

**Module Code & Module Title CS4051**

**Fundamentals of Computing**

**Assessment weightage & Type**

**60% Individual Coursework**

**Year and Semester**

**2024 Autum**

**Student’s Name: Janak Devkota**

**Group: C4**

**College ID: NP04CP4A230067**

**London Met ID: 230488060**

**Assessment Due Date: March 19, 2024**

**Assessment Submission Date: May 7, 2024**

Table of Contents

[1. Introduction 1](#_Toc165035293)

[1.1. Visual Studio Code 2](#_Toc165035294)

[1.2. Draw.io 2](#_Toc165035295)

[1.3. Python 2](#_Toc165035296)

[2. Discussion and Analysis 3](#_Toc165035297)

[**2.1. Algorithm** 3](#_Toc165035298)

[2.2. Flowchart 7](#_Toc165035299)

[**2.3. PseudoCode** 8](#_Toc165035300)

[**- PseudoCode of Main.py** 9](#_Toc165035301)

[**- PseudoCode of Read.py** 13](#_Toc165035302)

[**- PseudoCode of Write.py** 15](#_Toc165035303)

[**- PseudoCode of Operation.py** 16](#_Toc165035304)

[**2.4. Data Structure** 26](#_Toc165035305)

[3. Program 29](#_Toc165035306)

[i. Module: Main.py 29](#_Toc165035307)

[ii. Module: Read.py 30](#_Toc165035308)

[iii. Module: Write.py 30](#_Toc165035309)

[iv. Module: Operation.py 31](#_Toc165035310)

[4. Testing 32](#_Toc165035311)

[- Test 1 32](#_Toc165035312)

[- Test 2 33](#_Toc165035313)

[- Test 3 34](#_Toc165035314)

[- Test 4 35](#_Toc165035315)

[- Test 5 36](#_Toc165035316)

[5. Conclusion 37](#_Toc165035317)

[6. Bibliography 38](#_Toc165035318)

[7. Appendix 38](#_Toc165035319)

[- Main.py 38](#_Toc165035320)

[- Read.py 42](#_Toc165035321)

[- Write.py 43](#_Toc165035322)

[- Operation.py 44](#_Toc165035323)

**Table of Figures**

[Figure 1: Demo Figure 1](#_Toc133497508)

# 1. Introduction

In this project, I'm diving into creating a smart system for TechnoPropertyNepal, a big company in Nepal that rents out land. This system is all about making it easier for people to rent land from them. In the old days, renting land was a lot of work because everything was done by hand. But now, with new fancy technology, we need to update how things are done to make it faster and more accurate.

So, my idea is to use a mix of pictures, step-by-step instructions, and special codes to make renting land super easy. Instead of keeping all the land information in different places, we'll put it all in one computer file. This way, anyone can quickly see what land is available to rent.

One cool thing about the system is that it can automatically update when someone rents land. It'll make a detailed note with all the important details, like who rented the land, for how long, and how much they need to pay.

And when the land is returned, the system will make another note to keep track of everything. It can even handle special situations, like if someone is late returning the land or wants to rent it for longer.

The main focus of this course work is to find students' understanding and capability, in this course work students are to make Land renting system where a person he/she is to Purchased or return the Land and weather the rented Land are returned in the given period of time if he/she doesn't then they have to pay the fine for the returned Land delay.

So, this project is my chance to show what I know about making a really good land renting system. I'll use the Python programming language and follow the best ways to make sure it works great for TechnoPropertyNepal and sets a new standard for renting land in Nepal. The following technologies are used to construct the renting system.

## 1.1. Visual Studio Code

VS Code, also known as Visual Studio Code, stands out as an exemplary Integrated Development Environment (IDE), offering lightning-fast performance coupled with a robust set of features tailored for efficient code editing. Its primary functionality as a swift source code editor makes it an indispensable tool for routine development tasks. Boasting extensive language support and a plethora of utilities such as syntax highlighting, bracket matching, auto-indentation, box selection, and snippets, Visual Studio Code simplifies the process of getting projects off the ground swiftly. Leveraging minor tweaks, intelligent keyboard shortcuts, and straightforward text manipulation, navigating through codebases becomes a seamless experience. In my coursework, I harnessed the power of Visual Studio Code to craft Python scripts, utilizing community-contributed keyboard shortcut mappings. While Visual Studio Code excels in day-to-day coding endeavors, more sophisticated tools with deeper code understanding are requisite for tackling complex programming challenges (Visual Studio Code, 2023).

## 1.2. Draw.io

In our coursework, Draw.io is our go-to tool for creating flowcharts for our programs. It simplifies the process of drawing diagrams and charts, even for those who aren't experts. With its user-friendly interface, we can effortlessly generate flowcharts, mind maps, or network diagrams with just a few clicks. It's particularly useful for brainstorming sessions and visually explaining concepts. The fact that it's free and accessible directly in our web browser eliminates the hassle of downloads. Whether we're students crafting study guides or professionals mapping out projects, Draw.io proves invaluable for organizing ideas in a clear and efficient manner.

## 1.3. Python

Python is a high-level programming language known for its simplicity and versatility. It emphasizes readability and concise syntax, making it ideal for beginners and experts alike. With extensive libraries and frameworks, Python excels in web development, data analysis, artificial intelligence, and more. Its interpreted nature allows for quick prototyping and testing. Python's community-driven approach fosters a vast ecosystem of resources, tutorials, and support. Whether you're building web applications, automating tasks, or conducting scientific research, Python offers robust tools and a thriving community to help you succeed. Its widespread adoption across industries underscores its status as a premier language for modern software development.

# 2. Discussion and Analysis

## **2.1. Algorithm**

Step 1: Display the main menu offering options to:

1. Purchase Land

2. Return Previously Purchased Land

3. View Owned Lands

4. Exit the program

Step 2: Input user's choice, x

Step 3: If x == 1 then,

Go to Step 4

Otherwise, if x == 2 then,

Go to Step 5

Otherwise, if x == 3 then,

Go to Step 6

Otherwise, if x == 4 then,

Go to Step 7

Otherwise,

Display 'Invalid Choice!'

Return to Step 1

Step 4:

Invoke the function to handle land purchase:

Step 4.1: Input User confirmation to purchase lands.

Step 4.2: If user confirms (yes),

Go to Step 4.3

Otherwise,

Display 'Going back to main menu.'

Go to Step: 1

Step 4.3: Retrieve land data from the file.

Step 4.4: Input the user name.

Step 4.5: Validate name to ensure it contains no digits.

Step 4.6: Input users contact number.

Step 4.7: Validate contact to ensure it's a 10-digit number.

Step 4.8: Input No: of months they want to purchase lands for.

Step 4.9: Validate months to ensure it's a positive integer.

Step 4.10: Input the user if they want to purchase multiple lands.

Step 4.11: If user wants to purchase multiple lands,

Go to Step 4.12

Otherwise, if user wants to purchase only one land,

Go to Step 4.15

Otherwise,

Display 'Invalid choice.'

Return to Step 4.10

Step 4.12: Input the user for land ID or type 'done' to finish.

Step 4.13: If user enters 'done',

Go to Step 4.14

Otherwise,

Validate land ID and check its availability.

If available, add it to the list of selected land IDs.

If not available, Show message not available.

Return to Step 4.12

Step 4.14: If the list of selected land IDs is empty,

Display 'Empty Data Found. Cannot purchase!'

Return to Step 1

Otherwise,

Go to Step 4.15

Step 4.15: Generate invoices for the selected lands.

Step 4.16: Write the generated invoices to files.

Step 4.17: Update the data file with modified land availability.

Step 4.18: Display Success message with the land IDs and duration.

Return to Step 1

Step 5:

Invoke the function to handle returning previously purchased land:

Step 5.1: Input confirmation to return purchased lands.

Step 5.2: If user confirms (yes),

Go to Step 5.3

Otherwise,

Display 'Going back to main menu.'

Return to Step 1

Step 5.3: Retrieve land data from the file.

Step 5.4: Input the user for their name.

Step 5.5: Check if the user has any previously purchased lands.

Step 5.6: If user has purchased lands,

Go to Step 5.7

Otherwise,

Display 'No purchased lands found for this user.'

Return to Step 1

Step 5.7: Input No: of months they used the purchased lands.

Step 5.8: Validate months to ensure it's a positive integer.

Step 5.9: Input the land(id's) they want to return.

Step 5.10: If user wants to return multiple lands,

Go to Step 5.11

Otherwise, if user wants to return only one land,

Go to Step 5.14

Otherwise,

Display 'Invalid choice.'

Return to Step 5.9

Step 5.11: Input land ID or type 'done' to finish.

Step 5.12: If user enters 'done',

Go to Step 5.13

Otherwise,

Validate land ID and check its availability for return.

If available, add it to the list.

If not available, show message not available.

Return to Step 5.11

Step 5.13: If the list of selected land IDs for return is empty,

Display 'Empty Data Found. Cannot return!'

Return to Step 1

Otherwise,

Go to Step 5.14

Step 5.14: Generate invoices of returned lands with any applicable fines.

Step 5.15: Write the generated invoices to files.

Step 5.16: Update the data file with modified land availability.

Step 5.17: Display a Success message with land IDs and fines.

Return to Step 1

Step 6:

Display a list of owned lands with their details.

Return to Step 1

Step 7:

Display 'Exiting the program.'

Exit the program

## 

## 2.2. Flowchart

Flowchart is a graphical representation of steps in a sequential order which is used to present the flow of process of program.

A group of white rectangles with black text

Description automatically generated

## **2.3. PseudoCode**

Pseudocode is like a rough sketch for programming. It's a mix of plain language and code-like structures used to outline algorithms. For example, if you're making a sandwich, pseudocode might say: "1. Get bread. 2. Put ingredients on bread. 3. Put another bread on top." It's not real code but helps plan the logic. In coursework, pseudocode is handy for designing algorithms before actual coding. Think of it as a bridge between human understanding and computer language, helping you map out steps before diving into the technicalities.

### **- PseudoCode of Main.py**

algorithm main

import os as s

from operation import \*

from read import \*

from write import \*

display\_menu()

user\_choice = get\_user\_choice()

create\_directories()

if user\_choice == 1 then

purchased\_land()

else if user\_choice == 2 then

returned\_land()

else if user\_choice == 3 then

show\_lands()

else if user\_choice == 4 then

print("Exiting the program.")

exit()

else

print("Invalid choice. enter between 1 and 4.")

main()

end if

end main

algorithm display\_menu

print("

1. Purchase Land

2. Return Previously Purchased Land

3. View Owned Lands

4. Exit the program

")

end display\_menu

algorithm get\_user\_choice

user\_choice = validation(input("Please choose form menu: "))

return user\_choice

end get\_user\_choice

algorithm validation

input user\_choice

while not user\_choice.isdigit() do

print("Sorry, this is not valid. Please enter a number.")

user\_choice = input("Please choose an option from the menu: ")

end while

return int(user\_choice)

end validation

algorithm create\_directories

directories = ['Invoices/purchased', 'Invoices/return']

for each directory in directories do

if not s.path.exists(directory) then

s.makedirs(directory)

print(f"Created directory: {directory}")

end if

end for

end create\_directories

algorithm show\_lands

land\_data = show\_only\_values()

header = format\_header()

print(header)

print(separator)

display\_land\_records(land\_data)

print(separator)

main()

end show\_lands

algorithm format\_header

header = "{Space format}".format(

header all topic")

separator = "-" \* length\_of(header)

return header

end format\_header

algorithm display\_land\_records(land\_data)

for each record in land\_data do

print("{Space format}".format(\*record))

end for

end display\_land\_records

algorithm purchased\_land

choice = input("Conformation (yes or (Enter) for main menu): ")

if choice == "yes" then

user\_info = get\_refined\_user\_purchased\_info()

purchased\_data,\_datas\_for\_txt\_=\_user\_purchased\_data(user\_info)

list\_invoices, total = generate\_purchased\_invoices(purchased\_data)

if list\_of\_writable\_invoices then

generate\_invoic(list\_writable, user\_info, total)

save\_modified\_data\_to\_file(datas\_to\_write\_into\_data\_txt)

print("successfully purchased the land.")

else

print("Empty Data Found Can't purchase!")

end if

main()

else

print("Going back to the main menu.")

main()

end if

end purchased\_land

algorithm returned\_land

choice = input("Conformation (yes or (Enter) for main menu): ")

if choice == "yes" then

user\_information = get\_refined\_user\_return\_info()

user\_return\_data,\_returned\_datas\_txt\_=\_user\_return\_data(user\_information)

list\_invoices, total = generate\_invoices(user\_return\_data)

if list\_of\_returnable\_invoices then

write\_invoices\_file(list\_invoices, user\_information, total)

save\_modified\_file(returned\_datas\_to\_write\_into\_data\_txt)

print("successfully returned the land.")

else

print("Empty Data Found Can't Return!")

end if

main()

else

print("Going back to the main menu.")

main()

end if

end returned\_land

### **- PseudoCode of Read.py**

algorithm data\_into\_list

input filename

if filename is not provided then

set filename to "data.txt"

end if

open filename for reading

read contents of file

split contents by newline character into data

initialize an empty list called result

for each line in data do

if line is empty then

continue to the next iteration

end if

split line by ", " into values

create a dictionary called entry with keys

{"id",

"location",

"direction",

"anna",

"price", and

"availability"}

set entry["id"] to values[0]

set entry["location"] to values[1]

set entry["direction"] to values[2]

set entry["anna"] to values[3]

set entry["price"] to values[4]

set entry["availability"] to values[5]

append entry to result

end for

return result

end data\_into\_list

algorithm show\_only\_values

data = data\_into\_list()

initialize an empty list called values\_list

for each record in data do

only\_values = get values from record

append only\_values to values\_list

end for

return values\_list

end show\_only\_values

### **- PseudoCode of Write.py**

algorithm save\_modified\_data\_to\_file

input datas

open "data.txt" for writing

for each data in datas do

modified\_data = concatenate values of data with ", " separator "/n"

write modified\_data to file

end for

end save\_modified\_data\_to\_file

algorithm write\_generated\_invoice\_to\_file

input list\_of\_writable\_invoices, user\_info, allTotal

name = get name from user\_info

time = take time of now

file\_path = "Invoices/purchased/" + time +name + "Purchased.txt"

open file at file\_path for writing

for each invoice in list\_of\_writable\_invoices do

write invoice to file followed by two newline characters

end for

open file at file\_path for appending

append string " ALL TOTAL: {}"

end write\_generated\_invoice\_to\_file

algorithm write\_generated\_return\_invoice\_to\_file

input list\_of\_returnable\_invoices, user\_info, allTotal

name = get name from user\_info

time = take time now

file\_path = "Invoices/return/" + time + name + "Returned.txt"

open file at file\_path for writing

for each invoice in list\_of\_returnable\_invoices do

write invoice to file followed by two newline characters

end for

open file at file\_path for appending

append string " ALL TOTAL: {}"

end write\_generated\_return\_invoice\_to\_file

### **- PseudoCode of Operation.py**

algorithm get\_refined\_user\_purchased\_info

user\_data := an empty dictionary

land\_data := an empty dictionary

purchase\_multiple := input("Do you want to Rent multiple lands? (yes/no): ")

if purchase\_multiple is "yes" then

while True do

land\_id := input("Enter a Land ID (and type 'done' to finish): ")

if land\_id is "done" then

break

else

availability := purchasing\_availability\_check(land\_id)

if availability then

if land\_id is in land\_data then

print(f"You've already specified the duration for land")

else

while True do

month := input(f"months would you like to rent this kitta ?: ")

if purchased\_month.isdigit() and int(purchased\_month) > 0 then

land\_data[land\_id] := convert purchased\_month to integer

break

else

print("Please enter a valid number of months.")

else

print(f"land not available in our records for purchase.")

end while

else if purchase\_multiple is "no" then

while True do

land\_id := input("Enter a Land ID: ")

availability := purchasing\_availability\_check(land\_id)

if availability then

month := input(f"How many months would you like to rent this kitta:")

if purchased\_month.isdigit() and int(purchased\_month) > 0 then

land\_data[land\_id] := convert purchased\_month to integer

break

else

print("Please enter a valid number of months.")

else

print(f"land not available in our records for purchase.")

end while

else

print(f"Please enter 'yes' or 'no'.")

end if

user\_data['land\_data'] := land\_data

while True do

name := input("Enter your name: ")

if name.strip() then

if not any(char.isdigit() for char in name) then

user\_data['name'] := name

break

else

print(f"Please enter a valid name.")

else

print("Name cannot be empty")

end while

while True do

contact := input("📞 Enter your contact number: ")

if contact.isdigit() and len(contact) is 10 then

user\_data['contact'] := contact

break

else

print("Please enter a 10-digit number.")

end while

return user\_data

end algorithm get\_refined\_user\_purchased\_info

algorithm checking\_availability(entered\_kitta)

datas = data\_into\_list()

for each data in datas do

if data["id"] = entered\_kitta and data["availability"] is "Available" then

return list containing entered\_kitta

end for

print "This land {entered\_kitta} is not available in our records"

return empty list

end checking\_availability

algorithm user\_purchased\_full\_data(user\_info, raw\_datas)

initialize an empty list called purchased\_list

set datas\_to\_write to raw\_datas

timestamp = get current date and time as a string"

for each data in datas\_to\_write do

if str(data["id"]) is in user\_info["land\_ids"] then

price = convert data["price"] to an integer

purchased\_duration = calculate\_purchased\_duration(user\_info)

vatAmount, grandTotal = calculate\_grand\_total\_and\_vat(price)

Remarks = "null"

if data["availability"] is "Available" then

set data["availability"] to "Not Available"

end if

create a dictionary called data\_with\_user\_info with keys

{

"id",

"location", "direction", "Anna", "price",

"Remarks", "name", "contact", "months", "vat\_amount",

"Grand\_Total", "Timestamp", and

"duration"

}

set data\_with\_user\_info["id"] to data["id"]

set data\_with\_user\_info["location"] to data["location"]

set data\_with\_user\_info["direction"] to data["direction"]

set data\_with\_user\_info["Anna"] to data["anna"]

set data\_with\_user\_info["price"] to data["price"]

set data\_with\_user\_info["Remarks"] to Remarks

set data\_with\_user\_info["name"] to user\_info["name"]

set data\_with\_user\_info["contact"] to user\_info["contact"]

set data\_with\_user\_info["months"] to user\_info["month"]

set data\_with\_user\_info["vat\_amount"] to vatAmount

set data\_with\_user\_info["Grand\_Total"] to grandTotal

set data\_with\_user\_info["Timestamp"] to timestamp

set data\_with\_user\_info["duration"] to purchased\_duration

append data\_with\_user\_info to purchased\_list

end if

end for

return purchased\_list, datas\_to\_write

end user\_purchased\_full\_data

algorithm calculate\_grand\_total\_and\_vat(prices)

vat\_percentage = 13

vatAmount = (prices \* vat\_percentage) / 100

grand\_total = prices + vatAmount

return vatAmount, grand\_total

end calculate\_grand\_total\_and\_vat

algorithm calculate\_purchased\_duration(user\_info)

purchased\_for = user\_info["month"]

timestamp = get current date and time as a string

date\_expried = split timestamp by "-"

this\_month = convert date\_expried[1] to an integer

exp\_new\_month = this\_month + purchased\_for

set date\_expried[1] to str(exp\_new\_month)

final\_date = join date\_expried with "-"

return final\_date

end calculate\_purchased\_duration

algorithm generate\_purchased\_invoices(datas)

initialize an empty list called invoices

initialize allTotal to 0

for each item in datas do

add item["Grand\_Total"] to allTotal

create a formatted invoice string

append invoice to invoices

end for

return invoices, allTotal

end generate\_purchased\_invoices

algorithm get\_refined\_user\_return\_info

initialize an empty dictionary called user\_data

set user\_data["land\_ids"] to the result of get\_selected\_return\_land\_ids function

input name

if name is not empty and check\_filename(name) is true then

set user\_data["name"] to name

input contact

if contact is a valid 10-digit number then

set user\_data["contact"] to contact

input and validate months

set user\_data["months"] to validated months

return user\_data

end get\_refined\_user\_return\_info

algorithm get\_selected\_return\_land\_data

land\_data := an empty dictionary

return\_multiple := input("Do you want to return multiple lands? (yes/no): ")

if return\_multiple is "yes" then

while True do

land\_id := input("Enter a Kitta no (type 'done' to finish): ")

if land\_id.lower() is "done" then

break

elif not land\_id.isdigit() then

print("Invalid Kitta NO. Please enter a number.")

continue

availability := checking\_availability\_of\_return(land\_id)

if availability then

if land\_id is in land\_data then

print(f"You've already specified the duration for land ID {land\_id}.")

else

while True do

month := input(f"In how many months youreturn {land\_id} kitta: ")

if return\_month.isdigit() and int(return\_month) > 0 then

land\_data[land\_id] := convert return\_month to integer

break

else

print("Please enter a valid number of months.")

else

print(f"This land {land\_id} is not available so, we can't return.")

else if return\_multiple is "no" then

while True do

land\_id := input("Enter a Land ID: ")

availability := checking\_availability\_of\_return(land\_id)

if availability then

month := input(f"In how many months you return {land\_id} kitta: ")

if return\_month.isdigit() and int(return\_month) > 0 then

land\_data[land\_id] := convert return\_month to integer

break

else

print("Please enter a valid number of months.")

else

print(f"This land {land\_id} is not available in our records for return.")

else

print("Invalid choice. Please enter 'yes' or 'no'.")

return land\_data

end algorithm get\_selected\_return\_land\_data

algorithm checking\_availability\_of\_return

datas = data\_into\_list()

for each data in datas

if data['id'] is equal to land\_id and data['availability'] is 'Not Available' then

return True

return False

end checking\_availability\_of\_return

algorithm check\_filename

file\_path = "Invoices/purchased/"

files = s.listdir(file\_path)

for each file\_name in files

part = split file\_name by '\_' and take the last part

name\_part = split part by 'Purchased.txt' and take the first part

if user\_name is equal to name\_part then

return file\_name

return None

end check\_filename

algorithm user\_return\_full\_data

returned\_list = []

datas\_to\_write = raw\_datas

timestamp = current timestamp

for each data in datas\_to\_write

if str(data['id']) is in user\_info['land\_ids'] then

month = convert user\_info['months'] to integer

fine = calculate\_fine(month, user\_info)

Remarks = "null"

if data['availability'] is 'Not Available' then

data['availability'] = 'Available'

create a dictionary called data\_with\_user\_info with keys and values:

'id': data['id']

'location': data['location']

'direction': data['direction']

'Anna': data['anna']

'price': data['price']

'Remarks': Remarks

'name': user\_info['name']

'contact': user\_info['contact']

'months': user\_info['months']

'fine': fine

'Timestamp': timestamp

add data\_with\_user\_info to returned\_list

return returned\_list, datas\_to\_write

end user\_return\_full\_data

algorithm grab\_purchased\_month

expiration\_months = empty dictionary

kittano = None

name = user\_info["name"]

file\_paths = list files in "Invoices/purchased/" where name is in file

for each file\_name in file\_paths

file\_path = concatenate "Invoices/purchased/" and file\_name

open file\_path for reading as file

for each line in file

if "| null |" is in line then

kittano = extract kittano from line

kitta = convert kittano to integer

end if

if "Months:" is in line and kittano is not None then

purchased\_date = extract purchased date from line

purchase\_month = convert purchased date to integer

add (kitta, purchase\_month) to expiration\_months dictionary

kittano = None

end if

end for

end for

return expiration\_months

end grab\_purchased\_month

algorithm calculate\_fine

month\_purchased = grab\_purchased\_month(user\_return\_info)

fine\_per\_month = 1000

fine = 0

if month\_return > month\_purchased then

over\_time = month\_return - month\_purchased

fine = over\_time \* fine\_per\_month

return fine

end calculate\_fine

algorithm generate\_returned\_invoices

invoices = []

allTotal = 0

for each item in datas

calculate price from item

calculate fine from item

calculate vat\_amount as 13% of price

calculate grand\_total as sum of price, vat\_amount, and fine

add grand\_total to allTotal

create invoice string with customer details,

kitaa number, location, direction, total anna, price, remarks,

total Rs, VAT (13%) Rs, Grand Total Rs (Included Fine), additional data

append invoice to invoices

return invoices, allTotal

end generate\_returned\_invoices

## **2.4. Data Structure**

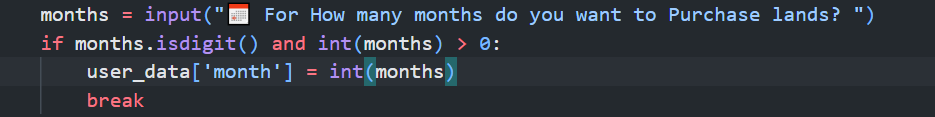
A data structure serves as a crucial asset for every programmer, facilitating the organized storage of data in memory during program execution. It elucidates the connection between data and the logical operations carried out on it, categorized into primitive and non-primitive data types.

In this Course Work, both primitive and non-primitive data types are utilized, encompassing: -

1. Integer

An integer is a fundamental numeric data type that stores whole numbers, both positive and negative, without any decimal points. Examples include 1, -3, and 100.

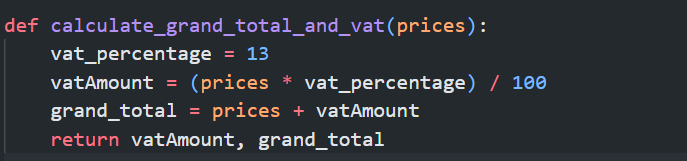
In the context of this coursework, integers are utilized for transforming string number values to integers.



1. Float

A float, short for floating-point number, is a data type used to represent real numbers with decimal points. It can store both small and large numbers, including fractions and numbers with decimal places. Examples of float values include 3.14, -0.5, and 100.0.

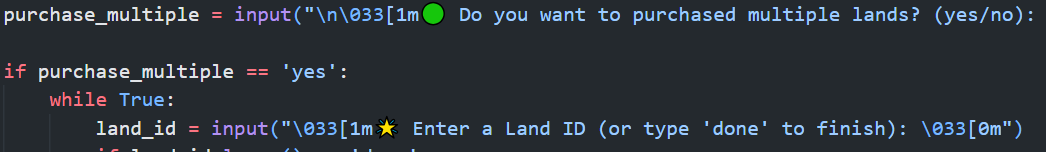
In this course, float data types are utilized for calculations involving decimal values and precise mathematical operations.



1. String

A string is a data type that represents a sequence of characters, such as letters, digits, and symbols. It is commonly used to store text-based data, such as names, sentences, and messages. Strings are enclosed within quotation marks, either single ('') or double (""). Examples of strings include "hello", "123abc", and "Special characters: @#$%".

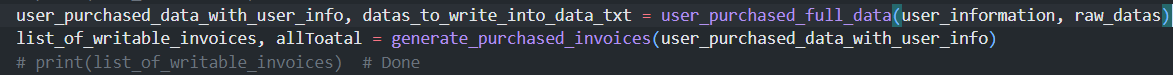
In this course, string data types are employed for handling textual information and manipulating text-based inputs and outputs.



1. Tuples

Tuples are immutable sequences in Python, meaning once created, their elements cannot be modified. They are similar to lists but are enclosed within parentheses instead of square brackets. For example, a tuple might represent a coordinates pair (x, y) or a person's information (name, age, gender).

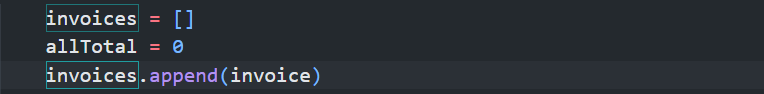
In this course, tuples are utilized for storing and passing around fixed collections of data that should not change during program execution.



1. List

A list is a versatile data structure in Python that stores an ordered collection of items. It can contain elements of different data types, including numbers, strings, and even other lists. For example, [1, 2, 3] is a list containing three integers. Lists are commonly used for tasks like storing data, iterating over elements, and implementing dynamic data structures.

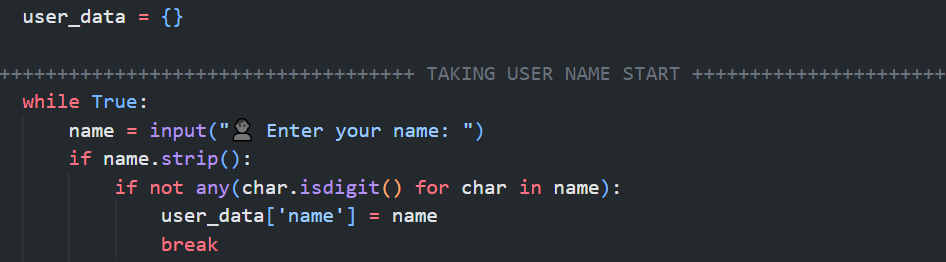
In this course, lists are utilized for managing collections of data that may change in size or content during program execution.



1. Dictionaries

Dictionaries are a powerful data structure in Python used to store key-value pairs. Dictionaries are defined using curly braces {} and elements are specified in the format key: value, separated by commas. For example, {"name": "John", "age": 30}.

In this course, dictionaries are employed for managing and organizing data in a flexible and efficient manner.



# 3. Program

## Module: Main.py

Purpose:

The main.py module serves as the entry point for the land transaction management system. It provides a user-friendly interface for managing various land transactions, including purchasing, returning, and viewing available lands.

**Functions:**

1. main():

This function serves as the main entry point for the module. It displays the menu interface, validates user input, and triggers the corresponding actions based on the user's choice.

2. validation(user\_choice):

Validates user input to ensure it's a number corresponding to a menu option. If the input is invalid, it prompts the user to enter a valid choice.

3. create\_directories():

Creates necessary directories to store invoices if they don't exist already.

4. show\_lands():

Displays a list of available lands with their details, fetched from the data source. Invokes further actions based on user input.

5. purchased\_land():

Handles the process of purchasing land. Prompts the user for confirmation and guides them through the purchase process. Generates invoices for purchased lands and saves data modifications.

6. returned\_land():

Handles the process of returning previously purchased land. Prompts the user for confirmation and guides them through the return process. Generates return invoices and saves data modifications.

## Module: Read.py

Purpose:

The Read.py module provides functions for reading data from a text file and processing it into a structured format. It primarily deals with retrieving land transaction records from a file and converting them into a list of dictionaries.

Functions:

1. data\_into\_list(filename="data.txt"):

Purpose: Reads data from a text file and converts it into a list of dictionaries, where each dictionary represents a record in the data file.

- Arguments:

filename (str): The name of the file to read from (default is "data.txt").

- Returns:

list: A list of dictionaries, each representing a record in the data file.

2. show\_only\_values():

- Purpose: Retrieves only the values from the data\_into\_list returns.

- Returns:

list: A list of lists, each containing the values of a record.

## Module: Write.py

Purpose:

The Write.py module is responsible for saving modified data to a text file and writing generated invoices for land transactions to separate files. It provides functions to save data modifications and generate and save invoices for purchases and returns.

Functions:

1. save\_modified\_data\_to\_file(datas):

Saves modified data to a text file. Accepts a list of dictionaries (datas) representing the data to be saved.

2. write\_generated\_invoice\_to\_file(list\_of\_writable\_invoices, user\_info, \_\_\_allTotal):

Writes generated invoices for purchased lands to separate files. Accepts a list of writable invoices (list\_of\_writable\_invoices), user information (user\_info), and the total amount including VAT (allTotal).

3. write\_generated\_return\_invoice\_to\_file(list\_of\_returnable\_invoices, \_\_\_\_\_\_user\_info, allTotal):

Writes generated invoices for returned lands to separate files. Accepts a list of returnable invoices (list\_of\_returnable\_invoices), user information (user\_info), and the total amount including fines (allTotal).

## Module: Operation.py

Purpose:

Module Operation.py orchestrates land transaction operations in the land transaction management system by integrating functionalities from Read.py and Write.py modules. It facilitates user interactions, data processing, and invoice generation for land purchases and returns.

Functions:

1. get\_refined\_user\_purchased\_info():

Collects and refines user input for land purchase, including user details and land selection.

2. purchasing\_availability\_check(entered\_kitta):

Checks land availability for purchase based on the entered land ID.

3. user\_purchased\_full\_data(user\_info, raw\_datas):

Generates detailed purchase data, updates data availability, and creates invoices for purchased lands.

4. generate\_purchased\_invoices(datas):

Generates invoices for purchased lands.

5. get\_refined\_user\_return\_info():

Collects and refines user input for land return, including user details and land selection.

6. checking\_availability\_of\_return(land\_id):

Checks land availability for return based on the entered land ID.

7. user\_return\_full\_data(user\_info, raw\_datas):

Generates detailed return data, updates data availability, and creates invoices for returned lands.

8. generate\_returned\_invoices(datas):

Generates invoices for returned lands.

Lets analize program

# 4. Testing

Testing plays a vital role in programming because it gives programmers or developers the opportunity to check if their program is functioning correctly. If there are any issues, testing helps the developer pinpoint the problem and resolve it.

## - Test 1

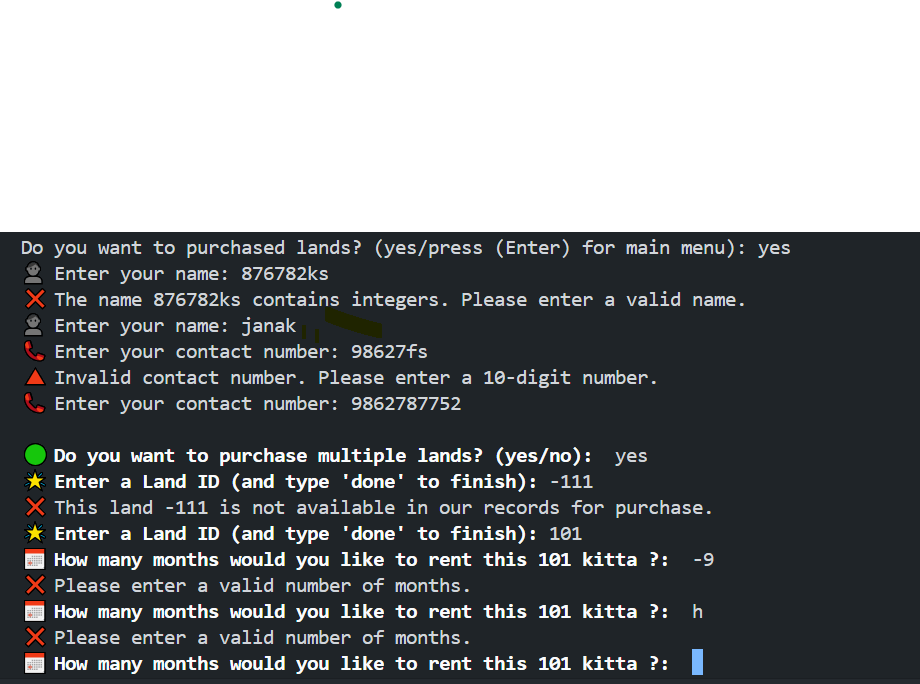
|  |  |
| --- | --- |
| Objectives | Testing if it can handle exceptions |
| Action | Entering invalid input |
| Expected Output | Ability to handle any input from the user, regardless of errors |
| Actual Output | The program successfully managed incorrect user input |
| conclusion | The program operates smoothly until the user decides to close it |



## - Test 2

* Test of Purchased:

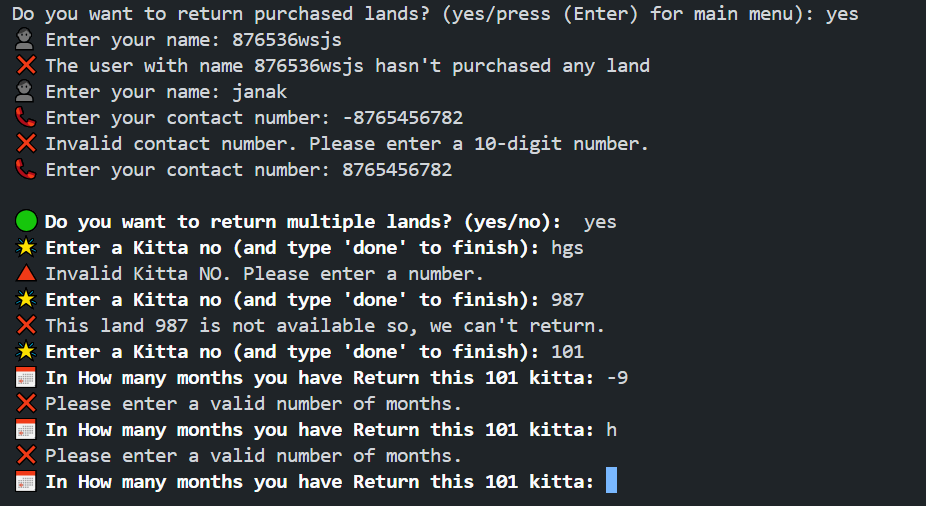
|  |  |
| --- | --- |
| Objectives | Verifying acceptance of negative or non-existing values on Purchased |
| Action | Providing negative and non-existing values on Purchased |
| Expected Output | Program should display error message |
| Actual Output | Display error message |
| conclusion | Test successful |



## - Test 3

* Test of Return:

|  |  |
| --- | --- |
| Objectives | Verifying acceptance of negative or non-existing values on Return |
| Action | Providing negative and non-existing values on Return |
| Expected Output | Program should display error message |
| Actual Output | Display error message |
| conclusion | Test successful |



## - Test 4

* Test table of purchased Bill:

|  |  |
| --- | --- |
| Objectives | To Purchase the available Land and show the bill generated in text file and shell. |
| Action | Run the Purchasing process and enter all the required values |
| Expected Output | The program should run smoothly and generate bill in text file and display in shell |
| Actual Output | The program run smoothly and generate bill in text file and display in shell |
| conclusion | Test successful |

* On Shell:-

A screenshot of a computer screen

Description automatically generated

- On (.txt) File:-

A screenshot of a computer

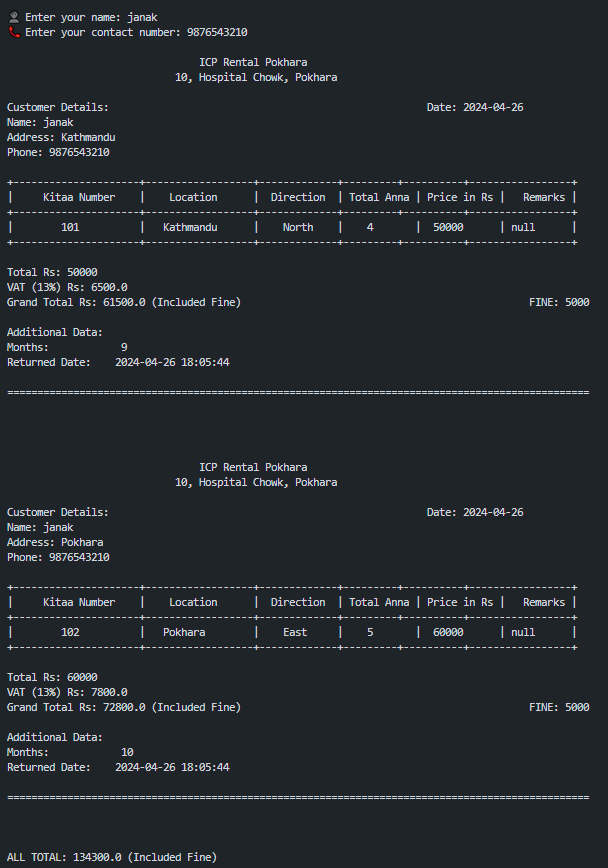
Description automatically generated

## - Test 5

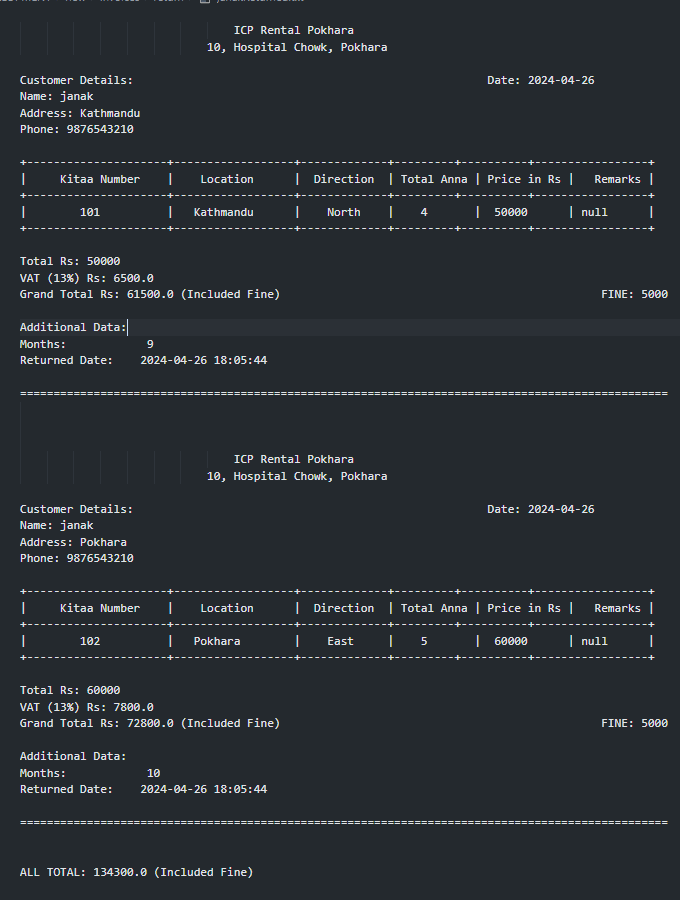
* Test table of Returned Bill:

|  |  |
| --- | --- |
| Objectives | To Return the Purchased Land and show the bill generated in text file and shell. |
| Action | Run the Returning process and enter all the required values |
| Expected Output | The program should run smoothly and generate bill in text file and display in shell |
| Actual Output | The program run smoothly and generate bill in text file and display in shell |
| conclusion | Test successful |

* On Shell:-



* On (.txt) File



# 5. Conclusion

The conclusion of this coursework reflects a journey marked by errors, corrections, and invaluable guidance from the module leader. Throughout the process, I encountered challenges that were overcome with the aid of pointers and support from the module leader, enabling me to navigate through the intricacies of the course tasks. Additionally, extensive study was undertaken, focusing on topics such as algorithms, flowcharts, and pseudocode, among others.

This coursework has not only enhanced my understanding of programming but also provided valuable insights into Python, including its built-in functions, loop structures, conditional statements, and more. The collaborative effort between the module leader and myself, coupled with diligent work, fuelled a sense of enthusiasm and determination to successfully complete the coursework within the stipulated timeframe.

The primary objective of this coursework was to develop an automated billing system to streamline processes and enhance efficiency. By leveraging Python, customers can now benefit from this system, which serves as a practical application to assess students' comprehension of the module. It served as a test of our ability to utilize appropriate data types and functions to ensure the system's proper functionality. This project underscores the broader goal of understanding system operations and harnessing the versatility and functionality of the Python language.

In addition to achieving the primary objectives of the coursework, the journey also fostered personal growth and development. Through hands-on experience and collaboration, I honed essential skills such as problem-solving, critical thinking, and project management. Moreover, the iterative process of identifying errors, making corrections, and seeking guidance instilled a sense of resilience and adaptability. As I reflect on this experience, I am grateful for the opportunity to deepen my understanding of programming concepts and refine my coding abilities. Moving forward, I am eager to apply the lessons learned from this coursework to future projects and endeavours, embracing challenges with confidence and enthusiasm.

# 6. Bibliography

# 7. Appendix

## - Main.py

|  |
| --- |
| import os as s  from operation import \*  from read import \*  from write import \*  def main():  """  Display a menu for managing land transactions, allowing users to:  1. View and manage purchased land.  2. Return previously purchased land.  3. Exit the program.  """  print("""  ,-------------------------------------------------------------------,  | Enchanced Lands Management |  | Where Dreams Come True |  |-------------------------------------------------------------------|  | |  | L A N D S |  | |  | 1. Purchase Land 2. Return Land |  | |  | 3. View Owned Lands 4. Exit |  | |  '-------------------------------------------------------------------'  """)  user\_choice = validation(input("Please choose an option from the menu: \n"))  create\_directories()    if user\_choice == 1:  purchased\_land()  elif user\_choice == 2:  returned\_land()  elif user\_choice == 3:  show\_lands()  elif user\_choice == 4:  print("Exiting the program.")  exit()  else:  print("Invalid choice. Please choose a number between 1 and 3.")  main()  def validation(user\_choice):  """  Validate user input to ensure it's a number.    Args:  - user\_choice (str): The user's input.    Returns:  - int: The user's choice as an integer.  """  while not user\_choice.isdigit():  print("Sorry, that's not a valid choice. Please enter a number.")  user\_choice = input("Please choose an option from the menu: \n")  return int(user\_choice)  def create\_directories():  """  Create folders to store invoices if they don't exist already.  """  directories = ['Invoices/purchased', 'Invoices/return']  for directory in directories:  if not s.path.exists(directory):  s.makedirs(directory)  print(f"Created directory: {directory}")  def show\_lands():  """  Display a list of available lands with their details.  """  # Fetching land data  land\_data = show\_only\_values()  # Formatting header  header = "{:<10} | {:<20} | {:<15} | {:<10} | {:<10} | {:<15}".format(  "KITTA NO", "LOCATION", "DIRECTION", "ANNA", "PRICE", "AVAILABILITY")  separator = "-" \* len(header)  # Displaying header  print(header)  print(separator)  # Displaying land records  for record in land\_data:  print("{:<10} | {:<20} | {:<15} | {:<10} | {:<10} | {:<15}".format(\*record))  # Displaying separator after records  print(separator)  print()  # Invoking further actions  main()  def purchased\_land():  # Prompt user for confirmation  choice = input("Do you want to purchased lands? (yes/press (Enter) for main menu): ").lower()  if choice == "yes":  raw\_datas = data\_into\_list()  user\_information = get\_refined\_user\_purchased\_info() # output: {'name': 'janak', 'contact': '9848010622', 'months': 9, 'land\_ids': ['101', '102']}  # print(user\_information) # Done  user\_purchased\_data\_with\_user\_info, datas\_to\_write\_into\_data\_txt = user\_purchased\_full\_data(user\_information, raw\_datas)  # print(user\_purchased\_data\_with\_user\_info,"/n/n") # Done  # print("#"\*30)  # print(datas\_to\_write\_into\_data\_txt) # Done  list\_of\_writable\_invoices, allToatal = generate\_purchased\_invoices(user\_purchased\_data\_with\_user\_info)  # print(list\_of\_writable\_invoices) # Done  if list\_of\_writable\_invoices:  write\_generated\_invoice\_to\_file(list\_of\_writable\_invoices, user\_information, allToatal)  save\_modified\_data\_to\_file(datas\_to\_write\_into\_data\_txt)  print(f"\nCongratulations, {user\_information['name']}! You've successfully purchased Land IDs:\n\t{', '.join(user\_information['land\_ids'])}\n\tFor {user\_information['month']} Months 🐱‍👤🤩")  else:  print("\nEmpty Data Found Can't purchased ! ")  main()  else:  # Go back to main menu if user presses Enter or enters anything else  print("\nGoing back to main menu.")  main()  def returned\_land():  # Prompt user for confirmation  choice = input("Do you want to return purchased lands? (yes/press (Enter) for main menu): ").lower()  if choice == "yes":  raw\_datas = data\_into\_list()  user\_information = get\_refined\_user\_return\_info()  # print(user\_information) # Done  user\_return\_data\_with\_user\_info, returned\_datas\_to\_write\_into\_data\_txt = user\_return\_full\_data(user\_information, raw\_datas)  # print(user\_return\_data\_with\_user\_info) # Done  # print("\n######\n")  # print(returned\_datas\_to\_write\_into\_data\_txt) # Done  list\_of\_returnable\_invoices, allTotal = generate\_returned\_invoices(user\_return\_data\_with\_user\_info)  if list\_of\_returnable\_invoices:  write\_generated\_return\_invoice\_to\_file(list\_of\_returnable\_invoices, user\_information, allTotal)  save\_modified\_data\_to\_file(returned\_datas\_to\_write\_into\_data\_txt)  print(f"\nCongratulations, {user\_information['name']}! You've successfully return Land IDs:\n\t{', '.join(user\_information['land\_ids'])}\n\tIn {user\_information['months']} Months 🤷‍♀️👏")  else:  print("\nEmpty Data Found Can't Return ! ")  main()  else:  # Go back to main menu  print("\nGoing back to main menu.")  main()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

## - Read.py

|  |
| --- |
| def data\_into\_list(filename="data.txt"):  """  Read data from a text file and convert it into a list of dictionaries.  Args:  - filename (str): The name of the file to read from (default is "data.txt").  Returns:  - list: A list of dictionaries, each representing a record in the data file.  """  with open(filename, 'r') as file:  contents = file.read()  data = contents.split("\n")  result = []  for line in data:  if line == "":  continue  values = line.split(", ")  entry = {  "id": values[0],  "location": values[1],  "direction": values[2],  "anna": values[3],  "price": values[4],  "availability": values[5].strip()  }  result.append(entry)  return result  def show\_only\_values():  """  Retrieve only the values from the records in the data file.  Returns:  - list: A list of lists, each containing the values of a record.  """  data = data\_into\_list()  values\_list = []  for record in data:  only\_values = record.values()  values\_list.append(only\_values)  return values\_list |

## - Write.py

|  |
| --- |
| def save\_modified\_data\_to\_file(datas):  """  Save data to a text file.  Args:  - datas (list): A list of dictionaries representing the data to be saved.  """  with open("data.txt", 'w') as file:  for data in datas:  modified\_data = ", ".join(data.values()) + "\n"  file.write(modified\_data)  def write\_generated\_invoice\_to\_file(list\_of\_writable\_invoices, user\_info, allTotal):  name = user\_info["name"]  file\_path = f"Invoices/purchased/{name}Purchased.txt"  with open(file\_path, 'w') as file:  for invoice in list\_of\_writable\_invoices:  file.write(invoice + '\n\n')  with open (file\_path, 'a') as files:  files.write(f""" ALL TOTAL: {allTotal} (Included Vat)""")  def write\_generated\_return\_invoice\_to\_file(list\_of\_returnable\_invoices, user\_info, allTotal):  name = user\_info["name"]  file\_path = f"Invoices/return/{name}Returned.txt"  with open(file\_path, 'w') as file:  for invoice in list\_of\_returnable\_invoices:  file.write(invoice + '\n\n')  with open (file\_path, 'a') as files:  files.write(f""" ALL TOTAL: {allTotal} (Included Fine)""") |

## - Operation.py

|  |
| --- |
| from read import \*  import os as s  from datetime import datetime  # Purchased Part start  ##################################################################################################################  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 1 START\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  def get\_refined\_user\_purchased\_info():  user\_data = {}  # ++++++++++++++++++++++++++++++++++++ TAKING USER NAME START +++++++++++++++++++++++++++++++++++++++++++++++++++++  while True:  name = input("👤 Enter your name: ")  if name.strip():  if not any(char.isdigit() for char in name):  user\_data['name'] = name  break  else:  print(f"❌ The name {name} contains integers. Please enter a valid name.")  else:  print(f"🔺 Name cannot be empty {name}. Please enter your name.")  # ++++++++++++++++++++++++++++++++++++ TAKING USER NAME END +++++++++++++++++++++++++++++++++++++++++++++++++++++  # ++++++++++++++++++++++++++++++++++++ TAKING USER CONTACT START +++++++++++++++++++++++++++++++++++++++++++++++++++++  while True:  contact = input("📞 Enter your contact number: ")  if contact.isdigit() and len(contact) == 10:  user\_data['contact'] = contact  break  else:  print(f"🔺 Invalid contact number {contact}. Please enter a 10-digit number.")  # ++++++++++++++++++++++++++++++++++++ TAKING USER CONTACT END +++++++++++++++++++++++++++++++++++++++++++++++++++++  # ++++++++++++++++++++++++++++++++++++ TAKING USER PURCHASING MONTHS START +++++++++++++++++++++++++++++++++++++++++++++++++++++  while True:  months = input("📅 For How many months do you want to Purchase lands? ")  if months.isdigit() and int(months) > 0:  user\_data['month'] = int(months)  break  else:  print(f"❌ Please enter a valid number of months. ({months})")  # ++++++++++++++++++++++++++++++++++++ TAKING USER PURCHASING MONTHS END +++++++++++++++++++++++++++++++++++++++++++++++++++++  # ++++++++++++++++++++++++++++++++++++ TAKING LAND KITTA NO START +++++++++++++++++++++++++++++++++++++++++++++++++++++  land\_ids = []    purchase\_multiple = input("\n\033[1m🟢 Do you want to purchased multiple lands? (yes/no): \033[0m").lower()  if purchase\_multiple == 'yes':  while True:  land\_id = input("\033[1m🌟 Enter a Land ID (and type 'done' to finish): \033[0m")  if land\_id.lower() == 'done':  break  else:  # Check if the returned list from checking\_availability() is not empty  availability = checking\_availability(land\_id)  if availability:  if land\_id not in land\_ids:  land\_ids.extend(availability)  else:  print(f"✔️\_\_This land {land\_id} is already selected")  else:  print(f"❌ This land {land\_id} is not Available in our records for purchase 🤳")  # If user wants to purchase only one land  elif purchase\_multiple == 'no':  land\_id = input("\033[1m🌟 Enter a Land ID: \033[0m")  availability = checking\_availability(land\_id)  if availability:  land\_ids.append(land\_id)  else:  print(f"❌ This land {land\_id} is not Available in our records for purchase 🤳")  else:  print(f"🔺 Invalid choice {purchase\_multiple}. Please enter 'yes' or 'no'.")    user\_data['land\_ids'] = land\_ids  # ++++++++++++++++++++++++++++++++++++ TAKING LAND KITTA NO END +++++++++++++++++++++++++++++++++++++++++++++++++++++  return user\_data  def checking\_availability(entered\_kitta):  datas = data\_into\_list()  for data in datas:  if data["id"] == entered\_kitta and data["availability"] == "Available":  return [entered\_kitta]  else:  # If the loop completes without finding a matching land ID, print a message and return an empty list  print(f"This land {entered\_kitta} is not available in our records")  return []  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 1 END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 2 START\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  # No ii  def user\_purchased\_full\_data(user\_info, raw\_datas):  purchased\_list = []  datas\_to\_write = raw\_datas  timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  for data in datas\_to\_write:  if str(data['id']) in user\_info['land\_ids']:  price = int(data['price'])  purchased\_duration = calculate\_purchased\_duration(user\_info)  vatAmount, grandTotal = calculate\_grand\_total\_and\_vat(price)  Remarks = "null"  if data['availability'] == 'Available':  data['availability'] = 'Not Available'  data\_with\_user\_info = {  'id': data['id'],  'location': data['location'],  'direction': data['direction'],  'Anna': data['anna'],  'price': data['price'],  'Remarks': Remarks,  'name': user\_info['name'],  'contact': user\_info['contact'],  'months': user\_info['month'],  'vat\_amount': vatAmount,  'Grand\_Total': grandTotal,  'Timestamp' : timestamp,  'duration' : purchased\_duration  }  purchased\_list.append(data\_with\_user\_info)    return purchased\_list, datas\_to\_write  def calculate\_grand\_total\_and\_vat(prices):  vat\_percentage = 13  vatAmount = (prices \* vat\_percentage) / 100  grand\_total = prices + vatAmount  return vatAmount, grand\_total  def calculate\_purchased\_duration(user\_info):  purchased\_for = user\_info['month']  timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  date\_expried = timestamp.split("-")  this\_month = int(date\_expried[1])  exp\_new\_month = this\_month + purchased\_for  date\_expried[1] = str(exp\_new\_month)  final\_date = "-".join(date\_expried)  return final\_date  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 2 END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 3 START\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  def generate\_purchased\_invoices(datas):  invoices = []  allTotal = 0  for item in datas:  allTotal += int(item["Grand\_Total"])  invoice = f"""  ICP Rental Pokhara  10, Hospital Chowk, Pokhara  Customer Details: Date: {item['Timestamp'].split()[0]}  Name: {item['name']}  Address: {item["location"]}  Phone: {item['contact']}  +---------------------+------------------+-------------+---------+----------+-----------------+  | Kitaa Number | Location | Direction | Total Anna | Price in Rs | Remarks |  +---------------------+------------------+-------------+---------+----------+-----------------+  | {item['id']: <12} | {item['location']: <14} | {item['direction']: <8} | {item['Anna']: <7} | {item['price']: <10} | {item['Remarks']: <10}|  +---------------------+------------------+-------------+---------+----------+-----------------+  Total Rs: {item["price"]}  VAT (13%) Rs: {item['vat\_amount']}  Grand Total Rs: {item['Grand\_Total']}  Additional Data:  Months: {item['months']: <20}  Purchased Duration: {item['duration']: <20}  """  invoices.append(invoice)  return invoices, allTotal  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 3 END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  # Purchased Part end  ##################################################################################################################  # Return Part Start  ###################################################################################################################  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 1 START\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  def get\_refined\_user\_return\_info():  user\_data = {}  while True:  name = input("👤 Enter your name: ")  status = check\_filename(name)  if name.strip():  if status:  user\_data['name'] = name  break  else:  print(f"❌ The user with name {name} Dosen't purchased any land")  else:  print(f"❌ Name cannot be empty {name}. Please enter your name.")  while True:  contact = input("📞 Enter your contact number: ")  if contact.isdigit() and len(contact) == 10:  user\_data['contact'] = contact  break  else:  print(f"❌ Invalid contact number {contact}. Please enter a 10-digit number.")  # Input and validate the number of months  while True:  months = input("📅 How many months have you used the purchased lands? ")  if months.isdigit() and int(months) > 0:  user\_data['months'] = int(months)  break  else:  print("🔺 Please enter a valid number of months.")  user\_data['land\_ids'] = get\_selected\_return\_land\_ids()  return user\_data  def get\_selected\_return\_land\_ids():  land\_ids = []  return\_multiple = input("\n\033[1m🟢 Do you want to return multiple lands? (yes/no): \033[0m").lower()  # If user wants to return multiple lands  if return\_multiple == 'yes':  while True:  land\_id = input("\033[1m🌟 Enter a Kitta NO (and type 'done' to finish): \033[0m")  if land\_id.lower() == 'done':  break  elif not land\_id.isdigit():  print(f"🔺 Invalid Kitta NO {land\_id}. Please enter a number.")  continue  availability = checking\_availability\_of\_return(land\_id)  if availability and land\_id not in land\_ids:  land\_ids.append(land\_id)  elif land\_id in land\_ids:  print(f"✔️\_\_This land with Kitta NO {land\_id} is already selected.")  else:  print(f"❌ This land {land\_id} is not Available so, we can't return")  # If user does not want to return multiple lands  elif return\_multiple == 'no':  land\_id = input("\033[1m🌟 Enter a Kitta NO: \033[0m")  if land\_id.isdigit():  availability = checking\_availability\_of\_return(land\_id)  if availability:  land\_ids.append(land\_id)  else:  print(f"❌ This land {land\_id} is not Available so, we can't return")  else:  print(f"🔺 Invalid Kitta NO {land\_id}. Please enter a number.")  # If user provides an invalid input  else:  print(f"🔺 Invalid choice {return\_multiple}. Please enter 'yes' or 'no'.")  return land\_ids  def checking\_availability\_of\_return(land\_id):  datas = data\_into\_list()  for data in datas:  if data['id'] == land\_id and data['availability'] == 'Not Available':  return True  return False  def check\_filename(user\_name):  file\_path = "Invoices/purchased/"  files = s.listdir(file\_path)  for file\_name in files:  name\_part = file\_name.split('Purchased.txt')[0]  if user\_name == name\_part:  return file\_name    return None  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 1 END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 2 START\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  def user\_return\_full\_data(user\_info, raw\_datas):  returned\_list = []  datas\_to\_write = raw\_datas  timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")  for data in datas\_to\_write:  if str(data['id']) in user\_info['land\_ids']:  month = int(user\_info['months'])  fine = calculate\_fine(month, user\_info)  Remarks = "null"  if data['availability'] == 'Not Available':  data['availability'] = 'Available'  data\_with\_user\_info = {  'id': data['id'],  'location': data['location'],  'direction': data['direction'],  'Anna': data['anna'],  'price': data['price'],  'Remarks': Remarks,  'name': user\_info['name'],  'contact': user\_info['contact'],  'months': user\_info['months'],  'fine': fine,  'Timestamp' : timestamp  }  returned\_list.append(data\_with\_user\_info)    return returned\_list, datas\_to\_write  def grab\_purchased\_month(user\_info):  name = user\_info["name"]  file\_path = f"Invoices/purchased/{name}Purchased.txt"  month = 0  with open(file\_path, 'r') as files:  for line in files:  if "Months:" in line:  exact = line.strip().split(":")[1]  month = int(exact)  return month  def calculate\_fine(month\_return, user\_return\_info):  month\_purchased = grab\_purchased\_month(user\_return\_info)  fine\_per\_month = 1000  fine = 0  if month\_return > month\_purchased:  over\_time = month\_return - month\_purchased  fine = over\_time \* fine\_per\_month  return fine  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 2 END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 3 START\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  def generate\_returned\_invoices(datas):  invoices = []  allTotal = 0  for item in datas:  # Calculate VAT and Grand Total  price = int(item['price'])  fine = int(item["fine"])  vat\_amount = price \* 0.13  grand\_total = price + vat\_amount + fine  allTotal += grand\_total    invoice = f"""  ICP Rental Pokhara  10, Hospital Chowk, Pokhara  Customer Details: Date: {item['Timestamp'].split()[0]}  Name: {item['name']}  Address: {item["location"]}  Phone: {item['contact']}  +---------------------+------------------+-------------+---------+----------+-----------------+  | Kitaa Number | Location | Direction | Total Anna | Price in Rs | Remarks |  +---------------------+------------------+-------------+---------+----------+-----------------+  | {item['id']: <12} | {item['location']: <14} | {item['direction']: <8} | {item['Anna']: <7} | {item['price']: <10} | {item['Remarks']: <10}|  +---------------------+------------------+-------------+---------+----------+-----------------+  Total Rs: {price}  VAT (13%) Rs: {vat\_amount}  Grand Total Rs: {grand\_total} (Included Fine) FINE: {item['fine']}  Additional Data:  Months: {item['months']: <20}  Purchased Date: {item['Timestamp']: <20}  """  invoices.append(invoice)  return invoices, allTotal  # \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_PART 3 END\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |