



CS4051NI Fundamentals of Computing

60% Individual Coursework

2023/24 Spring

Student Name: Evani Raut

London Met ID: 23047473

College ID: NP01NT4A230151

Assignment Due Date: Monday, July 29, 2024

Assignment Submission Date: Monday, July 29,

2024 Word Count: 5001

Project File Links:

YouTube Link:	Keep Unlisted YouTube URL of your Project Here
Google Drive Link:	Keep Google Drive URL of your Project Here with Anyone in Organization can View Option Enabled

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.

Contents

1. Introduction:	1
1.1 Brief introduction about the project:	1
Goals:	2
Objectives:	2
1.2 Tools used:	3
Visual Studio Code:	3
Microsoft World:	4
Draw.io:	4
2. Algorithm:	5
2.1 main.py:	5
2.2 read.py:	6
2.3 write.py	6
2.4 operation.py:	7
3. Flowchat:	7
4.Pseudocode:	12
5.Data structure:	19
5.1 Integer datatype:	19
5.2 String datatype:	20
5.3 List datatype:	21
6.Program:	22
6.1 Implementation of the Program:	22
6.2 Rent and Return of the Land	23
6.3 Creation of a TXT File.....	27
6.4 Opening Text and Showing the Bill.....	28
6.5 Termination of the Program	29
7. Testing:	29
7.1 Test 1	29
7.2 Test 2:	30
7.3 Test 3:	32
7.4 Test 4:	34
7.5 Test 5:	35
8. Conclusion:	38
9. Bibliography:	40
10. Appendix:	40
10.1 Main.py	40
10.2 write.py.....	45

10.3 read.py:	46
10.4 operation.py:	48

List of Figures

Figure 1:Image of VS code	3
Figure 2:Image of MS-Word.....	4
Figure 3:Image of Draw.io	4
Figure 4: Flowchart for main.py	9
Figure 5: Flowchart for operation.py	9
Figure 6:Flowchart for read.py	10
Figure 7:Flowchart for write.py.....	11
Figure 8:Image for int datatype	20
Figure 9:Image for str datatype	21
Figure 10:Image for list datatype	22
Figure 11:Image for rent process 1	24
Figure 12:Image to display lands	24
Figure 13:Image to rent land.....	25
Figure 14:Rent invoice.....	25
Figure 15:Return land.....	26
Figure 16:Return invoice.....	27
Figure 17:Bill for renting.....	28
Figure 18:Bill for returning	28
Figure 19:Exiting program	29
Figure 20:Image for test 1	30
Figure 21:First image for test 2	31
Figure 22:Second image for test 2	32
Figure 23:Image for test 3.....	33
Figure 24:Image for test 4.....	34
Figure 25:First image for test 5	36
Figure 26:Second image for test 5	36
Figure 27:Third image for test 5	37

List of Tables

Table 1:Testing 1 29

Table 2:Testing 2 30

Table 3:Testing 3 32

Table 4:Testing 4 34

Table 5:Testing 5 35

1. Introduction:

1.1 Brief introduction about the project:

TechnoPropertyNepal is a private company that manages a stock of different lands on a contract basis across various locations in Nepal. The company allows clients to rent these available lands in different areas. The primary goal of this project is to develop a Land Renting System for TechnoPropertyNepal that can efficiently manage the availability of lands, handle rental transactions, and generate invoices/notes for each transaction.

The system will read land details from a text file and display available and unavailable lands for rent. It can update land availability in the text file when land is rented or returned. Additionally, the system generates invoices for each rental transaction, including details like kitta number, location, area, customer name, rental duration, and total cost. Upon land return, the system creates invoices noting the return date, rental duration, and any fines for late returns.

The development of the Land Renting System will involve designing a modular and efficient program structure using appropriate data structures and algorithms in Python. Thorough testing will be conducted to validate the program's functionality, and a comprehensive report will be prepared documenting the project's development process, testing, and findings.

By implementing this Land Renting System, TechnoPropertyNepal will streamline its operations, enabling efficient land management, accurate transaction tracking, and improved customer service through timely invoice generation.

Goals:

- Improve Efficiency in Land Management:
 - Streamline management of land inventory, rentals, and returns.
 - Automate tasks such as updating availability and generating invoices.
- Enhance Customer Satisfaction:
 - Provide a smooth rental experience for customers.
 - Ensure timely and accurate invoicing.
- Maximize Revenue and Profitability:
 - Optimize land rental to maximize occupancy and revenue.
 - Implement penalty fees for late returns.
- Maintain Accurate and Up-to-Date Records:
 - Keep accurate land inventory and transaction records.
 - Facilitate auditing and reporting with organized records.

Objectives:

- Read and Display Land Inventory:
 - Load the text file with land inventory details.
 - Display available and unavailable lands to the administrator.
- Handle Land Rentals:
 - Update availability status in the inventory for rentals.
 - Generate invoices for each rental transaction, including land and customer details.
- Manage Land Returns:
 - Update availability status and store return details.
 - Generate invoices for each return, including land and customer details.
- Transactions and Maintain Records:

- Keep organized records of each rental and return transaction.
- Ensure easy access for reporting and auditing.
- Calculate and Apply Penalties:
 - Calculate and apply penalties for late returns.
 - Update invoices to include penalties when necessary.

1.2 Tools used:

Visual Studio Code:

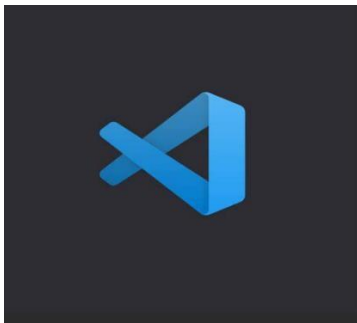


Figure 1:Image of VS code

In this project, I utilized Visual Studio Code (VS Code) to streamline development and boost productivity. Features such as syntax highlighting and auto-completion improved code readability and writing efficiency. The integrated terminal allowed me to execute scripts and test code directly within the editor for efficient debugging. Extensions like the Python extension provided linting, formatting, and virtual environment management, ensuring clean, consistent code. Additionally, VS Code's version control integration facilitated easy tracking of changes and collaboration. Overall, VS Code offered a comprehensive and user-friendly platform for effective development and maintenance of this project.

Microsoft Word:*Figure 2:Image of MS-Word*

For documentation task, I choose Microsoft word as a reliable tool. MS word provides a robust platform for creating and formatting project documentation. Its rich set of features supports the creation of detailed documents, allowing me to articulate project requirements, design specifications, and user manuals with ease.

Draw.io:*Figure 3:Image of Draw.io*

To visually represent the architecture of flowchart in my project, I turned to draw.io. As a web-based diagramming tool, it offers an intuitive interface for creating various diagrams, including flowchart diagrams. It's flexibility and ease make it valuable asset in conveying the design aspects of my project.

2. Algorithm:

2.1 main.py:

Step 1: Start

Step 2: Import required functions from read, write, and operation modules.

Step 3: Define the function show_menu(). This function prints the menu header and menu options including options to display available lands, rent a land, return a land, and exit.

Step 4: Define the function main(). This function handles the main flow of the program.

Step 5: In main(), specify the data file name as 'land_data.txt'.

Step 6: In main(), check if the data file exists. If the file is not found, print an error message and return.

Step 7: In main(), attempt to read land records from the file using read_land_data(data_file). If an error occurs during reading, print an error message and return.

Step 8: In main(), enter an infinite loop to display the menu using show_menu() and prompt the user for a selection.

Step 9: In main(), if the user chooses '1', filter and display available lands from the land_record_list.

Step 10: In main(), if the user chooses '2', enter another loop to handle multiple rentals. Prompt the user for rental details and process the rental using land_rental(land_record_list, kitta_number, renter_name, rental_duration). Write updated land data to the file using write_land_data(data_file, land_record_list). If an error occurs, print an error message. Prompt the user if they want to rent another land. Exit the loop if the user chooses 'no'.

Step 11: In main(), if the user chooses '3', prompt the user for return details and process the return using land_return(land_record_list, kitta_number, renter_name, rental_duration,

return_date). Write updated land data to the file using write_land_data(data_file, land_record_list). If an error occurs, print an error message.

Step 12: In main(), if the user chooses '4', print a message indicating the program is exiting and break the loop to exit the program.

Step 13: In main(), if the user chooses an invalid option, print an error message indicating an invalid choice.

Step 14: End

2.2 read.py:

Step 1: Start

Step 2: Define read_land_data(file_name). This function reads land data from the specified file.

Step 3: In read_land_data, initialize an empty list to store land records.

Step 4: In read_land_data, open the file specified by file_name for reading. Iterate over each line in the file. Split the line into parts based on comma and space. Check if the line has the correct number of parts.

Step 5: In read_land_data, extract and convert values from parts. Create a dictionary for the land record and append it to the list of land records.

Step 6: In read_land_data, handle any file not found or other exceptions by printing an error message.

Step 7: In read_land_data, return the list of land records.

Step 8: End

2.3 write.py

Step 1: Start

Step 2: Define write_land_data(file_name, land_record_list). This function writes land data to the specified file.

Step 3: In write_land_data, open the file specified by file_name for writing. Iterate over each land record in land_record_list. Format and write each land record to the file.

Step 4: End

2.4 operation.py:

Step 1: Start

Step 2: Define `create_invoice_report(land_info, customer_name, rental_duration, transaction_type, current_time, fine=0)`. This function creates an invoice report for the transaction.

Step 3: In `create_invoice_report`, create the invoice filename based on transaction type and kitta number. Define the border and header for the invoice. Construct the invoice info with land and rental details.

Step 4: In `create_invoice_report`, calculate total amount and total payment due. If the transaction type is 'return', apply a 10% fine. Print and save the invoice info to a file.

Step 5: In `create_invoice_report`, handle any errors during file writing by printing an error message.

Step 6: In `create_invoice_report`, return the invoice filename.

Step 7: Define `land_rental(land_records, kitta_number, customer_name, rental_duration)`. This function handles the process of renting a land.

Step 8: In `land_rental`, get the current date and time. Convert land records to a dictionary.

Step 9: In `land_rental`, check if the land is available. If available, update the land's status, create an invoice report, and print a success message. Return True if the rental is successful.

Step 10: In `land_rental`, if the land is not available, print an error message and return False.

Step 11: Define `land_return(land_records, kitta_number, customer_name, rental_duration, return_date)`. This function handles the process of returning a land.

Step 12: In `land_return`, get the current date and time. Convert land records to a dictionary.

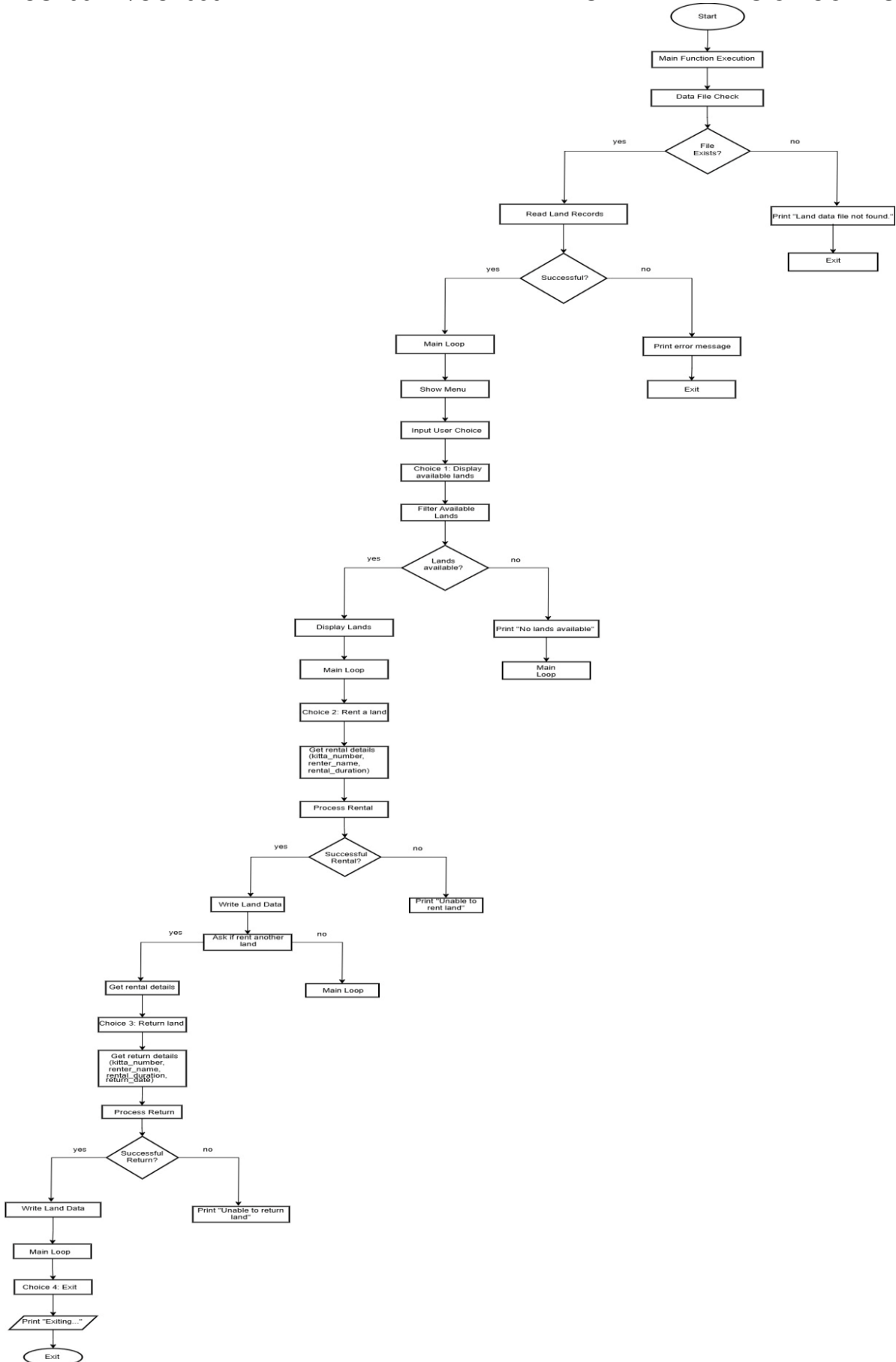
Step 13: In `land_return`, check if the land is currently rented. If rented, calculate the end date of the rental period and any applicable fine. Update the land's status, create an invoice report, and print a success message. Return True if the return is successful.

Step 14: In `land_return`, if the land is not currently rented, print an error message and return False.

Step 15: End

3. Flowchat:

- main.py



- operation.py:

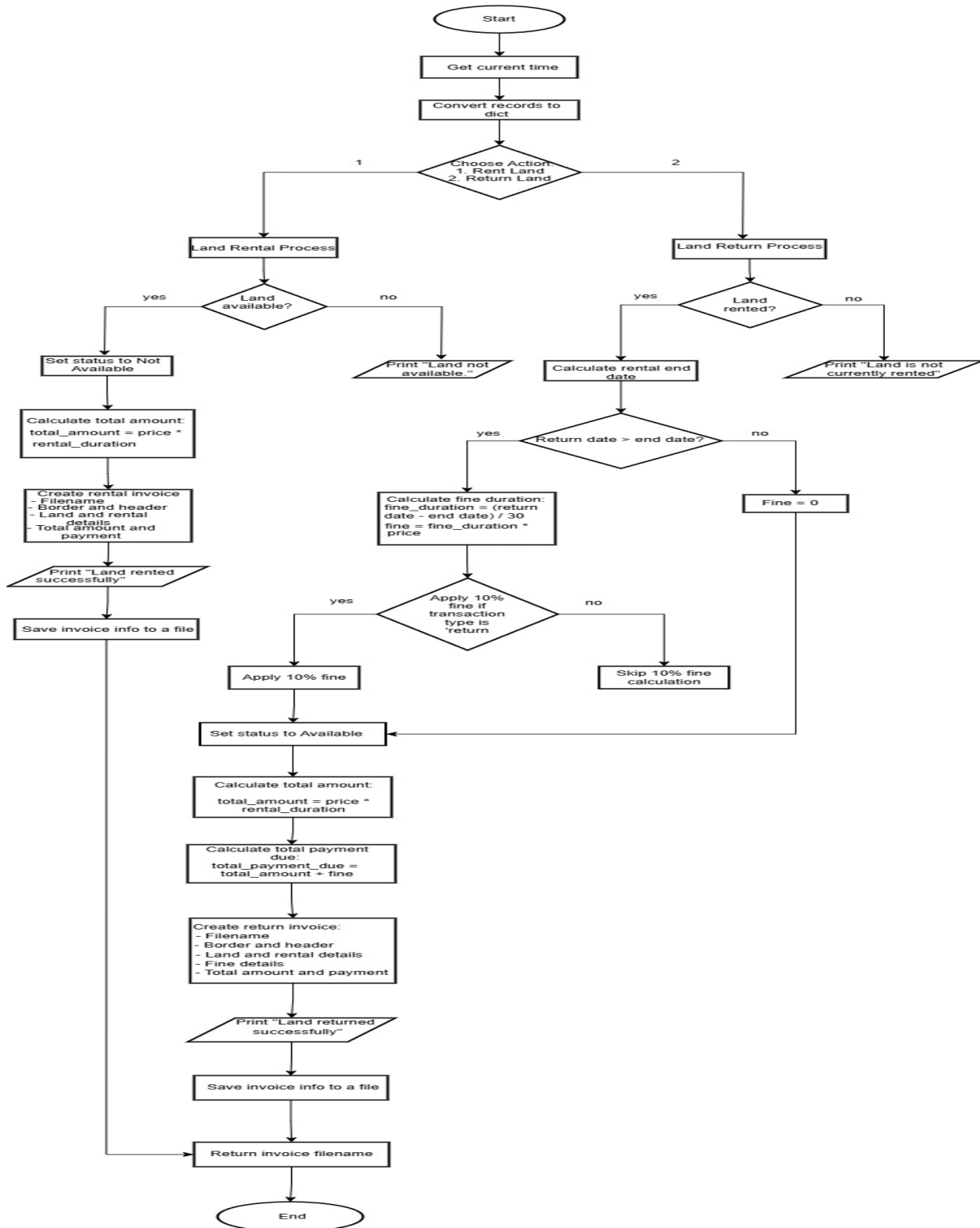


Figure 5: Flowchart for operation.py

- read.py:

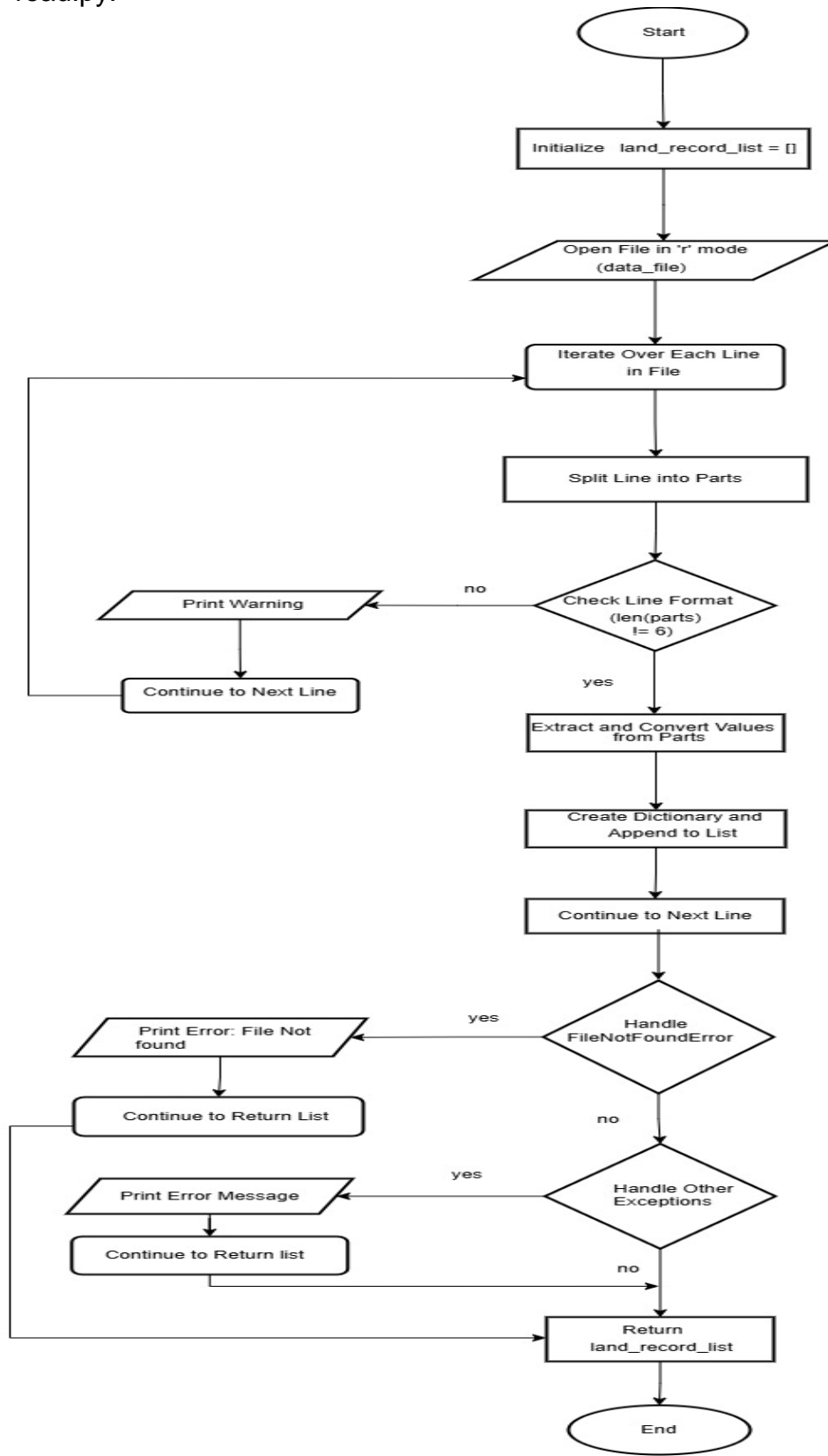


Figure 6:Flowchart for read.py

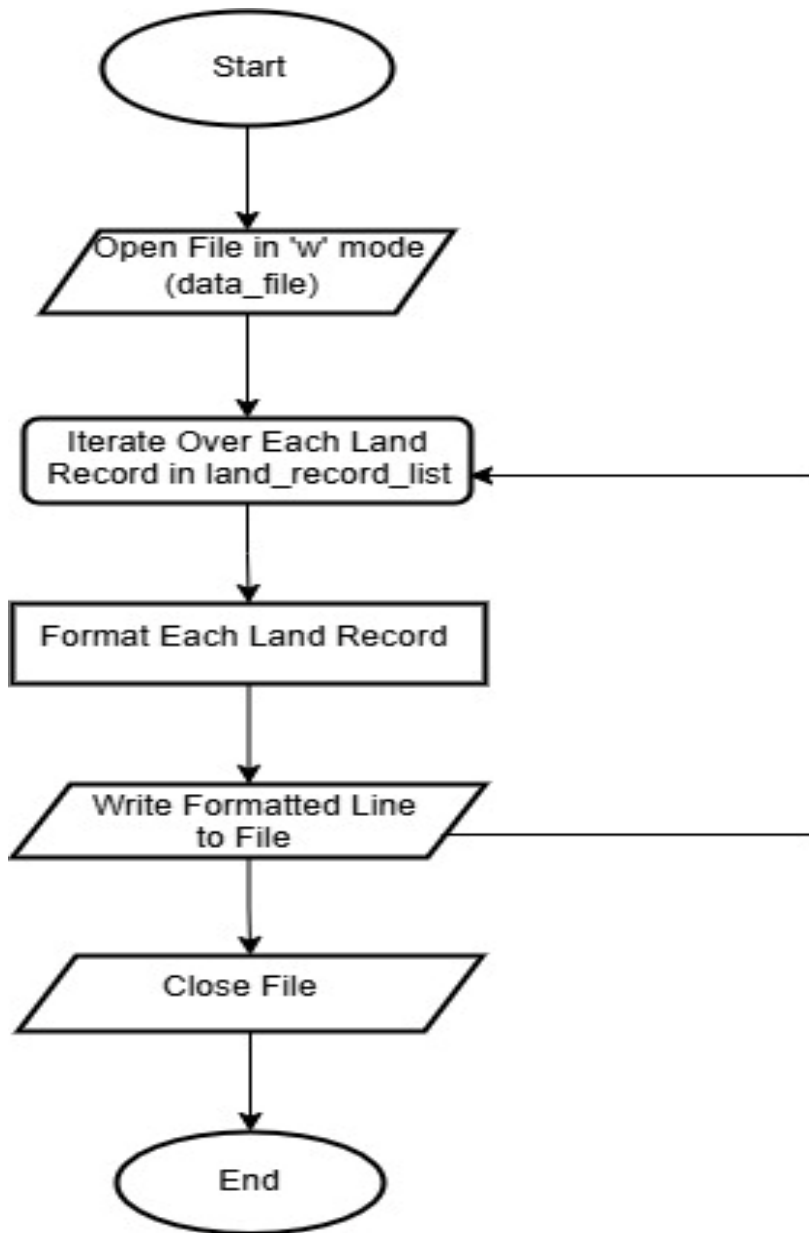


Figure 7:Flowchart for write.py

4.Pseudocode:

DEFINE FUNCTION SHOW_MENU:

PRINT a header with borders and title "Land Rental Management System"

DEFINE menu options:

- "1. Display available lands"
- "2. Rent a land"
- "3. Return a land"
- "4. Exit"

PRINT menu options

END FUNCTION

DEFINE FUNCTION CREATE_INVOICE_REPORT(land_info, customer_name, rental_duration, transaction_type, current_time, fine=0):

DEFINE invoice_filename based on transaction_type and kitta_number

DEFINE border and header for invoice

CONSTRUCT invoice_info:

- Include land details
- Calculate total_amount
- IF transaction_type is 'return':

Calculate and apply 10% fine

- Add total_amount_due to invoice_info

PRINT invoice_info

TRY:

OPEN invoice_filename for writing

WRITE invoice_info to file

EXCEPT IOError as e:

PRINT "Error writing invoice to file: " + str(e)

RETURN invoice_filename

END FUNCTION

DEFINE FUNCTION LAND_RENTAL(land_records, kitta_number, customer_name, rental_duration):

GET current date and time as current_time

CONVERT land_records to dictionary with kitta_number as key

IF land is available:

UPDATE land status to 'Not Available'

CALL CREATE_INVOICE_REPORT with parameters

PRINT "Land rented successfully. Invoice saved as " + invoice_filename

RETURN True

ELSE:

PRINT "Land not available for rent."

RETURN False

DEFINE FUNCTION LAND_RETURN(land_records, kitta_number, customer_name,
rental_duration, return_date):

 GET current date and time as current_time

 CONVERT land_records to dictionary with kitta_number as key

 IF land is currently rented:

 CALCULATE rent_end_date and fine

 UPDATE land status to 'Available'

 CALL CREATE_INVOICE_REPORT with parameters

 PRINT "Land returned successfully. Invoice saved as " + invoice_filename

 RETURN True

 ELSE:

 PRINT "Land is not currently rented."

 RETURN False

END FUNCTION

DEFINE FUNCTION READ_LAND_DATA(data_file):

 INITIALIZE land_record_list as empty

 TRY:

 OPEN data_file for reading

 FOR each line in file:

IF parts have correct length:

EXTRACT values and convert

CREATE land_info dictionary

APPEND land_info to land_record_list

ELSE:

PRINT warning for incorrect format

EXCEPT FileNotFoundError:

PRINT "Error: File " + data_file + " not found."

EXCEPT Exception as e:

PRINT "Error: " + str(e)

RETURN land_record_list

END FUNCTION

DEFINE FUNCTION WRITE_LAND_DATA(data_file, land_record_list):

OPEN data_file for writing

FOR each land_info in land_record_list:

FORMAT land_info into a line

WRITE line to file

END FUNCTION

DEFINE data_file as 'land_data.txt'

TRY:

OPEN data_file for reading

EXCEPT FileNotFoundError:

PRINT "Land data file not found."

RETURN

TRY:

READ land records from data_file using READ_LAND_DATA

EXCEPT Exception as e:

PRINT "Error reading land data: " + str(e)

RETURN

WHILE True:

CALL SHOW_MENU

PROMPT user for choice

IF choice is '1':

FILTER land records for status 'Available'

IF no lands available:

PRINT "No lands available for rent."

ELSE:

PRINT details of available lands

ELSE IF choice is '2':

WHILE True:

PROMPT user for kitta_number, renter_name, and rental_duration

TRY:

CALL LAND_RENTAL with parameters

IF successful:

CALL WRITE_LAND_DATA with updated data

ELSE:

PRINT "Unable to rent land. Please check availability."

EXCEPT ValueError:

PRINT "Invalid input for rental duration. Please enter a number."

PROMPT if user wants to rent another land

IF response is 'no':

BREAK

ELSE IF choice is '3':

PROMPT user for kitta_number, renter_name, rental_duration, and return_date

TRY:

CALL LAND_RETURN with parameters

IF successful:

CALL WRITE_LAND_DATA with updated data

ELSE:

PRINT "Unable to return land. Please check details."

EXCEPT ValueError as e:

PRINT "Invalid input: " + str(e)

ELSE IF choice is '4':

PRINT "Exiting the program."

BREAK

ELSE:

PRINT "Invalid choice. Please try again."

CALL WRITE_LAND_DATA with updated data

END FUNCTION

CALL MAIN

5.Data structure:

5.1 Integer datatype:

In my code, the int datatype is used to represent integer values, which is crucial for various calculations and data handling. For example, the rental duration entered by the user is converted to an integer to ensure accurate processing in functions like `land_rental` and `land_return`. This conversion allows the program to handle rental periods as precise whole numbers. Additionally, the int datatype is used to represent the area of the land (in aana) and the price of the land (in NPR). These values, read from the file, are converted to integers to facilitate straightforward arithmetic operations, such as calculating the total amount due and any applicable fines. Using the int datatype ensures precision and efficiency in arithmetic operations and aids in input validation by catching and handling errors when non-integer values are entered. For example, converting the rental duration to an integer helps validate user input, prompting the user to enter a valid number if they initially provide a non-integer value. (w3schools, 2024)


```
# Extract and convert values from parts
kitta_number = parts[0]
location = parts[1]
direction = parts[2]
area_anna = int(parts[3]) # Convert area_anna to integer
price = int(parts[4]) # Convert price to integer
status = parts[5]
```

Integer datatype
used

Figure 8: Image for int datatype

5.2 String datatype:

In context of my code, the str datatype is crucial for handling various textual data and ensuring smooth interaction with the user and file operations. The str datatype is used for kitta_number, location, direction, and status fields in land records, enabling clear identification and categorization of each land entry. For example, kitta_number uniquely identifies each land, while location and direction provide specific details about the land's position. The status field, also a string, indicates whether the land is available or not. Strings are also used in user prompts and to generate formatted text for invoices and menu options, making the data more readable and easier to manage. Overall, the use of str datatype facilitates efficient data handling, clear communication, and user interaction within the land rental system. (realpython, 2024).

```
if fine > 0:  
    invoice_info += "Initial Payment: " + str(fine) + " NPR\n"  
    invoice_info += "10% Fine: " + str(ten_percent_fine) + " NPR\n"
```

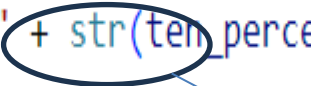


Figure 9: Image for str datatype

String datatype
used

5.3 List datatype:

In my code, the list datatype is utilized to manage and organize collections of data efficiently. Specifically, `land_record_list` serves as a container for multiple land records, where each record is represented as a dictionary. This list facilitates operations such as filtering available lands, iterating through records for display, and updating land statuses during rental or return transactions. The use of lists allows for dynamic handling of land records, enabling easy access, modification, and management of multiple entries. For example, in the main function, `land_record_list` is populated by reading data from the `land_data.txt` file using the `read_land_data` function. This list is then filtered to display available lands, and it is updated whenever a land is rented or returned. Additionally, lists are used to store menu options, streamlining the process of displaying choices to the user. Overall, the list datatype plays a crucial role in maintaining and manipulating data throughout the application. (geeksforgeeks, 2024).

```
.py > read_land_data
def read_land_data(data_file):
    land_record_list = [] # Initialize an
    try:
        with open(data_file, 'r') as file:
            for line in file: # Iterate o
```

Figure 10:Image for list datatype

List datatype used

6.Program:

6.1 Implementation of the Program:

The Land Rental Management System is a detailed application designed to efficiently handle the management of land rentals and returns. The system consists of several key modules:

1. **main.py:** This module organizes user interactions and controls the flow of the program. It presents a menu to the user, processes choices, and manages the overall execution of the rental and return processes.
2. **read.py:** This module is responsible for reading land data from text files. It parses the data and converts it into a format that can be used by other parts of the system.
3. **write.py:** This module handles writing data to text files. It updates land records and logs rental and return transactions.
4. **operation.py:** This module includes core functions for managing land rentals and returns. It performs tasks such as checking land availability, calculating rental amounts, and generating invoices.

6.2 Rent and Return of the Land

Rent Process:

1. **Menu Selection:** Users initiate the rental process by choosing the rental option from the main menu.

2. **User Input:** They provide the following details:

Kitta Number: The unique identifier for the land.

Renter's Name: The name of the person renting the land.

Rental Duration: The rental period in months.

3. **Land Availability Check:** The system verifies whether the land is currently available for rent.

4. **Update Land Status:** If available, the system updates the land status to "Not Available".

5. **Calculate Rental Amount:** The total rental amount is computed based on the land's price and the rental duration.

6. **Generate Invoice:** An invoice is created, including details such as:

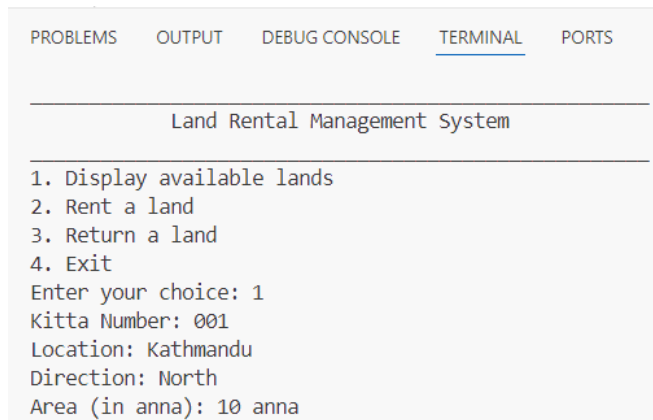
Kitta Number

Renter's Name

Rental Duration

Total Amount

7. **Save and Display Invoice:** The invoice is saved as a text file and displayed to the user, providing a complete summary of the rental transaction.



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

-----
                        Land Rental Management System
-----
1. Display available lands
2. Rent a land
3. Return a land
4. Exit
Enter your choice: 1
Kitta Number: 001
Location: Kathmandu
Direction: North
Area (in anna): 10 anna

```

Figure 11: Image for rent process 1

```

Price: 50000 NPR
=====
Kitta Number: 002
Location: Lalitpur
Direction: East
Area (in anna): 8 anna
Price: 40000 NPR
=====
Kitta Number: 003
Location: Butwal
Direction: East

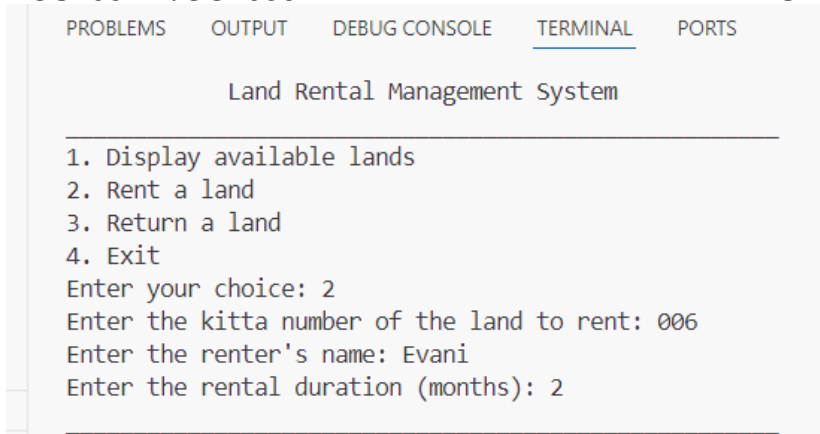
```

```

Area (in anna): 9 anna
Price: 50000 NPR
=====
Kitta Number: 005
Location: Lalitpur
Direction: North
Area (in anna): 20 anna
Price: 100000 NPR
=====

```

Figure 12: Image to display lands

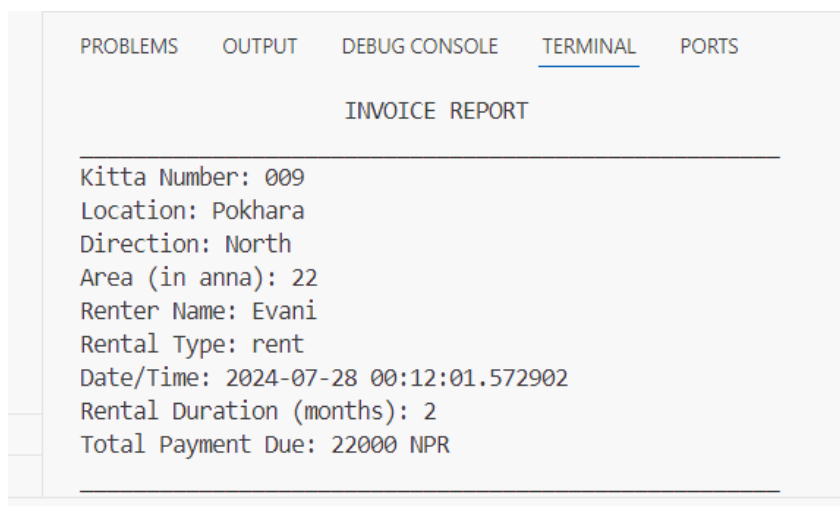


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Land Rental Management System

1. Display available lands
2. Rent a land
3. Return a land
4. Exit
Enter your choice: 2
Enter the kitta number of the land to rent: 006
Enter the renter's name: Evani
Enter the rental duration (months): 2
```

Figure 13:Image to rent land



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

INVOICE REPORT

Kitta Number: 009
Location: Pokhara
Direction: North
Area (in anna): 22
Renter Name: Evani
Rental Type: rent
Date/Time: 2024-07-28 00:12:01.572902
Rental Duration (months): 2
Total Payment Due: 22000 NPR
```

Figure 14:Rent invoice

Return Process:

1. **Menu Selection:** Users start the return process by selecting the return option from the main menu.
2. **User Input:** They provide the following details:

Kitta Number: The unique identifier for the land being returned.

Renter's Name: The name of the person returning the land.

Rental Duration: The original rental period in months.

Return Date: The date when the land is returned.

3. **Verify Rental Record:** The system checks if the land was previously rented and retrieves the rental record.
4. **Calculate Fines:** If the return date is past the rental period, fines are calculated based on the delay.
5. **Update Land Status:** The land status is updated to "Available" once returned.
6. **Generate Return Invoice:** An invoice is created, including details such as:

Kitta Number

Renter's Name

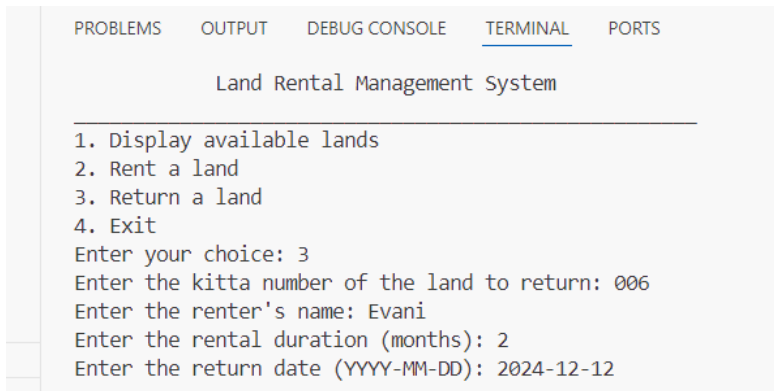
Rental Duration

Return Date

Total Amount

Any applicable fines

7. **Save and Display Invoice:** The return invoice is saved as a text file and shown to the user, providing a detailed summary of the return transaction and any additional charges.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

Land Rental Management System

1. Display available lands
2. Rent a land
3. Return a land
4. Exit
Enter your choice: 3
Enter the kitta number of the land to return: 006
Enter the renter's name: Evani
Enter the rental duration (months): 2
Enter the return date (YYYY-MM-DD): 2024-12-12
```

Figure 15:Return land

```
INVOICE REPORT
```

```
Kitta Number: 006
Location: Chitwan
Direction: East
Area (in anna): 10
Renter Name: Evani
Rental Type: return
Date/Time: 2024-07-27 23:59:51.146836
Rental Duration (months): 2
Initial Payment: 110000 NPR
```

```
10% Fine: 11000.0 NPR
Total Payment Due: 231000.0 NPR
```

```
Land returned successfully. Invoice saved as return_invoice_006.txt.
```

Figure 16:Return invoice

6.3 Creation of a TXT File

The program utilizes text files for storing and updating critical information:

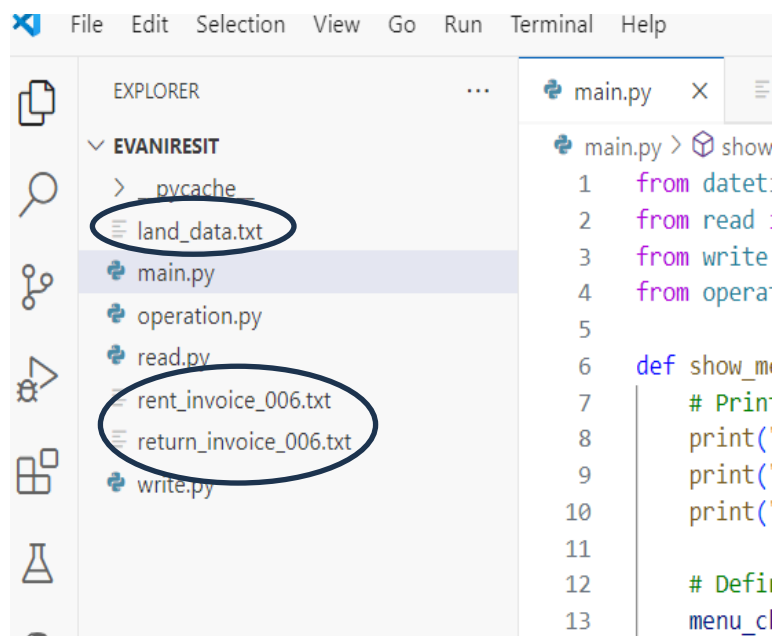


Figure 17:Image fir TXT

1. **land_data.txt**: Contains records of all land, including their availability and details.
2. **rented_land.txt**: Logs the details of all rented lands, including the rental transaction data.
3. **return_land.txt**: Records the details of land returns, including any fines and adjustments.

These text files ensure that data persists between program runs and is available for review or modification.

6.4 Opening Text and Showing the Bill

After each rental or return transaction, an invoice is generated and saved as a text file. This invoice includes detailed information about the transaction, such as the kitta number, rental or return date, customer details, and financial details. The invoice is also displayed to the user to confirm the transaction and provide a record for their reference.

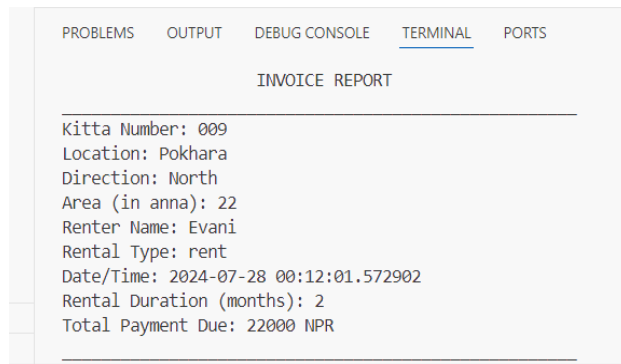


Figure 18: Bill for renting

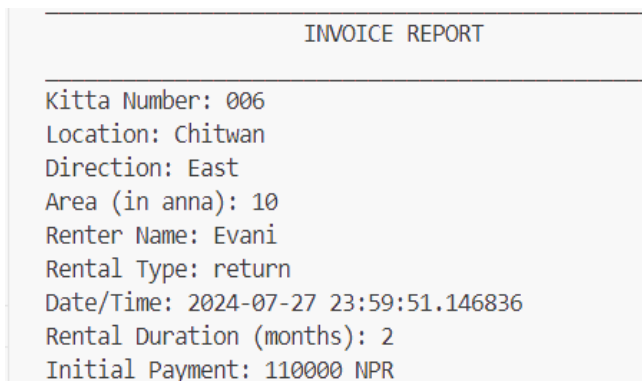


Figure 19: Bill for returning

6.5 Termination of the Program

The program concludes when the user selects the 'Exit' option from the menu. Before exiting, it ensures that all updates to the land data file are saved. A termination message is displayed to the user, confirming that the program has ended and all changes have been successfully recorded.

```

Land Rental Management System

1. Display available lands
2. Rent a land
3. Return a land
4. Exit
Enter your choice: 4
Exiting the program.

```

Figure 20:Exiting program

7. Testing:

7.1 Test 1

-Implementation of try, except:

Table 1:Testing 1

Objective	To show the implementation of try, except in the code.
Action	The input for kitta number is given in unsuitable data type.
Expected Result	A suitable error message should be shown.
Actual Result	A suitable error was shown when the input was in unsuitable data type.
Conclusion	Test was Successful.

```

File Edit Selection View Go Run Terminal Help
EVANIRESIT
> __pycache__
land_data.txt
main.py
operation.py
read.py
rent_invoice_006.txt
return_invoice_006.txt
write.py

main.py > show_menu
1 from datetime import datetime
2 from read import read_land_data
3 from write import write_land_data
4 from operation import land_rental, land_return
5
6 def show_menu():
7     # Print the menu header
8     print("\n" + "_"*53)
9     print(" " * 12 + "Land Rental Management System")
10    print("_"*53)
11
12    # Define and print menu options
13    menu_choices = [
14        "1. Display available lands",
15        "2. Rent a land",
16        "3. Return a land",
17        "4. Exit"
18    ]
19
20    for option in menu_choices:
21        print(option)
22
23    def main():
24        data_file = 'land_data.txt'
25
26    main()
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

1. Display available lands
2. Rent a land
3. Return a land
4. Exit
Enter your choice: 2
Enter the kitta number of the land to rent: EE
Enter the renter's name: Evani
Enter the rental duration (months): 1
Land not available for rent.
Unable to rent land. Please check the availability or kitta number.
Do you want to rent another land? (yes/no):

```

Figure 21: Image for test 1

7.2 Test 2:

- Selection of rent and return of lands:

Table 2: Testing 2

Objective	To show the selection of rent and return of lands.
Action	Non-existing values is given as input. Negative value is given as input.
Expected Result	A suitable error message should be shown.
Actual Result	A suitable message was shown when non-existing, negative values were given as input.
Conclusion	Test was Successful.

```
File Edit Selection View Go Run Terminal Help
EXPLORER
EVANIRESIT
  > __pycache__
  > land_data.txt
  > main.py
  > operation.py
  > read.py
  > rent_invoice_003.txt
  > rent_invoice_004.txt
  > rent_invoice_005.txt
  > rent_invoice_009.txt
  > return_invoice_009.txt
  > write.py
main.py
main.py > main
1 from datetime import datetime
2 from read import read_land_data
3 from write import write_land_data
4 from operation import land_rental, land_return
5
6 def show_menu():
7     # Print the menu header
8     print("\n" + "_" * 53)
9     print(" " * 12 + "Land Rental Management System")
10    print("_" * 53)
11
12    # Define and print menu options
13    menu_choices = [
14        "1. Display available lands",
15        "2. Rent a land",
16        "3. Return a land",
17        "4. Exit"
18    ]
19
20    for option in menu_choices:
21        print(option)
22
23    def main():
24        data_file = 'land_data.txt'
```

1. Display available lands
2. Rent a land
3. Return a land
4. Exit
Enter your choice: 2
Enter the kitta number of the land to rent: 369
Enter the renter's name: Evani
Enter the rental duration (months): 2
Land not available for rent.
Unable to rent land. Please check the availability or kitta number
Do you want to rent another land? (yes/no):

Figure 22: First image for test 2

```

1 from datetime import datetime
2 from read import read_land_data
3 from write import write_land_data
4 from operation import land_rental, land_return
5
6 def show_menu():
7     # Print the menu header
8     print("\n" + "_" * 53)
9     print(" " * 12 + "Land Rental Management System")
10    print("_" * 53)
11
12    # Define and print menu options
13    menu_choices = [
14        "1. Display available lands",
15        "2. Rent a land",
16        "3. Return a land",
17        "4. Exit"
18    ]
19
20    for option in menu_choices:
21        print(option)
22
23    def main():
24        data_file = 'land_data.txt'

```

Enter the kitta number of the land to rent: 369
Enter the renter's name: Evani
Enter the rental duration (months): 2
Land not available for rent.
Unable to rent land. Please check the availability or kitta number.
Do you want to rent another land? (yes/no): yes
Enter the kitta number of the land to rent: -333
Enter the renter's name: Evani
Enter the rental duration (months): 2
Land not available for rent.
Unable to rent land. Please check the availability or kitta number.
Do you want to rent another land? (yes/no):

Figure 23: Second image for test 2

Figure 9: Testing 2: Negative value as input

7.3 Test 3:

-File generation of renting of lands:

Table 3: Testing 3

Objective	To show the file generation process.
Action	Renting process done by entering the proper value.
Expected Result	A bill with unique id for rented items should be generated.

7.4 Test 4:

-File generation of renting of lands:

Table 4: Testing 4

Objective	To show the file generation process.
Action	Renting process done by entering the proper value.
Expected Result	A bill with unique id for returned items should be generated.
Actual Result	A bill and file with unique id for returned items was generated.
Conclusion	Test was Successful.

Table 4: Testing 4

The screenshot shows a code editor with the following components:

- EXPLORER:** A file tree on the left showing the project structure. The file `return_invoice_009.txt` is selected and highlighted.
- EDITOR:** The main area displays the content of `return_invoice_009.txt`, which is an invoice report. The text is as follows:


```

1
2
3
4 Kittu Number: 009
5 Location: Pokhara
6 Direction: North
7 Area (in anna): 22
8 Renter Name: Evani
9 Rental Type: return
10 Date/Time: 2024-07-27 22:34:58.410964
11 Rental Duration (months): 1
12 Initial Payment: 11000 NPR
13 10% Fine: 1100.0 NPR
14 Total Payment Due: 23100.0 NPR
15
      
```
- TERMINAL:** A panel at the bottom right showing the output of the program. It displays the same invoice details as the file content, followed by the message:


```

Land returned successfully. Invoice saved as return_invoice_009.txt.
      
```

Figure 25: Image for test 4

7.5 Test 5:*Table 5: Testing 5*

Objective	To show the update in stock of lands.
Action	<ol style="list-style-type: none">1. Availability being deducted while renting land and being added after returning the land.2. Not availability being showed after renting.
Expected Result	Stock of lands being updated.
Actual Result	Stock of lands was update.
Conclusion	Test was Successful.

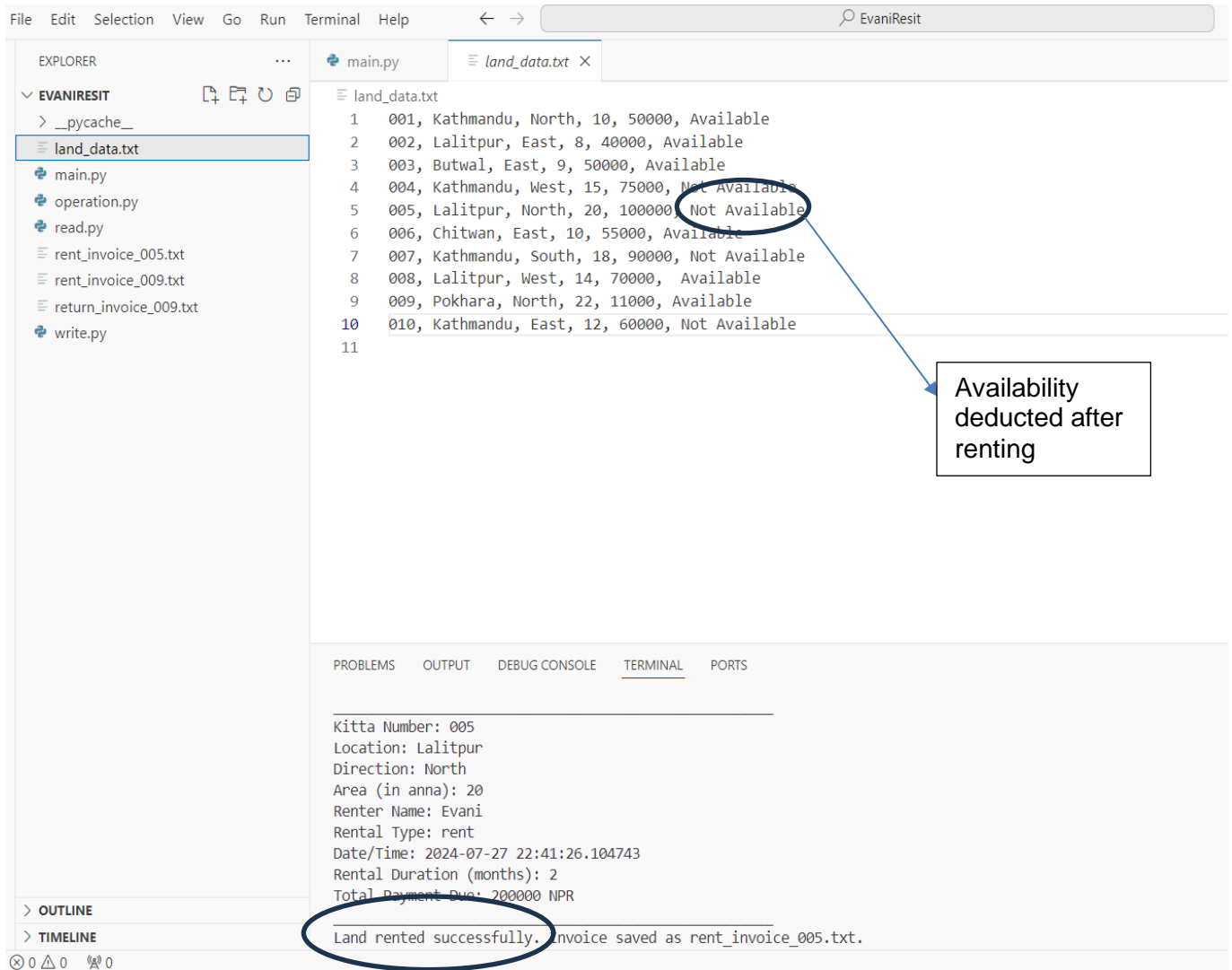


Figure 26: First image for test 5

Table 5: Testing 5

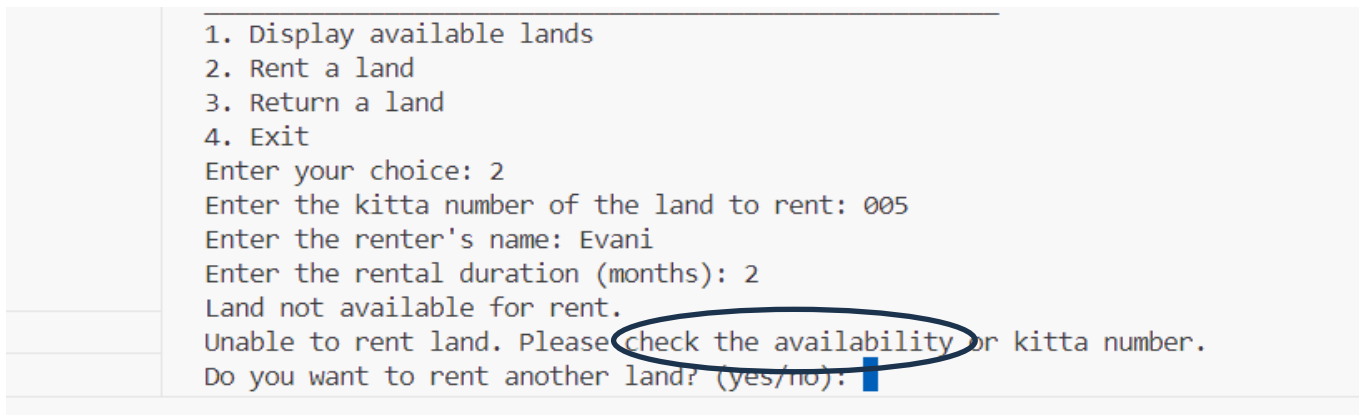


Figure 27: Second image for test 5

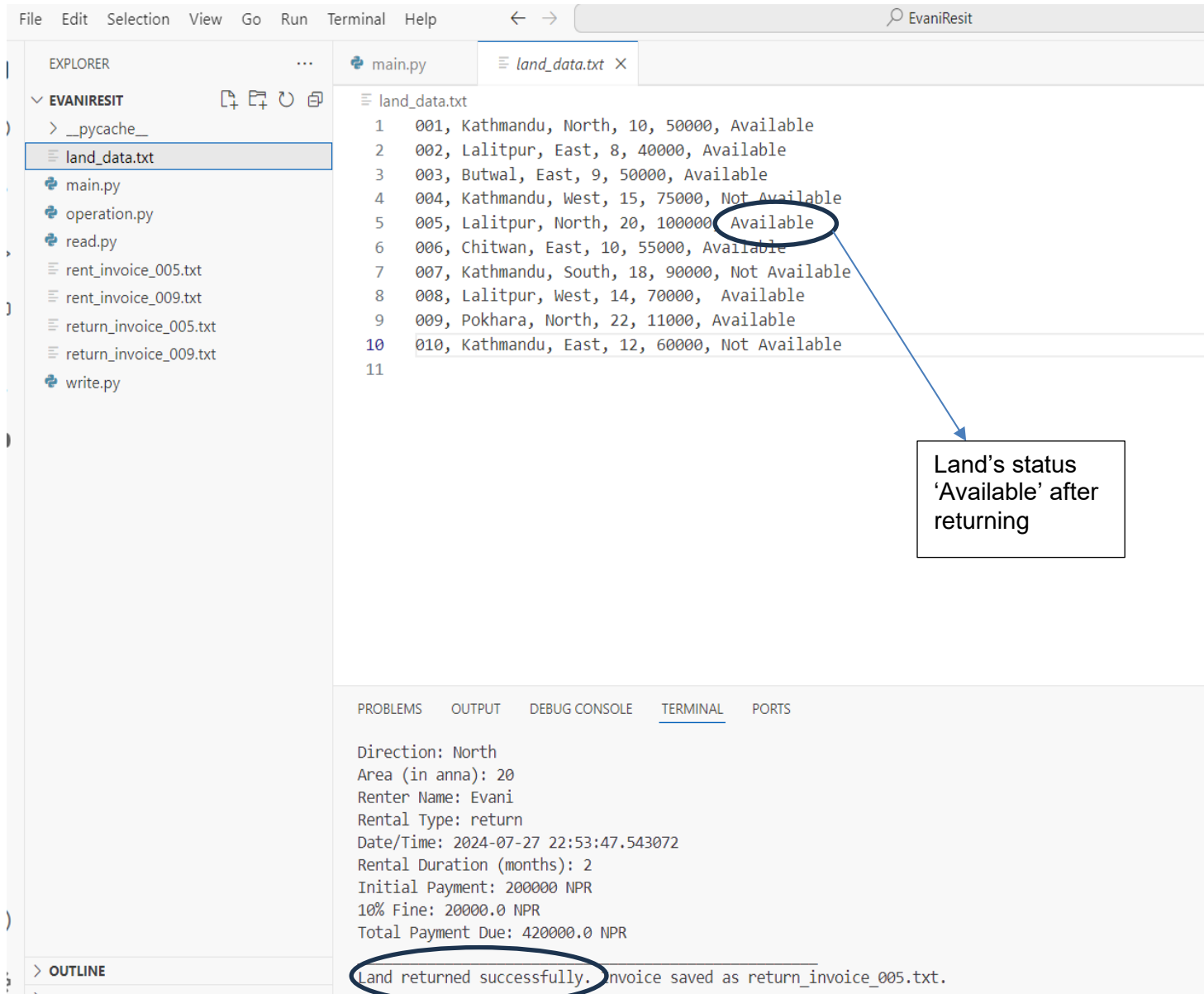


Figure 28: Third image for test 5

8. Conclusion:

The TechnoPropertyNepal Land Rental System has been successfully developed as a comprehensive software solution to manage the rental and return of land parcels across various locations in Nepal. The system provides efficient handling of land inventory, rental transactions, and returns while generating detailed invoices for each transaction. By leveraging Python IDLE, the project was built in a structured manner with a modular approach, allowing for easy maintenance and potential future expansions.

Key Achievements of the Project:

Streamlined Operations:

1. The TechnoPropertyNepal Land Rental System optimizes land rental processes by offering administrators a centralized platform to monitor and manage land inventory. This includes tracking available land, processing rental transactions efficiently, and handling land returns in a seamless manner.
2. The system reduces manual effort and speeds up decision-making for administrators, improving overall operational efficiency.

Detailed Invoices:

1. Detailed invoices are generated for each rental transaction, providing clear and accurate billing records for clients and administrators alike.
2. This feature enhances financial tracking and transparency, ensuring that all parties have a precise understanding of charges and payments.
3. It also aids in resolving disputes swiftly by providing a detailed record of each transaction.

Error Handling:

1. Comprehensive input validation and error handling mechanisms are implemented throughout the application to ensure smooth operation and reduce the occurrence of issues.
2. These measures help maintain data integrity and system stability by preventing invalid inputs and providing users with informative error messages and guidance.
3. This approach minimizes disruptions to the user experience and ensures consistent and reliable functionality.

User-Friendly Interface:

1. The system boasts an intuitive and user-centric interface that simplifies the process of managing land rental transactions.
2. Features such as clear menus, guided workflows, and straightforward data entry points make it easy for administrators to navigate and utilize the system effectively.
3. A well-designed interface reduces the learning curve for new users and promotes efficiency in day-to-day operations.

Fine Management:

1. The system includes a sophisticated fine management feature that calculates and applies fines for late returns.
2. By automating this process, the system ensures consistent treatment of customers, promoting accountability and timely returns.
3. This feature helps maintain fair business practices and strengthens the company's relationship with its clients by fostering trust and encouraging compliance with rental agreements.

Throughout the development process, the choice of Python IDLE as the primary tool facilitated code writing, testing, and debugging, contributing significantly to the successful completion of the project. Python IDLE's user-friendly interface and debugging tools enhanced productivity and ensured the system was robust and reliable. The implementation of this system supports TechnoPropertyNepal's business operations by offering an efficient and user-friendly solution for managing land rentals. It centralizes and automates key tasks, improving accuracy, reducing manual effort, and speeding up decision-making. In the future, potential enhancements could include automated contract renewals, advanced reporting tools, and integration with other systems for a more seamless experience. These upgrades would expand functionality, reduce manual workload, and enhance overall efficiency, contributing to TechnoPropertyNepal's success. Furthermore, regular updates and user feedback will be crucial in continuously improving the system, ensuring it remains aligned with evolving business needs.

8. Bibliography:

geeksforgeeks, 2024. *geeksforgeeks*. [Online]
Available at: <https://www.geeksforgeeks.org/python-data-types/>
realpython, 2024. *realpython*. [Online]
Available at: <https://realpython.com/python-data-types/>
w3schools, 2024. *w3schools*. [Online]
Available at: https://www.w3schools.com/python/python_datatypes.asp

10. Appendix:

10.1 Main.py

```
from datetime import datetime

from read import read_land_data

from write import write_land_data

from operation import land_rental, land_return

def show_menu():

    # Print the menu header

    print("\n" + "_"*53)

    print(" " * 12 + "Land Rental Management System")

    print("_"*53)


    # Define and print menu options

    menu_choices = [

        "1. Display available lands",

        "2. Rent a land",

        "3. Return a land",

        "4. Exit"

    ]

    for option in menu_choices:

        print(option)


def main():

    data_file = 'land_data.txt'
```

```
# Check if data file exists

try:

    with open(data_file, 'r') as file:

        pass # File exists and is accessible

except FileNotFoundError:

    print("Land data file not found.")

    return


try:

    # Read land records from the file

    land_record_list = read_land_data(data_file)

except Exception as e:

    # Print error message if reading data fails

    print("Error reading land data: " + str(e))

    return


while True:

    # Display the menu and get user's choice

    show_menu()

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

```
if select_choice == '1':

    # Filter and display available lands

    land_availability = [land for land in land_record_list if land['status'] ==
'Available']

    if not land_availability:

        print("No lands available for rent.")

    else:

        for land in land_availability:

            print("Kitta Number: " + land['kitta_number'])

            print("Location: " + land['location'])

            print("Direction: " + land['direction'])

            print("Area (in anna): " + str(land['area_anna']) + " anna")

            print("Price: " + str(land['price']) + " NPR")

            print("=" * 20)

elif select_choice == '2':

    while True: # For multiple rentals

        try:

            # Get rental details and process rental

            kitta_number = input("Enter the kitta number of the land to rent: ")

            renter_name = input("Enter the renter's name: ")

            rental_duration = int(input("Enter the rental duration (months): "))

            if land_rental(land_record_list, kitta_number, renter_name,
rental_duration):
```



```
        write_land_data(data_file, land_record_list)

    else:

        print("Unable to rent land. Please check the availability or kitta
number.")

except ValueError:

    # Handle invalid rental duration input

    print("Invalid input for rental duration. Please enter a number.")

    # Ask if the user wants to rent another land

    another_rent = input("Do you want to rent another land? (yes/no): ")

    if another_rent == 'no':

        break # Exit the loop if the user does not want to rent another land

elif select_choice == '3':

    try:

        # Get return details and process return

        kitta_number = input("Enter the kitta number of the land to return: ")

        renter_name = input("Enter the renter's name: ")

        rental_duration = int(input("Enter the rental duration (months): "))

        return_date_str = input("Enter the return date (YYYY-MM-DD): ")

        return_date = datetime.now()

        if land_return(land_record_list, kitta_number, renter_name,
```

```
        write_land_data(data_file, land_record_list)

    else:

        print("Unable to return land. Please check the kitta number and rental
details.")

    except ValueError as e:

        # Handle invalid return details input

        print("Invalid input: " + str(e))

    elif select_choice == '4':

        # Exit the program

        print("Exiting the program.")

        break

    else:

        # Handle invalid menu choice

        print("Invalid choice. Please try again.")

main()
```

10.2 write.py

```
def write_land_data(data_file, land_record_list):
    # Open the file in write mode
    with open(data_file, 'w') as file:
```

```
# Iterate over each land record
for land_info in land_record_list:
    # Format and write each land record to the file
    line = (str(land_info['kitta_number']) + ', ' +
            land_info['location'] + ', ' +
            land_info['direction'] + ', ' +
            str(land_info['area_anna']) + ', ' +
            str(land_info['price']) + ', ' +
            land_info['status'] + '\n')
    # Write the formatted line to the file
    file.write(line)
```

10.3 read.py:

```
def read_land_data(data_file):
    land_record_list = [] # Initialize an empty list to store land records
    try:
        with open(data_file, 'r') as file: # Open the file in read mode
            for line in file: # Iterate over each line in the file
                parts = line.strip().split(', ') # Split the line into parts based on comma
                and space
                if len(parts) != 6: # Check if the line has the correct number of parts
                    print("Warning: Incorrect format in line: " + line) # Print warning for
                    incorrect format
                    continue # Skip to the next line

                # Extract and convert values from parts
                kitta_number = parts[0]
                location = parts[1]
                direction = parts[2]
```

```
    area_anna = int(parts[3]) # Convert area_anna to integer
    price = int(parts[4]) # Convert price to integer
    status = parts[5]

    try:
        # Create a dictionary for the land record
        land_info = {
            'kitta_number': kitta_number,
            'location': location,
            'direction': direction,
            'area_anna': area_anna,
            'price': price,
            'status': status
        }
        land_record_list.append(land_info) # Append the land record to the list
    except ValueError:
        print("Warning: Invalid data in line: " + line) # Print warning for
invalid data
        continue # Skip to the next line
    except FileNotFoundError:
        print("Error: File '" + data_file + "' not found.") # Print error if the file is
not found
    except Exception as e:
        print("Error: " + str(e)) # Print any other exceptions that occur
    return land_record_list # Return the list of land records
```

10.4 operation.py:

```
from datetime import datetime

def create_invoice_report(
    land_info,
    customer_name,
    rental_duration,
    transaction_type,
    current_time,
    fine=0
):
    # Create the invoice filename based on transaction type and kitta number
    invoice_filename = transaction_type + '_invoice_' +
    str(land_info['kitta_number']) + '.txt'

    # Define border and header for invoice
    border = "_" * 53
    header = "INVOICE REPORT".center(53)

    # Construct the invoice info
    invoice_info = border + "\n"
    invoice_info += header + "\n"
    invoice_info += border + "\n"

    # Add land and rental details to the invoice
    invoice_info += "Kitta Number: " + str(land_info['kitta_number']) + "\n"
    invoice_info += "Location: " + land_info['location'] + "\n"
    invoice_info += "Direction: " + land_info['direction'] + "\n"
    invoice_info += "Area (in anna): " + str(land_info['area_anna']) + "\n"
    invoice_info += "Renter Name: " + customer_name + "\n"
    invoice_info += "Rental Type: " + transaction_type + "\n"
    invoice_info += "Date/Time: " + str(current_time) + "\n"
    invoice_info += "Rental Duration (months): " + str(rental_duration) + "\n"

    # Calculate total amount and total payment due
    total_amount = land_info['price'] * rental_duration
    total_amount_due = total_amount + fine

    # Apply 10% fine
    if transaction_type == 'return':
        ten_percent_fine = 0.10 * total_amount
        total_amount_due += ten_percent_fine

    if fine > 0:
        invoice_info += "Initial Payment: " + str(fine) + " NPR\n"
        invoice_info += "10% Fine: " + str(ten_percent_fine) + " NPR\n"

    # Add total payment due
    invoice_info += "Total Payment Due: " + str(total_amount_due) + " NPR\n"
```

```
    invoice_info += border

    # Print and save the invoice info
    print(invoice_info)
    try:
        with open(invoice_filename, 'w') as file:
            file.write(invoice_info)
    except IOError as e:
        print("Error writing invoice to file: " + str(e))

    return invoice_filename

def land_rental(
    land_records,
    kitta_number,
    customer_name,
    rental_duration
):
    # Get the current date and time
    current_time = datetime.now()

    # Convert land records to a dictionary
    land_dict = {land['kitta_number']: land for land in land_records}

    # Check if the land is available
    if kitta_number in land_dict and land_dict[kitta_number]['status'] ==
'Available':
        land_info = land_dict[kitta_number]
        land_info['status'] = 'Not Available'
        invoice_filename = create_invoice_report(
            land_info,
            customer_name,
            rental_duration,
            'rent',
            current_time
        )
        print("Land rented successfully. Invoice saved as " + invoice_filename +
".")
        return True

    print("Land not available for rent.")
    return False

def land_return(
    land_records,
    kitta_number,
    customer_name,
    rental_duration,
    return_date
):
    # Get the current date and time
    current_time = datetime.now()

    # Convert land records to a dictionary
```

```
land_dict = {land['kitta_number']: land for land in land_records}

# Check if the land is currently rented
if kitta_number in land_dict and land_dict[kitta_number]['status'] == 'Not
Available':
    land_info = land_dict[kitta_number]

    # Calculate the end date of the rental period
    rent_end_date = return_date.replace(month=return_date.month -
rental_duration)
    if rent_end_date.month <= 0:
        rent_end_date = rent_end_date.replace(year=rent_end_date.year - 1,
month=rent_end_date.month + 12)

    # Calculate fine if the return date is after the rental period
    if return_date > rent_end_date:
        fine_duration = (return_date - rent_end_date).days // 30
        fine = fine_duration * land_info['price']
    else:
        fine = 0

    land_info['status'] = 'Available'
    invoice_name = create_invoice_report(
        land_info,
        customer_name,
        rental_duration,
        'return',
        current_time,
        fine
    )
    print("Land returned successfully. Invoice saved as " + invoice_name +
".")
    return True

print("Land is not currently rented.")
return False
```