UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

INFORMATIC INSTITUTE OF TECHNOLOGY.

4COSC006C Software Development I

Module Code & Module Name : 4COSC006C.2 Software Development I.

Module Leader : Mr. Guhanathan Poravi.

Issue Date : 6th March 2024

Student Details:

Student Name	IIT ID	UOW ID
S.R Walakuluarachchi	20230436	2083527

TABLE OF CONTENT

ACKNOWLEDGMENT

I would like to express my sincere gratitude to Mr. Pooravi Guganathan, and our tutorial lecturer Mr. Lakshan Costa for their exceptional teaching during the programming module. Their dedication, clarity, and passion for the subject made the learning experience enjoyable and insightful.

Lectures guidance played a pivotal role in shaping my understanding of programming concepts and improving my coding skills. I appreciate their commitment to fostering a positive and conducive learning environment.

Thank you, Mr. Pooravi Guganathan, and Mr. Lakshan Costa for being an inspiring and supportive instructor throughout this module.

I would also like to thank my friends who helped me to complete this assignment

01.PROBLEM STATEMENT

The objective of this assignment is to create a Personal Finance Tracker using Python, emphasizing fundamental programming concepts such as lists, loops, functions, input/output operations, and input validation. The application will facilitate basic CRUD (Create, Read, Update, Delete) operations for managing financial transactions without employing dictionaries, instead relying on list manipulations. Data persistence will be achieved through JSON. By undertaking this project, learners will deepen their comprehension of data handling, program architecture, and testing within a real-world scenario, enhancing their proficiency in Python programming.

02.PSEUDOCODE

1. Add Transaction

```
Begin
Add_Transaction()
    try
        Input amount
        Input category
        Loop
            Input transaction type
            If transaction type is Income or Expense, break loop
            Else, print "Invalid Transaction type"
        End Loop
        Loop
            Input date in YYYY-MM-DD format
            If date is valid, break loop
            Else, print "Invalid date"
        End Loop
        Append [amount, category, transaction type, date] to transactions list
        Save transactions to file
        Print "Transaction added successfully."
    except ValueError
        Print "Invalid amount, Please enter a valid amount"
    end try
End
```

2. View Transaction

```
Begin
View_Transactions()
    If transactions list is empty
        Print "No transactions available."
        Exit Procedure
    End If
    Print "Transactions:"
    For each transaction in transactions list with index starting from 1
        Print index, amount, category, transaction type, date
    End For
End
```

3. Update Transaction

```
Begin
Update_Transaction()
    Call View_Transactions()
    try
        Input index of transaction to update
        If index is valid
            Input new amount
            Input new category
            Loop
                Input new transaction type
                If transaction type is Income or Expense, break loop
                Else, print "Invalid transaction type"
            End Loop
```

```
        Loop
            Input new date in YYYY-MM-DD format
            If date is valid, break loop
            Else, print "Invalid date"
        End Loop
        Update transaction at index with new details
        Save transactions to file
        Print "Transaction updated successfully"
    Else
        Print "Invalid index, Please enter a valid index"
  except ValueError
    Print "Invalid amount, Please enter a valid amount"
  end try
End
```

4. Delete Transaction

```
Begin
Delete_Transaction()
  Call View_Transactions()
  try
    Input index of transaction to delete
    If index is valid
      Delete transaction at index
      Save transactions to file
      Print "Transaction deleted successfully"
    Else
      Print "Invalid index, please enter a valid index"
```

```
    except ValueError
        Print "Invalid index. Please enter a valid index"
    end try
End
```

5. Display Summary

```
Begin
Display_Summary()
    Calculate total income as sum of amounts where transaction type is Income
    Calculate total expense as sum of amounts where transaction type is Expense
    Print "Total Income:", total income
    Print "Total Expense:", total expense
    Print "Balance:", total income - total expense
End
```

6. File Operation

```
Begin
Load_Transactions()
    Load transactions from 'transactions.json' file if exists
End Procedure

Procedure Save_Transactions()
    Save transactions to 'transactions.json' file
End
```

03.PYTHON CODE

```
import json
import datetime

# Global list to store transactions
transactions = []

# File handling functions
def load_transactions():
    pass

def save_transactions():
    with open('transactions.json', 'w') as file:
        json.dump(transactions, file)

""" dump() function to convert the Python objects into their respective JSON object,
so it makes it easy to write data to files. See the following table given below """

# Feature implementations
def add_transaction():
    try:
      amount = float(input("Enter amount: "))
      category = input("Enter category: ")
      while True:
          transaction_type = input("Enter type (Income/Expense): ").capitalize()
          if transaction_type in ["Income","Expense"]:
              break
```

```
            else:
                print("Invalid Transaction type")
        while True:
            date = input("Enter date (YYYY-MM-DD): ")
            if len(date)!=10:
                print("Invalid date")
                continue
            year,month,day = date.split("-")
            if len(date)==10 and int(month)<=12 and int(day)<=31:
                break
            else:
                print("Invalid date")
        transactions.append([amount, category, transaction_type, date]) # Creates a new list containing the inputed transaction details
        save_transactions()
        print("Transaction added successfully.")
    except ValueError:
        print("Invalid amount, Please enter a valid amount") # Print this instead of displaying error


def view_transactions():
    if not transactions:
        print("No transactions available.")
        return
    print("Transactions:")
    for index, transaction in enumerate(transactions, start=1): # If there are transactions in the list, the function iterates over each transaction using a for loop.
        print(f"{index}. Amount: {transaction[0]}, Category: {transaction[1]}, Type: {transaction[2]}, Date: {transaction[3]}")
```

```
# index: Each transaction is printed with an index number starting from 1


def update_transaction():
    view_transactions()
    try:
        index=int(input("Enter index of transaction to update: "))-1
        if index>=0 and index<len(transactions):
            new_amount = float(input("Enter new amount: "))
            new_catagory = input("Enter new catagory: ")
            while True:
                new_trans_type = input("Enter new transaction type(Income/Expense): ").capitalize()
                if new_trans_type in ["Income","Expense"]:
                    break
                else:
                    print("Invalid transaction type")
            while True:
                new_date = input("Enter new Date(YYYY-MM-DD): ")
                if len(new_date)!=10:
                    print("Invalid date")
                    continue
                year,month,day = new_date.split("-")
                if len(new_date)==10 and int(month)<=12 and int(day)<=31:
                    break
                else:
                    print("Invalid date")



            transactions[index]=[new_amount,new_catagory,new_trans_type,new_date]
```

```
            save_transactions()
            print("Transaction updated successfully")
        else:
            print("Invalid index, Please enter a valid index")
    except ValueError:
            print("Invalid amount, Please enter a valid amount")


def delete_transaction():
    view_transactions()
    try:
        index=int(input("Enter index of transaction to delete: "))-1
        if index>=0 and index<len(transactions):
            del transactions[index]
            save_transactions()
            print("Transaction deleted successfully")
        else:
            print("Invalid index, please enter a valid index")
    except ValueError:
        print("Invalid index. Please eneter a valid index")


def display_summary():
    total_income = sum(transaction[0] for transaction in transactions if transaction[2] ==
"Income")
    total_expense = sum(transaction[0] for transaction in transactions if transaction[2] ==
"Expense")
    print(f"Total Income: {total_income}")
    print(f"Total Expense: {total_expense}")
    print(f"Balance: {total_income - total_expense}")
```

```
def main_menu():
    load_transactions()  # Load transactions at the start
    while True:
        print("\nPersonal Finance Tracker")
        print("1. Add Transaction")
        print("2. View Transactions")
        print("3. Update Transaction")
        print("4. Delete Transaction")
        print("5. Display Summary")
        print("6. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            add_transaction()
        elif choice == '2':
            view_transactions()
        elif choice == '3':
            update_transaction()
        elif choice == '4':
            delete_transaction()
        elif choice == '5':
            display_summary()
        elif choice == '6':
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please try again.")
```

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

# if you are paid to do this assignment please delete this line of comment
```

04. DESIGNING OF THE CODE

4.1. Storing Transactions

The adding function comprises four components: amount, category, type, and date. These components are stored within a global empty list named transactions[].

4.2. Manipulating Transactions

4.2.1. Adding Transactions

New transactions are added to a smaller list and then appended to the transactions list.

4.2.2. Viewing Transactions

Displays all transactions stored in the transactions list.

4.2.3. Updating Transactions

User initiates an update.

Chooses the attribute (amount, category, type, date) to update.

Based on the user's selection, the chosen attribute is updated.

4.2.4. Deleting a Transaction

User selects a transaction for deletion.

The selected transaction is removed from the main transactions list.

4.2.5. Displaying the Summary

Calculates total income, total expenses, and total profit based on the transaction list.

05. TEST CASES

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	2.0	**Valid Input:** Amount: 1000 Category: Salary Type: Income Date: 2024-03-17	Display “Transaction Added Successfully”	Display “Transaction Added Successfully”	Pass
	2.1	**Invalid Input:** Amount : abc	Display “Invalid amount, Please enter a valid amount”	Display “Invalid amount, Please enter a valid amount”	Pass
	2.2	**Invalid Input:** Type: Test	Display “Invalid Transaction Type”	Display “Invalid Transaction Type”	Pass
View Transactions	3.0	**View transactions when there are no transactions:** Transactions[]	Display “No Transactions Available”	Display “No Transactions Available”	Pass
	3.1	**View transactions when there are existing transactions:** Transactions: [[1000.0, 'Salary', 'Income', '2024-03-17']	Transactions: 1. Amount: 1000.0, Category: Salary, Type: Income, Date: 2024-03-17	Transactions: 1. Amount: 1000.0, Category: Salary, Type: Income, Date: 2024-03-17	Pass

Update Transactions	4.0	**Valid Input:** Update an existing Transaction	Index of transaction to update: 1 New amount: 1200 New category: Rent New transaction type: Expense New date: 2024-03-15	Index of transaction to update: 1 New amount: 1200 New category: Rent New transaction type: Expense New date: 2024-03-15	Pass
	4.1	**Invalid Input: S** Index of transaction to update: 5	Displaying “Invalid index, Please enter a valid index”	Displaying “Invalid index, Please enter a valid index”	Pass
Delete Transaction.	5.0	**Valid input: Delete an existing transaction**. Index of transaction to delete: 1	Delete the selected transaction and Display “Transaction delete successfully.”	Delete the selected transaction and Display “Transaction delete successfully.”	Pass
Display Summary	6.0	**Display summary when there are existing transactions.** Transactions: [[1000.0, 'Salary', 'Income', '2024-03-17'], [500.0, 'Food', 'Expense', '2024-03-16']]	Total Income: 1000.0 Total Expense: 500.0 Balance: 500.0	Total Income: 1000.0 Total Expense: 500.0 Balance: 500.0	Pass
	6.1	**Display summary when there are no transactions.** Transactions: []	Total Income: 0.0 Total Expense: 0.0 Balance: 0.0	Total Income: 0.0 Total Expense: 0.0 Balance: 0.0	Pass

Exit	7.0	Option: 6	Exiting From the program	Exiting From the program	Pass
Save Transactions	8.0	None	When every time adding, updating, or deleting a Transactions. All the changes will be saved in the JSON type file.	When every time adding, updating, or deleting a Transactions. All the changes will be saved in the JSON type file.	Pass
Load Transactions	9.0	None	Display saved transactions when need to view.	Display saved transactions when need to view.	Pass