A logo for a computer company

Description automatically generated with medium confidenceA black text on a white background

Description automatically generated

**TEST PLAN SUMMARY**

STUDENT NAME : S.GOKULA NANDHAN

STUDENT ID : 20233027

UOW ID :20820910

**ASSIGNMENT RELATED DETAILS**

Course :Level 04(SE/CS) 2024 January Intake

Module :4COS006C0.2 Software Development

Assignment No :02

Assignment type :Individual

Issued date : 30th March 2024

Submission due date :on or before 15th April 2024

Module leader :Mr.Guhanathan Poravi

**STUDENT DETAILS**

Student Name : S.Gokula Nandhan

Student ID : 20233027

UOW number : 20820910

Course :B.Eng(Hons) Software Engineering

Email address :gokula.20233027@gmail.com

**CONTENTS**

1.Abstract

2.Acknowledgement

3.Specification of the task

-Task overview

-Task executing steps

4.Detailed steps of tasks

-Analyse the task

-Design the programme

-Implimentation and documentation

-Test

**ABSTRACT**

Mr. Guhanathan Poravi, the module leader, has assigned the job of creating a Personal Finance Tracker utilising the 4COSC006C0.2 Software Development module. This report acts as thorough documentation for the project, which entails using the basic of Python programming and JSON serialisation to create a Dictionary-based application. The problem statement is examined in the documentation, and a workable solution is suggested to address it. Internal testing processes have also been carried out, and the report contains thorough test cases and validations to guarantee the application's accuracy and resilience.

**ACKNOWLEDGEMENT**

My heartfelt appreciation goes out to everyone who contributed to the creation of this assignment report for the application of Personal Financial Tracker.Above all, I would want to express my sincere gratitude to the module leader, Sir ,Mr. Guhanathan Poravi, and his team for providing me an great opportunity to work on this assignment. They provided guidance, inspiration, and constant support throughout the entire assignment, which allowed it to conclude successfully.   
In addition, I would like to express my gratitude to my peers and colleagues who actively contributed constructive criticism and perceptive remarks during the review process. The report's quality and accuracy were enhanced by their perceptive criticisms and suggestions.   
Though I have tried my best to express my gratitude to everyone of the contributors.

Thankyou

**SPECIFICATION OF THE TASK**

**TASK OVERVIEW**

The goal of this assignment is to use Python to create a Personal Finance Tracker while concentrating on core programming concepts including input/output, functions, loops, dictionaries, and input validation.The application will now manage financial transactions using dictionaries rather than lists, as the original specification used lists to represent different types of expenses and values being lists of expenses.The motive of this update intends to improve comprehension of programme design, testing, and data handling in a real-world setting, with an emphasis on file input/output for large-scale data processing.

**TASK EXECUTING STEPS**

**1.Analyse the task**

**2.Design the programme**

**3.Implimentation and documentation**

**4.Test**

**DETAILED STEPS OF THE TASK**

**1.ANALYSE THE TASK**

These are the key factors to design this application.

1. Apply essential Python programming constructs.
2. Manage data using dictionary manipulations for more efficient data retrieval.
3. Implement CRUD operations on financial transactions, leveraging JSON serialization for data storage.
4. Introduce file operations for bulk reading of transactions from a text file, enhancing the program's usability for larger data sets.

5. Conduct thorough testing and validation to ensure program robustness.

**2.DESIGN THE PROGRAMME**

After analyzing above key factors I have designed a documentation plan with pseudocode.

Psuedocode as follows

**Psuedocode for the programme**

Start

IMPORT json

INITIALIZE transactions={}

FUNCTION load\_transactions()

INITIALIZE global transactions

OPENFILE ” transactions.json” for read

INITIALIZE a variable “load\_file” to

serialize python list into JSON data

CLOSEFILE

DISPLAY Transaction loaded successfully!

ENDFUNCTION

FUNCTION save\_transactions()

OPENFILE ”transactions.json” for write

INITIALIZE a variable “save\_file” to

Diserilize the transactions into python

CLOSEFILE

DISPLAY Transactions saved successfully!

ENDFUNCTION

FUNCTION read\_bulk\_transactions\_from\_file()

OPENFILE “exepense.txt” as expense for read

INITIALIZE a variable “text\_file” for expense

IF transaction\_type in transactions THEN

Append transactions[transaction\_type] with values

ELSE

GET transactions[transaction\_type]=Amount,date

ENDIF

END FUNCTION

FUNCTION add\_transactions()

Display Enter Transaction Details!

GET transaction\_type

DO WHILE true

GET amount

IF amount>=0 THEN

BREAK

ELSE

DISPLAY Negative value does not get for an amount

ENDIF

GET date

IF transaction\_type not in transactions THEN

Transactions[transaction\_type] is empty list

ENDIF

APPEND amount and date to transactions

PRINT Transaction added successfully!

ENDFUNCTION

FUNCTION view\_transactions()

IF transactions=[] THEN

DISPLAY No more transactions to view!

ELSE

FOR transaction\_type in transactions

DISPLAY transaction\_type

FOR sub\_transaction\_dictionary in transactions[transaction\_type]

DISPLAY(sub\_trasaction\_dictionary)

ENDIF

ENDFUNCTION

FUNCTION update\_transaction()

IF transactions=[] THEN

DISPLAY No more transactions to update!

ELSE

DISPLAY check Transaction type and index you want to update

FUNCTION view\_update()

ENDFUNCTION

DISPLAY Now you can update!

GET transaction\_type

IF transaction\_type in transactions THEN

DOWHILE true

GET new\_amount

IF new\_amount>=0 THEN

BREAK

ELSE

DISPLAY Negative value does not get for an amount!

GET new\_date

ADD new\_amount,new\_date to the global dictionary

DISPLAY Transaction updated successfully!

ELSE

DISPLAY Transaction not found!

ENDIF

END FUNCTION

FUNCTION delete\_transaction()

IF transactions=[] THEN

DISPLAY No more transactions to delete!

ELSE

DISPLAY Check transaction type and index you want to delete

FUNCTION view\_transactions()

END FUNCTION

DISPLAY now you can delete it!

GET transaction\_type

IF transaction\_type in transactions THEN

DOWHILE true

GET index

IF 0<=index<len(transactions[transaction\_type]) THEN

DELETE transaction mentioned index

DISPLAY Transaction deleted successfully!

FUNCTION save\_transactions()

BREAK

ELSE

DISPLAY Invalid index.Please try again!

ENDIF

ELSE

DISPLAY Type of expense not found!

ENDIF

END FUNCTION

FUNCTION display\_summary()

total\_expense=0

FOR transaction\_type in transactions DO

FOR sub\_transaction\_dictionary in transactions[transaction\_type]

total\_expense+=sub\_transaction\_dictionary[Amount]

DISPLAY your total expense is “total\_expense”

END FUNCION

FUNCTION main\_menu()

load\_transactions()

read\_bulk\_transactions\_from\_file()

DOWHILE 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 for choice

Get choice

IF choice =1 THEN

add\_transaction()

ELSE IF choice=2 THEN

view\_transactions()

ELSE IF choice=3 THEN

update\_transaction()

ELSE IF choice=4 THEN

delete\_transaction()

ELSE IF choice=5 THEN

display\_summary()

ELSE IF choice=6 THEN

save\_transactions()

Display -EXIT-

break

ELSE

Display Invalid choice.Please try again!

ENDIF

ENDIF

ENDIF

ENDIF  
 ENDIF

ENDIF

ENDIF

ENDWHILE

ENDFUNCTION

main\_menu()

END

**3.IMPLIMENTATION AND DOCUMENTATION**

After the designing phase the following code would be executed

**PYTHON CODE**

import json

#Global dictionary to store transactions

transactions={}

'''File handling functions'''

#function to load transactions from a json file

def load\_transactions():

global transactions

load\_file=open("transactions.json","r")

transactions=json.load(load\_file)

load\_file.close()

print("Transactions loaded successfully!")

#function to save transactions to a json file

def save\_transactions():

save\_file=open("transactions.json","w")

json.dump(transactions,save\_file)

save\_file.close()

print("Transactions saved successfully!")

#read transactions from a text file and add them to global dictionary

def read\_bulk\_transactions\_from\_file(expense="expense.txt"):

try:

text\_file=open(expense,"r")

for values in text\_file:

transaction\_type,amount,date=values.strip().split(",")

if transaction\_type in transactions:

transactions[transaction\_type].append({"Amount":float(amount,2),"Date":date})

else:

transactions[transaction\_type]=[{"Amount":float(amount,2),"Date":date}]

except FileNotFoundError:

print("File not found!")

'''Feature implementations'''

#add a new transaction to global dictionary

def add\_transaction():

print("\nEnter Transaction Details!")

transaction\_type=input("\nEnter the type of expense :")

while True:

try:

amount=float(input("\nEnter the amount :"))

if amount>=0:

break

else:

print("Negative value does not get for an amount!")

except ValueError as amount\_error:

print(amount\_error)

date=input("\nEnter the date(YYYY-MM-DD) :")

if transaction\_type not in transactions:

transactions[transaction\_type]=[]

transactions[transaction\_type].append({"Amount":amount,"Date":date})

print("\nTransaction added successfully!")

save\_transactions()

#view all transactions stored in global dictionary

def view\_transactions():

if not transactions:#empty global dictionary

print("No more transactions to view!")

else:

for transaction\_type in transactions:

print(transaction\_type)

for sub\_transaction\_dictionary in transactions[transaction\_type]:

print(sub\_transaction\_dictionary)

#update an existing transaction in the transactions(global dictionary)

def update\_transaction():

if not transactions:

print("No more transactions to update!")

else:

print("check Transaction type and index you want to update!")

view\_transactions()

print("\nNow you can update it!")

transaction\_type=input("\nEnter type of your expense :")

if transaction\_type in transactions:

while True:

try:

index=int(input("Enter index from purticular type of expense :"))

if 0 <= index <len(transactions[transaction\_type]):

break

else:

print("Invalid index.Please try again!")

except ValueError as error:

print(error)

while True:

try:

new\_amount=float(input("Enter new amount :"))

if new\_amount >= 0:

break

else:

print("Negative value does not get for an amount!")

except ValueError as new\_amount\_error:

print(new\_amount\_error)

new\_date=input("Enter new date(YYYY-MM-DD) :")

#will update new date and amount in nested dictionaries

transactions[transaction\_type][index]["Amount"]=new\_amount

transactions[transaction\_type][index]["Date"]=new\_date

print("Transaction updated successfully!")

save\_transactions()

else:

print("Transaction not found!")

#delete an transaction from global dictionary

def delete\_transaction():

if not transactions:

print("No more transactions to delete!")

else:

print("check Transaction type and index you want to delete!")

view\_transactions()

print("\nNow you can delete it!")

transaction\_type=input("\nEnter type of your expense :")

if transaction\_type in transactions:

while True:

try:

index = int(input("Enter index from purticular type of expense :"))

if 0 <= index < len(transactions[transaction\_type]):

del transactions[transaction\_type][index]

print("Transaction deleted successfully!")

save\_transactions()

break

else:

print("Invalid index.Please try again!")

except ValueError as error:

print(error)

else:

print("Type of expense not found!")

#function for display summary of total expenses

def display\_summary():

total\_expense=0

for transaction\_type in transactions:

for sub\_transaction\_dictionary in transactions[transaction\_type]:

total\_expense+=sub\_transaction\_dictionary["Amount"]

print("Your total expense is",total\_expense)

#user interactional function

def main\_menu():

#call functions to load transactions from file

load\_transactions()

read\_bulk\_transactions\_from\_file()

#main programme loop

while True:

print("\n---Personal Finance Tracker---")

print("\n1 - Add Transaction")

print("2 - View Transactions")

print("3 - Update Transaction")

print("4 - Delete Transaction")

print("5 - Display Summary")

print("6 - Exit")

try:

#prompt user for call the feature implimentation functions

choice=int(input("\nEnter your choice:"))

if 1<=choice<=6:

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()

else:

choice==6

save\_transactions()

print("-EXIT-")

break

else:

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

except ValueError as error:

print(error)

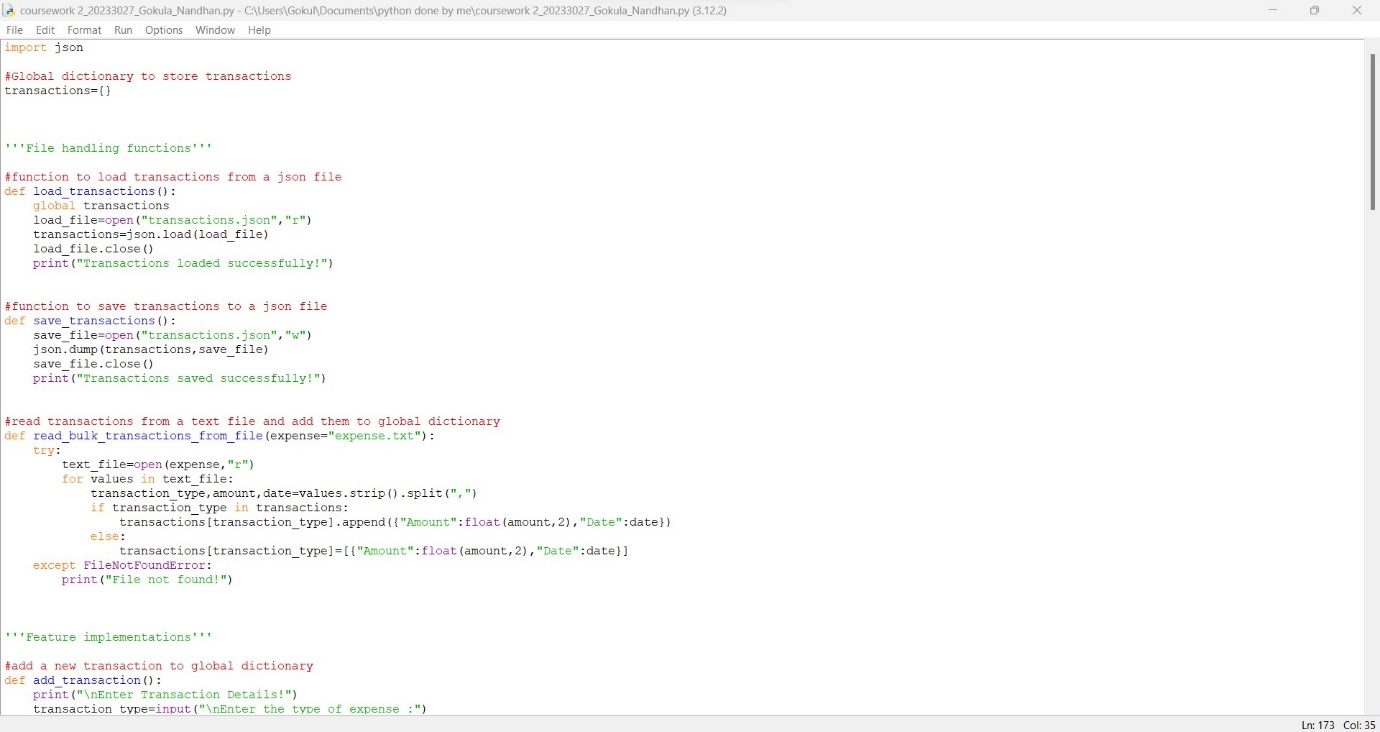
#programme start and calling for main menu function

if \_\_name\_\_=="\_\_main\_\_":

main\_menu()

**SCREEN SHOTS OF EXECUTED CODE**

**PAGE 1**

****

**A screenshot of a computer

Description automatically generated PAGE 2**

**A screenshot of a computer

Description automatically generated PAGE 3**

**A screenshot of a computer

Description automatically generatedPAGE 4**

**PAGE 5**

**A screenshot of a computer

Description automatically generated**

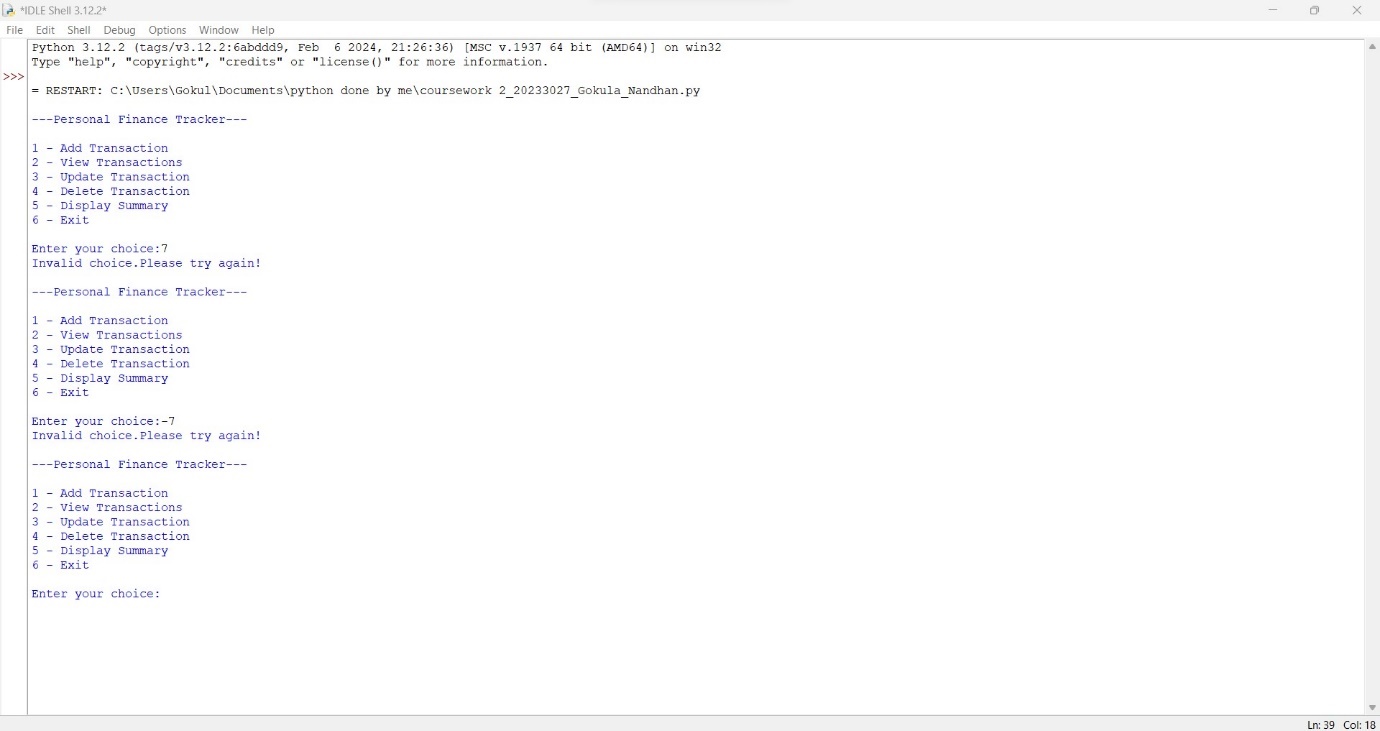
**4.TEST**

THE FOLLOWING CHARTS SHOWS THE ENTIRE TEST PHASE OF THE EXECUTED CODE

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test case** | **Discription** | **Inputs** | **Expected**  **Output** | **Actual**  **Output** | **Remarks** |
| 01 | Choice random  (without a range 1<=choice<=6) | Input a choice=7  Choice=-7 | Display  Invalid choice.Plese try again | Did expected output | Test pass |
| 02 | Choice random  (without string data type(string,float,boolean)) | Choice=ant  Choice=True  Choice=3.5 | invalid literal for int() | Did expected output | Test pass |
| 03 | Choice=1  (add transactions()) | Choice=1 | Collect the data and display “Transaction added successfully!” | Did expected output | Test pass |
| 04 | Choice=1  (add\_transaction()) | Input negative value for amount | Display error message | Did expected output | Test pass |
| 05 | Choice=2  (view\_transactions()) | Choice=2  After some added transactions | Display transactions | Did expected output | Test pass |
| 06 | Choice=3  (update\_transaction()) | Choice=3 and typing all the needs from application properly | Save all data and show the message | Did expected output | Test pass |
| 07 | Choice=3  (update transaction()) | input irrilevant key in dictionary | Display error message | Did expected output | Test pass |
| 08 | Choice=3  (update transaction()) | Input invalid index for nested dictionary | Display error message | Showed expected output | Test pass |
| 09 | Choice=3  (update transaction()) | Input negative value for new amount | Display error message | Showed  Expected output | Test pass |
| 10 | Choice=4  (delete transaction()) | Input valid data for positive output | Save relevant data and output success message | Showed expected output | Test pass |
| 11 | Choice=4  (delete transaction()) | Input invalid index for nested dictionary | Display error message | Showed expected output | Test pass |
| 12 | Choice=4  (delete transaction()) | input irrilevant key in dictionary | Display error message | Showed expected output | Test pass |
| 13 | Choice=5  (display\_summary()) | Choice=5 | Display the sum of all expenses | Showed expected output | Test pass |
| 14 | Choice=6  (exit()) | Choice=6 | Save and exit | Showed expected output | Test pass |

**SCREEN SHOTS OF TEST SUMMARY**

**TEST CASE 01**

****

**TEST CASE NO 2**

**A screenshot of a computer

Description automatically generated**

**TEST CASE NO 3A white rectangular frame with a black border

Description automatically generated**

**A white rectangular frame with a black border

Description automatically generated**

**TEST CASE NO 4**

**A screenshot of a computer

Description automatically generated**

**TEST CASE NO 5**

**A white screen with blue text

Description automatically generated**

**A white screen with a black text

Description automatically generated with medium confidence**

**A person standing in front of a white board

Description automatically generated**

**TEST CASE NO 6**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**TEST CASE NO 7**

**A computer screen shot of a computer screen

Description automatically generated**

**TEST CASE NO 8**

****

**TEST CASE NO 9**

**A close-up of a computer screen

Description automatically generated**

**TEST CASE NO 10**

**A close-up of a computer screen

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**TEST CASE NO 11**

**A screenshot of a computer

Description automatically generated**

**TEST CASE NO 12**

**A screenshot of a computer

Description automatically generated**

**TEST CASE NO 13**

**A screenshot of a computer

Description automatically generated**

**TEST CASE NO 14**

**A screenshot of a computer

Description automatically generated**