

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий  
Кафедра «Прикладная математика и информатика»  
Дисциплина «Языки программирования для анализа и обработки данных»

**КУРСОВАЯ РАБОТА**

Тема \_\_\_\_\_ «Анализ и обработка данных с использованием структуры данных  
Словарь «Фильмотека»» \_\_\_\_\_

Выполнил студент \_\_\_\_\_ / В. Д. Шувалова /  
подпись инициалы, фамилия

Курс \_\_\_\_\_ 3 \_\_\_\_\_ Группа \_\_\_\_\_ ПМбд-31 \_\_\_\_\_

Направление/ специальность \_\_\_\_\_ 01.03.04 Прикладная математика \_\_\_\_\_

Руководитель \_\_\_\_\_ доцент кафедры ПМИ, к.т.н., доцент \_\_\_\_\_  
должность, ученая степень, ученое звание  
Кадырова Гульнара Ривальевна \_\_\_\_\_  
фамилия, имя, отчество

Дата сдачи:  
« \_\_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_\_ г.

Дата защиты:  
« \_\_\_\_\_ » \_\_\_\_\_ 20 \_\_\_\_\_ г.

Оценка: \_\_\_\_\_

Ульяновск, 2025 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных систем и технологий  
Кафедра «Прикладная математика и информатика»  
Дисциплина «Языки программирования для анализа и обработки данных»

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ (РАБОТУ)**

студенту ПМбд-31 Шувалова В. Д.  
группа фамилия, инициалы

Тема проекта (работы) «Анализ и обработка данных с использованием структуры данных Словарь «Фильмотека»»

Срок сдачи законченной работы «\_\_» \_\_\_\_\_ 20\_\_ г.

Исходные данные к работе 1. Разработать программу на Python для работы с данными о фильмах, используя структуру данных «словарь». Реализовать широкий функционал для анализа этих данных. 2. Организовать корректный ввод данных из файла. Данные в файле должны быть различных типов и форматов. 3. Создать удобный консольный интерфейс для взаимодействия с пользователем, предусмотрев обработку возможных ошибок ввода и вывод информативных сообщений о невозможности выполнения запроса.

(базовое предприятие, характер курсового проекта (работы): задание кафедры, инициативная НИР, рекомендуемая литература, материалы практики)

Содержание пояснительной записки (перечень подлежащих разработке вопросов)  
Введение 1. Теоретическая часть 1.1. Структура данных - Словарь 1.2. Функции в Python 1.3. Организация файлового ввода 2. Практическая часть 2.1. Описание структуры данных 2.2. Структура программы 2.3. Описание программы Заключение  
Список литературы Приложение

Перечень графического материала (с точным указанием обязательных чертежей)

Руководитель доцент кафедры ПМИ / Г. Р. Кадырова /  
должность подпись инициалы, фамилия

«\_\_» \_\_\_\_\_ 20\_\_ г.

Студент \_\_\_\_\_ / В. Д. Шувалова /  
подпись инициалы, фамилия

«\_\_» \_\_\_\_\_ 20\_\_ г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**ОТЗЫВ**  
**руководителя на курсовую работу**

студента Шуваловой Валерии Дмитриевны

фамилия, имя и отчество

Факультет информационных систем и технологий группа ПМбд-31 курс 3  
Дисциплина Языки программирования для анализа и обработки данных  
Тема работы «Анализ и обработка данных с использованием структуры данных  
Словарь «Фильмотека»»

Курсовая работа Шуваловой В.Д. посвящена разработке программного обеспечения для анализа и обработки данных фильмотеки с использованием структуры данных «словарь» в языке Python.

Целью работы являлось создание программы, способной эффективно хранить, обрабатывать и предоставлять информацию о фильмах с использованием возможностей словарей Python и формата JSON для хранения данных.

Для достижения этой цели были успешно решены следующие задачи. Изучены особенности работы со словарями в Python. Реализованы механизмы загрузки и сохранения данных в JSON-формате. Разработаны алгоритмы поиска и сортировки фильмов по различным критериям. Создан удобный консольный интерфейс взаимодействия с пользователем.

В результате была разработана программа, использующая Python 3.12 и среду разработки PyCharm Community Edition 2024. Программа демонстрирует эффективную работу со сложными структурами данных, корректную реализацию файловых операций с JSON, гибкую систему поиска по множеству параметров, надёжную обработку пользовательского ввода.

Особого внимания заслуживает грамотное использование вложенных словарей для хранения информации о фильмах, применение лямбда-функций для реализации сложных условий поиска, качественная обработка ошибок ввода, модульная структура программы.

Тема работы была выбрана студенткой самостоятельно, что свидетельствует о заинтересованности в предметной области. В процессе выполнения работы Шувалова В.Д. проявляла инициативу, своевременно информировала о ходе работы и консультировалась по возникающим вопросам.

Теоретическая часть работы хорошо проработана и подтверждается практической реализацией. Структура и содержание работы полностью соответствует поставленным задачам. Курсовая работа выполнена на высоком уровне, в установленные сроки.

Руководитель доцент кафедры ПМИ, к.т.н., доцент / Г.Р. Кадырова  
должность, учёная степень, ученое звание подпись инициалы, фамилия

«    »      20     г.

## Оглавление

Введение.....	5
1. Теоретическая часть .....	6
1.1. Структура данных - Словарь .....	6
1.2. Функции в Python .....	7
1.3. Организация файлового ввода .....	8
2. Практическая часть.....	10
2.1. Описание структуры данных.....	10
2.2. Структура программы .....	11
2.3. Описание программы .....	14
Заключение .....	18
Список литературы .....	19
Приложение .....	20

## Введение

Современные информационные системы часто работают с большими объёмами данных, требующими эффективных методов хранения и обработки. В таких задачах ключевую роль играют структуры данных, обеспечивающие быстрый доступ, модификацию и анализ информации. Одной из наиболее удобных и часто используемых структур является словарь, который позволяет эффективно выполнять операции поиска и обновления.

В данной курсовой работе рассматривается применение словаря для анализа и обработки данных в контексте фильмотеки – базы данных о фильмах. В качестве языка программирования выбран Python, так как он широко используется в сфере анализа данных, имеет много библиотек, поддерживает интеграцию с другими инструментами, что делает его универсальным выбором для задач хранения, обработки и анализа информации, включая управление фильмотекой.

Целью курсовой работы является разработка программы на Python, использующей структуру данных «словарь» для управления фильмотекой, включая загрузку, обработку данных и выполнение запросов от пользователя.

Для достижения этой цели были поставлены следующие задачи:

1. Изучить научную литературу, учебные пособия и электронные ресурсы, в которых приводится информация о структурах данных в Python, особенностях работы со словарями, методах анализа и обработки данных.
2. Программно реализовать решение указанной задачи.
3. Создать исполняемый файл программы в качестве конечного продукта.

Курсовая работа состоит из двух глав. В теоретической части рассматриваются ключевые аспекты работы со структурой данных «словарь» и функциями в Python, а также организация файлового ввода. В практической части представлена реализация программы.

Код программы написан на языке программирования Python версии 3.12 с использованием среды разработки PyCharm Community Edition версии 2024 и прикреплён в приложении.

## 1. Теоретическая часть

### 1.1. Структура данных - Словарь

Словарь (dict) в Python представляет собой изменяемую структуру данных, которая хранит информацию в виде пар «ключ-значение». Эта структура является часто используемой для эффективной обработки данных благодаря своей гибкости и высокой производительности.

В основе реализации словаря в Python лежит такая структура данных как хеш-таблица. Механизм хеширования заключается в том, что при добавлении пары «ключ-значение» в словарь Python вычисляет хеш ключа с помощью встроенной функции `hash()`. Хеш – это целое число, которое определяет конкретную ячейку в памяти, где будет храниться соответствующее ключу значение. Такой подход позволяет достигать средней асимптотической сложности равной  $O(1)$  для таких операций как добавление, удаление и поиск элементов в словаре.

Важной особенностью словарей является требование к ключам – они должны быть неизменяемыми. Такое ограничение связано с тем, что изменение ключа нарушило бы всю систему хранения, так как для него пришлось бы пересчитывать хеш. Поэтому в качестве ключей используют такие типы данных как: строки (str), целые числа (int), кортежи (tuple) без изменяемых элементов внутри них. Вещественные числа (float) не рекомендуется использовать в качестве ключей, так как могут возникнуть проблемы из-за ошибок округления. Значения могут быть как неизменяемыми, так и изменяемыми типами данных.

Также ключи словаря должны быть уникальными. При наличии нескольких ключей сохраняется значение последнего из них. Значения необязательно должны быть уникальными.

Благодаря встроенным методам словарей, таким как `keys()`, `values()`, `items()`, `get()` и другим, можно легко манипулировать данными. Например, метод `get()` позволяет безопасно получать значения, избегая ошибок при обращении к несуществующим ключам. Важно отметить, что методы `keys()`,

`values()` и `items()` возвращают такие итерируемые объекты как представления. Они автоматически обновляются при изменении словаря, что очень удобно для практических задач. Также словари в Python сохраняют порядок вставки элементов, начиная с версии 3.7.

## 1.2. Функции в Python

Функции представляют собой особым образом сгруппированный блок кода. Они являются фундаментальным элементом структурной парадигмы программирования. Функции в Python выполняют несколько важных задач. Во-первых, они позволяют избежать дублирования кода. Во-вторых, делают программу более читаемой, разбивая сложные задачи на более простые подзадачи. В-третьих, они обеспечивают изоляцию областей видимости переменных.

Создание пользовательских функций осуществляется с помощью ключевого слова `def`. После указания имени функции задаются её параметры в круглых скобках. Параметры (формальные параметры) – это локальные переменные, которые определяются при объявлении функции и которым присваиваются значения аргументов (фактических параметров) в момент вызова функции. Тело функции, оформленное в виде блока с отступом, содержит последовательность выполняемых инструкций. Функция может возвращать результат с помощью оператора `return`. Можно использовать несколько операторов `return` – их называют точками возврата из функции. Если оператор возврата отсутствует или указан без значения, функция возвращает `None`. Также значения параметров можно определять по умолчанию.

Особого внимания заслуживает механизм пространств имен и областей видимости в функциях. Функции создают собственные локальные пространства имён. Python использует LEGB-правило (L – Local, E – Enclosing, G – global, B – Built-in). Оно заключается в том, что сначала имя переменной ищется в локальной области видимости, то есть внутри самой функции, затем – в охватывающей (когда функция находится внутри функции, внешняя функция является охватывающей), далее – в глобальной, после – во

встроенной. Ключевые слова `global` и `nonlocal` позволяют управлять доступом к переменным из внешних областей.

Для создания краткосрочных и простых операций в Python предусмотрены `lambda`-функции. Это анонимные функции, объявляемые с помощью ключевого слова `lambda`. Они могут содержать только одно выражение и автоматически возвращают результат его вычисления. `Lambda`-функции особенно полезны в случаях, когда их требуется использовать в качестве аргумента в других функциях, например, `map()`, `filter()`, `sorted()`.

### **1.3. Организация файлового ввода**

В современных программных системах организация корректного файлового ввода представляет собой важную задачу, особенно при работе со структурированными данными. Формат JSON (JavaScript Object Notation) стал фактическим стандартом для хранения и передачи данных благодаря своей простоте, читаемости и широкой поддержке различными языками программирования. JSON-файлы нашли широкое применение в различных областях разработки. В веб-разработке они применяются для API-интерфейсов, а в научных вычислениях и анализе данных JSON позволяет хранить сложные структуры данных, сохраняя при этом возможность их просмотра и редактирования в текстовом виде.

В языке Python работа с JSON-файлами реализована через стандартный модуль `json`, который предоставляет разработчику весь необходимый функционал для взаимодействия с данными в этом формате.

Основным методом загрузки данных из JSON-файлов является функция `json.load()`, которая выполняет чтение и преобразование данных в соответствующие структуры Python. Важной особенностью является автоматическое преобразование типов: JSON-объекты становятся словарями Python, массивы – списками, а простые значения преобразуются в соответствующие типы данных Python.

Для обеспечения надёжности работы с файлами рекомендуется использовать менеджер контекста `with`, который гарантирует корректное



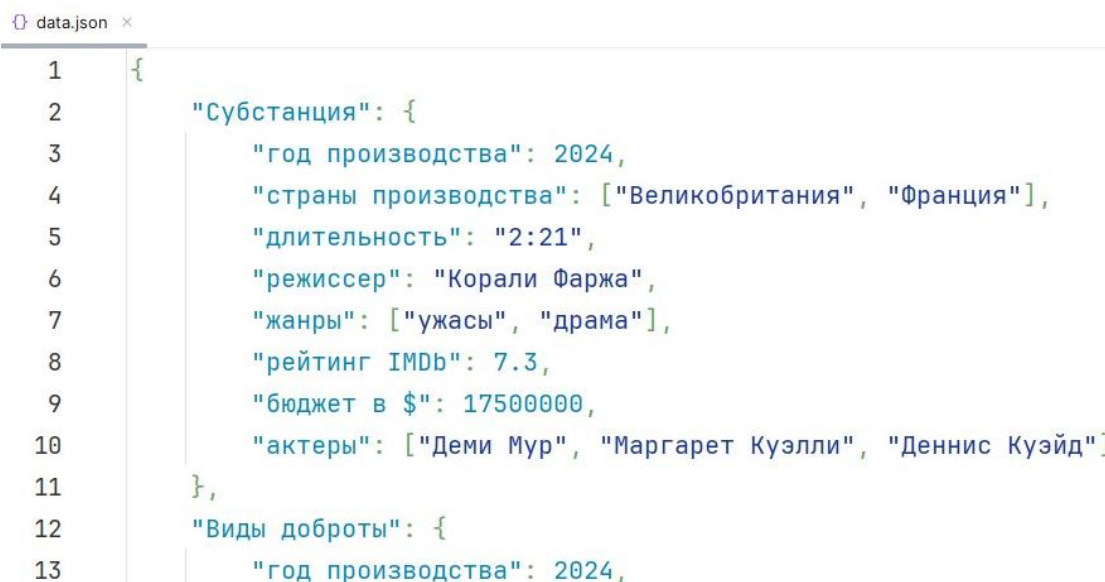
закрытие файлового дескриптора после завершения работы с файлом, даже в случае возникновения исключений. Это особенно важно в производственной среде, где утечки файловых дескрипторов могут привести к серьёзным проблемам. Также при работе с файловым вводом необходимо явно указывать кодировку, как правило UTF-8, чтобы обеспечить корректное чтение данных на разных платформах и операционных системах.

В более сложных сценариях, например при работе с большими JSON-файлами, стандартного функционала модуля `json` может быть недостаточно. В таких случаях можно использовать альтернативные подходы, такие как потоковая обработка данных или специализированные библиотеки, позволяющие работать с данными частично, без полной загрузки файла в оперативную память. Однако для большинства задач, как и для данной курсовой работы, базовых возможностей файла `json` достаточно.

## 2. Практическая часть

### 2.1. Описание структуры данных

Данные фильмотеки хранятся в JSON-файле, так как он хранит их в виде структур, которые почти идентичны спискам и словарям.



```
1  {
2      "Субстанция": {
3          "год производства": 2024,
4          "страны производства": ["Великобритания", "Франция"],
5          "длительность": "2:21",
6          "режиссер": "Корали Фаржа",
7          "жанры": ["ужасы", "драма"],
8          "рейтинг IMDb": 7.3,
9          "бюджет в $": 17500000,
10         "актеры": ["Деми Мур", "Маргарет Куэлли", "Деннис Куэйд"]
11     },
12     "Виды доброты": {
13         "год производства": 2024,
```

Рисунок 1. Файл data.json со структурой словаря.

В данном файле хранится словарь, где ключами являются строки, обозначающие названия фильмов, а значениями – другие словари. Эти словари в свою очередь имеют строковые ключи, которые обозначают характеристики фильма: год производства, страны производства, длительность, режиссер, жанры, рейтинг IMDb, бюджет в \$, актеры. Для ключей «год производства», «бюджет в \$» значения являются целыми числами, для «рейтинг IMDb» – вещественным числом от 0 до 10, для «режиссёр» – строкой, для «длительность» – строкой вида «чч:мм», где «чч» – часы, «мм» – минуты.

Отдельного внимания заслуживают ключи «страны производства», «жанры» и «актеры», так как их значениями являются списки, состоящие из строк. Таким образом, степень вложенности структуры, хранящейся в файле data.json, равняется трём. Именно такая структура позволяет хранить достаточно полную информацию о фильмах.

В самом начале программы эти данные считываются из файла с помощью использование встроенного модуля `json` и менеджера контекста `with` и сохраняются в переменную `films_dictionary`, которая хранит ту же самую структуру со словарями и списками.



```
main.py x
1  import json
2
3  with open(file="data.json", mode="r", encoding="utf-8") as file_in:
4      films_dictionary = json.load(file_in)
```

*Рисунок 2. Файловый ввод.*

## 2.2. Структура программы

Весь код программы написан в одном файле `main.py`. Программа реализована в соответствии с принципами модульного программирования: весь код разделён на функции, за исключением файлового ввода. Такой подход обеспечивает улучшение читаемости и устранение дублирования кода.



```
main.py x
1  import json
2  with open(file="data.json", mode="r", encoding="utf-8") as file_in:
3      films_dictionary = json.load(file_in)
4  > def request_films_complying_one_condition(par, val, rel):...
29 > def request_films_complying_many_conditions(conds, cond_rel):...
51 > def request_sort_films_by_criteria(par, criteria):...
72 > def get_main_option():...
89 > def get_one_condition():...
172 > def get_many_conditions():...
180 > def get_condition_relation():...
190 > def get_sorting_criteria():...
228 > def main():...
275 main()
```

*Рисунок 3. Функции файла `main.py`.*

Все функции программы можно разделить на три группы: главная функция; функции, реализующие основную логику; функции, реализующие считывание запросов от пользователя и отлов ошибок ввода. Главной функцией является функция `main()`. На рисунке 4, который показывает схему взаимодействия этих групп, видно, что она вызывает функции из остальных групп, то есть является точкой входа в программу.

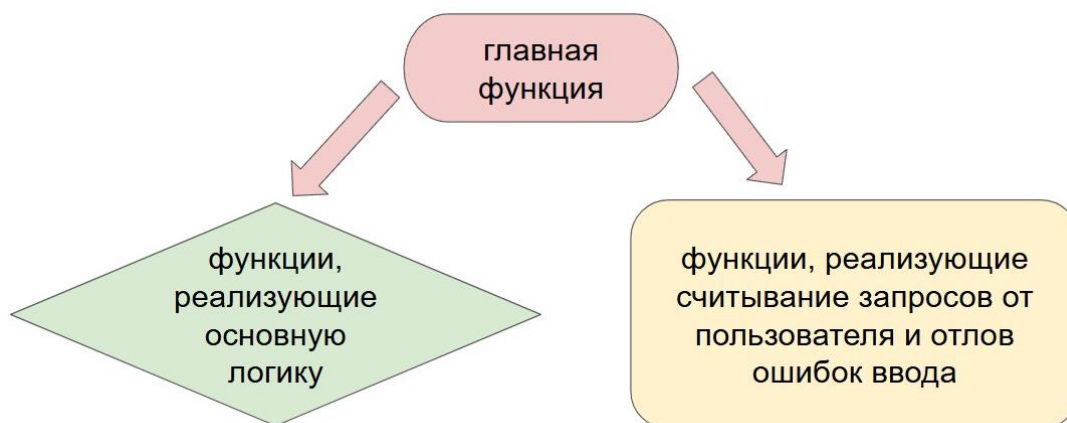


Рисунок 4. Схема взаимодействия групп функций.

Функциями, реализующими основную логику, являются функции `request_films_complying_one_condition()`, `request_films_complying_many_conditions()` и `request_sort_films_by_criteria()`. Первая запрашивает фильмы, удовлетворяющие одному условию. Вторая – удовлетворяющие нескольким условиям. Причём вторая функция опирается на первую. Более подробно они и их взаимодействие будет рассмотрено в главе 2.3. Третья функция запрашивает список фильмов и значений их параметров, отсортированных по заданному критерию.

Функциями, реализующими считывание запросов от пользователя и отлов ошибок ввода являются функции `get_main_option()`, `get_one_condition()`, `get_many_conditions()`, `get_condition_relation()` и `get_sorting_criteria()`. Первая запрашивает пункт главного меню. Вторая – информацию для отбора фильмов по одному условию. Третья – информацию для отбора фильмов по нескольким условиям. Причём третья функция опирается на вторую. Четвёртая функция

запрашивает отношение между условиями (и/или). Пятая – информацию для сортировки фильмов по критерию.

Отлов ошибок ввода нужен для того, чтобы программа продолжала свою работу и сообщала о ошибках в тех случаях, когда пользователь неправильно вводит данные.

```
Начало работы программы "Фильмотека"
Выберите пункт:
1. Вывести фильмы, удовлетворяющие заданным(-ому) условиям(-ю)
2. Вывести фильмы, отсортированные по заданному критерию
3. Вывести информацию по фильму
4. Вывести список всех фильмов в "Фильмотеке"
5. Завершить программу
Введите номер пункта: a
Вы ввели что-то, отличное от целого числа. Повторите ввод.
Введите номер пункта: 10
Вы ввели целое число, отличное от 1, 2, 3, 4 или 5. Повторите ввод.
Введите номер пункта:
```

*Рисунок 5. Пример отлова ошибок ввода.*

На рисунке 5 выше показано, что если пользователь вводит некорректные пункты меню, то программа сообщает ему об этом с указанием конкретной причины: «Вы ввели что-то отличное от целого числа.» или «Вы ввели целое число, отличное от 1, 2, 3, 4 или 5.». После этого программа пишет «Повторите ввод.» и даёт пользователю ещё одну попытку ввода. Программа будет просить повторно ввести информацию до тех пор, пока пользователь не сделает это правильно, либо пока аварийно не завершит программу.

Конечным продуктом является исполняемый файл Cinematheque.exe, который могут запускать обычные пользователи. Он лежит в папке CinemathequeProject, в которой также лежит файл с данными data.json и файл с кодом main.py. Файл с кодом могут использовать программисты, которые владеют средами разработки и могут запускать программу из них. Важно, чтобы файлы Cinematheque.exe и data.json находились в одной папке на одном уровне иерархии, иначе исполняемый файл работать не будет.

## 2.3. Описание программы

Поскольку функции, реализующие считывание запросов от пользователя и отлов ошибок ввода, не содержат идейной составляющей, они не представляют интереса. Поэтому в данной главе будут подробно рассмотрены лишь функции `request_films_complying_one_condition()`, `request_films_complying_many_conditions()` и `request_sort_films_by_criteria()`, которые относятся к группе функций, реализующих основную логику.

### Функции, реализующие основную логику:

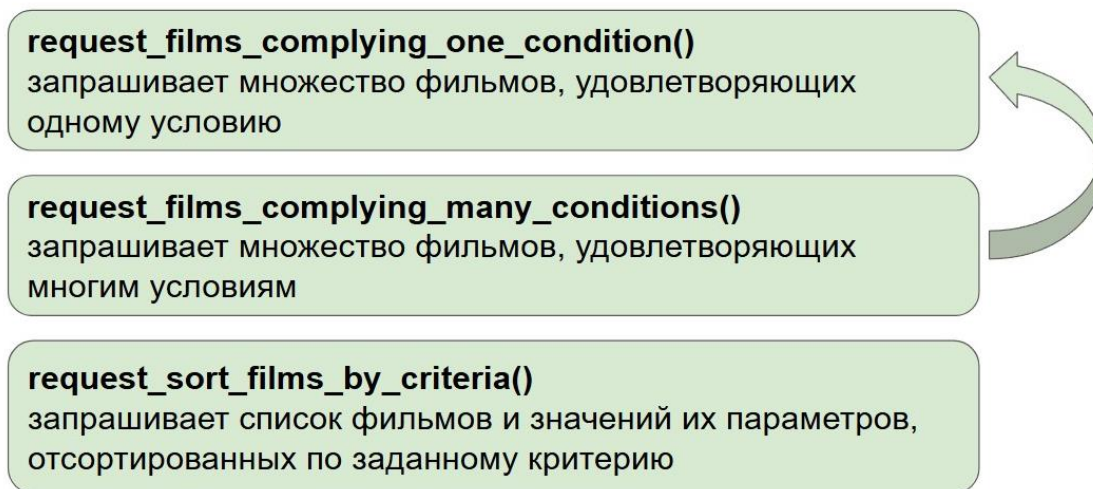


Рисунок 6. Функции, реализующие основную логику.

Функция `request_films_complying_one_condition()` принимает на вход три параметра: `par`, `val` и `rel`. Параметр `par` принимает в качестве значения строку, которая обозначает одну из характеристик: год производства, страны производства, длительность, режиссер, жанры, рейтинг IMDb, бюджет в \$ или актеры. Параметр `val` принимает значение этой характеристики. Параметр `rel` принимает отношение между параметром и значением. Причём если `par` равен «страны производства», «жанры» или «актеры», то `rel` = `None`. Если `par` = «режиссер», то `rel` = «=». В противном случае отношение `rel` принимает одно из значений: «=», «>», «<», «>=», «<=». Таким образом, запрос, формируемый этими тремя параметрами, может выглядеть так: `par` = «год производства», `val` = 2020, `rel` = «>». Тогда по данному запросу будут найдены все фильмы после



2020-го года выпуска. Или же можно составить запрос `par = «актеры»`, `val = «Роберт Паттинсон»`, `rel = None`. Тогда по данному запросу будут найдены все фильмы, в которых снимался Роберт Паттинсон.

```
def request_films_complying_one_condition(par, val, rel):
    if par == "длительность":
        f = lambda x: int(x.split(":")[0]) * 60 + int(x.split(":")[1])
        tmp_dict = {"=": lambda x, y: f(x) == f(y), ">": lambda x, y: f(x) > f(y),
                    "<": lambda x, y: f(x) < f(y), ">=": lambda x, y: f(x) >= f(y),
                    "<=": lambda x, y: f(x) <= f(y)}
    else:
        tmp_dict = {"=": lambda x, y: x == y, ">": lambda x, y: x > y,
                    "<": lambda x, y: x < y, ">=": lambda x, y: x >= y,
                    "<=": lambda x, y: x <= y, None: lambda x, y: y in x}
    check_func = tmp_dict[rel]

    films = set()
    for film_name, film_info in films_dictionary.items():
        if check_func(film_info[par], val):
            films.add(film_name)
    return films
```

Рисунок 7. Код функции `request_films_complying_one_condition()`.

Далее создаётся словарь `tmp_dict`, который содержит ключи, которые являются всеми возможными значениями отношения `rel`, и значения, которые являются лямбда-функциями, проверяющими на соответствие `par` и `val`. Причём если `par = «длительность»`, то лямбда-функции используют другую лямбда-функцию, привязанную к переменной `f`. Она преобразует длительность фильма из строкового формата «чч:мм» в целое число минут. После создания `tmp_dict` определяется функция `check_func()`, которая из всех лямбда-функций выбирает ту, которая соответствует отношению `rel`. Это происходит с помощью обращения к словарию `tmp_dict` по ключу `rel`. После этого создаётся пустое множество `films`, которое будет наполняться подходящими фильмами. Во время перебора `films_dictionary` в множество добавляются те фильмы, для которых функция `check_func` вернула `True`. В конце наполненное множество возвращается с помощью оператора `return`.

Функция `request_films_complying_many_conditions()` принимает на вход два параметра: `conds` и `cond_rel`. Параметр `conds` представляет собой список кортежей вида `[(par1, val1, rel1), (par2, val2, rel2), ...]`. Параметр `cond_rel` представляет собой отношение между перечисленными условиями: «И» - если должны выполняться все условия, «ИЛИ» - хотя бы одно.

```
def request_films_complying_many_conditions(conds, cond_rel):
    if cond_rel == "И":
        func = lambda x, y: x.intersection(y)
    else:
        func = lambda x, y: x.union(y)
    films = set()
    for i in range(len(conds)):
        par, val, rel = conds[i]
        films_i = request_films_complying_one_condition(par, val, rel)
        if i == 0:
            films = films_i
        else:
            films = func(films, films_i)
    return films
```

Рисунок 8. Код функции `request_films_complying_many_conditions()`.

Функция `request_films_complying_many_conditions()` опирается на функцию `request_films_complying_one_condition()`. Так как последняя возвращает множество фильмов, удовлетворяющих одному запросу, то первая вызывает её, чтобы пересекать эти множества в случае `cond_rel = «И»` или объединять в случае `cond_rel = «ИЛИ»`. Далее определяется лямбда-функция, привязанная к переменной `func`, которая будет применять метод пересечения или объединения к множествам. После этого создаётся пустое множество `films`, которое будет наполняться подходящими фильмами. Далее перебираются условия из `conds`, для каждого ищется множество фильмов `films_i`, удовлетворяющее этому условию с помощью вызова функции `request_films_complying_one_condition()`. И к множествам `films` и `films_i` на каждой итерации цикла, кроме самой первой, применяется функция `func()`. В конце функции возвращается итоговое множество `films`.



Функция `request_sort_films_by_criteria()` принимает на вход два параметра: `par` и `criteria`. Параметр `par` принимает в качестве значения одну из следующих характеристик: «год производства», «длительность», «рейтинг IMDb», «бюджет в \$». Параметр `criteria` в качестве значения принимает либо «increase», либо «decrease». Функция будет сортировать все фильмы по указанному параметру в порядке возрастания его значений, если `criteria = «increase»`, или в порядке убывания его значений, если `criteria = «decrease»`.

```
def request_sort_films_by_criteria(par, criteria):
    films_data = [(film_name, film_info[par])
                   for film_name, film_info in films_dictionary.items()]
    if criteria == "increase":
        reverse_par = False
    else:
        reverse_par = True
    if par == "длительность":
        cmp = lambda x: int(x[1].split(":")[0]) * 60 + int(x[1].split(":")[1])
    else:
        cmp = lambda x: x[1]
    sorted_films_data = sorted(films_data, key=cmp, reverse=reverse_par)
    return sorted_films_data
```

*Рисунок 9. Код функции `request_sort_films_by_criteria()`.*

Далее формируется список `films_data`, который состоит из кортежей пар вида («название фильма», значение указанного параметра). После этого настраивается значение параметра, отвечающего за порядок сортировки. После определяется функция `cmp`, которая будет использоваться в качестве ключа сортировки. Если `par = «длительность»`, то время из формата «чч:мм» преобразуется в целое количество минут в `cmp`, чтобы можно было корректно сравнивать. В конце формируется отсортированный список пар с помощью применения функции `sorted` и указания значений её параметров `key` и `reverse`. В конце эта структура данных возвращается с помощью оператора `return`.

## Заключение

В ходе выполнения курсовой работы была достигнута поставленная цель – разработана программа на Python, использующая структуру данных «словарь» для управления фильмотекой. Она позволяет загружать данные из JSON-файла, обрабатывать их и выполнять запросы пользователя, такие как поиск фильмов по заданным условиям, сортировка по критериям и вывод информации о фильмах.

Основные задачи выполнены. Изучены теоретические аспекты работы со словарями в Python, включая их структуру, методы и особенности. Реализована программа, которая эффективно использует словари для хранения и обработки данных о фильмах. Создан исполняемый файл, доступный для использования обычными пользователями.

Программа демонстрирует высокую эффективность благодаря использованию встроенных возможностей Python, таких как модуль `json`, лямбда-функции и методы сортировки. Особое внимание уделено обработке ошибок ввода, что делает программу удобной и устойчивой к неверным действиям пользователя. Благодаря модульной структуре кода, реализованные функции легко модифицировать и расширять, что делает проект гибким и адаптируемым под будущие изменения, а также показывает высокую культуру разработки кода.

Разработанное приложение может пригодиться не только в учебных целях, но и в реальной практике – например, для создания личной коллекции фильмов или работы с киноархивами. В будущем его можно доработать, добавив интеграцию с онлайн базами данных о фильмах, реализовав дополнительные возможности анализа данных и создав графический интерфейс приложения с помощью библиотеки `customtkinter`. Полученный опыт работы со структурами данных обязательно пригодится в дальнейшей учёбе и профессиональной деятельности.

## Список литературы

1. Бейдер, Д. Чистый Python. Тонкости программирования для профи / Д. Бейдер. – СПб.: Питер, 2018. – 288 с. – ISBN 978-5-4461-0803-9.
2. Всё о сортировке в Python: исчерпывающий гайд // Tproger. – 2023. [электронный ресурс] – URL: <https://tproger.ru/translations/python-sorting> (дата обращения: 21.03.2025)
3. Златопольский, Д. М. Основы программирования на языке Python / Д. М. Златопольский. – М.: ДМК Пресс, 2018. – 396 с. – ISBN 978-5-97060-552-3.
4. Кормен, Т. Алгоритмы. Построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест. – 3-е изд. – М.: Вильямс, 2013. – 1328 с. – ISBN 978-5-8459-1794-2.
5. Лутц, М. Изучаем Python, том 1 / М. Лутц. – 5-е изд. – СПб.: ООО «Диалектика», 2019. – 832 с. – ISBN 978-5-907144-52-1.
6. Маккини У. Python и анализ данных / У. Маккини. – М.: ДМК Пресс, 2022. – 482 с. – ISBN 978-5-94074-590-5.
7. Множества, словари // Яндекс Образование. – 2023. [электронный ресурс] – URL: <https://education.yandex.ru/handbook/python/article/mnozhestva-slovari> (дата обращения: 21.03.2025)
8. Официальная документация Python. Работа с JSON // Python Docs. – 2025. [электронный ресурс] – URL: <https://docs.python.org/3.12/library/json.html> (дата обращения: 15.03.2025)
9. Словари в Python: как они работают, зачем нужны и как использовать их эффективно // Skillfactory media. – 2025. [электронный ресурс] – URL: <https://blog.skillfactory.ru/slovari-v-python/> (дата обращения: 01.05.2025)
10. Структуры данных // Py3dev. – 2024. [электронный ресурс] – URL: [https://py3dev.ru/bop/data\\_structures/](https://py3dev.ru/bop/data_structures/) (дата обращения: 20.03.2025)

## Приложение

Файл data.json:

```
{
  "Субстанция": {
    "год производства": 2024,
    "страны производства": ["Великобритания", "Франция"],
    "длительность": "2:21",
    "режиссер": "Корали Фаржа",
    "жанры": ["ужасы", "драма"],
    "рейтинг IMDb": 7.3,
    "бюджет в $": 17500000,
    "актеры": ["Деми Мур", "Маргарет Куэлли", "Деннис Куэйд"]
  },
  "Виды доброты": {
    "год производства": 2024,
    "страны производства": ["Ирландия", "Великобритания", "США", "Греция"],
    "длительность": "2:44",
    "режиссер": "Йоргос Лантимос",
    "жанры": ["драма", "комедия"],
    "рейтинг IMDb": 6.5,
    "бюджет в $": 15000000,
    "актеры": ["Эмма Стоун", "Маргарет Куэлли", "Уиллем Дефо"]
  },
  "Бедные-несчастные": {
    "год производства": 2023,
    "страны производства": ["Ирландия", "Великобритания", "США", "Венгрия"],
    "длительность": "2:21",
    "режиссер": "Йоргос Лантимос",
    "жанры": ["драма", "комедия", "фантастика"],
    "рейтинг IMDb": 7.8,
    "бюджет в $": 35000000,
    "актеры": ["Эмма Стоун", "Уиллем Дефо", "Марк Руффало", "Рами Юссеф", "Маргарет Куэлли"]
  },
  "Круэлла": {
    "год производства": 2021,
    "страны производства": ["США", "Великобритания", "Канада", "Ирландия"],
    "длительность": "2:14",
    "режиссер": "Крейг Гиллепси",
    "жанры": ["комедия", "криминал", "драма"],
    "рейтинг IMDb": 7.3,
    "бюджет в $": 200000000,
    "актеры": ["Эмма Стоун", "Эмма Томпсон", "Джоэль Фрай", "Пол Уолтер Хаузер"]
  },
  "Ла-Ла Ленд": {
    "год производства": 2016,
    "страны производства": ["США", "Гонконг"],
    "длительность": "2:08",
    "режиссер": "Дэмыен Шазелл",
    "жанры": ["мюзикл", "драма", "мелодрама", "комедия"],
    "рейтинг IMDb": 8.0,
    "бюджет в $": 30000000,
    "актеры": ["Райан Гослинг", "Эмма Стоун", "Джон Ледженд", "Дж.К. Симмонс"]
  }
}
```

```

},
"Маяк": {
  "год производства": 2019,
  "страны производства": ["США", "Канада"],
  "длительность": "1:49",
  "режиссер": "Роберт Эггерс",
  "жанры": ["ужасы", "фэнтези", "драма"],
  "рейтинг IMDb": 7.4,
  "бюджет в $": 11000000,
  "актеры": ["Роберт Паттинсон", "Уиллем Дефо", "Валерия Караман"]
},
"Бегущий по лезвию 2049": {
  "год производства": 2017,
  "страны производства": ["США", "Канада", "Испания"],
  "длительность": "2:44",
  "режиссер": "Дени Вильнёв",
  "жанры": ["фантастика", "боевик", "триллер", "драма"],
  "рейтинг IMDb": 8.0,
  "бюджет в $": 150000000,
  "актеры": ["Райан Гослинг", "Харрисон Форд", "Ана де Армас", "Сильвия Хукс", "Джаред Лето"]
},
"Высшее общество": {
  "год производства": 2018,
  "страны производства": ["Великобритания", "Франция", "Германия", "Польша"],
  "длительность": "1:53",
  "режиссер": "Клер Дени",
  "жанры": ["фантастика", "триллер"],
  "рейтинг IMDb": 5.7,
  "бюджет в $": 7000000,
  "актеры": ["Роберт Паттинсон", "Жюльет Бинош", "Миа Гот", "Агата Бузек"]
},
"Микки 17": {
  "год производства": 2025,
  "страны производства": ["США"],
  "длительность": "2:19",
  "режиссер": "Пон Джун-хо",
  "жанры": ["фантастика", "комедия", "приключения"],
  "рейтинг IMDb": 7.1,
  "бюджет в $": 150000000,
  "актеры": ["Роберт Паттинсон", "Наоми Аки", "Тони Коллетт", "Стивен Ян", "Марк Руффало"]
},
"Солнцестояние": {
  "год производства": 2019,
  "страны производства": ["США", "Швеция"],
  "длительность": "2:28",
  "режиссер": "Ари Астер",
  "жанры": ["ужасы", "драма", "триллер"],
  "рейтинг IMDb": 7.1,
  "бюджет в $": 9000000,
  "актеры": ["Флоренс Пью", "Джек Рейнор", "Вильгельм Бломгрэн", "Уилл Поултер", "Уильям Джексон Харпер"]
}
}

```

## Файл main.py:

```
import json
```

```
with open(file="data.json", mode="r", encoding="utf-8") as file_in:  
    films_dictionary = json.load(file_in)
```

```
def request_films_complying_one_condition(par, val, rel):
```

```
    """
```

```
    Запрашивает фильмы, удовлетворяющие одному условию
```

```
    :param par: параметр ("год производства"/"страны производства"/.../"актеры")
```

```
    :param val: значение параметра
```

```
    :param rel: отношение между параметром и значением: "="/>"/<"/>="/<="
```

```
    :return: мн-во фильмов films
```

```
    Пример запроса №1: par="год производства", val=2020, rel=">" - фильмы после 2020 года  
    выпуска
```

```
    Пример запроса №2: par="актеры", val="Роберт Паттинсон", rel=None - фильмы, в которых  
    снимался Роберт Паттинсон
```

```
    """
```

```
    if par == "длительность":
```

```
        f = lambda x: int(x.split(":")[0]) * 60 + int(x.split(":")[1]) # преобразует длительность в str в минуты  
        в int
```

```
        tmp_dict = {"=": lambda x, y: f(x) == f(y), ">": lambda x, y: f(x) > f(y), "<": lambda x, y: f(x) < f(y),  
                    ">=": lambda x, y: f(x) >= f(y), "<=": lambda x, y: f(x) <= f(y)}
```

```
    else:
```

```
        tmp_dict = {"=": lambda x, y: x == y, ">": lambda x, y: x > y, "<": lambda x, y: x < y,  
                    ">=": lambda x, y: x >= y, "<=": lambda x, y: x <= y, None: lambda x, y: y in x}
```

```
    check_func = tmp_dict[rel]
```

```
    films = set()
```

```
    for film_name, film_info in films_dictionary.items():
```

```
        if check_func(film_info[par], val):
```

```
            films.add(film_name)
```

```
    return films
```

```
def request_films_complying_many_conditions(conds, cond_rel):
```

```
    """
```

```
    Запрашивает фильмы, удовлетворяющие многим условиям
```

```
    :param conds: список условий [(par1, val1, rel1), (par2, val2, rel2), ...]
```

```
    :param cond_rel: отношение между условиями: "И" - должны выполняться все условия, "ИЛИ" -  
    хотя бы одно
```

```
    :return: мн-во фильмов films
```

```
    """
```

```
    if cond_rel == "И":
```

```
        func = lambda x, y: x.intersection(y)
```

```
    else:
```

```
        func = lambda x, y: x.union(y)
```

```
    films = set()
```

```
    for i in range(len(conds)):
```

```

    par, val, rel = conds[i]
    films_i = request_films_complying_one_condition(par, val, rel)
    if i == 0:
        films = films_i
    else:
        films = func(films, films_i)
return films

```

```

def request_sort_films_by_criteria(par, criteria):

```

```

    """
    Запрашивает список фильмов и значения их параметров, отсортированных по заданному
    критерию

```

```

    :param par: параметр ("год производства"/"длительность"/"рейтинг IMDb"/"бюджет в $")

```

```

    :param criteria: критерий: "increase" - возрастание, "decrease" - убывание

```

```

    :return: sorted_films_data

```

```

    """

```

```

    films_data = [(film_name, film_info[par]) for film_name, film_info in films_dictionary.items()]

```

```

    if criteria == "increase":

```

```

        reverse_par = False

```

```

    else:

```

```

        reverse_par = True

```

```

    if par == "длительность":

```

```

        cmp = lambda x: int(x[1].split(":")[0]) * 60 + int(x[1].split(":")[1])

```

```

    else:

```

```

        cmp = lambda x: x[1]

```

```

    sorted_films_data = sorted(films_data, key=cmp, reverse=reverse_par)

```

```

    return sorted_films_data

```

```

def get_main_option():

```

```

    print("Выберите пункт:")

```

```

    print("1. Вывести фильмы, удовлетворяющие заданным(-ому) условиям(-ю)")

```

```

    print("2. Вывести фильмы, отсортированные по заданному критерию")

```

```

    print("3. Вывести информацию по фильму")

```

```

    print("4. Вывести список всех фильмов в \"Фильмотеке\"")

```

```

    print("5. Завершить программу")

```

```

    while True:

```

```

        try:

```

```

            main_option = int(input("Введите номер пункта: "))

```

```

        except ValueError:

```

```

            print("Вы ввели что-то, отличное от целого числа. Повторите ввод.")

```

```

        else:

```

```

            if main_option not in (1, 2, 3, 4, 5):

```

```

                print("Вы ввели целое число, отличное от 1, 2, 3, 4 или 5. Повторите ввод.")

```

```

            else:

```

```

                return main_option

```

```

def get_one_condition():

```

```

    print("Выберите параметр, для которого будет задано условие:")

```

```

print("1. год производства")
print("2. страны производства")
print("3. длительность")
print("4. режиссер")
print("5. жанры")
print("6. рейтинг IMDb")
print("7. бюджет в $")
print("8. актеры")
while True:
    try:
        parameter_number = int(input("Введите номер параметра: "))
    except ValueError:
        print("Вы ввели что-то, отличное от целого числа. Повторите ввод.")
    else:
        if parameter_number not in (1, 2, 3, 4, 5, 6, 7, 8):
            print("Вы ввели целое число, отличное от 1, 2, 3, 4, 5, 6, 7 или 8. Повторите ввод.")
        else:
            break
    tmp_dict = {1: "год производства", 2: "страны производства", 3: "длительность", 4: "режиссер", 5:
"жанры",
        6: "рейтинг IMDb", 7: "бюджет в $", 8: "актеры"}
    parameter = tmp_dict[parameter_number]
    if parameter in ("страны производства", "жанры", "актеры"):
        if parameter == "страны производства":
            print("По данному параметру будут отобраны фильмы, одно из мест съемок которых было в
указанной стране.")
            value = input("Введите название страны: ")
        elif parameter == "жанры":
            print("По данному параметру будут отобраны фильмы, которые относятся к указанному
жанру.")
            value = input("Введите название жанра: ")
        else:
            print("По данному параметру будут отобраны фильмы, в которых снимался указанный
актер.")
            value = input("Введите имя актера: ")
            relation = None
    elif parameter == "режиссер":
        value = input("Введите имя режиссёра: ")
        relation = "="
    else:
        print("После того, как вы введёте значение выбранного параметра,",
            "будет установлено отношение между ними (=/>/</...)")
        if parameter == "длительность":
            while True:
                value = input("Введите длительность в формате [часы]:[минуты] (без квадратных скобок):
")
                if value.count(":") != 1 or (not value.split(":")[0].isdigit()) or (not value.split(":")[1].isdigit()):
                    print("Вы нарушили формат ввода. Повторите ввод.")
                    continue
                if int(value.split(":")[1]) >= 60:
                    print("Минут должно быть меньше 60. Повторите ввод.")
                    continue
                break
            elif parameter in ("год производства", "бюджет в $"):

```



```

while True:
    try:
        value = int(input(f"Введите {parameter}: "))
    except ValueError:
        print("Вы ввели что-то, отличное от целого числа. Попробуйте ещё раз.")
    else:
        if parameter == "год производства" and (value > 2025 or value < 1895):
            print("Вы ввели некорректный год. Повторите ввод.")
            continue
        if parameter == "бюджет в $" and value <= 0:
            print("Вы ввели некорректный бюджет. Повторите ввод.")
            continue
        break
elif parameter == "рейтинг IMDb":
    while True:
        try:
            value = float(input("Введите рейтинг IMDb: "))
        except ValueError:
            print("Вы ввели что-то отличное от вещественного или целого числа, или ввели запятую.",
                  "Повторите ввод.")
        else:
            if not (0 <= value <= 10):
                print("Рейтинг не может выходить за рамки диапазона [0; 10]. Повторите ввод.")
                continue
            break

while True:
    relation = input("Введите отношение между параметром и значением: =/>/</>=<= (без символа слеша): ")
    if relation in ("=", ">", "<", ">=", "<="):
        break
    print("Вы ввели что-то отличное от =/>/</>=<=. Повторите ввод.")
return parameter, value, relation

def get_many_conditions():
    conditions = []
    while True:
        conditions.append(get_one_condition())
        answer = input("Хотите ввести ещё одно условие? +/-: ")
        if answer != "+":
            break
    return conditions

def get_condition_relation():
    print("Вы ввели несколько условий. Введите отношение между ними:",
          "И - найти фильмы, удовлетворяющие всем условиям",
          "ИЛИ - найти фильмы, удовлетворяющие хотя бы одному условию", sep="\n")
    while True:
        cond_rel = input("Отношение И/ИЛИ: ").lower()
        if cond_rel in ("и", "или"):
            break

```

```

    print("Некорректный ввод. Повторите ввод.")
    return cond_rel.upper()

```

```

def get_sorting_criteria():
    print("Выберите параметр, по которому будет выполнена сортировка:")
    print("1. год производства")
    print("2. длительность")
    print("3. рейтинг IMDb")
    print("4. бюджет в $")
    while True:
        try:
            parameter_number = int(input("Введите номер параметра: "))
        except ValueError:
            print("Вы ввели что-то, отличное от целого числа. Повторите ввод.")
        else:
            if parameter_number not in (1, 2, 3, 4):
                print("Вы ввели целое число, отличное от 1, 2, 3 или 4. Повторите ввод.")
            else:
                break
    tmp_dict = {1: "год производства", 2: "длительность", 3: "рейтинг IMDb", 4: "бюджет в $"}
    parameter = tmp_dict[parameter_number]

    print("Выберите критерий сортировки:")
    print("1. по возрастанию")
    print("2. по убыванию")
    while True:
        try:
            criteria_number = int(input("Выберите номер критерия: "))
        except ValueError:
            print("Вы ввели что-то, отличное от целого числа. Повторите ввод.")
        else:
            if criteria_number not in (1, 2):
                print("Вы ввели целое число, отличное от 1 или 2. Повторите ввод.")
            else:
                break
    if criteria_number == 1:
        criteria = "increase"
    else:
        criteria = "decrease"

    return parameter, criteria

```

```

def main():
    print("Начало работы программы \"Фильмотека\"")
    while True:
        main_option = get_main_option()
        if main_option == 1:
            conditions = get_many_conditions()
            if len(conditions) == 1:
                cond = conditions[0]
                par, val, rel = cond
                films = request_films_complying_one_condition(par, val, rel)

```

```

else:
    cond_rel = get_condition_relation()
    films = request_films_complying_many_conditions(conditions, cond_rel)
if len(films) > 0:
    films = sorted(films)
    for i, f in enumerate(films, start=1):
        print(f"{i}. {f}")
else:
    print("По вашему запросу ничего не найдено.")
elif main_option == 2:
    parameter, criteria = get_sorting_criteria()
    sorted_films_data = request_sort_films_by_criteria(parameter, criteria)
    for i, film_data in enumerate(sorted_films_data, start=1):
        print(f"{i}. {film_data[0]} - {film_data[1]}")
elif main_option == 3:
    film_name = input("Введите название фильма: ").lower()
    tmp_dict = {f_n.lower(): f_n for f_n in films_dictionary.keys()}
    if film_name not in tmp_dict:
        print("Такого фильма нет в \"Фильмотеке\"")
    else:
        correct_film_name = tmp_dict[film_name]
        print(f"{correct_film_name}:")
        for par, val in films_dictionary[correct_film_name].items():
            print(f"{par} - ", end="")
            if type(val) == list:
                print(*val, sep=" ")
            else:
                print(val)
elif main_option == 4:
    for i, film_name in enumerate(sorted(films_dictionary), start=1):
        print(f"{i}. {film_name}")
else:
    break
answer = input("Желаете продолжить? +/-: ")
if answer != "+":
    break
print("Завершение работы программы \"Фильмотека\"")

```

main()