INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

University of Westminster
Informatics Institute of Technology

Computer Science & Software Engineering

Module: 4COSC006C Software Development I

Module Leader: Mr. Poravi Guganathan

Assessment Type: Individual Coursework

Due Date: 18th March 2024

Student Name: Senuga Rathnayaka

UOW ID: w2083585

IIT ID: 20230478

Tutorial Group: Group O

Table of Content

1.Introduction

- The Personal Finance Tracker is a Python application made to assist people in effectively managing their financial transactions. Users may quickly and conveniently enter their income and expenses, check transaction information, edit, and remove transactions, and get an overview of their financial activity with this program.

- It guarantees the persistent storage of transaction data in a JSON format by utilizing file handling techniques and a modular layout. Errors are graciously handled by exception handling techniques, and human involvement is enabled via a simple command line interface. The application generates and displays transaction summaries, including total revenue, total expenses, and net balance, while data validation assures accuracy. Users may simply track and analyses their financial activity with the help of this program, which empowers them to make well-informed decisions for reaching their financial objectives.

2.Acknowledgement

My deepest gratitude goes out to Mr. Pooravi Guganathan and Mr. Lakshan Costa, our tutorial instructor, for their wonderful instruction during the programming module. Their commitment, lucidity, and enthusiasm for the subject matter rendered the educational process delightful and perceptive.

My comprehension of programming ideas and the development of my coding skills were greatly influenced by the advice provided by the lectures. I value their dedication to providing a supportive and encouraging learning atmosphere.

We are grateful to Mr. Lakshan Costa and Mr. Pooravi Guganathan for being such motivating and encouraging teachers during this module.

I also want to express my gratitude to my buddies for their assistance in finishing this task.

3.Pseudo code

START
IMPORT json and datetime modules.
INITIALIZE empty transactions list.

FUNCTION load_transactions()
IF “data.json” file exists:
OPENFILE 'data.json' in read mode.
READ data from file.
COPY JSON data into a Python list transactions.
FOR each transaction in the transactions list:
GET amount, description, type, and date from the transaction.
CREATE a temporary list containing the extracted data.
APPEND the temporary list to the transactions list.
END FOR
CLOSEFILE
IF there is no file:
OPENFILE 'data.json' in write mode.
WRITE an empty list to the file.
CLOSEFILE

FUNCTION save_transactions()
OPENFILE 'data.json' in write mode.
WRITE transactions list to the file in JSON format.
CLOSEFILE

FUNCTION add_transaction()

```
WHILE True:
  TRY:
    PROMPT user to enter transaction amount
    READ amount from user input
    IF amount <= 0:
      RAISE ValueError("Amount must be a positive number.")
    BREAK
  EXCEPT ValueError:
    DISPLAY error message
    RETRY

PROMPT user to enter transaction description
READ description from user input

WHILE True:
  TRY:
    DISPLAY transaction type options
    PROMPT user to enter transaction type choice
    IF type_choice is not ['1', '2']:
      RAISE ValueError("Invalid choice. Please enter 1 or 2.")
    IF type_choice == '1':
      SET transaction_type to "Income"
    ELSE:
      SET transaction_type to "Expense"
    BREAK
  EXCEPT ValueError:
    DISPLAY error message
    RETRY

WHILE True:
  TRY:
    PROMPT user to enter transaction date (YYYY-MM-DD)
    READ date from user input
    VALIDATE date format using datetime module
    BREAK
  EXCEPT ValueError:
    DISPLAY error message
    RETRY

CREATE transaction list containing [amount, description, transaction_type, date]
APPEND transaction list to transactions list
CALL save_transactions() function and DISPLAY success message
```

FUNCTION view_transactions()

```
IF transactions list is empty:
   DISPLAY "No transactions to display."
ELSE:
   FOR index IN range(length of transactions):
      GET transaction with the index from transactions list
      DISPLAY transaction index
      DISPLAY amount
      DISPLAY description
      DISPLAY type
      DISPLAY date
   END FOR
END IF
```

FUNCTION update_transaction()

```
CALL view_transactions()
IF transactions list is empty:
   DISPLAY "No transactions to update."
   RETURN
END IF

WHILE True:
   TRY:
      PROMPT user to enter the index of the transaction to update (0 to cancel)
      READ index from user input
      IF index == 0:
         DISPLAY "Update canceled."
         RETURN
      IF index < 0 or index >= length of transactions:
         RAISE IndexError("Invalid transaction index.")
      END IF

      WHILE True:
         TRY:
            DISPLAY update menu options
            PROMPT user to enter update choice
            IF choice not in ['1', '2', '3', '4', '5']:
               RAISE ValueError("Invalid choice. Please enter a number between 1 and 5.")
            END IF

            IF choice == '5':
               CALL save_transactions()
               DISPLAY "Updates saved successfully."
```

```
    RETURN
END IF

IF choice == '1':
    DISPLAY current amount of the transaction
    PROMPT user to enter new transaction amount
    READ amount from user input
    IF amount <= 0:
        RAISE ValueError("Amount must be a positive number.")
    END IF
    UPDATE amount of the transaction in the transactions list
END IF

ELIF choice == '2':
    DISPLAY current description of the transaction
    PROMPT user to enter new transaction description
    READ description from user input
    UPDATE description of the transaction in the transactions list
END IF

ELIF choice == '3':
    WHILE True:
        TRY:
            DISPLAY transaction type options
            PROMPT user to enter transaction type choice
            IF type_choice not in ['1', '2']:
                RAISE ValueError("Invalid choice. Please enter 1 or 2.")
            END IF
            UPDATE type of the transaction in the transactions list
            BREAK
        EXCEPT ValueError:
            DISPLAY error message
    END TRY
END IF

ELIF choice == '4':
    DISPLAY current date of the transaction
    WHILE True:
        TRY:
            PROMPT user to enter new transaction date (YYYY-MM-DD)
            READ date from user input
            VALIDATE date format
            BREAK
```

```
                EXCEPT ValueError:
                    DISPLAY "Invalid date format. Please enter date in YYYY-MM-DD format."
                END TRY
                UPDATE date of the transaction in the transactions list
            END IF

        EXCEPT ValueError:
            DISPLAY error message
        EXCEPT IndexError:
            DISPLAY error message
    END WHILE

  EXCEPT ValueError:
    DISPLAY "Invalid input. Please enter a valid number."
  EXCEPT IndexError:
    DISPLAY "Invalid transaction index."
  END TRY
END WHILE
```

FUNCTION delete_transaction()

```
CALL view_transactions()
IF transactions list is empty:
  DISPLAY "No transactions to delete."
  RETURN
END IF

TRY:
  PROMPT user to enter the index of the transaction to delete
  READ index from user input
  CALCULATE actual index by subtracting 1
  DELETE the transaction at the specified index
  CALL save_transactions()
  DISPLAY "Transaction deleted successfully."
EXCEPT IndexError:
  DISPLAY "Invalid transaction index."
EXCEPT ValueError:
  DISPLAY "Invalid input. Please enter a valid number."
END TRY
```

FUNCTION display_summary()

```
SET total_income to 0
SET total_expense to 0

FOR each transaction in transactions:
   IF transaction type is 'Income':
      ADD transaction amount to total_income
   END IF
   IF transaction type is 'Expense':
      ADD transaction amount to total_expense
   END IF
END FOR
DISPLAY Total Income
DISPLAY Total Expense
DISPLAY Net Balance (total_income - total_expense)
```

FUNCTION main_menu()

```
CALL load_transactions()

WHILE True:
   DISPLAY "Personal Finance Tracker"
   DISPLAY "1. Add Transaction"
   DISPLAY "2. View Transactions"
   DISPLAY "3. Update Transaction"
   DISPLAY "4. Delete Transaction"
   DISPLAY "5. Display Summary"
   DISPLAY "6. Exit"
   PROMPT user to enter their choice
   READ choice from user input

   IF choice == '1':
      CALL add_transaction()
   ELSE IF choice == '2':
      CALL view_transactions()
   ELSE IF choice == '3':
      CALL update_transaction()
   ELSE IF choice == '4':
      CALL delete_transaction()
   ELSE IF choice == '5':
      CALL display_summary()
   ELSE IF choice == '6':
      CALL save_transactions()
      DISPLAY "Saving Changes and exiting program!"
```

```
            BREAK
          ELSE:
            DISPLAY "Invalid choice. Please try again."
          END IF
        END WHILE
    END
END
```

4.Python code

```
import json
from datetime import datetime

# Global list to store transactions
transactions = []

# File handling functions
def load_transactions():
    try:
        with open("Transactions.json", "r") as file:
            transactions.extend(json.load(file))
    except FileNotFoundError:
        print("Transactions file not found. Starting with an empty transaction list.")
    except json.JSONDecodeError:
        print("Error decoding JSON. Starting with an empty transaction list.")

def save_transactions():
    with open("Transactions.json", "w") as file:
        file.write("[")
        file.write("\n")
        for transaction in transactions:
            file.write("\t")
            json.dump(transaction, file)
            file.write("\n")
        file.write("]")

# Validation functions
def is_valid_date(date_str):
    try:
        datetime.strptime(date_str, "%Y-%m-%d")
        return True
    except ValueError:
        return False

# Feature implementations
def add_transaction():
    while True:
        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")
            date = input("Enter date (YYYY-MM-DD): ")
            if not is_valid_date(date):
                raise ValueError("Invalid date format. Please enter date in YYYY-MM-DD format.")
            transactions.append([amount, category, transaction_type, date])
            save_transactions()
            print("Transaction added successfully")
            break
        except ValueError as e:
            print(e)

def view_transactions():
    if not transactions:
        print("No transactions available")
    else:
        for transaction in transactions:
            print(transaction)

def update_transaction():
    view_transactions()
    try:
        index = int(input("Enter index of transaction to update: "))
        if 0 <= index < len(transactions):
            new_amount = float(input("Enter new amount: "))
            new_category = input("Enter new category: ")
            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")
            new_date = input("Enter new Date (YYYY-MM-DD): ")
            if not is_valid_date(new_date):
                raise ValueError("Invalid date format. Please enter date in YYYY-MM-DD format.")
            transactions[index-1] = [new_amount, new_category, new_trans_type, new_date]
            save_transactions()
            print("Transaction updated successfully")
        else:
```

```
            print("Invalid index. Please enter a valid index")
    except ValueError as e:
        print(e)

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

def display_summary():
    total_income = 0
    total_expense = 0
    if not transactions:
        print("No transactions record yet")
        return
    for sublist in transactions:
        if sublist[2] == "Income":
            total_income += sublist[0]
        elif sublist[2] == "Expense":
            total_expense += sublist[0]
    total_balance = total_income - total_expense
    print(f"Total Income: {total_income}")
    print(f"Total Expense: {total_expense}")
    print(f"Total Balance: {total_balance}")

# Load transactions at the start
def main_menu():
    load_transactions()
    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.")

# Start the program execution
main_menu()
# if you are paid to do this assignment please delete this line of comment.
```

5.Test cases

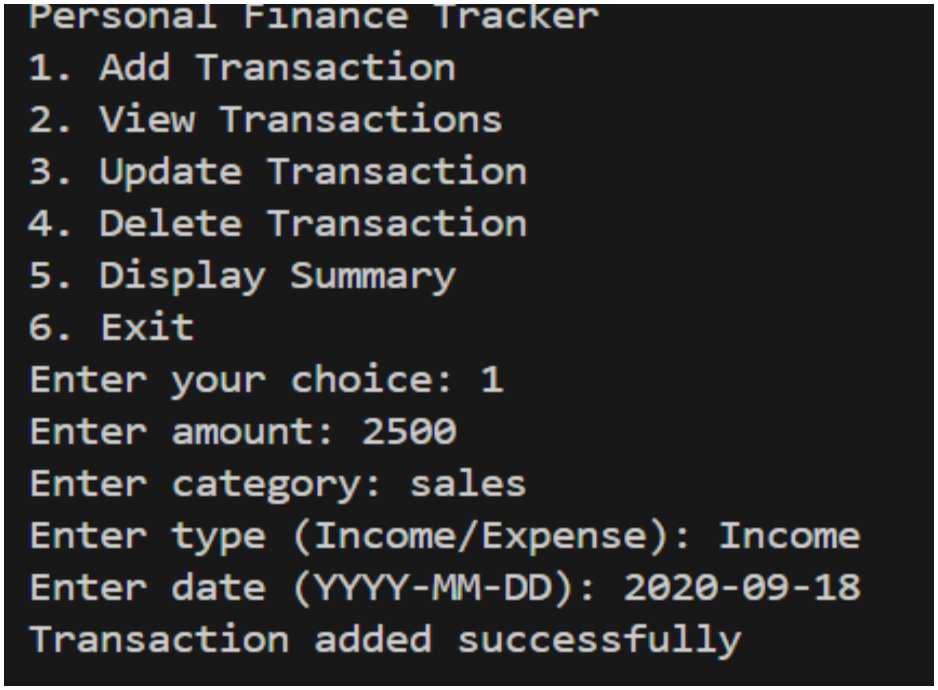
Test Component	Test No	Test Input	Expected Result	Actual Result	Pass / Fail
Main Menu	1	None	Displaying the main menu with options and asking choice.	Displaying the main menu with options and asking choice.	Pass
Add Transactions	2.0	**Valid Input:** Amount: 2500 Category: Sales Type: Income Date: 2020-09-18	Display “Transaction Added Successfully”	Display “Transaction Added Successfully”	Pass
	2.1	**Invalid Input:** Amount : ad	Display “Invalid amount, Please enter a valid amount”	could not convert string to float	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: [[2500.0, 'Sales', 'Income', '2020-09-18']	Transactions: 1. Amount: 2500.0, Category: Sales, Type: Income, Date: 2020-09-18	Transactions: 1. Amount: 2500.0, Category: Sales, Type: Income, Date: 2020-09-18	Pass

Update Transactions	4.0	**Valid Input:** Update an existing Transaction	Index of transaction to update: 1 New amount: 5000 New category: Raw transaction type: Expense New date: 2021-05-04	Index of transaction to update: 1 New amount: 5000 New category: Raw transaction type: Expense New date: 2021-05-04	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: [[7500.0, 'ted', 'Income', '2020-08-01'], [2500.0, 'wet', 'Expense', '2019-05-03']]	Total Income: 7500.0 Total Expense: 2500.0 Balance: 5000.0	Total Income: 7500.055 Total Expense: 2500.0 Balance: 5000.0	Pass
	6.1	**Display summary when there are no transactions.** Transactions: []	Total Income: 0.0 Total Expense: 0.0 Balance: 0.0	No transactions record yet	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

6.Screenshots

- Adding a transaction



- Viewing a transaction

```
Personal Finance Tracker
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Delete Transaction
5. Display Summary
6. Exit
Enter your choice: 2
[7500.0, 'ted', 'Income', '2020-08-01']
[2500.0, 'wet', 'Expense', '2019-05-03']
```

- Updating a transaction

```
Personal Finance Tracker
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Delete Transaction
5. Display Summary
6. Exit
Enter your choice: 3
[2500.0, 'sales', 'Income', '2020-09-18']
Enter index of transaction to update: 1
Enter new amount: 5000
Enter new category: raw
Enter new transaction type (Income/Expense): expense
Enter new Date (YYYY-MM-DD): 2021-05-04
Transaction updated successfully
```

- Deleting a transaction

```
Personal Finance Tracker
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Delete Transaction
5. Display Summary
6. Exit
Enter your choice: 4
[5000.0, 'raw', 'Expense', '2021-05-04']
Enter index of transaction to delete: 1
Transaction deleted successfully
```

- Display summary

```
Personal Finance Tracker
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Delete Transaction
5. Display Summary
6. Exit
Enter your choice: 5
Total Income: 7500.0
Total Expense: 2500.0
Total Balance: 5000.0
```

```
Personal Finance Tracker
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Delete Transaction
5. Display Summary
6. Exit
Enter your choice: 2
[7500.0, 'ted', 'Income', '2020-08-01']
[2500.0, 'wet', 'Expense', '2019-05-03']
```

- Exiting from the transaction

```
Personal Finance Tracker
1. Add Transaction
2. View Transactions
3. Update Transaction
4. Delete Transaction
5. Display Summary
6. Exit
Enter your choice: 6
Exiting program.
PS C:\Users\Insight>
```