

**Module Code & Module Title:**

**CS4051NT Fundamentals of Computing**

**Assessment Weightage & Type:**

**60% Individual Coursework**

**Year and Semester:**

**2023 Spring**

**Student Name: Anish Jaiswal**

**London Met ID: 22085385**

**College ID:NP05CP4S230010**

**Assignment Due Date: 25th August**

**Word Count:**

*I confirm that I understand my coursework needs to be submitted online via MySecondTeacher under the relevant module page before the deadline in order for my assignment to be accepted and marked. I am fully aware that late submissions will be treated as non-submission and a mark of zero will be awarded.*

Contents

[1. Introduction 4](#_Toc143850257)

[1.1 Introduction of Program 4](#_Toc143850258)

[1.2 Objective 5](#_Toc143850259)

[1.3 Tools used for Coursework 6](#_Toc143850260)

[-Vs Code 6](#_Toc143850261)

[-Microsoft Word 6](#_Toc143850262)

[2. Discussion and Analysis 7](#_Toc143850263)

[2.1 Algorithm 7](#_Toc143850264)

[2.2 Flowchart 11](#_Toc143850265)

[2.3 Pseudocode 12](#_Toc143850266)

[2.3.1 Pseudocode for main.py 12](#_Toc143850267)

[2.3.2 Pseudocode for Equipment.py 13](#_Toc143850268)

[2.3.3 Pseudocode for SellEquip.py 15](#_Toc143850269)

[2.3.4 Pseudocode for ReturnEquip.py 18](#_Toc143850270)

[2.4 Data Structure 22](#_Toc143850271)

[3. Program 23](#_Toc143850272)

[4 Testing 29](#_Toc143850273)

[4.1 Test 1 29](#_Toc143850274)

[4.2 Test 2 30](#_Toc143850275)

[4.2.1 test 2.1 30](#_Toc143850276)

[4.2.2 Test 2.2 31](#_Toc143850277)

[4.3 Test 3 33](#_Toc143850278)

[4.4 Test 4 35](#_Toc143850279)

[4.5 Test 5 36](#_Toc143850280)

[Test 5.1 39](#_Toc143850281)

[5. Conclusion 42](#_Toc143850282)

[Figure 1 Screentshot of python 4](#_Toc143850511)

[Figure 2 fig of Vs Code 6](#_Toc143850512)

[Figure 3 fig of MsWord 6](#_Toc143850513)

[Figure 4 Flowchart 11](#_Toc143850514)

[Figure 5 Screenshot of main file 24](#_Toc143850515)

[Figure 6 Screenshot of main fie 24](#_Toc143850516)

[Figure 7 Screenshot of exception handaling 25](#_Toc143850517)

[Figure 8 Update and save of equipments 26](#_Toc143850518)

[Figure 9 update and save equipment for SellEquip 26](#_Toc143850519)

[Figure 10 Update and save equipments for ReturnEquip 27](#_Toc143850520)

[Figure 11 Invoice of SellEquip 27](#_Toc143850521)

[Figure 12 Invoice of SellEquip in file 27](#_Toc143850522)

[Figure 13 Invoice of ReturnEquip 27](#_Toc143850523)

[Figure 14 Invoice of ReturnEquip in file 28](#_Toc143850524)

[Figure 15 Test 1 29](#_Toc143850525)

[Figure 16 test 2.1 30](#_Toc143850526)

[Figure 17 test 2.1 30](#_Toc143850527)

[Figure 18 Test 2.2 31](#_Toc143850528)

[Figure 19 test 2.2 32](#_Toc143850529)

[Figure 20 Renting Equipment for test 3 33](#_Toc143850530)

[Figure 21 Invoice of rented equipment of Test 3 34](#_Toc143850531)

[Figure 22 Returning process Test 4 35](#_Toc143850532)

[Figure 23 Invoice of Test 4 36](#_Toc143850533)

[Figure 24 data of txt file Test 5 36](#_Toc143850534)

[Figure 25 Renting Equipment Test 5 37](#_Toc143850535)

[Figure 26 Invoice of Rented Equipment Test 5 37](#_Toc143850536)

[Figure 27 Table of Decreased quantity Test 5 38](#_Toc143850537)

[Figure 28 Decreased Quantity Txt file Test 5 38](#_Toc143850538)

[Figure 29 txt file Test 5.1 39](#_Toc143850539)

[Figure 30 Returning item Test 5.1 39](#_Toc143850540)

[Figure 31 Invoice of Returned item Test 5.1 40](#_Toc143850541)

[Figure 32 View the item Test 5.1 40](#_Toc143850542)

[Figure 33 Updated Txt file Test5.1 41](#_Toc143850543)

# 1. Introduction

## 1.1 Introduction of Program



Figure 1 Screentshot of python

I was assigned to create a python application that automates the creation of invoices for a equipment rental shop which was the part of my fundamental of computing curriculum. The software had to update the stock of every item in real-time, maintain an extensive database of all of them, and create invoices each time the transaction was made. To do this, I created a Python application that streamlines the Event Equipment Rental Shop's sales and inventory management processes. The shop can effectively manage its inventory with the help of this software, which also allows it to produce invoices and receipts and keep precise records of all transactions.

The software has a sizable database that minutely specifies, details every item of equipment in the store, including its name, brand, price, and quantity that is accessible. Flexibility is increased by the ease with which equipment details can be added or removed. The software automatically creates a professional invoice after a customer rents or returns equipment, capturing crucial transaction facts such the customer's identification, the equipment's specifications, how long the rental will last, and the overall cost.

In short, my program gives the Event Equipment Rental Shop a practical, user-friendly way to create invoices, keep track of transactions, and concentrate on providing outstanding customer care.

## 1.2 Objective

The Event Equipment Rental Shop software was created with the primary goals of revolutionizing the billing process and streamlining transaction tracking. The software's primary features are described in the following objectives:

* To keep a complete database of every piece of equipment that is offered for rental in the store, including with information like name, brand, price, and number.
* To make it simple to add and edit equipment-specific information, allowing for seamless additions and modifications.
* To create expert invoices for each transaction that include crucial information about the client, the equipment rented, how long the rental will last, and the total cost.
* To rapidly alter stock levels in response to rentals and returns in order to guarantee that the store always has an accurate and up-to-date inventory of equipment.
* To offer a user-friendly, intuitive interface that enables store owners and employees to effectively manage business processes in the equipment rental industry.

## 1.3 Tools used for Coursework

### -Vs Code



Figure 2 fig of Vs Code

Microsoft created the free open source text editor known as Visual Studio Code (often referred to as VS Code). Linux, macOS, and Windows all support VS Code. VS Code has recently become one of the most widely used development environment tools, despite the editor's relatively modest weight and robust functionality.

VS Code supports a wide range of programming languages, including CSS, Go, and Dockerfile in addition to Java, C++, and Python. In addition, VS Code enables you to add new extensions like code linters, debuggers, and support for cloud and web development.

### -Microsoft Word



Figure 3 fig of MsWord

The word processing program Microsoft Word is produced by the Microsoft Corporation. Different styles, typefaces, designs, etc. can be used to create, modify, and write files and documents. Microsoft Word supports docx, doc, pdf, rtf files, making it incredibly practical and simple to use.

# 2. Discussion and Analysis

## 2.1 Algorithm

Step 1: Start

Step 2: Import the Equipment, SellEquip, and ReturnEquip modules

Step 3: Define a function named main()

Step 4: Print the welcome message for the event equipment rental shop

Step 5: Input the user's name and store it in the variable user\_name

Step 6: Start an infinite loop for user interaction

Step 7: Display the option for View The Item, Borrow The Item, Return The Item, Exit

Step 8: Ask the user to select an option and store it in a variable called option

* If option not from (1-4) print Enter a valid option

Step 9 : if option is 1, call the display\_equipment() function from Equipment module

* Import the tabulate module
* Define a function named load\_equipment\_data(file\_path)
* Initialize an empty dictionary named equipment
* Open the file at file\_path in read mode and loop through each line
* Split the line by commas and store the values in a list named values
* Extract equipment details from values and store them in variables
* Convert equipment\_id from the first element to an integer
* Assign name, brand, price, and quantity from the subsequent elements
* Create a sub-dictionary with keys "name", "brand", "price", and "quantity", and values corresponding to the extracted variables
* Add the sub-dictionary to the equipment dictionary with the equipment\_id as the key
* Return the equipment dictionary
* Define a function named display\_equipment()
* Print the header "Equipment Inventory"
* Call the load\_equipment\_data("data.txt") function and store the result in the equipment\_data dictionary
* Define headers for the table: "ID", "Name", "Brand", "Price ($)", and "Quantity"
* Initialize an empty list named equipment\_display
* Loop through each equipment in equipment\_data
* Extract equipment details from equipment\_data and store them in variables
  + Assign name, brand, price, and quantity from the equipment details
* Append the extracted details to the equipment\_display list
* Format the equipment\_display list into a table using tabulate
* Print the formatted table

Step 10: If option is 2, call the sell\_equipment() function from the SellEqup module

* Import the datetime module, tabulate module, and the Equipment module
* Define a function named sell\_equipment()
* Print the header "Available Equipment:"
* Call the Equipment.load\_equipment\_data("data.txt") function and store the result in the equipment dictionary
* Call the display\_equipment(equipment) function to display available equipment to the user
* Start an infinite loop
* Input the ID of the item to be sold (or 'exit' to go back) and store it in the variable sold\_item\_id
* If sold\_item\_id is 'exit', break the loop
* If sold\_item\_id is a valid number (digit), execute the following:
  + Convert sold\_item\_id to an integer and store it in the variable equipment\_id
  + If equipment\_id is present in the equipment dictionary:
  + Get the available quantity of the item from the equipment dictionary and store it in available\_quantity
  + If available\_quantity is greater than 0:
  + Input the customer's name and store it in customer\_name
  + Call get\_quantity\_to\_sell(available\_quantity) and store the value in quantity\_to\_sell
  + If quantity\_to\_sell is less than or equal to available\_quantity:
  + Call sell\_item(equipment, equipment\_id, customer\_name, quantity\_to\_sell)
  + Call Equipment.update\_quantity(equipment, equipment\_id, -quantity\_to\_sell)
  + If quantity\_to\_sell is greater than available\_quantity, print "Sorry, insufficient items."
  + If available\_quantity is 0, print "The item is not available for sale."
  + If equipment\_id is not in the equipment dictionary, print "Invalid equipment ID. Please enter a valid ID."
  + End the loop
* Create a formatted string called invoice with the following information:
  + The name of the store and its location
  + The date and time of the purchase using the datetime module
  + The name of the user
  + The product name, quantity and unit price
  + The subtotal, shipping and total amount
  + A thank you message

Step 11 : If option is 2, call the return\_equipment function from the ReturnEquip module

• Import the 'datetime' and 'tabulate' modules

• Import the 'Equipment' module

* Define a function named 'return\_equipment()' with no parameters
* Print "Rented Equipment:"
* Load rented equipment data from "data.txt" using 'load\_equipment\_data()' from 'Equipment' module and assign it to 'rented\_equipment'
* Call 'display\_rented\_equipment(rented\_equipment)' to show rented equipment details
* Enter a loop using 'while True':
* Input 'returned\_item\_id' with prompt "Enter the ID of the item you want to return (or 'exit' to go back): "
* If 'returned\_item\_id' is "exit", exit the loop
* If 'returned\_item\_id' is a digit:
* Convert 'returned\_item\_id' to an integer and assign it to 'equipment\_id'
* If 'equipment\_id' is in 'rented\_equipment':
* Get rented quantity as 'rented\_quantity'
* If 'rented\_quantity' > 0:
* Input 'rental\_duration' with prompt "Enter the number of days the item was rented: "
* Call 'get\_quantity\_to\_return(rented\_quantity)' and assign the result to 'quantity\_to\_return'
* If 'quantity\_to\_return' <= 'rented\_quantity':
* Call 'return\_item(rented\_equipment, equipment\_id, rental\_duration, quantity\_to\_return)' to process the return
* Update rented quantity using 'Equipment.update\_quantity(rented\_equipment, equipment\_id, -quantity\_to\_return)'
* Else, print "Sorry, invalid quantity."
* Else, print "The item was not rented."
* Else, print "Invalid equipment ID. Please enter a valid ID."
* •Define a function named 'display\_rented\_equipment(rented\_equipment)' with 'rented\_equipment' parameter
* Create a table with headers ["ID", "Name", "Brand", "Price ($)", "Rental Duration", "Quantity"]
* Loop through 'rented\_equipment':
* Extract name, brand, price, rental duration, and quantity
* Append these details to the table
* Display the table using 'tabulate'
* Define a function named 'get\_quantity\_to\_return(rented\_quantity)' with 'rented\_quantity' parameter
* Enter a loop using 'while True':
* Try to input 'quantity' with prompt "Enter the quantity you want to return (up to {rented\_quantity} rented): "
* If 'quantity' <= 0, print "Please enter a valid positive quantity."
* Else, return 'quantity'
* Except ValueError, print "Please enter a valid number."
* Define a function named 'return\_item(rented\_equipment, equipment\_id, rental\_duration, quantity\_to\_return)' with parameters 'rented\_equipment', 'equipment\_id', 'rental\_duration', and 'quantity\_to\_return'
* Get item details using 'equipment\_id'
* Extract item's name, brand, price
* Calculate rental charge using 'calculate\_rental\_charge(rental\_duration, item\_price)' and assign it to 'rental\_charge'
* Print "Return Summary:"
* Print equipment name, brand, return date, total quantity returned, rental duration, and total rental charge
* Call 'generate\_invoice()' to create an invoice with the details
* Call 'save\_invoice\_to\_file()' to save the invoice to a file
* Define a function named 'calculate\_rental\_charge(rental\_duration, item\_price)' with parameters 'rental\_duration' and 'item\_price'
* If 'rental\_duration' > 5:
* Calculate extra\_days as 'rental\_duration' - 5
* Return 'item\_price' + (10 \* 'extra\_days')
* Else, return 'item\_price'
* Define a function named 'generate\_invoice(item\_name, item\_brand, quantity\_to\_return, rental\_duration, rental\_charge)' with parameters 'item\_name', 'item\_brand', 'quantity\_to\_return', 'rental\_duration', and 'rental\_charge'
* Create an invoice string with formatted details including invoice number, return date, equipment name, brand, total quantity returned, rental duration, and total rental charge
* Return the invoice
* Define a function named 'save\_invoice\_to\_file(invoice, item\_name)' with parameters 'invoice' and 'item\_name'
* Create a file name using 'item\_name' and current date and time
* Open the file in write mode and write the invoice content
* Close the file
* Print "Invoice generated: {invoice\_file\_name}"
* If the script is run directly (if \_\_name\_\_ == "\_\_main\_\_"):
* Call the 'return\_equipment()' function

Step 12: If the option is 4, display a message Thank you for Visiting Us! And break the loop

Step 13: If the option is anything else, display a message saying to choose a valid option

Step 15: Stop

## 2.2 Flowchart

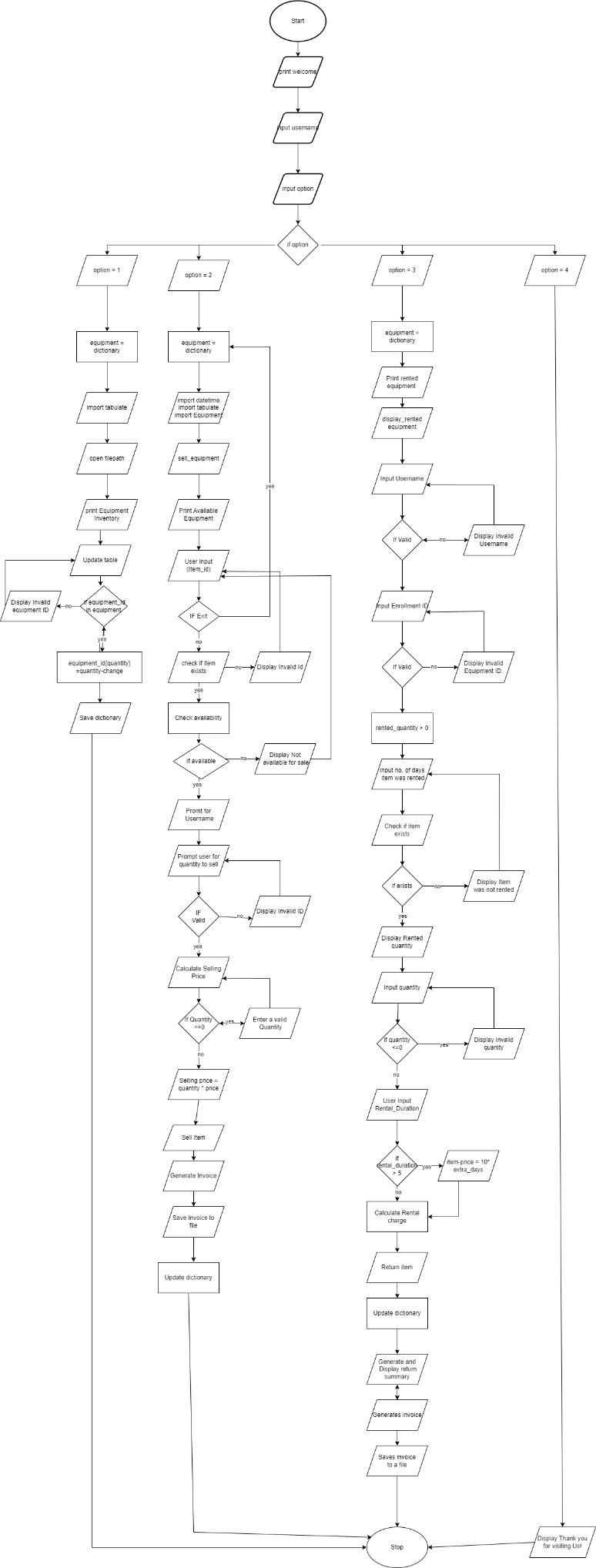


Figure 4 Flowchart

## 2.3 Pseudocode

### 2.3.1 Pseudocode for main.py

**IMPORT** Equipment.py

**IMPORT** SellEquip.py

**IMPORT** ReturnEquip.py

**FUNCTION** main():

welcome\_message = "..."

**OUTPUT** welcome\_message

user\_name = INPUT("Enter your name: ").strip().title()

**WHILE** True:

OUTPUT "Which option would you like to choose:"

OUTPUT "(1) -> View The Item"

OUTPUT "(2) -> Borrow The Item"

OUTPUT "(3) -> Return The Item"

OUTPUT "(4) -> Exit\n"

**TRY**:

OUTPUT "Hello, " + user\_name + "."

option = INPUT("Enter the option you would like to choose: ").strip()

**EXCEPT** ValueError:

OUTPUT "Enter a valid option (1-4)\n"

CONTINUE

**IF** option EQUALS '1':

**CALL** Equipment.display\_equipment()

**ELSE IF** option EQUALS '2':

**CALL** SellEquip.sell\_equipment()

**ELSE IF** option EQUALS '3':

**CALL** ReturnEquip.return\_equipment()

**ELSE IF** option EQUALS '4':

**OUTPUT** "Thank you for Visiting Us!"

**OUTPUT** "\*\*" \* 80

**BREAK**

**ELSE:**

**OUTPUT** "Enter a valid option (1-4)\n"

**END FUNCTION**

**IF** \_\_name\_\_ EQUALS '\_\_main\_\_':

**CALL** main()

### 2.3.2 Pseudocode for Equipment.py

**IMPORT** Tabulate

**FUNCTION** load\_equipment\_data(file\_path):

equipment = an empty dictionary

**WITH** **OPEN** "data.txt" in "r" mode as file:

**FOR** **EACH** line in file:

values = SPLIT line by ","

equipment\_id = CONVERT STRIP values[0] to integer

name = STRIP values[1]

brand = STRIP values[2]

price = CONVERT CONVERT STRIP values[3] by removing "$" to float

quantity = CONVERT STRIP values[4] to integer

equipment[equipment\_id] = {"name": name, "brand": brand, "price": price, "quantity": quantity}

**RETURN** equipment

**FUNCTION** display\_equipment():

**OUTPUT** "Equipment Inventory"

equipment\_data = **CALL** load\_equipment\_data("data.txt")

headers = ["ID", "Name", "Brand", "Price ($)", "Quantity"]

equipment\_display = an empty list

**FOR** **EACH** equipment\_id, details in equipment\_data:

name = details["name"]

brand = details["brand"]

price = details["price"]

quantity = details["quantity"]

**APPEND** [equipment\_id, name, brand, price, quantity] to equipment\_display

table = **CALL** tabulate(equipment\_display, headers=headers, tablefmt="grid")

**OUTPUT** table

**FUNCTION** update\_quantity(equipment, equipment\_id, quantity\_change):

**IF** equipment\_id EXISTS in equipment:

equipment[equipment\_id]["quantity"] += quantity\_change

**ELSE**:

**OUTPUT** "Invalid equipment ID."

**FUNCTION** save\_equipment\_data(equipment, file\_path):

**WITH** **OPEN** "data.txt" in "w" mode as file:

**FOR** **EACH** equipment\_id, details in equipment:

line = equipment\_id + "," + details["name"] + "," + details["brand"] + "," + "$" + details["price"] + "," + details["quantity"] + "\n"

**WRITE** line to file

**IF** \_\_name\_\_ EQUALS "\_\_main\_\_":

CALL display\_equipment()

### 2.3.3 Pseudocode for SellEquip.py

**IMPORT** Equipment.py

**IMPORT** dateTime

From tabulate **IMPORT** tabulate

**FUNCTION** sell\_equipment():

**OUTPUT** "Available Equipment:"

equipment = CALL Equipment.load\_equipment\_data("data.txt")

**CALL** display\_equipment(equipment)

**WHILE** True:

sold\_item\_id = INPUT("Enter the ID of the item you want to sell (or 'exit' to go back): ")

**IF** sold\_item\_id in lowercase EQUALS 'exit':

**BREAK**

**IF** sold\_item\_id is numeric:

equipment\_id = CONVERT sold\_item\_id to integer

**IF** equipment\_id EXISTS in equipment:

available\_quantity = equipment[equipment\_id]["quantity"]

**IF** available\_quantity > 0:

customer\_name = INPUT("Enter customer name: ")

quantity\_to\_sell = CALL get\_quantity\_to\_sell(available\_quantity)

**IF** quantity\_to\_sell <= available\_quantity:

**CALL** sell\_item(equipment, equipment\_id, customer\_name, quantity\_to\_sell)

**CALL** Equipment.update\_quantity(equipment, equipment\_id, -quantity\_to\_sell)

**CALL** Equipment.save\_equipment\_data(equipment, "data.txt") # Save updated data

**ELSE**:

**OUTPUT** "Sorry, insufficient items."

**ELSE**:

**OUTPUT** "The item is not available for sale."

**ELSE**:

**OUTPUT** "Invalid equipment ID. Please enter a valid ID."

**FUNCTION** display\_equipment(equipment):

headers = ["ID", "Name", "Brand", "Price ($)", "Quantity"]

equipment\_display = []

**FOR EACH** equipment\_id, details in equipment:

name = details["name"]

brand = details["brand"]

price = details["price"]

quantity = details["quantity"]

**APPEND** [equipment\_id, name, brand, price, quantity] to equipment\_display

table = CALL tabulate(equipment\_display, headers=headers, tablefmt="grid")

**OUTPUT** table

**FUNCTION** get\_quantity\_to\_sell(available\_quantity):

**WHILE** True:

**TRY**:

quantity = CONVERT INPUT("Enter the quantity you want to sell (up to " + available\_quantity + " available): ") to integer

**IF** quantity <= 0:

**OUTPUT** "Please enter a valid positive quantity."

**ELSE**:

**RETURN** quantity

**EXCEPT** ValueError:

**OUTPUT** "Please enter a valid number."

**FUNCTION** get\_days\_to\_rent():

**WHILE** True:

**TRY**:

            days = int(input("Enter the number of days to rent: "))

**IF** days <= 0:

**OUTPUT** "Please enter a valid number of days.")

**ELSE**:

**RETURN** days

**EXCEPT** ValueError:

**OUTPUT** "Please enter a valid number."

**FUNCTION** sell\_item(equipment, equipment\_id, customer\_name, quantity\_to\_sell):

selling\_price = equipment[equipment\_id]["price"] \* quantity\_to\_sell

**OUTPUT** "\nSale Summary:"

**OUTPUT** "Equipment: " + equipment[equipment\_id]['name']

**OUTPUT** "Brand: " + equipment[equipment\_id]['brand']

**OUTPUT** "Customer: " + customer\_name

**OUTPUT** "Selling Date: " + CURRENT\_DATE\_TIME in specified format

**OUTPUT** "Total Quantity Sold: " + quantity\_to\_sell

**OUTPUT** "Total Selling Price: $" + selling\_price with 2 decimal places

invoice = CALL generate\_invoice(customer\_name, equipment[equipment\_id]['name'], equipment[equipment\_id]['brand'], quantity\_to\_sell, selling\_price)

**CALL** save\_invoice\_to\_file(invoice, customer\_name)

**FUNCTION** generate\_invoice(customer\_name, equipment\_name, brand, quantity\_to\_sell, selling\_price):

invoice = """

**RETURN** invoice

**FUNCTION** save\_invoice\_to\_file(invoice, customer\_name):

invoice\_file\_name = customer\_name + "\_SaleInvoice\_" + CURRENT\_DATE\_TIME in specified format + ".txt"

**WITH** **OPEN** invoice\_file\_name in "w" mode as file:

**WRITE** invoice to file

**OUTPUT** invoice\_file\_name

**IF** \_\_name\_\_ EQUALS "\_\_main\_\_":

**CALL** sell\_equipment()

### 2.3.4 Pseudocode for ReturnEquip.py

**IMPORT** datetime

form tabulate **IMPORT** tabulate

**IMPORT** Equipment.py

**FUNCTION** return\_equipment():

**OUTPUT** "Rented Equipment:"

rented\_equipment = CALL Equipment.load\_equipment\_data("data.txt")

**CALL** display\_rented\_equipment(rented\_equipment)

**WHILE** True:

**TRY**:

user\_name = INPUT("Enter your name: ").strip()

**IF** user\_name is empty:

OUTPUT "/nPlease enter a valid name."

**ELSE**:

**BREAK**

**EXCEPT** Exception:

**OUTPUT** "Please enter a valid name."

**WHILE** True:

**OUTPUT** "Hello " + user\_name

returned\_item\_id = INPUT("Enter the ID of the item you want to return (or 'exit' to go back): ")

**IF** returned\_item\_id in lowercase EQUALS 'exit':

**BREAK**

**IF** returned\_item\_id is numeric:

equipment\_id = CONVERT returned\_item\_id to integer

**IF** equipment\_id EXISTS in rented\_equipment:

rented\_quantity = rented\_equipment[equipment\_id]["quantity"]

**IF** rented\_quantity > 0:

rental\_duration = CONVERT INPUT("Enter the number of days the item was rented: ") to integer

quantity\_to\_return = CALL get\_quantity\_to\_return(rented\_quantity)

**IF** quantity\_to\_return <= rented\_quantity:

**CALL** return\_item(rented\_equipment, equipment\_id, rental\_duration, quantity\_to\_return)

**CALL** Equipment.update\_quantity(rented\_equipment, equipment\_id, quantity\_to\_return) # Increase the quantity

**CALL** Equipment.save\_equipment\_data(rented\_equipment, "data.txt") # Save updated data

**ELSE**:

**OUTPUT** "Sorry, invalid quantity."

**ELSE**:

**OUTPUT** "The item was not rented."

**ELSE**:

**OUTPUT** "Invalid equipment ID. Please enter a valid ID."

**FUNCTION** display\_rented\_equipment(rented\_equipment):

headers = ["ID", "Name", "Brand", "Price ($)", "Rental Duration", "Quantity"]

rented\_display = []

**FOR EACH** equipment\_id, details in rented\_equipment:

name = details["name"]

brand = details["brand"]

price = details["price"]

rental\_duration = details.GET("rental\_duration", 0)

quantity = details["quantity"]

**APPEND** [equipment\_id, name, brand, price, rental\_duration, quantity] to rented\_display

table = **CALL** tabulate(rented\_display, headers=headers, tablefmt="grid")

**OUTPUT** table

**FUNCTION** get\_quantity\_to\_return(rented\_quantity):

**WHILE** True:

**TRY**:

quantity = CONVERT INPUT("Enter the quantity you want to return (up to " + rented\_quantity + " rented): ") to integer

**IF** quantity <= 0:

**OUTPUT** "Please enter a valid positive quantity."

**ELSE**:

**RETURN** quantity

**EXCEPT** ValueError:

**OUTPUT** "Please enter a valid number."

**FUNCTION** return\_item(rented\_equipment, equipment\_id, rental\_duration, quantity\_to\_return):

item = rented\_equipment[equipment\_id]

item\_name = item["name"]

item\_brand = item["brand"]

item\_price = item["price"]

rental\_charge = CALL calculate\_rental\_charge(rental\_duration, item\_price)

**OUTPUT** "\nReturn Summary:"

**OUTPUT** "Equipment: " + item\_name

**OUTPUT** "Brand: " + item\_brand

**OUTPUT** "Return Date: " + CURRENT\_DATE\_TIME in specified format

**OUTPUT** "Total Quantity Returned: " + quantity\_to\_return

**OUTPUT** "Rental Duration: " + rental\_duration + " days"

**OUTPUT** "Total Rental Charge: $" + rental\_charge with 2 decimal places

OUTPUT”Fine for ““rental\_duration”-5”:$fine amount

rented\_equipment[equipment\_id]["quantity"] -= quantity\_to\_return

**CALL** Equipment.update\_quantity(rented\_equipment, equipment\_id, -quantity\_to\_return)

**CALL** Equipment.save\_equipment\_data(rented\_equipment, "data.txt”

invoice = **CALL** generate\_invoice(item\_name, item\_brand, quantity\_to\_return, rental\_duration, rental\_charge)

**CALL** save\_invoice\_to\_file(invoice, item\_name)

**FUNCTION** calculate\_rental\_charge(rental\_duration, item\_price):

**IF** rental\_duration > 5:

extra\_days = rental\_duration - 5

**RETURN** item\_price + (10 \* extra\_days)

**ELSE**:

**RETURN** item\_price

**FUNCTION** generate\_invoice(item\_name, item\_brand, quantity\_to\_return, rental\_duration, rental\_charge):

invoice = """

**RETURN** invoice

**FUNCTION** save\_invoice\_to\_file(invoice, item\_name):

invoice\_file\_name = item\_name + "\_ReturnInvoice\_" + CURRENT\_DATE\_TIME in specified format + ".txt"

**WITH OPEN** invoice\_file\_name in "w" mode as file:

**WRITE** invoice to file

**OUTPUT** "Invoice generated: Your invoice is generated in " + invoice\_file\_name + " file"

**IF** \_\_name\_\_ EQUALS "\_\_main\_\_":

**CALL** return\_equipment()

## 2.4 Data Structure

Data structures in Python are methods to arrange and keep data in memory so that it may be accessed and utilized effectively. They serve as containers for groups of related data items, such as strings, lists, and integers, and they also give users a mechanism to manipulate the data. Data structures including lists, tuples, sets, and dictionaries are included in Python.

Lists: a group of components with a fixed arrangement that is flexible. Lists allow for duplication and are mutable.

Tuples: In contrast to lists, they are immutable, which means that once generated, they cannot be changed. Duplicates are also acceptable.

Sets: A group of distinct, variable items that are not ordered. Sets prevent redundancy.

Dictionaries: a set of mutable key-value pairs that are not ordered. The keys you use to access values in dictionaries must all be distinct.

To store information about equipment, the code use a hierarchical dictionary data structure. There is a list of dictionaries that contain information on equipment. A dictionary is a group of key-value pairs, where each key has a specific value assigned to it. The outer dictionary in this instance uses the Equipment ID of equipment as keys, and each associated value is a different dictionary that provides information about the equipment.

The name, brand, price, and quantity available are the matching values for the keys “name”, "brand," "price," and "quantity" in the inner dictionary used to store information on the equipment.

The code offers an easy and effective approach to store and retrieve data about several equipment by utilizing a nested dictionary. Each equipments has a special name that can be used as a key to access its characteristics, and the inner dictionary makes it simple to find out about particular features of the equipment.

# 3. Program

The provided code represents a program that allows users to rent or return equipments from “Event Equipment Rental Shop”. Here’s a description of how the program works:

The main() function serves as the entry point for the program. It starts by printing welcome message and prompting the user to enter their name. Then, it enters a while loop that keeps running until the user chooses to exit the program.

Within the loop, a menu is displayed with four options: Viewing the equipment, Renting the equipment, returning the equipment, or exiting the program. The user is asked to select an option by entering the corresponding number. Based on the selected option, the program performs the following actions:

If option ‘1’ is chosen, the display\_equipment() function is called. This function is responsible for displaying the equipment inside the table in a tabular form and updates the table when a renting or returning process is made.

If option ‘2’ is chosen, the sell\_equipment() function is called. This function is responsible for handling the renting process, such as displaying the equipments, asking the user input’s, generating an invoice, and updating the inventory.

If option “3” is chosen the return\_equipment() function is called. This function handles the returning process, including displaying equipments, taking input from user, generating an invoice, and updating the inventory.

If option '4' is chosen, the program displays a message indicating that it is exiting and breaks out of the loop, terminating the program.

If the user enters an invalid option, a message is displayed, prompting them to choose a valid option.

The program follows a modular approach, utilizing the RentEquip and ReturnEquip modules to encapsulate the specific functionality for renting and returning equipments, respectively. These modules should contain the actual implementation of the renting and returning processes, including interactions with the user and necessary data manipulation.

Overall, the program provides a simple interface for users to interact with the Event Equipment Rental Shop, enabling them to rent or return equipments based on their preferences.

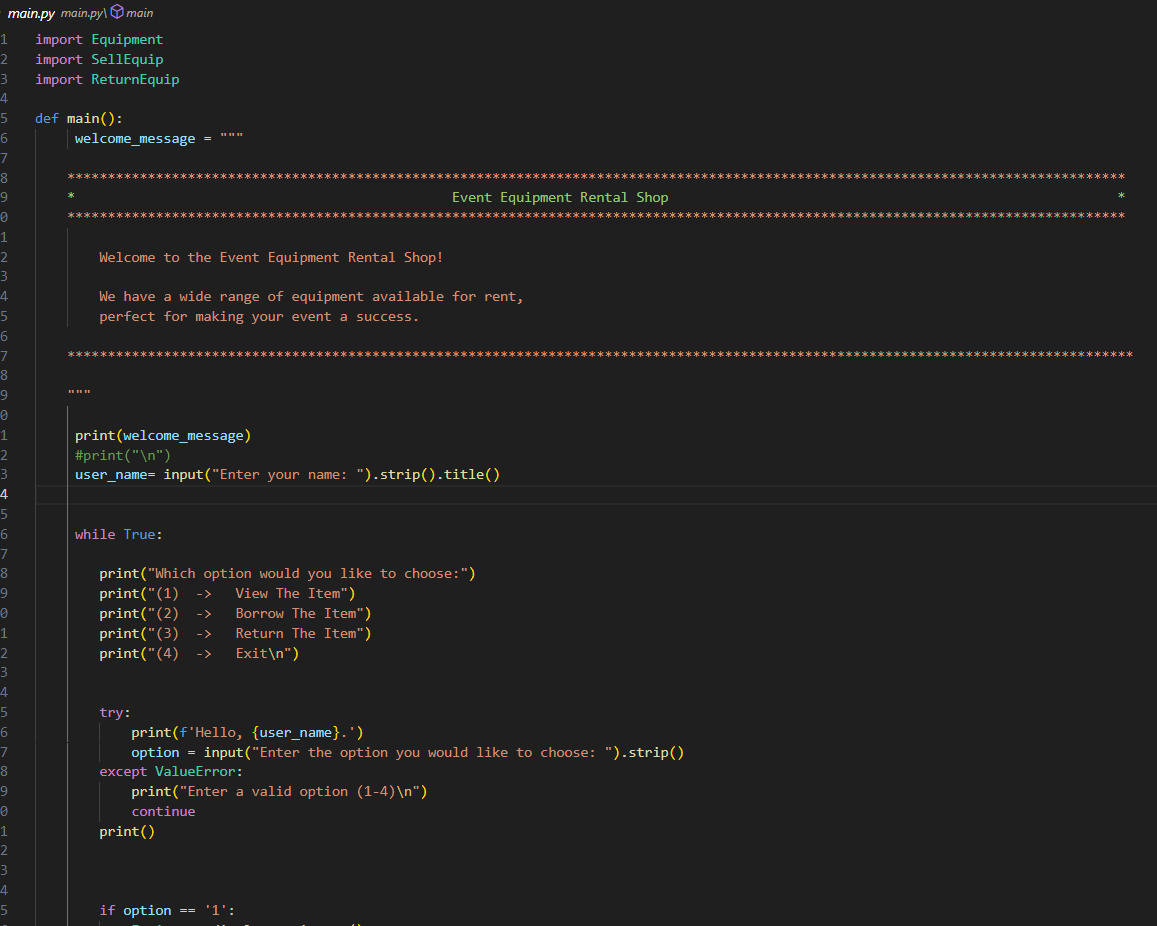


Figure 5 Screenshot of main file



Figure 6 Screenshot of main fie

It can handle any exception if users inputs invalid input, it will display an error message and makes the user to rewrite the correct input of the data

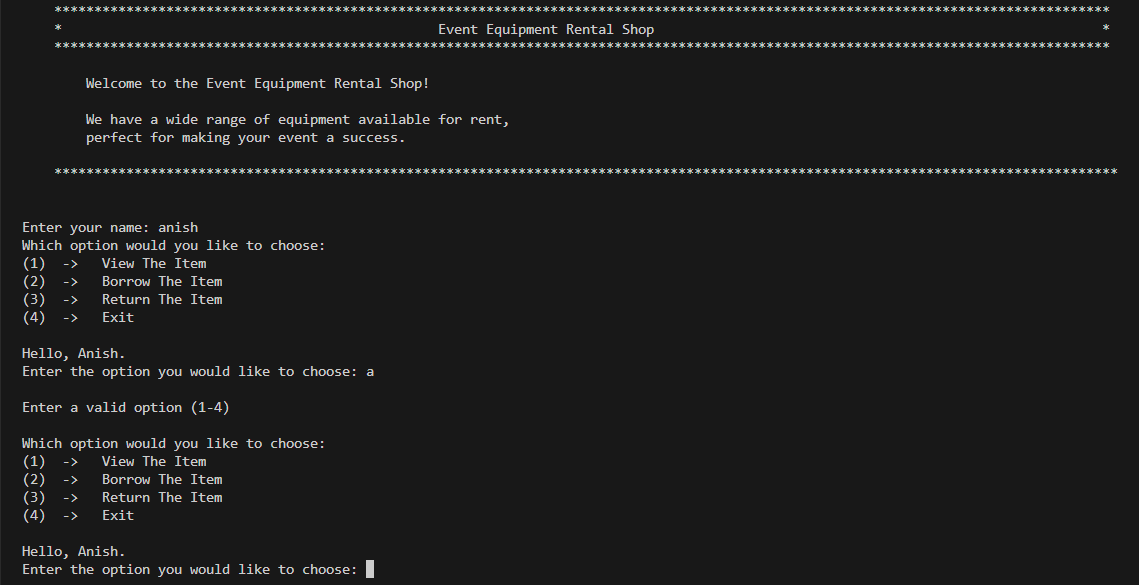


Figure 7 Screenshot of exception handaling

Here user is asked to enter option between 1 to 4 but string value is entered, The program without crashing gave an error message and and asked the user to rewrite the option. In my program the exception are handaled properly.

The main() function code snippet is a function that asks the user about equipments in a remanage text file when a equipment is sold. Here's an explanation of how the program works:

The Equipment module function reads equipment data from the specified file path ("data.txt"). It creates an empty dictionary named equipment to store equipment details. It then opens the file and iterates over each line. If the line is not empty (using if line:), it splits the line into individual values and extracts information such as equipment ID, name, brand, price, and quantity. This information is stored in the equipment dictionary with the equipment ID as the key.

The function display\_equipment() displays the equipment inventory using the loaded equipment data.

The function update\_quantity(equipment, equipment\_id, quantity\_change) is used to update the quantity of a specific equipment item. It takes the equipment dictionary, the equipment's ID, and the quantity change (positive for addition, negative for subtraction) as arguments. If the equipment ID is valid (exists in the equipment dictionary), the script updates the quantity of that equipment item accordingly.

The function save\_equipment\_data(equipment, file\_path) is responsible for saving the updated equipment data back to the file. It opens the file in write mode and iterates over the equipment items. For each item, it constructs a line with the equipment ID, name, brand, price, and quantity, and writes it to the file.

The Equipment module provides functions to manage equipment inventory, display the inventory to users, update equipment quantities, and save the updated data back to a file.

To return equipment

Using the file object as file, the function writes a formatted string to a file, containing the equipments details separated by commas. Each line represents a single equipment and follows the format:

“name”. “brand”, “price” , “quantity”. The \n character is used to move to the next line after writing each equipment's details.

After writing all the equipment’s information, the function concludes by returning the equipment dictionary.

Overall, this function updates and save the "data.txt" file with the updated quantity of equipment after a sale. It ensures that the inventory information is synchronized with the changes made during the renting process

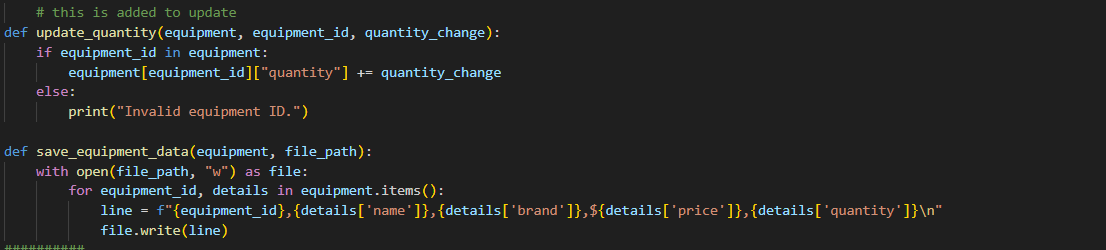


Figure 8 Update and save of equipments

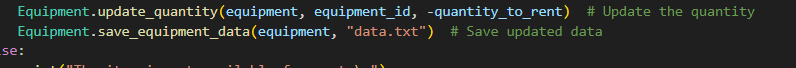


Figure 9 update and save equipment for SellEquip

For returning too the above function is used with different update value

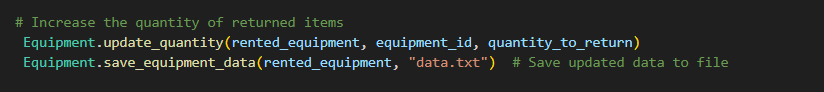


Figure 10 Update and save equipments for ReturnEquip

After updating the equipment’s quantities in the file, the function proceeds to return two values: invoice and item\_name. The invoice variable holds a string containing the invoice details, while the item\_name variable is the updated dictionary of equipment’s information.

In summary, this function updates the "data.txt" file with the modified quantities of equipment’s after a returned. It ensures that the inventory information is synchronized with the changes made during the returning process. Additionally, it returns the invoice details and the updated equipment dictionary for further processing or display.

Here are the screenshots of both rent and return invoice on how it is generated

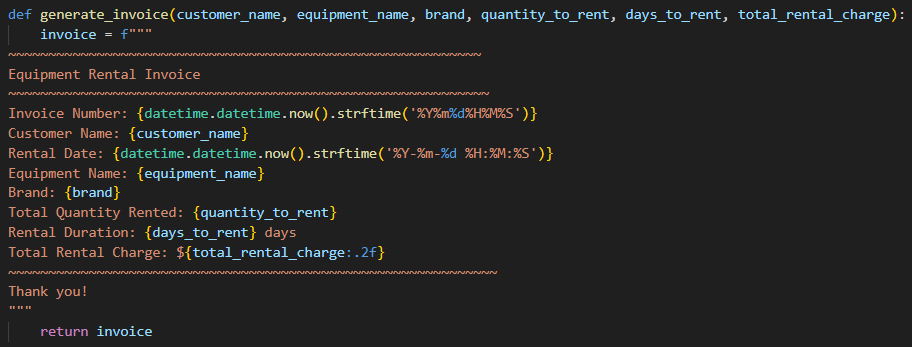


Figure 11 Invoice of SellEquip

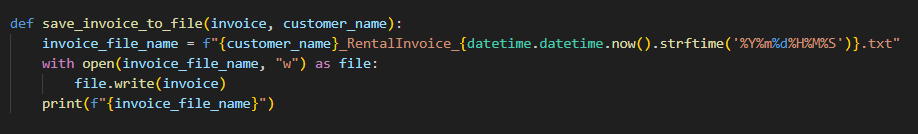


Figure 12 Invoice of SellEquip in file

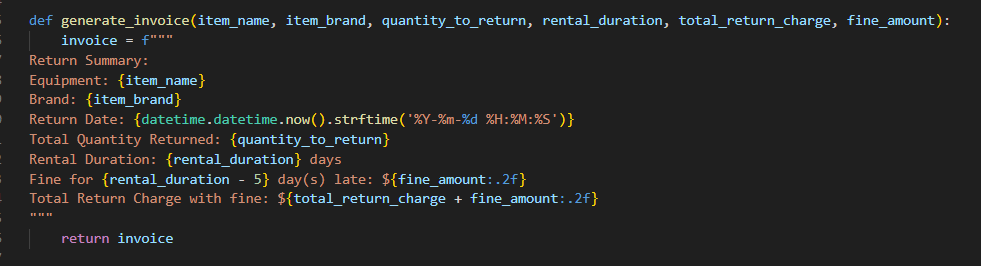


Figure 13 Invoice of ReturnEquip

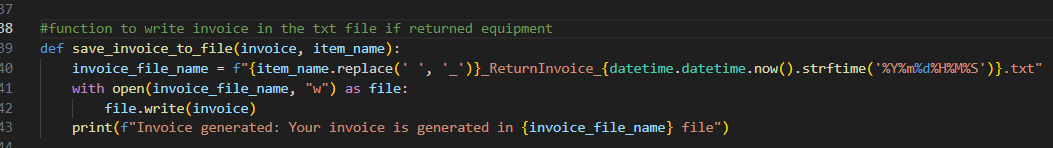


Figure 14 Invoice of ReturnEquip in file

# 4 Testing

## 4.1 Test 1

Table 1 To analyze the use of try and except

|  |  |
| --- | --- |
| objective | To analyze the use of try and except. |
| Action | Invalid input to be put when choosing option |
| Expected Result | The appropriate message should be shown |
| Actual Result | The appropriate message was shown |
| Conclusion | Test is Successful |

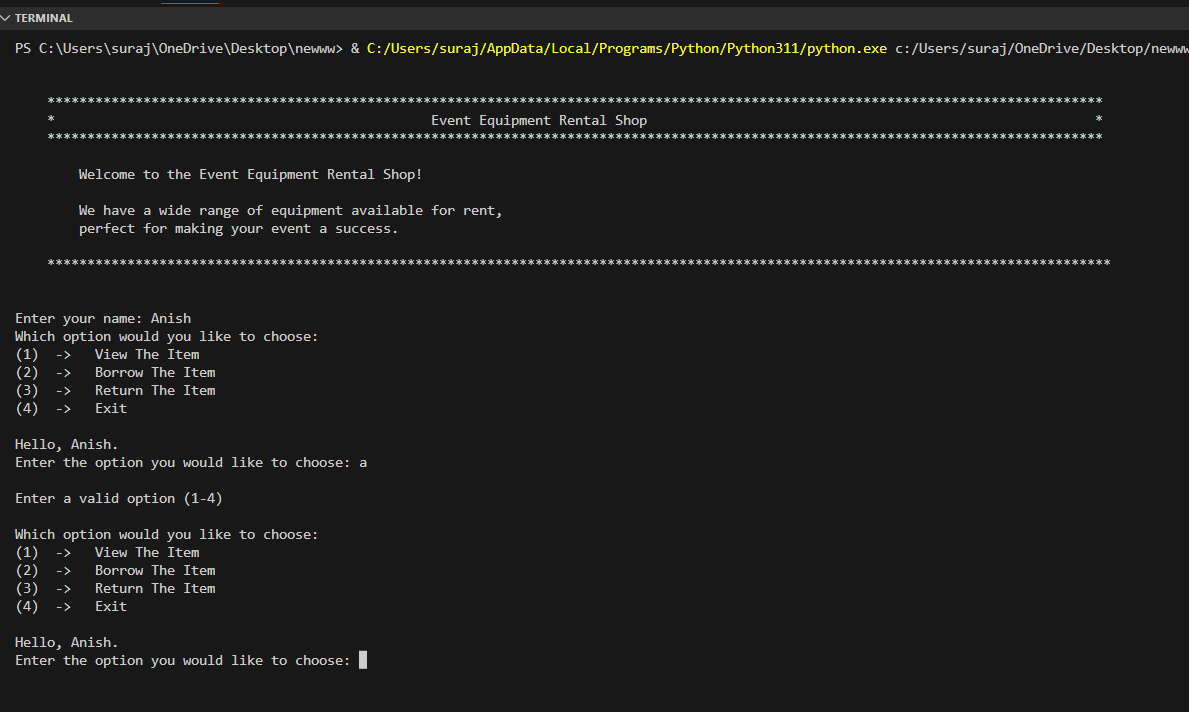


Figure 15 Test 1

## 4.2 Test 2

### 4.2.1 test 2.1

Table 2 test 2.1

|  |  |
| --- | --- |
| objective | To check the input’s quantity value when Renting |
| Action | 1. A negative value was input.  2. Unknown value was entered |
| Expected Result | The appropriate error message should be shown |
| Actual Result | The appropriate message was shown |
| Conclusion | Test is Successful |

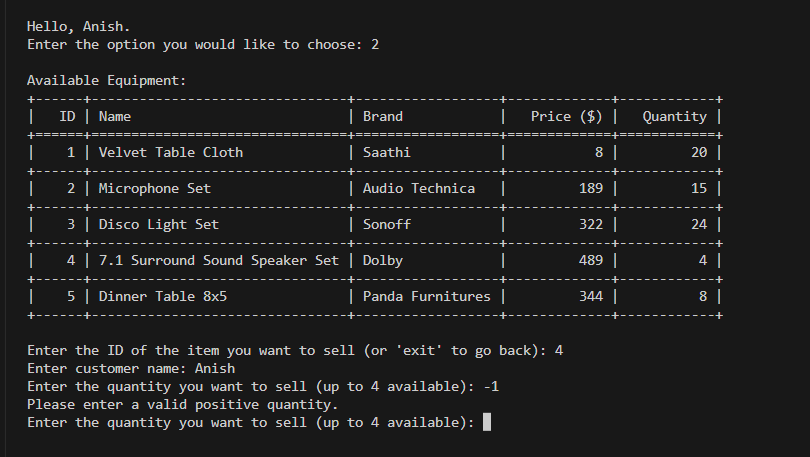


Figure 16 test 2.1

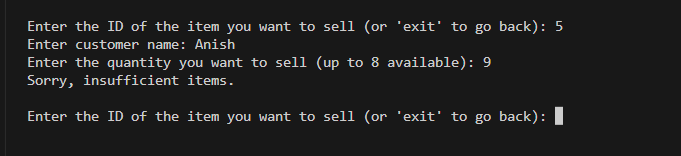


Figure 17 test 2.1

### 4.2.2 Test 2.2

Table 3 Test 2.2

|  |  |
| --- | --- |
| objective | To check the input’s quantity value when Returning |
| Action | 1. A negative value was input.  2. Unknown value was entered |
| Expected Result | The appropriate error message should be shown |
| Actual Result | The appropriate message was shown |
| Conclusion | Test is Successful |

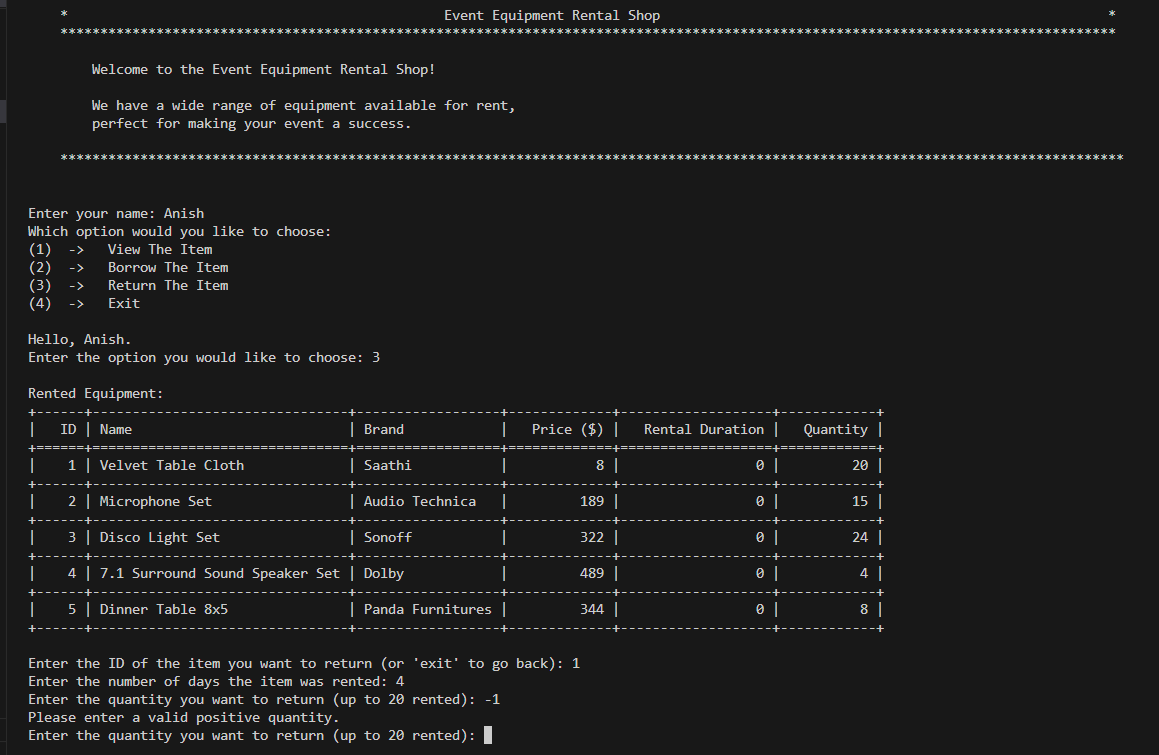


Figure 18 Test 2.2

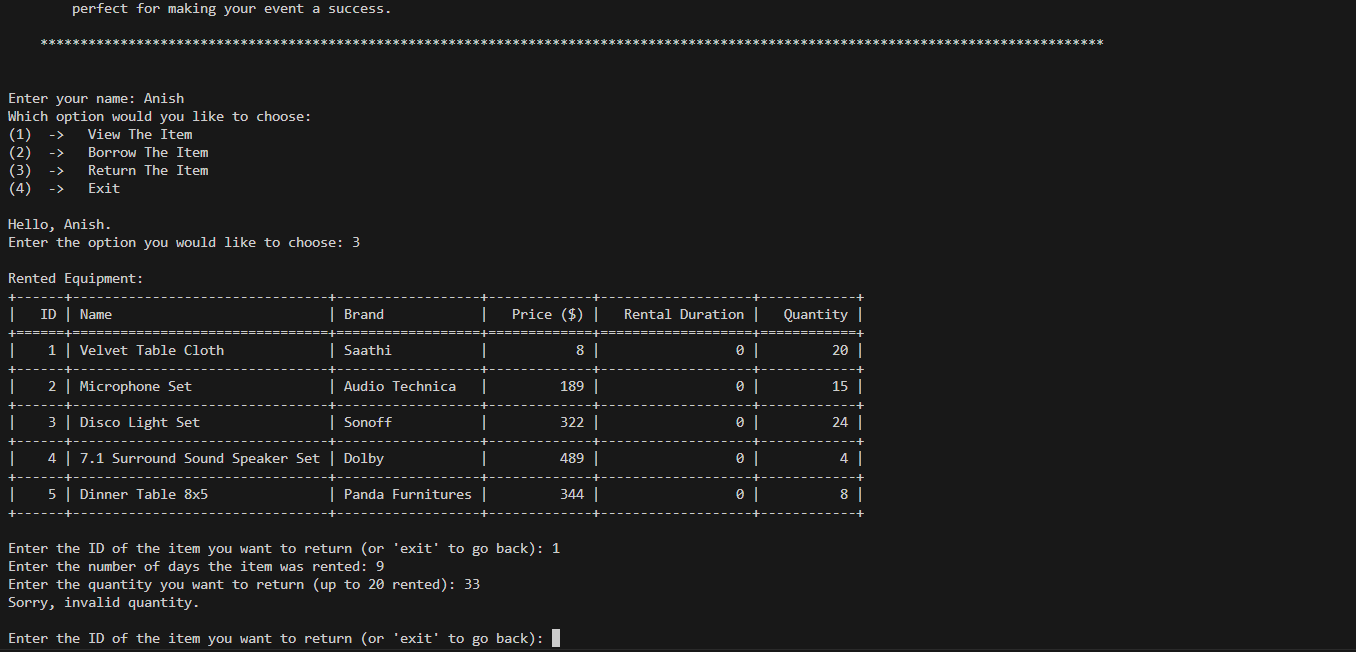


Figure 19 test 2.2

## 4.3 Test 3

Table 4 Test 3

|  |  |
| --- | --- |
| objective | To display the Renting process with invoice. |
| Action | Each and every request made for the user is filled. |
| Expected Result | Invoice of rented equipment is to be shown |
| Actual Result | Invoice of rented equipment was shown |
| Conclusion | Test is Successful |

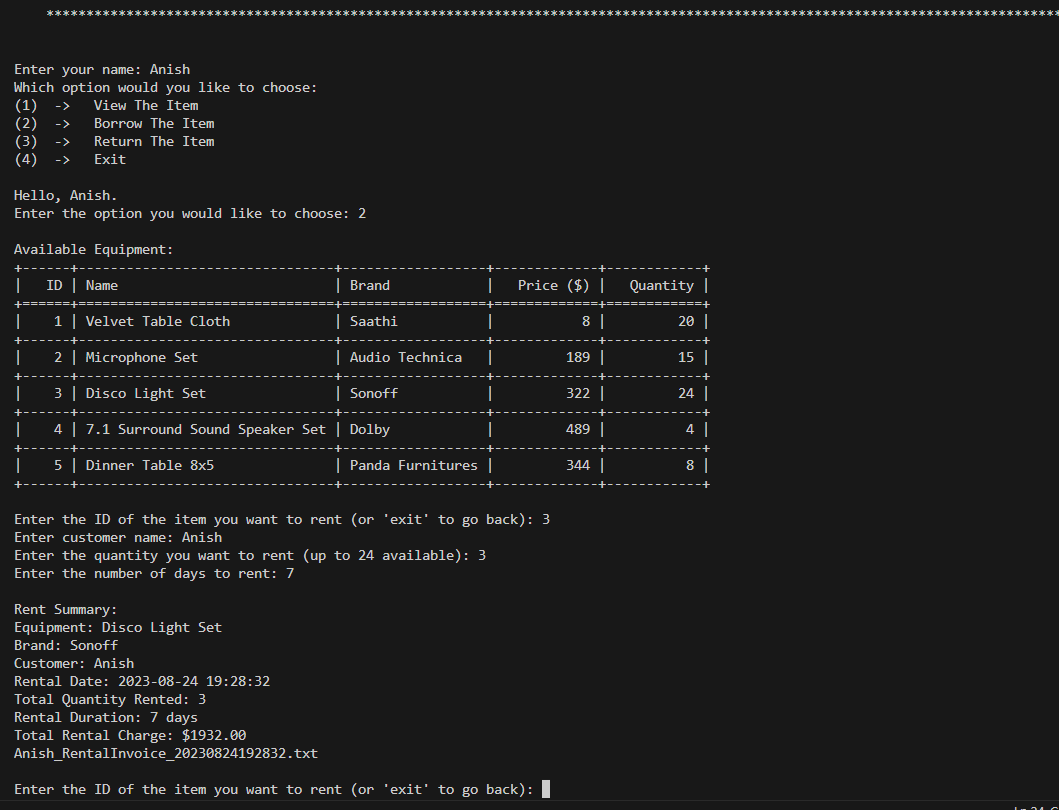


Figure 20 Renting Equipment for test 3

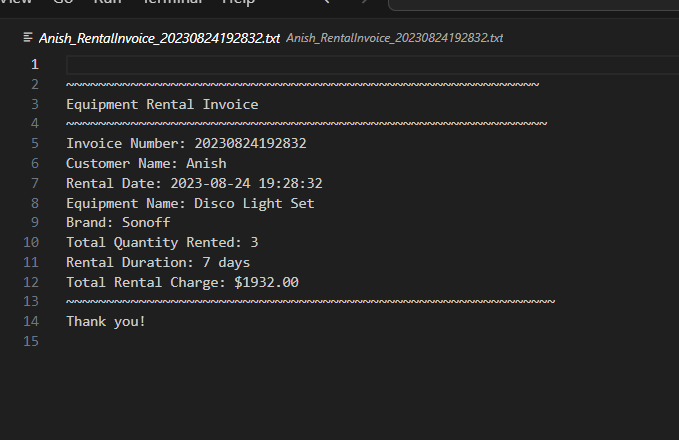


Figure 21 Invoice of rented equipment of Test 3

## 4.4 Test 4

Table 5 Test 4

|  |  |
| --- | --- |
| objective | To display the Returning process with invoice. |
| Action | Each and every request made for the user is filled. |
| Expected Result | Invoice of returned equipment is to be shown |
| Actual Result | Invoice of returned equipment was shown |
| Conclusion | Test is Successful |

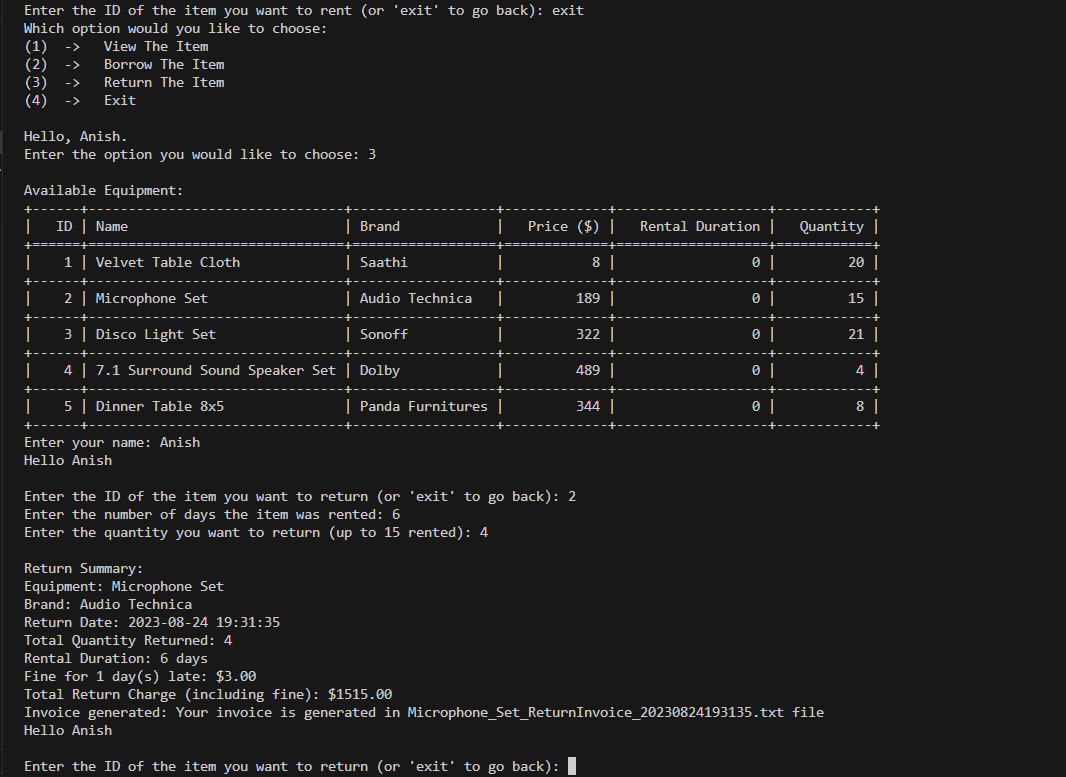


Figure 22 Returning process Test 4

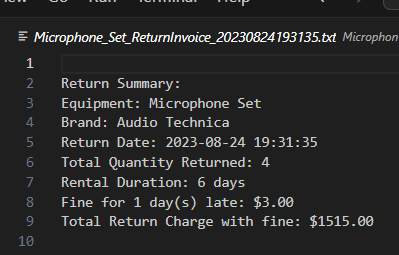


Figure 23 Invoice of Test 4

## 4.5 Test 5

Table 6 Test 5

|  |  |
| --- | --- |
| objective | To check the updates in stock items of RentEquip file |
| Action | A item was sold, and the product was examined both before and after the sale. |
| Expected Result | As items are sold, their supply should decrease. |
| Actual Result | As the items were sold, the supply droped. |
| Conclusion | Test is Successful |

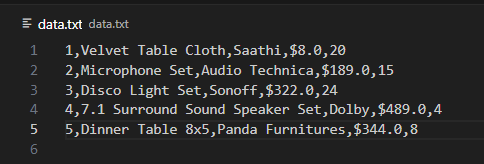


Figure 24 data of txt file Test 5

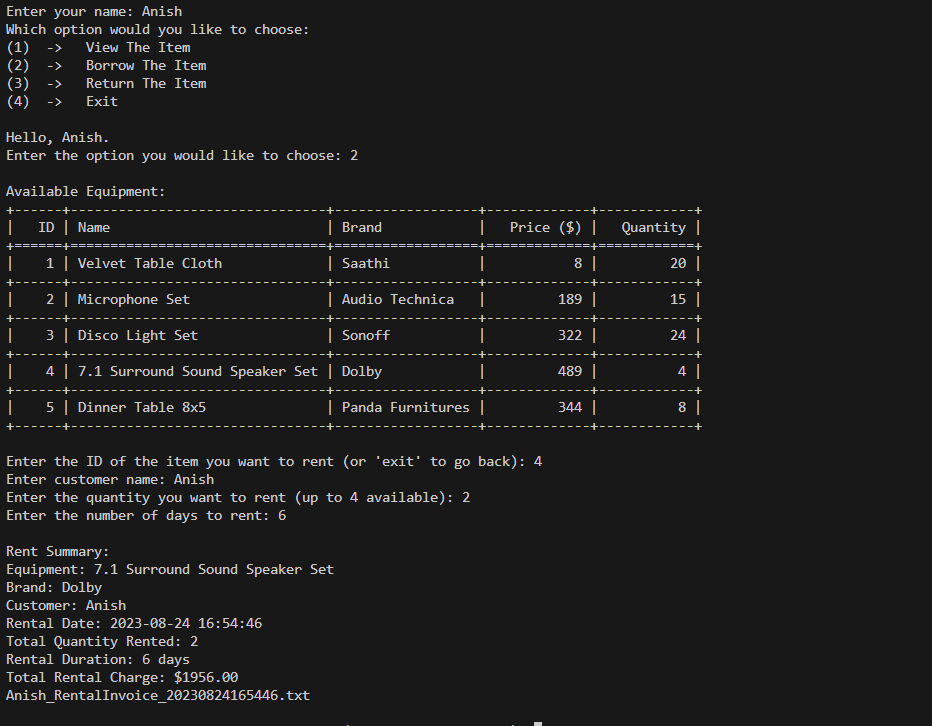


Figure 25 Renting Equipment Test 5

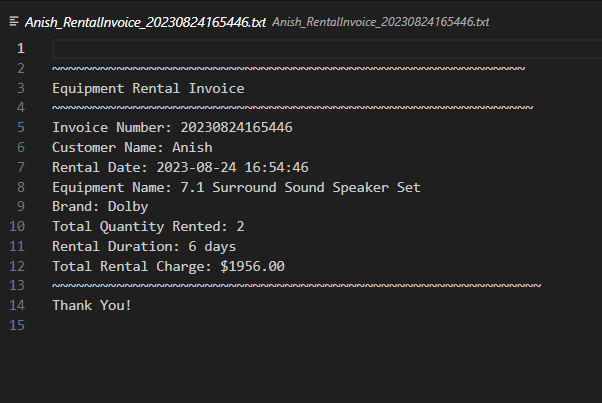


Figure 26 Invoice of Rented Equipment Test 5

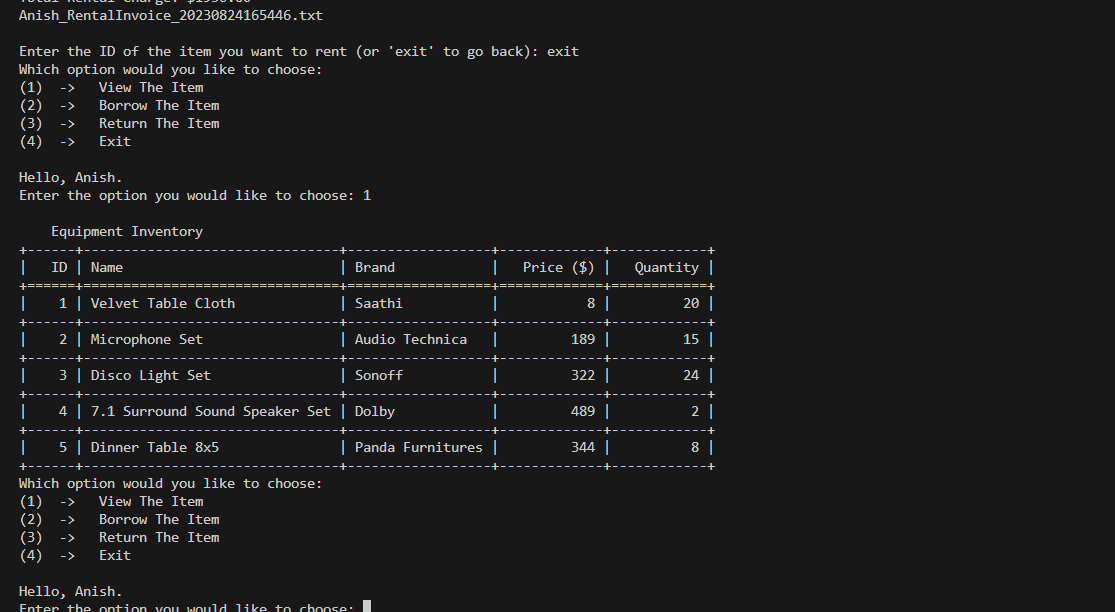


Figure 27 Table of Decreased quantity Test 5

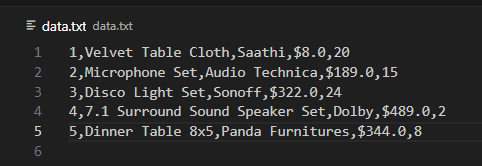


Figure 28 Decreased Quantity Txt file Test 5

### Test 5.1

Table 7 Test 5.1

|  |  |
| --- | --- |
| objective | To check the updates in stock items of ReturnEquip file |
| Action | An item was rented, and the product was examined both before and after until it was returned. |
| Expected Result | As items are returned, their supply should increase. |
| Actual Result | As the items were returned, the supply increased. |
| Conclusion | Test is Successful |

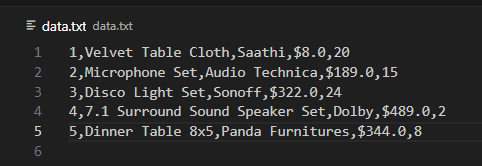


Figure 29 txt file Test 5.1

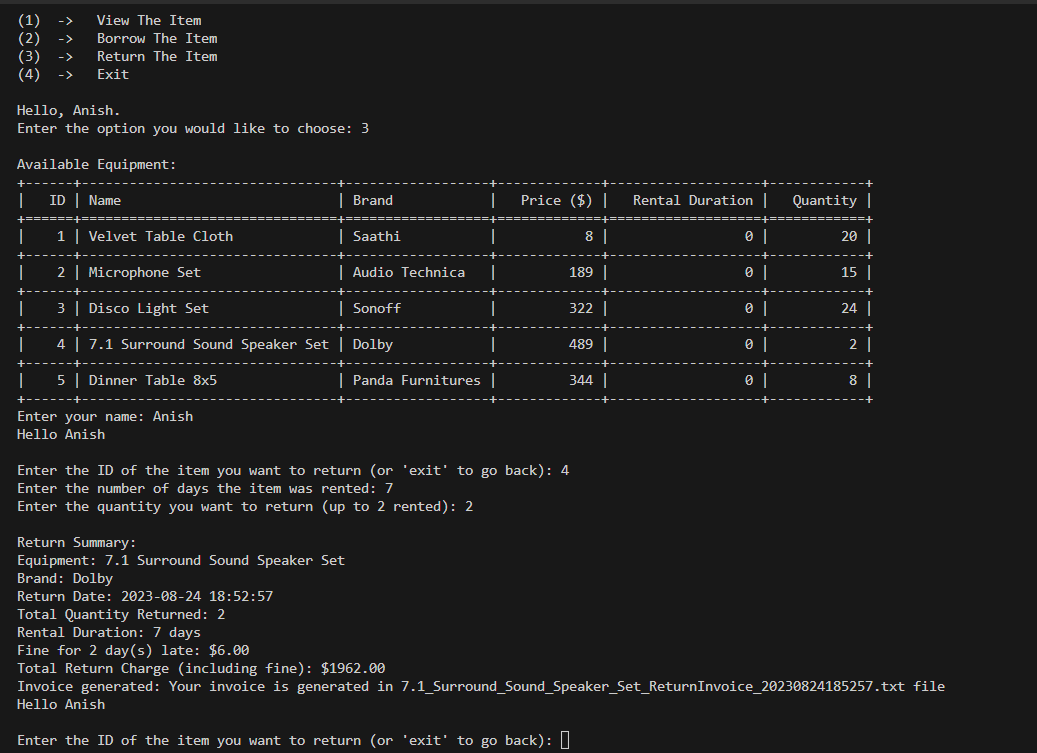


Figure 30 Returning item Test 5.1

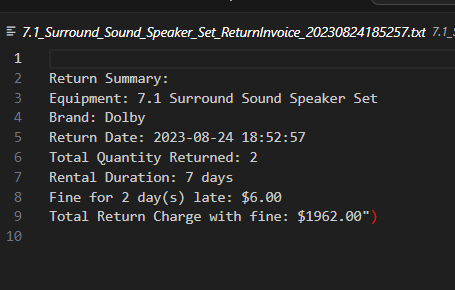


Figure 31 Invoice of Returned item Test 5.1

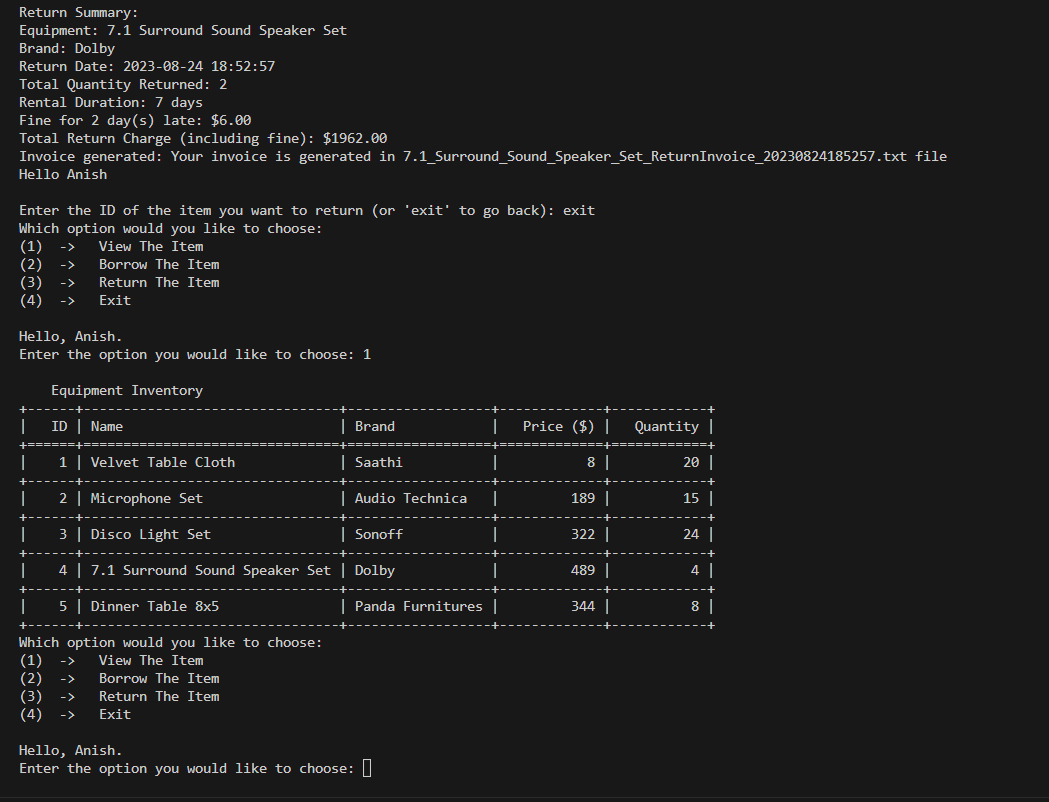


Figure 32 View the item Test 5.1

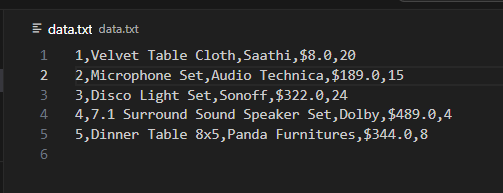


Figure 33 Updated Txt file Test5.1

# 5. Conclusion

After finishing the Fundamentals of Computing program, I have acquired a lot of information and abilities that have really improved my comprehension of programming. I had the chance to develop a thorough Python application just for the coursework because of this experience. This program facilitates the buying and selling of equipment, keeps track of inventories in a text file, and creates thorough invoices for every transaction. Along with improving my understanding of Python programming, the creation of the application gave me a firm base on which to build.

I improved my ability to solve problems by identifying and fixing issues in the application by participating in the program's curriculum. In addition, I have mastered the use of Python's several flexible data structures, like dictionaries and lists, which I anticipate will be crucial for my upcoming projects. My perspective on programming's function across various sectors and industries has been widened by this practical grasp of its real-world applications. My desire to explore more complex programming approaches has risen, driving me to pursue more proficiency in the field of computing.

I am happy with the results after reviewing the coursework. My future pursuits stand to benefit significantly from the skills I have gained. I'm ready to continue my studies in the subject of computing and am excited to apply the knowledge I've learned from this module to my forthcoming programming endeavors. I am well-equipped to manage the rapidly changing technological landscape and make a valuable contribution to the programming community thanks to these foundational abilities.