



**Module Code & Module Title**

**CS4051NI Fundamentals of Computing**

**Assessment Weightage & Type**

**60% Individual Coursework**

**Year and Semester**

**2022-23 Spring**

**Student Name: Reeya Hyoju**

**London Met ID: 22085586**

**College ID: NP01CP4S230090**

**Group: C20**

**Assignment Due Date:** Friday, August 25, 2023

**Assignment Submission Date:** 25th August 2023

*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 marks of zero will be awarded.*

Table of Contents

[Introduction 1](#_Toc143810303)

[1.1. About the project 1](#_Toc143810304)

[1.2. Goals and Objective 1](#_Toc143810305)

[2. Discussion and Analysis 2](#_Toc143810306)

[2.1. Algorithm 2](#_Toc143810307)

[2.2 Flowchart 5](#_Toc143810308)

[2.3 Pseudocode 7](#_Toc143810309)

[2.3.1 Pseudocode for Main.py 7](#_Toc143810310)

[2.3.3 Pseudocode for write.py 9](#_Toc143810311)

[2.4 Data Structures 27](#_Toc143810312)

[2.4.1 Primitive Data Structures 28](#_Toc143810313)

[. Integer 28](#_Toc143810314)

[. Float 28](#_Toc143810315)

[. Boolean 29](#_Toc143810316)

[2.4.2. Non Primitive Data Structures 30](#_Toc143810317)

[. List 30](#_Toc143810318)

[. Tuple 30](#_Toc143810319)

[. Set 30](#_Toc143810320)

[. Dictionary 30](#_Toc143810321)

[3. Program 31](#_Toc143810322)

[3.1. Implementation 31](#_Toc143810323)

[i. Main.py 31](#_Toc143810324)

[ii. Read.py 31](#_Toc143810325)

[iii. Write.py 32](#_Toc143810326)

[iv. Operation.py 32](#_Toc143810327)

[3.2. Renting and Returning Equipment 33](#_Toc143810328)

[3.2.1 Renting Equipment 33](#_Toc143810329)

[3.2.2. Returning Equipment 37](#_Toc143810330)

[3.2.3. Termination of the program 39](#_Toc143810331)

[4. Testing 40](#_Toc143810332)

[4.1. Test 1 40](#_Toc143810333)

[4.2. Test 2 42](#_Toc143810334)

[4.3 Test 3 44](#_Toc143810335)

[4.4. Test 4 48](#_Toc143810336)

[4.5. Test 5 52](#_Toc143810337)

[5 Conclusion 56](#_Toc143810338)

[6, Appendix 57](#_Toc143810339)

[**6.1 Code for main.py** 57](#_Toc143810340)

[**6.2. Code for read.py** 59](#_Toc143810341)

[**6.3 Code for write.py** 60](#_Toc143810342)

[**6.4 Code for operation.py** 69](#_Toc143810343)

[Figure 1:Flow chART 6](#_Toc143810409)

[Figure 2:Data structure 27](#_Toc143810410)

[Figure 3: Integer 28](#_Toc143810411)

[Figure 4:String 29](#_Toc143810412)

[Figure 5:Renting equipment 1 33](#_Toc143810413)

[Figure 6: Renting Equipment 1.1 34](#_Toc143810414)

[Figure 7:Renting Equipment 1.3 34](#_Toc143810415)

[Figure 8:Renting Equipment 1.4 35](#_Toc143810416)

[Figure 9:Renting Equipment 1.5 36](#_Toc143810417)

[Figure 10:Renting Equipment 1.6 36](#_Toc143810418)

[Figure 11:Returning Equipment 1.1 37](#_Toc143810419)

[Figure 12:Returning Equipment 1.2 38](#_Toc143810420)

[Figure 13:Returning Equipment 1.3 38](#_Toc143810421)

[Figure 14:Returning Equipment 1.4 39](#_Toc143810422)

[Figure 15:Termination of the program 39](#_Toc143810423)

[Figure 16:Test 1 40](#_Toc143810424)

[Figure 17:Test 1.1 41](#_Toc143810425)

[Figure 18:Test 1.2 41](#_Toc143810426)

[Figure 19:Test 1.3 41](#_Toc143810427)

[Figure 20:Test 2.1 42](#_Toc143810428)

[Figure 21:Test 2.2 43](#_Toc143810429)

[Figure 22:Test 2.3 43](#_Toc143810430)

[Figure 23:Test 2.4 43](#_Toc143810431)

[Figure 24:Test 3.1 44](#_Toc143810432)

[Figure 25:Test 3.2 45](#_Toc143810433)

[Figure 26:Test 3.3 45](#_Toc143810434)

[Figure 27::Test 3.4 46](#_Toc143810435)

[Figure 28:Test 3.5 46](#_Toc143810436)

[Figure 29::Test 3.6 47](#_Toc143810437)

[Figure 30:Test 3.7 47](#_Toc143810438)

[Figure 31:Test 4.1 49](#_Toc143810439)

[Figure 32:Test 4.2 49](#_Toc143810440)

[Figure 33:Test 4.3 50](#_Toc143810441)

[Figure 34:Test 4.4 50](#_Toc143810442)

[Figure 35:Test 4.5 51](#_Toc143810443)

[Figure 36:Test4.6 51](#_Toc143810444)

[Figure 37:Test 5.1 52](#_Toc143810445)

[Figure 38:Test 5.2 53](#_Toc143810446)

[Figure 39:Test 5.3 53](#_Toc143810447)

[Figure 40:Test 5.4 54](#_Toc143810448)

[Figure 41:Test 5.5 54](#_Toc143810449)

[Figure 42:Test 5.6 55](#_Toc143810450)

[Figure 43:Test 5.7 55](#_Toc143810451)

# Table of Table

[Table 1:Test 1 40](#_Toc143810559)

[Table 2:Test 2 42](#_Toc143810560)

[Table 3:Test 3 44](#_Toc143810561)

[Table 4:Test 4 48](#_Toc143810562)

[Table 5:Test 5 52](#_Toc143810563)

# 

# Introduction

## About the project

This project involves creating an application system for an equipment rental store, which aids in maintaining equipment -related information. This application manages borrower names, tracks equipment loans, and maintains stock quantities. It also generates invoices post-transaction. Implementing this system empowers the equipment rental shop to efficiently manage equipment records, renters' details, returns, and late-return penalties. The system is crafted using the Python programming language within the IDLE platform. Python stands as a widely embraced and adaptable programming language capable of serving diverse functions, including web development, data analysis, automation, and machine learning.

## Goals and Objective

This coursework involves the development of an application system catering to a equipment Rental shop. Our task entails constructing a system that effectively maintains records of rented and returned equipment. The overarching objective of this project is to establish a user-friendly application that streamlines the experience for rental shop staff. The project aims to showcase the following proficiencies:

1. Demonstrating a solid grasp of fundamental Python programming concepts.
2. Crafting a system that is intuitive and accessible to individuals without programming backgrounds.
3. Displaying competence in utilizing Python's dictionary and list structures to address specific needs.
4. Exhibiting familiarity with employing text files as data repositories, including reading and updating data directly from the program.
5. Utilizing the current date and time to meet program requirements.
6. Proficiency in using the IDLE platform for coding purposes.

# Discussion and Analysis

## Algorithm

An algorithm embodies a pre-defined method for solving a well-defined computational problem. Every facet of computer science, encompassing realms such as artificial intelligence, databases, graphics, networking, operating systems, and security, relies on the formulation and analysis of algorithms. Algorithm development transcends mere programming, involving comprehension of potential resolutions to computational challenges, along with an understanding of hardware, networking, programming languages, and the performance constraints linked to each solution.

**Step 1**: Start.

**Step 2**: Show a welcoming message.

**Step 3**: Present the choices to read, rent, return, or exit from the primary file.

**Step 4:** If the user selects to read, exhibit all the equipment particulars.

**Step 5**: If the user opts to rent, display all equipment details, then proceed to Step 8.

**Step 6**: If the user decides to return, display all equipment details, then proceed to Step 19. Step 7: If the user elects to exit, show a message conveying gratitude for their visit.

**Step 8**: Input the equipment ID intended for rental. Step 9: If an invalid ID is input, display an invalid message and return to Step 8; otherwise, proceed to Step 10.

**Step 10**: Input the quantity of equipment with the given ID for rental.

**Step 11**: If an erroneous input is provided, display an invalid message and return to Step 10; otherwise, move to Step 11.

**Step 12**: Display a message indicating the cost of a single equipment for the entered ID.

**Step 13:** An input message is displayed asking if other equipment are to be rented too. If yes, go to Step 5. If no, go to Step 14.

**Step 14**: Input the name of the customer.

**Step 13**: Display an input prompt inquiring whether additional equipments will be rented. If yes, proceed to Step 5. If no, move to Step 14.

**Step 14**: Provide the customer's name.

**Step 15**: Input the customer's phone number.

**Step 16**: If an invalid entry is given, show an invalid message and return to Step 14; otherwise, continue to Step 16.

**Step 17**: Showcase a bill or invoice, encompassing the date, customer's name and phone number, total cost, rented equipment names, equipment brands, a reminder to return within 5 days, and an expression of gratitude.

**Step 18**: Return to Step 3.

**Step 19**: Input the equipment ID for the intended return.

**Step 20**: If an incorrect ID is input, display an invalid message and return to Step 19; otherwise, proceed to Step 21.

**Step 21**: Input the quantity of equipment associated with the given ID to be returned.

**Step 22**: If an incorrect entry is made, present an invalid message and return to Step 21; otherwise, go to Step 23.

**Step 23**: Display the cost of a single equipment for the provided ID.

**Step 24**: Present an input query about returning other equipment. If affirmative, proceed to Step 6. If negative, go to Step 25.

**Step 25**: Input the customer's name.

**Step 26**: Input the customer's phone number.

**Step 27**: If an invalid input is entered, display an invalid message and return to Step 26; otherwise, continue to Step 28.

**Step 28**: Ask for the number of days the equipment has been rented.

**Step 29**: If an erroneous input is given, present an invalid message and return to Step 28; otherwise, move to Step 30.

**Step 30**: If the equipment has been rented for more than 5 days, impose a $5 fine for each additional day per equipment.

**Step 31**: Showcase a bill or invoice containing the date, customer's name and phone number, total cost, rented equipment names, equipment brands, fine amount, overall payable amount, and an expression of gratitude.

**Step 32**: Proceed to Step 3.

**Step 33**: Stop

## 2.2 Flowchart

A flowchart is a visual representation of a series of actions. It is commonly employed to illustrate the progression of algorithms, workflows, or processes by depicting steps in a sequential manner. Flowcharts typically utilize distinct types of boxes to represent various processes and connect them with arrows to indicate their sequence. An algorithm can be graphically presented through a flowchart. The utilization of flowcharts can be advantageous both in the creation of programs and in effectively conveying them to others.

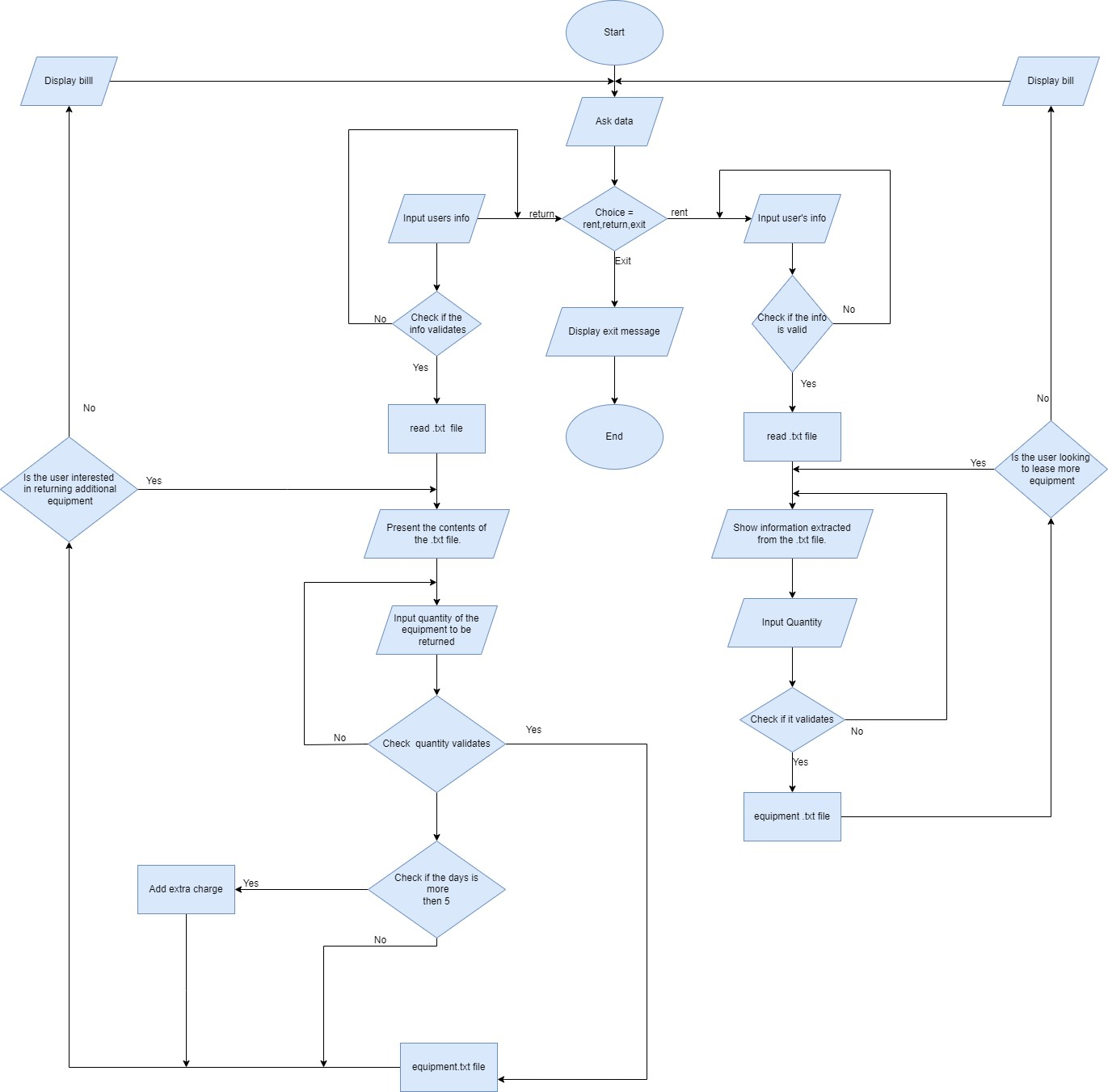


Figure 1:Flow chart

## 2.3 Pseudocode

Pseudocode serves as a universal method to depict programming logic without being bound by specific syntax rules or technology constraints. It is employed to create an initial version or outline of a program's design.

### 2.3.1 Pseudocode for Main.py

**IMPORT** read.py

**IMPORT** write\_.py

**IMPORT** operation.py

**DISPLAY** “++++++++++++++++++++++++++++++++++++++++++++++”

**DISPLAY** “Welcome to Equipment Rental Application”

**DISPLAY** “++++++++++++++++++++++++++++++++++++++++++++++”

**CREATE** function named “invalid\_message”

**DISPLAY** "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

**DISPLAY** "Invalid option. Please provide valid option."

**DISPLAY** "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

**CREATE** function named “main”

**ASSIGN** ‘loop’ as True

**WHILE** loop equals True

**ASSIGN** ‘e’ as False

**WHILE** e equals False

**TRY**

**INPUT** the desired option to read, rent, return or exit as 1, 2, 3 or

4 and store the value in variable ‘choice’

**CHANGE** ‘e’ to False

**EXCEPT:**

**CALL** ‘invalid\_message’ function

**IF ‘**choice’ equals 1:

**CALL** ‘read\_equipment’ function from read.py

**ELSE IF** ‘choice’ equals 2:

**CALL** ‘rent\_equipment’ function from rent.py

**ELSE IF** ‘choice’ equals 3:

**CALL** ‘return\_equipment ()’ function from return\_.py

**ELSE IF** ‘choice’ equals 4:

**DISPLAY** “Thank you for visiting us :)”

**CHANGE** ‘loop’ to False

**ELSE**

**CALL** ‘invalid\_message’ function

**CALL** ‘main’ function

## 2.3.3 Pseudocode for write.py

**IMPORT** datetime

**CREATE** function ‘rent\_equipment’

**CREATE** function ‘available’

**DISPLAY** "\n-----------------------------------"

**DISPLAY** "The Equipment you want to rent is available."

**DISPLAY** "------------------------------------\n"

**CREATE** function ‘unavailable’

**DISPLAY** "\n-----------------------------------")

**DISPLAY** "The equipment is unavailable at the moment."

**DISPLAY** "------------------------------------\n"

**CREATE** function ‘invalid\_message’

**DISPLAY** "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

**DISPLAY** "Invalid option. Please provide valid option."

**DISPLAY** "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

**CREATE** function ‘stock’ with ‘dictionary’ as parameter

**OPEN** ‘data.txt’ textfile for writing and store it into ‘tfile’

**FOR** i in dictionary.values():

**ASSIGN** ‘line’ to str(i[0] + "," + str(i[1]) + "," + str(i[2]) + "," + str(i[3]))

**WRITE** ‘line’ into ‘tfile’

**WRITE** "\n" into ‘tfile’

**CLOSE** ‘tfile’

**CREATE** function create\_rentfile with parameters (cname, phnumber,date, total, equipmentName, equipmentBrand)

**SET** the text file name as “Rent” + “\_” + cname+ “\_” + second and store

into tfile\_name

**OPEN** ‘tfile\_name’ in writing mode and store it into ‘tfile’

**WRITE** (“Name of the customer : " + cname + "\n")" into ‘tfile’

**WRITE** ("Phone number: " + str(phnumber)) into ‘tfile’

**WRITE** ("Date of rent : " + str(date) + "\n") into ‘tfile’

**WRITE** ("Total price of equipment: " + str(total) + "\n") into ‘tfile’

**WRITE** ("Rented equipments: " + "\n") into ‘tfile’

**FOR** i, j in zip(equipmentName,equipmentBrand)

**WRITE** (i + ":- " + j) in ‘tfile’

**WRITE** ("\n") in ‘tfile’

**CLOSE** ‘tfile’

**CREATE** function ‘total\_price’ with parameters(dictionary, quantitydet,equipmentID)

**REPLACE** "$" with “ “ in key rentID and value 2 of dictionary and convert

it into float and store it in ‘price’

**DISPLAY** "The price of equipment:" with value of ‘price’

**ASSIGN** multiplication of ‘price’ and ‘quantitydet’ to ‘priceperitem’

**RETURN** ‘priceperitem’

**CREATE** function ‘quantity\_equipment’ with parameter (quantity\_stock):

**ASSIGN** ‘exc’ to False

**WHILE** ‘exc’ equals False

**TRY**

**INPUT** the number of equipments to rent and store it in ‘quantity’

**CHANGE** ‘exc’ to True

**EXCEPT**

**CALL** function ‘invalid\_message’

**WHILE** quantity is less than or equals 0 OR quantity is greater than

quantity\_stock

**IF** quantity is less than or equals 0

**DISPLAY** "\n---------------------------------------------"

**DISPLAY** "Input is invalid. Please provide valid input."

**DISPLAY** "------------------------------------------------\n"

**ELSE IF** quantity is greater than quantity\_stock

**DISPLAY** "\n--------------------------------------------------"

**DISPLAY** "The entered quantity is greater than the quantity we have. Please enter again."

**DISPLAY** "---------------------------------------------------\n"

**TRY**

**INPUT** a valid quantity for number of equipments to rent and store it in

‘quantity’

**EXCEPT**

**CALL** function ‘invalid\_message’

**RETURN** ‘quantity’

**CREATE** function ‘rent\_dictionary’

**OPEN** ‘data.txt’ textfile for reading and store it into ‘tfile’

**ASSIGN** ‘counter’ to 0

**ASSIGN** dictionary to {}

**FOR** b in file:

**CHANGE** ‘counter’ to ‘counter’ + 1

**REPLACE** ‘\n’ with ‘ ‘ in b

**SPLIT** b in ‘,’

**ASSIGN** dictionary[counter] to b

**RETURN** ‘dictionary’

**CLOSE** ‘tfile’

**CREATE** function ‘rent\_display’

**OPEN** ‘data.txt’ textfile for reading and store it into ‘tfile’

**DISPLAY** "------------------------------------------------------------------------------"

**DISPLAY** "\tID \tCustomer Name equipment Brand Price Quantity"

**DISPLAY** "------------------------------------------------------------------------------"

**ASSIGN** counter to 0

**FOR** a in file:

**CHANGE** ‘counter’ to ‘counter’ + 1

**DISPLAY** ("\t", counter, "\t" + a.replace(",","\t"))

**DISPLAY**"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

**CLOSE** ‘tfile’

**CREATE** function ‘validID’

**ASSIGN** exception to False

**WHILE** exception equals False:

**TRY**

**INPUT** the ID of the equipment to be rented

**CHANGE** exception to True

**EXCEPT**

**CALL** function ‘invalid\_message’

**WHILE** valid\_ID is less than or equals 0 or valid\_ID is greater than thelength of rent\_dictionary()

**TRY**

**DISPLAY** "\nPlease provide a valid equipment ID.\n"

**CALL** function ‘rent\_display’

**INPUT** the ID of the equipment to be rented

**EXCEPT**

**CALL** function ‘invalid\_message’

**RETURN** ‘valid\_ID’

**CALL** function ‘rent\_display’

**CALL** ‘rent\_dictionary’ and store to dictionary

**CALL** ‘validID’ and store to rentID

**ASSIGN** ‘rentNamelist’ to []

**ASSIGN** ‘rentBrandlist’ to []

**IF** int(dictionary[rentID][3]) is greater than 0:

**CALL** function ‘available’

**ASSIGN** quantity\_equipment(int(dictionary[rentID][3])) to ‘quantity’

**ASSIGN** dictionary[rentID][3] = int(dictionary[rentID][3]) - quantity

**APPEND** rentNamelist.append(dictionary[rentID][0])

**APPEND** rentBrandlist.append(dictionary[rentID][1])

**CALL** function ‘stock’ with parameter(dictionary)

**CALL** function ‘total\_price’ with parameters(dictionary,quantity,rentID)

and store to totalPrice

**DISPLAY** "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"

**DISPLAY** "Do you want to rent another equipment too?"

**INPUT** ‘y’ to rent other equipments too

**ASSIGN** loop to True

**WHILE** loop equals True:

**IF** rentanother equals "y":

**CALL** function ‘rent\_display’

**CALL** ‘rent\_dictionary’ and store to dictionary

**CALL** ‘validID’ and store to rentID

**IF** int(dictionary[rentID][3]) is greater than 0

**CALL** function ‘available’

**ASSIGN** quantity\_equipment(int(dictionary[rentID][3])) to ‘quantity’

**ASSIGN** dictionary[rentID][3] = int(dictionary[rentID][3]) - quantity

**APPEND** rentNamelist.append(dictionary[rentID][0])

**APPEND** rentBrandlist.append(dictionary[rentID][1])

**CALL** function ‘stock’ with parameter(dictionary)

**ASSIGN** totalPrice = total\_price(dictionary,quantity,rentID) +

totalPrice

**DISPLAY**"================================================")

**DISPLAY** "Do you want to rent another equipment as well?"

**INPUT** ‘y’ to rent other equipments too

**ELSE**

**INPUT** the name of the customer

**ASSIGN** ab to True

**WHILE** ab equals True:

**TRY**

**INPUT** the phone number of customer

**CHANGE** ab to False

**EXCEPT**

**CALL** function ‘invalid\_message’

**STORE** the date and time in ‘tDate’

**STORE** values from rentNamelist without the brackets in ‘listname’

**STORE** values from rentBrandlist without the brackets in ‘listbrand’

**DISPLAY** "\n-------------------------------------------------------------------"

**DISPLAY** "\t\t\t\t INVOICE"

**DISPLAY** "--------------------------------------------------------------------\n"

**DISPLAY** "\t\t\t\t\t", tDate

**DISPLAY** "Name of customer:", name

**DISPLAY** "Phone number: ", number

**DISPLAY** "Total price:"," $", totalPrice

**DISPLAY** "Name of rented equipments:", listname

**DISPLAY** "Brand of rented equipments:", listbrand

**DISPLAY** "\n"

**DISPLAY** "The equipment has been rented."

**DISPLAY** "Be sure to return the equipment within 5 days to avoidgetting fined."

DISPLAY "\t\t\t\t Thank you."

**CALL** function ‘create\_rentfile’ with parameters(name, number,tDate, totalPrice, listname, listbrand)

**CHANGE** loop to False

**ELSE**

**CALL** function ‘unavailable’

2.3.4. Pseudocode for operation.py

**IMPORT** datetime

**CREATE** function ‘return\_equipment’

**CREATE** function ‘available’

**DISPLAY** "\n-----------------------------------"

**DISPLAY** "You can return the equipment."

**DISPLAY** "------------------------------------\n"

**CREATE** function ‘unavailable’

**DISPLAY** "\n-----------------------------------")

**DISPLAY** "The equipment you want to return is not in the database."

**DISPLAY** "------------------------------------\n"

**CREATE** function ‘invalid\_message’

**DISPLAY** "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

**DISPLAY** "Invalid option. Please provide valid option."

**DISPLAY** "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"

**CREATE** function ‘stock’ with ‘dictionary’ as parameter

**OPEN** ‘data.txt’ textfile for writing and store it into ‘tfile’

**FOR** i in dictionary.values():

**ASSIGN** ‘line’ to str(i[0] + "," + str(i[1]) + "," + str(i[2]) + "," + str(i[3]))

**WRITE** ‘line’ into ‘tfile’

**WRITE** "\n" into ‘tfile’

**CLOSE** ‘tfile’

**CREATE** function create\_returnfile with parameters (cname, phnumber,

date, total, equipmentName, equipmentBrand, cfine)

**SET** the text file name as “Return” + “\_” + cname+ “\_” + second and

store into ‘tfile\_name’

**OPEN** ‘tfile\_name’ in writing mode and store it into ‘tfile’

**WRITE** (“Name of the customer : " + cname + "\n") "into ‘tfile’

**WRITE** ("Phone number: " + str(phnumber)) into ‘tfile’

**WRITE** ("Date of return : " + str(date) + "\n") into ‘tfile’

**WRITE** ("Total price of equipment: " + str(total) + "\n") into ‘tfile’

**WRITE** ("Returned equipments: " + "\n") into ‘tfile’

**WRITE** ("Fine: " + str(cfine) + "\n") into ‘tfile’

**FOR** i, j in zip(equipmentName,equipmentBrand)

**WRITE** (i + ":- " + j) in ‘tfile’

**WRITE** ("\n") in ‘tfile’

**CLOSE** ‘tfile’

**CREATE** function ‘total\_price’ with parameters(dictionary, quantitydet,

equipmentID)

**REPLACE** "$" with “ “ in key rentID and value 2 of dictionary and convert

it into float and store it in ‘price’

**DISPLAY** "The price of equipment:" with value of ‘price’

**ASSIGN** multiplication of ‘price’ and ‘quantitydet’ to ‘priceperitem’

**RETURN** ‘priceperitem’

**CREATE** function ‘quantity\_equipment’ with parameter (quantity\_stock):

**ASSIGN** ‘exc’ to False

**WHILE** ‘exc’ equals False

**TRY**

**INPUT** the number of equipments to return and store it in ‘quantity’

**CHANGE** ‘exc’ to True

**EXCEPT**

**CALL** function ‘invalid\_message’

**WHILE** quantity is less than or equals 0 OR quantity is greater than

quantity\_stock

**IF** quantity is less than or equals 0

**DISPLAY** "\n---------------------------------------------"

**DISPLAY** "Input is invalid. Please provide valid input."

**DISPLAY** "------------------------------------------------\n"

**ELSE** **IF** quantity is greater than quantity\_stock

**DISPLAY** "\n--------------------------------------------------"

**DISPLAY** "The entered quantity is greater than the quantity we have. Please enter again."

**DISPLAY** "---------------------------------------------------\n"

**ASSIGN** ex to False

**WHILE** ex equals False

**TRY**

**INPUT** a valid quantity for number of equipments to rent and

store it in ‘quantity’

**EXCEPT**

**CALL** function ‘invalid\_message’

**RETURN** ‘quantity’

**CREATE** function ‘return\_dictionary’

**OPEN** ‘data.txt’ textfile for reading and store it into ‘tfile’

**ASSIGN** ‘counter’ to 0

**ASSIGN** dictionary to {}

**FOR** b in tfile:

**CHANGE** ‘counter’ to ‘counter’ + 1

**REPLACE** ‘\n’ with ‘ ‘ in b

**SPLIT** b in ‘,’

**ASSIGN** dictionary[counter] to b

**RETURN** ‘dictionary’

**CLOSE** ‘tfile’

**CREATE** function ‘return\_display’

**OPEN** ‘data.txt’ textfile for reading and store it into ‘tfile’

**DISPLAY** "------------------------------------------------------------------------------"

**DISPLAY** "\tID \tEquipment Name equipment Brand Price Quantity"

**DISPLAY** "------------------------------------------------------------------------------"

**ASSIGN** counter to 0

**FOR** b in tfile:

**CHANGE** ‘counter’ to ‘counter’ + 1

**DISPLAY** ("\t", counter, "\t" + a.replace(",","\t"))

**DISPLAY** "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

**CLOSE** ‘tfile’

**ASSIGN** exception to False

**WHILE** exception equals False:

**TRY**

**INPUT** the ID of the equipment to be returned

**CHANGE** exception to True

**EXCEPT**

**CALL** function ‘invalid\_message’

**WHILE** valid\_ID is less than or equals 0 or valid\_ID is greater than the

length of rent\_dictionary()

**TRY**

**DISPLAY** "\nPlease provide a valid equipment ID.\n"

**CALL** function ‘return\_display’

**INPUT** the ID of the equipment to be rented

**EXCEPT**

**CALL** function ‘invalid\_message’

**RETURN** ‘valid\_ID’

**CALL** function ‘return\_display’

**CALL** ‘return\_dictionary’ and store to dictionary

**CALL** ‘validID’ and store to returnID

**ASSIGN** ‘returnNamelist’ to []

**ASSIGN** ‘returnBrandlist’ to []

**IF** int(dictionary[returnID][3]) is greater than 0:

**CALL** function ‘available’

**ASSIGN** quantity\_equipment(int(dictionary[rentID][3])) to ‘quantity’

**ASSIGN** dictionary[rentID][3] = int(dictionary[rentID][3]) - quantity

**APPEND** returnNamelist.append(dictionary[returnID][0])

**APPEND** returnBrandlist.append(dictionary[returnID][1])

**CALL** function ‘stock’ with parameter(dictionary)

**CALL** function ‘total\_price’ with parameters(dictionary,quantity,rentID)

and store to totalPrice

**DISPLAY** "^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^"

**DISPLAY** "Do you want to return another equipment too?"

**INPUT** ‘y’ to return other equipments too

**ASSIGN** loop to True

**WHILE** loop equals True:

**IF** returnanother equals "y":

**CALL** function ‘return\_display’

**CALL** ‘return\_dictionary’ and store to dictionary

**CALL** ‘validID’ and store to returnID

**IF** int(dictionary[rentID][3]) is greater than 0

**CALL** function ‘available’

**ASSIGN** quantity\_equipment(int(dictionary[returnID][3])) to

‘quantity’

**ASSIGN** dictionary[returnID][3] = int(dictionary[returnID][3]) -

quantity

**APPEND** returnNamelist.append(dictionary[returnID][0])

**APPEND** returnBrandlist.append(dictionary[returnID][1])

**CALL** function ‘stock’ with parameter(dictionary)

**ASSIGN** totalPrice = total\_price(dictionary,quantity,returnID) +

totalPrice

**DISPLAY**"================================================")

**DISPLAY** "Do you want to return another equipment as well?")

**INPUT** ‘y’ to return other equipments too

**ELSE**

**ASSIGN** fineperequipment to 5

**ASSIGN** Totalfine as 0

**INPUT** the name of the customer

**ASSIGN** ab to True

**WHILE** ab equals True:

**TRY**

**INPUT** the phone number of customer

**CHANGE** ab to False

**EXCEPT**

**CALL** function ‘invalid\_message’

**ASSIGN** cd as True

**WHILE** cd equals True

**TRY**

**INPUT** the number of days of rent

CHANGE to False

**EXCEPT**

**CALL** function ‘invalid\_message’

**STORE** the date and time in ‘tDate’

**STORE** values from rentNamelist without the brackets in ‘listname’

**STORE** values from rentBrandlist without the brackets in ‘listbrand’

**IF** daysafter is greater than 5

**ASSIGN** subtraction of daysafter with 5 to extradays

**FOR** k in returnNamelist

**ASSIGN** Totalfine +=

extradays\*int(quantity)\*fineperequipment

**ELSE**

**ASSIGN** Totalfile to 0

**ASSIGN** Totalcost equals addition of totalPrice and Totalfine

**DISPLAY** "\n-------------------------------------------------------------------"

**DISPLAY** "\t\t\t\t INVOICE"

**DISPLAY** "--------------------------------------------------------------------\n"

**DISPLAY** "\t\t\t\t\t", tDate

**DISPLAY** "Name of customer:", name

**DISPLAY** "Phone number: ", number

**DISPLAY** "Total price:"," $", totalPrice

**DISPLAY** "Name of rented equipments:", listname

**DISPLAY** "Brand of rented equipments:", listbrand

**DISPLAY** "Fine:", Totalfine

**DISPLAY** "Total cost:", Totalcost

**DISPLAY** "\n"

**DISPLAY** "The equipment has been returned."

**DISPLAY** "\t\t\t\t Thank you."

**CALL** function ‘create\_rentfile’ with parameters(name, number,

tDate, totalPrice, listname, listbrand, Totalfine)

CHANGE loop to False

**ELSE**

**CALL** function ‘unavailable’

# 2.4 Data Structures

A well-crafted framework crafted to organize, handle, retrieve, and store data is referred to as a data structure. Data structures manifest in both elementary and intricate configurations, all tailored to arrange data for specific applications. Data structures ensure convenient data retrieval and utilization, facilitating users in comprehending and employing the required information adeptly. These structures establish a structured presentation of information that is comprehensible to both machines and individuals.

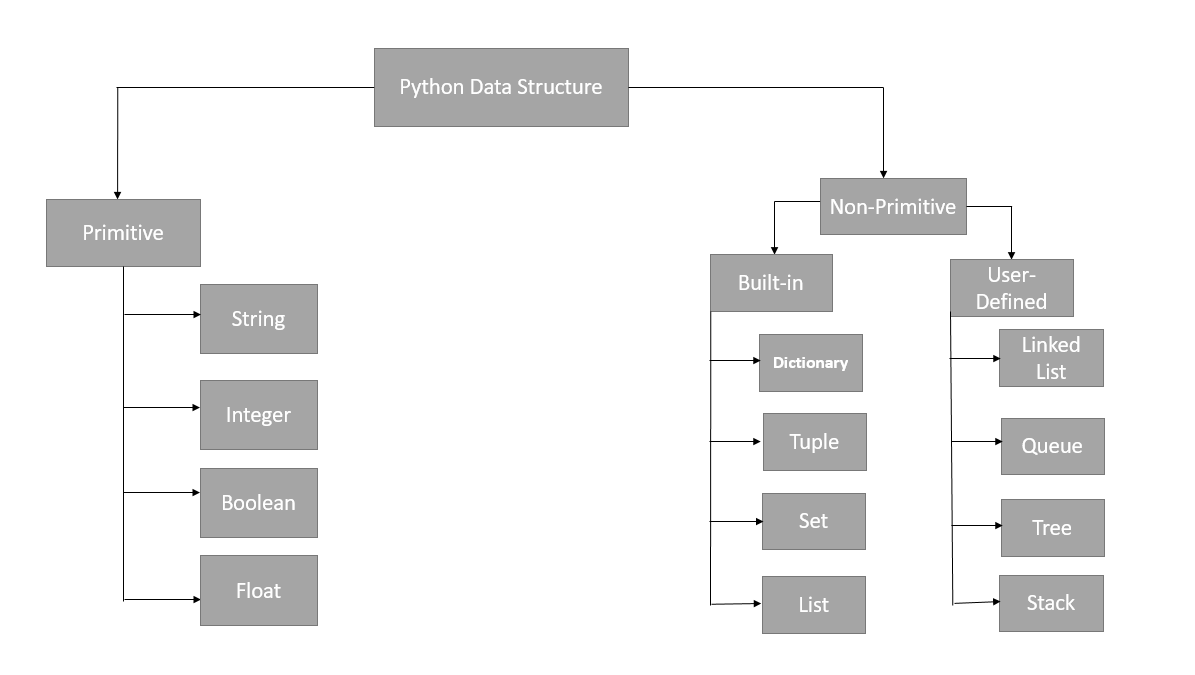


Figure 2:Data structure

## 2.4.1 Primitive Data Structures

### . Integer

In Python, an integer denotes a whole number including zero, positive, or negative values, and it holds infinite precision. For instance, it could be 100, 0, or -1000000000. To convert other data types to integers, the int() function is utilized. I've used this method in my code to change string values into integer values, following a comparable approach.

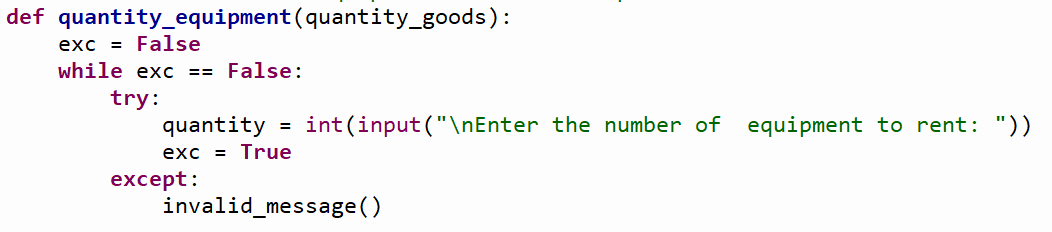


Figure 3: Integer

### 

### . Float

In Python, the float type is used to represent floating-point numbers. The term "float" signifies real numbers, and it is characterized by a decimal point that separates the integer and fractional components. Examples of floating-point numbers encompass 97.98, 69.2+e18, and -57.02+e100, among others.

. String

In Python, a string is described as a contiguous sequence of characters enclosed within quotation marks. The string type is referred to as "str" in Python. To transform various data types into the string data type, the str() function is employed. I've employed this technique in my code to convert diverse data types into strings, following a similar procedure.

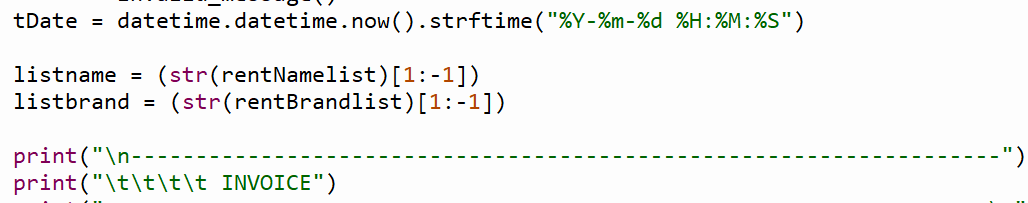


Figure 4:String

### . Boolean

Boolean values encompass "True" or "False" states. In Python, these states are defined using the "True" and "False" keywords to create Boolean variables. In my program, I've incorporated Boolean data types within loops. This determines whether the loop should persist or terminate.

## 2.4.2. Non Primitive Data Structures

Non-primitive data structures are a category of data structures designed to not only hold individual values but also to store collections of values.

### . List

Lists store their data in a sequential manner, utilizing integer index positions. Lists offer flexibility, enabling the addition, removal, and modification of elements. The versatility of lists allows the storage of multiple elements within a single variable.

### . Tuple

A tuple signifies an ordered collection of items that remains immutable. Both tuples and lists are classified as sequences. However, tuples and lists diverge in that tuples are unchangeable, unlike lists which can be modified. Another distinction lies in the use of parentheses for tuples and square brackets for lists.

### . Set

Comparable to lists or tuples, a set is a modifiable assembly of distinct hash able items. Due to its unordered nature, a set does not retain the placement or sequence of inserted elements, rendering it unfeasible to retrieve items using indices.

### . Dictionary

A dictionary constitutes an unordered compilation containing key-value pairs enclosed within curly braces and separated by commas. Unlike other data types limited to single values as elements, a Python dictionary comprises keys and corresponding values, serving as a mechanism to store data values akin to a map.

# Program

This project encompasses a software application enabling users to maintain a log of rented and returned equipments. Through this application, users can effortlessly track changes to the stock count during both rental and return procedures, while also computing the total customer charges, including any fines. Additionally, the application generates an invoice or bill upon transaction completion. The development of this application was carried out using the Python programming language within the IDLE platform.

## Implementation

The program has been created, comprising four Python files (.py) and an additional text file (.txt). The text file holds comprehensive information about the data structure employed within the rental shop.

The program encompasses specific Python files, each with distinct roles, which are outlined briefly below:

### Main.py

This particular file serves as the program's primary module, initiating its execution to generate the desired output. It imports and utilizes other associated Python files within the program. In this file, users are presented with several choices, including viewing, renting, returning, or exiting. The implementation involves the utilization of loops and conditional statements (If-else-else if) to facilitate these actions. Furthermore, the program incorporates error handling, where an erroneous input prompts the display of an "Invalid input" message, prompting the user to re-enter the appropriate value.

### Read.py

This Python script enables the user to access and review equipment information available in the store. By opening a text file in read mode, the script facilitates the presentation of data from the file in a visually appealing and user-friendly format.

### Write.py

Contained within this Python file is the script for Equipment rental, encompassing the updating of stock quantities as equipments are rented. Initially, it presents the data from a text file in a comprehensible manner, much like the "read.py" module. Subsequently, the user is prompted to input the Equipment ID for rental. In the event of invalid input, a try-except block is employed, displaying a message indicating incorrect input and prompting the user to provide valid input. Should the requested equipments be unavailable, a message communicates its unavailability. Conversely, if the desired equipments is in stock, the user is prompted to specify the number of equipments they wish to rent for that equipments ID. Similarly, an invalid input message appears upon erroneous entries.

Upon valid input, the cost of a single equipments linked to the Equipment ID is displayed. Subsequently, the user is queried about renting additional equipments. A positive response redirects the user to input another Equipment ID. Conversely, a negative response leads to soliciting the user's name and phone number. Following this, an invoice surfaces, encompassing the date and time, customer's name and phone number, the total rental cost, and details about the rented equipments ' name and brand. The invoice concludes with an expression of gratitude and a reminder to return the equipments within five days to prevent potential fines.

### Operation.py

The code within this Python file shares similarities with "write.py" while also featuring distinct variations. This file facilitates the return of one or more equipments, updating the stock quantities accordingly. The initial step involves presenting data from the text file in a lucid manner, akin to the "read.py" module. Subsequently, the user is prompted to input the Equipment ID of the attire they intend to return. Invalid inputs elicit a request for valid ones, supported by a try-except block to display an "invalid input" message. If the requested equipments is unavailable, a message indicating its absence is displayed.

For available equipments, users are prompted to input the quantity of the specified Equipment ID they wish to return. Similar to prior instances, an invalid input notification is presented if applicable. Upon accurate input, the cost of an individual equipment linked to the specified Equipment ID is displayed. Subsequently, users are queried about returning more equipments. Affirmative responses direct users back to the Equipment ID input screen. Conversely, negative responses prompt users to provide their name, contact details, and the number of days the equipments was rented—a factor affecting potential fines.

Following this, the invoice is generated, encompassing the date, time, customer's name and contact information, any applicable fines, the total cost including fines, details about the rented equipments name and brand, and a note of gratitude.

### Renting and Returning Equipment

### 3.2.1 Renting Equipment

To initiate the process of renting equipment, the user needs to input '2' from the main menu options. Once this input is provided, the program will present the equipment data in a well-organized manner, displaying details about each equipment."

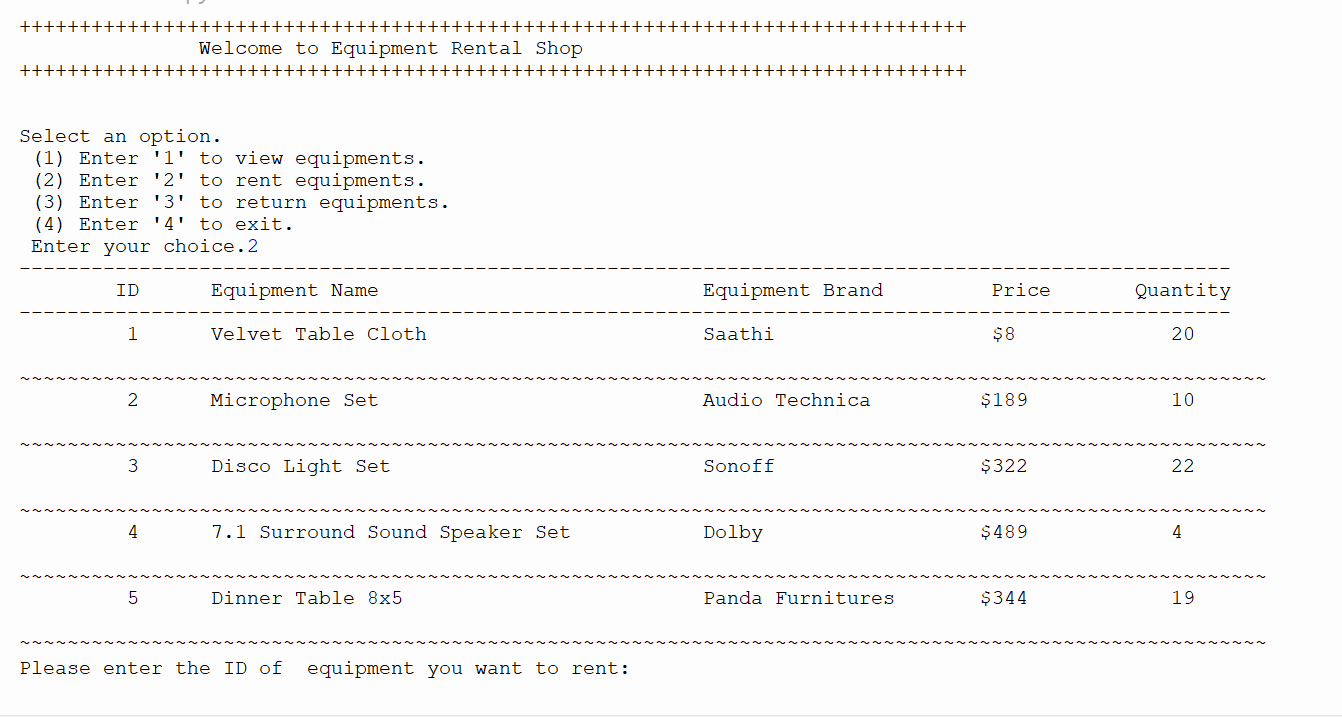


Figure 5:Renting equipment 1

Next, a message will pop up, requesting the user to input the equipment ID they're interested in renting. Once done, the program will ask for the quantity of equipment needed. If any incorrect inputs are provided, the program will respond with an error message for those inputs. When valid details are entered, the program will display the cost of a single equipment based on the given ID. Following this, the user will be asked if they're considering renting another equipment.

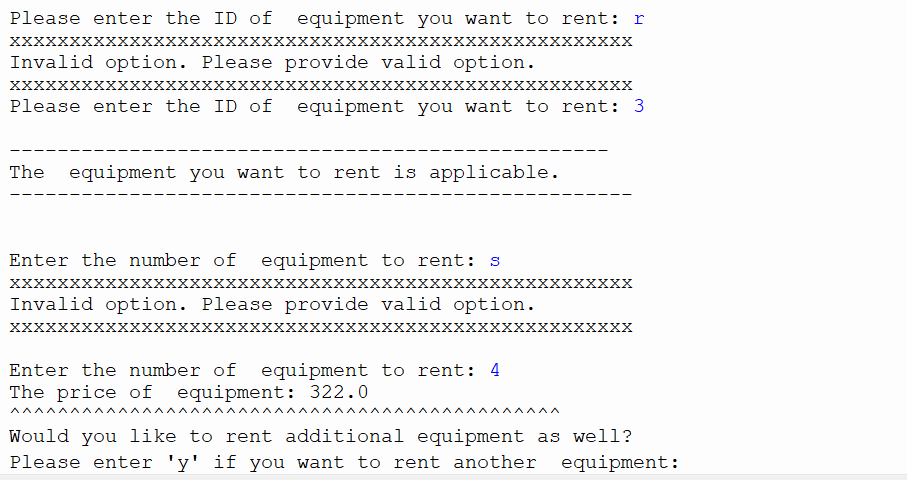


Figure 6: Renting Equipment 1.1

If the user inputs 'y', the program will return to showcasing the equipments, now reflecting the updated quantities. Following this, the user will once more be prompted to provide the ID of the equipment they wish to rent.

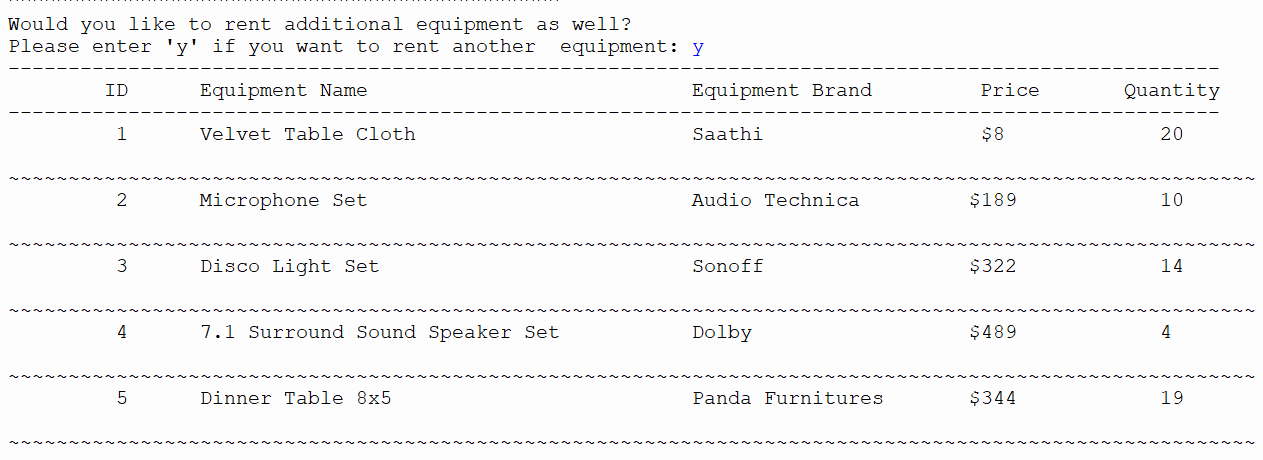


Figure 7:Renting Equipment 1.3

Next, just like previously, the program will request the user to enter the number of equipments they wish to rent. After that, the cost will be shown as it was before. Following this, the user will be asked if they'd like to rent another equipments once again.

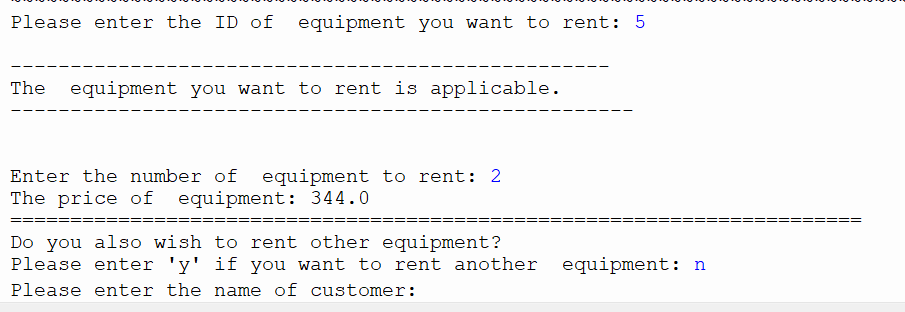


Figure 8:Renting Equipment 1.4

If the user decides not to rent another equipment, they will be requested to input their name and phone number. When valid details are provided, the program will show an invoice containing the date and time, the customer's name and phone number, the total cost, and the names and brands of the rented equipments. This will be accompanied by a thank you message and a friendly reminder to return the equipment within 5 days.

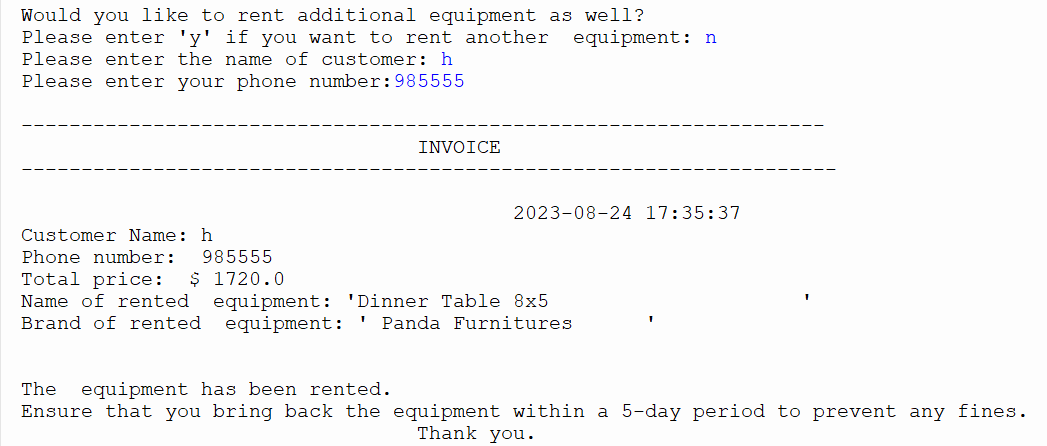
 Top of Form

Figure 9:Renting Equipment 1.5

A text document is generated to store the record of the transaction's invoice.

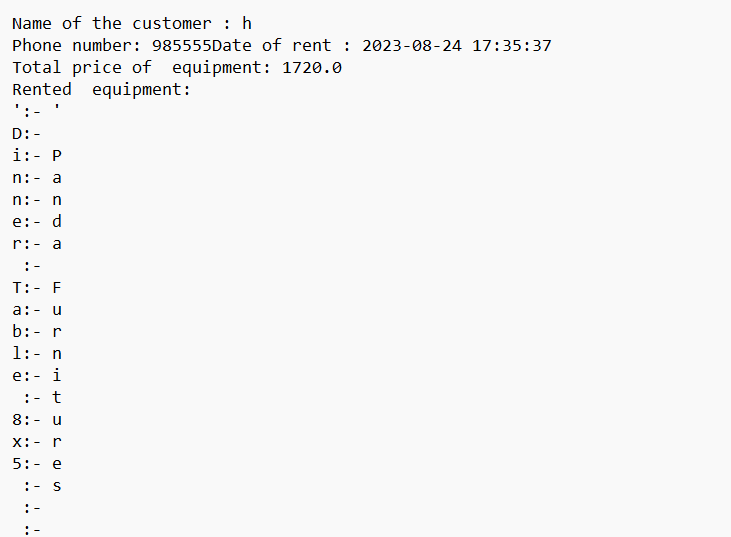


Figure 10:Renting Equipment 1.6

## 3.2.2. Returning Equipment

To initiate the process of returning equipment, the user should input '3'. Subsequently, the equipment details will be presented in a well-organized manner.

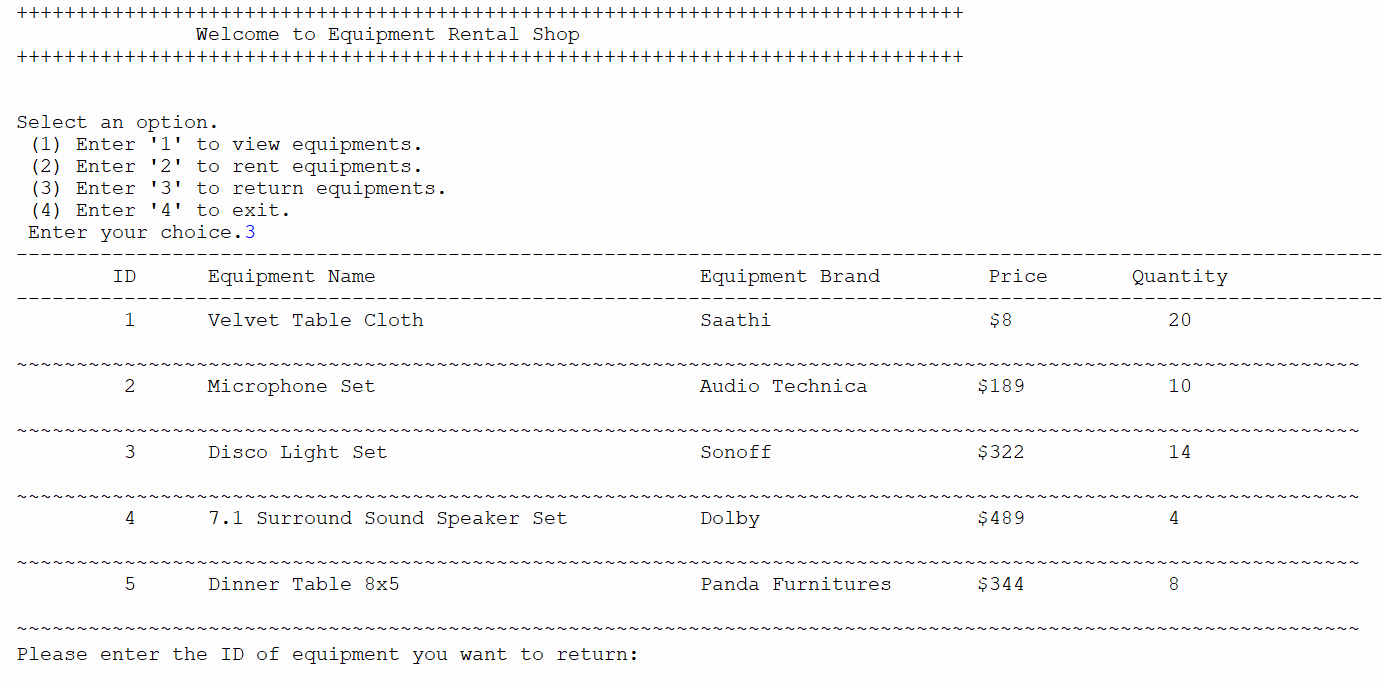


Figure 11:Returning Equipment 1.1

The user is prompted with an input message, guiding them to provide the ID of the equipment they intend to return. Subsequently, the program asks for the quantity of equipments to be returned. If invalid inputs are given, a message indicating their invalidity is shown. Irrespective of the input's accuracy, the cost for renting a single equipment associated with the given ID is shown. Afterward, the user is inquired whether they want to proceed with returning additional equipments. For returning another equipment, the user should input 'y'. In case this input is received, the equipments are displayed once more, now reflecting updated values. The user is again requested to input the ID of the equipment they wish to return.

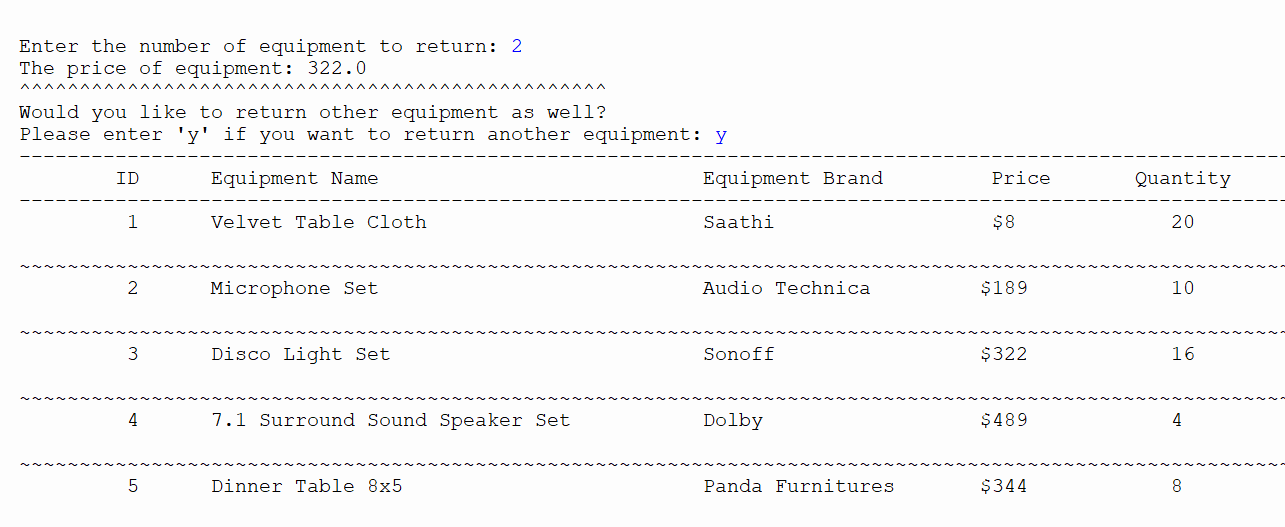


Figure 12:Returning Equipment 1.2

If the user decides not to return another equipment, they should input anything other than 'y'. Following this, the user is prompted to provide their name, phone number, and the number of days the equipment was rented. This duration helps calculate any fine to be added to the total cost. No fine is added if the equipment is returned within 5 days. Additionally, an invoice is shown with the date and time, customer's name and phone number, total equipment cost, equipment names and brands, fine (if applicable), and the overall cost including any fine. A thank you message is displayed as well. After this, the program returns to offering the user various options.

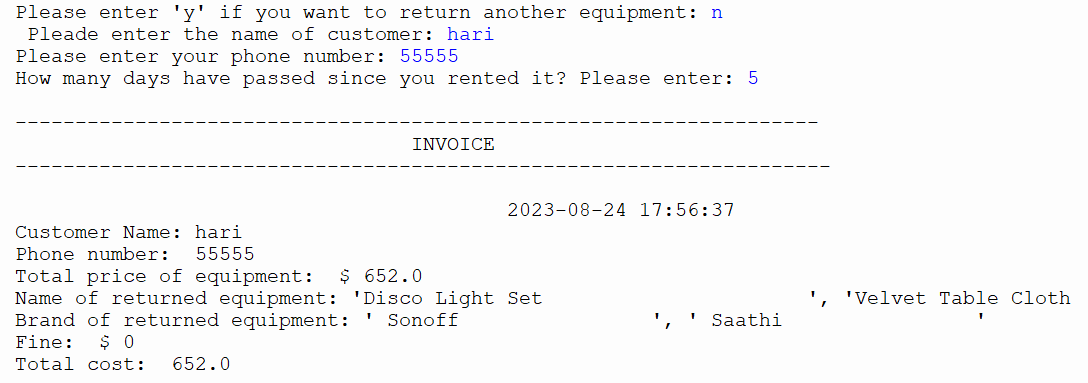


Figure 13:Returning Equipment 1.3

Moreover, a text document is generated to store the bill or invoice of the transaction.

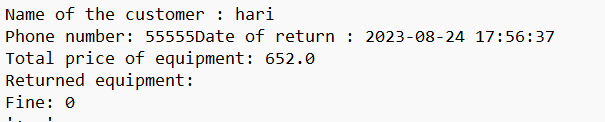


Figure 14:Returning Equipment 1.4

## 3.2.3. Termination of the program

The program concludes by displaying an expression of gratitude when the user selects option 4.

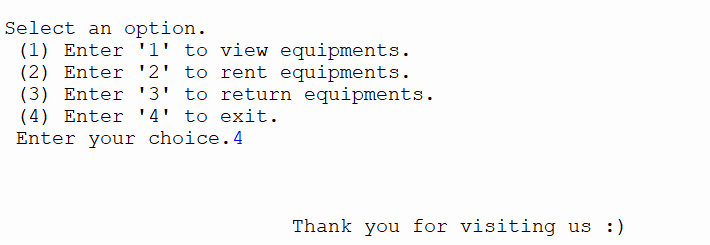


Figure 15:Termination of the program

# Testing

## Test 1

To demonstrate the utilization of the try and except used within the program.

|  |  |
| --- | --- |
| Test Number | 1 |
| Objective | To demonstrate how the program uses try and except. |
| Action | * The code included try and except blocks where there could be problems or errors. * The program was executed. * Invalid input was given on purpose |
| Expected Result | The try and except would function correctly, and an error message would appear. |
| Actual Result | The try and except worked well, and an error message showed up. |
| Conclusion | The test was successful |

Table 1:Test 1

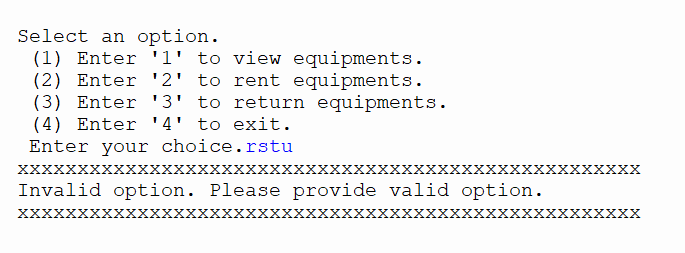


Figure 16:Test 1

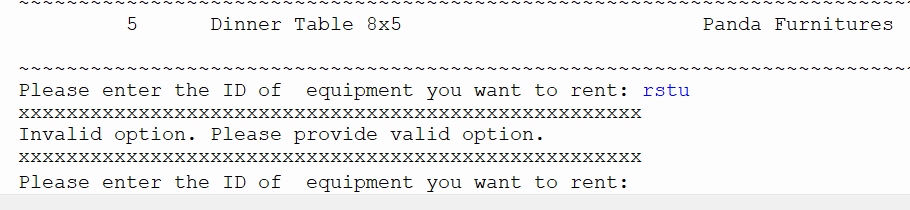


Figure 17:Test 1.1

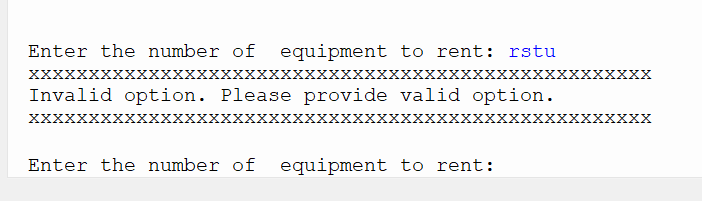


Figure 18:Test 1.2

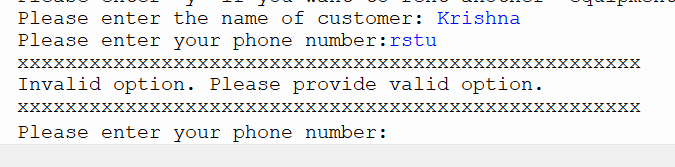


Figure 19:Test 1.3

## Test 2

To demonstrate choosing to either rent or return equipment.

|  |  |
| --- | --- |
| Test Number | 2 |
| Objective | To illustrate the option to either rent or return equipment within the program.  Top of Form |
| Action | Negative and non-existent values were input when renting and returning equipment. |
| Expected Result | Error notifications would show up.  Top of Form |
| Actual Result | Error notifications were show up.  Top of Form |
| Conclusion | The test was successful |

Table 2:Test 2

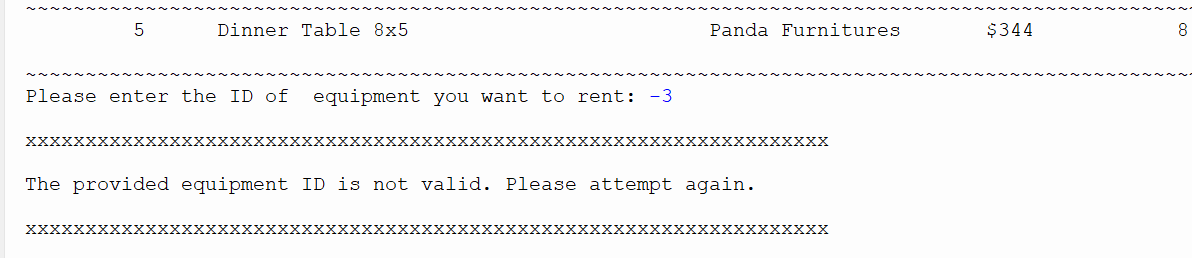


Figure 20:Test 2.1

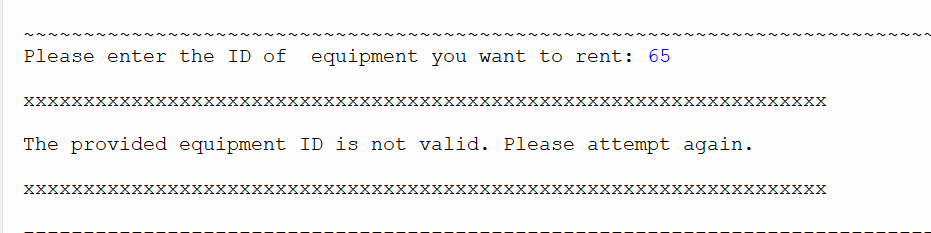


Figure 21:Test 2.2

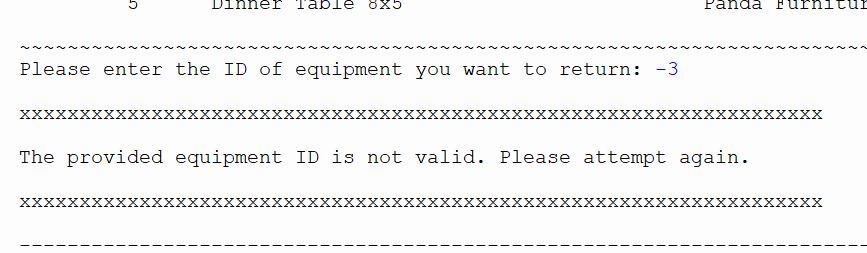


Figure 22:Test 2.3

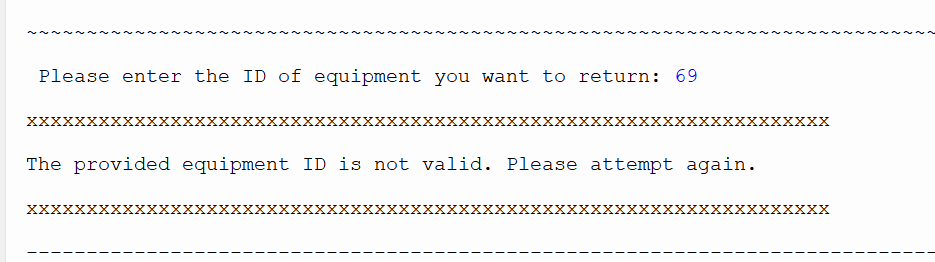


Figure 23:Test 2.4

## 4.3 Test 3

To demonstrate file creation when renting multiple pieces of equipment.

|  |  |
| --- | --- |
| Test Number | 3 |
| Objective | To illustrate the process of creating files when renting multiple equipment.  Top of Form |
| Action | * Several equipments are rented. * The result is displayed. * The newly created text file is also displayed. |
| Expected Result | A .txt file will be created. |
| Actual Result | A .txt file is created. |
| Conclusion | The test was successful |

Table 3:Test 3

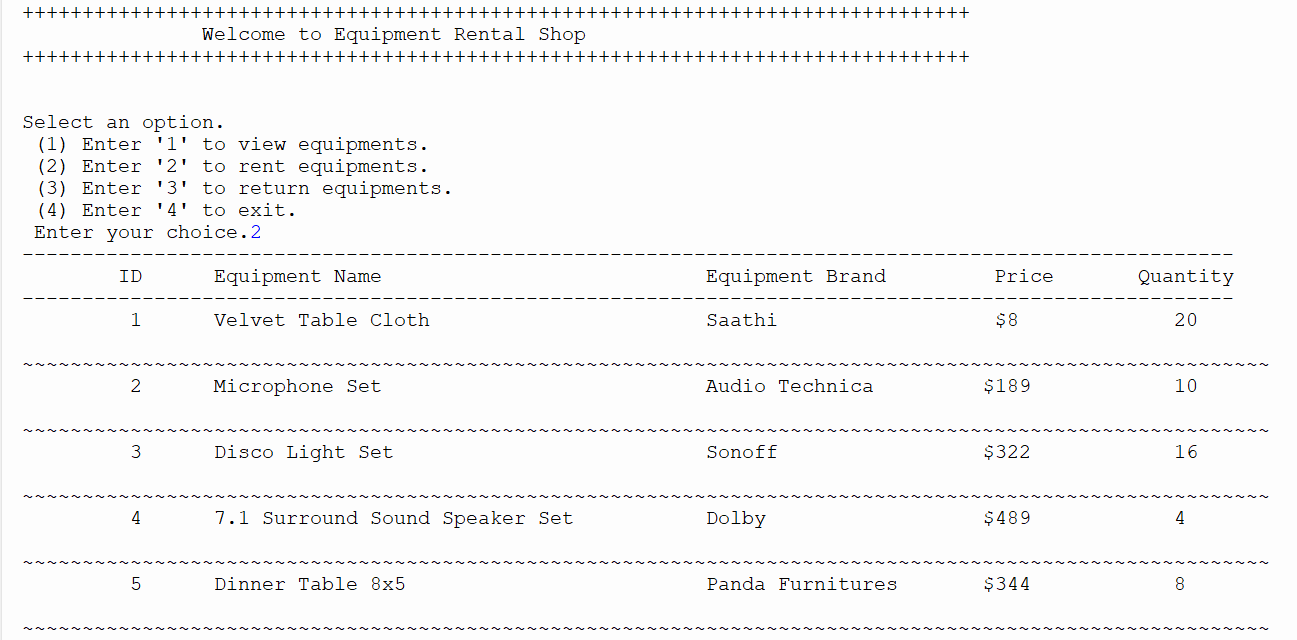


Figure 24:Test 3.1

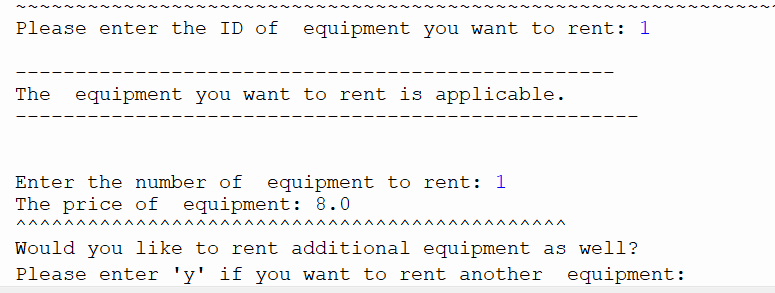


Figure 25:Test 3.2

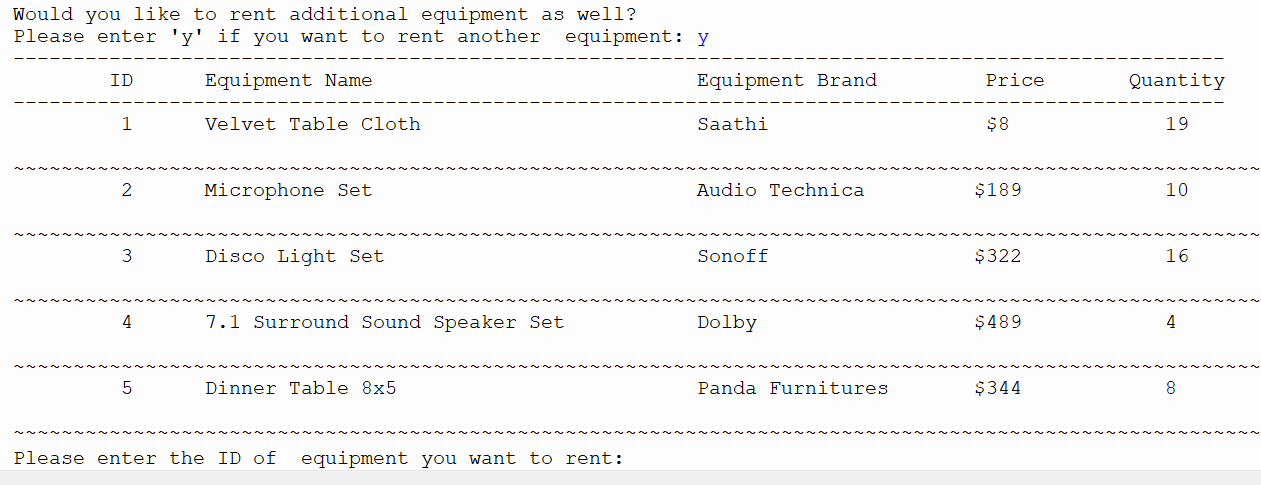


Figure 26:Test 3.3

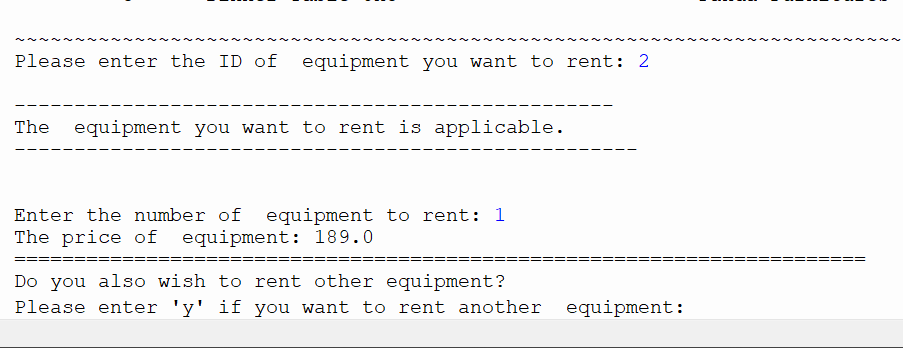


Figure 27::Test 3.4

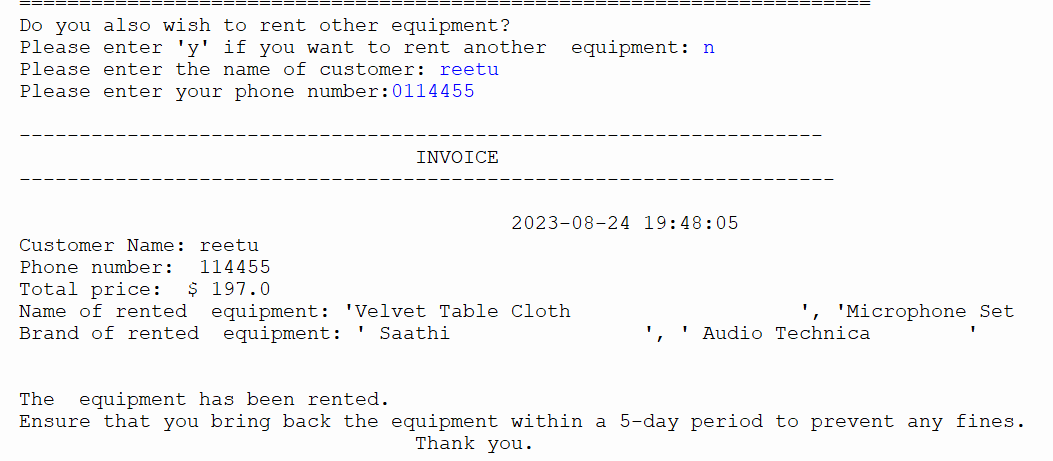


Figure 28:Test 3.5

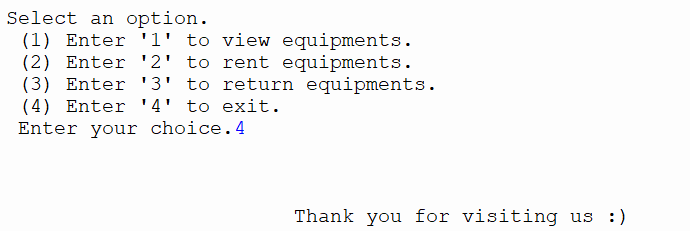


Figure 29::Test 3.6

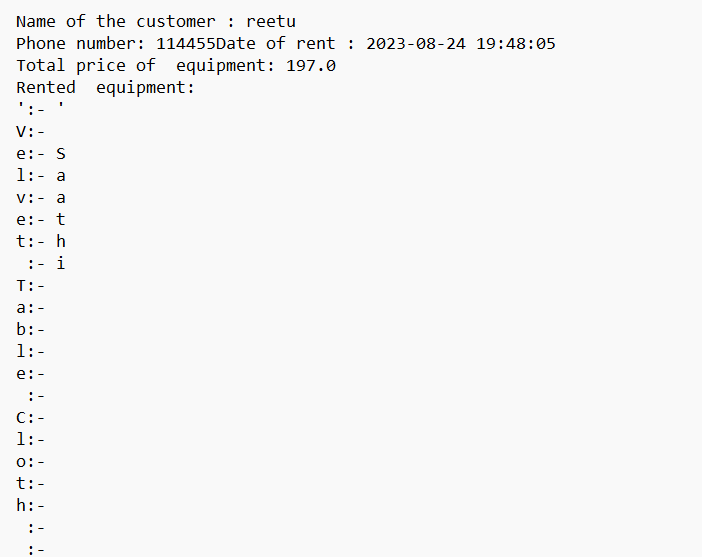


Figure 30:Test 3.7

## Test 4

To demonstrate the process of creating files when returning multiple pieces of equipment.

|  |  |
| --- | --- |
| Test Number | 4 |
| Objective | To illustrate the creation of files when returning multiple pieces of equipment.  Top of Form |
| Action | * Several pieces of equipment are returned * The result is displayed. * The newly created text file is also displayed. |
| Expected Result | A .txt file will be created. |
| Actual Result | A .txt file is created. |
| Conclusion | The test was successful |

Table 4:Test 4

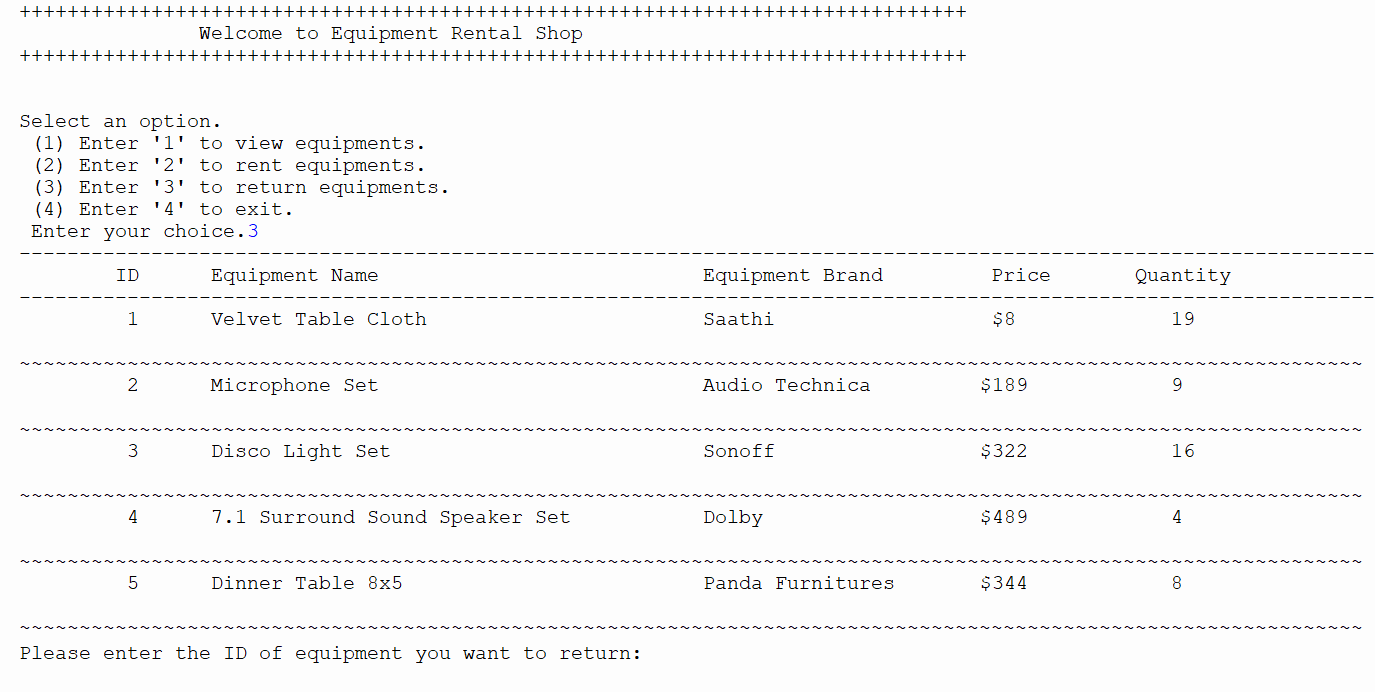


Figure 31:Test 4.1

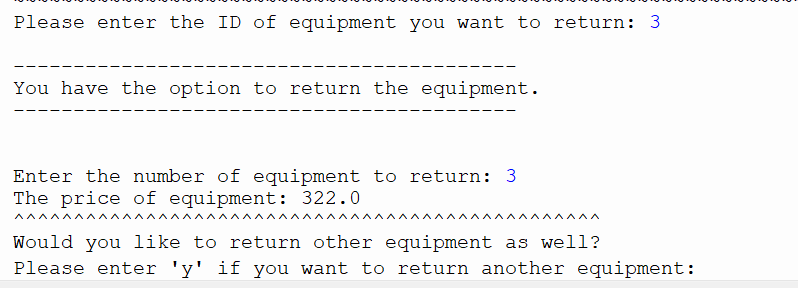


Figure 32:Test 4.2

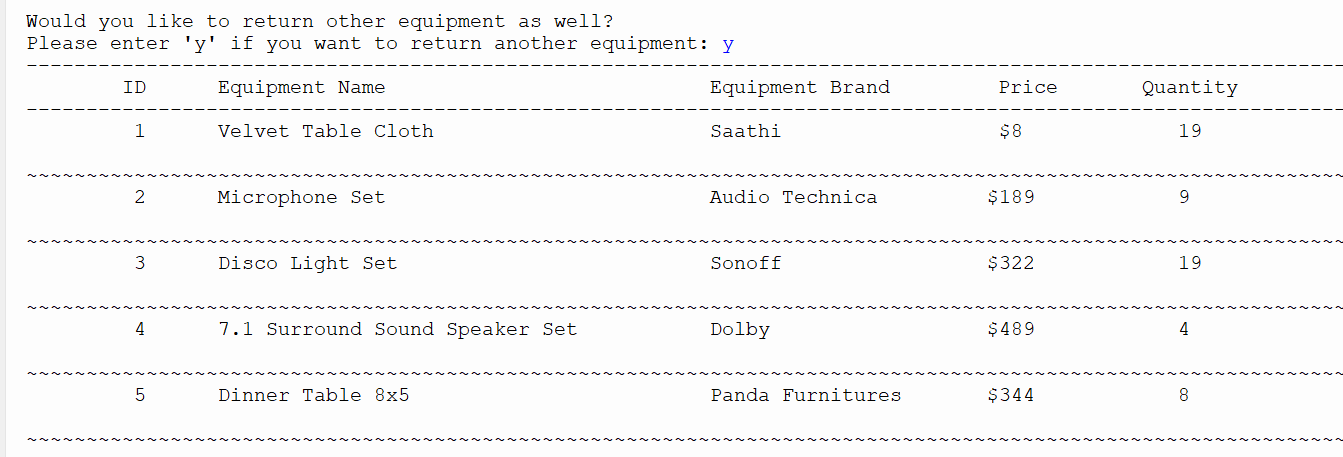


Figure 33:Test 4.3

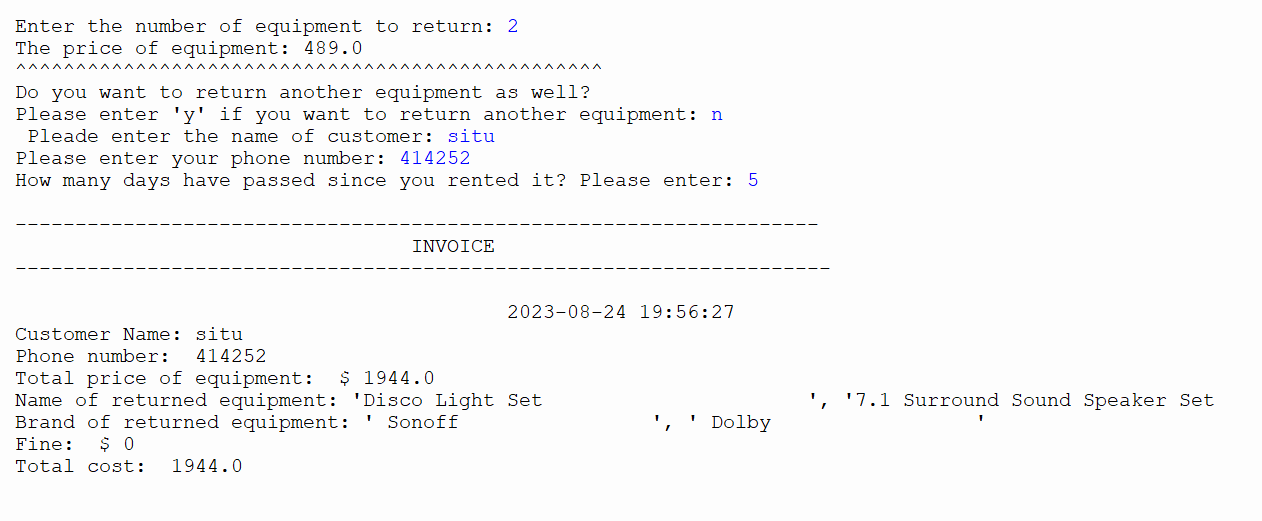


Figure 34:Test 4.4

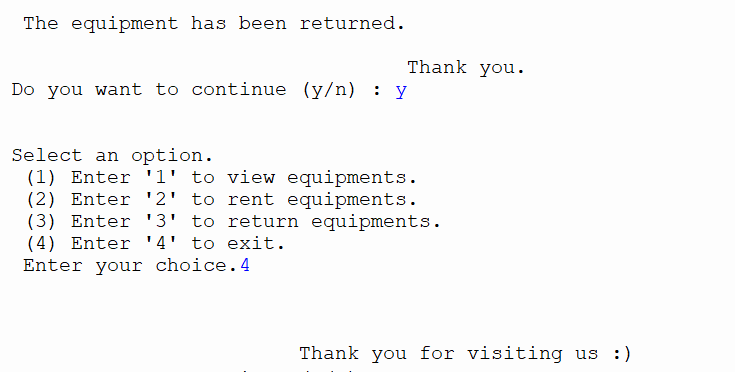


Figure 35:Test 4.5

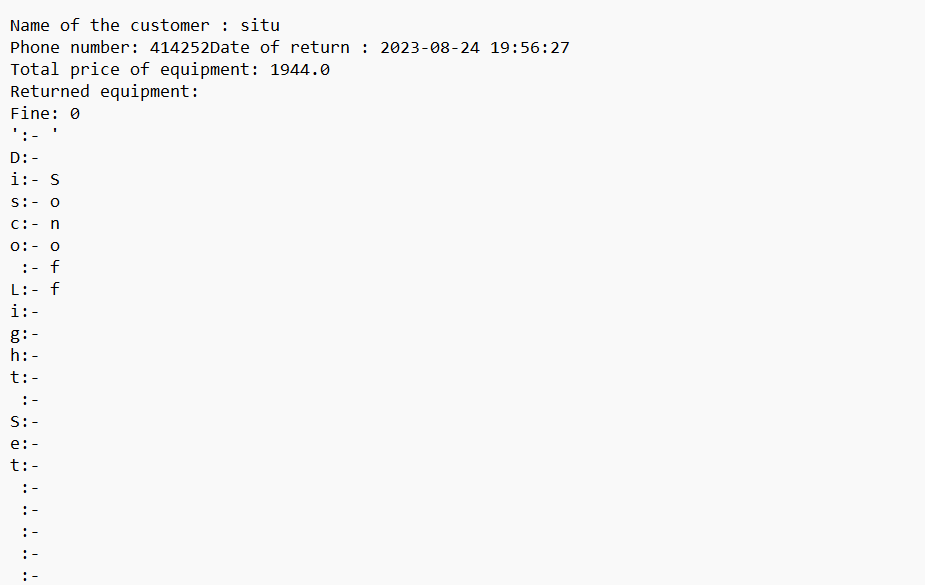


Figure 36:Test

## Test 5

To demonstrate changes in the stock of equipments.

|  |  |
| --- | --- |
| Test Number | 5 |
| Objective | To display changes in the equipment stock.  Top of Form |
| Action | * Several equipments were rented. * Several equipments were returned. |
| Expected Result | The stock will be updated when viewed and in the .txt file too. |
| Actual Result | The stock is updated when viewed and in the .txt file too. |
| Conclusion | The test was successful |

Table 5:Test 5

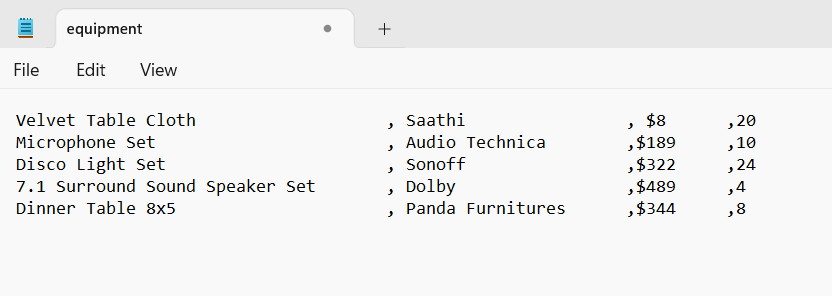


Figure 37:Test 5.1

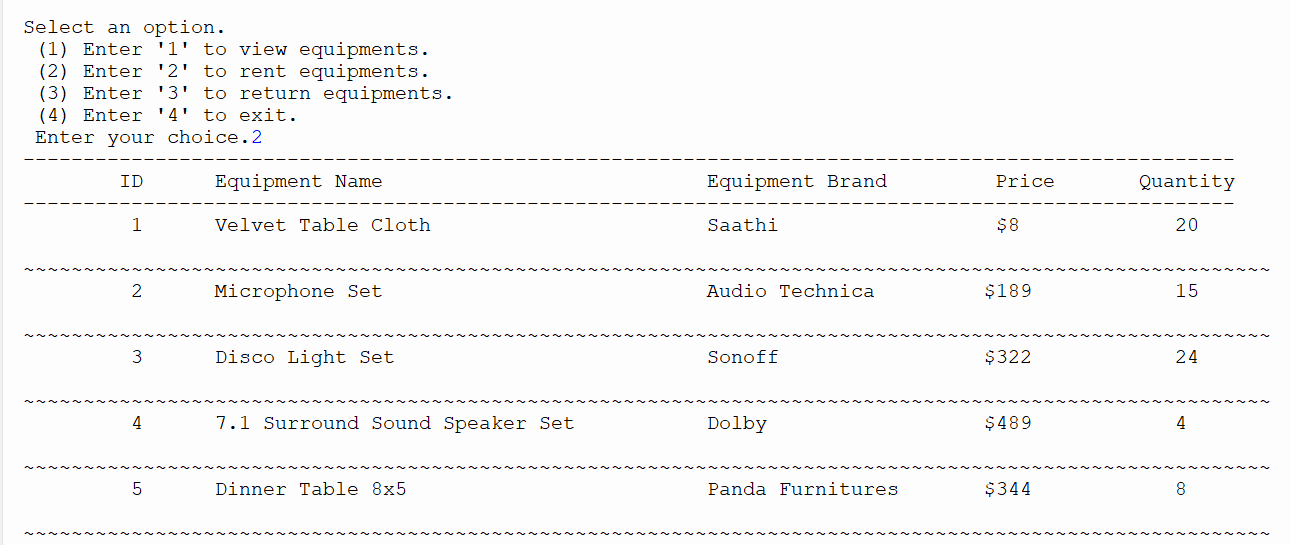


Figure 38:Test 5.2

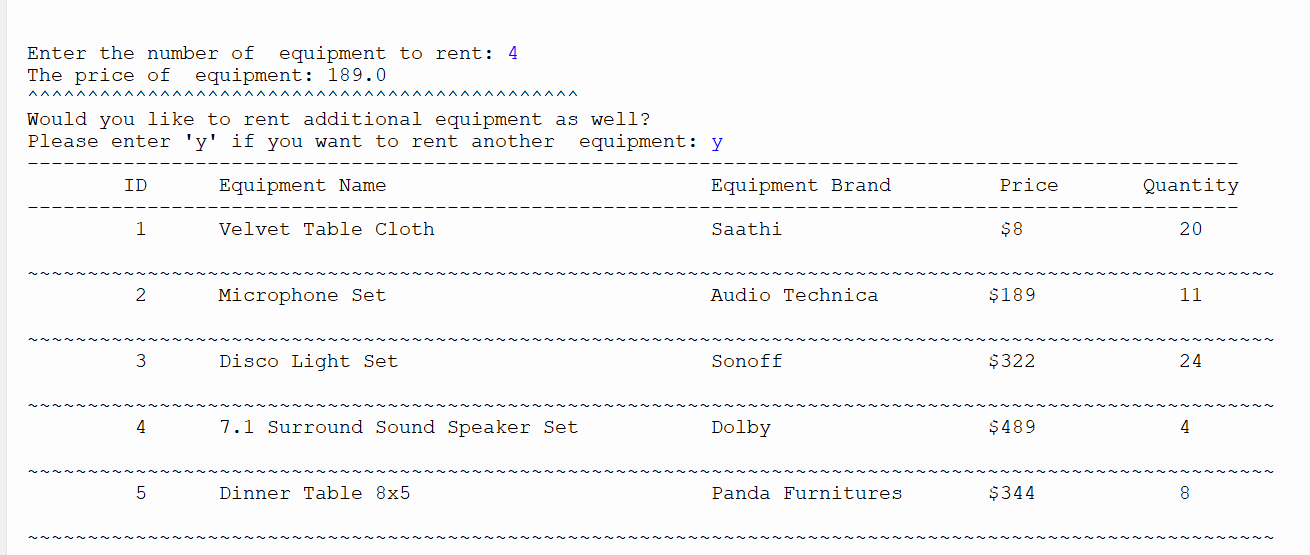


Figure 39:Test 5.3

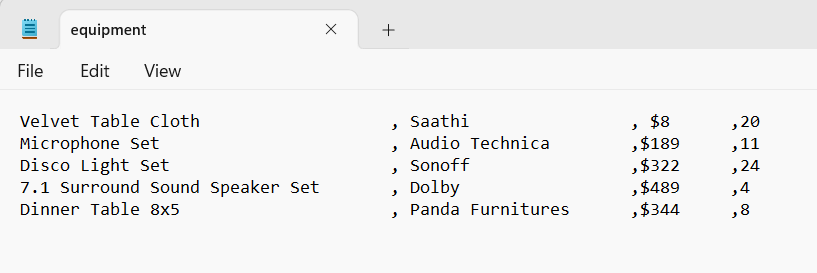


Figure 40:Test 5.4

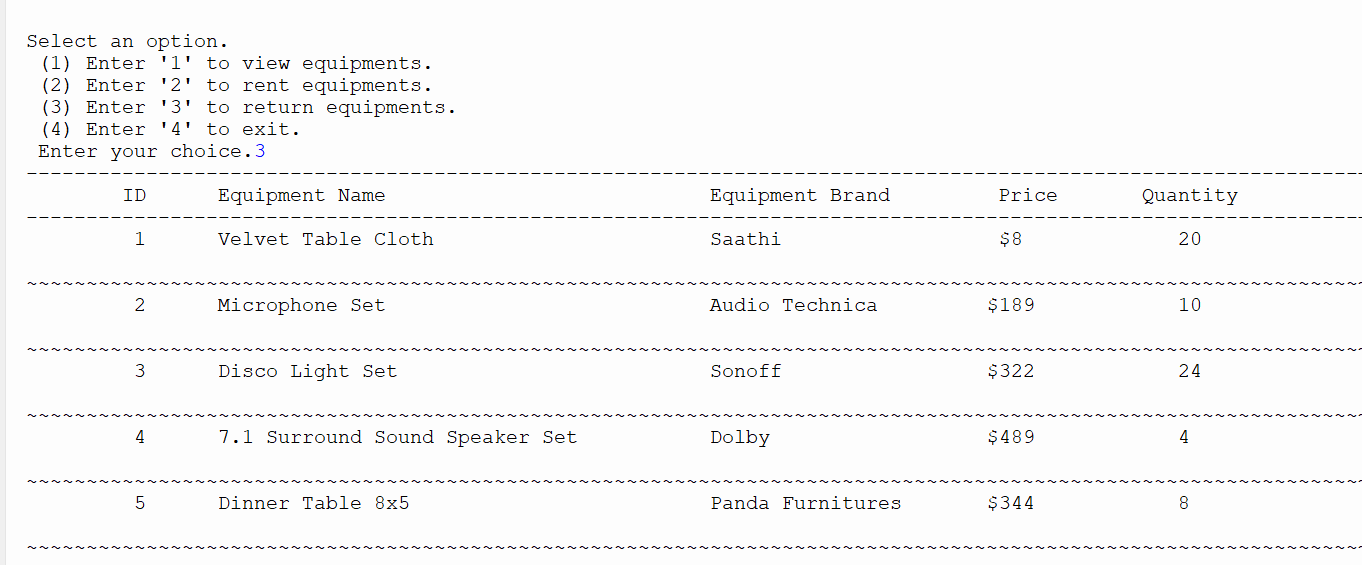


Figure 41:Test 5.5

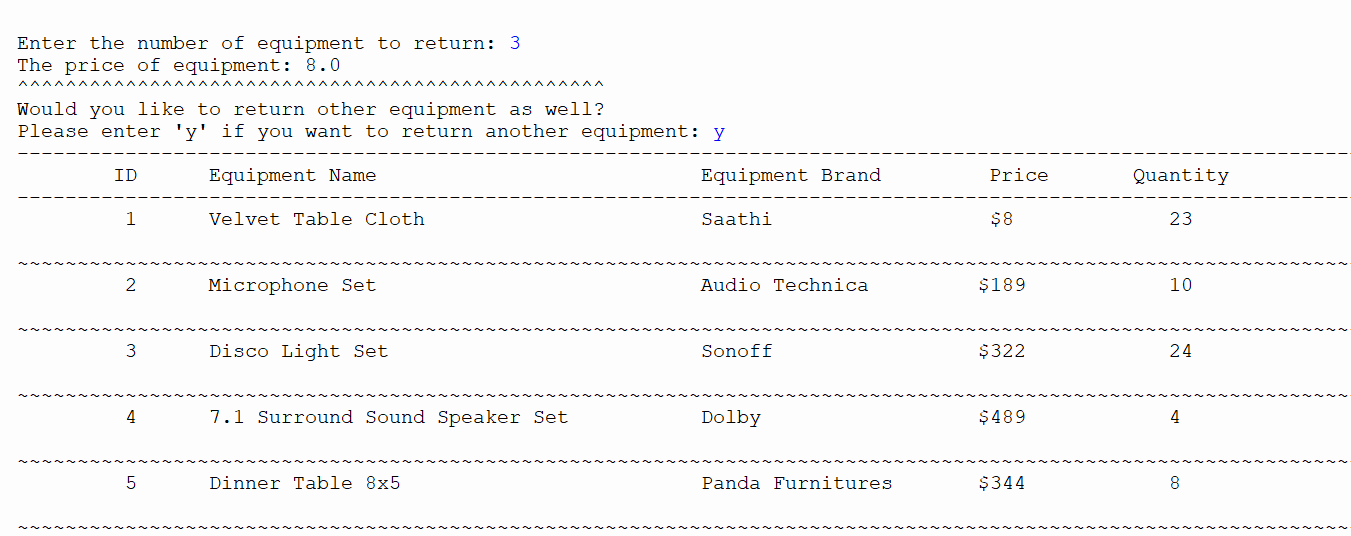


Figure 42:Test 5.6

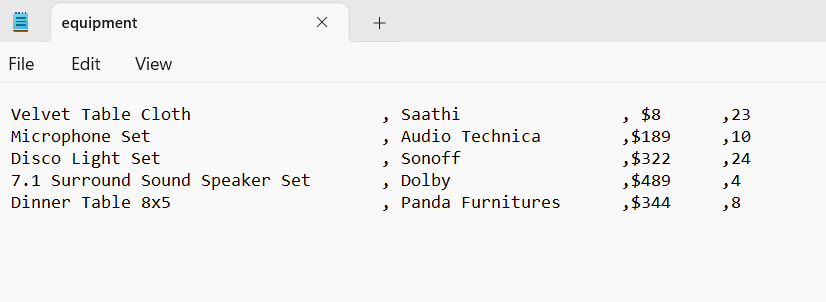


Figure 43:Test 5.7

# 5 Conclusion

Our assignment involved creating a functional rental application system for an equipment rental shop. The main aim of this task was to demonstrate our foundational understanding of the Python programming language and our ability to develop a basic yet effective application system. This project also provided an opportunity to improve our writing skills and become proficient in using the IDLE platform.

Throughout the project, we utilized draw.io to generate flowcharts, contributing to a better hold of algorithms, flowcharts, and pseudocode. Furthermore, the coursework supplies us with knowledge about working with Python dictionaries. We learned how to store and update values in text files, utilizing this skill to meet project requirements.

Without a doubt, this coursework was exceptionally informative and offered a valuable learning experience. With guidance from our instructors, lecturer, and peers, along with independent research, I successfully completed the project within the set timeframe. Having gained a deeper understanding of Python programming, I am now on the verge of a vast realm brimming with possibilities in the field. I aspire to make significant strides in this area.

I extend my gratitude to our module leader, educators, and fellow students who supported me throughout this learning journey. "Fundamentals of Computing" holds profound significance for us as students of Computing and programming, providing the essential groundwork. I express sincere appreciation to Islington College for granting me this valuable learning opportunity. This experience has left me genuinely thankful.

# 6, Appendix

## **6.1 Code for main.py**

import read

import write

import operation

print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')

print(' Welcome to Equipment Rental Shop')

print('+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++')

#display invalid message

def invalid\_message():

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

print("Invalid option. Please provide valid option.")

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

#create main function

def main():

e = False

while e == False:

try:

choice=int(input("\n\nSelect an option.\n (1) Enter '1' to view equipments. \n (2) Enter '2' to rent equipments. \n (3) Enter '3' to return equipments. \n (4) Enter '4' to exit. \n Enter your choice."))

e = True

except:

invalid\_message()

if choice == 1:

read.read()

elif choice == 2:

operation.rent\_equipments()

elif choice == 3:

write.return\_equipments()

elif choice == 4:

print('\n\n\n\t\t\tThank you for visiting us :)')

loop = False

else:

invalid\_message()

'calling main function'

loop = True

while loop == True:

main()

data = input("Do you want to continue (y/n) : ");

if data == "n" :

loop = False

else :

loop = True

## **6.2. Code for read.py**

def read():

file = open("equipment.txt", "r")

lines = file.read().splitlines()

dictionary = {}

for x in range(len(lines)):

word = lines[x].split(",")

name = word[0]

brand = word[1]

price = float(word[2].replace('$', ''))

quantity = int(word[3])

dictionary[x + 1] = {

'Name': name,

'Brand': brand,

'Price': price,

'Quantity': quantity

}

header\_row = "|{:<4} |{:<40} |{:<30} |{:<10} | {:<5}".format("ID", "Name", "Brand", "Price", "Quantity")

formatted\_rows = []

for item\_id, item\_data in dictionary.items():

formatted\_row = "|{:<4} | {:<40} |{:<30} |${:<20.3f} |{:<9}|".format(

item\_id, item\_data['Name'], item\_data['Brand'], item\_data['Price'], item\_data['Quantity'])

formatted\_rows.append(formatted\_row)

print(header\_row)

print('-' \* len(header\_row))

print('\n'.join(formatted\_rows))

print('-' \* len(header\_row))

file.close()

return dictionary

## **6.3 Code for write.py**

import datetime

'create return\_equipments function'

def return\_equipments():

'to print if okay to return'

def applicable():

print("\n------------------------------------------")

print("You have the option to return the equipment.")

print("------------------------------------------\n")

'to print if not okay to return.'

def unapplicable():

print("\n--------------------------------------------------------------")

print("The equipment you wish to return cannot be found in the database.")

print("---------------------------------------------------------------\n")

'to print if invalid input is entered.'

def invalid\_message():

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

print("invalid option. Please provide valid option.")

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

def goods(dictionary):

tfile = open("equipment.txt","w")#Opening equipment.txt in writing mode

for i in dictionary.values():

line = str(i[0] + "," + str(i[1]) + "," + str(i[2]) + "," + str(i[3]))

tfile.write(line)

tfile.write("\n")

tfile.close()

'to create a text file after each transaction'

def create\_writefile(cname, phnumber, date, total, equipmentName, equipmentBrand, cfine):

fileName = "Return\_" + cname + "\_" + str(datetime.datetime.now().second) + str(datetime.datetime.now().microsecond) + str(datetime.datetime.now().hour) + ".txt"

tfile = open(fileName, "w")#Opening fileName in writing mode

tfile.write("Name of the customer : " + cname + "\n")

tfile.write("Phone number: " + str(phnumber))

tfile.write("Date of return : " + str(date) + "\n")

tfile.write("Total price of equipment: " + str(total) + "\n")

tfile.write("Returned equipment: " + "\n")

tfile.write("Fine: " + str(cfine) + "\n")

for i, j in zip(equipmentName,equipmentBrand):

tfile.write(i + ":- " + j)

tfile.write("\n")

tfile.close()

'to calculate the total price'

def total\_price(dictionary, quantitydet, equipmentID):

price = float(dictionary[returnID][2].replace("$",""))

print("The price of equipment:", price)

priceperitem = price \* quantitydet

return priceperitem

'to take number of equipment to return as input'

def quantity\_equipment(quantity\_goods):

exc = False

while exc == False:

try:

quantity = int(input("\nEnter the number of equipment to return: "))

exc = True

except:

invalid\_message()

while quantity <=0 :

if quantity <= 0:

print("\n----------------------------------------")

print("Input is invalid. Please provide valid input.")

print("---------------------------------------------\n")

ex = False

while ex == False:

try:

quantity = int(input("Please enter a valid quantity: "))

ex = True

except:

invalid\_message()

return quantity

'to create dictionary and store data in it.'

def return\_dictionary():

tfile = open("equipment.txt", "r")#Opening data.txt in reading mode

counter = 0

dictionary = {}#creating dictionary

for b in tfile:

counter = counter + 1

b = b.replace("\n","")

b = b.split(',')

dictionary[counter] = b

return dictionary

tfile.close()

'to display data from the textfile in an organized way'

def return\_display():

tfile = open("equipment.txt","r")

print("-----------------------------------------------------------------------------------------------------------------")

print("\tID \tEquipment Name Equipment Brand Price Quantity")

print("-----------------------------------------------------------------------------------------------------------------")

counter = 0

for b in tfile:

counter = counter + 1

print("\t", counter, "\t" + b.replace(",","\t"))

print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

tfile.close()

'to check if the id is valid'

def validID():

exception = False

while exception == False:

try:

valid\_ID = int(input("Please enter the ID of equipment you want to return: "))

exception = True

except:

invalid\_message()

while valid\_ID <= 0 or valid\_ID > len(return\_dictionary()):

try:

print("\nxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

print("\nThe provided equipment ID is not valid. Please attempt again.\n")

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\n")

return\_display()

valid\_ID = int(input("\n Please enter the ID of equipment you want to return: "))

except:

invalid\_message()

return valid\_ID

'calling functions'

return\_display()

dictionary = return\_dictionary()

returnID = validID()

returnNamelist = []

returnBrandlist = []

if int(dictionary[returnID][3]) > 0:

applicable()

quantity = quantity\_equipment(int(dictionary[returnID][3]))

dictionary[returnID][3] = int(dictionary[returnID][3]) + quantity#Adding quantity after returning

returnNamelist.append(dictionary[returnID][0])

returnBrandlist.append(dictionary[returnID][1])

goods(dictionary)

totalPrice = total\_price(dictionary,quantity,returnID)

print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")

print("Would you like to return other equipment as well?")

returnanother = input("Please enter 'y' if you want to return another equipment: ").lower()

loop = True

while loop == True:

if returnanother == "y":

return\_display()

dictionary = return\_dictionary()

returnID = validID()

if int(dictionary[returnID][3]) > 0:

applicable()

quantity = quantity\_equipment(int(dictionary[returnID][3]))

dictionary[returnID][3] = int(dictionary[returnID][3]) + quantity

returnNamelist.append(dictionary[returnID][0])

returnBrandlist.append(dictionary[returnID][1])

goods(dictionary)

totalPrice = total\_price(dictionary,quantity,returnID) + totalPrice

print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")

print("Do you want to return another equipment as well?")

returnanother = input("Please enter 'y' if you want to return another equipment: ").lower()

else:

fineperequipment = 5

Totalfine = 0

name = input(" Pleade enter the name of customer: ")

ab = True

while ab == True:

try:

number = int(input("Please enter your phone number: "))

ab = False

except:

invalid\_message()

cd = True

while cd == True:

try:

daysafter = int(input("How many days have passed since you rented it? Please enter: "))

cd = False

except:

invalid\_message()

tDate = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

listname = (str(returnNamelist)[1:-1])

listbrand = (str(returnBrandlist)[1:-1])

if daysafter > 5:

extradays = daysafter - 5

for k in returnNamelist:

Totalfine += extradays\*int(quantity)\*fineperequipment

else:

Totalfine = 0

Totalcost = totalPrice + Totalfine

'To print invoice/bill'

print("\n-------------------------------------------------------------------")

print("\t\t\t\t INVOICE")

print("--------------------------------------------------------------------\n")

print("\t\t\t\t\t", tDate)

print("Customer Name:", name)

print("Phone number: ", number)

print("Total price of equipment:"," $", totalPrice)

print("Name of returned equipment:", listname)

print("Brand of returned equipment:", listbrand)

print("Fine: ","$", Totalfine)

print("Total cost: ", Totalcost)

print("\n")

print("\n The equipment has been returned.\n")

print("\t\t\t\t Thank you.")

create\_writefile(name, number, tDate, totalPrice, listname, listbrand, Totalfine)

loop = False#loop break

else:

unapplicable()

## **6.4 Code for operation.py**

import datetime

'to create function rent\_equipments'

def rent\_equipments():

'to print if the equipment is applicable to rent.'

def applicable():

print("\n--------------------------------------------------")

print("The equipment you want to rent is applicable.")

print("----------------------------------------------------\n")

'to print if the equipment is not applicable.'

def unapplicable():

print("\n--------------------------------------------")

print("The equipment is unapplicable at the moment.")

print("-------------------------------------------\n")

'to print if invalid input is entered'

def invalid\_message():

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

print("Invalid option. Please provide valid option.")

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

def goods(dictionary):

tfile = open("equipment.txt","w")

for i in dictionary.values():

line = str(i[0] + "," + str(i[1]) + "," + str(i[2]) + "," + str(i[3]))

tfile.write(line)

tfile.write("\n")

tfile.close()

'to create a text file after each transaction'

def create\_operationfile(cname, phnumber, date, total, name\_equipment, brand\_equipment):

tfile\_name = "Rent\_" + cname + "\_" + str(datetime.datetime.now().second) + ".txt"

tfile = open(tfile\_name, "w")

tfile.write("Name of the customer : " + cname + "\n")

tfile.write("Phone number: " + str(phnumber))

tfile.write("Date of rent : " + str(date) + "\n")

tfile.write("Total price of equipment: " + str(total) + "\n")

tfile.write("Rented equipment: " + "\n")

for i, j in zip(name\_equipment, brand\_equipment):

tfile.write(i + ":- " + j)

tfile.write("\n")

tfile.close()

'to calculate the total price per item'

def total\_price(dictionary, quantitydet, equipmentID):

price = float(dictionary[rentID][2].replace("$",""))

print("The price of equipment:", price)

priceperitem = price \* quantitydet

return priceperitem

'to take the number of equipment to rent as input.'

def quantity\_equipment(quantity\_goods):

exc = False

while exc == False:

try:

quantity = int(input("\nEnter the number of equipment to rent: "))

exc = True

except:

invalid\_message()

while quantity <=0 or quantity > quantity\_goods:

if quantity <= 0:

print("\n---------------------------------------------")

print("Input is invalid. Please provide valid input.")

print("------------------------------------------------\n")

elif quantity > quantity\_goods:

print("\n------------------------------------------------------------------------------------")

print("The quantity you've entered exceeds our available stock. Please input a new quantity..")

print("------------------------------------------------------------------------------------\n")

ex = False

while ex ==False:

try:

quantity = int(input("Please enter a valid quantity: "))

ex = True

except:

invalid\_message()

return quantity

'to create dictionary'

def rent\_dictionary():

tfile = open("equipment.txt", "r")

counter = 0

dictionary = {}

for b in tfile:

counter = counter + 1

b = b.replace("\n","")

b = b.split(',')

dictionary[counter] = b

return dictionary

tfile.close()

'to display contents from textfile in an organized way'

def rent\_display():

tfile = open("equipment.txt","r")

print("-----------------------------------------------------------------------------------------------------")

print("\tID \tEquipment Name Equipment Brand Price Quantity")

print("-----------------------------------------------------------------------------------------------------")

counter = 0

for a in tfile:

counter = counter + 1

print("\t", counter, "\t" + a.replace(",","\t"))

print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")

tfile.close()

'to check if the id entered is valid or not'

def validID():

exception = False

while exception == False:

try:

valid\_ID = int(input("Please enter the ID of equipment you want to rent: "))

exception = True

except:

invalid\_message()

while valid\_ID <= 0 or valid\_ID > len(rent\_dictionary()):

try:

print("\nxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")

print("\nThe provided equipment ID is not valid. Please attempt again.\n")

print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\n")

rent\_display()

valid\_ID = int(input("\nEnter the ID of equipment you want to rent: "))

except:

invalid\_message()

return valid\_ID

rent\_display()

dictionary = rent\_dictionary()

rentID = validID()

rentNamelist = []

rentBrandlist = []

if int(dictionary[rentID][3]) > 0:

applicable()

quantity = quantity\_equipment(int(dictionary[rentID][3]))

dictionary[rentID][3] = int(dictionary[rentID][3]) - quantity

file = open("equipments.txt","w")

for values in dictionary.values():

file.write(str(values[0])+","+str(values[1])+","+str(values[2])+","+str(values[3]))

file.write("\n")

file.close()

rentNamelist.append(dictionary[rentID][0])

rentBrandlist.append(dictionary[rentID][1])

goods(dictionary)

totalPrice = total\_price(dictionary,quantity,rentID)

print("^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^")

print("Would you like to rent additional equipment as well?")

rentanother = input("Please enter 'y' if you want to rent another equipment: ").lower()

loop = True

while loop == True:

if rentanother == "y":

rent\_display()

dictionary = rent\_dictionary()

rentID = validID()

if int(dictionary[rentID][3]) > 0:

applicable()

quantity = quantity\_equipment(int(dictionary[rentID][3]))

dictionary[rentID][3] = int(dictionary[rentID][3]) - quantity

file = open("equipment.txt","w")

for values in dictionary.values():

file.write(str(values[0])+","+str(values[1])+","+str(values[2])+","+str(values[3]))

file.write("\n")

file.close()

rentNamelist.append(dictionary[rentID][0])

rentBrandlist.append(dictionary[rentID][1])

goods(dictionary)

totalPrice = total\_price(dictionary,quantity,rentID) + totalPrice

print("=======================================================================")

print("Do you also wish to rent other equipment?")

rentanother = input("Please enter 'y' if you want to rent another equipment: ").lower()

else:

name = input("Please enter the name of customer: ")

ab = True

while ab == True:

try:

number = int(input("Please enter your phone number:"))

ab = False

except:

invalid\_message()

tDate = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

listname = (str(rentNamelist)[1:-1])

listbrand = (str(rentBrandlist)[1:-1])

print("\n-------------------------------------------------------------------")

print("\t\t\t\t INVOICE")

print("--------------------------------------------------------------------\n")

print("\t\t\t\t\t", tDate)

print("Customer Name:", name)

print("Phone number: ", number)

print("Total price:"," $", totalPrice)

print("Name of rented equipment:", listname)

print("Brand of rented equipment:", listbrand)

print("\n")

print("The equipment has been rented.")

print("Ensure that you bring back the equipment within a 5-day period to prevent any fines.")

print("\t\t\t\t Thank you.")

create\_operationfile(name, number, tDate, totalPrice, listname, listbrand)

loop = False #loop break

else:

unapplicable()