* **Name : Duwe Siriwardhana Arachchige Deshan Chinthaka**
* **Specification : Personal Finance Tracker**

**(GUI Implementation with Tkinter and OOP)**

* **Documentation**

**Overview :-**

This Python code is written to check the personal financial status of a person. This is a computer program that can be easily use by any person. It has been improved with the GUI by using Tkinter to give a special experience for user. The main thing I have considered here is to review and show the expenses made by a person. It has been displayed from a GUI window.

The main window of this program displays the details of all transactions done by the user so far. The value of the transaction, the date of the transaction and the description of the transaction were displayed.

A search button is also provided to facilitate the work of the user and to find any specific transaction whenever he or she wants.

A sorting option is also provided by this computer program. It also arranges all transactions in ascending order to apply to the column with that title when the user taps on a topic in the GUI.

***Functionalities.***

Let’s consider the functions provided by the computer program :-

* **Main Menu**

This id the main menu of the program. This function prepares a menu so that the user can see all the choices and a number has been given for each of those choices. After entering the desired selection number, the user is given the opportunity to perform the operation related to that selection. Thus, a number is presented for each choice to make this program user friendly.

Here is the python code for main menu module,

# create a module for main menu

def main\_menu(self):

while True:

print("\n--- Personal Finance Tracker Main Menu ---") # heading prompt

print("1. Add Transaction") # prompts for user input

print("2. View Transactions")

print("3. Delete Transaction")

print("4. Update Transaction")

print("5. Transaction Menu")

print("6. Display Summary")

print("7. Save and Quit")

choice = input("Enter your choice (1-7): ") # get user input as choice

if choice == '1':

self.add\_transaction()

elif choice == '2':

self.view\_transactions()

elif choice == '3':

self.delete\_transaction()

elif choice == '4':

self.update\_transaction()

elif choice == '5':

self.open\_gui()

elif choice == '6':

self.display\_summary()

elif choice == '7':

self.save\_transactions()

break

else:

print("Invalid choice. Please try again.") # for invalid user inputs

* **Create the “FinanceTrackerCLI “ class**

This class created for all the modules related to Command Line Interface (CLI)

Here is the python code for FinanceTrackerCLI module,

class FinanceTrackerCLI:

def \_\_init\_\_(self):

self.transactions = {} # create transaction dictionary to store transactions

self.load\_transactions("transactions.json") # load the JSON file

self.main\_menu() # calling main menu to display all selections for the user

* **Load transactions**

This function loads all the transactions in the JSON file for the main program.

Here is the python code for load transaction module,

# create a module for load transactions

def load\_transactions(self, filename):

try: # validating the "transactions.json" file is exist or not

with open(filename, "r") as file: # open the "transaction.json" file in read mode

self.transactions = json.load(file) # loading the JSON file

print("Transactions loaded successfully.")

except FileNotFoundError:

print("No transactions file found.")

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* **Add transaction**

This item allows the user to enter new transactions. Here as the requirements of the transaction,

1. The nature of the transaction
2. The value of the transaction
3. The date of the transaction

The computer program is stopped if the user gives an incorrect value for the value of the transaction. Also, if the user enters a wrong date, the transaction date will be updated to today’s date under that specific item.

Here is the python code of add transaction module,

def add\_transaction(self):

try:

program will give a error message

category = input("Please enter the Description (category): ")

amount = float(input("Please enter the amount: "))

date = input("Please enter the date of transaction (YYYY-MM-DD): ")

# makeing sure that date is not to be a empty cell.

if date == "":

# Get the current date and time.

current\_datetime = datetime.now()

# Take out the date part only.

current\_date = current\_datetime.date()

# Converet the date as a string in a special format.

formatted\_date = current\_date.strftime("%Y-%m-%d")

# set the transaction date as formatted date.

date = formatted\_date

try:

# checking the inputed date is valid date or not.

datetime.strptime(date, '%Y-%m-%d')

except ValueError:

print("Invalid date format!!!. Therefore the date updated to current date...")

# Get the current date and time.

current\_datetime = datetime.now()

# Take out the date part only.

current\_date = current\_datetime.date()

# Converet the date as a string in a special format.

formatted\_date = current\_date.strftime("%Y-%m-%d")

# set the transaction date as formatted date.

date = formatted\_date

if category not in self.transactions: # check user entered category is already exist. if it doesn't exist then a new category will create in the JSON file

self.transactions[category] = [] # create a new list for new categories

self.transactions[category].append({"amount": amount, "date": date}) # if entered category is already exist then the transaction may added to that category as a children

print("Transaction added successfully.")

self.save\_transactions() # call save transaction module for save the new transactions

except ValueError:

print("Please enter valid values for amount.") # the error message for invalid amount entries

* **Save transaction**

This option saves transactions entered correctly by the user.

Here is the python code for save transactions module,

# create a module for save transaction

def save\_transactions(self):

with open("transactions.json", "w") as file: # open the JSON file in write mode

json.dump(self.transactions, file)

print("Transactions saved.")

* **Delete transaction**

This option allows the user to delete a particular transaction when required. All transactions are displayed to the user before the transaction is deleted and then the number of the transaction to be deleted is obtained from the user.

Here is the python code for delete transaction module,

# create a module for delete a specific transaction that given by user

def delete\_transaction(self):

self.view\_transactions() # before delete a transaction user could be watch wll the transactions. call view transaction module for that task

category = input("Enter the category to delete transaction from: ")

if category in self.transactions: # validation for the given category in the transactions or not

number = int(input("Enter the number of the transaction to delete: "))

if 1 <= number <= len(self.transactions[category]): # validating the given number is valid or not

del self.transactions[category][number - 1]

print("Transaction deleted successfully.")

if len(self.transactions[category]) == 0 : # if a specific transaction hasn't any child then that ategory will removed

del self.transactions[category]

self.save\_transactions()

else:

print("Invalid number.") # if user gave a invalid number then the error will displayed

else:

print("Category not found.") # if the user gave a invalid category then the error will displayed

* **View transactions (For CLI)**

This item displays all the transactions that have been done so far.

Here is the python code for view transaction module,

def view\_transactions(self):

for category, transactions\_list in self.transactions.items():

print(f"{category}:")

counter = 1

for transaction in transactions\_list:

amount = transaction['amount']

date = transaction['date']

print(f"{counter}. Amount: {amount:.2f}, Date: {date}")

counter += 1

* **Display summary**

This option shows a summary of all the transactions that have been done so far.

Here is the python code for display summary module,

# create a module for display a summary of all transactions had been done so far

def display\_summary(self):

for category, transactions\_list in self.transactions.items():

total\_amount = sum(transaction['amount'] for transaction in transactions\_list)

print(f"Total {category} Amount: {total\_amount:.2f}")

* **Update transaction**

If the user wants to modify a transaction which currently in the transaction list, he can do so using this option. This will display all transactions in the JSON file to the user before updating that transaction. Then the category of transaction required to be updated and the number of the transaction is obtained from the user and the task is performed. Also important to note that a transaction is updated only if the user provides a valid amount for the transaction.otherwise, the transaction will ignore.

Here is the python code for update transaction module

def update\_transaction(self):

self.view\_transactions() # before update a transaction user could be watch wll the transactions. call view transaction module for that task

category = input("Enter the Description (category) of the transaction to update: ")

if category in self.transactions: # validation for the given category in the transactions or not

try:

number = int(input("Enter the number of the transaction to update: "))

if 1 <= number <= len(self.transactions[category]): #validating the index number is valid or not

amount = float(input("Enter new amount: ")) #get new details related to the updatable transaction

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

# makeing sure that date is not to be a empty cell.

if date == "":

# Get the current date and time.

current\_datetime = datetime.now()

# Take out the date part only.

current\_date = current\_datetime.date()

# Converet the date as a string in a special format.

formatted\_date = current\_date.strftime("%Y-%m-%d")

# set the transaction date as formatted date.

date = formatted\_date

try:

# checking the inputed date is valid date or not.

datetime.strptime(date, '%Y-%m-%d')

except ValueError:

print("Invalid date format!!!. Therefore the date updated to current date...")

# Get the current date and time.

current\_datetime = datetime.now()

# Take out the date part only.

current\_date = current\_datetime.date()

# Converet the date as a string in a special format.

formatted\_date = current\_date.strftime("%Y-%m-%d")

# set the transaction date as formatted date.

date = formatted\_date

self.transactions[category][number - 1] = {"amount": amount, "date": date}

print("Transaction updated successfully.")

else:

print("\n!!! Enter a valid index value.!!!") #if user entered a invalid index number

except ValueError:

print("\nPlease enter a valid value.\nRecorded previous transaction was discarded!") # if user gave a invalid value then the update will discard

else:

print("Category not found.")

let’s consider the modules in GUI (Graphical User Interface)

create a class mainly for doing stuff in the GUI. It was named FinanceTrackerGUI

Here is the python code of FinanceTrackerGIU class

class FinanceTrackerGUI:

def \_\_init\_\_(self, root, transactions):

self.root = root

self.root.title("Personal Finance Tracker GUI (Using Tkinter)")

self.transactions = transactions

self.filtered\_transactions = None # To store filtered transactions

self.create\_widgets()

self.display\_transactions() # call display transaction module to display all transactions in GUI Treeview

* **Create widgets**

What was done here was to create all the windows with their geometries and all the elements needed for the GUI. Here I have offered two features for the user. They are search option sort option.

Here is the python code for create widgets module

# create a module for create widgets

def create\_widgets(self):

# Frame for table and scrollbar

table\_frame = ttk.Frame(self.root)

table\_frame.pack(pady=10, padx=5)

# create tree heading to display transaction details with relevent colomns in Treeview

self.tree = ttk.Treeview(table\_frame, columns=("Description", "Amount", "Date"), show="headings")

self.tree.heading("Description", text="Description", command=lambda: self.sort\_by\_column("Description")) # Description colomn heading

self.tree.heading("Amount", text="Amount", command=lambda: self.sort\_by\_column("Amount")) # Amount colomn heading

self.tree.heading("Date", text="Date", command=lambda: self.sort\_by\_column("Date")) # Date colomn heading

self.tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10) # set the geometries of the main table in GUI

# create a vertical scrollbar for the Treeview

scrollbar = ttk.Scrollbar(table\_frame, orient="vertical", command=self.tree.yview)

scrollbar.pack(side=tk.RIGHT, fill=tk.Y) # the scrollbar set to the right side side of the Treeview

self.tree.config(yscrollcommand=scrollbar.set)

# Create a frame frame for the "Search"

search\_frame = ttk.Frame(self.root)

search\_frame.pack(pady=10, padx=30)

# create a StringVar to store the search item inputs (this is a special variable type in tkinter)

self.search\_var = tk.StringVar()

# create a search input cell for the user inputs

self.search\_entry = ttk.Entry(search\_frame, textvariable=self.search\_var)

self.search\_entry.pack(side=tk.LEFT, padx=8)

# create a button for search

search\_button = ttk.Button(search\_frame, text="Search", command=self.search\_transactions)

search\_button.pack(side=tk.LEFT) # the search button is displayed to the left of the search entry cell

**Display transactions (For GUI)**

The main program loads the GUI if the user enters the transaction menu item in main menu as an option. Then this module is triggered and display all the transactions that done so far.

Here is the python code for display transactions module

# create a module for display transactions

def display\_transactions(self):

# Clear existing entries in the treeview

for item in self.tree.get\_children():

self.tree.delete(item)

# Determine which transactions to display (filtered or all)

# if searched transaction category already exist, then "filtered\_transaction" will triggered

# id searched transaction doesn't exist, then all transactions will displayed

transactions\_to\_display = self.filtered\_transactions if self.filtered\_transactions else self.transactions

# Populate treeview with transactions

for category, transactions\_list in transactions\_to\_display.items():

for index, transaction in enumerate(transactions\_list):

amount = transaction['amount']

date = transaction['date']

self.tree.insert("", "end", values=(category, f"{amount:.2f}", date)) # amount of transaction is display with two floating points

* **Search transaction**

If userwants to search for a particular transaction, he can do so through the search bar provided in GUI. Here it is sufficient to enter the description or date of the transaction as the requirements for the search option.

Here is the python code for search transaction module

# create a module for search transaction

def search\_transactions(self):

search\_term = self.search\_var.get().strip().lower()

if not search\_term:

messagebox.showinfo("Search", "Please enter a search term.")

return

self.filtered\_transactions = {} # create a dictionary to store searched transactions

for category, transactions\_list in self.transactions.items():

filtered\_transactions\_list = [transaction for transaction in transactions\_list

if search\_term in category.lower() or search\_term in transaction['date'].lower()]

if filtered\_transactions\_list:

self.filtered\_transactions[category] = filtered\_transactions\_list # confermation for searched transaction is exist in the JSON file

self.display\_transactions() # call "display transaction" module if searched transaction doesn't exist

* **Sort transaction**

What is done here is to sort all transactions related to an item that the user enters. Here, the magnitude of the value of the transactions and the ascending order of the date are used as the method of arranging transactions.

Here is the python code for sort transaction module,

def sort\_by\_column(self, column):

# Determine which transactions to sort (filtered or all)

transactions\_to\_sort = self.filtered\_transactions if self.filtered\_transactions else self.transactions

# Sort transactions list based on the selected column (considering the letter case and acsending order)

for category, transactions\_list in transactions\_to\_sort.items():

transactions\_list.sort(key=lambda x: x[column.lower()])

self.display\_transactions() # display the transactions, after sorting by given order