T1A3 TERMINAL APPLICATION

By Brett Russell

STRUCTURE

User Selection

Main Menu

Add Income

Add Expenses

Calculate Finances

New User

Switch User

Delete User

USER SELECTION

```
(.venv) → BrettRussell_T1A3 git:(main) X python3 main.py
File not found. Creating a new file.
Please enter a name:
```

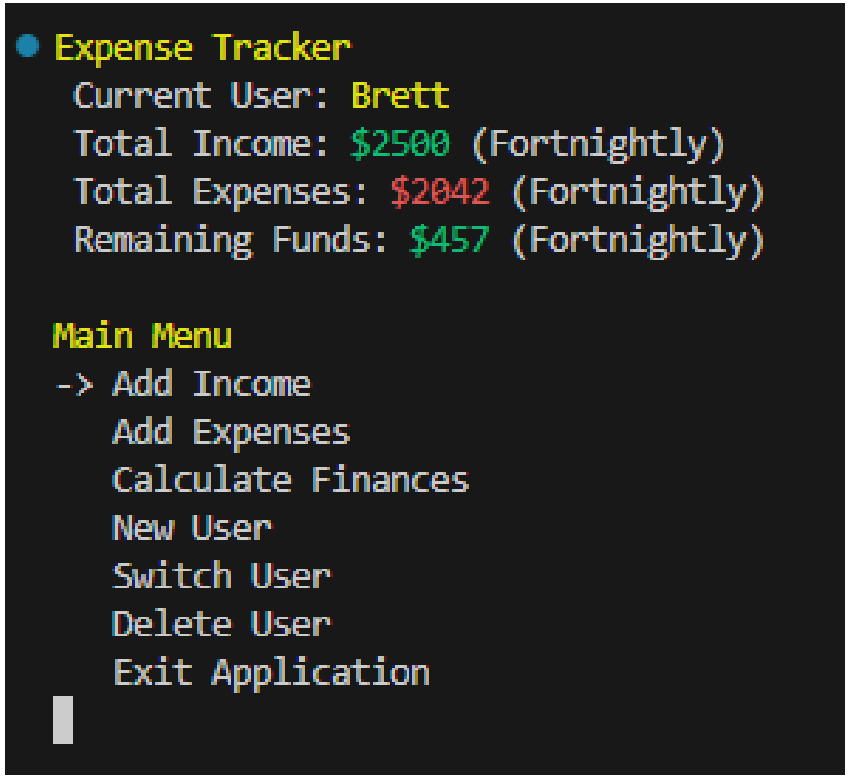
- If no user is found a name input is required.
- If users are found, then a user is selected when the program opens.

```
Select User
-> Brett
   John
   Mary
   New User
```

MAIN MENU

Displays the current user and financial totals

User can access all features through selection.



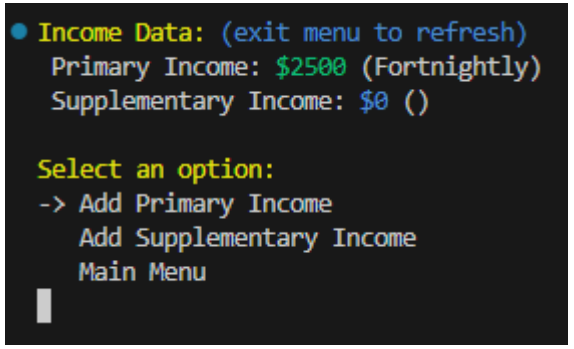
ADD INCOME

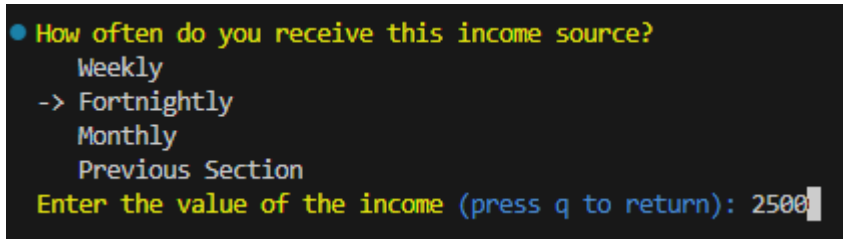
Shows recorded income data.

Allows user to select income type.

Allows user to select how often income is received.

User then inputs income value to be saved to profile dictionary.





ADD EXPENSES

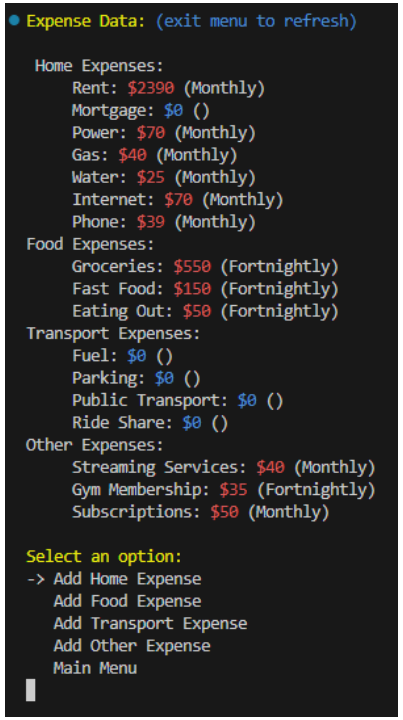
Shows recorded expense data.

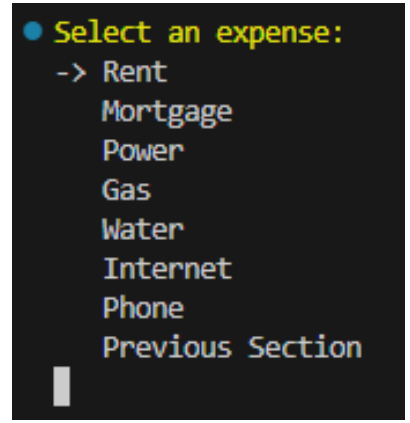
User selects an expense category.

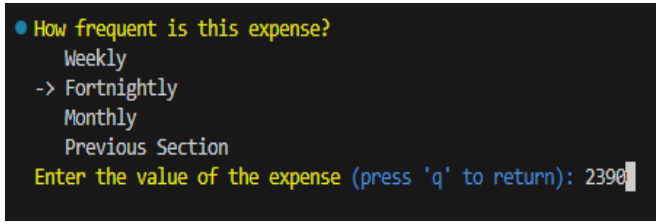
User selects a type of expense.

User selects how often that expense is deducted/calculated for.

Input is taken for each expense value.





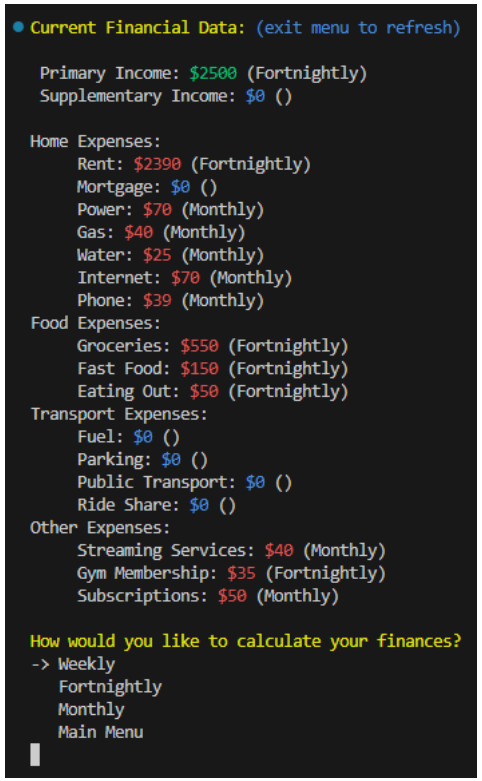


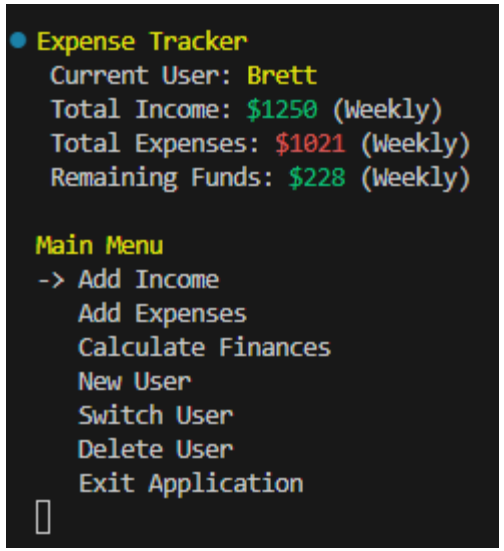
CALCULATE FINANCES

All financial data is displayed.

User selects time frame for calculation

User is taken back to main menu and their totals are updated.



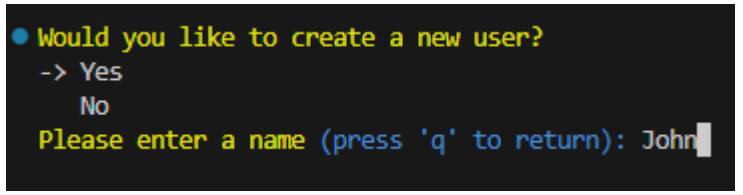


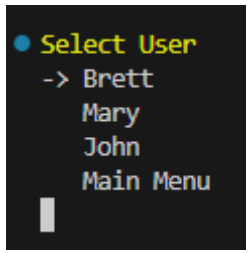
NEW USER
SWITCH USER
DELETE USER

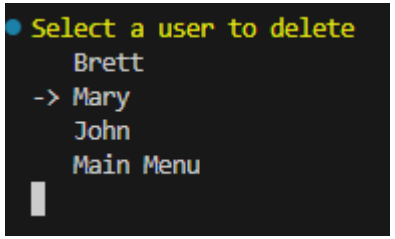
User profiles can be created in New User.

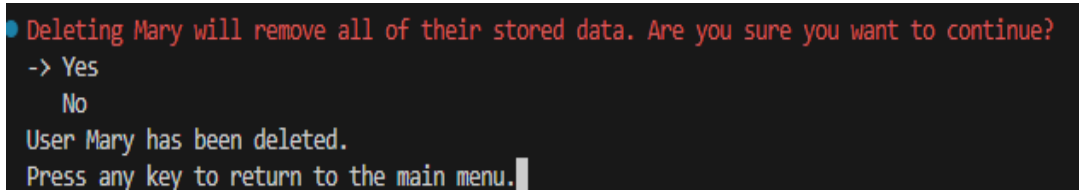
Users can change between profiles.

Profiles can be deleted if they are no longer used.









MAIN LOGIC

First imports necessary functions, packages and data sets.

Retrieves data from users_data.

Creates User objects and adds them to saved_users.

If no user is found, there is a prompt to create one.

main.py > ...

```python
from colored import fg, attr

from functions import display_menu, save_users, switch_user
from functions import user_selection_menu, new_user_creation, save_user_data
from functions import add_income, add_expenses, generate_income_info
from functions import generate_expense_info, calculate_finance, delete_user
from functions import saved_users, users_data, filename
from user import User
from lists import main_menu_options, add_income_options
from lists import add_expenses_options, calculate_average_options


COLOR_YELLOW = fg('yellow')
COLOR_RED = fg('red')
COLOR_BLUE = fg('blue')
RESET_COLOR = attr('reset')


# retrieves user data when the program is run
for name, data in users_data.items():
    try:
        user = User(name)
        for key, value in data.items():
            setattr(user, key, value)
        saved_users.append(user)
    except Exception as e:
        print(
            f"{COLOR_RED}Error retrieving user data for "
            f"{name}: {e}{RESET_COLOR}"
        )
```

```python
# handles creating a new user if none exists
if not saved_users:
    while True:
        try:
            user_name = input(
                f"{COLOR_YELLOW}Please enter a name:{RESET_COLOR} "
            ).strip()
            if not user_name:
                print(
                    f"{COLOR_RED}Name field cannot be empty, "
                    f"please enter a name.{RESET_COLOR}"
                )
            else:
                new_user = User(user_name)
                saved_users.append(new_user)
                users_data[user_name] = new_user.to_dict()
                save_users(users_data, filename)
                current_user = new_user
                break
        except Exception as e:
            print(
                f"{COLOR_RED}Error creating a new user: {e}{RESET_COLOR}"
            )

else:
    # handles user selection or creation if a user exists
    while True:
        try:
            selected_user_index = user_selection_menu(saved_users)
            if selected_user_index < len(saved_users):
                current_user = saved_users[selected_user_index]
                break
            else:
                new_user = new_user_creation()
                if new_user:
                    saved_users.append(new_user)
                    users_data[new_user.name] = new_user.to_dict()
                    save_users(users_data, filename)
                    current_user = new_user
                    break
        except Exception as e:
            print(
                f"{COLOR_RED}Error encountered during user selection: "
                f"{e}{RESET_COLOR}"
            )
```

MAIN LOGIC CONTINUED

The program then executes is main loop.

Variables are defined to present data.

The function display_menu handles the selection of menu options.

A match case is used to designate functions to each section.

```
# main loop for program logic
while True:
    try:
        current_user_data = users_data[current_user.name]
    except KeyError:
        print(f"{COLOR_RED}Error: User data not found.{RESET_COLOR}")
        continue
    # variables for financial data displayed in program interface
    user_finance_info = generate_income_info({
        "Total Income": current_user_data['total_income'],
        "Total Expenses": current_user_data['total_expense'],
        "Remaining Funds": current_user_data['remaining_funds']
    })

    user_income_info = generate_income_info({
        'Primary Income': current_user_data['primary_income'],
        'Supplementary Income': current_user_data['supplementary_income']
    })

    home_expense_info = generate_expense_info(
        current_user_data['expense']['home']
        )
    food_expense_info = generate_expense_info(
        current_user_data['expense']['food']
        )
    transport_expense_info = generate_expense_info(
        current_user_data['expense']['transport']
        )
    other_expense_info = generate_expense_info(
        current_user_data['expense']['other']
        )
```

```
    # function call for main menu selection
    selected_option = display_menu(main_menu_options, main_prompt)
except Exception as e:
    print(f"{COLOR_RED}An error occured: {e}{RESET_COLOR}")
    continue

match selected_option:
    # match case for all selection options
    case 0:
        # add income section
        while True:
            add_income_prompt = (
                "{yellow}Income Data:{reset} "
                "{blue}(exit menu to refresh){reset}\n"
                " {income_info}\n"
                "{yellow}Select an option:{reset}"
            ).format(
                income_info=user_income_info,
                yellow=COLOR_YELLOW,
                blue=COLOR_BLUE,
                reset=RESET_COLOR
            )

            selected_sub_option = display_menu(
                add_income_options,
                add_income_prompt
            )
            if selected_sub_option in [0, 1]:
                income_type = (
                    'primary' if selected_sub_option == 0
                    else 'supplementary'
                )
                # functions called for add income
                add_income(current_user, income_type)
                save_user_data(users_data, current_user, filename)
            else:
                break

    case 1:
        while True:
            # add expenses section
            add_expenses_prompt = (
                "{yellow}Expense Data:{reset} "
                "{blue}(exit menu to refresh){reset}\n\n"
                " {expense_info}\n\n"
                "{yellow}Select an option:{reset}"
            ).format(
                expense_info=user_expense_info,
```

FILE HANDLING

Two main functions for handling user data.

save_user_data takes users_data dictionary and then uses the JSON module to write that data into the external file.

load_users opens and reads the JSON file and then converts it into a python dictionary when the program is executed.

```python
# saves new data and converts into user JSON dictionary
def save_user_data(users_data, user, filename):
    users_data[user.name] = user.to_dict()
    try:
        with open(filename, 'w') as file:
            json.dump(users_data, file, indent=4)
    except IOError as e:
        print(
            f"{COLOR_RED}Failed to save user data: "
            f"{e}{RESET_COLOR}"
        )
    except json.JSONDecodeError as e:
        print(
            f"{COLOR_RED}Failed to format user data as JSON: "
            f"{e}{RESET_COLOR}"
        )
    except Exception as e:
        print(
            f"{COLOR_RED}An unexpected error occurred: "
            f"{e}{RESET_COLOR}"
        )
```

```python
# function for reading json file data an returning as a dictionary.
def load_users(filename):
    try:
        with open(filename, "r") as file:
            # reads json content to convert information to dictionary
            data = json.load(file)
            return {
                name: (info if isinstance(info, dict) else {})
                for name, info in data.items()
                }
    except FileNotFoundError:
        print(f"{COLOR_RED}File not found. Creating a new file.{RESET_COLOR}")
        return {}
    except json.JSONDecodeError:
        print(
            f"{COLOR_RED}Error reading the file. "
            f"Data may be corrupted.{RESET_COLOR}"
            )
        return {}


users_data = load_users(filename)
```

INCOME/EXPENSE DATA FUNCTIONS

The functions that handle the input of key data are add_income and add_expense.

They work by taking a selection to define the ‘occurrence’ key/value pair and then taking numerical input for the ‘amount’ key/value pair for each income and expense item in the users dictionary.

```python
# function for taking user input to store income data
def add_income(user, income_type):
    occurrence = display_menu(
        occurrence_options,
        f"{COLOR_YELLOW}How often do you receive this income source?"
        f"{RESET_COLOR}")

    if occurrence_options[occurrence] == "Previous Section":
        return

    while True:
        # takes numerical input and converts to float data
        income_value_input = input(
            f"{COLOR_YELLOW}Enter the value of the income{RESET_COLOR} "
            f"{COLOR_BLUE}(press q to return):{RESET_COLOR} "
        )
        if income_value_input.lower() == 'q':
            return
        try:
            income_value = float(income_value_input)
            # sets how data is catagorised for storage
            income_info = {
                "amount": income_value,
                "occurrence": occurrence_options[occurrence]
            }
            # organises type of income data to be stored
            if income_type == 'primary':
                user.primary_income = income_info
            else:
                user.supplementary_income = income_info
            save_user_data(users_data, user, filename)
            break
        except ValueError:
            print(
                f"{COLOR_RED}Invalid input,"
                f"please use only numbers.{RESET_COLOR}"
                )
```

```python
def add_expenses(user, expense_category):
    while True:
        match expense_category:
            case "home":
                options = home_expense_options
            case "food":
                options = food_expense_options
            case "transport":
                options = transport_expense_options
            case "other":
                options = other_expense_options

        option = display_menu(
            options,
            f"{COLOR_YELLOW}Select an expense:{RESET_COLOR}")
        if option == len(options) - 1:
            break

        expense_name = options[option]
        # sets occurrence value for later calculation
        occurrence = display_menu(
            occurrence_options,
            f"{COLOR_YELLOW}How frequent is this expense?{RESET_COLOR}")
        if occurrence_options[occurrence] == "Previous Section":
            return
        # takes input for expense data
        while True:
            expense_value_input = input(
                f"{COLOR_YELLOW}Enter the value of the expense{RESET_COLOR} "
                f"{COLOR_BLUE}(press 'q' to return):{RESET_COLOR} "
                )
            if expense_value_input.lower() == 'q':
                return
            try:
                # converts data type then stores new data
                expense_value = float(expense_value_input)
                user.expense[expense_category][expense_name] = {
                    "amount": expense_value,
                    "occurrence": occurrence_options[occurrence]
                }
                save_user_data(users_data, user, filename)
                break
            except ValueError:
                print(
                    f"{COLOR_RED}Invalid input,"
                    f"please use only numbers.{RESET_COLOR}"
                    )
```

CALCULATE FINANCE FUNCTION

- This function takes the users previous data input and converts into total sums specified by time frame.
- It defines a conversion dictionary that contains conversion ratios for income and expenses across different occurrences.
- It uses conversion to find an appropriate ratio for conversion then defines total_income and total_expense variables with a sum of the corresponding converted data types.
- It then defines remaining_funds by subtracting the two totals.
- Finally it displays all three variables on the main menu and stores their data in the user dictionary.

```python
def calculate_finance(user, time_frame):
    try:
        # values for conversion calculation
        conversion = {
            "Weekly": {
                "Weekly": 1,
                "Fortnightly": 0.5,
                "Monthly": 12 / 52
            },
            "Fortnightly": {
                "Weekly": 2,
                "Fortnightly": 1,
                "Monthly": 12 / 26
            },
            "Monthly": {
                "Weekly": 52 / 12,
                "Fortnightly": 26 / 12,
                "Monthly": 1
            }
        }
        total_income = 0
        total_expense = 0
        # accesses user income data
        for income in [user.primary_income, user.supplementary_income]:
            if income['amount'] > 0:
                # type of conversion which is performed
                income_calc = (
                    conversion[time_frame][income['occurrence']]
                )
                # amount being converted
                income_contribution = (
                    income['amount'] * income_calc
                )
                # sum of all converted amounts
                total_income += income_contribution
        # loop for accessing all expenses stored
        for category, expenses in user.expense.items():
            for expense in expenses.values():
                if expense['amount'] > 0:
                    expense_calc = (
                        conversion[time_frame][expense['occurrence']]
                    )
                    expense_contribution = (
                        expense['amount'] * expense_calc
                    )
                    total_expense += expense_contribution
        remaining_funds = total_income - total_expense
        # dictionary storage device for new data
        user.total_income = {
            "amount": total_income,
            "occurrence": time_frame}
        user.total_expense = {
            "amount": total_expense,
            "occurrence": time_frame
            }
        user.remaining_funds = {
            "amount": remaining_funds,
            "occurrence": time_frame
            }

        return total_income, total_expense, remaining_funds

    except KeyError as e:
        print(
            f"{COLOR_RED}Invalid time frame or occurrence: "
            f"{e}{RESET_COLOR}"
        )
    except TypeError as e:
        print(
            f"{COLOR_RED}Invalid data type in financial information: "
            f"{e}{RESET_COLOR}"
        )
    except Exception as e:
        print(
            f"{COLOR_RED}An unexpected error occured in finance calculation: "
            f"{e}{RESET_COLOR}"
        )
```

THANKYOU FOR WATCHING!