the functions utilized in this program)

```python
def read_inventory():
    items = []
    with open(inventory_file, 'r') as filehandler:
        for line in filehandler.readlines():
            item = line.strip().split(',')
            items.append(item)
    return items
```

Figure 1.14.2 shows source code for read inventory function.

```
function read_inventory():
    items as empty list
    with open inventory_file as filehandler to read:
        for line in filehandler.readlines():
            item = line.strip().split(',')
            append item into items
    return items
```

Figure 1.14.3 shows psuedocode for read inventory function.

-1.15 CONCLUSION

In conclusion, this program uses different functions categorized into each action taken .The program starts with a login function that verifies user credentials and directs them to their respective functionality based on their user type. The admin user type has more extensive privileges, including the ability to modify inventory and user information, while the purchaser user type has limited access.

Throughout the documentation, various functions and their functionalities are described, such as viewing inventory, inserting and deleting items, as well as stock replenishment, searching for items based on different criteria, and managing user information.

This assignment had allowed me to gain deeper knowledge on the programming language ‘ Python ’ ,as well as learning to plan , draw out blue prints using flowcharts and pseudocode that enable me to code more effectively.