

Projekt zaliczeniowy – Języki skryptowe (Python)

1.

Nazwa uczelni: Politechnika Świętokrzyska

Nazwa przedmiotu: Języki skryptowe

Prowadzący zajęcia: Dr inż. Dariusz Michalski

Tytuł projektu: Lokalna Sieć Wymiany Książek

Imiona i nazwiska członków zespołu: Kacper Grabalski, Krzysztof Giemza

Grupa studencka: 2ID12A

Data oddania: 24.06.2025

2. Opis projektu

Cel projektu

Celem projektu było stworzenie aplikacji umożliwiającej użytkownikom wymianę książek w ramach lokalnej społeczności. Aplikacja pozwala na rejestrację użytkowników, dodawanie książek, przeglądanie dostępnych pozycji oraz wymianę książek między użytkownikami.

Funkcje aplikacji

- Rejestracja nowych użytkowników
- Dodawanie książek do biblioteki użytkownika
- Wyszukiwanie książek po tytule, autorze lub gatunku
- Wymiana książek między dwoma użytkownikami
- Wyświetlanie statystyk biblioteki (liczba książek, rozkład gatunków)
- Generowanie wykresu przedstawiającego rozkład książek według gatunków
- Zapis i odczyt danych do pliku JSON

Zaimplementowano następujące funkcje:

- Interfejs użytkownika w trybie tekstowym (CLI)
- Trwałe przechowywanie danych użytkowników i książek
- Wizualizacja danych z użyciem biblioteki matplotlib
- Pełny zestaw testów: jednostkowych, funkcjonalnych, integracyjnych, granicznych i wydajnościowych

3. Struktura projektu

Opis plików i folderów

```
.
├── main.py           # Główny skrypt uruchamiający program
├── storage.py        # Obsługa zapisu i odczytu danych (JSON)
├── user_interface.py # Interfejs użytkownika (menu, formularze)
├── user.py           # Definicja klasy User
├── book.py           # Definicja klasy Book
├── utils.py          # Funkcje pomocnicze (dekoratory, funkcje rekurencyjne)
└── visualization.py  # Generowanie wykresu gatunków książek
```

— test_units.py	# Testy jednostkowe
— test_functional.py	# Testy funkcjonalne
— test_integration.py	# Testy integracyjne
— test_boundary.py	# Testy graniczne
— test_performance.py	# Testy wydajności
— test_memory.py	# Testy pamięci
— library_data.json	# Baza danych książek i użytkowników (tworzona automatycznie)
— README.md	# Dokumentacja projektu

Krótkie omówienie każdego modułu

main.py: Inicjalizuje dane, uruchamia główne menu i zapisuje dane po zakończeniu

- storage.py: Zawiera funkcje load_data i save_data do obsługi pliku JSON
- user_interface.py: Implementuje interfejs tekstowy z menu i funkcjami dodawania, wyszukiwania, wymiany
- user.py: Klasa User reprezentująca użytkownika (atrybuty: id, nazwa, lista książek)
- book.py: Klasa Book reprezentująca książkę (tytuł, autor, id, gatunek, id właściciela)
- utils.py: Funkcje pomocnicze, w tym dekorator log_activity i funkcja rekurencyjna recursive_factorial
- visualization.py: Generuje wykres słupkowy gatunków książek i zapisuje go do pliku PNG

4. Technologie i biblioteki

Python 3.8

Biblioteki standardowe:

- json - zapis i odczyt danych
- os - operacje na systemie plików
- datetime - obsługa dat (w backupach)
- unittest - framework do testów
- re - wyrażenia regularne (walidacja)

Biblioteki zewnętrzne:

- matplotlib - wizualizacja danych (wykresy)
- memory_profiler - testy użycia pamięci
- flake8, pylint - analiza jakości kodu
- pyinstaller - tworzenie pliku wykonywalnego

5. Sposób działania programu

Instrukcja uruchomienia

Uruchomienie Aplikacji:

Pobranie .rar z repozytorium. Wypakowanie zawartości pliku do wybranego przez siebie folderu. Uruchomienie pliku .exe

Uwaga!

Program został podzielony na dwa pliki .rar i żeby aplikacja działała poprawnie należy pobrać oba pliki!

nazwa

pliku: Jezyki_Skryptowe_Proj_2ID12A_Kacper_Grabalski_Krzysztof_Giemza_Aplikacja.part1 oraz Jezyki_Skryptowe_Proj_2ID12A_Kacper_Grabalski_Krzysztof_Giemza_Aplikacja.part2

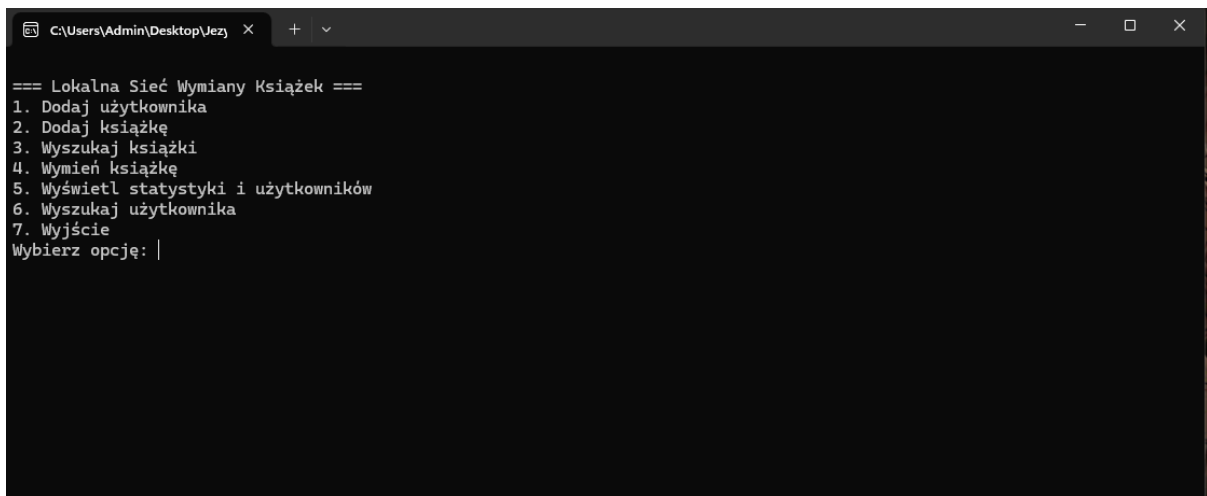
Uruchomienie Kodu:

Pobranie .rar z repozytorium. Wypakowanie zawartości pliku do wybranego przez siebie folderu. Uruchomienie kompilatora z pythonem (np. PyCharm).

nazwa pliku:

Jezyki_Skryptowe_Proj_2ID12A_Kacper_Grabalski_Krzysztof_Giemza

Zrzut ekranu z aplikacji:



```
=== Lokalna Sieć Wymiany Książek ===
1. Dodaj użytkownika
2. Dodaj książkę
3. Wyszukaj książki
4. Wymień książkę
5. Wyświetl statystyki i użytkowników
6. Wyszukaj użytkownika
7. Wyjście
Wybierz opcję: |
```

Przykładowe dane wejściowe/wyjściowe

Dodanie użytkownika:

Wybierz opcję: 1

Imię i nazwisko: Anna Nowak

Utworzono użytkownika: Anna Nowak (ID: 1)

Dodanie książki:

Wybierz opcję: 2

ID właściciela: 1

Tytuł: Hobbit

Autor: J.R.R. Tolkien

Gatunek: Fantasy

Dodano książkę: Hobbit (ID: 101)

Wyszukiwanie użytkownika:

Wybierz opcję: 6

Wprowadź imię, nazwisko lub ID: Anna

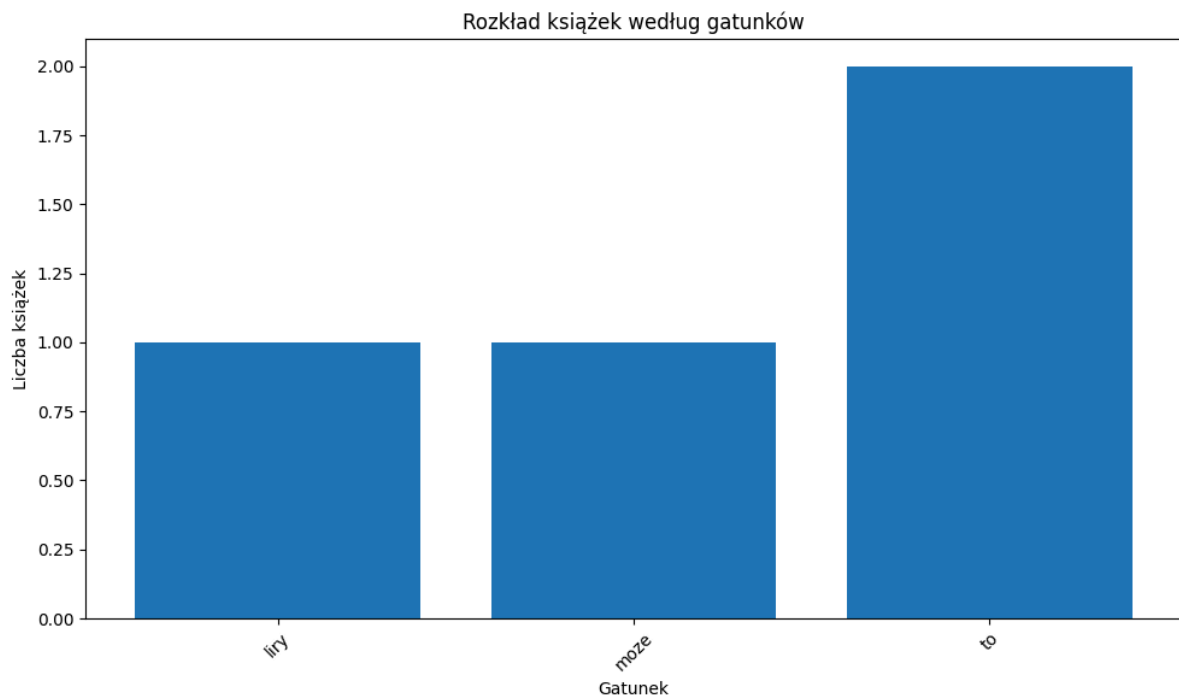
Znalezieni użytkownicy:

ID: 1, Imię i nazwisko: Anna Nowak

Liczba posiadanych książek: 1

ID książek: 101

Przykładowy zrzut ekranu z wykresu:



6. Przykłady kodu

Fragment funkcji funkcyjnej (użycie filter i lambda)

Kod:

```
# user_interface.py
def search_books(books):
    term = input("Szukaj (tytuł/autor/gatunek): ").lower()
    found_books = list(filter(lambda b: term in b.title.lower()
                              or term in b.author.lower()
                              or term in b.genre.lower(),
                              books))
    # ... wyświetlanie wyników ...
```

Fragment klasy (klasa User)

Kod:

```
# user.py
class User:
    def __init__(self, name, user_id, books_owned=None):
        self.name = name
```

```

self.user_id = user_id
self.books_owned = books_owned if books_owned else []

def add_book(self, book_id):
    if book_id not in self.books_owned:
        self.books_owned.append(book_id)

def remove_book(self, book_id):
    if book_id in self.books_owned:
        self.books_owned.remove(book_id)

```

Obsługa wyjątków

Kod:

storage.py

```

def load_data():
    try:
        with open(DATA_FILE, 'r') as f:
            data = json.load(f)
            # ... przetwarzanie danych ...
    except (FileNotFoundError, json.JSONDecodeError) as e:
        print(f"Błąd wczytywania danych: {e}")
        return [], []

```

7. Testowanie

Opis sposobu testowania

- Testy jednostkowe: Sprawdzają poprawność pojedynczych metod i klas (np. tworzenie użytkownika, dodawanie książki)
- Testy funkcjonalne: Symulują interakcje użytkownika (np. wybór opcji w menu)
- Testy integracyjne: Weryfikują współdziałanie modułów (np. proces wymiany książek)
- Testy graniczne: Sprawdzają zachowanie systemu przy dużej ilości danych i danych specjalnych
- Testy wydajności: Mierzą czas zapisu/odczytu danych
- Testy pamięci: Monitorują użycie pamięci przy dużych zbiorach danych

Obsługa przypadków granicznych

- Duże zbiory danych: Test z 1000 użytkowników i 10000 książek
- Specjalne znaki: Obsługa polskich znaków i znaków specjalnych w tytułach i autorach
- Nieprawidłowe dane: Testy walidacji wprowadzanych danych (np. niepoprawne ID)

8. Wnioski

Co się udało

- Zaimplementowano wszystkie zaplanowane funkcje
- Stworzono kompleksowy system testów
- Uzyskano czytelny i dobrze zorganizowany kod
- Wykonano wizualizację danych

Co można było zrobić lepiej

- Dodanie interfejsu graficznego (GUI) zamiast tekstowego
- Rozszerzenie funkcji wyszukiwania o zaawansowane filtry
- Implementacja systemu ocen i recenzji książek

Jakie kompetencje zostały rozwinięte

- Praca z plikami JSON
- Stosowanie zasad programowania obiektowego
- Tworzenie testów jednostkowych i integracyjnych
- Wizualizacja danych z użyciem matplotlib
- Zarządzanie zależnościami i środowiskiem projektu