UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

School of Computing

4C0SC006C.2 Software Development 1

Module: 4C0SC006C.2 Software Development 1

Module Leader: Mr. Pooravi Guganathan

Assessment Type: Individual

Issue Date: 06/04/2024

Student Name	IIT ID	UOW ID
R.M.W.D.Anjalee	20230179	2084393

i. Acknowledgment

I extend my deepest gratitude to Professor Mr. Pooravi Guganathan and our tutorial lecturer. Their guidance, patience, and expertise have profoundly shaped our programming journey. We are thankful for their unwavering support, invaluable advice, and passion for sharing knowledge. Their enthusiasm and dedication have inspired us to excel and pursue our goals with vigor. We are truly fortunate to have such remarkable mentors who continually ignite our curiosity and drive for learning. Their impact on our understanding and appreciation of programming is immeasurable, and we are sincerely grateful for all they have done to enrich our educational experience.

ii. Abstract

This assignment involves enhancing the Personal Finance Tracker by developing a graphical user interface (GUI) using Tkinter, building on our knowledge of Python, dictionaries, and file I/O. This advanced version should not only display the information from a provided JSON file but also incorporate object-oriented programming (OOP) concepts for the GUI components. Additionally, your application will include a search function and a sorting feature, similar to a file explorer, to manage and analyze financial transactions more effectively.

Table of Contents

iii. List of tables

iv. List of figures

1. Pseudo Codes

1.1 Part B

```
BEGIN
    Import json module

    Define FILENAME as "FinanceTracker.json"

    Function load_transactions():
        Try to open FILENAME for reading
        If FileNotFoundError:
            Initialize transactions as an empty dictionary
        Otherwise:
            Load transactions from the file
        Return transactions

    Function save_transactions(transactions):
        Open FILENAME for writing
        Write transactions to the file in JSON format
        Close the file

    Function read_bulk_transactions_from_file(filename):
        Try to open the specified filename for reading
        If FileNotFoundError:
            Print a message indicating the file was not found
            Return an empty dictionary
        Otherwise:
            Load transactions from the file
            Return transactions
```

```
Function add_transaction(category, amount, date):
    Load existing transactions
    If category not in transactions:
        Create a new list for the category
    Append a new transaction to the category with amount and date
    Save transactions to the file
    Print "Transaction added!"

Function view_transactions():
    Load all transactions
    For each category and its transactions:
        Print category name
        For each transaction in transactions:
            Print index, amount, and date

Function delete_transaction(category, index):
    Load existing transactions
    Calculate the new index by subtracting 1 from the provided index
    If category exists in transactions and new index is valid:
        Delete the transaction at the new index from the category
        If the category becomes empty, delete it
        Save transactions to the file
        Print "Transaction Deleted!"
    Otherwise, print "Invalid category or index!"

Function update_transaction(category, index, amount, date):
    Load existing transactions
    Calculate the new index by subtracting 1 from the provided index
    If category exists in transactions and new index is valid:
        Update the transaction at the new index with the new amount and date
        Save transactions to the file
```

Print "Successfully Updated!"

Otherwise, print "Invalid category or index!"

Function display_summary():

Load all transactions

For each category and its transactions:

Calculate the total amount for each category

Print category name and total amount

Function launch_gui():

Import subprocess module

Run the Finance Tracker GUI script using subprocess

Function main_menu():

Loop indefinitely:

Print menu options

Prompt user for choice

Based on user choice:

Add Transaction: Prompt for category, amount, and date, then call add_transaction()

View Transactions: Call view_transactions()

Update Transaction: Prompt for category, index, new amount, and new date, then call update_transaction()

Summary of Transactions: Call display_summary()

Delete Transaction: Prompt for category and index, then call delete_transaction()

Launch GUI: Call launch_gui() to launch GUI

Exit: Print exit message and break the loop

If __name__ is "__main__":

Call main_menu() to start the program

END

1.2 Part C (GUI)

```
BEGIN
    Import tkinter module as tk
    Import ttk module from tkinter
    Import json module

    Define a class named FinanceTrackerGUI:

        Constructor __init__(self, root):
            Initialize root window with title "Personal Finance Tracker"
            Set window size to 400x50 pixels
            Disable window resizing
            Call method create_widgets() to create GUI widgets
            Load transactions from "transactions.json" file

        Method create_widgets(self):
            Create a label widget for the title "Personal Finance Tracker"
            Customize label font and pack it
            Create a frame widget for table and scrollbar
            Configure frame properties and pack it
            Create a Treeview widget for displaying transactions
            Set column headings for category, amount, and date
            Configure column widths and pack the Treeview
            Create a vertical scrollbar for the Treeview
            Configure the scrollbar and link it with the Treeview
            Create a search bar Entry widget
            Create a search button Button widget and link it to search_transactions()

        Method load_transactions(self, filename):
            Try to open "FinanceTracker.json" file for reading
            If FileNotFoundError occurs, initialize transactions as an empty dictionary
            Otherwise, load transactions from the file
            Return transactions

        Method display_transactions(self, transactions):
            Delete existing entries from the Treeview
            Iterate over transactions dictionary:
                Iterate over transactions for each category:
                    Insert transaction values into the Treeview

        Method search_transactions(self):
            Get search query from the search bar
            Create a filtered_transactions dictionary
            Iterate over transactions dictionary:
                Iterate over transactions for each category:
                    Check if query matches transaction amount, date, or category
```

If matches found, add transaction to filtered_transactions
Display filtered transactions in the Treeview

Method sort_by_column(self, col, reverse):
Get data from Treeview children
Sort data based on column and reverse flag
Rearrange Treeview entries according to sorted data

Define a function named main():
Create a root Tkinter window
Instantiate FinanceTrackerGUI object with root window
Display transactions on startup
Start the Tkinter event loop

If the script is run directly:
Call main() function to start the program

END

2. Python Codes

2.1 Part B

```
#importing json
import json

# Global dictionary to store transactions
FILENAME = "FinanceTracker.json"

  # File handling functions

# Function to load transactions from the file
def load_transactions():
    try:
        with open(FILENAME, "r") as file:
            transactions = json.load(file)
    except FileNotFoundError:
        transactions = {}  # If file not found, initialize an empty dictionary
    return transactions

# Function to save transactions to the file
def save_transactions(transactions):
    with open(FILENAME, "w") as file:
        json.dump(transactions, file, indent=2)

# Function to read transactions from a file
def read_bulk_transactions_from_file(filename):
    try:
        with open(filename, "r") as file:
            transactions = json.load(file)
        return transactions
    except FileNotFoundError:
        print(f"File '{filename}' not found.")
        return {}  # If file not found, return an empty dictionary

# Feature implementations

# Function to add a new transaction
def add_transaction(category, amount, date):
    transactions = load_transactions()  # Load existing transactions
    if category not in transactions:
        transactions[category] = []  # If category doesn't exist, create a new one
    transactions[category].append({"amount": amount, "date": date})  # Add new
transaction
```

```
    save_transactions(transactions)  # Save updated transactions to the file
    print("Transaction added!")

# Function to view all transactions
def view_transactions():
    all_transactions = load_transactions()  # Load all transactions
    for category, transactions in all_transactions.items():
        print(f"{category}:")
        for index, transaction in enumerate(transactions, start=1):
            print(f" {index}. Amount: {transaction['amount']}, Date: {transaction['date']}")
        print()

def delete_transaction(category, index):
    transactions = load_transactions()  # Load existing transactions
    new_index = index - 1
    if category in transactions and 0 <= new_index < len(transactions[category]):
        del transactions[category][new_index]  # Delete transaction
        if len(transactions[category]) == 0:
            del transactions[category]  # If no transactions left in the category, delete the
category
        save_transactions(transactions)  # Save updated transactions to the file
        print("Transaction Deleted!")
    else:
        print("Invalid category or index!")

# Function to update a transaction
def update_transaction(category, index, amount, date):
    transactions = load_transactions()  # Load existing transactions
    new_index = index - 1
    if category in transactions and 0 <= new_index < len(transactions[category]):
        transactions[category][new_index] = {"amount": amount, "date": date}  # Update
transaction
        save_transactions(transactions)  # Save updated transactions to file
        print("Successfully Updated!")
    else:
        print("Invalid category or index!")

# Function to display summary of transactions
def display_summary():
    all_transactions = load_transactions()  # Load all transactions
    for category, transactions in all_transactions.items():
        total_amount = sum(transaction['amount'] for transaction in transactions)  #
Calculate total amount
        print(f"{category}: Total Amount: {total_amount}")

# Function to launch gui
def launch_gui():
```

```
    import subprocess
    subprocess.run(["python", "Finance Tracker GUI.py"])

# Main menu function
def main_menu():
    while True:
        print("\n1. Add Transaction")
        print("2. View Transactions")
        print("3. Update Transaction")
        print("4. Summary of Transaction")
        print("5. Delete Transaction")
        print("6. Launch GUI")
        print("7. Exit")
        choice = input("Enter your choice: ")
        if choice == "1":
            category = input("Enter category: ")
            amount = float(input("Enter amount: "))
            date = input("Enter date (YYYY-MM-DD): ")
            add_transaction(category, amount, date)
        elif choice == "2":
            view_transactions()
        elif choice == "3":
            view_transactions()
            category = input("Enter category: ")
            index = int(input("Enter index to update: "))
            amount = float(input("Enter new amount: "))
            date = input("Enter new date (YYYY-MM-DD): ")
            update_transaction(category, index, amount, date)
        elif choice == "4":
            display_summary()
        elif choice == "5":
            view_transactions()
            category = input("Enter category: ")
            index = int(input("Enter index to delete: "))
            delete_transaction(category, index)
        elif choice == "6":
            launch_gui()
        elif choice == "7":
            print("Thanks for using FinanceTracker!")
            break
        else:
            print("Invalid choice!")

# Entry point of the program
if __name__ == "__main__":
    main_menu()  # Call the main_menu function to start the program
```

2.2 Part C (GUI)

```
 import tkinter as tk
from tkinter import ttk
import json


class FinanceTrackerGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Personal Finance Tracker")# Set window title
        self.root.title=ttk.Frame(self.root, width=400, height=50)
        self.root.resizable(False,False)
        self.create_widgets()# Call method to create GUI widgets
        self.transactions = self.load_transactions("transactions.json")# Load transactions from
file


    def create_widgets(self):


        #Title name lable
        label=tk.Label(self.root.title,text='Personal Finance Tracker')
        label['font']=('Britannic Bold',15)
        label.pack()
        self.root.title.pack()


        # Frame for table and scrollbar
        self.frame = ttk.Frame(self.root, width=400, height=300, borderwidth=10,
relief=tk.GROOVE)
        self.frame.pack_propagate(False)
        self.frame.pack(side='bottom')


        # Treeview for displaying transactions
```

```
        self.treeview=ttk.Treeview(self.frame, columns=('Category','Amount','Date'), show=
'headings')
        self.treeview.heading('Category', text='Category', command=lambda:
self.sort_by_column("Category", False))
        self.treeview.heading('Amount', text='Amount', command=lambda:
self.sort_by_column("Amount", False))
        self.treeview.heading('Date', text='Date', command=lambda:
self.sort_by_column("Date", False))
        self.treeview.column('Category', width=100)
        self.treeview.column('Amount', width=100)
        self.treeview.column('Date', width=100)
        self.treeview.pack(fill='both', expand=True)

        # Scrollbar for the Treeview
        self.scrollbar = ttk.Scrollbar(self.treeview, orient='vertical',
command=self.treeview.yview)
        self.scrollbar.pack(side="right", fill="y")
        self.treeview.configure(yscrollcommand=self.scrollbar.set)

        # Search bar and button
        self.search_var = tk.StringVar()
        self.search_entry = ttk.Entry(self.root, textvariable=self.search_var)
        self.search_entry.pack(side="top")
        self.search_button = ttk.Button(self.root, text="Search",
command=self.search_transactions)
        self.search_button.pack(side="top")

    def load_transactions(self, filename):
        try:
            with open("FinanceTracker.json", "r") as file:
                transactions = json.load(file)# Load transactions from JSON file
        except FileNotFoundError:
            transactions = {}# If file not found, initialize transactions as empty dictionary
```

```
        return transactions

    def display_transactions(self, transactions):
        # Remove existing entries
        for item in self.treeview.get_children():
            self.treeview.delete(item)

        # Add transactions to the treeview
        for category, transactions_list in transactions.items():
            for transaction_dict in transactions_list:
                self.treeview.insert('', 'end', values=(category, transaction_dict['amount'],
transaction_dict['date']))

    def search_transactions(self):
        # Placeholder for search functionality
        query = self.search_var.get().lower()
        filtered_transactions = {}
        for category, category_transactions in self.transactions.items():
            filtered_category_transactions = []
            for transaction in category_transactions:
                if query in str(transaction["amount"]).lower() or query in transaction["date"].lower()
or query in category.lower():
                    filtered_category_transactions.append(transaction)
            if filtered_category_transactions:
                filtered_transactions[category] = filtered_category_transactions
        self.display_transactions(filtered_transactions)

    def sort_by_column(self, col, reverse):
        # Placeholder for sorting functionality
        data = [(self.treeview.set(child, col), child) for child in self.treeview.get_children("")]
        data.sort(reverse=reverse)
        for index, (val, child) in enumerate(data):
```

```
            self.treeview.move(child, "", index)
        self.treeview.heading(col, command=lambda: self.sort_by_column(col, not reverse))


def main():
    root = tk.Tk()
    app = FinanceTrackerGUI(root)
    app.display_transactions(app.transactions)# Display transactions on startup
    root.mainloop()

if __name__ == "__main__":
    main()
```

3. Test Plan Summary

3.1 Test cases and screen shots

3.1.1 Part B

Test Component	Test No	Test Input	Expected Result	Actual Result	Pass / Fail
Main Menu	1	None	Displaying the main menu with options and asking choice.	Displaying the main menu with options and asking choice.	Pass
Add Transactions	2.1	Choice:1 Category: Salary Amount: 100000 Date (YYYY-MMDD): 2024-04-01	Display “Transaction added!”	Display “Transaction added!”	Pass
	2.2	Choice: 1 Category: Tax Amount: 1000 Date (YYYY-MMDD): 2024-04-10	Display “Transaction added!”	Display “Transaction added!”	Pass
	2.3	Choice: 1 Category: tax Amount: 200 Date (YYYY-MMDD): 2024-04-15	Display “Transaction added!”	Display “Transaction added!”	Pass

	2.4	Choice: 1 Category: House holds Amount: 50000 Date (YYYY-MMDD): 2024-04-17	Display "Transaction added!"	Display "Transaction added!"	Pass
View Transactions	3	Choice: 2	Display all saved transactions.	Display all saved transactions.	Pass
Update Transactions	4.1	Choice:3 Category: House holds Index to update: 1 New Amount: 60000 New Date (YYYY-MMDD): 2024-04-21	Display "Successfully Updated!"	Display "Successfully Updated!"	Pass
Display Summary	5	Choice: 4	Display all the Transaction categories and their Total amount	Display all the Transaction categories and their Total amount	Pass
Delete Transaction	6	Choice: 5	Delete the selected transaction and Display "Transaction deleted!"	Delete the selected transaction and Display "Transaction deleted!"	Pass
Launch GUI	7	Choice: 6	Launches the graphical user interface	Launches the graphical user interface	Pass
Exit	8	Choice: 7	Exiting From the program and display "Thanks for using FinanceTracker"	Exiting From the program and display "Thanks for using FinanceTracker	Pass

Table 1/Part B test cases

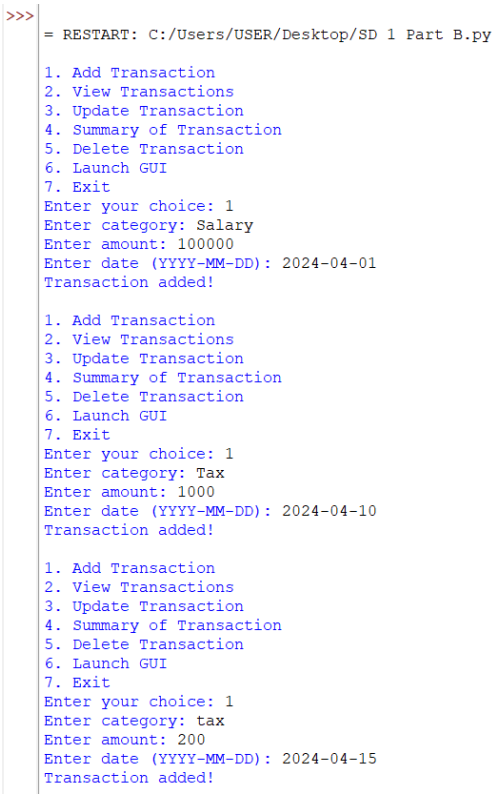
>>>
= RESTART: C:/Users/USER/Desktop/SD 1 Part B.py
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Summary of Transaction
5. Delete Transaction
6. Launch GUI
7. Exit
Enter your choice: 1
Enter category: Salary
Enter amount: 100000
Enter date (YYYY-MM-DD): 2024-04-01
Transaction added!
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Summary of Transaction
5. Delete Transaction
6. Launch GUI
7. Exit
Enter your choice: 1
Enter category: Tax
Enter amount: 1000
Enter date (YYYY-MM-DD): 2024-04-10
Transaction added!
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Summary of Transaction
5. Delete Transaction
6. Launch GUI
7. Exit
Enter your choice: 1
Enter category: tax
Enter amount: 200
Enter date (YYYY-MM-DD): 2024-04-15
Transaction added!

Figure 1

```
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Summary of Transaction
5. Delete Transaction
6. Launch GUI
7. Exit
Enter your choice: 1
Enter category: House holds
Enter amount: 50000
Enter date (YYYY-MM-DD): 2024-04-17
Transaction added!

1. Add Transaction
2. View Transactions
3. Update Transaction
4. Summary of Transaction
5. Delete Transaction
6. Launch GUI
7. Exit
Enter your choice: 2
Salary:
 1. Amount: 100000.0, Date: 2024-04-01

Tax:
 1. Amount: 1000.0, Date: 2024-04-10

tax:
 1. Amount: 200.0, Date: 2024-04-15

House holds :
 1. Amount: 50000.0, Date: 2024-04-17
```

Figure 2

```
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Summary of Transaction
5. Delete Transaction
6. Launch GUI
7. Exit
Enter your choice: 3
Salary:
 1. Amount: 100000.0, Date: 2024-04-01

Tax:
 1. Amount: 1000.0, Date: 2024-04-10

tax:
 1. Amount: 200.0, Date: 2024-04-15

House holds :
 1. Amount: 50000.0, Date: 2024-04-17

Enter category: House holds
Enter index to update: 1
Enter new amount: 60000
Enter new date (YYYY-MM-DD): 2024-04-21
Successfully Updated!

1. Add Transaction
2. View Transactions
3. Update Transaction
4. Summary of Transaction
5. Delete Transaction
6. Launch GUI
7. Exit
Enter your choice: 4
Salary: Total Amount: 100000.0
Tax: Total Amount: 1000.0
tax: Total Amount: 200.0
House holds : Total Amount: 60000.0
```

Figure 3

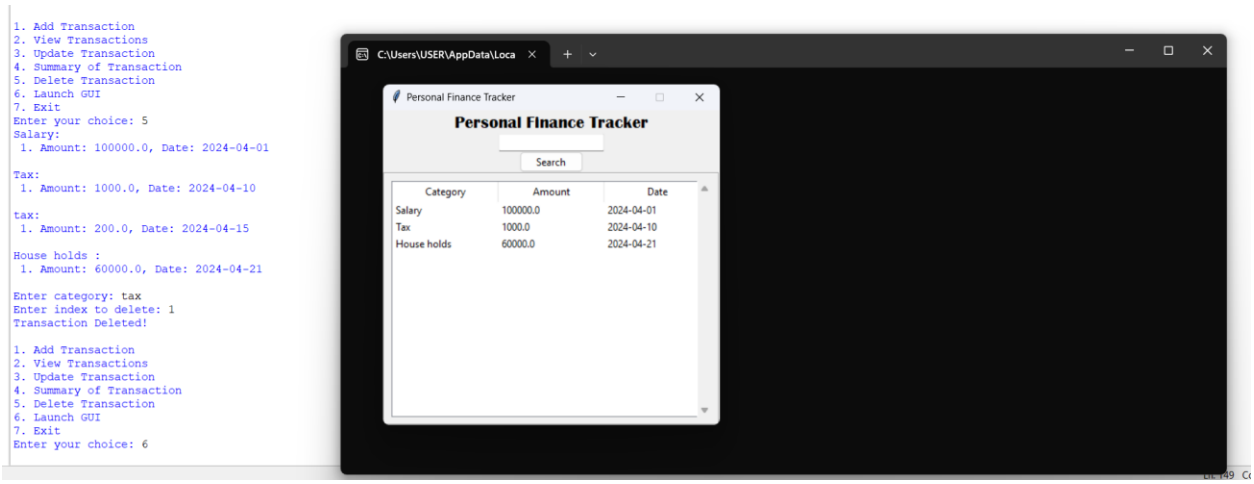

Figure 4

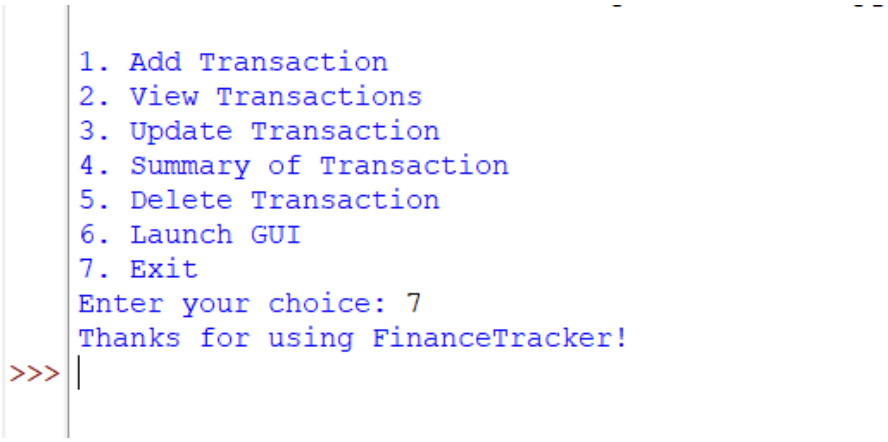

Figure 5

3.1.2 Part C (GUI)

Test No	Test Component	Test case	Expected Result	Actual Result	Pass / Fail
01	Loading Transactions	Verify that the transactions from the JSON file load properly.	Transactions are loaded and displayed in the GUI.	Transactions are loaded and displayed in the GUI.	Pass
02	Displaying Transactions	Verify that transactions are displayed properly in the GUI.	Only transactions matching the search query are shown.	Only transactions matching the search query are shown.	Pass
03	Searching Transactions	Verify that the search feature accurately filters transactions.	Transactions are displayed with correct category, amount, and date.	Transactions are displayed with correct category, amount, and date.	Pass
04	Sorting Transactions	Verify that sorting functionality works as expected	Transactions are sorted in ascending and descending order when clicking on column headers.	Transactions are sorted in ascending and descending order when clicking on column headers.	Pass
05	Scrolling Transactions	Verify that the scrollbar allows scrolling through a large number of transactions	All transactions can be smoothly scrolled through using the scrollbar, either vertically or horizontally as needed.	All transactions can be smoothly scrolled through using the scrollbar, either vertically or horizontally as needed.	Pass

Table 2/Part C test cases(gui)

3.2 Storing of the Transactions

3.2.1 Part B

The transactions are stored in a file named "FinanceTracker.json" using JSON format. The “Save_transactions” function opens this file for writing and writes the transactions to it in JSON format. Conversely, the “Load_transactions” function attempts to open the same file for reading and loads the transactions from it. Additionally, there's a function called “Read_bulk_transactions_from_file” which can read transactions from any specified filename. This setup allows for easy storage and retrieval of transaction data in a structured format.

3.2.2 Part C (GUI)

The transactions are stored in a JSON file named FinanceTracker.json, as specified in the load_transactions method within the FinanceTrackerGUI class. This method loads transactions from the JSON file when the GUI is initialized. If the file is not found, it initializes transactions as an empty dictionary.

3.3 Process Description

3.3.1 Part B

3.3.1.1 Adding Transactions

- Ask user for transaction details
- Create a new transaction and add it to Finance Tracker
- Save the updated Finance Tracker data to the file

3.3.1.2 Viewing Transactions

- Display all transactions with their details

3.3.1.3 Updating Transactions

- Update the transaction with new details

3.3.1.4 Display the summary

- Calculate total income, total expenses, and net income and display

3.3.1.5 Deleting a transaction

- Deleting an existing transaction

3.3.1.6 Launching GUI

- Provides a similar functionalities with a more user-friendly interface.

3.3.2 Part C (GUI)

3.3.2.1 Initialization

- The program initializes a Tkinter root window for the GUI
- The "FinanceTrackerGUI" class is created, configuring the window's title, dimensions, and further attributes.

3.3.2.2 Creating Widgets

- create_widgets method sets up the various GUI components such as labels, frames, Treeview for displaying transactions, scrollbar, search bar, and search button.
- reeview is configured with columns for Category, Amount, and Date, along with headings.
- A search bar and button are provided for searching transactions.

3.3.2.3 Loading Transactions

- load_transactions method attempts to load transactions from the "FinanceTracker.json" file.

3.3.2.4 Displaying Transactions

- Display_transactions method populates the Treeview with the loaded transactions.
- It provides each transaction, along with its matching category, amount, and date, into the Treeview by iterating through the transactions dictionary.

3.3.2.5 Searching Transactions

- search_transactions method is called when the user clicks the search button.
- Iteratively going over the transactions dictionary, it filters transactions according to date, quantity, or category that fit the search query.

3.3.2.6 Sorting Transactions

- sort_by_column method is called when a column heading is clicked.
- It reads the data out of the Treeview, sorts it according to the chosen column, and then updates the Treeview.
- Clicking on the column heading again reverses the sorting order.

3.3.2.7 Scrolling Transactions

- All transactions can be smoothly scrolled through using the scrollbar, either vertically or horizontally as needed.