

University of Westminster

4COSCOO6C

Software Development

Documentation Report and Test Plan Report.

Coursework Details :

Module Leader : Mr. Guhanathan Poravi

Assignment Number : 1

Assignment Type : Individual

Issue Date : 6th March 2024 Submission

Deadline : on or before 18th of March 2024

Student Details :

Student Name : Ravichandran Shavinkar

Student ID : 20233155

UOW number : w2083094

# 1 . Abstract

The report that Mr. Guhanathan Poravi assigned for the 4COSC006C Software Development module contains documentation for a Python personal finance tracker that is list-based and uses JSON serialization. The documentation investigates the problem statement and uses the Python programming language to come up with a solution. Test cases are used for internal testing, and the report also includes the validations.

# 2 . Acknowledgement

I would like to express my sincere gratitude and advice over the course of this project to Mr. Salman Faraj. His knowledge, support, and criticism were crucial to its eventual completion. I sincerely appreciate his assistance and inquiring. I also want to acknowledge the assistance and encouragement received from my friends and colleagues who helped with this project, whether directly or indirectly. This project would not have been possible without your combined efforts, and I am sincerely appreciative of the chance to collaborate with such gifted people. I appreciate your contributions, everyone.

# Table of figures

[**Figure 1 coding** 24](#_Toc162550775)

# Tables

[**Table 1 test execution** 31](#_Toc162550837)

# Task 1- Pseudo Code

BEGIN

#Global list

Initialize transactions to []

FUNCTION add\_transactions ()

DISPLAY “Enter Transaction Details”

GET amount from user as input.

GET category from user as input.

GET type (“Income or Expense”) from user as input.

GET date(“DD-MM-YYYY”) from user as input.

Set transaction to [amount, category, type, date]

Add transaction to transactions

DISPLAY “Transactions Added Successfully”

ENDFUNCTION

FUNCTION view\_transactions ()

IF transactions list is empty THEN

PRINT “No transactions to view”

ELSE

DISPLAY “Transactions are listed below”

FOR view\_ transaction in transactions DO

DISPLAY “Amount:” , view\_ transaction [0]

DISPLAY “Category:” , view\_ transaction [1]

DISPLAY “Type:” , view\_ transaction [2]

DISPLAY “Date:” , view\_ transaction [3]

NEXT transaction

ENDIF

ENDFUNCTON

FUNCTION update\_transaction ()

IF transactions list is empty THEN

PRINT “No transactions were found to update”

ELSE

PRINT “Enter transactions index to update”

GET index from user input

IF index is valid THEN

PRINT “Enter new transactions details”

GET correct amount from user input

GET correct category from user input

GET correct type (“Income or Expense”) from user input

GET correct date (“DD-MM-YYYY”) from user input

Set transactions[index] to [correct amount, correct category, correct type, correct date]

save\_transactions ()

DISPLAY “Transaction Updated Successfully”

ELSE

PRINT “Invalid index to process”

ENDIF

ENDIF

ENDFUNCTION

FUNCTION delete\_transaction ()

IF transactions list is empty THEN

DISPLAY “Transactions list is empty to be deleted”

ELSE

Prompt for index of transactions to delete

GET index from user input

IF index is valid THEN

Delete transactions[index]

save\_transactions ()

PRINT “Transaction deleted successfully”

ELSE

PRINT “Invalid index to process”

ENDIF

ENDIF

ENDFUNCTION

FUNCTION display\_summary ()

Set total\_income to 0

Set total\_expense to 0

FOR transaction in transactions DO

IF transaction[2] == “ Income” THEN

total\_income = total\_income + transaction[0]

ELSE

total\_expense = total\_expense + transaction[0]

ENDIF

NEXT transaction

IF total\_ income > total\_expense THEN

Compute Net\_income = total\_income - total\_expense

ELSE

Compute Net\_expense = total\_ expense - total\_ income

ENDIF

PRINT total\_income

PRINT total\_ expense

IF total\_ income > total\_expense THEN

PRINT Net\_income

ELSE

PRINT Net\_expense

ENDIF

ENDFUNCTION

FUNCTION save\_transactions ()

OPENFILE “transactions.json” for WRITE as F1

Serialize the transactions list to JSON

WRITE the serialized JSON data to F1

CLOSEFILE F1

DISPLAY “Successfully saved”

ENDFUNCTION

FUNCTION load\_transactions ()

OPENFILE “transactions.json” for READ as F1

Deserialize the transactions list to python

CLOSEFILE F1

DISPLAY “Successful”

ENDFUNCTION

FUNCTION main\_menu ()

load\_transactions ()

WHILE True DO

PRINT “Personal Finance Tracker”

PRINT “1. Add Transaction”

PRINT “2. View Transaction”

PRINT “3. Update Transaction”

PRINT “4. Delete Transaction”

PRINT “5. Display Summary”

PRINT “6. Exit”

Prompt for choice as input

GET choice from user input

IF choice == 1 THEN

add\_transactions ()

ELSE IF choice == 2 THEN

view\_transactions ()

ELSE IF choice == 3 THEN

update\_transactions ()

ELSE IF choice == 4 THEN

delete\_transactions ()

ELSE IF choice == 5 THEN

display\_summary ()

ELSE IF choice == 6 THEN

PRINT “Exiting the program and saving….”

ELSE

PRINT “Invalid choice. Please try again”

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDIF

ENDWHILE

ENDFUNCTION

main\_menu ()

END

# Task 2 - Solution

## 2.1- A brief synopsis

-With the use of a JSON file for transaction storage, this Python application tracks personal finances. A summary of incomes and expenses is also displayed, and there are options for adding, viewing, editing and removing transactions. A command-line interface made available by the main\_menu () function allows users to interact with the tracker.

## 2.2 - Python Code

import json

transactions = [] #Global list to store transactions.

#File handling functions

def load\_transactions():

try:

file = open("transactions.json" , "r+") #Opening the file in read mode.

transactions = json.load (file) #loads the data to the file

file.close () #Closing the opened file

except FileNotFoundError:

print("File 'transactions.json' is unavailable")

return

def save\_transactions(): #Function used to save the processed transactions in json file.

file = open("transactions.json","w") #Opening the json file in write mode.

json.dump(transactions,file) #Stores the transactions data in json file.

file.close() #Closing the opened file

return

#Feature implementations

def add\_transaction(): #permits user to add a new transaction

while True:

print("Enter the Transactions Details")

while True: #looping until user inputs a valid number for amount

try:

Amount = int(input("Enter the amount :"))

if Amount >= 0: #checks if the amount is a whole number

break #exits the loop when the user inputs a valid amount

else:

print("Amount should be a whole number")

except ValueError: #finds for valuerror if amount is non-integer

print("Amount is always an integral value")

Category = input("Enter a category :")

while True:

Type = input("Enter the type(\"Income or Expense\") :")#User should input either Income or Expense

if Type != "Income" and Type != "Expense": #checks if input type is neither Income and Expense

print("Type should be either \"Income\" or \"Expense\". Please try again!")

else:

break

Date = input("Enter the date in (YYYY-MM-DD) :") #User should input date in YYYY-MM-DD format

transaction =[Amount, Category, Type, Date] #Stores the added transaction to the transaction list

transactions.append(transaction) #Combines the transaction list with empty transactions list

print("Transactions Added Successfully.")

break

return

def view\_transactions(): #permits user to view the added transactions

if transactions == []: #checks whether the transactions list is empty

print("No transactions to view")

else:

print("\nTransactions are listed below\n")

for view\_transaction in transactions: #used to iterate over the transactions list and access values using index

print("Amount :",view\_transaction[0] , end=' , ') #end is used print amount,category,type and date on the same line

print("Category :",view\_transaction[1] , end=' , ')

print("Type :",view\_transaction[2] , end=' , ')

print("Date :",view\_transaction[3])

return

def update\_transaction(): #Defining the function to update a transaction

try:

if transactions == []: #checks whether the transactions list is empty

print("No transactions were found to update")

else:

while True: #looping until user inputs a valid number for index

try:

index = int(input("Enter the specific index to update:"))

if index < 0 or index >= len(transactions): #checks for index whether it lies between 0 and (len(transactions)-1)

print("\nInvalid index to update.\n")

continue #continue to next loop when invalid index is entered

else:

break #stops the loop when user enters a valid index

except:

print("\nInvalid input. Index is an integral value.\n")

continue #continue to next loop when non\_integral index is entered

while True: #looping until user inputs a valid number for amount

try:

correctAmount = int(input("Enter the correct amount :"))

if correctAmount >= 0: #checks if the amount is a whole number

break #exits the loop when the user inputs a valid amount

else:

print("Amount should be a whole number")

except ValueError: #finds for valuerror if amount is non-integer

print("Amount is always an integral value")

correctCategory = input("Enter a correct category :")

while True:

correctType = input("Enter the correct type(\"Income or Expense\") :")#User should input either Income or Expense

if correctType != "Income" and correctType != "Expense": #checks if input type is neither Income and Expense

print("Type should be either \"Income\" or \"Expense\". Please try again!")

else:

break #exits the loop when "Income" or "Expense" is entered

correctDate = input("Enter the correct date(YYYY-MM-DD) :")

transactions[index] = [correctAmount, correctCategory, correctType, correctDate] #updates the corrected details in the transactions list at the specified index

save\_transactions() #function is called to save the updated transaction to the file

print("Transaction updated successfully")

except:

print("Invalid index to update")

return

def delete\_transaction(): #allows user to delete an existing transaction

if transactions == []: #checks whether the transactions list is empty

print("Transactions list is empty to be deleted")

else:

while True: #looping until user inputs a valid number for index

try:

index = int(input("Enter the specific index to delete :")) #prompts the user to input the specified index to be deleted

if index < 0 or index >= len(transactions): #checks for index whether it lies between 0 and (len(transactions)-1)

print("\nInvalid index to delete.\n")

continue #continue to next loop when invalid index is entered

else:

break #stops the loop when user enters a valid index

except:

print("\nInvalid input. Index is an integral value.\n")

continue #continue to next loop when non\_integral index is entered

del transactions[index] #deletes the transaction at the specified index of the transactions list

save\_transactions() #save the transactions to the file after deleting

print("Transaction deleted successfully")

return

def display\_summary(): #displays the summary of transaction details

total\_income = 0

total\_expense = 0

for transaction in transactions: #iterates each transaction from transactions list

if transaction[2] == "Income": #checks the type at the index 2 whether its Income or Expense

total\_income = total\_income + transaction[0] #adds the amount at the index 0 to total\_income

else:

total\_expense = total\_expense + transaction[0] #adds the amount at the index 0 to total\_expense

if total\_income > total\_expense:

Net\_income = total\_income - total\_expense

else:

Net\_expense = total\_expense - total\_income

print("Total income:",total\_income)

print("Total expense:",total\_expense)

if total\_income > total\_expense:

print("Net income",Net\_income)

else:

print("Net expense",Net\_expense)

return

def main\_menu(): #starting code of the program

try:

load\_transactions() #Load transactions at the start. If the file doesn't exist then goes to exception block

except:

save\_transactions()

load\_transactions()

while True: #enters an endless loop to prompt input an option from the user

try:

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 = int(input("Enter your choice: ")) #converts the variable 'choice' from string to integer

if 1 <= choice <= 6: #checks if the choice is inside the range

pass #if it lies within the range then goes to the next part of the loop

else:

print("Enter a number between 1 and 6")

except ValueError:

print("Enter an input that is an integer")

continue #continues to next iteration when user inputs a non-integral value

if choice == 1: #checks if the choice equal to 1

add\_transaction() #calls the add\_transaction() function adds a transaction

elif choice == 2:

view\_transactions() #at choice '2 'calls the view\_transaction() function views a transaction

elif choice == 3:

update\_transaction() #at choice '3' calls the update\_transaction() function updates a transaction

elif choice == 4:

delete\_transaction() #at choice '4' calls the delete\_transaction() function deletes a transaction

elif choice == 5:

display\_summary()#at choice '5' calls the display\_summary() function shows the income or expense

elif choice == 6:

save\_transactions() #at choice '6' calls the save\_transactions() function to save and exit the program

print("Transaction saved successfully") #Displays the message after storing the data successfully.

print("Exiting program.")

break

else:

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

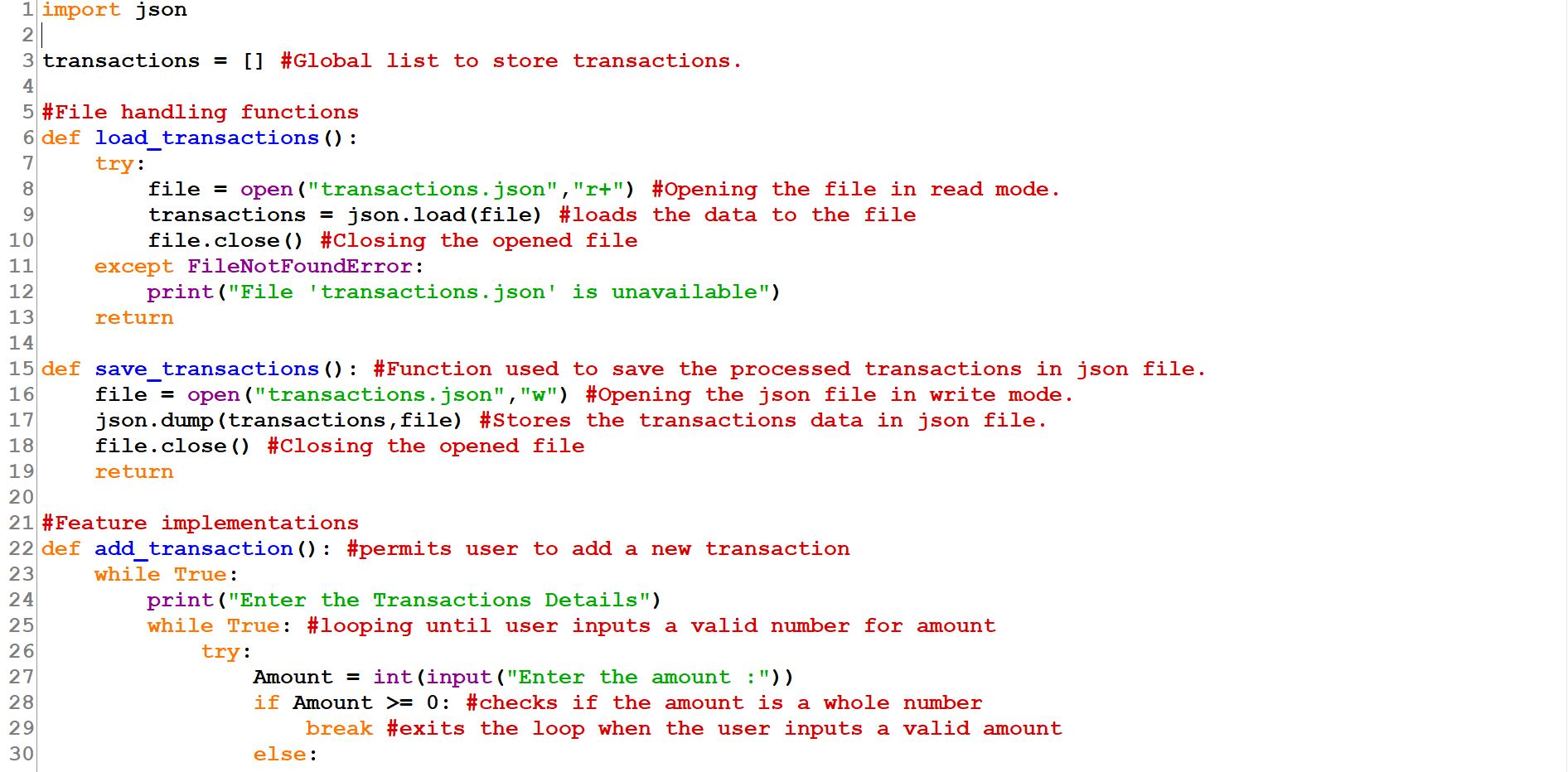
return

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

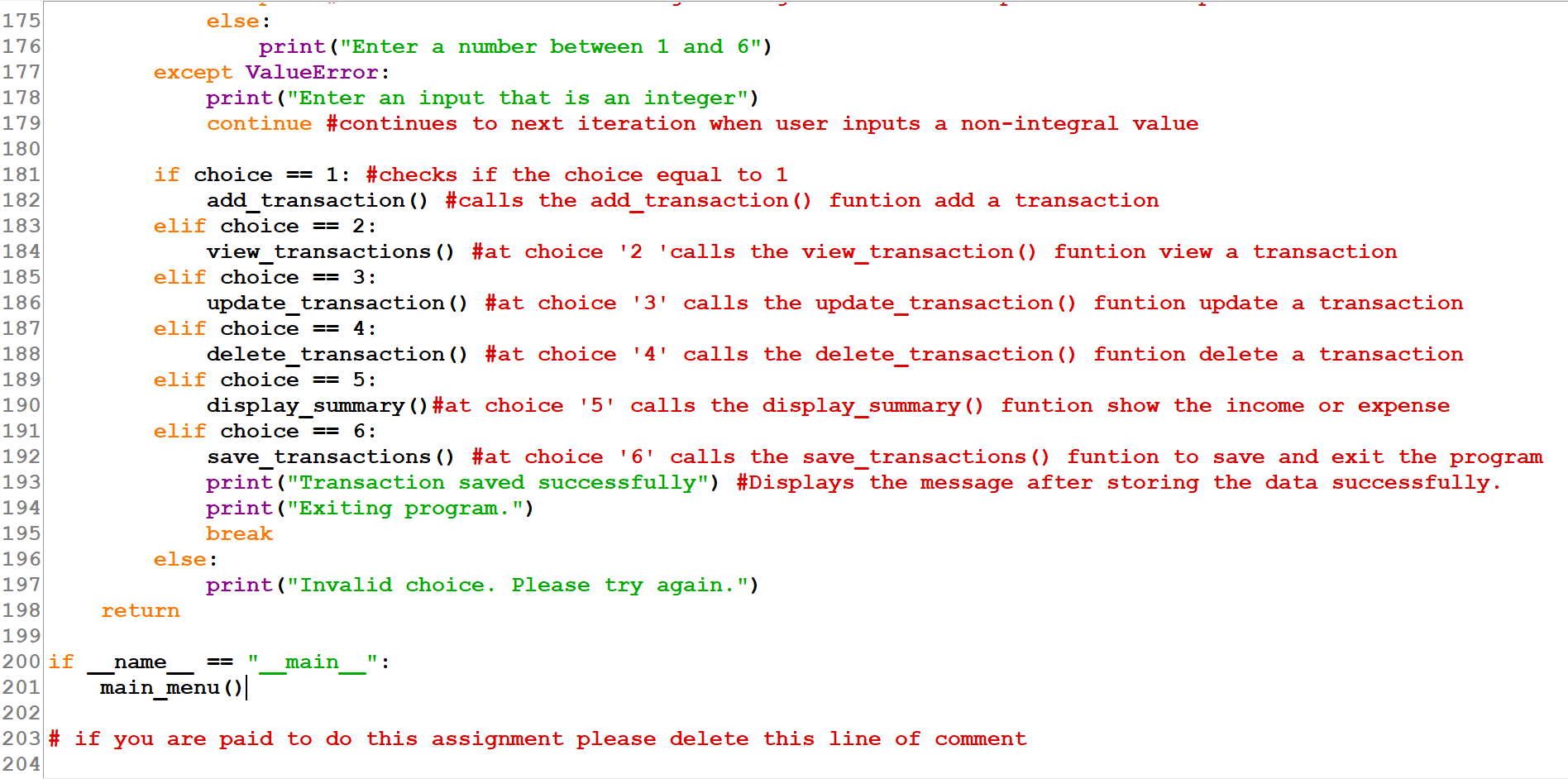
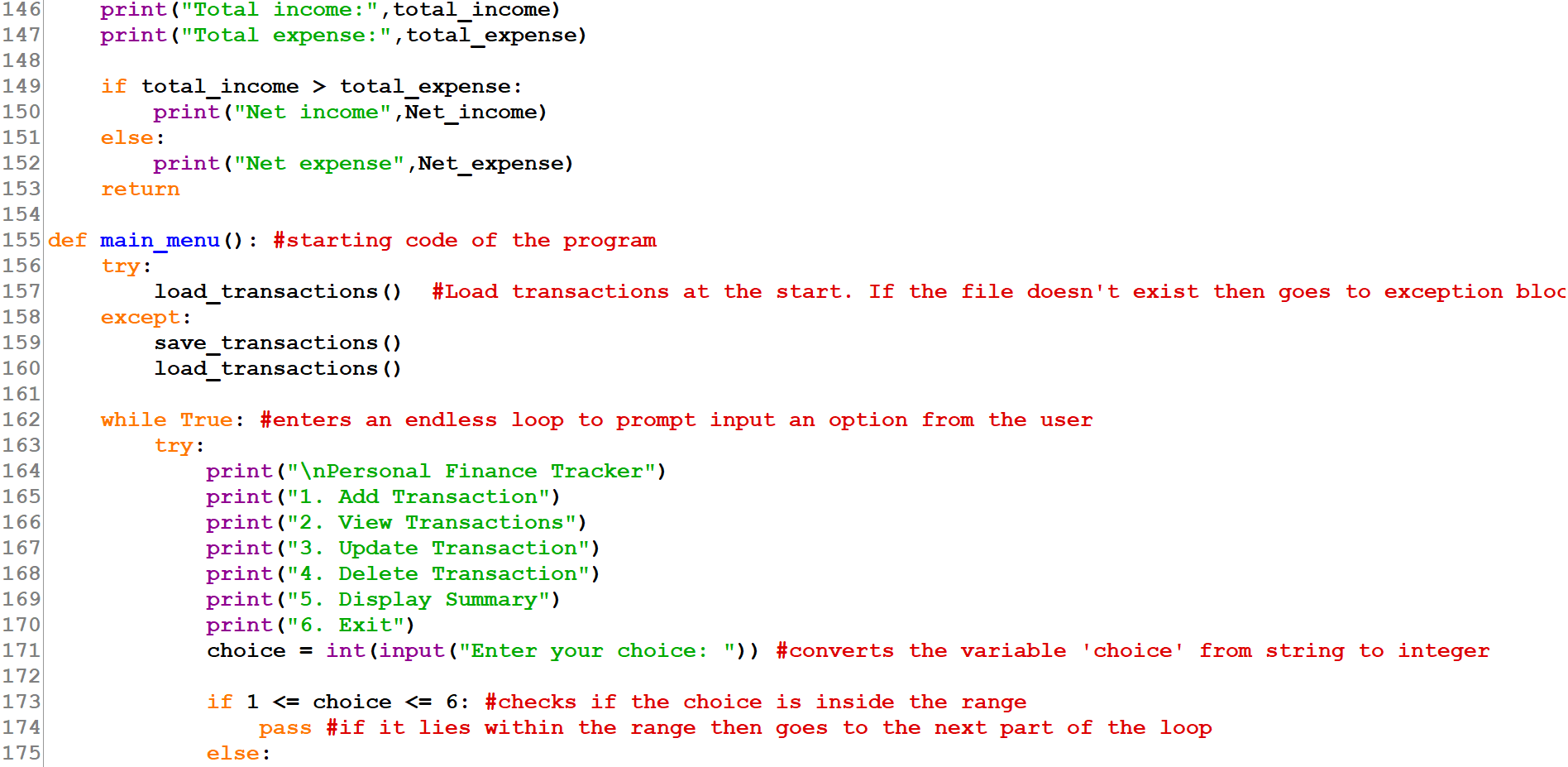
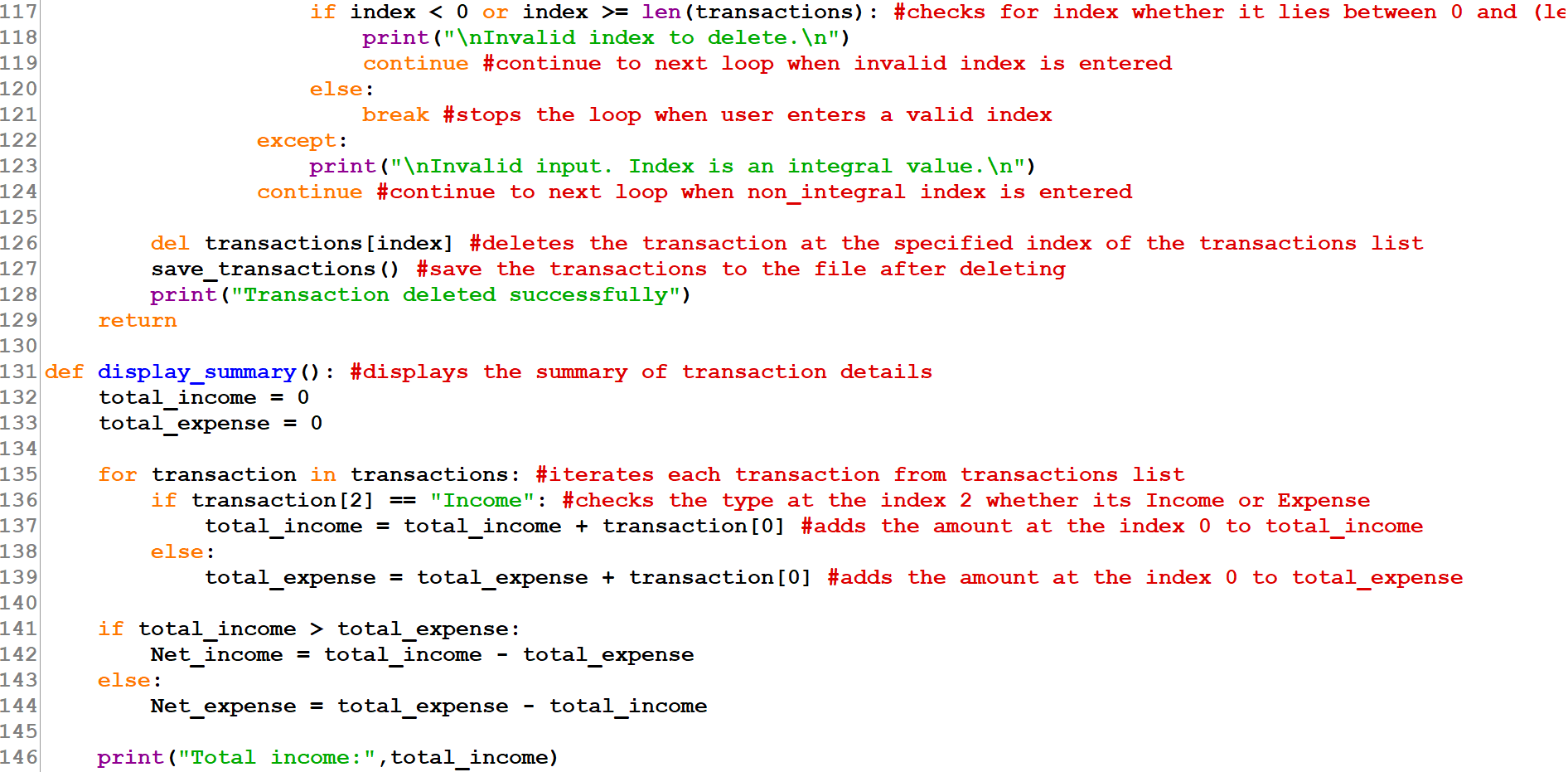
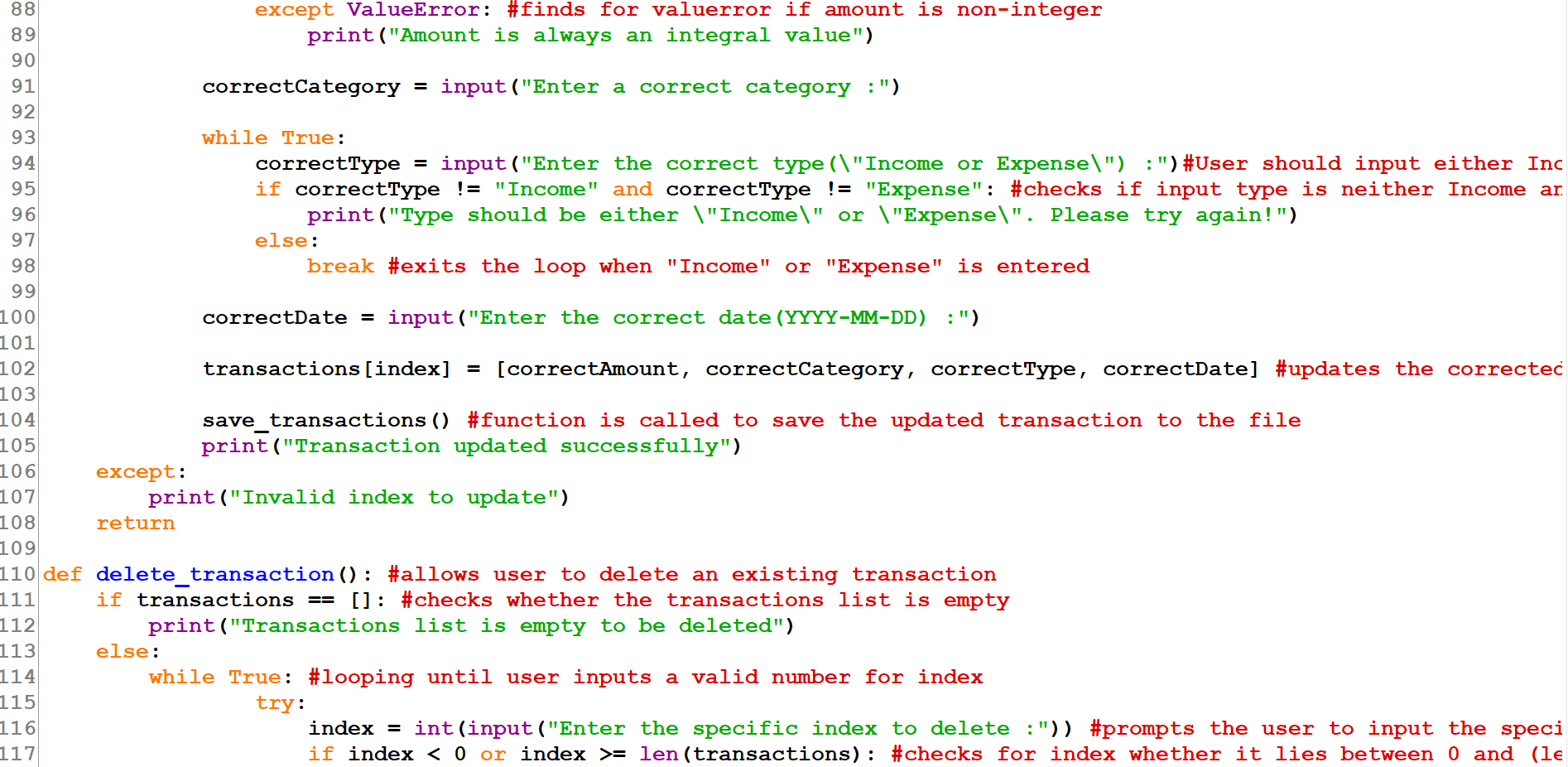
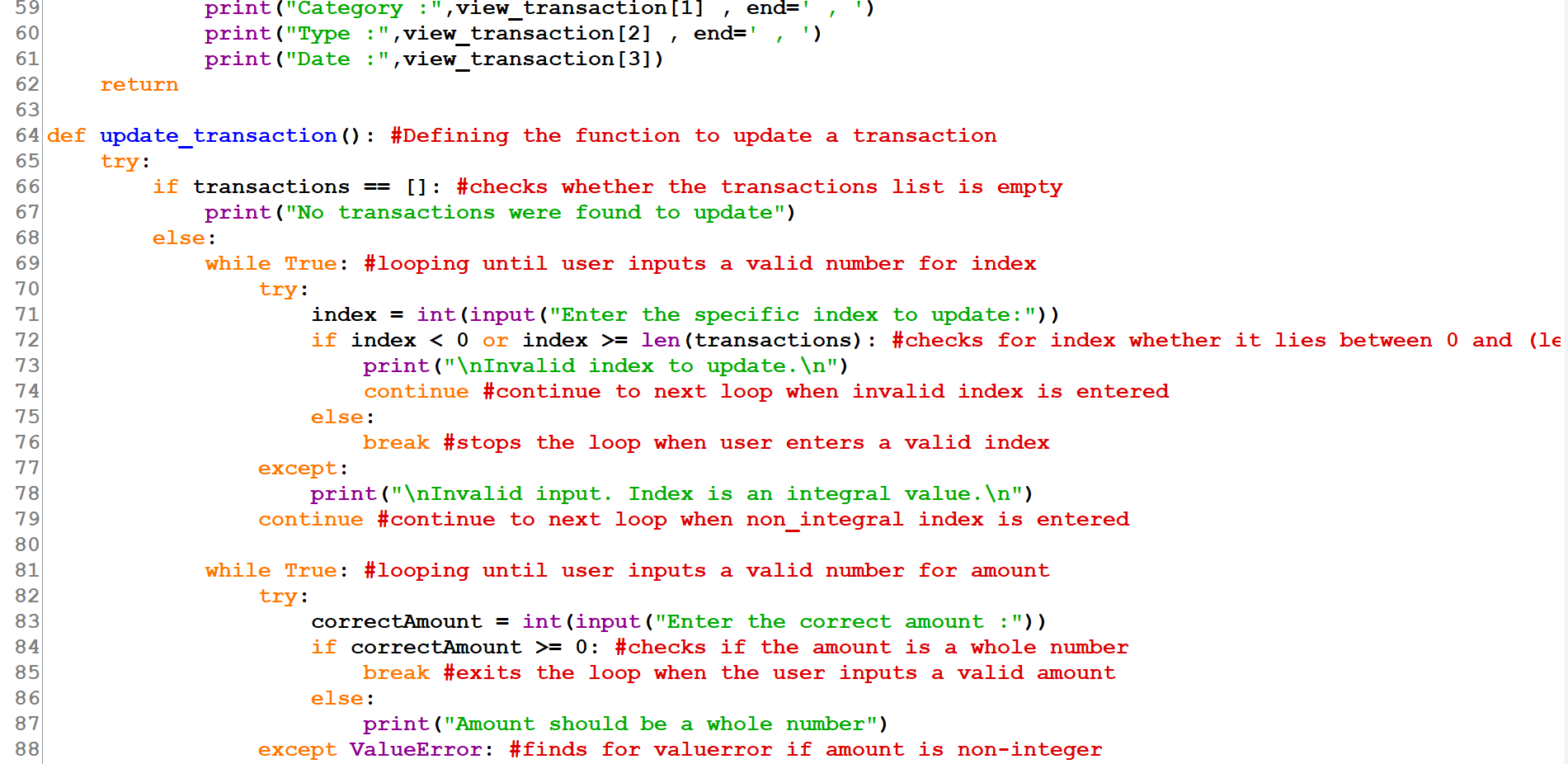
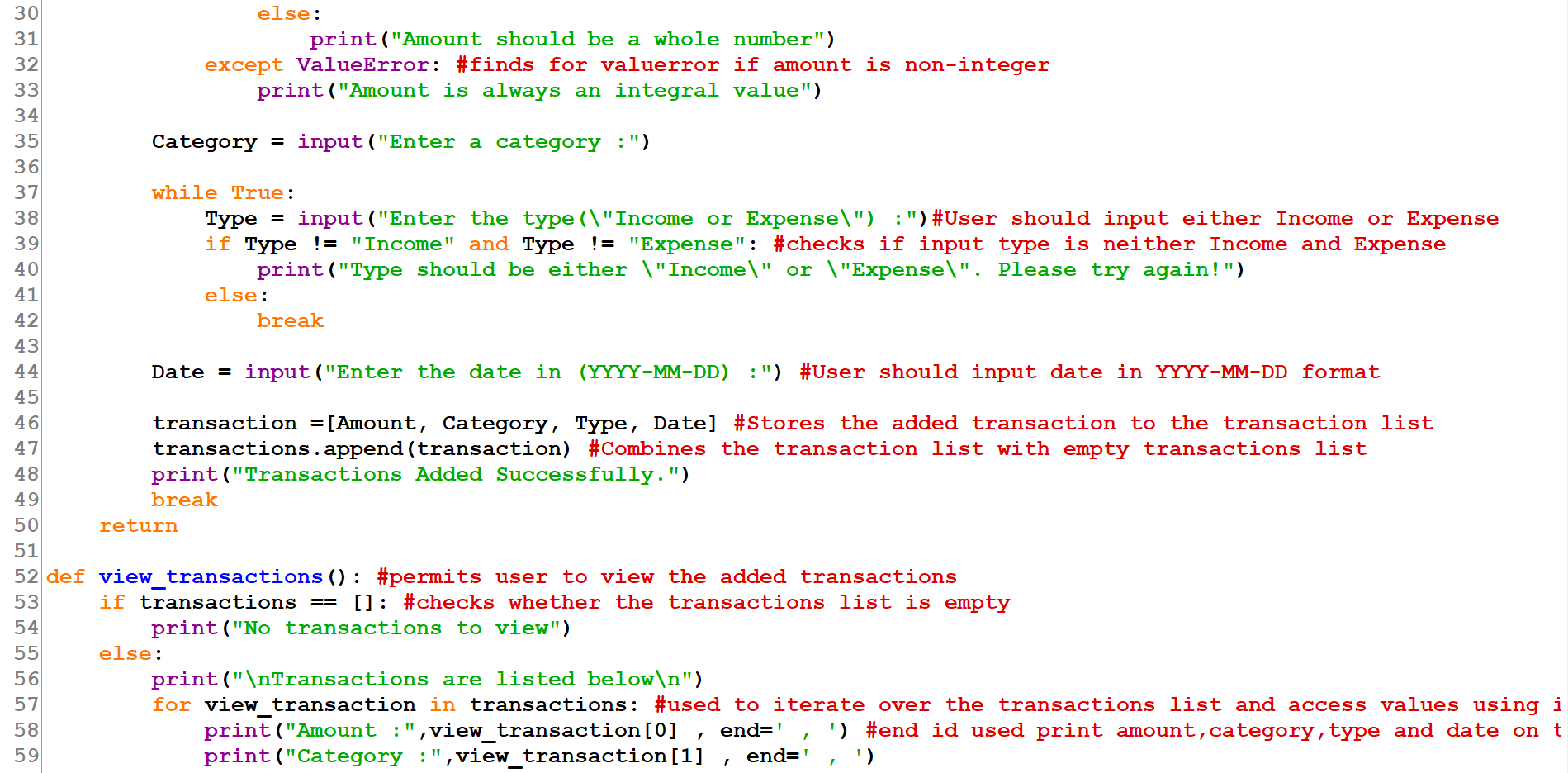
main\_menu()

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

## 2.3 - Python Code Statements



**Figure 1 coding**



# Task 3 - Test Plan and Execution

## 3.1 - Test Plan

There are 8 functions defined to implement pseudo code

File handling functions

1. def load\_transactions()

2. def save\_transactions():

Feature implementations

3.def add\_transaction():

4.def view\_transactions():

5.def update\_transaction():

6.def delete\_transaction():

7.def display\_summary():

8.def main\_menu():

The code comprises a main function called def main\_menu(). This function includes all other feature implementations. File handling functions are contained in feature implements.   
Thus, I've decided to set up six test cases.

Inside the main function which is main\_menu() it has six choices to run the test plan. See below code

if choice == 1: #checks if the choice equal to 1

add\_transaction() #calls the add\_transaction() funtion adds a transaction

elif choice == 2:

view\_transactions() #at choice '2 'calls the view\_transaction() funtion views a transaction

elif choice == 3:

update\_transaction() #at choice '3' calls the update\_transaction() funtion updates a transaction

elif choice == 4:

delete\_transaction() #at choice '4' calls the delete\_transaction() funtion deletes a transaction

elif choice == 5:

display\_summary()#at choice '5' calls the display\_summary() funtion shows the income or expense

elif choice == 6:

save\_transactions() #at choice '6' calls the save\_transactions() funtion to save and exit the program

print("Transaction saved successfully") #Displays the message after storing the data successfully.

print("Exiting program.")

break

else:

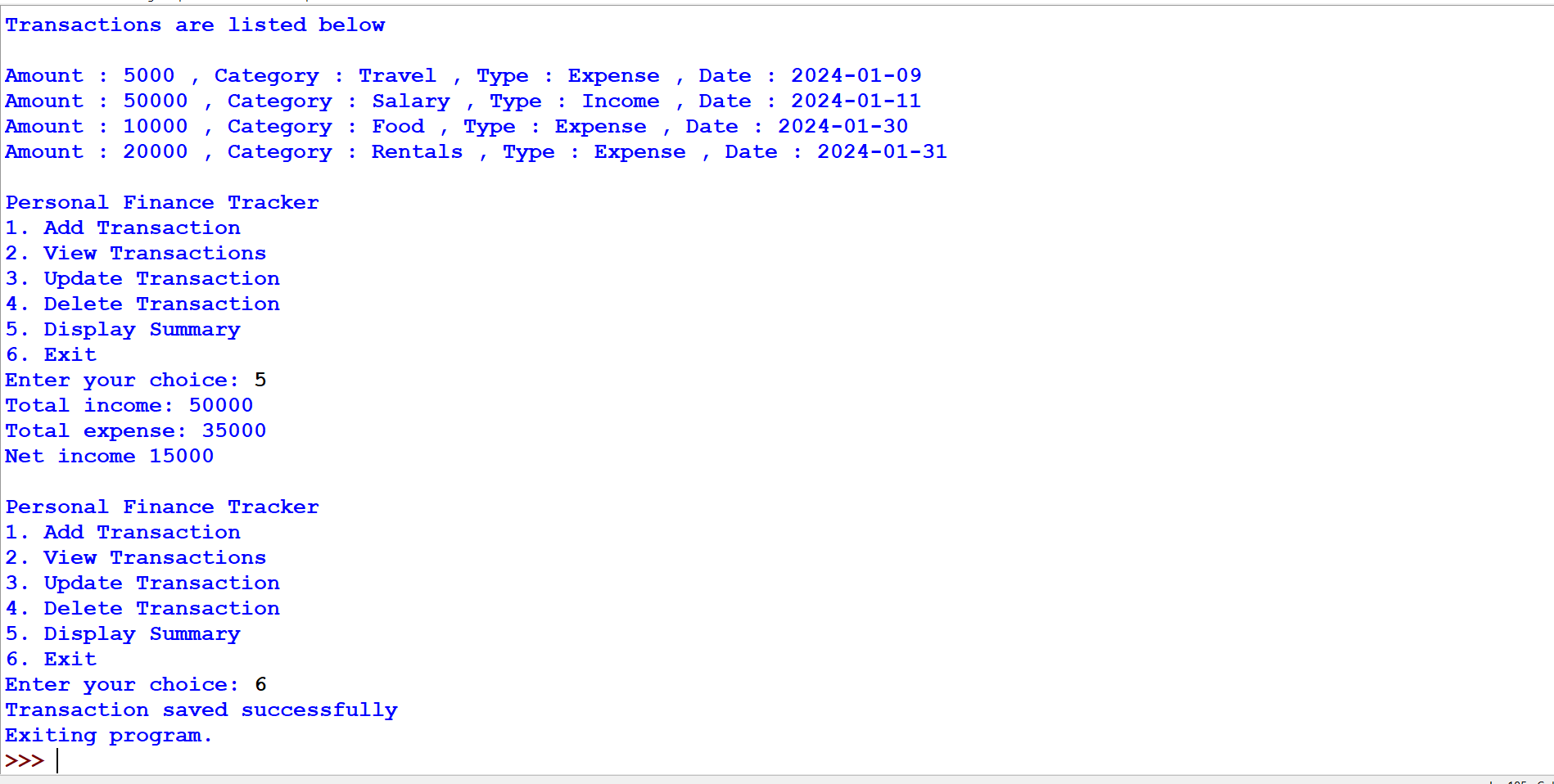
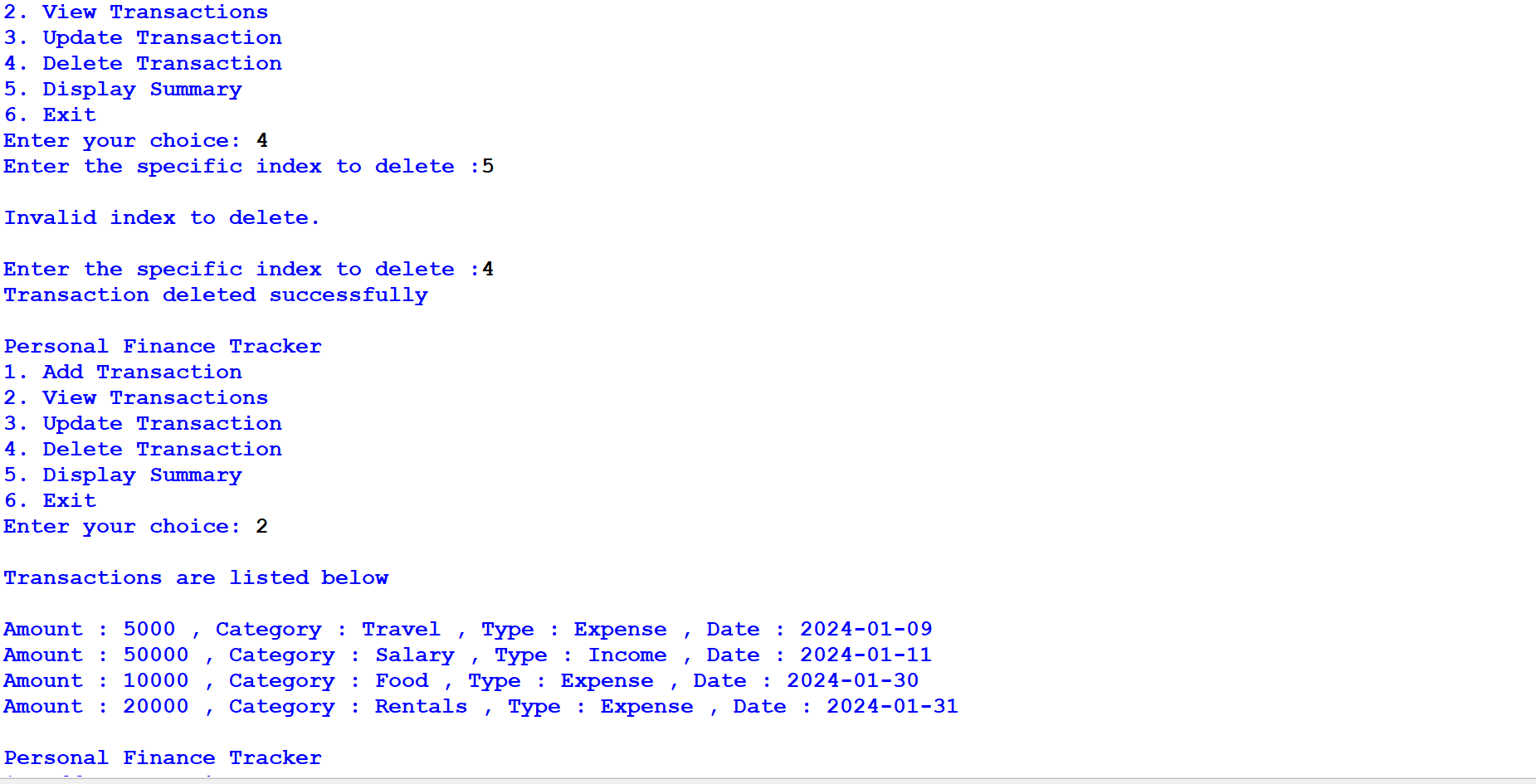
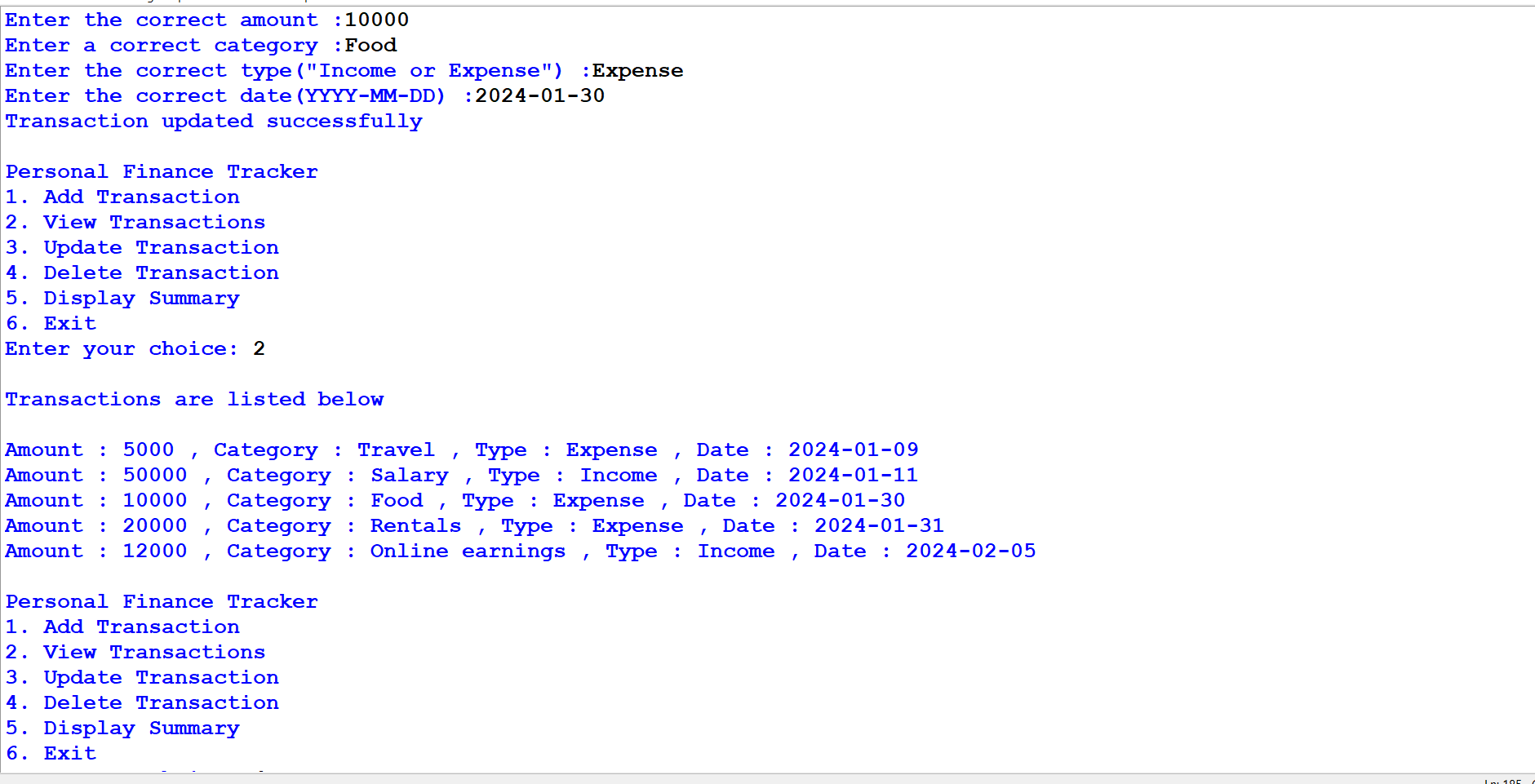
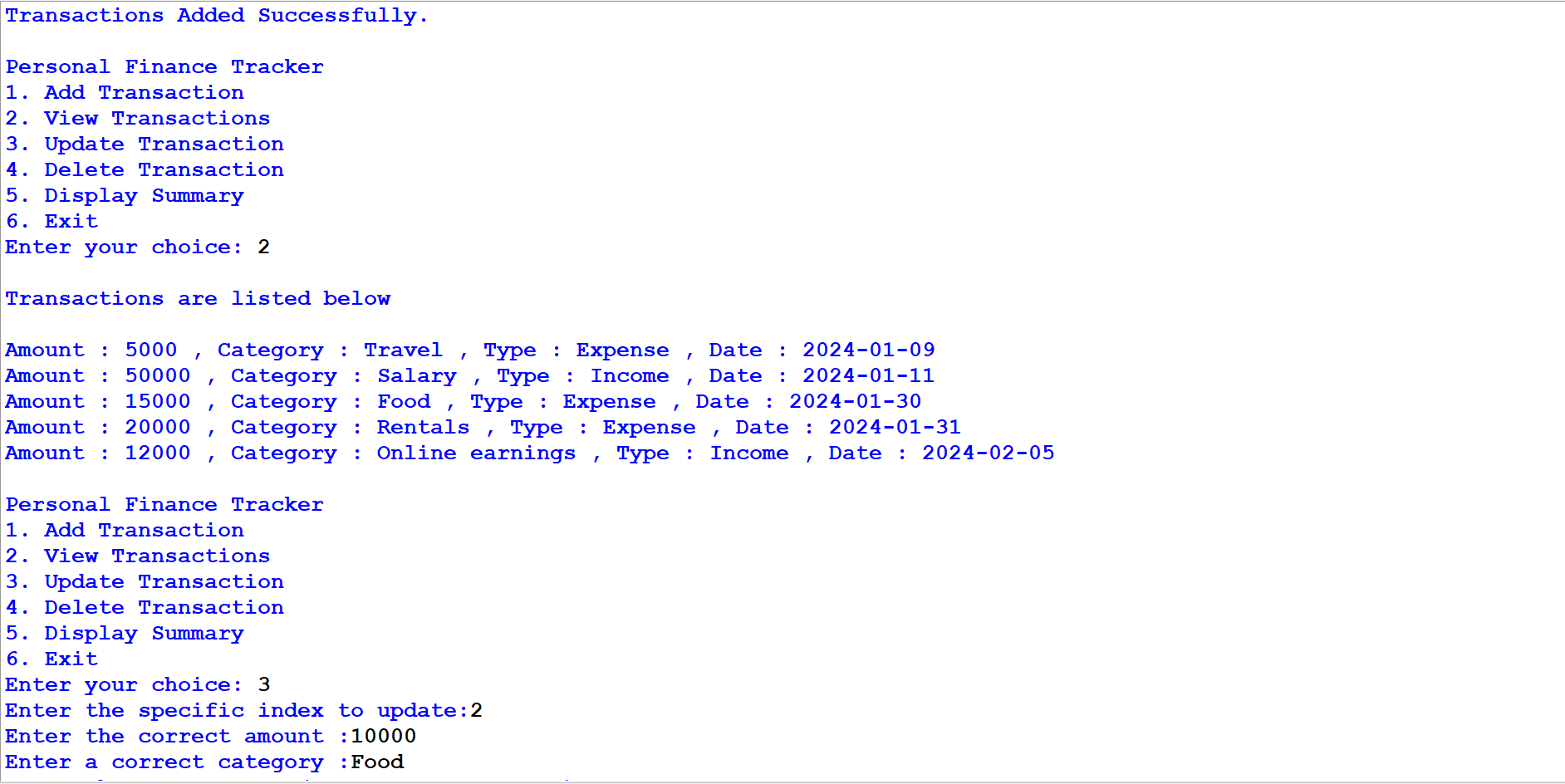
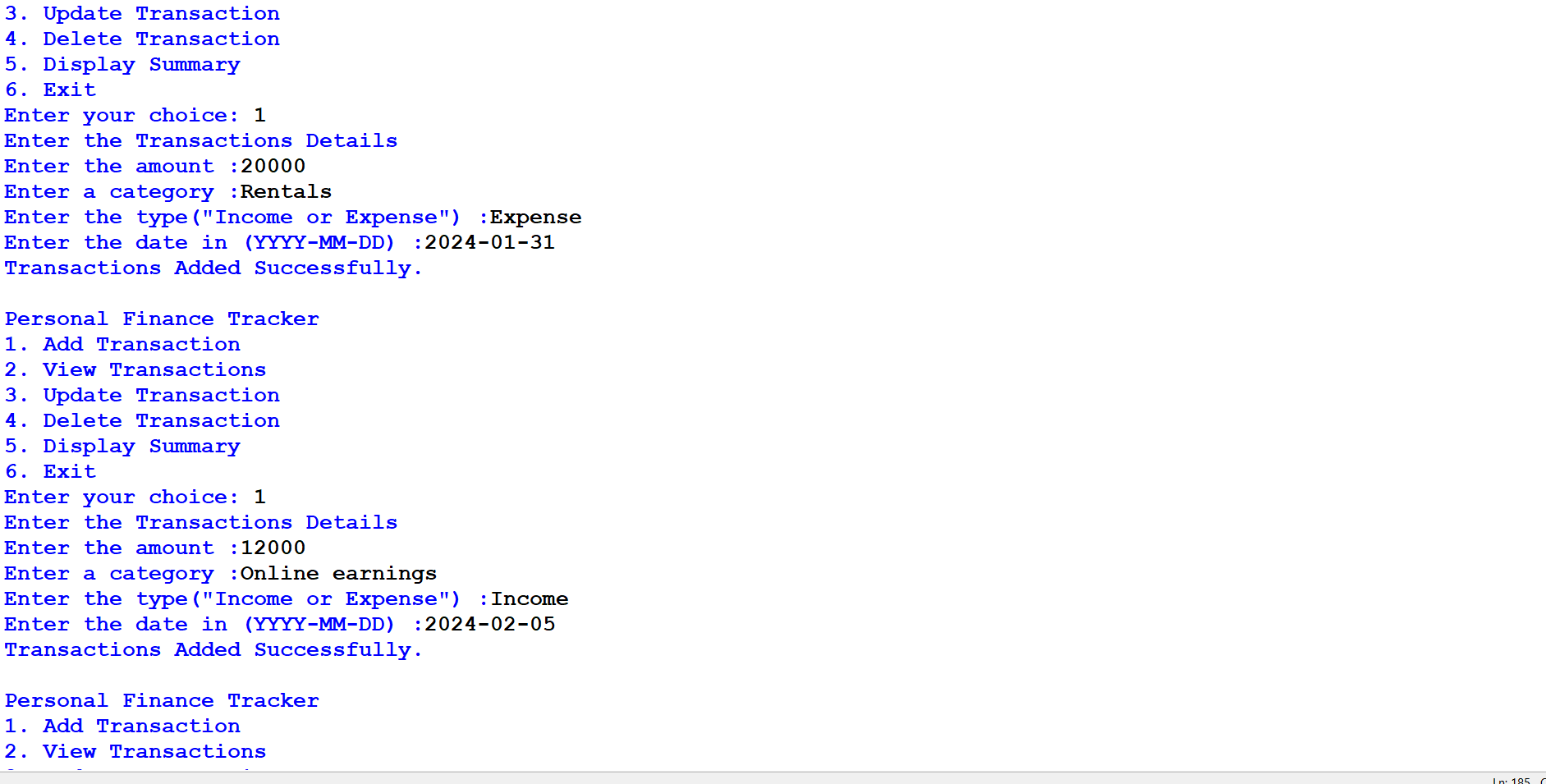
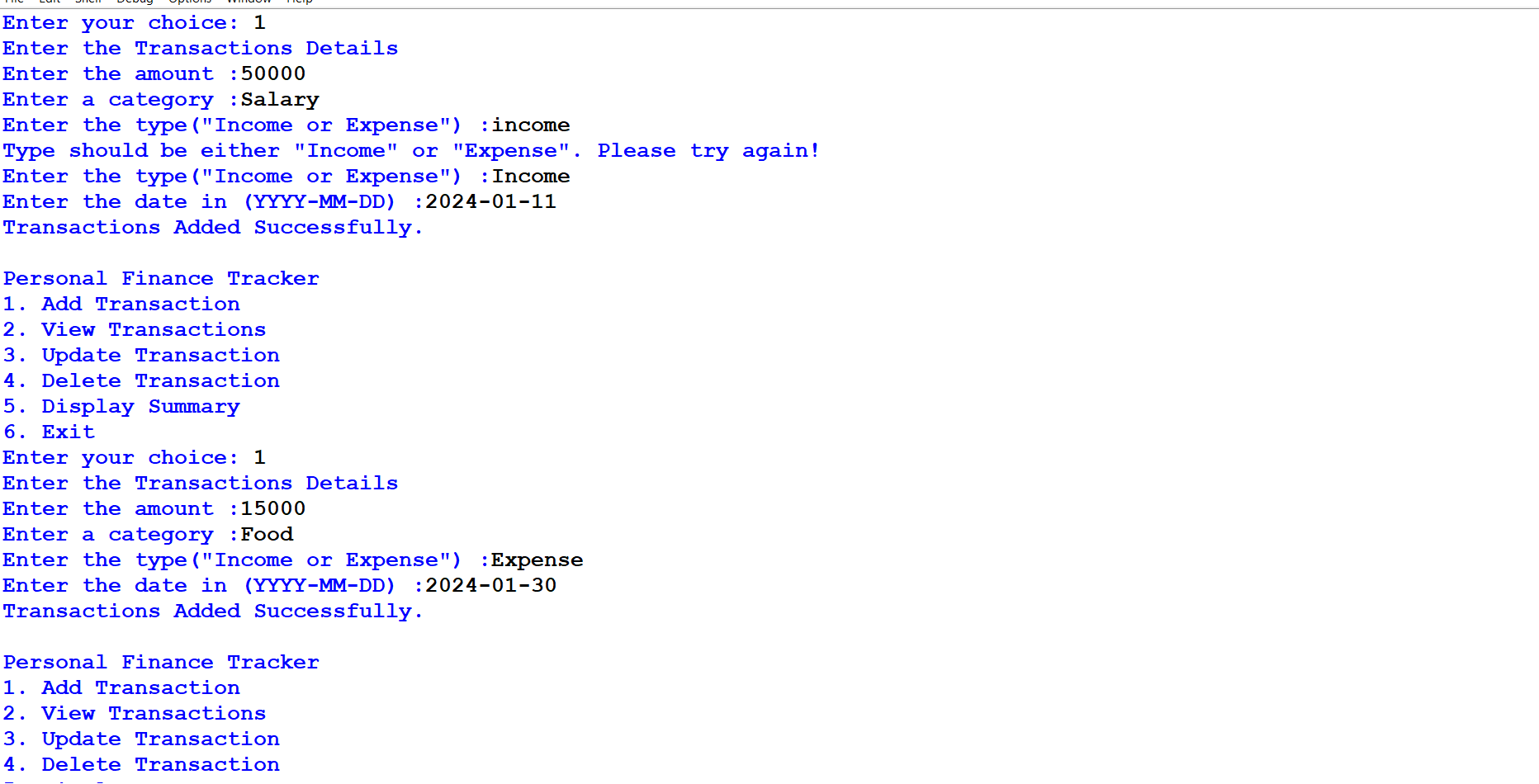
print("Invalid choice. Please try again.")

## 3.2. Test Execution

**Table 1 test execution**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Cases | Description | Inputs | Expected Outputs | Actual Outputs | Test Status |
| 1 | add\_  transaction | amount, category,  type,  date | Transactions  .json file is created and stores the transactions list | Yes. Actual output was same as expected output | Test PASS |
| 2 | view\_  transaction |  | Able to view all the transactions of the transactions list | Same as the expected output | Test PASS |
| 3 | update\_  transaction | Input index to update a list.  correct Amount, correct category,  correct type,  correct date | Updates list at the specified index of transactions list | Transaction updated successfully | Test PASS |
| 4 | delete\_  transaction | Input index to remove a list | Removes list at the specified index of transactions list | Transaction deleted successfully | Test PASS |
| 5 | display\_  summary | No input | total\_income, total\_expense, net\_income or net\_expense | Outputs the total\_income, total\_expense, net\_income or net\_expense | Test PASS |
| 6 | main\_menu | Input the choice to select an option (1-6) | Runs the program throughout prompting the user to input a choice and exits . | Same as the expected output | Test PASS |

# Task 4. Screenshots of Test outputs



4.1 Text file Output