Warsaw University of Technology

Faculty of Mechatronics

Programming in Python

Project: Expenses Manager

Author:

Dawid Radłowski

1. Introduction

This project was prepared as part of the course-based assessment for the subject Programming in Python. The project topic involved creating an application that allows users to manage their budget. The application was developed using the Python programming language.

Python is a high-level language. The creators' primary goal was to simplify the programming process as much as possible. Python was also intended to be a language where coding would be as close as possible to describing tasks in natural language. This has made the language, while continuously evolving, is one of the most transparent tools in the field of programming.

Many operations can be implemented in much simpler way than in other programming languages. The principles guiding its creators and the communities involved in its continuous development ensure that code performing the same tasks as its counterparts in competing languages like C++ or Java is more concise and readable.

The ease of using this language is certainly influenced by its syntax and grammar, which show a stronger convergence with natural languages than many other programming languages. Additionally, a Python developer has access to an innumerable amount of libraries that can be used for many common tasks a programmer might encounter. These libraries often allow a given task to be accomplished with a single line of code, instead of having to create a series of functions or classes/methods that would be required when solving the problem from scratch.

The fundamental mechanisms on which the language is based are also a factor that positively influences its simplicity. Dynamic typing is certainly worth mentioning here – a variable can be assigned values of different types, depending on the developer's current needs. Furthermore, it significantly relieves the programmer from the burden of managing dynamically allocated memory through the use of a garbage collector.

Like any human creation, Python is not free from flaws. Many critics argue that its overly liberal syntax rules cause chaos in the programming style. This becomes particularly evident in larger projects involving different programmers. Each programmer might write the same functions using different syntactic constructions. When it becomes necessary to check, modify, or reuse code fragments among colleagues, one person's programming habits can be incomprehensible to another's. Such problems do not occur in languages that do not provide different ways to write the same operations.

Moreover, Python is a scripting language. Scripts written in this language are executed by special interpreters in the order in which the source code is written, instruction by instruction. A benefit of this approach is certainly a more economical use of RAM – it is only needed as much memory as currently available variables require. However, the downside here is the program's execution time, which is significantly longer compared to languages that use compilers.

2. Conceptual assumptions

This project involves an application for managing expenses. It aims to provide the user with tools to help monitor the status of their finances. The application is designed to have the necessary functionality for the user to create relevant summaries of costs incurred and profits recorded over selected time periods. Additionally, the application should allow for basic analysis of the user's payment operations.

The list of features the prepared application should have includes:

- Creating, adding, saving, and deleting accounts for new and existing users.
- Entering user's income and expenses:
    - Entering single items of this type.
    - Loading ready-made lists of income and expenses from a file.
- Updating the wallet status based on the date of the loaded transaction—distinguishing between completed and planned transactions.
- Graphical analysis of transactions on the account balance:
    - Pie chart—counting the occurrences of a given expense category or the sum of expenses by a selected category.
    - Bar chart—comparing data series over time periods.
    - Bar chart—displaying the balance of expenses and income.
- Displaying loaded income and expense tables.
- Validating the correctness of entered data—handling exceptions.

3. Implemented logical modules

The application was divided into a logical part and a graphical part. The former is responsible for all data operations that create the core functionality of the program, but their execution is not essential from the user's perspective—only their results are of practical importance. This includes actions such as creating new objects of classes used in the application's code, browsing object collections, and performing operations on values stored in selected objects, etc. The second, graphical part is responsible for communicating the logical layer with the user and transferring data entered externally to it. This part is discussed in point 4. The components that make up the logical layer are described below.

3.1. Basic classes: Interfejs, Posiadacz, TabWydatki

These are the classes prepared for this project. They reflect the elements that can be distinguished in operations related to analyzing a user's budget. Their fields store all the data that undergoes operations during the program's execution. The methods of these classes describe the necessary actions on the aforementioned data.

The **Interfejs** class is the most senior in the logical class hierarchy. It manages all the others. It is responsible for creating objects of classes from lower hierarchical levels and calling their respective methods. It stores objects of the **Posiadacz** class in appropriate dictionaries. It gains access to a desired object by providing a unique key. This class is directly managed by the graphical user interface class. Any data that a user wants to obtain from subordinate classes must be passed through this class.

The next class is **Posiadacz**. Its objects are referring to the users of the application. This class contains basic user information, including name, surname, account balance, and tables of income and expenses. Each time a new user is added, an object of this class is created. All operations on the income and expense tables are performed through this class by calling the methods of the **TabWydatki** class.

The final class is **TabWydatki**. Its objects store both income and expense tables. This depends on the parameters with which the constructor is called during object creation. The methods of this class are used to perform operations on the data contained in the tables. Only the results of these methods' operations are then passed to the **Posiadacz** class, which manages this class.

3.2. Classes of exceptions

In the developed application, five new exception classes are applied. They are:

- **OutOfBondsError** – when the provided value is outside the required range.
- **BadFileFormatError –** in case an incorrect file extension is given for the file from which income and expense data is to be loaded.
- **MissingKeyError –** used when a key is not provided for a dictionary element that is supposed to be selected.
- **RepKeyError –** when an element with the given key value already exists while a new dictionary element is being added with the same key.
- **BadDFCols –** when the column names of a pandas.DataFrame table do not match the established key.

All of the above exception classes were implemented by inheriting from the native `ValueError` class.

3.3. External libraries

The most important libraries used to create the application's logic layer are:

- **pandas**
- **numpy**
- **datetime**

The first two are used for creating and performing operations on data tables. The **pandas** library provides tools that make it possible to create **DataFrame** objects, which store data in a table structure. Values for specific rows and columns are saved in cells. This format is very well-suited for operations on an user's income and expense statements. Many built-in methods allow you to avoid having to implement standard table operations from scratch. The **numpy** library, in turn, is primarily used to properly format the resulting data structure, which is then passed to the graphics class.

The **datetime** library is used in all operations where it's necessary to take into account the time assigned to a given budgetary transaction.

4. GUI

For more convenient user interaction with the application, a graphical user interface was prepared, enabling more intuitive performance of available operations and entry of necessary information. The design of this interface is based on classic windows. They contain all kinds of selection boxes, data entry fields, buttons, and other elements essential for the program's correct operation.

During program execution, a series of different types of windows can be displayed. Each of them allows for distinct actions. The list of these windows includes:

- **Start window** – allows you to select an appropriate user or add a new one.
- **User window** – contains basic user data and a panel with available budget operations.
- **Loaded tables display window**
- **New table addition window**
- **Charts window** – where available chart creation options are located and where the current charts are displayed.
- **Finances window** – provides the option to add a single income or expense directly from the running application.
- **Messages** – various windows with information for the user about actions taken.

Below there are screenshots for each of the windows listed above.

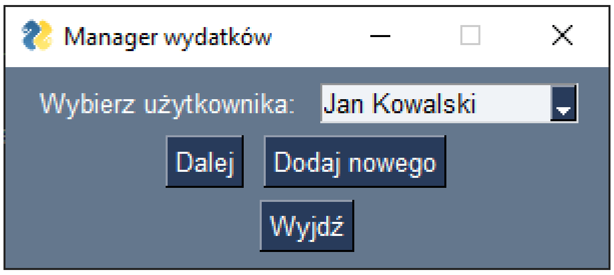
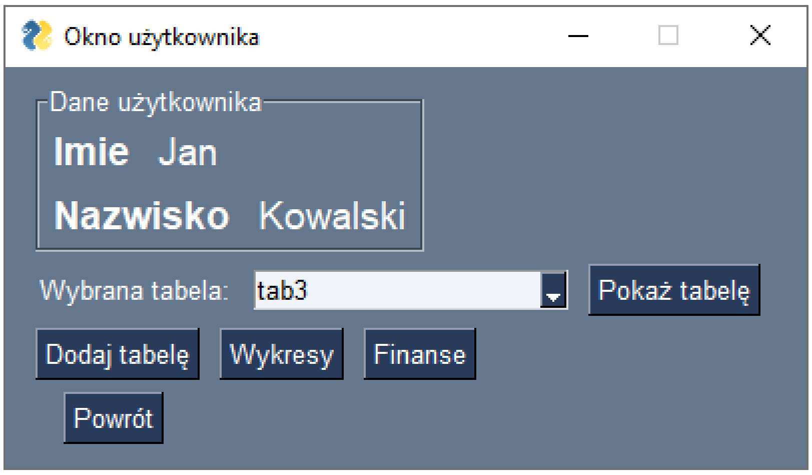


Fig. 1: Start window


Okno użytkownika

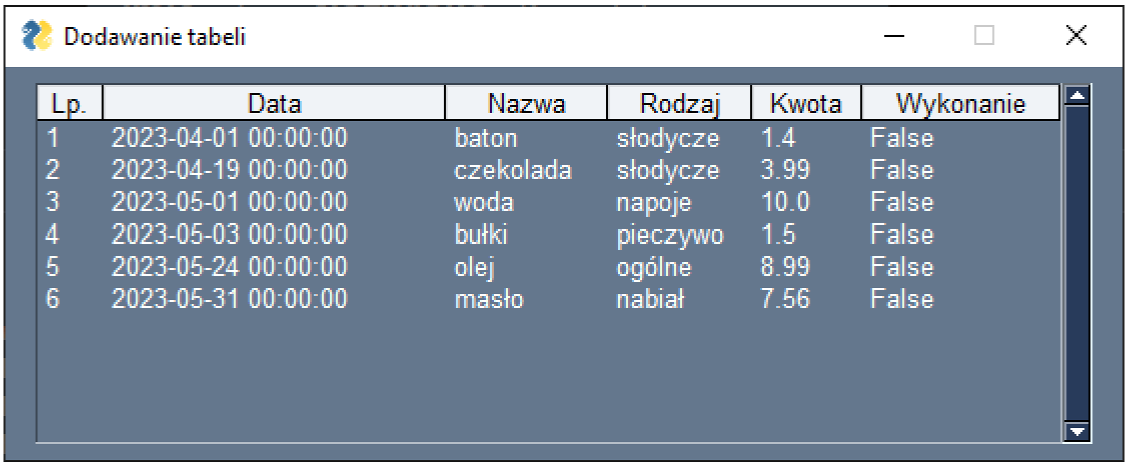
Dane użytkownika

Imie Jan

Nazwisko Kowalski

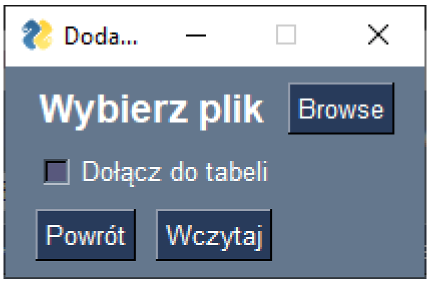
Wybrana tabela: tab3

Fig. 2: User window


Dodawanie tabeli

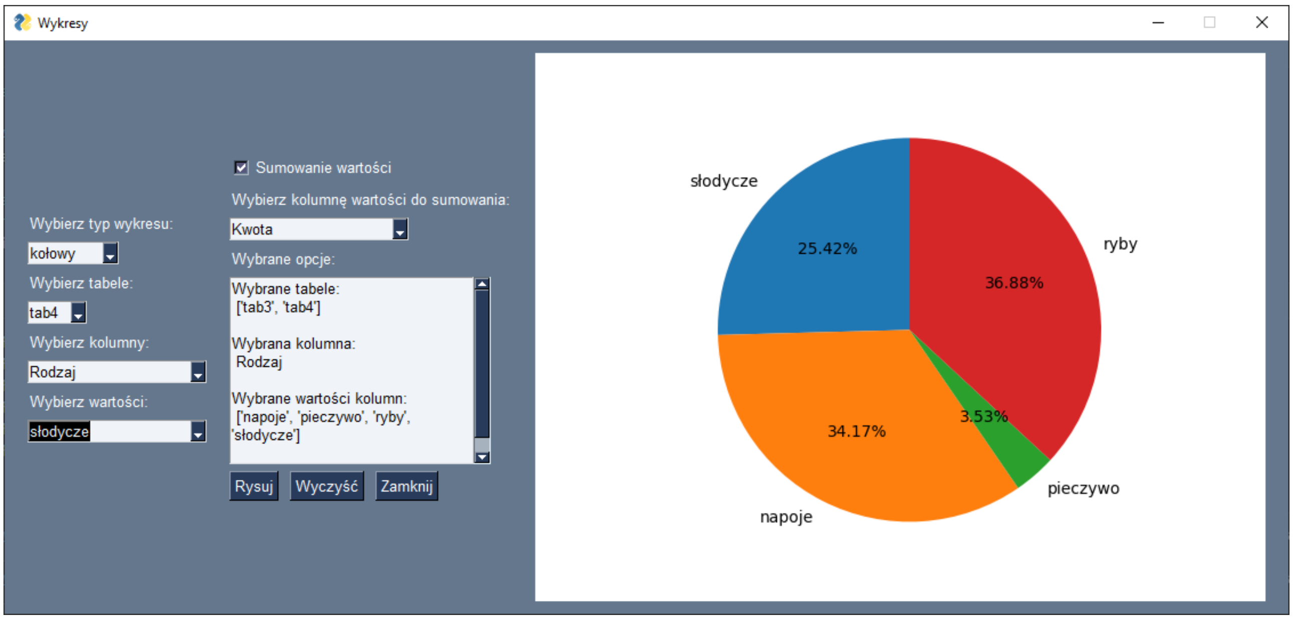
Lp.	Data	Nazwa	Rodzaj	Kwota	Wykonanie
1	2023-04-01 00:00:00	baton	słodycze	1.4	False
2	2023-04-19 00:00:00	czekolada	słodycze	3.99	False
3	2023-05-01 00:00:00	woda	napoje	10.0	False
4	2023-05-03 00:00:00	bułki	pieczywo	1.5	False
5	2023-05-24 00:00:00	olej	ogólne	8.99	False
6	2023-05-31 00:00:00	masło	nabiał	7.56	False

Fig. 3: Loaded tables display window


Doda...

Wybierz plik

Fig. 4: New table addition window



Wykresy

Wybierz typ wykresu: kołowy

Wybierz tabele: tab4

Wybierz kolumny: Rodzaj

Wybierz wartości: słodycze

☑ Sumowanie wartości

Wybierz kolumnę wartości do sumowania: Kwota

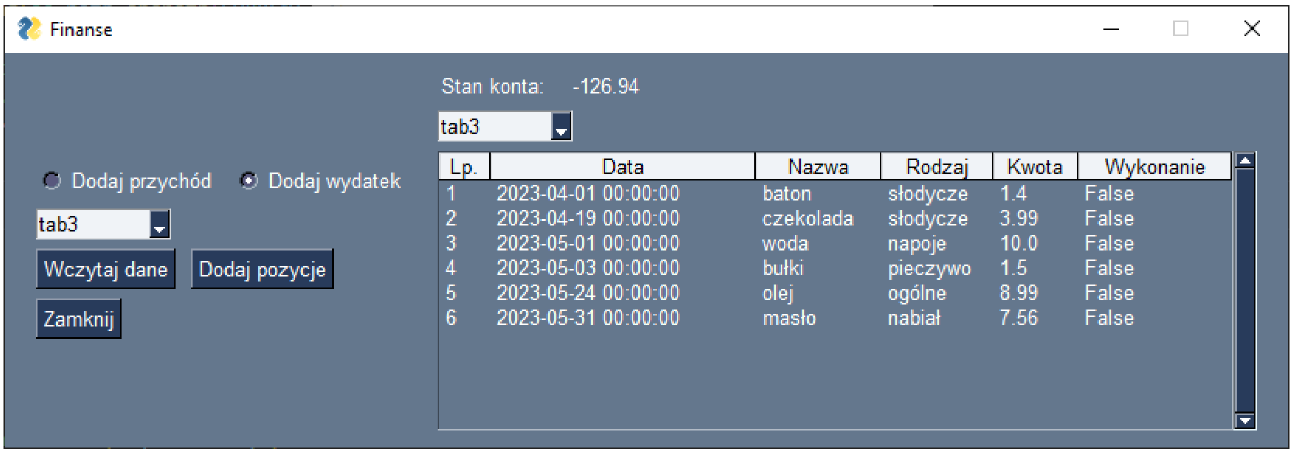
Wybrane opcje:

```
Wybrane tabele:
 ['tab3', 'tab4']

Wybrana kolumna:
 Rodzaj

Wybrane wartości kolumn:
 ['napoje', 'pieczywo', 'ryby',
'słodycze']
```

Fig. 5: Charts window



Finanse

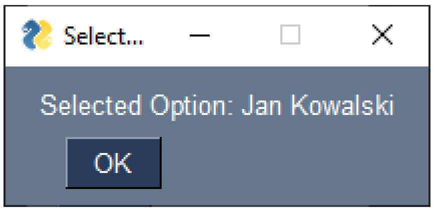
Stan konta: -126.94

○ Dodaj przychód [✓ ● Dodaj wydatek]

tab3

Lp.	Data	Nazwa	Rodzaj	Kwota	Wykonanie
1	2023-04-01 00:00:00	baton	słodycze	1.4	False
2	2023-04-19 00:00:00	czekolada	słodycze	3.99	False
3	2023-05-01 00:00:00	woda	napoje	10.0	False
4	2023-05-03 00:00:00	bułki	pieczywo	1.5	False
5	2023-05-24 00:00:00	olej	ogólne	8.99	False
6	2023-05-31 00:00:00	masło	nabiał	7.56	False

Fig. 6: Finances window



Select...

Selected Option: Jan Kowalski

Fig. 7: Message window

From a programming perspective, each window is implemented as an object of the **Window** class, which belongs to the **PySimpleGUI** library. They are created separately within special methods of the main graphical layer class—**MainWindow**. In each of these methods, in addition to initializing a specific window with a specially adapted list of elements and their layout, a window event loop is also started. This loop is responsible for detecting signals from elements that may be sensitive to a user's chosen actions.

The main window is an exception; it is created directly in the **MainWindow** class constructor. Message windows are also an exception, as they are displayed independently of the other windows through default functions of the **PySimpleGUI** library.

5. Summary

By developing this application with the Python language, it was possible to achieve the full functionality specified in the project assumptions. In addition, the final version of the program's code turned out to be significantly less extensive compared to similar projects written in languages such as C++ and Java.

The graphical layer proved to be the most complex. Nevertheless, in comparison with graphical interface libraries from other programming languages, PySimpleGUI allows for a much simpler implementation of this type of interface. Creating most graphical elements is limited to calling the constructor for a selected class object. PySimpleGUI also significantly assists the programmer with the layout of these elements in individual windows. It provides options to specify how certain objects should adjoin each other, and the environment automatically determines their positions and even sizes.

As for the practical use of the application, more extensive testing would certainly be required. In its current form, it has been tested by a limited number of users. One can assume that during tests with a larger group, additional aspects would be noticed that should be modified in the application to better meet requirements. After just the first tests, we can also list some elements of functionality that are currently missing from the application. These include: accounting for recurring financial transactions, a function for predicting/matching trends over time for income or expenses, exporting created charts in a graphical format, and analyzing the feasibility of a given investment or transaction at a specific time in the future.

Taking all this into account, it can be concluded that the application at its current stage could be used by a user for basic control of a household budget. However, assuming more professional use, further development is definitely required. Completing and eliminating the shortcomings and deficiencies mentioned above would be a good starting point for further work on the project.