ФТФ, 2-ый курс, ИВТ-Б(2), Королёв Алексей.

**Ссылка на GitHub:** [https://github.com/ShEIH24/Lab4\_Py](https://github.com/ShEIH24/Lab3_Py)

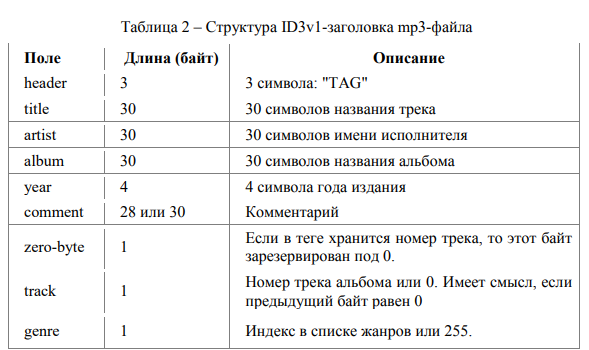
**ЛАБОРАТОРНАЯ РАБОТА №3**

**Тема:** Прикладные пакеты Python.

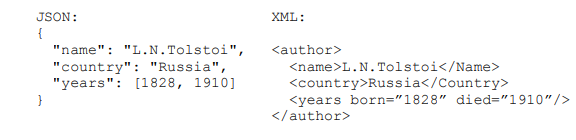
**Цель:** Научиться проектировать и создавать одно- и многопоточные скрипты с графическим интерфейсом, позволяющие работать с базами данных, сетью и слабоструктурированной информацией.

**Индивидуальные задания:**

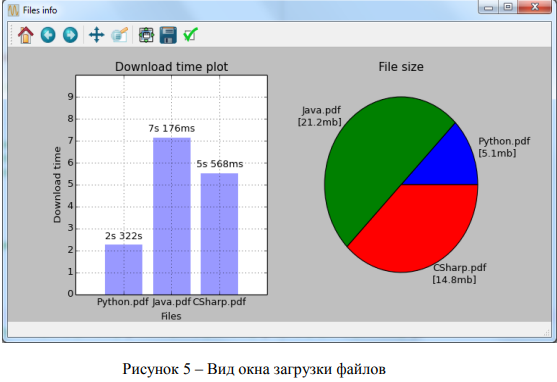
1. Напишите скрипт, читающий во всех mp3-файлах указанной директории ID3v1-теги и выводящий информацию о каждом файле в виде: [имя исполнителя] - [название трека] - [название альбома]. Если пользователь при вызове скрипта задает ключ -d, то выведите для каждого файла также 16-ричный дамп тега. Скрипт должен также автоматически проставить номера треков и жанр (номер жанра задается в параметре командной строки), если они не проставлены. Используйте модуль struct. ID3v1-заголовки располагаются в последних 128 байтах mp3-файла. Структура заголовка отражена в табл. 2.



1. Напишите скрипт для информационной системы библиотеки. База данных библиотеки включает таблицы «Авторы» с полями «id», «имя», «страна», «годы жизни», и «Книги» с полями «id автора», «название», «количество страниц», «издательство», «год издания»). Необходимо производить авторизацию пользователей, логины и пароли которых хранятся в отдельной таблице. Пароли должны храниться в зашифрованном виде (например, хэш SHA-1 или MD5). В программе должны быть окна для отображения информации о всех книгах и авторах, окно добавления книги/автора. Реализуйте также возможность сохранения информации о выделенном авторе в файле в формате json или XML (по выбору пользователя). При добавлении нового автора в базу допускается не заполнять поля в соответствующем окне, а распарсить файл, указанный пользователем (файл необходимо заранее создать и заполнить информацией вручную, в текстовом редакторе). Для преобразования в формат XML и json напишите собственный код; парсинг можно делать с помощью сторонних библиотек. Форматы файлов:



1. Выполните задание № 2 средствами SQLAlchemy, включая создание и редактирование таблиц, а также выполнение таких запросов, как: - вывод фамилий всех авторов, родившихся в диапазоне между X и Y годами (задайте программно числа X и Y); - вывод всех книг, написанных авторами из России; - вывод всех книг с количеством страниц более N; - вывод всех авторов с числом книг более N.
2. Выполните задание № 3, используя в качестве базы данных NoSqlтехнологию MongoDB.
3. Напишите приложение для загрузки файлов из интернета. В главном окне должно быть три текстовых поля, в которые можно вводить URL файла на закачку; под каждым из текстовых полей должны быть индикаторы загрузки и рядом поля с процентом загрузки каждого файла. Необходимо организовать возможность качать от одного до трех файлов параллельно (использовать потоки обязательно, файлы загружать фрагментами по 4 Кб). Загрузка должна инициироваться нажатием кнопки «Start downloading!». По окончанию загрузки последнего файла должно появиться окно со столбчатой диаграммой со значениями времени загрузки каждого 27 файла в формате «2s 322ms» и размерами файлов (используйте библиотеку matplotlib).



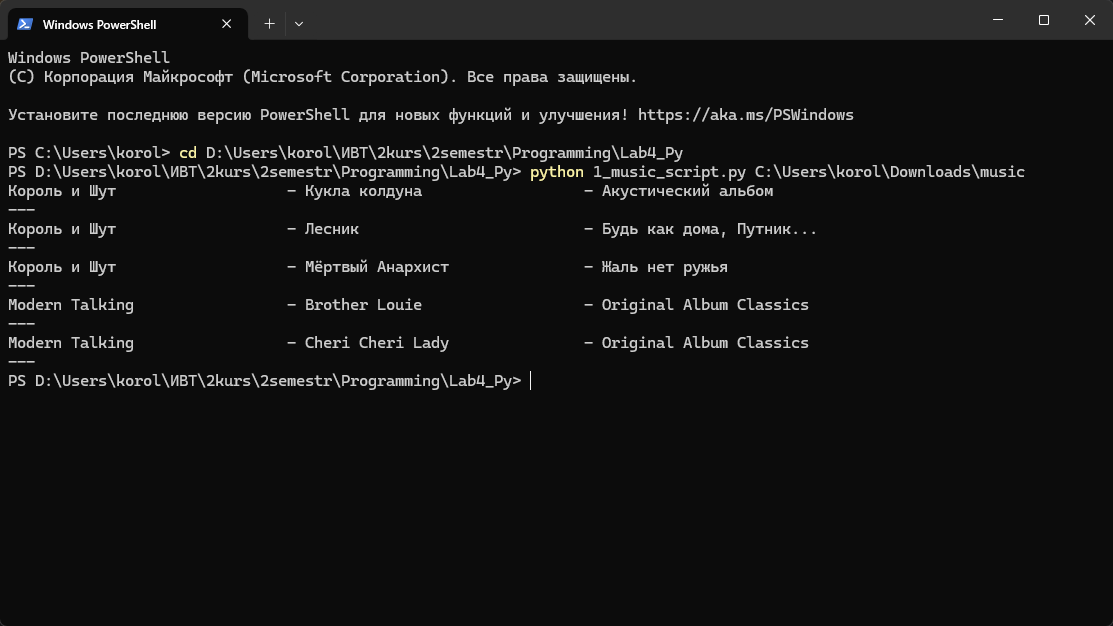
1. С помощью модуля numPy реализуйте следующие операции: 1) умножение произвольных матриц А (размерности 3х5) и В (5х2); 2) умножение матрицы (5х3) на трехмерный вектор; 3) решение произвольной системы линейных уравнений; 4) расчет определителя матрицы; 5) получение обратной и транспонированной матриц. Также продемонстрируйте на примере матрицы 5х5 тот факт, что определитель равен произведению собственных значений матрицы.
2. Выберите произвольную дифференцируемую и интегрируемую функцию одной переменной. С помощью модуля symPy найдите и отобразите ее производную и интеграл в аналитическом и графическом виде. Напишите код для решения произвольного нелинейного урванения и системы нелинейных уравнений.

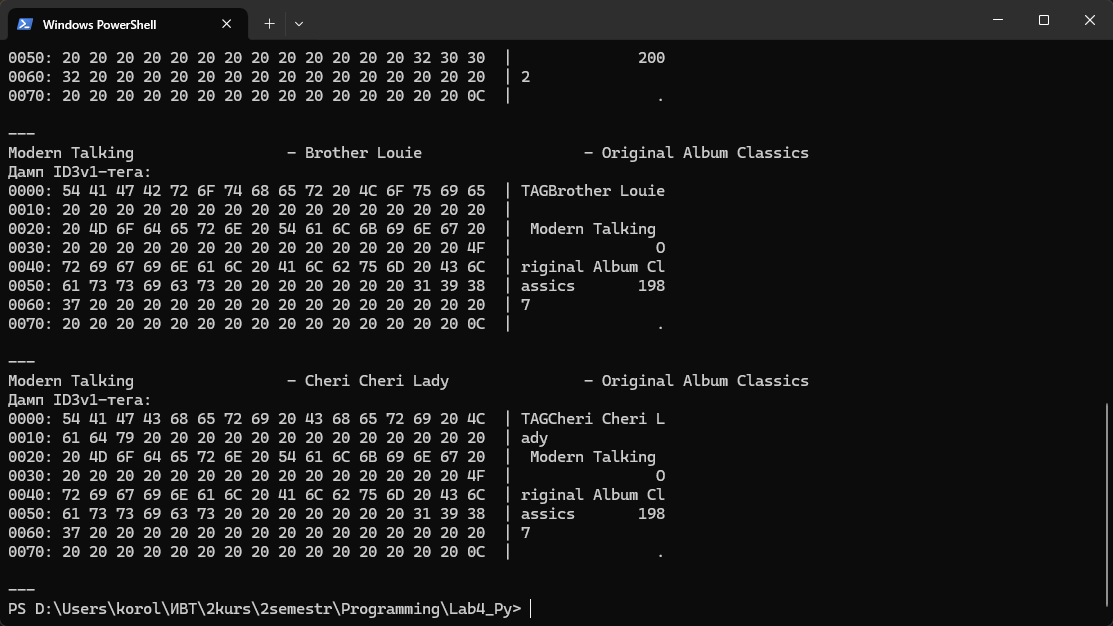
**Ход работы:**

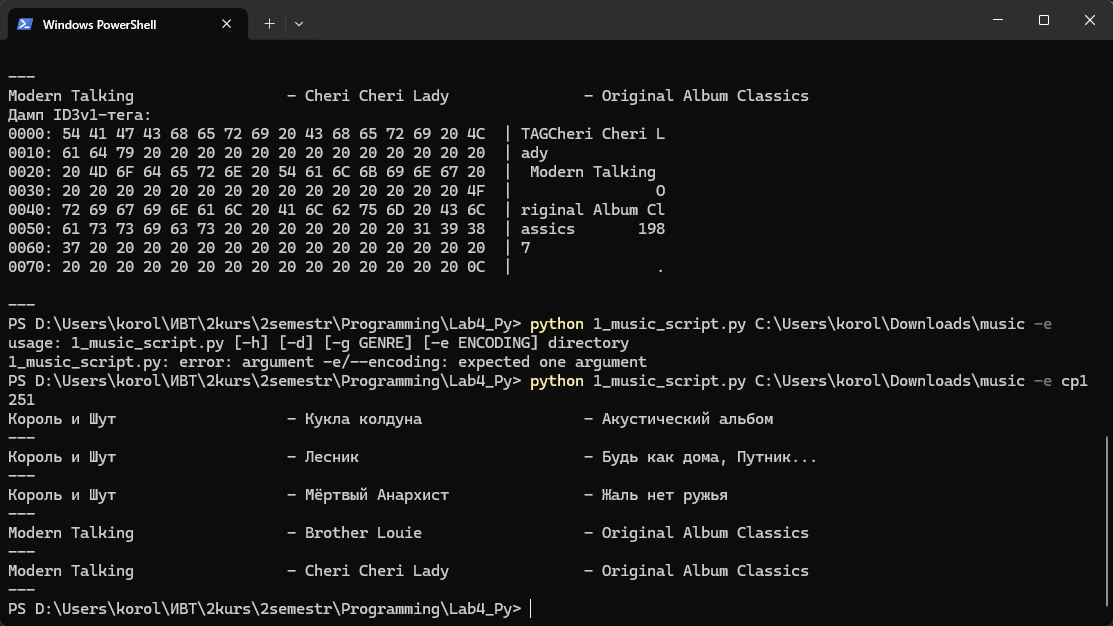
1. Код программы:

*"""  
Скрипт, читающий во всех mp3-файлах указанной  
директории ID3v1-теги и выводящий информацию о каждом файле в  
виде: [имя исполнителя] - [название трека] - [название альбома].  
"""*# !/usr/bin/env python3  
import os  
import argparse  
import chardet  
from typing import Dict, Optional, Tuple  
  
  
def detect\_encoding(data: bytes) -> str:  
 *"""  
 Определяет возможную кодировку данных.  
  
 Args:  
 data: Байты для определения кодировки  
  
 Returns:  
 Строка с названием кодировки  
 """* # Пробуем определить кодировку с помощью chardet  
 detection = chardet.detect(data)  
 encoding = detection['encoding']  
  
 # Если кодировка не обнаружена или confidence низкий, выбираем windows-1251 для кириллицы  
 if not encoding or detection['confidence'] < 0.6:  
 if any(b > 127 for b in data): # Наличие символов вне ASCII может указывать на кириллицу  
 return 'windows-1251'  
 return 'latin1'  
  
 return encoding  
  
  
def decode\_string(byte\_data: bytes) -> str:  
 *"""  
 Декодирует строку с определением кодировки.  
  
 Args:  
 byte\_data: Байты для декодирования  
  
 Returns:  
 Декодированная строка  
 """* # Удаляем нулевые байты в конце  
 clean\_data = byte\_data.rstrip(b'\x00')  
  
 # Если данных нет, возвращаем пустую строку  
 if not clean\_data:  
 return ""  
  
 # Пытаемся определить кодировку  
 encoding = detect\_encoding(clean\_data)  
  
 # Пробуем декодировать с определенной кодировкой  
 try:  
 return clean\_data.decode(encoding)  
 except UnicodeDecodeError:  
 # Если не получилось, пробуем распространенные кодировки для кириллицы  
 for enc in ['windows-1251', 'koi8-r', 'utf-8', 'latin1']:  
 try:  
 return clean\_data.decode(enc)  
 except UnicodeDecodeError:  
 continue  
  
 # Если ничего не помогло, возвращаем строку с заменой непонятных символов  
 return clean\_data.decode('latin1', errors='replace')  
  
  
def get\_id3v1\_tag(file\_path: str) -> Tuple[Optional[Dict], bytes]:  
 *"""  
 Извлекает ID3v1-тег из MP3 файла.  
  
 Args:  
 file\_path: Путь к MP3 файлу  
  
 Returns:  
 Кортеж (словарь с тегами или None, если тег не найден, и сырые данные тега)  
 """* try:  
 with open(file\_path, 'rb') as f:  
 # Перемещаемся к концу файла минус 128 байт (размер ID3v1 тега)  
 f.seek(-128, os.SEEK\_END)  
 tag\_data = f.read(128)  
  
 # Проверяем, что это действительно ID3v1-тег (первые 3 байта должны быть "TAG")  
 if tag\_data[:3] != b'TAG':  
 return None, b''  
  
 # Распаковываем данные с учетом кодировки  
 title = decode\_string(tag\_data[3:33])  
 artist = decode\_string(tag\_data[33:63])  
 album = decode\_string(tag\_data[63:93])  
 year = decode\_string(tag\_data[93:97])  
  
 # Проверяем, содержит ли комментарийный раздел информацию о треке  
 if tag\_data[125] == 0:  
 comment = decode\_string(tag\_data[97:125])  
 track = tag\_data[126]  
 else:  
 comment = decode\_string(tag\_data[97:127])  
 track = 0  
  
 genre = tag\_data[127]  
  
 return {  
 'title': title,  
 'artist': artist,  
 'album': album,  
 'year': year,  
 'comment': comment,  
 'track': track,  
 'genre': genre  
 }, tag\_data  
 except Exception as e:  
 print(f"Ошибка при чтении файла {file\_path}: {e}")  
 return None, b''  
  
  
def encode\_string(text: str, max\_length: int, encoding: str = 'windows-1251') -> bytes:  
 *"""  
 Кодирует строку в определенную кодировку и обрезает до заданной длины.  
  
 Args:  
 text: Текст для кодирования  
 max\_length: Максимальная длина в байтах  
 encoding: Кодировка (по умолчанию windows-1251 для русских текстов)  
  
 Returns:  
 Байты закодированного текста  
 """* try:  
 encoded = text.encode(encoding)  
 except UnicodeEncodeError:  
 # Если не получилось закодировать, пробуем другие кодировки  
 for enc in ['utf-8', 'latin1']:  
 try:  
 encoded = text.encode(enc)  
 break  
 except UnicodeEncodeError:  
 continue  
 else:  
 # Если все плохо, используем latin1 с заменой  
 encoded = text.encode('latin1', errors='replace')  
  
 # Обрезаем до нужной длины  
 return encoded[:max\_length]  
  
  
def write\_id3v1\_tag(file\_path: str, tag\_data: Dict, encoding: str = 'windows-1251') -> bool:  
 *"""  
 Записывает ID3v1-тег в MP3 файл.  
  
 Args:  
 file\_path: Путь к MP3 файлу  
 tag\_data: Словарь с тегами для записи  
 encoding: Кодировка для записи текста (по умолчанию windows-1251)  
  
 Returns:  
 True если запись прошла успешно, иначе False  
 """* try:  
 # Определяем, есть ли уже ID3v1-тег  
 has\_tag = False  
 with open(file\_path, 'rb') as f:  
 f.seek(-128, os.SEEK\_END)  
 has\_tag = f.read(3) == b'TAG'  
  
 with open(file\_path, 'r+b') as f:  
 if has\_tag:  
 # Если тег уже есть, перемещаемся к его началу  
 f.seek(-128, os.SEEK\_END)  
 else:  
 # Если тега нет, перемещаемся в конец файла  
 f.seek(0, os.SEEK\_END)  
  
 # Формируем новый тег  
 new\_tag = bytearray(128)  
  
 # Заголовок "TAG"  
 new\_tag[0:3] = b'TAG'  
  
 # Записываем поля с указанной кодировкой  
 title\_bytes = encode\_string(tag\_data['title'], 30, encoding)  
 new\_tag[3:3 + len(title\_bytes)] = title\_bytes  
  
 artist\_bytes = encode\_string(tag\_data['artist'], 30, encoding)  
 new\_tag[33:33 + len(artist\_bytes)] = artist\_bytes  
  
 album\_bytes = encode\_string(tag\_data['album'], 30, encoding)  
 new\_tag[63:63 + len(album\_bytes)] = album\_bytes  
  
 year\_bytes = encode\_string(tag\_data['year'], 4, encoding)  
 new\_tag[93:93 + len(year\_bytes)] = year\_bytes  
  
 comment\_bytes = encode\_string(tag\_data['comment'], 28, encoding)  
 new\_tag[97:97 + len(comment\_bytes)] = comment\_bytes  
  
 # Zero-byte для трека  
 new\_tag[125] = 0  
  
 # Номер трека  
 new\_tag[126] = tag\_data['track']  
  
 # Жанр  
 new\_tag[127] = tag\_data['genre']  
  
 # Записываем тег  
 f.write(new\_tag)  
  
 return True  
 except Exception as e:  
 print(f"Ошибка при записи тега в файл {file\_path}: {e}")  
 return False  
  
  
def hex\_dump(data: bytes) -> str:  
 *"""  
 Создает 16-ричный дамп данных.  
  
 Args:  
 data: Байты для дампа  
  
 Returns:  
 Строка с 16-ричным дампом  
 """* hex\_lines = []  
 for i in range(0, len(data), 16):  
 chunk = data[i:i + 16]  
 hex\_values = ' '.join(f'{b:02X}' for b in chunk)  
  
 # Добавляем символьное представление  
 ascii\_repr = ''.join(chr(b) if 32 <= b < 127 else '.' for b in chunk)  
  
 # Форматируем строку  
 line = f"{i:04X}: {hex\_values:<48} | {ascii\_repr}"  
 hex\_lines.append(line)  
  
 return '\n'.join(hex\_lines)  
  
  
def main():  
 parser = argparse.ArgumentParser(description='Обработка ID3v1-тегов в MP3 файлах')  
 parser.add\_argument('directory', help='Директория с MP3 файлами')  
 parser.add\_argument('-d', '--dump', action='store\_true', help='Вывести 16-ричный дамп тега')  
 parser.add\_argument('-g', '--genre', type=int, default=255,  
 help='Номер жанра для автоматической простановки (0-255, по умолчанию 255)')  
 parser.add\_argument('-e', '--encoding', default='windows-1251',  
 help='Кодировка для чтения/записи тегов (по умолчанию windows-1251)')  
  
 args = parser.parse\_args()  
  
 # Проверяем, что указанная директория существует  
 if not os.path.isdir(args.directory):  
 print(f"Ошибка: директория {args.directory} не существует")  
 return  
  
 # Проверка диапазона жанра  
 if not 0 <= args.genre <= 255:  
 print(f"Ошибка: номер жанра должен быть в диапазоне 0-255")  
 return  
  
 # Проходим по всем файлам в директории  
 for filename in os.listdir(args.directory):  
 if filename.lower().endswith('.mp3'):  
 file\_path = os.path.join(args.directory, filename)  
  
 # Получаем теги  
 tag\_info, tag\_data = get\_id3v1\_tag(file\_path)  
  
 if tag\_info:  
 # Выводим информацию о файле  
 print(f"{tag\_info['artist']} - {tag\_info['title']} - {tag\_info['album']}")  
  
 # Если запрошен дамп, выводим его  
 if args.dump:  
 print("Дамп ID3v1-тега:")  
 print(hex\_dump(tag\_data))  
 print()  
  
 # Проверяем, нужно ли проставить трек или жанр  
 needs\_update = False  
  
 if tag\_info['track'] == 0:  
 # В качестве примера, мы можем извлечь номер трека из имени файла, если оно начинается с цифр  
 # Например, "01 - Название трека.mp3"  
 try:  
 track\_num = int(filename.split()[0])  
 tag\_info['track'] = min(track\_num, 255) # Ограничиваем 255  
 needs\_update = True  
 print(f"Проставлен номер трека: {tag\_info['track']}")  
 except (ValueError, IndexError):  
 pass  
  
 if tag\_info['genre'] == 255:  
 tag\_info['genre'] = args.genre  
 needs\_update = True  
 print(f"Проставлен жанр: {args.genre}")  
  
 # Если нужно обновить теги, делаем это  
 if needs\_update:  
 if write\_id3v1\_tag(file\_path, tag\_info, args.encoding):  
 print(f"Файл {filename} обновлен успешно")  
 else:  
 print(f"Не удалось обновить теги в файле {filename}")  
  
 print("---")  
 else:  
 print(f"Файл {filename} не содержит ID3v1-тегов")  
 print("---")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод программы:



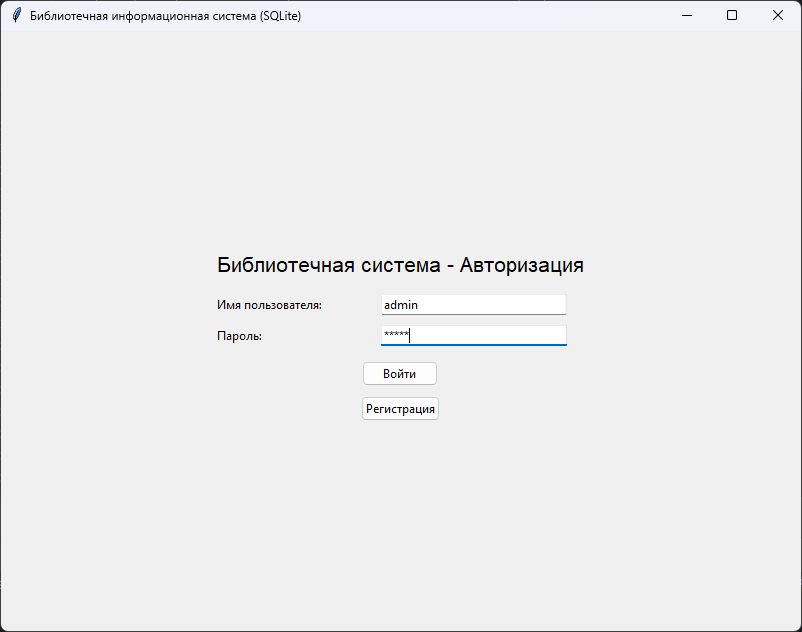


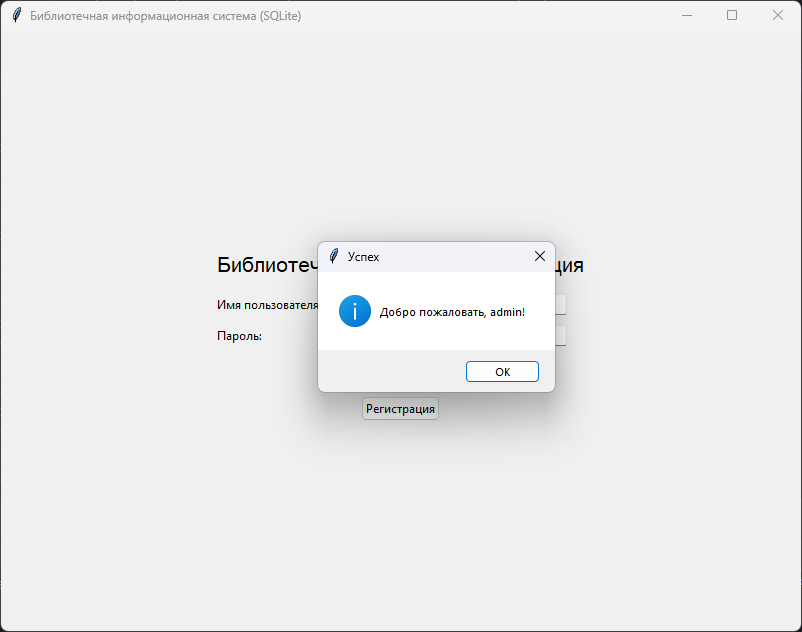


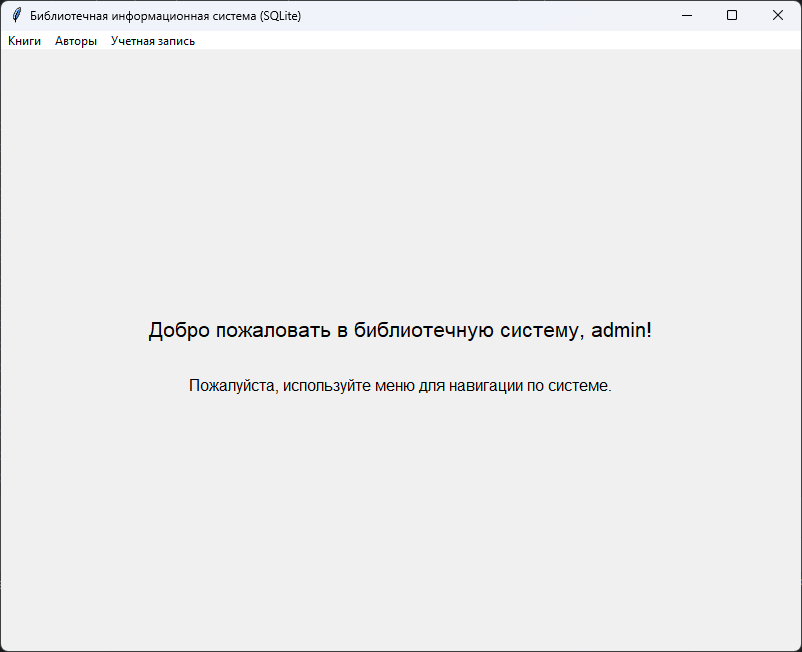
1. Код программы:

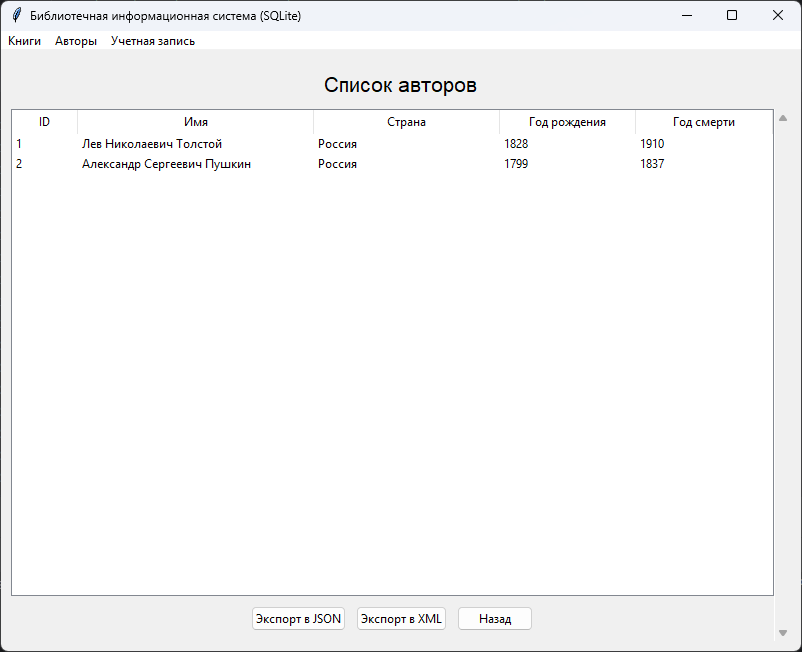
*"""  
Скрипт для информационной системы библиотеки. База  
данных библиотеки включает таблицы «Авторы» с полями «id»,  
«имя», «страна», «годы жизни», и «Книги» с полями «id автора»,  
«название», «количество страниц», «издательство», «год издания»).  
Необходимо производить авторизацию пользователей, логины и  
пароли которых хранятся в отдельной таблице. Пароли должны  
храниться в зашифрованном виде (например, хэш SHA-1 или MD5).  
"""*import sqlite3  
import hashlib  
import json  
import xml.etree.ElementTree as ET  
import tkinter as tk  
from tkinter import ttk, messagebox, filedialog  
import os  
  
class LibrarySystem:  
 def \_\_init\_\_(self):  
 self.conn = None  
 self.cursor = None  
 self.current\_user = None  
  
 # Инициализация БД  
 self.initialize\_database()  
  
 # Создание основного окна приложения  
 self.root = tk.Tk()  
 self.root.title('Библиотечная информационная система (SQLite)')  
 self.root.geometry('800x600')  
  
 # Открытие окна авторизации  
 self.show\_login\_screen()  
  
 def initialize\_database(self):  
 *"""Инициализация БД и создание сущностей"""* self.conn = sqlite3.connect('library.db')  
 self.cursor = self.conn.cursor()  
  
 # Создание таблицы пользователей  
 self.cursor.execute('''  
 CREATE TABLE IF NOT EXISTS users (  
 id INTEGER PRIMARY KEY,  
 username TEXT UNIQUE NOT NULL,  
 password TEXT NOT NULL,  
 is\_admin INTEGER DEFAULT 0  
 )  
 ''')  
  
 # Создание таблицы авторов  
 self.cursor.execute('''  
 CREATE TABLE IF NOT EXISTS authors (  
 id INTEGER PRIMARY KEY,  
 name TEXT NOT NULL,  
 country TEXT,  
 birth\_year INTEGER,  
 death\_year INTEGER  
 )  
 ''')  
  
 # Создание таблицы книг  
 self.cursor.execute('''  
 CREATE TABLE IF NOT EXISTS books (  
 id INTEGER PRIMARY KEY,  
 author\_id INTEGER,  
 title TEXT NOT NULL,  
 pages INTEGER,  
 publisher TEXT,  
 publication\_year INTEGER,  
 FOREIGN KEY (author\_id) REFERENCES authors (id)  
 )  
 ''')  
  
 # Добавление тестового администратора, если таблица пользователей пуста  
 self.cursor.execute("SELECT COUNT(\*) FROM users")  
 if self.cursor.fetchone()[0] == 0:  
 admin\_password = self.hash\_password("admin")  
 self.cursor.execute("INSERT INTO users (username, password, is\_admin) VALUES (?, ?, ?)", ("admin", admin\_password, 1))  
  
 self.conn.commit()  
  
 def hash\_password(self, password):  
 *"""Хэширование пароля с использованием SHA-256"""* return hashlib.sha256(password.encode()).hexdigest()  
  
 def authenticate(self, username, password):  
 *"""Проверка учётных данных пользователя"""* hashed\_password = self.hash\_password(password)  
 self.cursor.execute("SELECT id, is\_admin FROM users WHERE username = ? AND password = ?",  
 (username, hashed\_password))  
 user = self.cursor.fetchone()  
  
 if user:  
 self.current\_user = {"id:": user[0], "username": username, "is\_admin": user[1]}  
 return True  
 return False  
  
 def show\_login\_screen(self):  
 *"""Отображение экрана авторизации"""* # Очистка текущего окна  
 for widget in self.root.winfo\_children():  
 widget.destroy()  
  
 # Создание фрейма авторизации  
 login\_frame = ttk.Frame(self.root, padding='20')  
 login\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(login\_frame, text="Библиотечная система - Авторизация", font=("Arial", 16)).grid(row=0, column=0,  
 columnspan=2,  
 pady=10)  
  
 # Поля ввода  
 ttk.Label(login\_frame, text="Имя пользователя:").grid(row=1, column=0, sticky=tk.W, pady=5)  
 username\_entry = ttk.Entry(login\_frame, width=30)  
 username\_entry.grid(row=1, column=1, pady=5)  
  
 ttk.Label(login\_frame, text="Пароль:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 password\_entry = ttk.Entry(login\_frame, width=30, show="\*")  
 password\_entry.grid(row=2, column=1, pady=5)  
  
 # Кнопка входа  
 ttk.Button(login\_frame, text="Войти",  
 command=lambda: self.login(username\_entry.get(), password\_entry.get())).grid(row=3, column=0,  
 columnspan=2, pady=10)  
  
 # Кнопка регистрации (только для демонстрации)  
 ttk.Button(login\_frame, text="Регистрация",  
 command=self.show\_registration\_screen).grid(row=4, column=0, columnspan=2)  
  
 def show\_registration\_screen(self):  
 *"""Отображение экрана регистрации"""* # Очистка текущего окна  
 for widget in self.root.winfo\_children():  
 widget.destroy()  
  
 # Создание фрейма регистрации  
 reg\_frame = ttk.Frame(self.root, padding="20")  
 reg\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(reg\_frame, text="Библиотечная система - Регистрация", font=("Arial", 16)).grid(row=0, column=0,  
 columnspan=2, pady=10)  
  
 # Поля ввода  
 ttk.Label(reg\_frame, text="Имя пользователя:").grid(row=1, column=0, sticky=tk.W, pady=5)  
 username\_entry = ttk.Entry(reg\_frame, width=30)  
 username\_entry.grid(row=1, column=1, pady=5)  
  
 ttk.Label(reg\_frame, text="Пароль:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 password\_entry = ttk.Entry(reg\_frame, width=30, show="\*")  
 password\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Label(reg\_frame, text="Подтверждение пароля:").grid(row=3, column=0, sticky=tk.W, pady=5)  
 confirm\_entry = ttk.Entry(reg\_frame, width=30, show="\*")  
 confirm\_entry.grid(row=3, column=1, pady=5)  
  
 # Кнопка регистрации  
 ttk.Button(reg\_frame, text="Зарегистрироваться",  
 command=lambda: self.register\_user(username\_entry.get(), password\_entry.get(),  
 confirm\_entry.get())).grid(row=4, column=0, columnspan=2, pady=10)  
  
 # Кнопка возврата  
 ttk.Button(reg\_frame, text="Вернуться к авторизации",  
 command=self.show\_login\_screen).grid(row=5, column=0, columnspan=2)  
  
 def register\_user(self, username, password, confirm\_password):  
 *"""Регистрация нового пользователя"""* if not username or not password:  
 messagebox.showerror("Ошибка", "Необходимо заполнить все поля")  
 return  
  
 if password != confirm\_password:  
 messagebox.showerror("Ошибка", "Пароли не совпадают")  
 return  
  
 try:  
 hashed\_password = self.hash\_password(password)  
 self.cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)",  
 (username, hashed\_password))  
 self.conn.commit()  
 messagebox.showinfo("Успех", "Пользователь успешно зарегистрирован")  
 self.show\_login\_screen()  
 except sqlite3.IntegrityError:  
 messagebox.showerror("Ошибка", "Пользователь с таким именем уже существует")  
  
 def login(self, username, password):  
 *"""Обработка входа пользователя"""* if not username or not password:  
 messagebox.showerror("Ошибка", "Введите имя пользователя и пароль")  
 return  
  
 if self.authenticate(username, password):  
 messagebox.showinfo("Успех", f"Добро пожаловать, {username}!")  
 self.show\_main\_menu()  
 else:  
 messagebox.showerror("Ошибка", "Неверное имя пользователя или пароль")  
  
 def show\_main\_menu(self):  
 *"""Отображение главного меню программы"""* # Очистка текущего окна  
 for widget in self.root.winfo\_children():  
 widget.destroy()  
  
 # Создание главного меню  
 menubar = tk.Menu(self.root)  
 self.root.config(menu=menubar)  
  
 # Меню "Книги"  
 book\_menu = tk.Menu(menubar, tearoff=0)  
 book\_menu.add\_command(label="Список книг", command=self.show\_books)  
 book\_menu.add\_command(label="Добавить книгу", command=self.show\_add\_book)  
 menubar.add\_cascade(label="Книги", menu=book\_menu)  
  
 # Меню "Авторы"  
 author\_menu = tk.Menu(menubar, tearoff=0)  
 author\_menu.add\_command(label="Список авторов", command=self.show\_authors)  
 author\_menu.add\_command(label="Добавить автора", command=self.show\_add\_author)  
 author\_menu.add\_command(label="Импорт автора из файла", command=self.show\_import\_author)  
 menubar.add\_cascade(label="Авторы", menu=author\_menu)  
  
 # Меню учетной записи  
 account\_menu = tk.Menu(menubar, tearoff=0)  
 account\_menu.add\_command(label="Выход", command=self.show\_login\_screen)  
 menubar.add\_cascade(label="Учетная запись", menu=account\_menu)  
  
 # Создание начального фрейма для приветствия  
 main\_frame = ttk.Frame(self.root, padding="20")  
 main\_frame.pack(expand=True)  
  
 welcome\_label = ttk.Label(main\_frame,  
 text=f"Добро пожаловать в библиотечную систему, {self.current\_user['username']}!",  
 font=("Arial", 16))  
 welcome\_label.pack(pady=20)  
  
 instruction\_label = ttk.Label(main\_frame,  
 text="Пожалуйста, используйте меню для навигации по системе.",  
 font=("Arial", 12))  
 instruction\_label.pack(pady=10)  
  
 def show\_books(self):  
 *"""Отображение списка всех книг"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения книг  
 books\_frame = ttk.Frame(self.root, padding="10")  
 books\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(books\_frame, text="Список книг", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения книг  
 columns = ('id', 'title', 'author', 'pages', 'publisher', 'year')  
 tree = ttk.Treeview(books\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('title', text='Название')  
 tree.heading('author', text='Автор')  
 tree.heading('pages', text='Страниц')  
 tree.heading('publisher', text='Издательство')  
 tree.heading('year', text='Год издания')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('title', width=200)  
 tree.column('author', width=150)  
 tree.column('pages', width=80)  
 tree.column('publisher', width=150)  
 tree.column('year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(books\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Получение данных из БД  
 self.cursor.execute('''  
 SELECT books.id, books.title, authors.name, books.pages, books.publisher, books.publication\_year  
 FROM books  
 LEFT JOIN authors ON books.author\_id = authors.id  
 ''')  
  
 books = self.cursor.fetchall()  
  
 # Заполнение таблицы данными  
 for book in books:  
 tree.insert('', tk.END, values=book)  
  
 # Добавление кнопки возврата  
 ttk.Button(books\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_authors(self):  
 *"""Отображение списка всех авторов"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения авторов  
 authors\_frame = ttk.Frame(self.root, padding="10")  
 authors\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(authors\_frame, text="Список авторов", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения авторов  
 columns = ('id', 'name', 'country', 'birth\_year', 'death\_year')  
 tree = ttk.Treeview(authors\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('name', text='Имя')  
 tree.heading('country', text='Страна')  
 tree.heading('birth\_year', text='Год рождения')  
 tree.heading('death\_year', text='Год смерти')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('name', width=200)  
 tree.column('country', width=150)  
 tree.column('birth\_year', width=100)  
 tree.column('death\_year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(authors\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Получение данных из БД  
 self.cursor.execute('SELECT id, name, country, birth\_year, death\_year FROM authors')  
 authors = self.cursor.fetchall()  
  
 # Заполнение таблицы данными  
 for author in authors:  
 tree.insert('', tk.END, values=author)  
  
 # Добавление кнопок  
 button\_frame = ttk.Frame(authors\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Экспорт в JSON",  
 command=lambda: self.export\_author\_to\_json(tree)).pack(side=tk.LEFT, padx=5)  
 ttk.Button(button\_frame, text="Экспорт в XML",  
 command=lambda: self.export\_author\_to\_xml(tree)).pack(side=tk.LEFT, padx=5)  
 ttk.Button(button\_frame, text="Назад", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def export\_author\_to\_json(self, tree):  
 *"""Экспорт выбранного автора в формат JSON"""* # Получение выбранного элемента  
 selected\_item = tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Пожалуйста, выберите автора для экспорта")  
 return  
  
 author\_id = tree.item(selected\_item, "values")[0]  
  
 # Получение данных автора  
 self.cursor.execute('SELECT name, country, birth\_year, death\_year FROM authors WHERE id = ?', (author\_id,))  
 author = self.cursor.fetchone()  
  
 if not author:  
 messagebox.showerror("Ошибка", "Автор не найден")  
 return  
  
 # Создание JSON объекта  
 author\_data = {  
 "name": author[0],  
 "country": author[1],  
 "years": [author[2], author[3]] if author[2] and author[3] else []  
 }  
  
 # Выбор места сохранения файла  
 file\_path = filedialog.asksaveasfilename(  
 defaultextension=".json",  
 filetypes=[("JSON файлы", "\*.json"), ("Все файлы", "\*.\*")]  
 )  
  
 if not file\_path:  
 return  
  
 try:  
 with open(file\_path, 'w', encoding='utf-8') as f:  
 json.dump(author\_data, f, ensure\_ascii=False, indent=4)  
 messagebox.showinfo("Успех", f"Данные автора успешно экспортированы в {file\_path}")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить файл: {str(e)}")  
  
 def export\_author\_to\_xml(self, tree):  
 *"""Экспорт выбранного автора в формат XML"""* # Получение выбранного элемента  
 selected\_item = tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Пожалуйста, выберите автора для экспорта")  
 return  
  
 author\_id = tree.item(selected\_item, "values")[0]  
  
 # Получение данных автора  
 self.cursor.execute('SELECT name, country, birth\_year, death\_year FROM authors WHERE id = ?', (author\_id,))  
 author = self.cursor.fetchone()  
  
 if not author:  
 messagebox.showerror("Ошибка", "Автор не найден")  
 return  
  
 # Создание XML структуры  
 root = ET.Element("author")  
  
 name\_elem = ET.SubElement(root, "name")  
 name\_elem.text = author[0]  
  
 country\_elem = ET.SubElement(root, "country")  
 country\_elem.text = author[1] if author[1] else ""  
  
 years\_elem = ET.SubElement(root, "years")  
 if author[2]:  
 years\_elem.set("born", str(author[2]))  
 if author[3]:  
 years\_elem.set("died", str(author[3]))  
  
 # Преобразование в строку XML  
 xml\_str = ET.tostring(root, encoding='utf-8')  
  
 # Добавление форматирования XML  
 import xml.dom.minidom  
 xml\_pretty = xml.dom.minidom.parseString(xml\_str).toprettyxml(indent=" ")  
  
 # Выбор места сохранения файла  
 file\_path = filedialog.asksaveasfilename(  
 defaultextension=".xml",  
 filetypes=[("XML файлы", "\*.xml"), ("Все файлы", "\*.\*")]  
 )  
  
 if not file\_path:  
 return  
  
 try:  
 with open(file\_path, 'w', encoding='utf-8') as f:  
 f.write(xml\_pretty)  
 messagebox.showinfo("Успех", f"Данные автора успешно экспортированы в {file\_path}")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось сохранить файл: {str(e)}")  
  
 def show\_add\_book(self):  
 *"""Отображение формы добавления новой книги"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для добавления книги  
 book\_frame = ttk.Frame(self.root, padding="20")  
 book\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(book\_frame, text="Добавление новой книги", font=("Arial", 16)).grid(row=0, column=0, columnspan=2,  
 pady=10)  
  
 # Выбор автора  
 ttk.Label(book\_frame, text="Автор:").grid(row=1, column=0, sticky=tk.W, pady=5)  
  
 # Получение списка авторов из БД  
 self.cursor.execute('SELECT id, name FROM authors')  
 authors = self.cursor.fetchall()  
  
 # Создание комбобокса с авторами  
 author\_var = tk.StringVar()  
 author\_combo = ttk.Combobox(book\_frame, textvariable=author\_var, width=30)  
 author\_combo['values'] = [f"{author[0]}: {author[1]}" for author in authors]  
 author\_combo.grid(row=1, column=1, pady=5)  
  
 # Поля для ввода данных о книге  
 ttk.Label(book\_frame, text="Название:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 title\_entry = ttk.Entry(book\_frame, width=30)  
 title\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Label(book\_frame, text="Количество страниц:").grid(row=3, column=0, sticky=tk.W, pady=5)  
 pages\_entry = ttk.Entry(book\_frame, width=30)  
 pages\_entry.grid(row=3, column=1, pady=5)  
  
 ttk.Label(book\_frame, text="Издательство:").grid(row=4, column=0, sticky=tk.W, pady=5)  
 publisher\_entry = ttk.Entry(book\_frame, width=30)  
 publisher\_entry.grid(row=4, column=1, pady=5)  
  
 ttk.Label(book\_frame, text="Год издания:").grid(row=5, column=0, sticky=tk.W, pady=5)  
 year\_entry = ttk.Entry(book\_frame, width=30)  
 year\_entry.grid(row=5, column=1, pady=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(book\_frame)  
 button\_frame.grid(row=6, column=0, columnspan=2, pady=10)  
  
 ttk.Button(button\_frame, text="Добавить",  
 command=lambda: self.add\_book(  
 author\_var.get(), title\_entry.get(), pages\_entry.get(),  
 publisher\_entry.get(), year\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Отмена", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def add\_book(self, author, title, pages, publisher, year):  
 *"""Добавление новой книги в БД"""* if not title:  
 messagebox.showerror("Ошибка", "Название книги обязательно для заполнения")  
 return  
  
 try:  
 # Парсинг ID автора из строки формата "id: name"  
 author\_id = int(author.split(':')[0]) if author else None  
 pages = int(pages) if pages else None  
 year = int(year) if year else None  
  
 self.cursor.execute(  
 "INSERT INTO books (author\_id, title, pages, publisher, publication\_year) VALUES (?, ?, ?, ?, ?)",  
 (author\_id, title, pages, publisher, year)  
 )  
 self.conn.commit()  
  
 messagebox.showinfo("Успех", "Книга успешно добавлена")  
 self.show\_books()  
 except ValueError:  
 messagebox.showerror("Ошибка", "Проверьте правильность ввода числовых значений")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось добавить книгу: {str(e)}")  
  
 def show\_add\_author(self):  
 *"""Отображение формы добавления нового автора"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для добавления автора  
 author\_frame = ttk.Frame(self.root, padding="20")  
 author\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(author\_frame, text="Добавление нового автора", font=("Arial", 16)).grid(row=0, column=0, columnspan=2,  
 pady=10)  
  
 # Поля для ввода данных об авторе  
 ttk.Label(author\_frame, text="Имя:").grid(row=1, column=0, sticky=tk.W, pady=5)  
 name\_entry = ttk.Entry(author\_frame, width=30)  
 name\_entry.grid(row=1, column=1, pady=5)  
  
 ttk.Label(author\_frame, text="Страна:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 country\_entry = ttk.Entry(author\_frame, width=30)  
 country\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Label(author\_frame, text="Год рождения:").grid(row=3, column=0, sticky=tk.W, pady=5)  
 birth\_entry = ttk.Entry(author\_frame, width=30)  
 birth\_entry.grid(row=3, column=1, pady=5)  
  
 ttk.Label(author\_frame, text="Год смерти:").grid(row=4, column=0, sticky=tk.W, pady=5)  
 death\_entry = ttk.Entry(author\_frame, width=30)  
 death\_entry.grid(row=4, column=1, pady=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(author\_frame)  
 button\_frame.grid(row=5, column=0, columnspan=2, pady=10)  
  
 ttk.Button(button\_frame, text="Добавить",  
 command=lambda: self.add\_author(  
 name\_entry.get(), country\_entry.get(), birth\_entry.get(), death\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Отмена", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def add\_author(self, name, country, birth\_year, death\_year):  
 *"""Добавление нового автора в БД"""* if not name:  
 messagebox.showerror("Ошибка", "Имя автора обязательно для заполнения")  
 return  
  
 try:  
 birth\_year = int(birth\_year) if birth\_year else None  
 death\_year = int(death\_year) if death\_year else None  
  
 self.cursor.execute(  
 "INSERT INTO authors (name, country, birth\_year, death\_year) VALUES (?, ?, ?, ?)",  
 (name, country, birth\_year, death\_year)  
 )  
 self.conn.commit()  
  
 messagebox.showinfo("Успех", "Автор успешно добавлен")  
 self.show\_authors()  
 except ValueError:  
 messagebox.showerror("Ошибка", "Проверьте правильность ввода годов жизни")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось добавить автора: {str(e)}")  
  
 def show\_import\_author(self):  
 *"""Отображение формы импорта автора из файла"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для импорта автора  
 import\_frame = ttk.Frame(self.root, padding="20")  
 import\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(import\_frame, text="Импорт автора из файла", font=("Arial", 16)).grid(row=0, column=0, columnspan=3,  
 pady=10)  
  
 # Выбор формата файла  
 ttk.Label(import\_frame, text="Формат файла:").grid(row=1, column=0, sticky=tk.W, pady=5)  
  
 format\_var = tk.StringVar(value="json")  
 ttk.Radiobutton(import\_frame, text="JSON", variable=format\_var, value="json").grid(row=1, column=1, pady=5)  
 ttk.Radiobutton(import\_frame, text="XML", variable=format\_var, value="xml").grid(row=1, column=2, pady=5)  
  
 # Выбор файла  
 ttk.Label(import\_frame, text="Файл:").grid(row=2, column=0, sticky=tk.W, pady=5)  
  
 file\_path\_var = tk.StringVar()  
 file\_entry = ttk.Entry(import\_frame, textvariable=file\_path\_var, width=40)  
 file\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Button(import\_frame, text="Обзор", command=lambda: self.browse\_file(file\_path\_var, format\_var.get())).grid(  
 row=2, column=2, pady=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(import\_frame)  
 button\_frame.grid(row=3, column=0, columnspan=3, pady=10)  
  
 ttk.Button(button\_frame, text="Импортировать",  
 command=lambda: self.import\_author\_from\_file(file\_path\_var.get(), format\_var.get())  
 ).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Отмена", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def browse\_file(self, file\_path\_var, format\_type):  
 *"""Выбор файла для импорта"""* filetypes = [("JSON файлы", "\*.json"), ("XML файлы", "\*.xml"), ("Все файлы", "\*.\*")]  
 if format\_type == "json":  
 filetypes = [("JSON файлы", "\*.json"), ("Все файлы", "\*.\*")]  
 elif format\_type == "xml":  
 filetypes = [("XML файлы", "\*.xml"), ("Все файлы", "\*.\*")]  
  
 filename = filedialog.askopenfilename(  
 title="Выберите файл",  
 filetypes=filetypes  
 )  
  
 if filename:  
 file\_path\_var.set(filename)  
  
 def import\_author\_from\_file(self, file\_path, format\_type):  
 *"""Импорт автора из файла JSON или XML"""* if not file\_path:  
 messagebox.showerror("Ошибка", "Выберите файл для импорта")  
 return  
  
 try:  
 author\_data = None  
  
 if format\_type == "json":  
 author\_data = self.parse\_author\_from\_json(file\_path)  
 elif format\_type == "xml":  
 author\_data = self.parse\_author\_from\_xml(file\_path)  
 else:  
 messagebox.showerror("Ошибка", "Неподдерживаемый формат файла")  
 return  
  
 if author\_data:  
 name = author\_data.get('name')  
 country = author\_data.get('country')  
 birth\_year = author\_data.get('birth\_year')  
 death\_year = author\_data.get('death\_year')  
  
 # Добавление автора в БД  
 self.cursor.execute(  
 "INSERT INTO authors (name, country, birth\_year, death\_year) VALUES (?, ?, ?, ?)",  
 (name, country, birth\_year, death\_year)  
 )  
 self.conn.commit()  
  
 messagebox.showinfo("Успех", "Автор успешно импортирован")  
 self.show\_authors()  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось импортировать автора: {str(e)}")  
  
 def parse\_author\_from\_json(self, file\_path):  
 *"""Парсинг данных автора из JSON файла"""* with open(file\_path, 'r', encoding='utf-8') as f:  
 data = json.load(f)  
  
 author\_data = {  
 'name': data.get('name'),  
 'country': data.get('country'),  
 'birth\_year': None,  
 'death\_year': None  
 }  
  
 # Обработка годов жизни  
 years = data.get('years', [])  
 if len(years) >= 2:  
 author\_data['birth\_year'] = years[0]  
 author\_data['death\_year'] = years[1]  
  
 return author\_data  
  
 def parse\_author\_from\_xml(self, file\_path):  
 *"""Парсинг данных автора из XML файла"""* tree = ET.parse(file\_path)  
 root = tree.getroot()  
  
 author\_data = {  
 'name': None,  
 'country': None,  
 'birth\_year': None,  
 'death\_year': None  
 }  
  
 # Получение имени автора  
 name\_elem = root.find('name')  
 if name\_elem is not None and name\_elem.text:  
 author\_data['name'] = name\_elem.text  
  
 # Получение страны автора  
 country\_elem = root.find('country')  
 if country\_elem is not None and country\_elem.text:  
 author\_data['country'] = country\_elem.text  
  
 # Получение годов жизни  
 years\_elem = root.find('years')  
 if years\_elem is not None:  
 if 'born' in years\_elem.attrib:  
 try:  
 author\_data['birth\_year'] = int(years\_elem.attrib['born'])  
 except ValueError:  
 pass  
  
 if 'died' in years\_elem.attrib:  
 try:  
 author\_data['death\_year'] = int(years\_elem.attrib['died'])  
 except ValueError:  
 pass  
  
 return author\_data  
  
 def clear\_workspace(self):  
 *"""Очистка рабочей области, сохраняя меню"""* # Сохраняем главное меню  
 menu = self.root.winfo\_children()[0] if self.root.winfo\_children() and isinstance(self.root.winfo\_children()[0],  
 tk.Menu) else None  
  
 # Удаляем все виджеты, кроме меню  
 for widget in self.root.winfo\_children():  
 if widget != menu:  
 widget.destroy()  
  
 def run(self):  
 *"""Запуск приложения"""* self.root.mainloop()  
  
 # Закрытие соединения с БД при выходе  
 if self.conn:  
 self.conn.close()  
  
app = LibrarySystem()  
app.run()

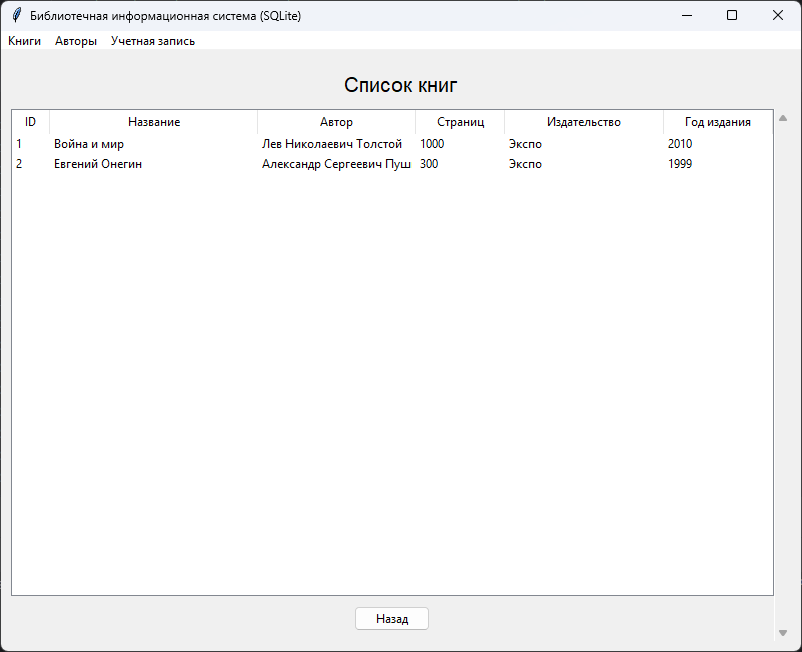
Вывод программы:

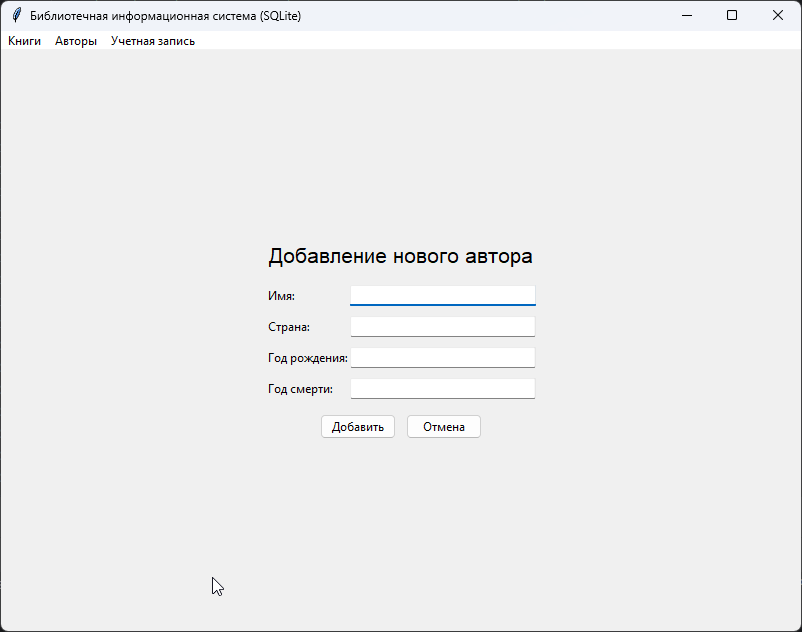


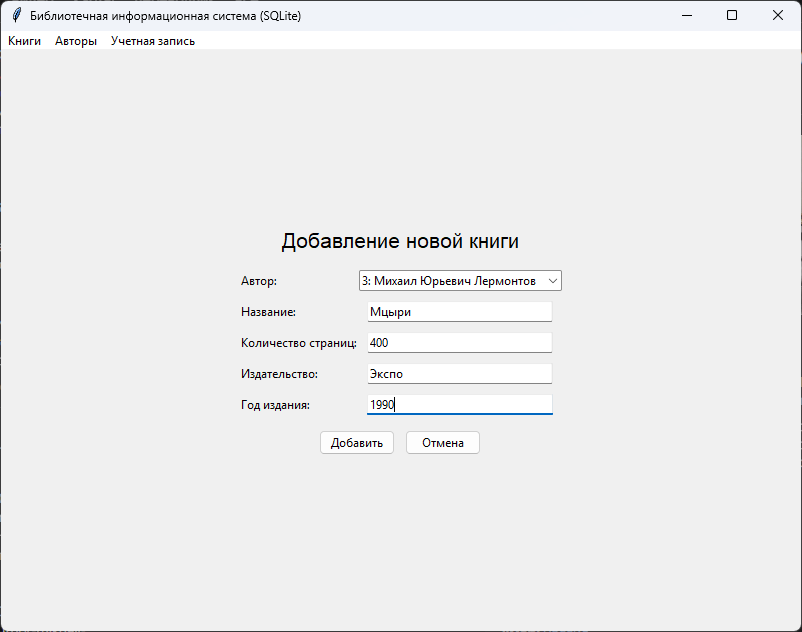


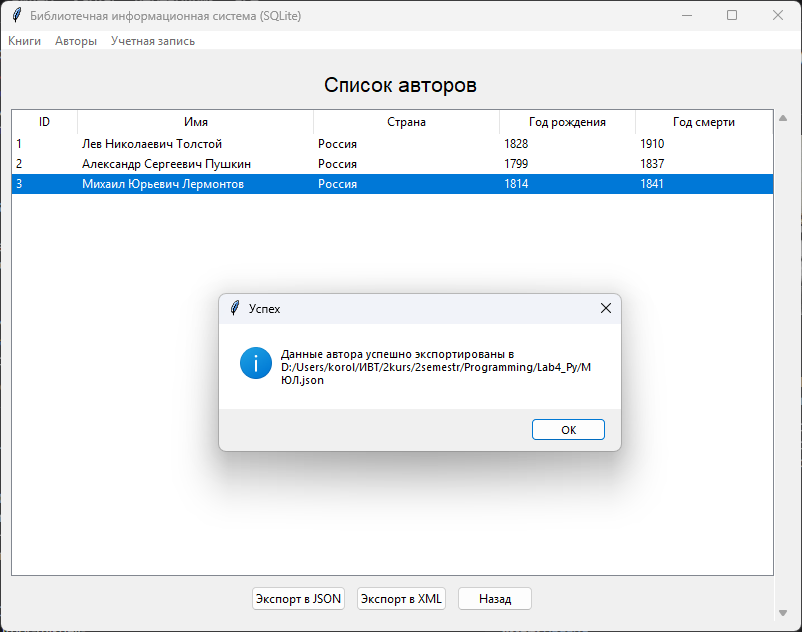


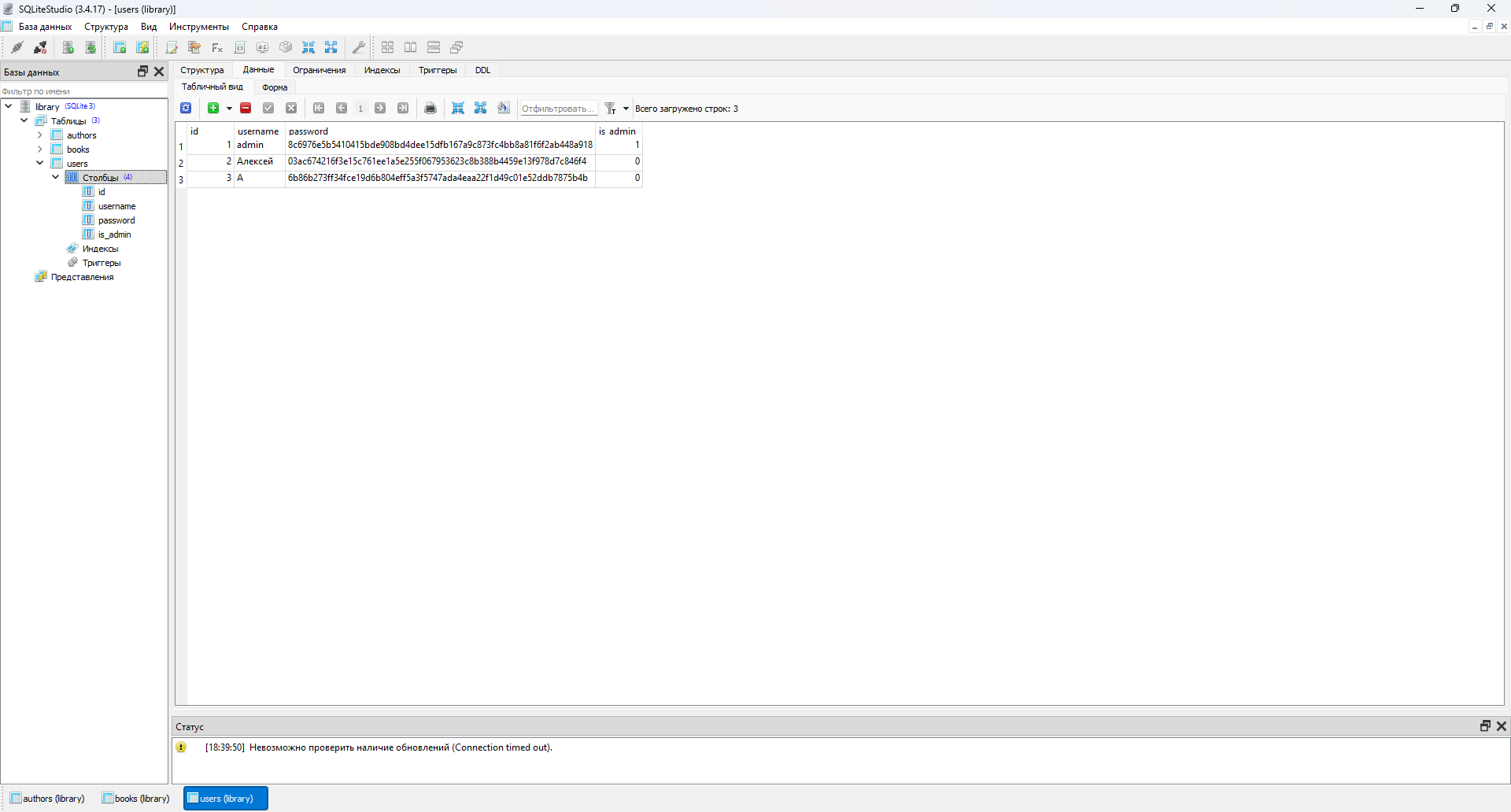








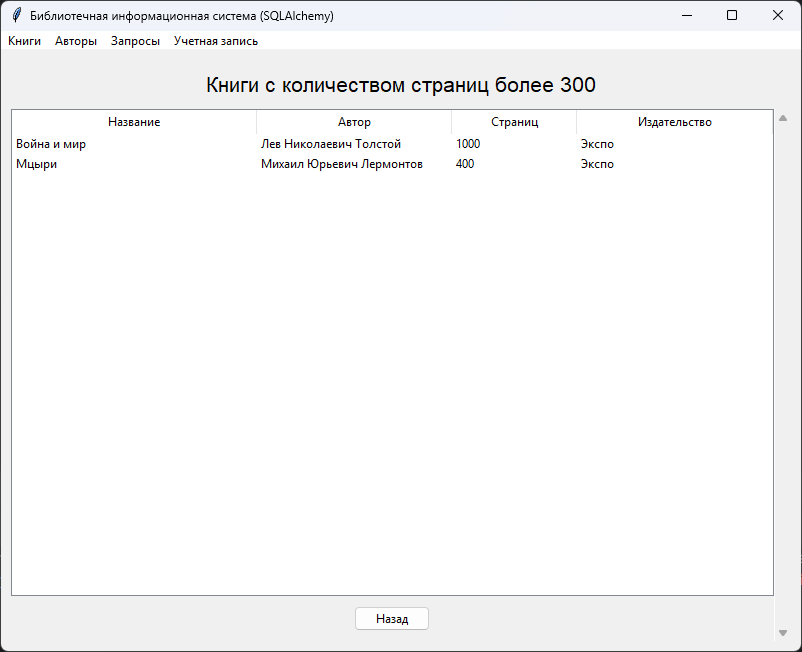


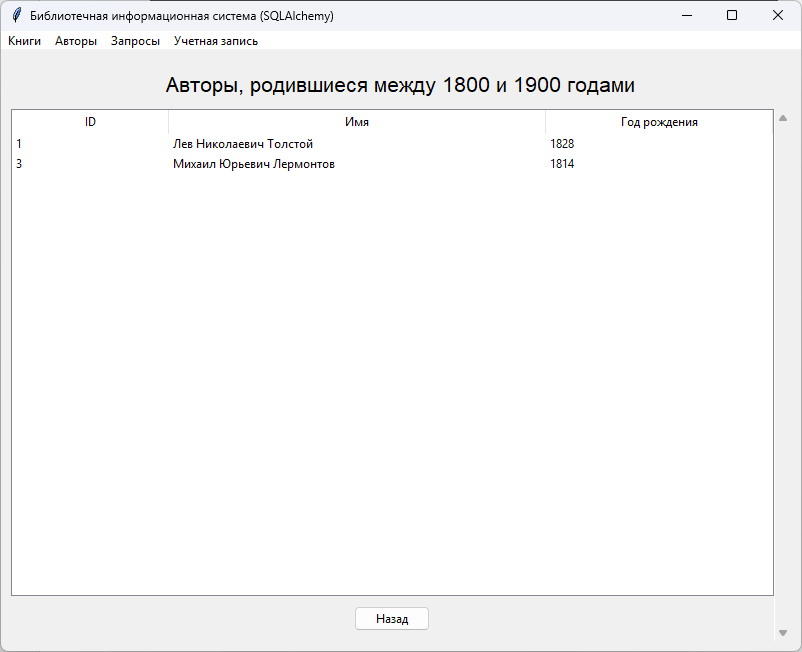


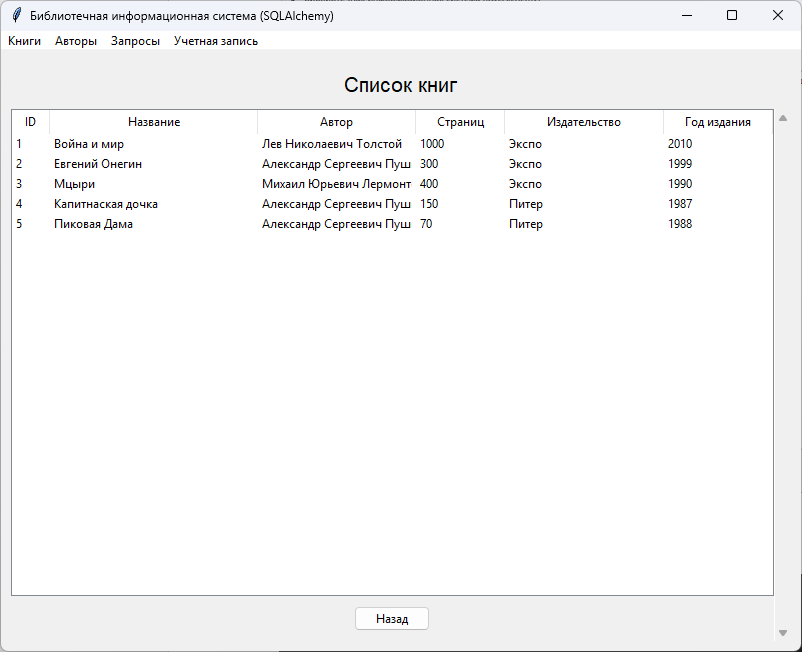
1. Код программы:

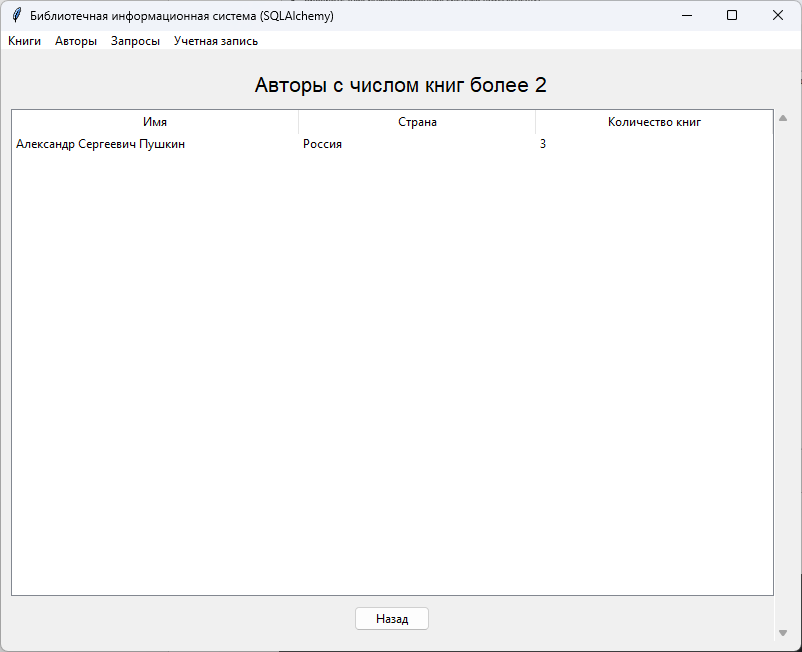
*"""  
Реализация библиотечной системы с использованием SQLAlchemy  
"""*import hashlib  
import json  
import xml.etree.ElementTree as ET  
import tkinter as tk  
from tkinter import ttk, messagebox, filedialog  
  
# Импорт необходимых модулей SQLAlchemy  
from sqlalchemy import create\_engine, Column, Integer, String, ForeignKey, Table, func  
from sqlalchemy.ext.declarative import declarative\_base  
from sqlalchemy.orm import sessionmaker, relationship  
from sqlalchemy.sql import text  
  
# Создаем базовый класс для декларативных определений  
Base = declarative\_base()  
  
  
# Определение моделей SQLAlchemy  
class User(Base):  
 \_\_tablename\_\_ = 'users'  
  
 id = Column(Integer, primary\_key=True)  
 username = Column(String, unique=True, nullable=False)  
 password = Column(String, nullable=False)  
 is\_admin = Column(Integer, default=0)  
  
  
class Author(Base):  
 \_\_tablename\_\_ = 'authors'  
  
 id = Column(Integer, primary\_key=True)  
 name = Column(String, nullable=False)  
 country = Column(String)  
 birth\_year = Column(Integer)  
 death\_year = Column(Integer)  
  
 # Отношение один-ко-многим с книгами  
 books = relationship("Book", back\_populates="author")  
  
  
class Book(Base):  
 \_\_tablename\_\_ = 'books'  
  
 id = Column(Integer, primary\_key=True)  
 author\_id = Column(Integer, ForeignKey('authors.id'))  
 title = Column(String, nullable=False)  
 pages = Column(Integer)  
 publisher = Column(String)  
 publication\_year = Column(Integer)  
  
 # Отношение многие-к-одному с автором  
 author = relationship("Author", back\_populates="books")  
  
  
class LibrarySystem:  
 def \_\_init\_\_(self):  
 self.current\_user = None  
  
 # Инициализация БД с использованием SQLAlchemy  
 self.initialize\_database()  
  
 # Создание основного окна приложения  
 self.root = tk.Tk()  
 self.root.title('Библиотечная информационная система (SQLAlchemy)')  
 self.root.geometry('800x600')  
  
 # Открытие окна авторизации  
 self.show\_login\_screen()  
  
 def initialize\_database(self):  
 *"""Инициализация БД с использованием SQLAlchemy"""* # Создание подключения к базе данных SQLite  
 self.engine = create\_engine('sqlite:///library.db')  
  
 # Создание таблиц  
 Base.metadata.create\_all(self.engine)  
  
 # Создание сессии  
 Session = sessionmaker(bind=self.engine)  
 self.session = Session()  
  
 # Добавление тестового администратора, если таблица пользователей пуста  
 admin\_count = self.session.query(User).count()  
 if admin\_count == 0:  
 admin\_password = self.hash\_password("admin")  
 admin\_user = User(username="admin", password=admin\_password, is\_admin=1)  
 self.session.add(admin\_user)  
 self.session.commit()  
  
 def hash\_password(self, password):  
 *"""Хэширование пароля с использованием SHA-256"""* return hashlib.sha256(password.encode()).hexdigest()  
  
 def authenticate(self, username, password):  
 *"""Проверка учётных данных пользователя с использованием SQLAlchemy"""* hashed\_password = self.hash\_password(password)  
 user = self.session.query(User).filter\_by(username=username, password=hashed\_password).first()  
  
 if user:  
 self.current\_user = {"id": user.id, "username": username, "is\_admin": user.is\_admin}  
 return True  
 return False  
  
 def register\_user(self, username, password, confirm\_password):  
 *"""Регистрация нового пользователя с использованием SQLAlchemy"""* if not username or not password:  
 messagebox.showerror("Ошибка", "Необходимо заполнить все поля")  
 return  
  
 if password != confirm\_password:  
 messagebox.showerror("Ошибка", "Пароли не совпадают")  
 return  
  
 try:  
 hashed\_password = self.hash\_password(password)  
 new\_user = User(username=username, password=hashed\_password)  
 self.session.add(new\_user)  
 self.session.commit()  
 messagebox.showinfo("Успех", "Пользователь успешно зарегистрирован")  
 self.show\_login\_screen()  
 except Exception as e:  
 self.session.rollback()  
 messagebox.showerror("Ошибка", f"Пользователь с таким именем уже существует: {str(e)}")  
  
 def show\_books(self):  
 *"""Отображение списка всех книг с использованием SQLAlchemy"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения книг  
 books\_frame = ttk.Frame(self.root, padding="10")  
 books\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(books\_frame, text="Список книг", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения книг  
 columns = ('id', 'title', 'author', 'pages', 'publisher', 'year')  
 tree = ttk.Treeview(books\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('title', text='Название')  
 tree.heading('author', text='Автор')  
 tree.heading('pages', text='Страниц')  
 tree.heading('publisher', text='Издательство')  
 tree.heading('year', text='Год издания')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('title', width=200)  
 tree.column('author', width=150)  
 tree.column('pages', width=80)  
 tree.column('publisher', width=150)  
 tree.column('year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(books\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Получение данных из БД с использованием SQLAlchemy  
 books = self.session.query(Book, Author.name).join(Author, Book.author\_id == Author.id, isouter=True).all()  
  
 # Заполнение таблицы данными  
 for book, author\_name in books:  
 tree.insert('', tk.END,  
 values=(book.id, book.title, author\_name, book.pages, book.publisher, book.publication\_year))  
  
 # Добавление кнопки возврата  
 ttk.Button(books\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_authors(self):  
 *"""Отображение списка всех авторов с использованием SQLAlchemy"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения авторов  
 authors\_frame = ttk.Frame(self.root, padding="10")  
 authors\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(authors\_frame, text="Список авторов", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения авторов  
 columns = ('id', 'name', 'country', 'birth\_year', 'death\_year')  
 tree = ttk.Treeview(authors\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('name', text='Имя')  
 tree.heading('country', text='Страна')  
 tree.heading('birth\_year', text='Год рождения')  
 tree.heading('death\_year', text='Год смерти')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('name', width=200)  
 tree.column('country', width=150)  
 tree.column('birth\_year', width=100)  
 tree.column('death\_year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(authors\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Получение данных из БД с использованием SQLAlchemy  
 authors = self.session.query(Author).all()  
  
 # Заполнение таблицы данными  
 for author in authors:  
 tree.insert('', tk.END,  
 values=(author.id, author.name, author.country, author.birth\_year, author.death\_year))  
  
 # Добавление кнопок  
 button\_frame = ttk.Frame(authors\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Экспорт в JSON",  
 command=lambda: self.export\_author\_to\_json(tree)).pack(side=tk.LEFT, padx=5)  
 ttk.Button(button\_frame, text="Экспорт в XML",  
 command=lambda: self.export\_author\_to\_xml(tree)).pack(side=tk.LEFT, padx=5)  
 ttk.Button(button\_frame, text="Назад", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def add\_book(self, author, title, pages, publisher, year):  
 *"""Добавление новой книги в БД с использованием SQLAlchemy"""* if not title:  
 messagebox.showerror("Ошибка", "Название книги обязательно для заполнения")  
 return  
  
 try:  
 # Парсинг ID автора из строки формата "id: name"  
 author\_id = int(author.split(':')[0]) if author else None  
 pages = int(pages) if pages else None  
 year = int(year) if year else None  
  
 new\_book = Book(  
 author\_id=author\_id,  
 title=title,  
 pages=pages,  
 publisher=publisher,  
 publication\_year=year  
 )  
  
 self.session.add(new\_book)  
 self.session.commit()  
  
 messagebox.showinfo("Успех", "Книга успешно добавлена")  
 self.show\_books()  
 except ValueError:  
 messagebox.showerror("Ошибка", "Проверьте правильность ввода числовых значений")  
 except Exception as e:  
 self.session.rollback()  
 messagebox.showerror("Ошибка", f"Не удалось добавить книгу: {str(e)}")  
  
 def add\_author(self, name, country, birth\_year, death\_year):  
 *"""Добавление нового автора в БД с использованием SQLAlchemy"""* if not name:  
 messagebox.showerror("Ошибка", "Имя автора обязательно для заполнения")  
 return  
  
 try:  
 birth\_year = int(birth\_year) if birth\_year else None  
 death\_year = int(death\_year) if death\_year else None  
  
 new\_author = Author(  
 name=name,  
 country=country,  
 birth\_year=birth\_year,  
 death\_year=death\_year  
 )  
  
 self.session.add(new\_author)  
 self.session.commit()  
  
 messagebox.showinfo("Успех", "Автор успешно добавлен")  
 self.show\_authors()  
 except ValueError:  
 messagebox.showerror("Ошибка", "Проверьте правильность ввода годов жизни")  
 except Exception as e:  
 self.session.rollback()  
 messagebox.showerror("Ошибка", f"Не удалось добавить автора: {str(e)}")  
  
 def import\_author\_from\_file(self, file\_path, format\_type):  
 *"""Импорт автора из файла JSON или XML с использованием SQLAlchemy"""* if not file\_path:  
 messagebox.showerror("Ошибка", "Выберите файл для импорта")  
 return  
  
 try:  
 author\_data = None  
  
 if format\_type == "json":  
 author\_data = self.parse\_author\_from\_json(file\_path)  
 elif format\_type == "xml":  
 author\_data = self.parse\_author\_from\_xml(file\_path)  
 else:  
 messagebox.showerror("Ошибка", "Неподдерживаемый формат файла")  
 return  
  
 if author\_data:  
 name = author\_data.get('name')  
 country = author\_data.get('country')  
 birth\_year = author\_data.get('birth\_year')  
 death\_year = author\_data.get('death\_year')  
  
 # Добавление автора в БД с использованием SQLAlchemy  
 new\_author = Author(  
 name=name,  
 country=country,  
 birth\_year=birth\_year,  
 death\_year=death\_year  
 )  
  
 self.session.add(new\_author)  
 self.session.commit()  
  
 messagebox.showinfo("Успех", "Автор успешно импортирован")  
 self.show\_authors()  
 except Exception as e:  
 self.session.rollback()  
 messagebox.showerror("Ошибка", f"Не удалось импортировать автора: {str(e)}")  
  
 # Реализация запрошенных запросов  
 def show\_authors\_by\_birth\_year\_range(self, start\_year, end\_year):  
 *"""Вывод фамилий всех авторов, родившихся в диапазоне между X и Y годами"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(  
 result\_frame,  
 text=f"Авторы, родившиеся между {start\_year} и {end\_year} годами",  
 font=("Arial", 16)  
 ).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('id', 'name', 'birth\_year')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('name', text='Имя')  
 tree.heading('birth\_year', text='Год рождения')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('name', width=250)  
 tree.column('birth\_year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием SQLAlchemy  
 authors = self.session.query(Author).filter(  
 Author.birth\_year >= start\_year,  
 Author.birth\_year <= end\_year  
 ).all()  
  
 # Заполнение таблицы данными  
 for author in authors:  
 tree.insert('', tk.END, values=(author.id, author.name, author.birth\_year))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_books\_by\_russian\_authors(self):  
 *"""Вывод всех книг, написанных авторами из России"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(result\_frame, text="Книги авторов из России", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('title', 'author', 'publisher', 'year')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('title', text='Название')  
 tree.heading('author', text='Автор')  
 tree.heading('publisher', text='Издательство')  
 tree.heading('year', text='Год издания')  
  
 # Настройка ширины столбцов  
 tree.column('title', width=200)  
 tree.column('author', width=150)  
 tree.column('publisher', width=150)  
 tree.column('year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием SQLAlchemy  
 books = self.session.query(Book, Author.name).join(Author).filter(  
 Author.country.like('%Россия%')  
 ).all()  
  
 # Заполнение таблицы данными  
 for book, author\_name in books:  
 tree.insert('', tk.END, values=(book.title, author\_name, book.publisher, book.publication\_year))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_books\_by\_page\_count(self, min\_pages):  
 *"""Вывод всех книг с количеством страниц более N"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(result\_frame, text=f"Книги с количеством страниц более {min\_pages}", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('title', 'author', 'pages', 'publisher')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('title', text='Название')  
 tree.heading('author', text='Автор')  
 tree.heading('pages', text='Страниц')  
 tree.heading('publisher', text='Издательство')  
  
 # Настройка ширины столбцов  
 tree.column('title', width=200)  
 tree.column('author', width=150)  
 tree.column('pages', width=80)  
 tree.column('publisher', width=150)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием SQLAlchemy  
 books = self.session.query(Book, Author.name).join(Author, isouter=True).filter(  
 Book.pages > min\_pages  
 ).all()  
  
 # Заполнение таблицы данными  
 for book, author\_name in books:  
 tree.insert('', tk.END, values=(book.title, author\_name, book.pages, book.publisher))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_authors\_by\_book\_count(self, min\_books):  
 *"""Вывод всех авторов с числом книг более N"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(result\_frame, text=f"Авторы с числом книг более {min\_books}", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('name', 'country', 'book\_count')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('name', text='Имя')  
 tree.heading('country', text='Страна')  
 tree.heading('book\_count', text='Количество книг')  
  
 # Настройка ширины столбцов  
 tree.column('name', width=200)  
 tree.column('country', width=150)  
 tree.column('book\_count', width=150)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием SQLAlchemy  
 authors = self.session.query(  
 Author, func.count(Book.id).label('book\_count')  
 ).outerjoin(Book).group\_by(Author.id).having(  
 func.count(Book.id) > min\_books  
 ).all()  
  
 # Заполнение таблицы данными  
 for author, book\_count in authors:  
 tree.insert('', tk.END, values=(author.name, author.country, book\_count))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_main\_menu(self):  
 *"""Отображение главного меню программы с добавлением новых запросов"""* # Очистка текущего окна  
 for widget in self.root.winfo\_children():  
 widget.destroy()  
  
 # Создание главного меню  
 menubar = tk.Menu(self.root)  
 self.root.config(menu=menubar)  
  
 # Меню "Книги"  
 book\_menu = tk.Menu(menubar, tearoff=0)  
 book\_menu.add\_command(label="Список книг", command=self.show\_books)  
 book\_menu.add\_command(label="Добавить книгу", command=self.show\_add\_book)  
 book\_menu.add\_command(label="Книги с более N страниц",  
 command=lambda: self.show\_books\_by\_page\_count(300)) # Пример: 300 страниц  
 menubar.add\_cascade(label="Книги", menu=book\_menu)  
  
 # Меню "Авторы"  
 author\_menu = tk.Menu(menubar, tearoff=0)  
 author\_menu.add\_command(label="Список авторов", command=self.show\_authors)  
 author\_menu.add\_command(label="Добавить автора", command=self.show\_add\_author)  
 author\_menu.add\_command(label="Импорт автора из файла", command=self.show\_import\_author)  
 author\_menu.add\_command(label="Авторы по годам рождения",  
 command=lambda: self.show\_authors\_by\_birth\_year\_range(1800, 1900)) # Пример: 1800-1900  
 author\_menu.add\_command(label="Авторы с более N книг",  
 command=lambda: self.show\_authors\_by\_book\_count(2)) # Пример: 2 книги  
 menubar.add\_cascade(label="Авторы", menu=author\_menu)  
  
 # Меню "Запросы"  
 query\_menu = tk.Menu(menubar, tearoff=0)  
 query\_menu.add\_command(label="Книги российских авторов", command=self.show\_books\_by\_russian\_authors)  
 menubar.add\_cascade(label="Запросы", menu=query\_menu)  
  
 # Меню учетной записи  
 account\_menu = tk.Menu(menubar, tearoff=0)  
 account\_menu.add\_command(label="Выход", command=self.show\_login\_screen)  
 menubar.add\_cascade(label="Учетная запись", menu=account\_menu)  
  
 # Создание начального фрейма для приветствия  
 main\_frame = ttk.Frame(self.root, padding="20")  
 main\_frame.pack(expand=True)  
  
 welcome\_label = ttk.Label(main\_frame,  
 text=f"Добро пожаловать в библиотечную систему, {self.current\_user['username']}!",  
 font=("Arial", 16))  
 welcome\_label.pack(pady=20)  
  
 instruction\_label = ttk.Label(main\_frame,  
 text="Пожалуйста, используйте меню для навигации по системе.",  
 font=("Arial", 12))  
 instruction\_label.pack(pady=10)  
  
 def show\_login\_screen(self):  
 *"""Отображение экрана авторизации"""* # Очистка текущего окна  
 for widget in self.root.winfo\_children():  
 widget.destroy()  
  
 # Создание фрейма авторизации  
 login\_frame = ttk.Frame(self.root, padding='20')  
 login\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(login\_frame, text="Библиотечная система - Авторизация", font=("Arial", 16)).grid(row=0, column=0,  
 columnspan=2,  
 pady=10)  
  
 # Поля ввода  
 ttk.Label(login\_frame, text="Имя пользователя:").grid(row=1, column=0, sticky=tk.W, pady=5)  
 username\_entry = ttk.Entry(login\_frame, width=30)  
 username\_entry.grid(row=1, column=1, pady=5)  
  
 ttk.Label(login\_frame, text="Пароль:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 password\_entry = ttk.Entry(login\_frame, width=30, show="\*")  
 password\_entry.grid(row=2, column=1, pady=5)  
  
 # Кнопка входа  
 ttk.Button(login\_frame, text="Войти",  
 command=lambda: self.login(username\_entry.get(), password\_entry.get())).grid(row=3, column=0,  
 columnspan=2, pady=10)  
  
 # Кнопка регистрации (только для демонстрации)  
 ttk.Button(login\_frame, text="Регистрация",  
 command=self.show\_registration\_screen).grid(row=4, column=0, columnspan=2)  
  
 def show\_registration\_screen(self):  
 *"""Отображение экрана регистрации"""* # Очистка текущего окна  
 for widget in self.root.winfo\_children():  
 widget.destroy()  
  
 # Создание фрейма регистрации  
 reg\_frame = ttk.Frame(self.root, padding="20")  
 reg\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(reg\_frame, text="Библиотечная система - Регистрация", font=("Arial", 16)).grid(row=0, column=0,  
 columnspan=2, pady=10)  
  
 # Поля ввода  
 ttk.Label(reg\_frame, text="Имя пользователя:").grid(row=1, column=0, sticky=tk.W, pady=5)  
 username\_entry = ttk.Entry(reg\_frame, width=30)  
 username\_entry.grid(row=1, column=1, pady=5)  
  
 ttk.Label(reg\_frame, text="Пароль:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 password\_entry = ttk.Entry(reg\_frame, width=30, show="\*")  
 password\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Label(reg\_frame, text="Подтверждение пароля:").grid(row=3, column=0, sticky=tk.W, pady=5)  
 confirm\_entry = ttk.Entry(reg\_frame, width=30, show="\*")  
 confirm\_entry.grid(row=3, column=1, pady=5)  
  
 # Кнопка регистрации  
 ttk.Button(reg\_frame, text="Зарегистрироваться",  
 command=lambda: self.register\_user(username\_entry.get(), password\_entry.get(),  
 confirm\_entry.get())).grid(row=4, column=0, columnspan=2, pady=10)  
  
 # Кнопка возврата  
 ttk.Button(reg\_frame, text="Вернуться к авторизации",  
 command=self.show\_login\_screen).grid(row=5, column=0, columnspan=2)  
  
 def login(self, username, password):  
 *"""Обработка входа пользователя"""* if not username or not password:  
 messagebox.showerror("Ошибка", "Введите имя пользователя и пароль")  
 return  
  
 if self.authenticate(username, password):  
 messagebox.showinfo("Успех", f"Добро пожаловать, {username}!")  
 self.show\_main\_menu()  
 else:  
 messagebox.showerror("Ошибка", "Неверное имя пользователя или пароль")  
  
 def show\_add\_book(self):  
 *"""Отображение формы добавления новой книги"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для добавления книги  
 book\_frame = ttk.Frame(self.root, padding="20")  
 book\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(book\_frame, text="Добавление новой книги", font=("Arial", 16)).grid(row=0, column=0, columnspan=2,  
 pady=10)  
  
 # Выбор автора  
 ttk.Label(book\_frame, text="Автор:").grid(row=1, column=0, sticky=tk.W, pady=5)  
  
 # Получение списка авторов из БД с использованием SQLAlchemy  
 authors = self.session.query(Author.id, Author.name).all()  
  
 # Создание комбобокса с авторами  
 author\_var = tk.StringVar()  
 author\_combo = ttk.Combobox(book\_frame, textvariable=author\_var, width=30)  
 author\_combo['values'] = [f"{author.id}: {author.name}" for author in authors]  
 author\_combo.grid(row=1, column=1, pady=5)  
  
 # Поля для ввода данных о книге  
 ttk.Label(book\_frame, text="Название:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 title\_entry = ttk.Entry(book\_frame, width=30)  
 title\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Label(book\_frame, text="Количество страниц:").grid(row=3, column=0, sticky=tk.W, pady=5)  
 pages\_entry = ttk.Entry(book\_frame, width=30)  
 pages\_entry.grid(row=3, column=1, pady=5)  
  
 ttk.Label(book\_frame, text="Издательство:").grid(row=4, column=0, sticky=tk.W, pady=5)  
 publisher\_entry = ttk.Entry(book\_frame, width=30)  
 publisher\_entry.grid(row=4, column=1, pady=5)  
  
 ttk.Label(book\_frame, text="Год издания:").grid(row=5, column=0, sticky=tk.W, pady=5)  
 year\_entry = ttk.Entry(book\_frame, width=30)  
 year\_entry.grid(row=5, column=1, pady=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(book\_frame)  
 button\_frame.grid(row=6, column=0, columnspan=2, pady=10)  
  
 ttk.Button(button\_frame, text="Добавить",  
 command=lambda: self.add\_book(  
 author\_var.get(), title\_entry.get(), pages\_entry.get(),  
 publisher\_entry.get(), year\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Отмена", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def show\_add\_author(self):  
 *"""Отображение формы добавления нового автора"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для добавления автора  
 author\_frame = ttk.Frame(self.root, padding="20")  
 author\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(author\_frame, text="Добавление нового автора", font=("Arial", 16)).grid(row=0, column=0, columnspan=2,  
 pady=10)  
  
 # Поля для ввода данных об авторе  
 ttk.Label(author\_frame, text="Имя:").grid(row=1, column=0, sticky=tk.W, pady=5)  
 name\_entry = ttk.Entry(author\_frame, width=30)  
 name\_entry.grid(row=1, column=1, pady=5)  
  
 ttk.Label(author\_frame, text="Страна:").grid(row=2, column=0, sticky=tk.W, pady=5)  
 country\_entry = ttk.Entry(author\_frame, width=30)  
 country\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Label(author\_frame, text="Год рождения:").grid(row=3, column=0, sticky=tk.W, pady=5)  
 birth\_entry = ttk.Entry(author\_frame, width=30)  
 birth\_entry.grid(row=3, column=1, pady=5)  
  
 ttk.Label(author\_frame, text="Год смерти:").grid(row=4, column=0, sticky=tk.W, pady=5)  
 death\_entry = ttk.Entry(author\_frame, width=30)  
 death\_entry.grid(row=4, column=1, pady=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(author\_frame)  
 button\_frame.grid(row=5, column=0, columnspan=2, pady=10)  
  
 ttk.Button(button\_frame, text="Добавить",  
 command=lambda: self.add\_author(  
 name\_entry.get(), country\_entry.get(), birth\_entry.get(), death\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Отмена", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def show\_import\_author(self):  
 *"""Отображение формы импорта автора из файла"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для импорта автора  
 import\_frame = ttk.Frame(self.root, padding="20")  
 import\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(import\_frame, text="Импорт автора из файла", font=("Arial", 16)).grid(row=0, column=0, columnspan=3,  
 pady=10)  
  
 # Выбор формата файла  
 ttk.Label(import\_frame, text="Формат файла:").grid(row=1, column=0, sticky=tk.W, pady=5)  
  
 format\_var = tk.StringVar(value="json")  
 ttk.Radiobutton(import\_frame, text="JSON", variable=format\_var, value="json").grid(row=1, column=1, pady=5)  
 ttk.Radiobutton(import\_frame, text="XML", variable=format\_var, value="xml").grid(row=1, column=2, pady=5)  
  
 # Выбор файла  
 ttk.Label(import\_frame, text="Файл:").grid(row=2, column=0, sticky=tk.W, pady=5)  
  
 file\_path\_var = tk.StringVar()  
 file\_entry = ttk.Entry(import\_frame, textvariable=file\_path\_var, width=40)  
 file\_entry.grid(row=2, column=1, pady=5)  
  
 ttk.Button(import\_frame, text="Обзор", command=lambda: self.browse\_file(file\_path\_var, format\_var.get())).grid(  
 row=2, column=2, pady=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(import\_frame)  
 button\_frame.grid(row=3, column=0, columnspan=3, pady=10)  
  
 ttk.Button(button\_frame, text="Импортировать",  
 command=lambda: self.import\_author\_from\_file(file\_path\_var.get(), format\_var.get())  
 ).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Отмена", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def clear\_workspace(self):  
 *"""Очистка рабочей области, сохраняя меню"""* # Сохраняем главное меню  
 menu = self.root.winfo\_children()[0] if self.root.winfo\_children() and isinstance(self.root.winfo\_children()[0],  
 tk.Menu) else None  
  
 # Удаляем все виджеты, кроме меню  
 for widget in self.root.winfo\_children():  
 if widget != menu:  
 widget.destroy()  
  
 def parse\_author\_from\_json(self, file\_path):  
 *"""Парсинг данных автора из JSON файла"""* with open(file\_path, 'r', encoding='utf-8') as f:  
 data = json.load(f)  
  
 author\_data = {  
 'name': data.get('name'),  
 'country': data.get('country'),  
 'birth\_year': None,  
 'death\_year': None  
 }  
  
 # Обработка годов жизни  
 years = data.get('years', [])  
 if len(years) >= 2:  
 author\_data['birth\_year'] = years[0]  
 author\_data['death\_year'] = years[1]  
  
 return author\_data  
  
 def parse\_author\_from\_xml(self, file\_path):  
 *"""Парсинг данных автора из XML файла"""* tree = ET.parse(file\_path)  
 root = tree.getroot()  
  
 author\_data = {  
 'name': None,  
 'country': None,  
 'birth\_year': None,  
 'death\_year': None  
 }  
  
 # Получение имени автора  
 name\_elem = root.find('name')  
 if name\_elem is not None and name\_elem.text:  
 author\_data['name'] = name\_elem.text  
  
 # Получение страны автора  
 country\_elem = root.find('country')  
 if country\_elem is not None and country\_elem.text:  
 author\_data['country'] = country\_elem.text  
  
 # Получение годов жизни  
 years\_elem = root.find('years')  
 if years\_elem is not None:  
 if 'born' in years\_elem.attrib:  
 try:  
 author\_data['birth\_year'] = int(years\_elem.attrib['born'])  
 except ValueError:  
 pass  
  
 if 'died' in years\_elem.attrib:  
 try:  
 author\_data['death\_year'] = int(years\_elem.attrib['died'])  
 except ValueError:  
 pass  
  
 return author\_data  
  
 def run(self):  
 *"""Запуск приложения"""* self.root.mainloop()  
  
 # Закрытие соединения с БД при выходе  
 if hasattr(self, 'session'):  
 self.session.close()  
  
app = LibrarySystem()  
app.run()

Вывод программы:





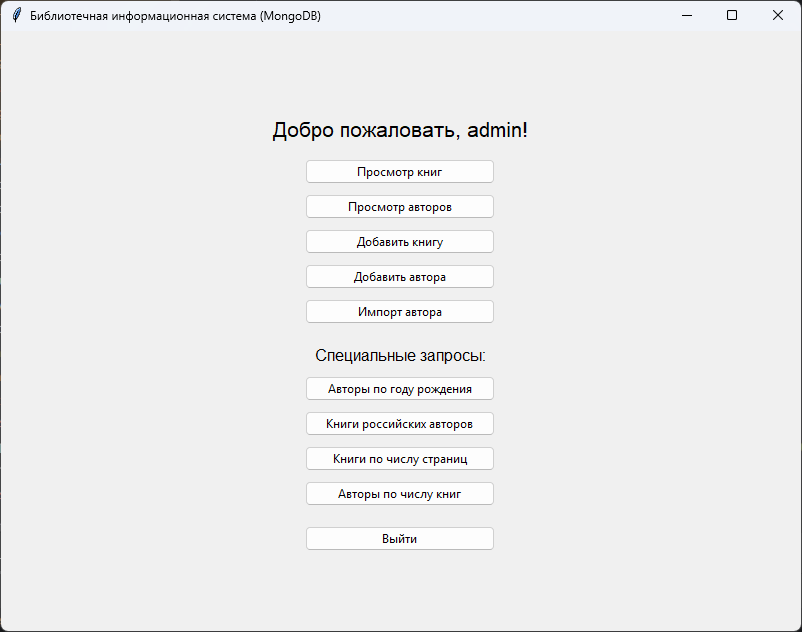




1. Код программы:

*"""  
Реализация библиотечной системы с использованием MongoDB  
"""*import hashlib  
import json  
import xml.etree.ElementTree as ET  
import tkinter as tk  
from tkinter import ttk, messagebox, filedialog  
from pymongo import MongoClient  
from bson.objectid import ObjectId  
from datetime import datetime  
  
  
class LibrarySystem:  
 def \_\_init\_\_(self):  
 self.current\_user = None  
  
 # Инициализация БД с использованием MongoDB  
 self.initialize\_database()  
  
 # Создание основного окна приложения  
 self.root = tk.Tk()  
 self.root.title('Библиотечная информационная система (MongoDB)')  
 self.root.geometry('800x600')  
  
 # Открытие окна авторизации  
 self.show\_login\_screen()  
  
 def initialize\_database(self):  
 *"""Инициализация БД с использованием MongoDB"""* # Создание подключения к MongoDB  
 self.client = MongoClient('mongodb://localhost:27017/')  
 self.db = self.client['library\_db']  
  
 # Получение коллекций (аналог таблиц в SQL)  
 self.users\_collection = self.db['users']  
 self.authors\_collection = self.db['authors']  
 self.books\_collection = self.db['books']  
  
 # Добавление тестового администратора, если коллекция пользователей пуста  
 admin\_count = self.users\_collection.count\_documents({})  
 if admin\_count == 0:  
 admin\_password = self.hash\_password("admin")  
 admin\_user = {  
 "username": "admin",  
 "password": admin\_password,  
 "is\_admin": 1,  
 "created\_at": datetime.now()  
 }  
 self.users\_collection.insert\_one(admin\_user)  
  
 def hash\_password(self, password):  
 *"""Хэширование пароля с использованием SHA-256"""* return hashlib.sha256(password.encode()).hexdigest()  
  
 def authenticate(self, username, password):  
 *"""Проверка учётных данных пользователя с использованием MongoDB"""* hashed\_password = self.hash\_password(password)  
 user = self.users\_collection.find\_one({"username": username, "password": hashed\_password})  
  
 if user:  
 self.current\_user = {  
 "id": str(user["\_id"]),  
 "username": username,  
 "is\_admin": user.get("is\_admin", 0)  
 }  
 return True  
 return False  
  
 def register\_user(self, username, password, confirm\_password):  
 *"""Регистрация нового пользователя с использованием MongoDB"""* if not username or not password:  
 messagebox.showerror("Ошибка", "Необходимо заполнить все поля")  
 return  
  
 if password != confirm\_password:  
 messagebox.showerror("Ошибка", "Пароли не совпадают")  
 return  
  
 try:  
 # Проверка существования пользователя  
 existing\_user = self.users\_collection.find\_one({"username": username})  
 if existing\_user:  
 messagebox.showerror("Ошибка", "Пользователь с таким именем уже существует")  
 return  
  
 hashed\_password = self.hash\_password(password)  
 new\_user = {  
 "username": username,  
 "password": hashed\_password,  
 "is\_admin": 0,  
 "created\_at": datetime.now()  
 }  
  
 self.users\_collection.insert\_one(new\_user)  
 messagebox.showinfo("Успех", "Пользователь успешно зарегистрирован")  
 self.show\_login\_screen()  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось зарегистрировать пользователя: {str(e)}")  
  
 def show\_books(self):  
 *"""Отображение списка всех книг с использованием MongoDB"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения книг  
 books\_frame = ttk.Frame(self.root, padding="10")  
 books\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(books\_frame, text="Список книг", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения книг  
 columns = ('id', 'title', 'author', 'pages', 'publisher', 'year')  
 tree = ttk.Treeview(books\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('title', text='Название')  
 tree.heading('author', text='Автор')  
 tree.heading('pages', text='Страниц')  
 tree.heading('publisher', text='Издательство')  
 tree.heading('year', text='Год издания')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('title', width=200)  
 tree.column('author', width=150)  
 tree.column('pages', width=80)  
 tree.column('publisher', width=150)  
 tree.column('year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(books\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Получение данных из БД с использованием MongoDB (с выполнением JOIN через $lookup)  
 pipeline = [  
 {  
 "$lookup": {  
 "from": "authors",  
 "localField": "author\_id",  
 "foreignField": "\_id",  
 "as": "author\_info"  
 }  
 },  
 {  
 "$unwind": {  
 "path": "$author\_info",  
 "preserveNullAndEmptyArrays": True  
 }  
 }  
 ]  
  
 books = list(self.books\_collection.aggregate(pipeline))  
  
 # Заполнение таблицы данными  
 for book in books:  
 author\_name = book.get("author\_info", {}).get("name", "Неизвестен")  
 book\_id = str(book.get("\_id"))[:8] # Сокращаем ID для отображения  
 tree.insert('', tk.END, values=(  
 book\_id,  
 book.get("title", ""),  
 author\_name,  
 book.get("pages", ""),  
 book.get("publisher", ""),  
 book.get("publication\_year", "")  
 ))  
  
 # Добавление кнопки возврата  
 ttk.Button(books\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_authors(self):  
 *"""Отображение списка всех авторов с использованием MongoDB"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения авторов  
 authors\_frame = ttk.Frame(self.root, padding="10")  
 authors\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(authors\_frame, text="Список авторов", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения авторов  
 columns = ('id', 'name', 'country', 'birth\_year', 'death\_year')  
 tree = ttk.Treeview(authors\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('name', text='Имя')  
 tree.heading('country', text='Страна')  
 tree.heading('birth\_year', text='Год рождения')  
 tree.heading('death\_year', text='Год смерти')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('name', width=200)  
 tree.column('country', width=150)  
 tree.column('birth\_year', width=100)  
 tree.column('death\_year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(authors\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Получение данных из БД с использованием MongoDB  
 authors = list(self.authors\_collection.find())  
  
 # Заполнение таблицы данными  
 for author in authors:  
 author\_id = str(author.get("\_id"))[:8] # Сокращаем ID для отображения  
 tree.insert('', tk.END, values=(  
 author\_id,  
 author.get("name", ""),  
 author.get("country", ""),  
 author.get("birth\_year", ""),  
 author.get("death\_year", "")  
 ))  
  
 # Добавление кнопок  
 button\_frame = ttk.Frame(authors\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Экспорт в JSON",  
 command=lambda: self.export\_author\_to\_json(tree)).pack(side=tk.LEFT, padx=5)  
 ttk.Button(button\_frame, text="Экспорт в XML",  
 command=lambda: self.export\_author\_to\_xml(tree)).pack(side=tk.LEFT, padx=5)  
 ttk.Button(button\_frame, text="Назад", command=self.show\_main\_menu).pack(side=tk.LEFT, padx=5)  
  
 def add\_book(self, author, title, pages, publisher, year):  
 *"""Добавление новой книги в БД с использованием MongoDB"""* if not title:  
 messagebox.showerror("Ошибка", "Название книги обязательно для заполнения")  
 return  
  
 try:  
 # Парсинг ID автора из строки формата "id: name"  
 author\_id = None  
 if author:  
 try:  
 author\_id = ObjectId(author.split(':')[0].strip())  
 except:  
 pass  
  
 # Преобразование строковых данных в соответствующие типы  
 pages\_int = int(pages) if pages and pages.isdigit() else None  
 year\_int = int(year) if year and year.isdigit() else None  
  
 new\_book = {  
 "author\_id": author\_id,  
 "title": title,  
 "pages": pages\_int,  
 "publisher": publisher,  
 "publication\_year": year\_int,  
 "created\_at": datetime.now()  
 }  
  
 self.books\_collection.insert\_one(new\_book)  
 messagebox.showinfo("Успех", "Книга успешно добавлена")  
 self.show\_books()  
 except ValueError:  
 messagebox.showerror("Ошибка", "Проверьте правильность ввода числовых значений")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось добавить книгу: {str(e)}")  
  
 def add\_author(self, name, country, birth\_year, death\_year):  
 *"""Добавление нового автора в БД с использованием MongoDB"""* if not name:  
 messagebox.showerror("Ошибка", "Имя автора обязательно для заполнения")  
 return  
  
 try:  
 # Преобразование строковых данных в соответствующие типы  
 birth\_year\_int = int(birth\_year) if birth\_year and birth\_year.isdigit() else None  
 death\_year\_int = int(death\_year) if death\_year and death\_year.isdigit() else None  
  
 new\_author = {  
 "name": name,  
 "country": country,  
 "birth\_year": birth\_year\_int,  
 "death\_year": death\_year\_int,  
 "created\_at": datetime.now()  
 }  
  
 self.authors\_collection.insert\_one(new\_author)  
 messagebox.showinfo("Успех", "Автор успешно добавлен")  
 self.show\_authors()  
 except ValueError:  
 messagebox.showerror("Ошибка", "Проверьте правильность ввода годов жизни")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось добавить автора: {str(e)}")  
  
 def import\_author\_from\_file(self, file\_path, format\_type):  
 *"""Импорт автора из файла JSON или XML с использованием MongoDB"""* if not file\_path:  
 messagebox.showerror("Ошибка", "Выберите файл для импорта")  
 return  
  
 try:  
 author\_data = None  
  
 if format\_type == "json":  
 author\_data = self.parse\_author\_from\_json(file\_path)  
 elif format\_type == "xml":  
 author\_data = self.parse\_author\_from\_xml(file\_path)  
 else:  
 messagebox.showerror("Ошибка", "Неподдерживаемый формат файла")  
 return  
  
 if author\_data and author\_data['name']:  
 # Добавление дополнительных полей для MongoDB  
 author\_data['created\_at'] = datetime.now()  
  
 # Преобразование типов данных  
 if 'birth\_year' in author\_data and author\_data['birth\_year']:  
 author\_data['birth\_year'] = int(author\_data['birth\_year'])  
 if 'death\_year' in author\_data and author\_data['death\_year']:  
 author\_data['death\_year'] = int(author\_data['death\_year'])  
  
 # Добавление автора в MongoDB  
 self.authors\_collection.insert\_one(author\_data)  
  
 messagebox.showinfo("Успех", "Автор успешно импортирован")  
 self.show\_authors()  
 else:  
 messagebox.showerror("Ошибка", "Не удалось получить необходимые данные из файла")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось импортировать автора: {str(e)}")  
  
 def export\_author\_to\_json(self, tree):  
 *"""Экспорт выбранного автора в JSON файл"""* selected\_item = tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите автора для экспорта")  
 return  
  
 try:  
 # Получение данных выбранного автора  
 item\_values = tree.item(selected\_item[0])['values']  
 author\_id = item\_values[0]  
  
 # Поиск автора в базе данных по ID  
 author = self.authors\_collection.find\_one({"\_id": ObjectId(author\_id)})  
  
 if not author:  
 messagebox.showerror("Ошибка", "Автор не найден в базе данных")  
 return  
  
 # Получение пути для сохранения файла  
 file\_path = filedialog.asksaveasfilename(  
 defaultextension=".json",  
 filetypes=[("JSON файлы", "\*.json"), ("Все файлы", "\*.\*")]  
 )  
  
 if not file\_path:  
 return  
  
 # Подготовка данных для экспорта  
 export\_data = {  
 "name": author.get("name", ""),  
 "country": author.get("country", ""),  
 "years": [author.get("birth\_year"), author.get("death\_year")]  
 }  
  
 # Удаление полей MongoDB перед экспортом  
 if "\_id" in export\_data:  
 del export\_data["\_id"]  
  
 # Запись данных в файл  
 with open(file\_path, 'w', encoding='utf-8') as f:  
 json.dump(export\_data, f, ensure\_ascii=False, indent=4)  
  
 messagebox.showinfo("Успех", f"Автор успешно экспортирован в файл {file\_path}")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось экспортировать автора: {str(e)}")  
  
 def export\_author\_to\_xml(self, tree):  
 *"""Экспорт выбранного автора в XML файл"""* selected\_item = tree.selection()  
 if not selected\_item:  
 messagebox.showerror("Ошибка", "Выберите автора для экспорта")  
 return  
  
 try:  
 # Получение данных выбранного автора  
 item\_values = tree.item(selected\_item[0])['values']  
 author\_id = item\_values[0]  
  
 # Поиск автора в базе данных по ID  
 author = self.authors\_collection.find\_one({"\_id": ObjectId(author\_id)})  
  
 if not author:  
 messagebox.showerror("Ошибка", "Автор не найден в базе данных")  
 return  
  
 # Получение пути для сохранения файла  
 file\_path = filedialog.asksaveasfilename(  
 defaultextension=".xml",  
 filetypes=[("XML файлы", "\*.xml"), ("Все файлы", "\*.\*")]  
 )  
  
 if not file\_path:  
 return  
  
 # Создание XML-документа  
 root = ET.Element("author")  
  
 name\_elem = ET.SubElement(root, "name")  
 name\_elem.text = author.get("name", "")  
  
 country\_elem = ET.SubElement(root, "country")  
 country\_elem.text = author.get("country", "")  
  
 years\_elem = ET.SubElement(root, "years")  
 if author.get("birth\_year"):  
 years\_elem.set("born", str(author.get("birth\_year")))  
 if author.get("death\_year"):  
 years\_elem.set("died", str(author.get("death\_year")))  
  
 # Запись XML в файл  
 tree = ET.ElementTree(root)  
 tree.write(file\_path, encoding='utf-8', xml\_declaration=True)  
  
 messagebox.showinfo("Успех", f"Автор успешно экспортирован в файл {file\_path}")  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось экспортировать автора: {str(e)}")  
  
 # Реализация запрошенных запросов  
 def show\_authors\_by\_birth\_year\_range(self, start\_year, end\_year):  
 *"""Вывод фамилий всех авторов, родившихся в диапазоне между X и Y годами"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(  
 result\_frame,  
 text=f"Авторы, родившиеся между {start\_year} и {end\_year} годами",  
 font=("Arial", 16)  
 ).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('id', 'name', 'birth\_year')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('id', text='ID')  
 tree.heading('name', text='Имя')  
 tree.heading('birth\_year', text='Год рождения')  
  
 # Настройка ширины столбцов  
 tree.column('id', width=30)  
 tree.column('name', width=250)  
 tree.column('birth\_year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием MongoDB  
 authors = list(self.authors\_collection.find({  
 "birth\_year": {"$gte": start\_year, "$lte": end\_year}  
 }))  
  
 # Заполнение таблицы данными  
 for author in authors:  
 author\_id = str(author.get("\_id"))[:8] # Сокращаем ID для отображения  
 tree.insert('', tk.END, values=(  
 author\_id,  
 author.get("name", ""),  
 author.get("birth\_year", "")  
 ))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_books\_by\_russian\_authors(self):  
 *"""Вывод всех книг, написанных авторами из России"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(result\_frame, text="Книги авторов из России", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('title', 'author', 'publisher', 'year')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('title', text='Название')  
 tree.heading('author', text='Автор')  
 tree.heading('publisher', text='Издательство')  
 tree.heading('year', text='Год издания')  
  
 # Настройка ширины столбцов  
 tree.column('title', width=200)  
 tree.column('author', width=150)  
 tree.column('publisher', width=150)  
 tree.column('year', width=100)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием MongoDB (аналог JOIN в SQL)  
 pipeline = [  
 {  
 "$lookup": {  
 "from": "authors",  
 "localField": "author\_id",  
 "foreignField": "\_id",  
 "as": "author\_info"  
 }  
 },  
 {  
 "$unwind": {  
 "path": "$author\_info",  
 "preserveNullAndEmptyArrays": False  
 }  
 },  
 {  
 "$match": {  
 "author\_info.country": {"$regex": "Россия", "$options": "i"}  
 }  
 }  
 ]  
  
 books = list(self.books\_collection.aggregate(pipeline))  
  
 # Заполнение таблицы данными  
 for book in books:  
 author\_name = book.get("author\_info", {}).get("name", "")  
 tree.insert('', tk.END, values=(  
 book.get("title", ""),  
 author\_name,  
 book.get("publisher", ""),  
 book.get("publication\_year", "")  
 ))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_books\_by\_page\_count(self, min\_pages):  
 *"""Вывод всех книг с количеством страниц более N"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(result\_frame, text=f"Книги с количеством страниц более {min\_pages}", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('title', 'author', 'pages', 'publisher')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('title', text='Название')  
 tree.heading('author', text='Автор')  
 tree.heading('pages', text='Страниц')  
 tree.heading('publisher', text='Издательство')  
  
 # Настройка ширины столбцов  
 tree.column('title', width=200)  
 tree.column('author', width=150)  
 tree.column('pages', width=80)  
 tree.column('publisher', width=150)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием MongoDB  
 pipeline = [  
 {  
 "$match": {  
 "pages": {"$gt": min\_pages}  
 }  
 },  
 {  
 "$lookup": {  
 "from": "authors",  
 "localField": "author\_id",  
 "foreignField": "\_id",  
 "as": "author\_info"  
 }  
 },  
 {  
 "$unwind": {  
 "path": "$author\_info",  
 "preserveNullAndEmptyArrays": True  
 }  
 }  
 ]  
  
 books = list(self.books\_collection.aggregate(pipeline))  
  
 # Заполнение таблицы данными  
 for book in books:  
 author\_name = book.get("author\_info", {}).get("name", "Неизвестен")  
 tree.insert('', tk.END, values=(  
 book.get("title", ""),  
 author\_name,  
 book.get("pages", ""),  
 book.get("publisher", "")  
 ))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def show\_authors\_by\_book\_count(self, min\_books):  
 *"""Вывод всех авторов с числом книг более N"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для отображения результатов  
 result\_frame = ttk.Frame(self.root, padding="10")  
 result\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 ttk.Label(result\_frame, text=f"Авторы с числом книг более {min\_books}", font=("Arial", 16)).pack(pady=10)  
  
 # Создание таблицы для отображения результатов  
 columns = ('name', 'country', 'book\_count')  
 tree = ttk.Treeview(result\_frame, columns=columns, show='headings')  
  
 # Настройка заголовков столбцов  
 tree.heading('name', text='Имя')  
 tree.heading('country', text='Страна')  
 tree.heading('book\_count', text='Количество книг')  
  
 # Настройка ширины столбцов  
 tree.column('name', width=200)  
 tree.column('country', width=150)  
 tree.column('book\_count', width=150)  
  
 # Добавление скроллбара  
 scrollbar = ttk.Scrollbar(result\_frame, orient=tk.VERTICAL, command=tree.yview)  
 tree.configure(yscroll=scrollbar.set)  
 scrollbar.pack(side=tk.RIGHT, fill=tk.Y)  
 tree.pack(fill=tk.BOTH, expand=True)  
  
 # Выполнение запроса с использованием MongoDB (аналог GROUP BY и HAVING в SQL)  
 pipeline = [  
 {  
 "$lookup": {  
 "from": "books",  
 "localField": "\_id",  
 "foreignField": "author\_id",  
 "as": "books"  
 }  
 },  
 {  
 "$project": {  
 "name": 1,  
 "country": 1,  
 "book\_count": {"$size": "$books"}  
 }  
 },  
 {  
 "$match": {  
 "book\_count": {"$gt": min\_books}  
 }  
 }  
 ]  
  
 authors = list(self.authors\_collection.aggregate(pipeline))  
  
 # Заполнение таблицы данными  
 for author in authors:  
 tree.insert('', tk.END, values=(  
 author.get("name", ""),  
 author.get("country", ""),  
 author.get("book\_count", 0)  
 ))  
  
 # Добавление кнопки возврата  
 ttk.Button(result\_frame, text="Назад", command=self.show\_main\_menu).pack(pady=10)  
  
 def parse\_author\_from\_json(self, file\_path):  
 *"""Парсинг данных автора из JSON-файла"""* try:  
 with open(file\_path, 'r', encoding='utf-8') as f:  
 author\_data = json.load(f)  
  
 # Проверяем необходимые поля  
 if 'name' not in author\_data:  
 raise ValueError("В файле отсутствует обязательное поле 'name'")  
  
 # Преобразование полей для соответствия структуре БД  
 result = {  
 "name": author\_data.get("name", ""),  
 "country": author\_data.get("country", "")  
 }  
  
 # Обработка годов жизни (могут быть в разных форматах)  
 if "years" in author\_data and isinstance(author\_data["years"], list):  
 if len(author\_data["years"]) >= 1 and author\_data["years"][0]:  
 result["birth\_year"] = int(author\_data["years"][0])  
 if len(author\_data["years"]) >= 2 and author\_data["years"][1]:  
 result["death\_year"] = int(author\_data["years"][1])  
  
 return result  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось разобрать JSON-файл: {str(e)}")  
 return None  
  
 def parse\_author\_from\_xml(self, file\_path):  
 *"""Парсинг данных автора из XML-файла"""* try:  
 tree = ET.parse(file\_path)  
 root = tree.getroot()  
  
 if root.tag != "author":  
 raise ValueError("Неверный формат XML: корневой элемент должен быть 'author'")  
  
 result = {  
 "name": "",  
 "country": ""  
 }  
  
 # Парсинг имени  
 name\_elem = root.find("name")  
 if name\_elem is not None and name\_elem.text:  
 result["name"] = name\_elem.text  
 else:  
 raise ValueError("В XML-файле отсутствует обязательное поле 'name'")  
  
 # Парсинг страны  
 country\_elem = root.find("country")  
 if country\_elem is not None and country\_elem.text:  
 result["country"] = country\_elem.text  
  
 # Парсинг годов жизни  
 years\_elem = root.find("years")  
 if years\_elem is not None:  
 if "born" in years\_elem.attrib:  
 result["birth\_year"] = int(years\_elem.attrib["born"])  
 if "died" in years\_elem.attrib:  
 result["death\_year"] = int(years\_elem.attrib["died"])  
  
 return result  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Не удалось разобрать XML-файл: {str(e)}")  
 return None  
  
 def show\_login\_screen(self):  
 *"""Отображение экрана авторизации"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для авторизации  
 login\_frame = ttk.Frame(self.root, padding="20")  
 login\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(login\_frame, text="Вход в систему", font=("Arial", 16)).pack(pady=10)  
  
 # Поля ввода  
 username\_frame = ttk.Frame(login\_frame)  
 username\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(username\_frame, text="Имя пользователя:").pack(side=tk.LEFT)  
 username\_entry = ttk.Entry(username\_frame, width=30)  
 username\_entry.pack(side=tk.LEFT, padx=5)  
  
 password\_frame = ttk.Frame(login\_frame)  
 password\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(password\_frame, text="Пароль:").pack(side=tk.LEFT)  
 password\_entry = ttk.Entry(password\_frame, width=30, show="\*")  
 password\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(login\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Войти",  
 command=lambda: self.login(username\_entry.get(), password\_entry.get())  
 ).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Регистрация",  
 command=self.show\_registration\_screen  
 ).pack(side=tk.LEFT, padx=5)  
  
 def show\_registration\_screen(self):  
 *"""Отображение экрана регистрации"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для регистрации  
 reg\_frame = ttk.Frame(self.root, padding="20")  
 reg\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(reg\_frame, text="Регистрация пользователя", font=("Arial", 16)).pack(pady=10)  
  
 # Поля ввода  
 username\_frame = ttk.Frame(reg\_frame)  
 username\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(username\_frame, text="Имя пользователя:").pack(side=tk.LEFT)  
 username\_entry = ttk.Entry(username\_frame, width=30)  
 username\_entry.pack(side=tk.LEFT, padx=5)  
  
 password\_frame = ttk.Frame(reg\_frame)  
 password\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(password\_frame, text="Пароль:").pack(side=tk.LEFT)  
 password\_entry = ttk.Entry(password\_frame, width=30, show="\*")  
 password\_entry.pack(side=tk.LEFT, padx=5)  
  
 confirm\_frame = ttk.Frame(reg\_frame)  
 confirm\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(confirm\_frame, text="Подтверждение:").pack(side=tk.LEFT)  
 confirm\_entry = ttk.Entry(confirm\_frame, width=30, show="\*")  
 confirm\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(reg\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Зарегистрироваться",  
 command=lambda: self.register\_user(  
 username\_entry.get(),  
 password\_entry.get(),  
 confirm\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Назад",  
 command=self.show\_login\_screen  
 ).pack(side=tk.LEFT, padx=5)  
  
 def login(self, username, password):  
 *"""Обработка входа пользователя"""* if not username or not password:  
 messagebox.showerror("Ошибка", "Необходимо заполнить все поля")  
 return  
  
 if self.authenticate(username, password):  
 self.show\_main\_menu()  
 else:  
 messagebox.showerror("Ошибка", "Неверное имя пользователя или пароль")  
  
 def show\_main\_menu(self):  
 *"""Отображение главного меню"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для главного меню  
 menu\_frame = ttk.Frame(self.root, padding="20")  
 menu\_frame.pack(expand=True)  
  
 # Приветствие  
 ttk.Label(  
 menu\_frame,  
 text=f"Добро пожаловать, {self.current\_user['username']}!",  
 font=("Arial", 16)  
 ).pack(pady=10)  
  
 # Кнопки основных функций  
 ttk.Button(menu\_frame, text="Просмотр книг",  
 command=self.show\_books, width=30  
 ).pack(pady=5)  
  
 ttk.Button(menu\_frame, text="Просмотр авторов",  
 command=self.show\_authors, width=30  
 ).pack(pady=5)  
  
 # Кнопки для добавления данных (только для администратора)  
 if self.current\_user['is\_admin']:  
 ttk.Button(menu\_frame, text="Добавить книгу",  
 command=self.show\_add\_book\_form, width=30  
 ).pack(pady=5)  
  
 ttk.Button(menu\_frame, text="Добавить автора",  
 command=self.show\_add\_author\_form, width=30  
 ).pack(pady=5)  
  
 ttk.Button(menu\_frame, text="Импорт автора",  
 command=self.show\_import\_author\_form, width=30  
 ).pack(pady=5)  
  
 # Кнопки специальных запросов  
 ttk.Label(menu\_frame, text="Специальные запросы:", font=("Arial", 12)).pack(pady=(15, 5))  
  
 ttk.Button(menu\_frame, text="Авторы по году рождения",  
 command=self.show\_authors\_by\_birth\_year\_form, width=30  
 ).pack(pady=5)  
  
 ttk.Button(menu\_frame, text="Книги российских авторов",  
 command=self.show\_books\_by\_russian\_authors, width=30  
 ).pack(pady=5)  
  
 ttk.Button(menu\_frame, text="Книги по числу страниц",  
 command=self.show\_books\_by\_page\_count\_form, width=30  
 ).pack(pady=5)  
  
 ttk.Button(menu\_frame, text="Авторы по числу книг",  
 command=self.show\_authors\_by\_book\_count\_form, width=30  
 ).pack(pady=5)  
  
 # Кнопка выхода  
 ttk.Button(menu\_frame, text="Выйти",  
 command=self.logout, width=30  
 ).pack(pady=(15, 5))  
  
 def logout(self):  
 *"""Выход из системы"""* self.current\_user = None  
 self.show\_login\_screen()  
  
 def clear\_workspace(self):  
 *"""Очистка рабочей области от всех виджетов"""* for widget in self.root.winfo\_children():  
 widget.destroy()  
  
 def show\_add\_book\_form(self):  
 *"""Отображение формы для добавления новой книги"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для формы  
 form\_frame = ttk.Frame(self.root, padding="20")  
 form\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(form\_frame, text="Добавление новой книги", font=("Arial", 16)).pack(pady=10)  
  
 # Поля формы  
 # Выбор автора из выпадающего списка  
 author\_frame = ttk.Frame(form\_frame)  
 author\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(author\_frame, text="Автор:").pack(side=tk.LEFT)  
  
 # Получение списка авторов из БД  
 authors = list(self.authors\_collection.find().sort("name"))  
 author\_list = [""] + [f"{str(a['\_id'])}: {a['name']}" for a in authors]  
  
 author\_var = tk.StringVar()  
 author\_combobox = ttk.Combobox(author\_frame, textvariable=author\_var, width=30)  
 author\_combobox['values'] = author\_list  
 author\_combobox.pack(side=tk.LEFT, padx=5)  
  
 # Название книги  
 title\_frame = ttk.Frame(form\_frame)  
 title\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(title\_frame, text="Название:").pack(side=tk.LEFT)  
 title\_entry = ttk.Entry(title\_frame, width=30)  
 title\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Количество страниц  
 pages\_frame = ttk.Frame(form\_frame)  
 pages\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(pages\_frame, text="Страниц:").pack(side=tk.LEFT)  
 pages\_entry = ttk.Entry(pages\_frame, width=30)  
 pages\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Издательство  
 publisher\_frame = ttk.Frame(form\_frame)  
 publisher\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(publisher\_frame, text="Издательство:").pack(side=tk.LEFT)  
 publisher\_entry = ttk.Entry(publisher\_frame, width=30)  
 publisher\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Год издания  
 year\_frame = ttk.Frame(form\_frame)  
 year\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(year\_frame, text="Год издания:").pack(side=tk.LEFT)  
 year\_entry = ttk.Entry(year\_frame, width=30)  
 year\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(form\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Добавить",  
 command=lambda: self.add\_book(  
 author\_var.get(),  
 title\_entry.get(),  
 pages\_entry.get(),  
 publisher\_entry.get(),  
 year\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Назад",  
 command=self.show\_main\_menu  
 ).pack(side=tk.LEFT, padx=5)  
  
 def show\_add\_author\_form(self):  
 *"""Отображение формы для добавления нового автора"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для формы  
 form\_frame = ttk.Frame(self.root, padding="20")  
 form\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(form\_frame, text="Добавление нового автора", font=("Arial", 16)).pack(pady=10)  
  
 # Поля формы  
 # Имя автора  
 name\_frame = ttk.Frame(form\_frame)  
 name\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(name\_frame, text="Имя:").pack(side=tk.LEFT)  
 name\_entry = ttk.Entry(name\_frame, width=30)  
 name\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Страна  
 country\_frame = ttk.Frame(form\_frame)  
 country\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(country\_frame, text="Страна:").pack(side=tk.LEFT)  
 country\_entry = ttk.Entry(country\_frame, width=30)  
 country\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Год рождения  
 birth\_frame = ttk.Frame(form\_frame)  
 birth\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(birth\_frame, text="Год рождения:").pack(side=tk.LEFT)  
 birth\_entry = ttk.Entry(birth\_frame, width=30)  
 birth\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Год смерти  
 death\_frame = ttk.Frame(form\_frame)  
 death\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(death\_frame, text="Год смерти:").pack(side=tk.LEFT)  
 death\_entry = ttk.Entry(death\_frame, width=30)  
 death\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(form\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Добавить",  
 command=lambda: self.add\_author(  
 name\_entry.get(),  
 country\_entry.get(),  
 birth\_entry.get(),  
 death\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Назад",  
 command=self.show\_main\_menu  
 ).pack(side=tk.LEFT, padx=5)  
  
 def show\_import\_author\_form(self):  
 *"""Отображение формы для импорта автора из файла"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для формы  
 form\_frame = ttk.Frame(self.root, padding="20")  
 form\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(form\_frame, text="Импорт автора из файла", font=("Arial", 16)).pack(pady=10)  
  
 # Переменные для хранения пути к файлу и формата  
 file\_path\_var = tk.StringVar()  
 format\_var = tk.StringVar(value="json")  
  
 # Выбор файла  
 file\_frame = ttk.Frame(form\_frame)  
 file\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(file\_frame, text="Файл:").pack(side=tk.LEFT)  
 ttk.Entry(file\_frame, textvariable=file\_path\_var, width=30).pack(side=tk.LEFT, padx=5)  
 ttk.Button(file\_frame, text="Обзор",  
 command=lambda: file\_path\_var.set(filedialog.askopenfilename(  
 filetypes=[("JSON файлы", "\*.json"), ("XML файлы", "\*.xml"), ("Все файлы", "\*.\*")]  
 ))  
 ).pack(side=tk.LEFT)  
  
 # Выбор формата  
 format\_frame = ttk.Frame(form\_frame)  
 format\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(format\_frame, text="Формат:").pack(side=tk.LEFT)  
 ttk.Radiobutton(format\_frame, text="JSON", variable=format\_var, value="json").pack(side=tk.LEFT, padx=5)  
 ttk.Radiobutton(format\_frame, text="XML", variable=format\_var, value="xml").pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(form\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Импортировать",  
 command=lambda: self.import\_author\_from\_file(  
 file\_path\_var.get(),  
 format\_var.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Назад",  
 command=self.show\_main\_menu  
 ).pack(side=tk.LEFT, padx=5)  
  
 def show\_authors\_by\_birth\_year\_form(self):  
 *"""Отображение формы для поиска авторов по диапазону лет рождения"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для формы  
 form\_frame = ttk.Frame(self.root, padding="20")  
 form\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(form\_frame, text="Поиск авторов по году рождения", font=("Arial", 16)).pack(pady=10)  
  
 # Поля формы  
 # Начальный год  
 start\_frame = ttk.Frame(form\_frame)  
 start\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(start\_frame, text="С года:").pack(side=tk.LEFT)  
 start\_entry = ttk.Entry(start\_frame, width=30)  
 start\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Конечный год  
 end\_frame = ttk.Frame(form\_frame)  
 end\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(end\_frame, text="По год:").pack(side=tk.LEFT)  
 end\_entry = ttk.Entry(end\_frame, width=30)  
 end\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(form\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Найти",  
 command=lambda: self.find\_authors\_by\_birth\_year(  
 start\_entry.get(),  
 end\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Назад",  
 command=self.show\_main\_menu  
 ).pack(side=tk.LEFT, padx=5)  
  
 def find\_authors\_by\_birth\_year(self, start\_year\_str, end\_year\_str):  
 *"""Поиск и отображение авторов по диапазону лет рождения"""* try:  
 # Проверка и преобразование ввода  
 if not start\_year\_str or not end\_year\_str:  
 messagebox.showerror("Ошибка", "Необходимо заполнить оба поля")  
 return  
  
 start\_year = int(start\_year\_str)  
 end\_year = int(end\_year\_str)  
  
 if start\_year > end\_year:  
 messagebox.showerror("Ошибка", "Начальный год должен быть меньше или равен конечному")  
 return  
  
 # Вызов метода для отображения результатов  
 self.show\_authors\_by\_birth\_year\_range(start\_year, end\_year)  
 except ValueError:  
 messagebox.showerror("Ошибка", "Годы должны быть целыми числами")  
  
 def show\_books\_by\_page\_count\_form(self):  
 *"""Отображение формы для поиска книг по количеству страниц"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для формы  
 form\_frame = ttk.Frame(self.root, padding="20")  
 form\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(form\_frame, text="Поиск книг по количеству страниц", font=("Arial", 16)).pack(pady=10)  
  
 # Поля формы  
 # Минимальное количество страниц  
 pages\_frame = ttk.Frame(form\_frame)  
 pages\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(pages\_frame, text="Минимум страниц:").pack(side=tk.LEFT)  
 pages\_entry = ttk.Entry(pages\_frame, width=30)  
 pages\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(form\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Найти",  
 command=lambda: self.find\_books\_by\_page\_count(  
 pages\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Назад",  
 command=self.show\_main\_menu  
 ).pack(side=tk.LEFT, padx=5)  
  
 def find\_books\_by\_page\_count(self, min\_pages\_str):  
 *"""Поиск и отображение книг по минимальному количеству страниц"""* try:  
 # Проверка и преобразование ввода  
 if not min\_pages\_str:  
 messagebox.showerror("Ошибка", "Необходимо указать минимальное количество страниц")  
 return  
  
 min\_pages = int(min\_pages\_str)  
  
 if min\_pages < 0:  
 messagebox.showerror("Ошибка", "Количество страниц должно быть положительным числом")  
 return  
  
 # Вызов метода для отображения результатов  
 self.show\_books\_by\_page\_count(min\_pages)  
 except ValueError:  
 messagebox.showerror("Ошибка", "Количество страниц должно быть целым числом")  
  
 def show\_authors\_by\_book\_count\_form(self):  
 *"""Отображение формы для поиска авторов по количеству книг"""* # Очистка рабочей области  
 self.clear\_workspace()  
  
 # Создание фрейма для формы  
 form\_frame = ttk.Frame(self.root, padding="20")  
 form\_frame.pack(expand=True)  
  
 # Заголовок  
 ttk.Label(form\_frame, text="Поиск авторов по количеству книг", font=("Arial", 16)).pack(pady=10)  
  
 # Поля формы  
 # Минимальное количество книг  
 books\_frame = ttk.Frame(form\_frame)  
 books\_frame.pack(pady=5, fill=tk.X)  
 ttk.Label(books\_frame, text="Минимум книг:").pack(side=tk.LEFT)  
 books\_entry = ttk.Entry(books\_frame, width=30)  
 books\_entry.pack(side=tk.LEFT, padx=5)  
  
 # Кнопки  
 button\_frame = ttk.Frame(form\_frame)  
 button\_frame.pack(pady=10)  
  
 ttk.Button(button\_frame, text="Найти",  
 command=lambda: self.find\_authors\_by\_book\_count(  
 books\_entry.get()  
 )).pack(side=tk.LEFT, padx=5)  
  
 ttk.Button(button\_frame, text="Назад",  
 command=self.show\_main\_menu  
 ).pack(side=tk.LEFT, padx=5)  
  
 def find\_authors\_by\_book\_count(self, min\_books\_str):  
 *"""Поиск и отображение авторов по минимальному количеству книг"""* try:  
 # Проверка и преобразование ввода  
 if not min\_books\_str:  
 messagebox.showerror("Ошибка", "Необходимо указать минимальное количество книг")  
 return  
  
 min\_books = int(min\_books\_str)  
  
 if min\_books < 0:  
 messagebox.showerror("Ошибка", "Количество книг должно быть положительным числом")  
 return  
  
 # Вызов метода для отображения результатов  
 self.show\_authors\_by\_book\_count(min\_books)  
 except ValueError:  
 messagebox.showerror("Ошибка", "Количество книг должно быть целым числом")  
  
 def run(self):  
 *"""Запуск приложения"""* self.root.mainloop()  
  
app = LibrarySystem()  
app.run()

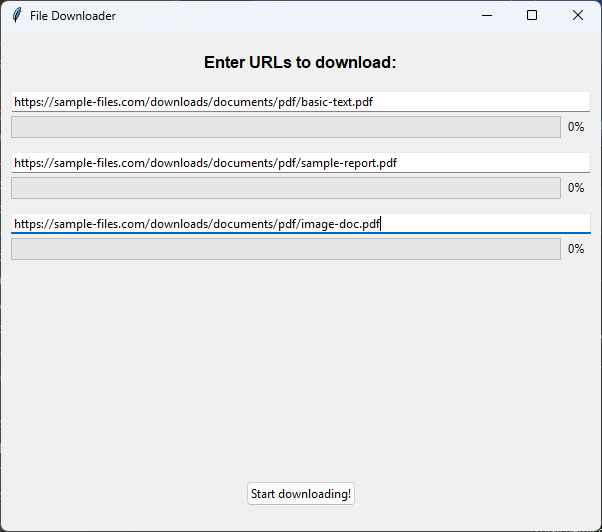
Вывод программы:

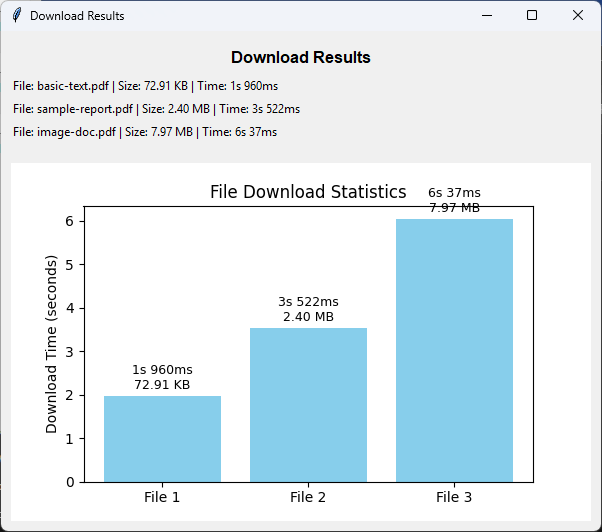


1. Код программы:

*"""  
Напишите приложение для загрузки файлов из интернета. В главном  
окне должно быть три текстовых поля, в которые можно вводить  
URL файла на закачку; под каждым из текстовых полей должны  
быть индикаторы загрузки и рядом поля с процентом загрузки  
каждого файла. Необходимо организовать возможность качать от  
одного до трех файлов параллельно.  
"""*import tkinter as tk  
from tkinter import ttk, messagebox  
import threading  
import requests  
import time  
import os  
import matplotlib.pyplot as plt  
from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg  
from io import BytesIO  
import re  
  
  
class DownloaderApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("File Downloader")  
 self.root.geometry("600x500")  
 self.root.resizable(True, True)  
  
 # Создаем переменные для хранения URL-адресов и активных потоков  
 self.urls = []  
 self.progress\_bars = []  
 self.progress\_labels = []  
 self.active\_threads = []  
 self.download\_times = [0, 0, 0]  
 self.file\_sizes = [0, 0, 0]  
 self.file\_names = ["", "", ""]  
 self.focused\_entry = None # Для отслеживания текущего активного поля  
  
 # Создаем и размещаем элементы интерфейса  
 self.create\_widgets()  
  
 # Привязываем обработчик Ctrl+V  
 self.root.bind('<Control-v>', self.paste\_from\_clipboard)  
  
 def create\_widgets(self):  
 # Главный фрейм  
 main\_frame = ttk.Frame(self.root, padding=10)  
 main\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Заголовок  
 header\_label = ttk.Label(main\_frame, text="Enter URLs to download:", font=("Arial", 12, "bold"))  
 header\_label.pack(pady=10)  
  
 # Контейнер для полей ввода и индикаторов  
 url\_frame = ttk.Frame(main\_frame)  
 url\_frame.pack(fill=tk.BOTH, expand=True)  
  
 # Создаем три поля для URL, индикаторы прогресса и метки  
 for i in range(3):  
 # Фрейм для группировки элементов  
 entry\_frame = ttk.Frame(url\_frame)  
 entry\_frame.pack(fill=tk.X, pady=5)  
  
 # Поле ввода URL  
 url\_entry = ttk.Entry(entry\_frame, width=50)  
 url\_entry.pack(fill=tk.X, pady=2)  
 # Привязываем обработчики фокуса к полю ввода  
 url\_entry.bind("<FocusIn>", lambda event, idx=i: self.on\_entry\_focus\_in(event, idx))  
 url\_entry.bind("<FocusOut>", self.on\_entry\_focus\_out)  
 self.urls.append(url\_entry)  
  
 # Фрейм для прогресс-бара и метки процента  
 progress\_frame = ttk.Frame(entry\_frame)  
 progress\_frame.pack(fill=tk.X, pady=2)  
  
 # Индикатор прогресса  
 progress\_bar = ttk.Progressbar(progress\_frame, orient=tk.HORIZONTAL, length=450, mode='determinate')  
 progress\_bar.pack(side=tk.LEFT, fill=tk.X, expand=True)  
 self.progress\_bars.append(progress\_bar)  
  
 # Метка для отображения процента  
 progress\_label = ttk.Label(progress\_frame, text="0%")  
 progress\_label.pack(side=tk.RIGHT, padx=5)  
 self.progress\_labels.append(progress\_label)  
  
 # Кнопка для начала загрузки  
 self.download\_button = ttk.Button(main\_frame, text="Start downloading!", command=self.start\_download)  
 self.download\_button.pack(pady=15)  
  
 # Устанавливаем фокус на первое поле ввода по умолчанию  
 self.urls[0].focus\_set()  
 self.focused\_entry = 0  
  
 def on\_entry\_focus\_in(self, event, idx):  
 *"""Обработчик события получения фокуса полем ввода"""* self.focused\_entry = idx  
  
 def on\_entry\_focus\_out(self, event):  
 *"""Обработчик события потери фокуса полем ввода"""* self.focused\_entry = None  
  
 def paste\_from\_clipboard(self, event):  
 *"""Обработчик для вставки из буфера обмена по Ctrl+V"""* try:  
 # Если есть активное поле ввода  
 if self.focused\_entry is not None:  
 # Получаем текст из буфера обмена  
 clipboard\_text = self.root.clipboard\_get()  
 # Вставляем в текущее поле  
 self.urls[self.focused\_entry].delete(0, tk.END)  
 self.urls[self.focused\_entry].insert(0, clipboard\_text)  
 return "break" # Предотвращаем стандартную обработку Ctrl+V  
 except Exception as e:  
 # Игнорируем ошибки буфера обмена  
 pass  
  
 def start\_download(self):  
 # Сбрасываем прогресс и останавливаем активные потоки  
 self.stop\_active\_threads()  
  
 # Сбрасываем информацию о предыдущих загрузках  
 self.download\_times = [0, 0, 0]  
 self.file\_sizes = [0, 0, 0]  
 self.file\_names = ["", "", ""]  
  
 # Проверяем введенные URL  
 urls\_to\_download = []  
 for i, url\_entry in enumerate(self.urls):  
 url = url\_entry.get().strip()  
 if url:  
 urls\_to\_download.append((i, url))  
 # Сбрасываем прогресс  
 self.progress\_bars[i]['value'] = 0  
 self.progress\_labels[i]['text'] = "0%"  
  
 if not urls\_to\_download:  
 messagebox.showwarning("Предупреждение", "Пожалуйста, введите хотя бы один URL для загрузки.")  
 return  
  
 # Запускаем загрузку в отдельных потоках  
 for idx, url in urls\_to\_download:  
 thread = threading.Thread(target=self.download\_file, args=(idx, url))  
 thread.daemon = True  
 thread.start()  
 self.active\_threads.append(thread)  
  
 def download\_file(self, idx, url):  
 try:  
 start\_time = time.time()  
  
 # Отправляем HEAD-запрос для получения размера файла  
 response = requests.head(url)  
 total\_size = int(response.headers.get('content-length', 0))  
  
 # Получаем имя файла из URL  
 if 'content-disposition' in response.headers:  
 cd = response.headers['content-disposition']  
 filename = re.findall('filename="(.+)"', cd) or re.findall('filename=(.+)', cd)  
 if filename:  
 filename = filename[0]  
 else:  
 filename = os.path.basename(url)  
 else:  
 filename = os.path.basename(url)  
  
 # Если имя пустое, используем часть URL  
 if not filename:  
 filename = f"file\_{idx + 1}"  
  
 self.file\_names[idx] = filename  
  
 # Создаем объект BytesIO для хранения загружаемых данных  
 file\_data = BytesIO()  
  
 # Инициируем загрузку по частям  
 response = requests.get(url, stream=True)  
 downloaded\_size = 0  
  
 for chunk in response.iter\_content(chunk\_size=4096):  
 if chunk:  
 file\_data.write(chunk)  
 downloaded\_size += len(chunk)  
  
 # Обновляем прогресс  
 if total\_size > 0:  
 progress = (downloaded\_size / total\_size) \* 100  
 # Обновляем интерфейс из основного потока  
 self.root.after(0, lambda prog=progress, i=idx: self.update\_progress(i, prog))  
  
 # Сохраняем файл  
 with open(filename, 'wb') as f:  
 f.write(file\_data.getvalue())  
  
 # Завершение загрузки  
 end\_time = time.time()  
 download\_time = end\_time - start\_time  
 self.download\_times[idx] = download\_time  
 self.file\_sizes[idx] = downloaded\_size  
  
 # Обновляем прогресс до 100%  
 self.root.after(0, lambda i=idx: self.update\_progress(i, 100))  
  
 # Проверяем, все ли загрузки завершены  
 self.root.after(100, self.check\_all\_downloads\_completed)  
  
 except Exception as e:  
 self.root.after(0, lambda err=str(e), i=idx: self.show\_error(i, err))  
  
 def update\_progress(self, idx, progress):  
 self.progress\_bars[idx]['value'] = progress  
 self.progress\_labels[idx]['text'] = f"{int(progress)}%"  
  
 def show\_error(self, idx, error\_message):  
 self.progress\_labels[idx]['text'] = "Error"  
 messagebox.showerror("Ошибка загрузки", f"Не удалось загрузить файл #{idx + 1}: {error\_message}")  
  
 def check\_all\_downloads\_completed(self):  
 all\_completed = True  
 for i, url\_entry in enumerate(self.urls):  
 if url\_entry.get().strip() and self.progress\_bars[i]['value'] < 100:  
 all\_completed = False  
 break  
  
 if all\_completed and any(self.download\_times):  
 self.show\_results()  
  
 def show\_results(self):  
 # Создаем всплывающее окно с графиком  
 results\_window = tk.Toplevel(self.root)  
 results\_window.title("Download Results")  
 results\_window.geometry("600x500")  
  
 # Создаем метки для отображения времени загрузки и размеров файлов  
 info\_frame = ttk.Frame(results\_window, padding=10)  
 info\_frame.pack(fill=tk.X)  
  
 # Добавляем заголовок  
 ttk.Label(info\_frame, text="Download Results", font=("Arial", 12, "bold")).pack(pady=5)  
  
 # Выводим информацию о загрузке для каждого файла  
 for i in range(3):  
 if self.download\_times[i] > 0:  
 time\_str = self.format\_time(self.download\_times[i])  
 size\_str = self.format\_size(self.file\_sizes[i])  
 file\_name = self.file\_names[i]  
 info\_text = f"File: {file\_name} | Size: {size\_str} | Time: {time\_str}"  
 ttk.Label(info\_frame, text=info\_text).pack(anchor='w', pady=2)  
  
 # Создаем график  
 fig = plt.Figure(figsize=(8, 6), dpi=100)  
 ax = fig.add\_subplot(111)  
  
 # Подготавливаем данные для графика  
 labels = []  
 times = []  
 sizes = []  
  
 for i in range(3):  
 if self.download\_times[i] > 0:  
 labels.append(f"File {i + 1}")  
 times.append(self.download\_times[i])  
 sizes.append(self.file\_sizes[i])  
  
 # Рисуем столбчатую диаграмму  
 bars = ax.bar(labels, times, color='skyblue')  
  
 # Добавляем метки с временем и размером  
 for i, bar in enumerate(bars):  
 time\_str = self.format\_time(times[i])  
 size\_str = self.format\_size(sizes[i])  
 ax.text(bar.get\_x() + bar.get\_width() / 2, bar.get\_height() + 0.1,  
 f"{time\_str}\n{size\_str}",  
 ha='center', va='bottom', fontsize=9)  
  
 ax.set\_ylabel('Download Time (seconds)')  
 ax.set\_title('File Download Statistics')  
  
 # Добавляем график на форму  
 canvas = FigureCanvasTkAgg(fig, master=results\_window)  
 canvas.draw()  
 canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True, padx=10, pady=10)  
  
 def format\_time(self, seconds):  
 *"""Форматирует время в формате '2s 322ms'"""* s = int(seconds)  
 ms = int((seconds - s) \* 1000)  
 return f"{s}s {ms}ms"  
  
 def format\_size(self, size\_bytes):  
 *"""Форматирует размер файла в читаемый вид"""* for unit in ['B', 'KB', 'MB', 'GB', 'TB']:  
 if size\_bytes < 1024.0:  
 return f"{size\_bytes:.2f} {unit}"  
 size\_bytes /= 1024.0  
 return f"{size\_bytes:.2f} PB"  
  
 def stop\_active\_threads(self):  
 # Очищаем список активных потоков  
 self.active\_threads.clear()  
  
  
def main():  
 root = tk.Tk()  
 app = DownloaderApp(root)  
 root.mainloop()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Вывод программы:

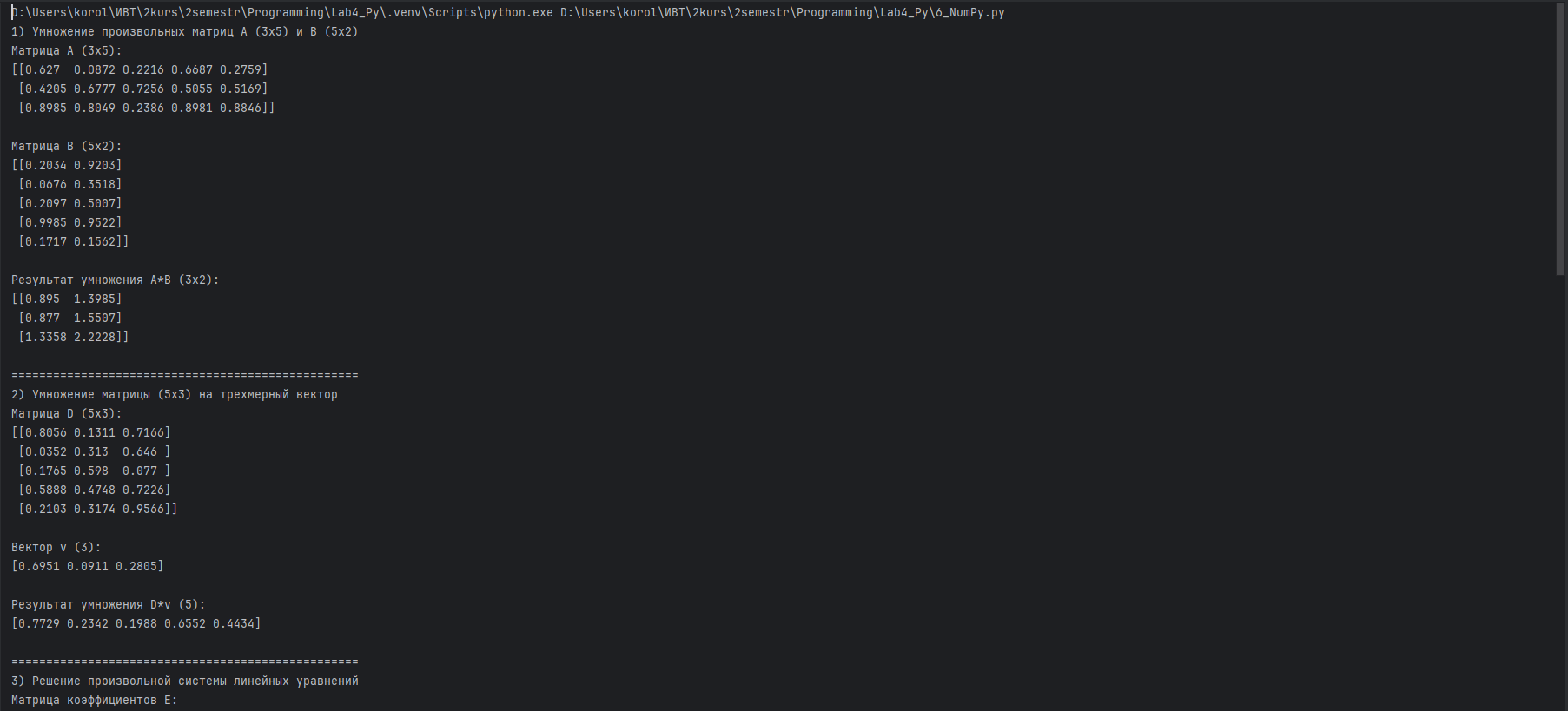




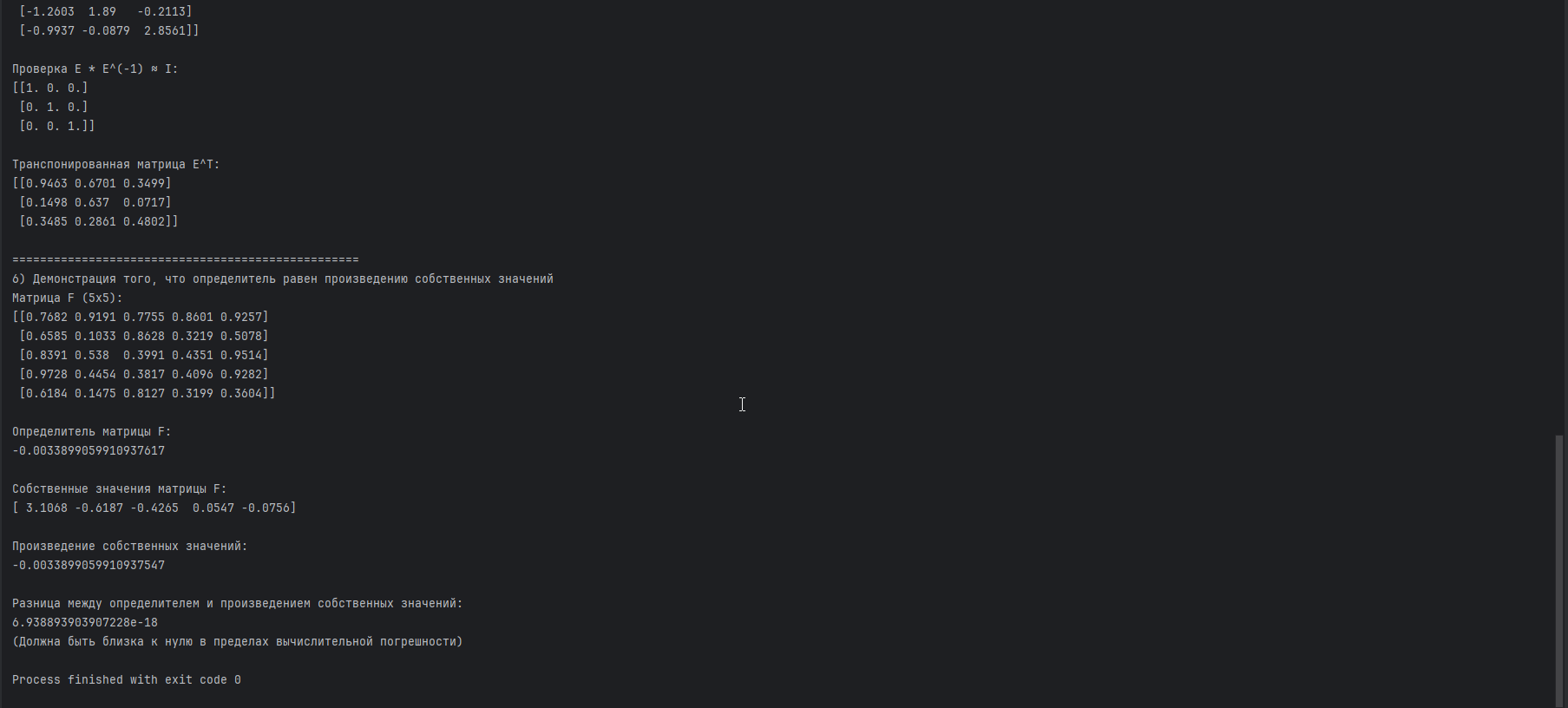
1. Код программы:

*"""  
С помощью модуля numPy реализуйте следующие операции: 1)  
умножение произвольных матриц А (размерности 3х5) и В (5х2); 2)  
умножение матрицы (5х3) на трехмерный вектор; 3) решение  
произвольной системы линейных уравнений; 4) расчет определителя  
матрицы; 5) получение обратной и транспонированной матриц.  
Также продемонстрируйте на примере матрицы 5х5 тот факт, что  
определитель равен произведению собственных значений матрицы.  
"""*import numpy as np  
np.set\_printoptions(precision=4, suppress=True) # Для более компактного вывода  
  
print("1) Умножение произвольных матриц A (3x5) и B (5x2)")  
# Создаем произвольные матрицы с помощью генератора случайных чисел  
A = np.random.rand(3, 5)  
B = np.random.rand(5, 2)  
  
print("Матрица A (3x5):")  
print(A)  
print("\nМатрица B (5x2):")  
print(B)  
  
# Умножение матриц  
C = np.dot(A, B) # или C = A @ B (оператор матричного умножения в Python 3.5+)  
print("\nРезультат умножения A\*B (3x2):")  
print(C)  
  
print("\n" + "="\*50)  
  
print("2) Умножение матрицы (5x3) на трехмерный вектор")  
# Создаем произвольную матрицу 5x3  
D = np.random.rand(5, 3)  
# Создаем произвольный вектор размерности 3  
v = np.random.rand(3)  
  
print("Матрица D (5x3):")  
print(D)  
print("\nВектор v (3):")  
print(v)  
  
# Умножение матрицы на вектор  
result = np.dot(D, v) # или result = D @ v  
print("\nРезультат умножения D\*v (5):")  
print(result)  
  
print("\n" + "="\*50)  
  
print("3) Решение произвольной системы линейных уравнений")  
# Создаем случайную матрицу коэффициентов  
n = 3 # размерность системы  
E = np.random.rand(n, n)  
# Создаем случайный вектор правой части  
b = np.random.rand(n)  
  
print("Матрица коэффициентов E:")  
print(E)  
print("\nВектор правой части b:")  
print(b)  
  
# Решение системы уравнений E\*x = b  
x = np.linalg.solve(E, b)  
print("\nРешение системы E\*x = b:")  
print(x)  
  
# Проверка решения  
print("\nПроверка E\*x ≈ b:")  
print(np.dot(E, x))  
  
print("\n" + "="\*50)  
  
print("4) Расчет определителя матрицы")  
# Используем ранее созданную матрицу E  
det\_E = np.linalg.det(E)  
print("Определитель матрицы E:")  
print(det\_E)  
  
print("\n" + "="\*50)  
  
print("5) Получение обратной и транспонированной матриц")  
# Обратная матрица  
E\_inv = np.linalg.inv(E)  
print("Обратная матрица E^(-1):")  
print(E\_inv)  
  
# Проверка, что E \* E^(-1) ≈ I (единичная матрица)  
print("\nПроверка E \* E^(-1) ≈ I:")  
print(np.dot(E, E\_inv))  
  
# Транспонированная матрица  
E\_trans = E.T # или np.transpose(E)  
print("\nТранспонированная матрица E^T:")  
print(E\_trans)  
  
print("\n" + "="\*50)  
  
print("6) Демонстрация того, что определитель равен произведению собственных значений")  
# Создаем случайную матрицу 5x5  
F = np.random.rand(5, 5)  
print("Матрица F (5x5):")  
print(F)  
  
# Вычисляем определитель  
det\_F = np.linalg.det(F)  
print("\nОпределитель матрицы F:")  
print(det\_F)  
  
# Вычисляем собственные значения  
eigenvalues = np.linalg.eigvals(F)  
print("\nСобственные значения матрицы F:")  
print(eigenvalues)  
  
# Вычисляем произведение собственных значений  
product\_eigenvalues = np.prod(eigenvalues)  
print("\nПроизведение собственных значений:")  
print(product\_eigenvalues)  
  
print("\nРазница между определителем и произведением собственных значений:")  
print(np.abs(det\_F - product\_eigenvalues))  
print("(Должна быть близка к нулю в пределах вычислительной погрешности)")

Вывод программы:



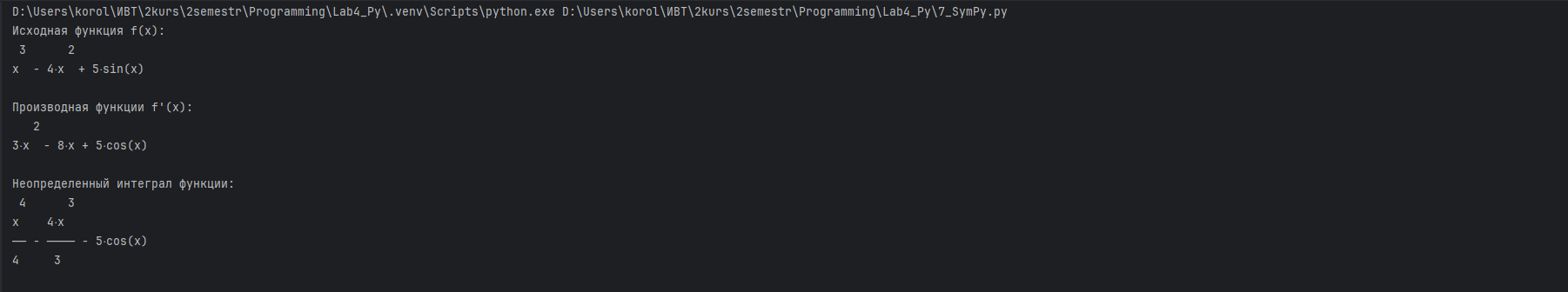


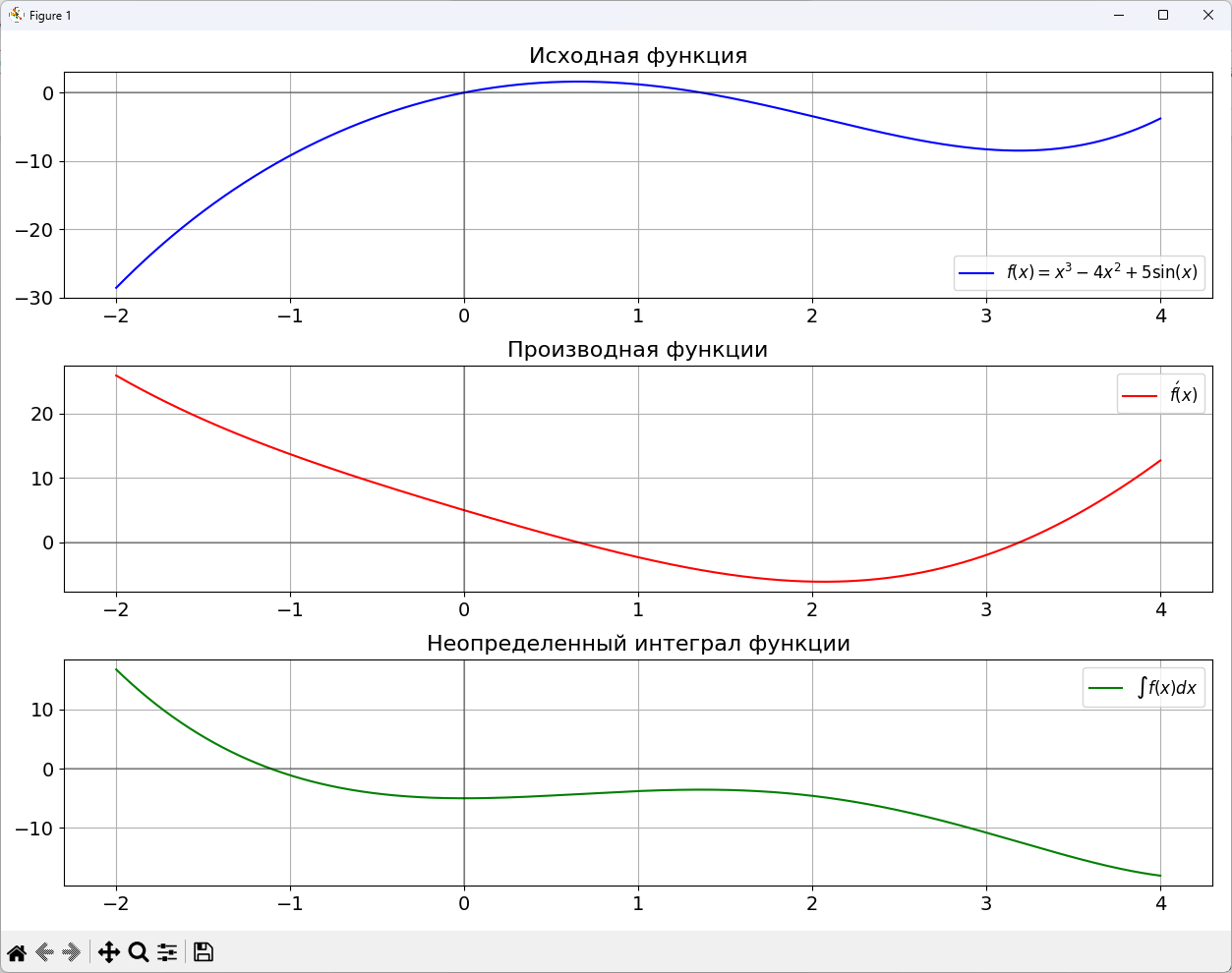


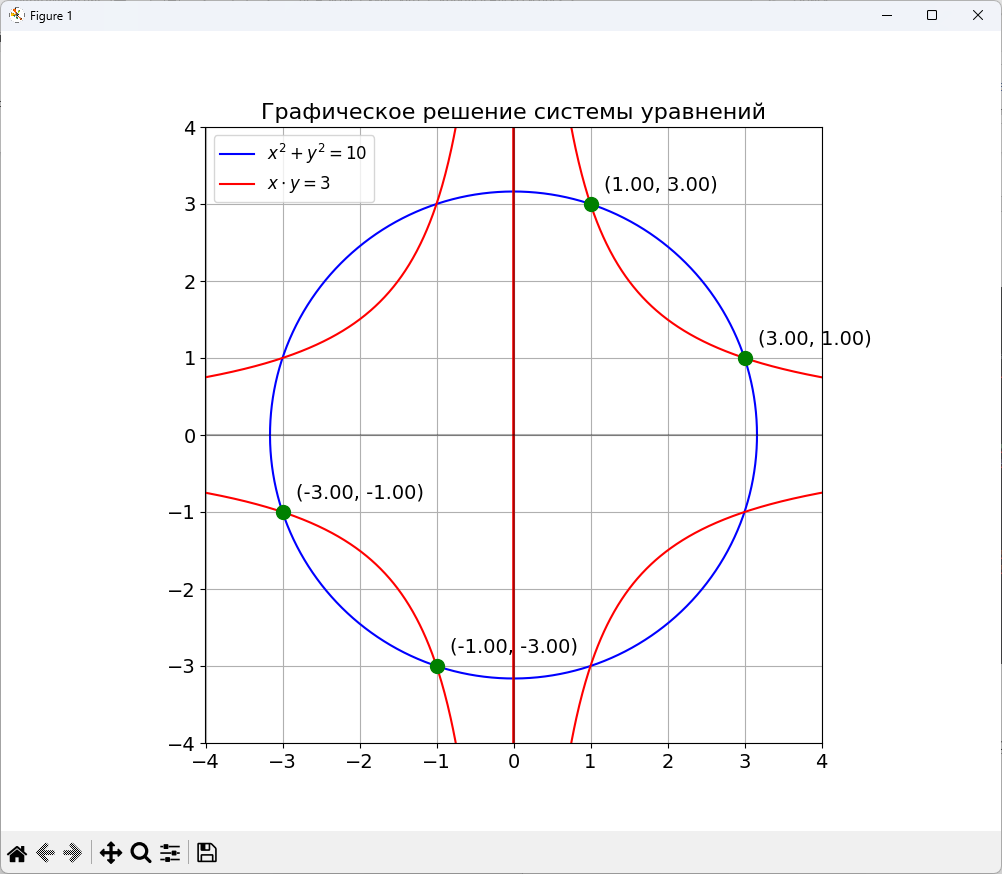
1. Код программы:

*"""  
Выберите произвольную дифференцируемую и интегрируемую  
функцию одной переменной. С помощью модуля symPy найдите и  
отобразите ее производную и интеграл в аналитическом и  
графическом виде. Напишите код для решения произвольного  
нелинейного урванения и системы нелинейных уравнений.  
"""*import sympy as sp  
import numpy as np  
import matplotlib.pyplot as plt  
from sympy.plotting import plot  
from matplotlib import rcParams  
  
# Улучшаем отображение графиков  
rcParams['figure.figsize'] = 12, 8  
rcParams['font.size'] = 14  
  
# Определяем символьную переменную  
x = sp.Symbol('x')  
  
# Выберем функцию: f(x) = x^3 - 4\*x^2 + 5\*sin(x)  
f = x\*\*3 - 4\*x\*\*2 + 5\*sp.sin(x)  
  
# 1. ДИФФЕРЕНЦИРОВАНИЕ  
# Найдем производную функции  
f\_prime = sp.diff(f, x)  
  
# 2. ИНТЕГРИРОВАНИЕ  
# Найдем неопределенный интеграл  
f\_integral = sp.integrate(f, x)  
  
# 3. ВЫВОД РЕЗУЛЬТАТОВ В АНАЛИТИЧЕСКОМ ВИДЕ  
print("Исходная функция f(x):")  
print(sp.pretty(f))  
print("\nПроизводная функции f'(x):")  
print(sp.pretty(f\_prime))  
print("\nНеопределенный интеграл функции:")  
print(sp.pretty(f\_integral))  
  
# 4. ГРАФИЧЕСКОЕ ПРЕДСТАВЛЕНИЕ  
# Преобразуем функции в численные функции для построения графиков  
f\_lambda = sp.lambdify(x, f, "numpy")  
f\_prime\_lambda = sp.lambdify(x, f\_prime, "numpy")  
f\_integral\_lambda = sp.lambdify(x, f\_integral, "numpy")  
  
# Создаем массив значений x  
x\_vals = np.linspace(-2, 4, 1000)  
  
# Вычисляем значения функций  
f\_vals = f\_lambda(x\_vals)  
f\_prime\_vals = f\_prime\_lambda(x\_vals)  
f\_integral\_vals = f\_integral\_lambda(x\_vals)  
  
# Построение графиков  
plt.figure(figsize=(14, 10))  
  
# График функции  
plt.subplot(3, 1, 1)  
plt.plot(x\_vals, f\_vals, 'b-', label=r'$f(x) = x^3 - 4x^2 + 5\sin(x)$')  
plt.grid(True)  
plt.legend(fontsize=12)  
plt.title('Исходная функция', fontsize=16)  
plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)  
plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)  
  
# График производной  
plt.subplot(3, 1, 2)  
plt.plot(x\_vals, f\_prime\_vals, 'r-', label=r'$f\'(x)$')  
plt.grid(True)  
plt.legend(fontsize=12)  
plt.title('Производная функции', fontsize=16)  
plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)  
plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)  
  
# График интеграла  
plt.subplot(3, 1, 3)  
plt.plot(x\_vals, f\_integral\_vals, 'g-', label=r'$\int f(x) dx$')  
plt.grid(True)  
plt.legend(fontsize=12)  
plt.title('Неопределенный интеграл функции', fontsize=16)  
plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)  
plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)  
  
plt.tight\_layout()  
plt.savefig('function\_analysis.png')  
plt.show()  
  
# 5. РЕШЕНИЕ НЕЛИНЕЙНОГО УРАВНЕНИЯ  
print("\n\nРЕШЕНИЕ НЕЛИНЕЙНОГО УРАВНЕНИЯ")  
print("===============================")  
  
# Определим нелинейное уравнение: x^3 - 6\*x^2 + 11\*x - 6 = 0  
eq = x\*\*3 - 6\*x\*\*2 + 11\*x - 6  
  
print(f"Уравнение: {eq} = 0")  
  
# Решаем уравнение  
solutions = sp.solve(eq, x)  
print("\nРешения:")  
for i, sol in enumerate(solutions):  
 print(f"x\_{i+1} = {sol}")  
  
# Проверка решений  
print("\nПроверка решений:")  
for i, sol in enumerate(solutions):  
 result = eq.subs(x, sol)  
 print(f"Подставляем x\_{i+1} = {sol} в уравнение: {result}")  
  
# 6. РЕШЕНИЕ СИСТЕМЫ НЕЛИНЕЙНЫХ УРАВНЕНИЙ  
print("\n\nРЕШЕНИЕ СИСТЕМЫ НЕЛИНЕЙНЫХ УРАВНЕНИЙ")  
print("=====================================")  
  
# Определим вторую символьную переменную  
y = sp.Symbol('y')  
  
# Определим систему уравнений:  
# 1. x^2 + y^2 = 10  
# 2. x\*y = 3  
eq1 = x\*\*2 + y\*\*2 - 10  
eq2 = x\*y - 3  
  
print("Система уравнений:")  
print(f"1. {eq1} = 0 (x^2 + y^2 = 10)")  
print(f"2. {eq2} = 0 (x\*y = 3)")  
  
# Решаем систему уравнений  
system\_solutions = sp.solve((eq1, eq2), (x, y))  
print("\nРешения системы:")  
for i, sol in enumerate(system\_solutions):  
 print(f"Решение {i+1}: x = {sol[0]}, y = {sol[1]}")  
  
# Проверка решений  
print("\nПроверка решений системы:")  
for i, sol in enumerate(system\_solutions):  
 result1 = eq1.subs([(x, sol[0]), (y, sol[1])])  
 result2 = eq2.subs([(x, sol[0]), (y, sol[1])])  
 print(f"Решение {i+1}: x = {sol[0]}, y = {sol[1]}")  
 print(f"Подставляем в первое уравнение: {result1}")  
 print(f"Подставляем во второе уравнение: {result2}")  
 print()  
  
# Визуализация системы уравнений  
plt.figure(figsize=(10, 8))  
plt.title('Графическое решение системы уравнений', fontsize=16)  
  
# Построение окружности x^2 + y^2 = 10  
theta = np.linspace(0, 2\*np.pi, 1000)  
circle\_x = np.sqrt(10) \* np.cos(theta)  
circle\_y = np.sqrt(10) \* np.sin(theta)  
plt.plot(circle\_x, circle\_y, 'b-', label=r'$x^2 + y^2 = 10$')  
  
# Построение гиперболы x\*y = 3  
xy\_vals = np.linspace(-5, 5, 1000)  
hyperbola\_y1 = 3 / xy\_vals  
hyperbola\_y2 = -3 / xy\_vals # отрицательная ветвь  
plt.plot(xy\_vals, hyperbola\_y1, 'r-', label=r'$x \cdot y = 3$')  
plt.plot(-xy\_vals, hyperbola\_y1, 'r-')  
  
# Отметим точки пересечения  
for sol in system\_solutions:  
 plt.plot(float(sol[0]), float(sol[1]), 'go', markersize=10)  
 plt.annotate(f'({float(sol[0]):.2f}, {float(sol[1]):.2f})',  
 (float(sol[0]), float(sol[1])),  
 xytext=(10, 10),  
 textcoords='offset points')  
  
plt.grid(True)  
plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)  
plt.axvline(x=0, color='k', linestyle='-', alpha=0.3)  
plt.xlim(-4, 4)  
plt.ylim(-4, 4)  
plt.legend(fontsize=12)  
plt.gca().set\_aspect('equal')  
plt.savefig('system\_solution.png')  
plt.show()

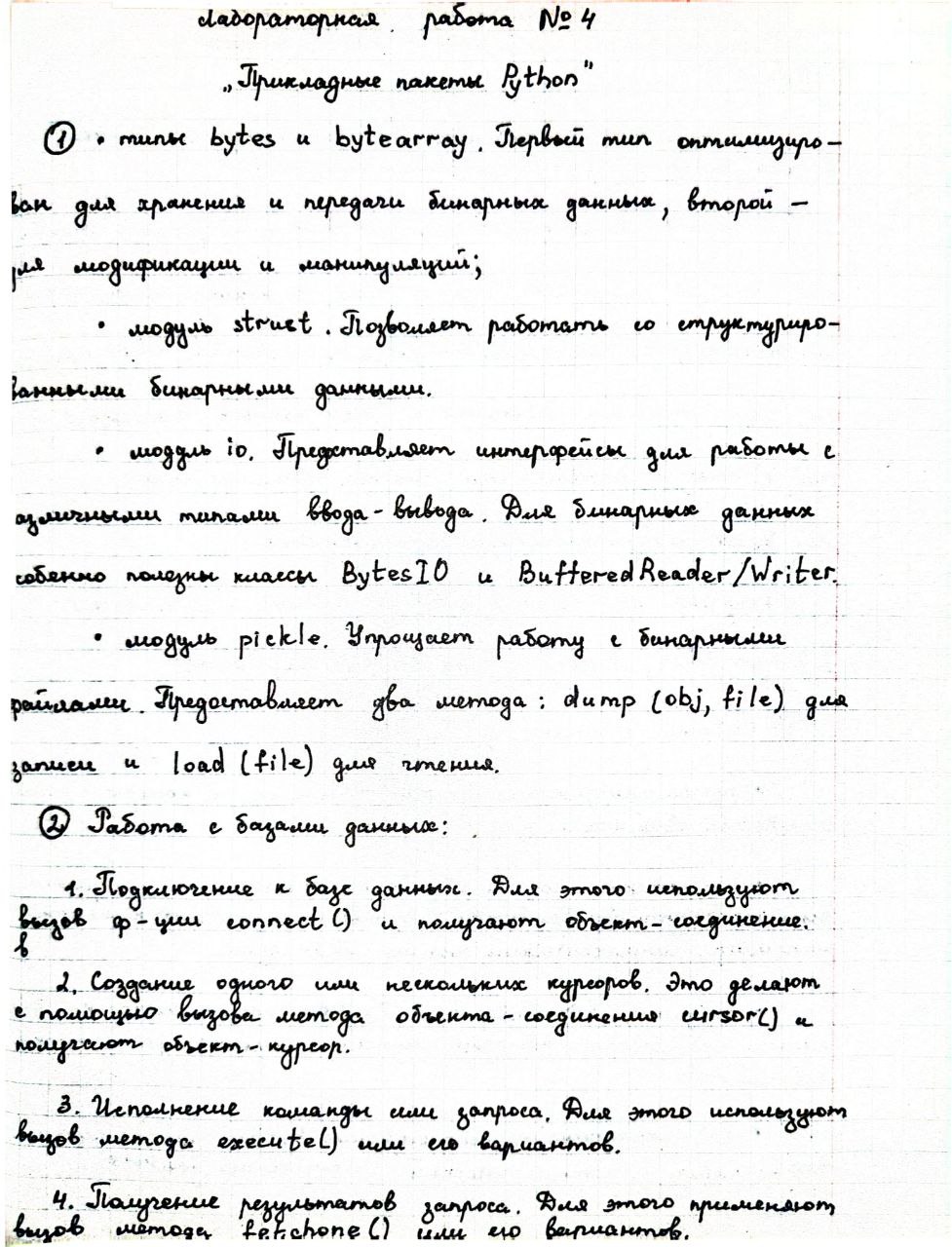
Вывод программы:

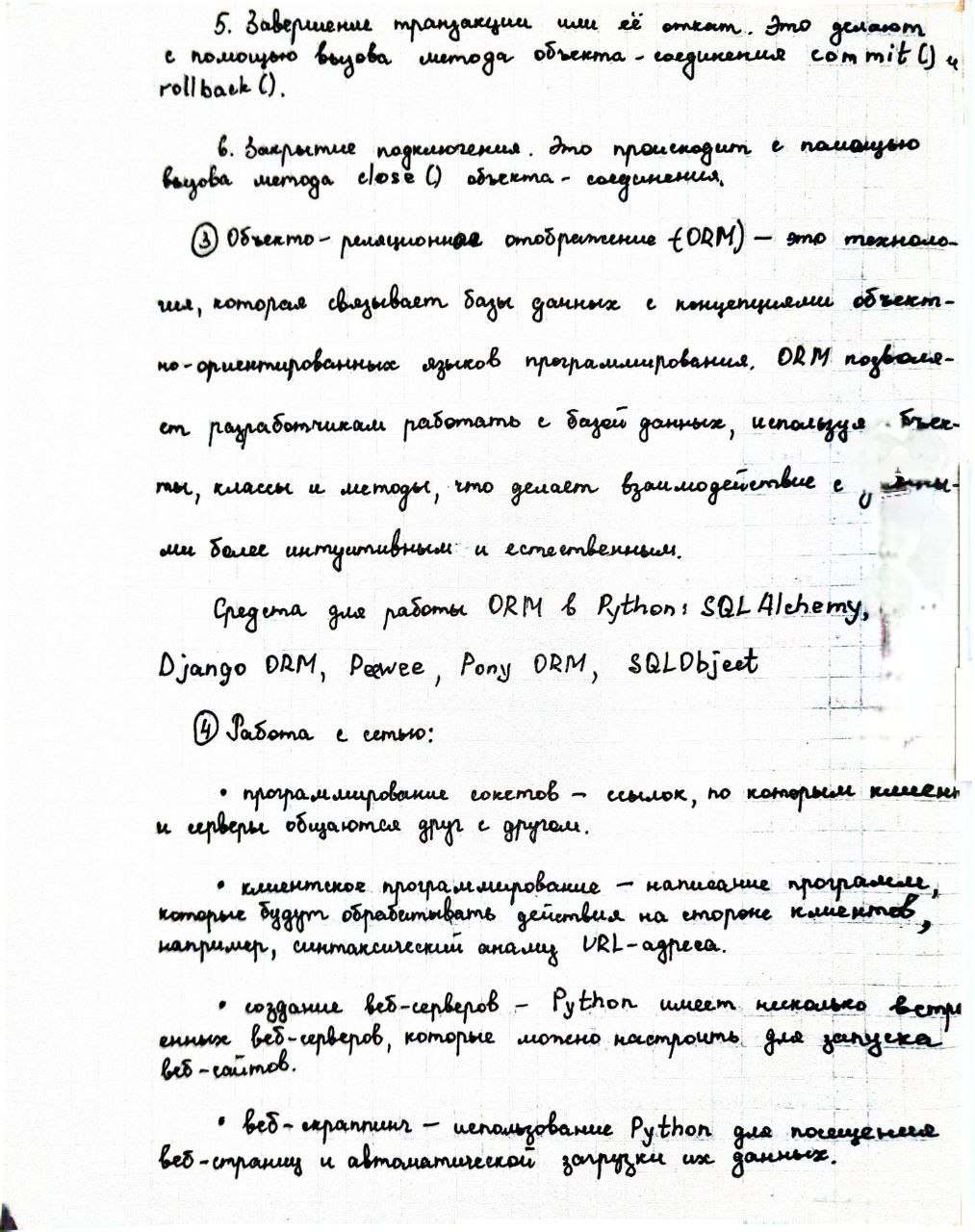


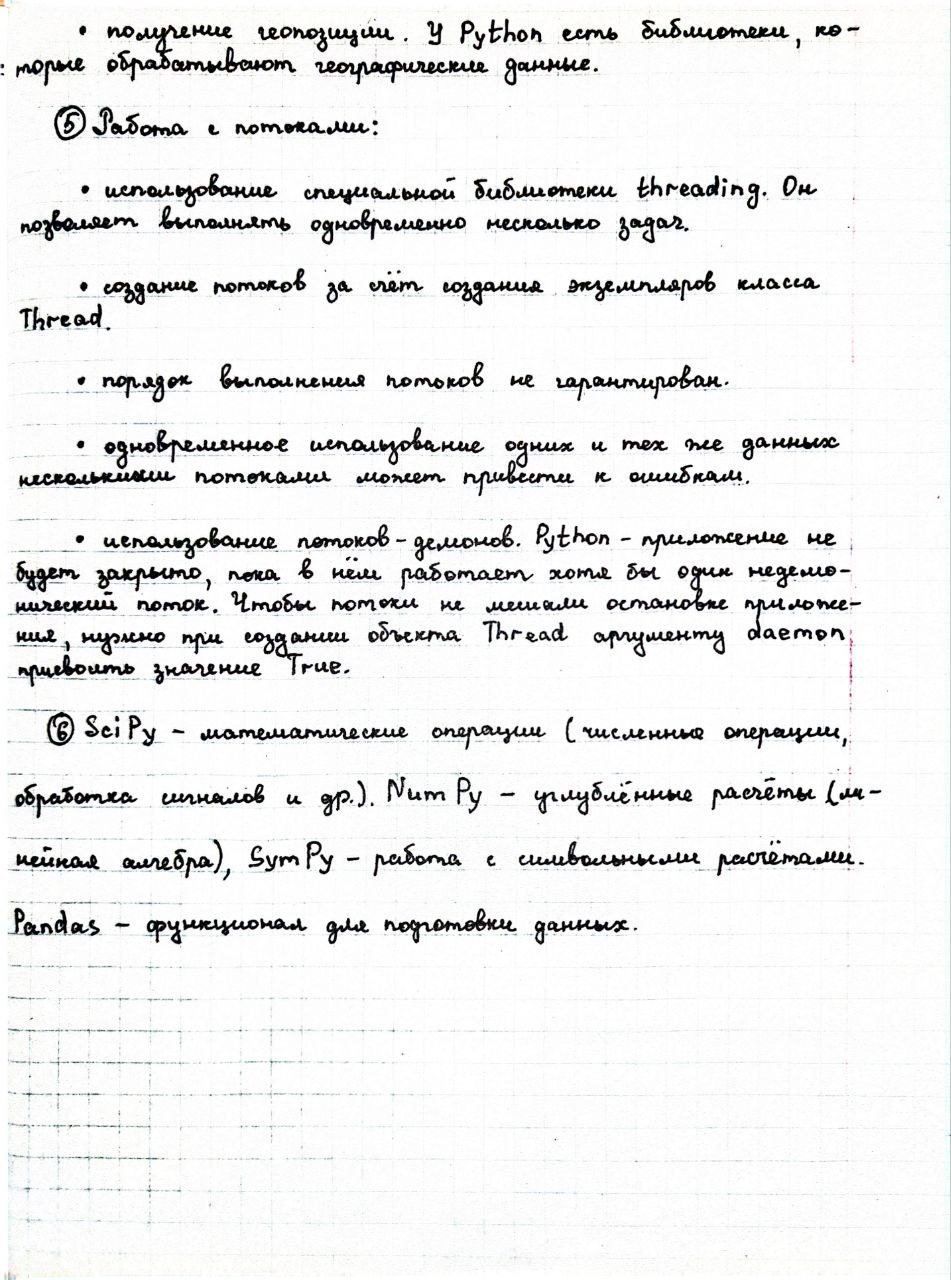




Контрольные вопросы







Вывод: Научился проектировать и создавать одно- и многопоточные скрипты с графическим интерфейсом, позволяющие работать с базами данных, сетью и слабоструктурированной информацией.