



Islington college  
(इस्लिङ्टन कलेज)

## **CS4051NI Fundamentals of Computing**

**60% Individual Coursework**

**2023/24 Spring**

**Student Name: Salina Thing**

**London Met ID: : 23047540**

**College ID: : np01cp4a230017**

**Assignment Due Date: Tuesday, May 7, 2024**

**Assignment Submission Date: Tuesday, May 7, 2024**

**Word Count: 242**

*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

<b>1. Introduction</b>	1
<b>2. Algorithm</b>	2
a. Steps	2
b. FlowChart	3
c. Pseudocode	4
<b>Testing</b>	11
a. Test 1(Show implementation of try, except)	11
b. Test 2(Selection rent and return of lands)	11
c. Test 3(File generation of renting of land(s) (Renting multiple land(s))	12
d. Test 4(File generation of returning process of land(s) (Returning multiple land(s))	13
e. Test 5(Show the update in stock of land(s))	14
<b>Appendix</b>	14
a. main.py	14
b. read.py	16
c. write.py	17
d. operation.py	17
e. data.txt	23
<b>Bibliograph</b>	25

## 1. Introduction

The main theme of this project is to be friendly with the real-world project where we learned about the Python Programming language. As we know, Python is a readable, dynamic, pleasant, flexible and powerful languages. It has multi-purpose (General, data science, scientific computing, etc.), supports multiple programming styles (Object-oriented, functional, procedural) and focuses on readability and productivity. Python has large and comprehensive library. Python is an Interpreted Languages and case sensitive in nature. The land rental management project aims to develop a software application that facilitates the efficient management of land rental transactions.

### Goals and Objectives:

1. **Efficiency:** The primary goal of the project is to enhance the efficiency of land rental transactions by automating key processes such as renting, returning, and invoicing. By providing a user-friendly interface so that users can easily access and manage land rental information, saving time and effort.
2. **Accuracy:** The project aims to ensure the accuracy of land rental data. By centralizing data storage and implementing validation checks, the application will minimize errors in rental records, promoting trust and reliability among users.
3. **Convenience:** The project seeks to provide convenience to both landowners and renters by offering an accessible platform for managing land rental activities. Users can easily rent or return land, view rental history, and generate invoices with minimal complexity.
4. **Security:** By implementing robust authentication mechanisms and data encryption techniques, the application will safeguard sensitive information such as user details and financial transactions, ensuring the confidentiality and privacy of users' data.

Overall, the land rental management project aims to revolutionize the way land rental transactions are conducted, offering a comprehensive solution that meets the needs of

both landowners and renters while promoting efficiency, accuracy, transparency, convenience, and security.

## 2. Algorithm

### Algorithm for Land Rental Management Application:

#### a. Steps

##### Step1:Initialization:

- Initialize the land rental data structure.
- Load existing data from a file/database if available.

##### Step2:Main Menu:

- Display the main menu options:
  - Rent Land
  - Return Land
  - View Land Details
  - Exit

##### Step3:Rent Land:

- Prompt user for kitta number.
- Check if the land is available for rent.
- If available:
  - Prompt user for details: name, phone number, rental duration.
  - Update land status to "Not Available".
  - Generate rental invoice.
- Otherwise, display error message and return to main menu.

##### Step3:Return Land:

- Prompt user for kitta number of the land being returned.
- Check if the land is currently rented.
- If rented:
  - Prompt user for return details: rental duration.
  - Calculate any fines or additional charges.
  - Update land status to "Available".

- Generate return invoice.
- Otherwise, display error message and return to main menu.

#### Step4:View Land Details:

- Display all land details including kitta number, owner details, land area, rent price, and rental status.

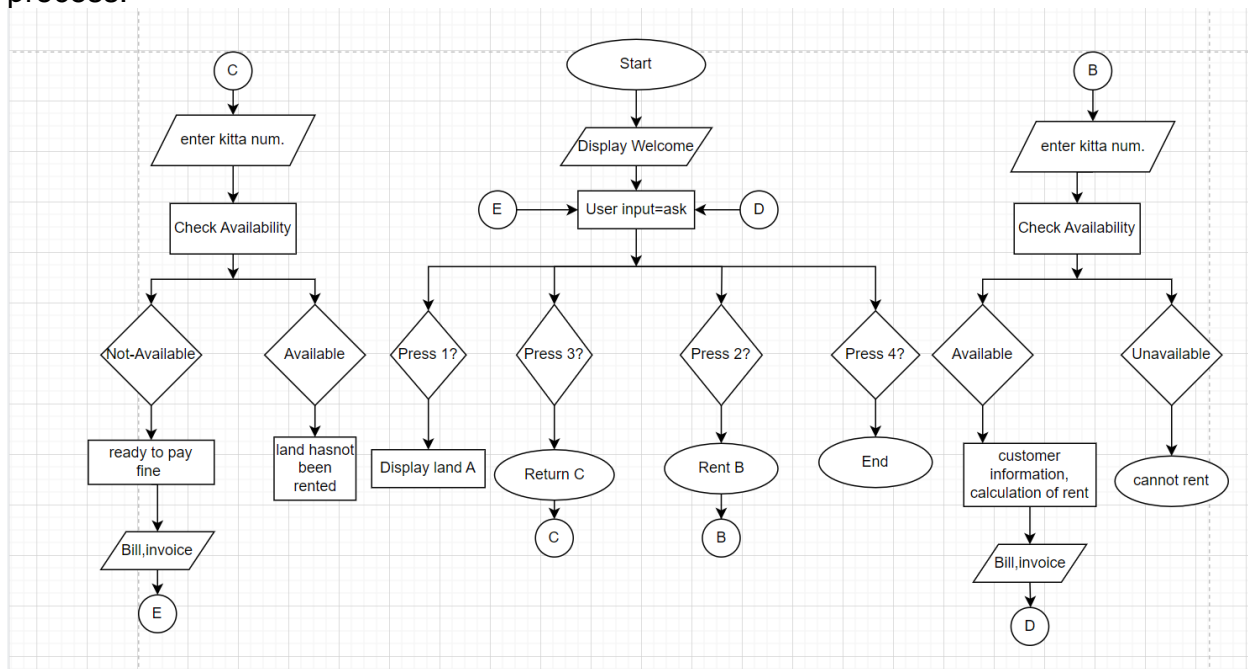
#### Step5:Exit:

- Save updated land data to file/database if necessary.
- Terminate the program.

This algorithm outlines the essential steps involved in managing land rental data, including renting, returning, data management, error handling, user interaction, security, and maintenance. Detailed implementation of each step can be carried out based on specific requirements and technical considerations.

### b. FlowChart

Creating a flowchart for the entire process involves visualizing the steps involved in renting land and generating an invoice. Here's a high-level flowchart outlining the process:



**c. Pseudocode****I. Main.py**

Function welcome\_message():

    Print "Techno Property Nepal"

    Print "Contact no: 98435435556 || Email: technorental@gmail.com"

    Print "Putalisadak, Kathmandu"

    Print "-" repeated 115 times

Function main\_loop():

    land\_data = call read\_land\_data()

    While True:

        Print "Choose an option:"

        Print "-" repeated 150 times

        Print "Press 1 to Display Available Lands"

        Print "Press 2 to Rent Land"

        Print "Press 3 to Return Land"

        Print "Press 4 to Exit"

        Print "-" repeated 150 times

        Get user\_choice input

        If user\_choice is not a digit:

            Print "Invalid input. Please enter a number."

            Continue to the next iteration of the loop

        Convert user\_choice to an integer

        If user\_choice is less than 1 or greater than 4:

            Print "Invalid choice. Please enter a number between 1 and 4."

            Continue to the next iteration of the loop

        If user\_choice is 1:

            Call display\_lands(land\_data)

        Else if user\_choice is 2:

            Call rent\_land(land\_data)

            Get input from user if they wish to continue (Yes/No)

            If user enters "No":

                Break out of the loop

        Else if user\_choice is 3:

            Call return\_land(land\_data)

            Get input from user if they wish to continue (Yes/No)

```
    If user enters "No":
        Break out of the loop
    Else if user_choice is 4:
        Print "Thank you for using the system!"
        Break out of the loop
```

```
Call welcome_message()
Call main_loop()
```

## II. read.py

```
Function read_land_data():
    land_data = an empty dictionary

    try:
        Open the file "data.txt" in read mode as file
        Initialize kitta_num to 101

        For each line in the file:
            Split the line by comma and store the result in line_data
            If the length of line_data is not equal to 6:
                Print "Error: Invalid format for land details in data.txt."
                Return an empty dictionary
            Initialize an empty list for the current kitta_num in land_data

            For each item in line_data:
                If item contains a comma:
                    Print "Error: Comma not allowed in land details."
                    Return an empty dictionary
                Append the item to the list for the current kitta_num in land_data

            Increment kitta_num by 1

    except FileNotFoundError:
        Print "Error: data.txt file not found."

    Return land_data
```

## III. write.py

```
Function update_land_data(land_data):
    Try:
        Open the file "data.txt" in write mode as file

        For each kitta_num, land_details pair in the sorted land_data dictionary:
            Check if land_details has all required fields
            If the length of land_details is not equal to 6:
```

Raise a ValueError with the message "Invalid land details format."

Convert numerical values to strings for writing

For each item in land\_details:

Convert item to a string

Write the comma-separated concatenation of land\_details followed by a newline character to the file

Except IOError as e:

Print an error message indicating that an error occurred while updating land data, including the specific error (e.g., "Error occurred while updating land data: <error\_message>")

Except ValueError as e:

Print an error message indicating that a ValueError occurred, including the specific error (e.g., "Error: <error\_message>")

#### **IV. operation.py**

Function display\_land(land\_data):

Print a formatted header for displaying available lands

For each kitta\_num, land\_details pair in sorted land\_data dictionary:

Print the land details including kitta number, district, direction, aana, price, and availability

Function rent\_land(land\_data):

Try:

Get the kitta number from user input

If the kitta number is not within the valid range:

Print an error message and return

If the land with the given kitta number is not available for rent:

Print a message indicating the land is not available and return

Get user details including name, phone, and rental duration

Update the availability status of the land

Generate a bill for the rental

Except ValueError:

Print "Invalid input. Please try again."

Function return\_land(land\_data):

Try:

Get the kitta number from user input

If the kitta number is not within the valid range:

Print an error message and return

If the land with the given kitta number is not currently rented:

Print a message indicating the land is not currently rented and return

Get user details including name, phone, rental duration, and return duration



- Update the availability status of the land
- Calculate the total price with fine if applicable
- Generate a return invoice

Except ValueError:

- Print "Invalid input. Please try again."

Function generate\_bill(bill\_name, name, phone, user\_land):

- Check if input arguments are valid
- Write customer and billing details to a file
- Write the rental details including kitta number, aana, rental duration, price per month, and total price to the file
- Write a thank you message to the file

Function generate\_return\_invoice(bill\_name, name, phone, kitta\_number, rented\_month, returned\_month, total\_price\_with\_fine, per\_month\_price):

- Check if input arguments are valid
- Write customer and return invoice details to a file
- Write the return invoice details including kitta number, rented month, returned month, total price with fine, and applicable fine to the file
- Write a thank you message to the file

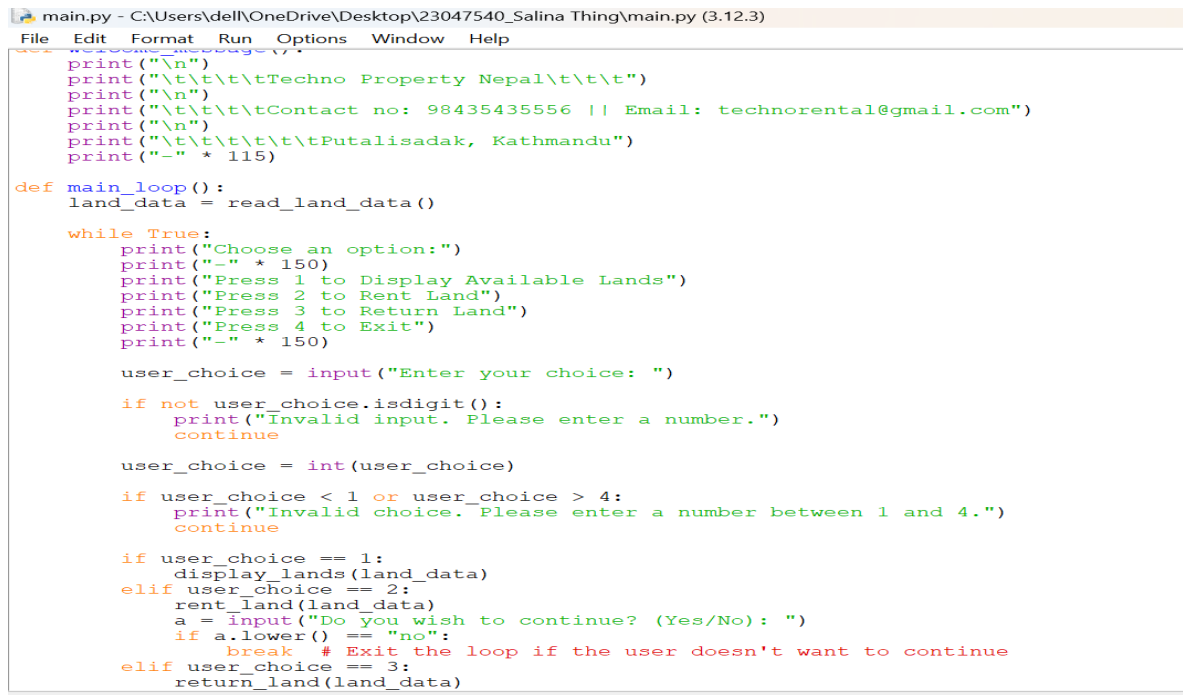
Function calculate\_fine\_or\_total\_price(rented\_month, returned\_month, per\_month\_price, anna\_land):

- Check if input arguments are valid
- Calculate the total price without fine
- Calculate the total price with fine if applicable
- Return the total price

#### d. DataStructure

##### 1.Dictionary (land\_data):

In the **display\_lands()**, **rent\_land()**, and **return\_land()** functions, **land\_data** is used. It is a dictionary where each key represents a kitta number, and the corresponding value is a list containing details about the land associated with that kitta number.



```

main.py - C:\Users\dell\OneDrive\Desktop\23047540_Salina Thing\main.py (3.12.3)
File Edit Format Run Options Window Help
welcome_message = """
print("\n")
print("\t\t\t\t\tTechno Property Nepal\t\t\t\t")
print("\n")
print("\t\t\t\t\tContact no: 98435435556 || Email: technorental@gmail.com")
print("\n")
print("\t\t\t\t\tPutalisadak, Kathmandu")
print("-" * 115)

def main_loop():
    land_data = read_land_data()

    while True:
        print("Choose an option:")
        print("-" * 150)
        print("Press 1 to Display Available Lands")
        print("Press 2 to Rent Land")
        print("Press 3 to Return Land")
        print("Press 4 to Exit")
        print("-" * 150)

        user_choice = input("Enter your choice: ")

        if not user_choice.isdigit():
            print("Invalid input. Please enter a number.")
            continue

        user_choice = int(user_choice)

        if user_choice < 1 or user_choice > 4:
            print("Invalid choice. Please enter a number between 1 and 4.")
            continue

        if user_choice == 1:
            display_lands(land_data)
        elif user_choice == 2:
            rent_land(land_data)
            a = input("Do you wish to continue? (Yes/No): ")
            if a.lower() == "no":
                break # Exit the loop if the user doesn't want to continue
        elif user_choice == 3:
            return_land(land_data)

```

The **land\_data** variable, which is initialized by calling the **read\_land\_data()** function, is a dictionary.

In the **read\_land\_data()** function (assuming it returns a dictionary), you likely read land data from a file or some other data source and populate the **land\_data** dictionary with the information. Each key in the dictionary represents a kitta number, and the corresponding value is a list containing details about the land associated with that kitta number.

For example:

- {
  - 101: ["Kathmandu", "North", 5, 10, 50000, "Available"],
  - 102: ["Pokhara", "South", 7, 12, 60000, "Not Available"], # Other kitta numbers and corresponding details...

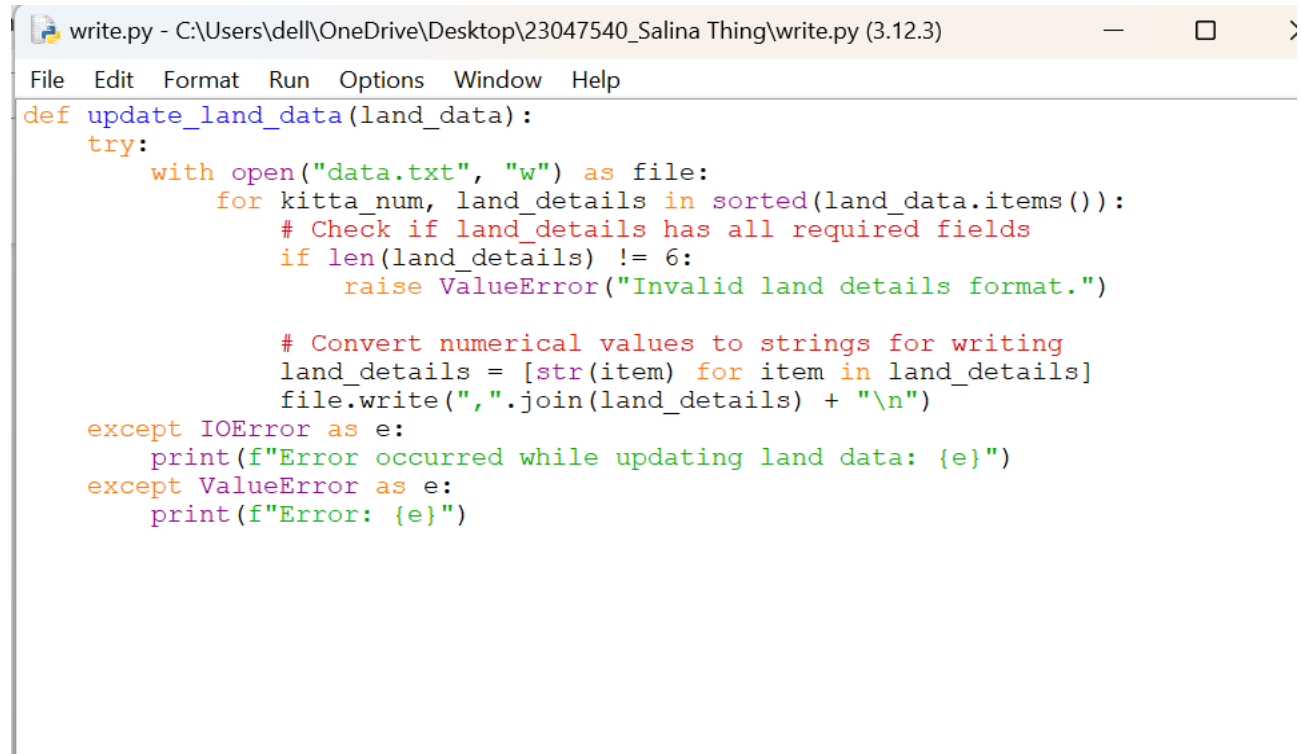
In this dictionary:

- Keys (e.g., 101, 102) represent kitta numbers.
- Values (e.g., ["Kathmandu", "North", 5, 10, 50000, "Available"]) represent land details, such as district, direction, aana, price, and availability.

You then pass this **land\_data** dictionary to functions like **display\_lands()**, **rent\_land()**, and **return\_land()** to perform operations on the land data based on user input.

## 2.File Handling:

The **update\_land\_data()** function uses file handling to write the updated land data back to the "data.txt" file. This function doesn't directly use a specific data structure, but it interacts with the file system to update the data.



```

write.py - C:\Users\dell\OneDrive\Desktop\23047540_Salina Thing\write.py (3.12.3)
File Edit Format Run Options Window Help
def update_land_data(land_data):
    try:
        with open("data.txt", "w") as file:
            for kitta_num, land_details in sorted(land_data.items()):
                # Check if land_details has all required fields
                if len(land_details) != 6:
                    raise ValueError("Invalid land details format.")

                # Convert numerical values to strings for writing
                land_details = [str(item) for item in land_details]
                file.write(",".join(land_details) + "\n")
    except IOError as e:
        print(f"Error occurred while updating land data: {e}")
    except ValueError as e:
        print(f"Error: {e}")

```

In this dictionary:

- Keys (e.g., 101, 102) represent kitta numbers.
- Values (e.g., ["Kathmandu", "North", 5, 10, 50000, "Available"]) represent land details, such as district, direction, aana, price, and availability.

The **update\_land\_data()** function takes this **land\_data** dictionary as input and writes its contents to a file named "data.txt". It iterates over the items of the dictionary, converts each land detail to a string, and writes them to the file in a comma-separated format. Therefore, the primary data structure used in your code is a dictionary.

### Other Data Structures:

a. **Lists:** Lists are ordered collections of items in Python. They can contain elements of different types and support various operations such as indexing, slicing, appending, and removing elements. Example:

```
my_list = [1, 2, 3, "apple", "banana"]
```

b. **Sets:** Sets are unordered collections of unique elements. They are useful for eliminating duplicate values and performing set operations like union, intersection, and difference. Example:

```
my_set = {1, 2, 3, 4, 5}
```

c. **Tuples:** Tuples are immutable sequences of elements. They are similar to lists but cannot be modified after creation. Tuples are commonly used for representing fixed collections of items. Example:

EG:

```
my_tuple = (1, 2, 3, 4, 5)
```

d. **Arrays:** Arrays are collections of elements of the same data type stored in contiguous memory locations. They are more memory-efficient than lists for certain operations. Example:

Eg:

```
import array my_array = array.array('i', [1, 2, 3, 4, 5]) # 'i' represents integer type
```

These are some of the commonly used data structures in Python, each with its own characteristics and use cases. Choosing the appropriate data structure depends on the specific requirements of the problem.

### 3.Program

#### 1. Overall Program:

- The program is a rental management system for lands.
- It allows users to display available lands, rent a land, return a rented land, and exit the system.
- The program is structured using modular programming, with functions for different operations like displaying lands, renting, returning, etc.
- It utilizes file handling to read and update land data stored in a "data.txt" file.

#### 2. Renting and Returning Lands:

- To rent a land, the user selects the kitta number of the available land, enters their name, phone number, and the number of months they want to rent. A bill is generated for the rent.
- To return a rented land, the user selects the kitta number of the rented land, enters their name, phone number, the number of months rented, and the number of months they are returning after. A return invoice is generated.

#### 3. Creation of a Text File:

- When a land is rented or returned, a text file (bill or return invoice) is created to provide a record of the transaction.
- The text file contains details such as the date, customer name, phone number, land details, rental duration, price, etc.

#### 4. Opening Text and Showing the Bill:

- After renting or returning a land, the program displays a message prompting the user to collect their bill or return invoice.
- The user can then open the respective text file (bill or return invoice) to view the details of the transaction.

#### 5. Termination of the Program:

- At the end of the program, the user is given the option to continue using the system or exit.
- If the user chooses to exit, the program terminates.

Overall, the program provides a simple and user-friendly interface for managing land rentals, with features for creating bills, handling rental transactions, and generating return invoices. It ensures data persistence by storing land information in a text file and provides clear feedback to the user throughout the process.

## 4. Testing

### a. Test 1(Show implementation of try, except)

Objective	Show implementation of try, except(IO Error)
Action	→ Try to write to a read-only-file. → Attempt to write to a directory where the user doesn't have permission → Verify that program prints the error message indicating the issue with file writing.
Expected Result	→ Ensure that the try-except blocks are not triggered when there are no errors. → Verify that the program executes the file writing operation without any issues when the land details are correct and the file is writable.
Actual Result	→ The IO runs smoothly
Conclusion	→ The test is successful

### b. Test 2(Selection rent and return of lands)

Objective	Selection rent and return of lands
Action	→ Provide the negative value as input → Provide the non existed value as input
Expected Result	

Actual Result	→Pass only the available option like 1,2,3,4
Conclusion	→Successfully rented and return

Output:

```

IDLE Shell 3.12.3
File Edit Shell Debug Options Window Help
-----
Kitta no. District Direction Aana Price Availability
-----
101 Kathmandu North 4 50000 Available
102 Pokhara East 5 60000 Not Available
103 Lalitpur South 10 100000 Available
104 Dhading West 4 40000 Not Available
105 Bhaktapur East 8 100000 Available
106 Sindhuli South 6 80000 Available
107 Chitwan East 10 1060000 Not Available
108 Banepa North 4 50000 Not Available
-----
Choose an option:
-----
Press 1 to Display Available Lands
Press 2 to Rent Land
Press 3 to Return Land
Press 4 to Exit
-----
Enter your choice: 2
Please select the kitta number to rent: 110
Invalid kitta number. Please try again.
Do you wish to continue? (Yes/No): yes
Choose an option:
-----
Press 1 to Display Available Lands
Press 2 to Rent Land
Press 3 to Return Land
Press 4 to Exit
-----
Enter your choice: 2
Please select the kitta number to rent: -10
Invalid kitta number. Please try again.
Do you wish to continue? (Yes/No): |

```

### c.Test 3(File generation of renting of land(s) (Renting multiple land(s))

Objective	File generation of renting of land(s) (Renting multiple land(s))
Action	→Show complete renting process →Show output in the shell as well →Finally show the rented lands details in a text file
Expected Result	→Show together multiple rent
Actual Result	
Conclusion	

## Output:

```

Python 3.12.3 (tags/v3.12.3:f6650f9, Apr 9 2024, 14:05:25) [MSC v.1938 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> = RESTART: C:\Users\dell\OneDrive\Desktop\23047540_Salina Thing\main.py

Techno Property Nepal

Contact no: 98435435556 || Email: technorental@gmail.com

Putalisadak, Kathmandu

-----
Choose an option:
-----
Press 1 to Display Available Lands
Press 2 to Rent Land
Press 3 to Return Land
Press 4 to Exit
-----
Enter your choice: 2
Please select the kitta number to rent: 106
Enter your name: niwesh
Enter your phone number: 987766345
Kitta number 106 has 6 anna of land.
Enter the number of months you want to rent for: 6

Land rented successfully.
Please collect your bill at 'Bill_niwesh_20240507123046.txt'
Thank you for using TechnoRental services
Choose an option:
-----
Press 1 to Display Available Lands
Press 2 to Rent Land
Press 3 to Return Land
Press 4 to Exit
-----
Enter your choice: |

```

#### d.Test 4(File generation of returning process of land(s) (Returning multiple land(s))

Return Invoice  
 Date: 2024-05-07 12:40:36  
 Customer Name: niwesh  
 Phone: 987766345  
 Returned Land Kitta Number: 106  
 Rented Month: 6  
 Returned Month: 9  
 Total Price (with fine if applicable): 744000.0  
 Applicable Fine: 264000.0

Thank you for using TechnoRental services

Return Invoice  
 Date: 2024-05-07 12:39:40  
 Customer Name: salina  
 Phone: 567  
 Returned Land Kitta Number: 101  
 Rented Month: 4  
 Returned Month: 9  
 Total Price (with fine if applicable): 475000.0  
 Applicable Fine: 275000.0

Thank you for using TechnoRental services|

Objective	File generation of returning process of land(s) (Returning multiple land(s))
Action	→ Show the complete returning process of the land(s) → Show output in the shell as well → Finally show the returned land(s) details in text file
Expected Result	Show multiple return at a time
Actual Result	Show multiple return
Conclusion	Successfully return

### e.Test 5(Show the update in stock of land(s))

## 5.Conclusion

In conclusion, the coursework has provided a comprehensive overview of managing land rental operations using Python programming. Throughout the coursework, various key concepts and techniques have been covered, including file handling, exception handling, and data manipulation.

The implementation involved the use of Python functions to perform operations such as displaying available lands, renting lands, and returning lands. These functions interacted with a data structure, specifically dictionaries, to store and manage land data efficiently. Additionally, the coursework demonstrated the importance of error handling to ensure robustness and reliability in the program.

Overall, the coursework has equipped learners with practical skills in Python programming and data structures, which are essential for developing software applications and solving computational problems.

## 6.Appendix

### a.main.py

```

from read import read_land_data
from operation import display_lands, rent_land, return_land
from datetime import datetime

def welcome_message():
    print("\n")
    print("\t\t\t\t\tTechno Property Nepal")
    print("\n")
    print("\t\t\t\t\tContact no: 98435435556 || Email: technorental@gmail.com")

```



```
print("\n")
print("\t\t\t\t\tPutalisadak, Kathmandu")
print("-" * 115)
```

```
def main_loop():
```

```
    land_data = read_land_data()
```

```
    while True:
```

```
        print("Choose an option:")
```

```
        print("-" * 115)
```

```
        print("Press 1 to Display Available Lands")
```

```
        print("Press 2 to Rent Land")
```

```
        print("Press 3 to Return Land")
```

```
        print("Press 4 to Exit")
```

```
        print("-" * 115)
```

```
    user_choice = input("Enter your choice: ")
```

```
    if not user_choice.isdigit():
```

```
        print("Invalid input. Please enter a number.")
```

```
        continue
```

```
    user_choice = int(user_choice)
```

```
    if user_choice < 1 or user_choice > 4:
```

```
        print("Invalid choice. Please enter a number between 1 and 4.")
```

```
        continue
```

```
    if user_choice == 1:
```

```
        display_lands(land_data)
```

```
    elif user_choice == 2:
```

```
rent_land(land_data)
a = input("Do you wish to continue? (Yes/No): ")
if a.lower() == "no":
    break # Exit the loop if the user doesn't want to continue
elif user_choice == 3:
    return_land(land_data)
    a = input("Do you wish to continue? (Yes/No): ")
    if a.lower() == "no":
        break # Exit the loop if the user doesn't want to continue
elif user_choice == 4:
    print("\nThank you for using the system!")
    break
```

welcome\_message()

main\_loop()

### **b.read.py**

```
def read_land_data():
    land_data = {}
    try:
        with open("data.txt", "r") as file:
            kitta_num = 101
            for line in file:
                line_data = line.split(',')
                if len(line_data) != 6:
                    print("Error: Invalid format for land details in data.txt.")
                    return {}
                land_data[kitta_num] = []
                for item in line_data:
                    if ',' in item:
                        print("Error: Comma not allowed in land details.")
```

```
        return {}
        land_data[kitta_num].append(item)
        kitta_num += 1
except FileNotFoundError:
    print("Error: data.txt file not found.")
return land_data
```

**c.write.py**

```
def update_land_data(land_data):
    try:
        with open("data.txt", "w") as file:
            for kitta_num, land_details in sorted(land_data.items()):
                # Check if land_details has all required fields
                if len(land_details) != 6:
                    raise ValueError("Invalid land details format.")

                # Convert numerical values to strings for writing
                land_details = [str(item) for item in land_details]
                file.write(",".join(land_details) + "\n")
    except IOError as e:
        print(f"Error occurred while updating land data: {e}")
    except ValueError as e:
        print(f"Error: {e}")
```

**d.operation.py**

```
import datetime
from write import update_land_data

def display_lands(land_data):
    print("\nAvailable Lands:")
    print("-" * 98)
    print("Kitta no.\tDistrict\tDirection\tAana\tPrice\tAvailability")
```

```
print("-" * 98)

for kitta_num, land_details in sorted(land_data.items()):

    print(f"{kitta_num}\t{land_details[1]}\t{land_details[2]}\t{land_details[3]}\t{land_details[4]}\t{land_details[5]}")

def rent_land(land_data):
    try:
        kitta_number = int(input("Please select the kitta number to rent: "))
        if kitta_number < 101 or kitta_number >= 101 + len(land_data):
            print("Invalid kitta number. Please try again.")
            return

        land_details = land_data[kitta_number]
        if land_details[5] != "Available":
            print(f"Land with kitta number {kitta_number} is not available for rent.")
            return

        name = input("Enter your name: ")
        phone = input("Enter your phone number: ")
        anna_land = int(land_details[3])
        print(f"Kitta number {kitta_number} has {anna_land} anna of land.")
        month = int(input("Enter the number of months you want to rent for: "))
        per_month_price = int(land_details[4])

        land_data[kitta_number][5] = "Not Available"
        update_land_data(land_data)

        timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")
        bill_name = f"Bill_{name}_{timestamp}.txt"
        bill_kitta_number = kitta_number
```

```
anna = anna_land
rented_month = month
per_month_price_land = per_month_price
user_land = [[bill_kitta_number, anna, rented_month, per_month_price_land]]

print("\nLand rented successfully.")
print(f"Please collect your bill at '{bill_name}'")
print("Thank you for using TechnoRental services")

generate_bill(bill_name, name, phone, user_land)

except ValueError:
    print("Invalid input. Please try again.")

def return_land(land_data):
    try:
        kitta_number = int(input("Enter the kitta number of the land you want to return: "))
        if kitta_number < 101 or kitta_number >= 101 + len(land_data):
            print("Invalid kitta number. Please try again.")
            return

        land_details = land_data[kitta_number]
        if land_details[5] != "Not Available":
            print(f"Land with kitta number {kitta_number} is not currently rented.")
            return

        name = input("Enter your name: ")
        phone = input("Enter your phone number: ")
        rented_month = int(input("Enter the number of months you rented the land for: "))
        returned_month = int(input("Enter the number of months you are returning the land
after: "))
```

```
per_month_price = int(land_details[4])
anna_land = int(land_details[3])

land_data[kitta_number][5] = "Available"
update_land_data(land_data)

total_price_with_fine = calculate_fine_or_total_price(rented_month,
returned_month, per_month_price, anna_land)

timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")
bill_name = f"RBill_{name}_{timestamp}.txt"

print("\nLand returned successfully.")
print(f"Please collect your return invoice at '{bill_name}'")
print("Thank you for using TechnoRental services")

generate_return_invoice(bill_name, name, phone, kitta_number, rented_month,
returned_month, total_price_with_fine, per_month_price)

except ValueError:
    print("Invalid input. Please try again.")

def generate_bill(bill_name, name, phone, user_land):
    if not isinstance(bill_name, str) or not bill_name:
        raise ValueError("Bill name must be a non-empty string.")

    if not isinstance(name, str) or not name:
        raise ValueError("Customer name must be a non-empty string.")

    if not isinstance(phone, str) or not phone:
        raise ValueError("Phone number must be a non-empty string.")
```

```
if not isinstance(user_land, list) or not all(isinstance(item, list) and len(item) == 4 for
item in user_land):
```

```
    raise ValueError("User land must be a list of lists with each inner list containing 4
elements.")
```

```
with open(bill_name, "w") as file:
```

```
    file.write("\n")
```

```
    file.write("\t\t\t\t\t Technoproperty\n")
```

```
    file.write("\tAddress: Kamalpokhari, Kathmandu Metropolitan\n")
```

```
    file.write("Contact no: 98435435556 Email: technorental@gmail.com\n\n")
```

```
    file.write(f"Date: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
```

```
    file.write(f"Name of the customer: {name}\n")
```

```
    file.write(f"Phone of the customer: {phone}\n")
```

```
    file.write("~" * 115)
```

```
    file.write("\n\t\t\t Kitta No.\tAana\tMonth\tPrice per month\tTotal price\n")
```

```
    file.write("~" * 115)
```

```
    file.write("\n")
```

```
for details in user_land:
```

```
file.write(f"\t\t\t{details[0]}\t\t\t{details[1]}\t\t\t{details[2]}\t\t\t{details[3]}\t\t\t{details[3]*details[2]
}\n")
```

```
file.write("\nThank you for using TechnoRental services")
```

```
def generate_return_invoice(bill_name, name, phone, kitta_number, rented_month,
returned_month, total_price_with_fine, per_month_price):
```

```
    if not isinstance(bill_name, str) or bill_name == "":
```

```
        raise ValueError("Bill name must be a non-empty string.")
```

```
    if not isinstance(name, str) or name == "":
```

```
        raise ValueError("Customer name must be a non-empty string.")
    if not isinstance(phone, str) or phone == "":
        raise ValueError("Phone number must be a non-empty string.")
    if not isinstance(kitta_number, int) or kitta_number < 0:
        raise ValueError("Kitta number must be a non-negative integer.")
    if not isinstance(rented_month, int) or rented_month < 0:
        raise ValueError("Rented month must be a non-negative integer.")
    if not isinstance(returned_month, int) or returned_month < 0:
        raise ValueError("Returned month must be a non-negative integer.")
    if not isinstance(total_price_with_fine, int) or total_price_with_fine < 0:
        raise ValueError("Total price with fine must be a non-negative integer.")
    if not isinstance(per_month_price, int) or per_month_price < 0:
        raise ValueError("Per month price must be a non-negative integer.")

    with open(bill_name, "w") as file:
        file.write("Return Invoice\n")
        file.write(f>Date: {datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\n")
        file.write(f>Customer Name: {name}\n")
        file.write(f>Phone: {phone}\n")
        file.write(f>Returned Land Kitta Number: {kitta_number}\n")
        file.write(f>Rented Month: {rented_month}\n")
        file.write(f>Returned Month: {returned_month}\n")
        file.write(f>Total Price (with fine if applicable): {total_price_with_fine}\n")
        if returned_month > rented_month:
            fine = total_price_with_fine - (rented_month * per_month_price)
            file.write(f>Applicable Fine: {fine}\n")
        else:
            file.write("Applicable Fine: 0\n")
        file.write("\nThank you for using TechnoRental services")
```



```
def calculate_fine_or_total_price(rented_month, returned_month, per_month_price,
anna_land):
    # Check if all input arguments are integers and non-negative
    if not isinstance(rented_month, int) or rented_month < 0:
        raise ValueError("Rented month must be a non-negative integer.")
    if not isinstance(returned_month, int) or returned_month < 0:
        raise ValueError("Returned month must be a non-negative integer.")
    if not isinstance(per_month_price, int) or per_month_price < 0:
        raise ValueError("Per month price must be a non-negative integer.")
    if not isinstance(anna_land, int) or anna_land < 0:
        raise ValueError("Anna land must be a non-negative integer.")

    # Calculate total price without fine
    total_price = per_month_price * rented_month

    # Calculate total price with fine if applicable
    if returned_month > rented_month:
        delayed_month = returned_month - rented_month
        fine_price = int(0.1 * delayed_month * per_month_price)
        total_price += fine_price

    return total_price
```

**e.data.txt**

```
101 ,Kathmandu ,North ,4 ,50000 ,Available
102 ,Pokhara ,East ,5 ,60000 ,Not Available
103 ,Lalitpur ,South ,10 ,100000 ,Available
104 ,Dhading ,West ,4 ,40000 ,Not Available
105 ,Bhaktapur ,East ,8 ,100000 ,Available
106 ,Sindhuli ,South ,6 ,80000 ,Available
107 ,Chitwan ,East ,10 ,1060000 ,Not Available
```

108 ,Banepa ,North ,4 ,50000 ,Not Available

## **Bibliograph**