ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ НЕТИПОВОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«САНКТ-ПЕТЕРБУРГСКИЙ ГОРОДСКОЙ ДВОРЕЦ ТВОРЧЕСТВА ЮНЫХ»

АНИЧКОВ ЛИЦЕЙ



**Индивидуальный проект**

**на тему**

**«Разработка программного обеспечения для обработки базы данных аптеки»**

Обучающегося 10 Б класса

Калинина Александра Георгиевича

Научный руководитель:

учитель информатики Иван Михайлович Колосов

Санкт-Петербург

2023

**Содержание**

[Введение 3](#_Toc136028797)

[1. Теоретические основы разработки программного обеспечения для обработки информации из базы данных 4](#_Toc136028798)

[1.1 Основы работы с файлами в Python 4](#_Toc136028799)

[1.2 Открытие и закрытие файла 5](#_Toc136028800)

[1.3 Чтение и запись файлов в Python 7](#_Toc136028801)

[1.4 Подпрограммы: процедуры и функции в Python 10](#_Toc136028802)

[2. Практическая реализация алгоритмов в виде программного кода 15](#_Toc136028803)

[Заключение 18](#_Toc136028804)

[Список литературы 19](#_Toc136028805)

# Введение

Актуальность выбранной мною темы обусловлена повсеместной информатизацией всех сфер жизнедеятельности человека и внедрением современных компьютерных технологий в нашу повседневную жизнь.

В своём индивидуальном проекте я поставил перед собой цель разобраться в процессах автоматизации обработки разнородной информации на примере аптеки и научиться разрабатывать программное обеспечение для работы с базами данных. Для достижения поставленной цели необходимо было решить несколько концептуальных задач.

1. Прежде всего разобраться в новой для меня предметной области, так как ни к фармацевтике, ни к аптечным сетям ранее я никакого отношения не имел
2. Изучение основных принципов ведения электронного документооборота в аптечной сети
3. Разбор технического задания по разработке программного обеспечения и планирование основных этапов реализации. Исследование структуры файла формата «SST» (Semicolon Separated Text) и способов обработки массива данных из него.
4. Непосредственная реализация задуманных алгоритмов для обработки информации из базы данных в виде программного кода

На мой взгляд, значимость проделанной работы заключается в неопровержимой пользе от изучения нового материала по обработке информации из баз данных и разработке программного обеспечения, получении новых и актуальных в наше время навыков в сфере написания программ. Отдельно стоит отметить практическую ценной разработанного проекта, которая заключается в реальной возможности эксплуатации созданного программного обеспечения и его дальнейшей доработки под изменяющиеся нужды пользователя.

# 1. Теоретические основы разработки программного обеспечения для обработки информации из базы данных

## 1.1 Основы работы с файлами в Python

Взаимодействие с файловой системой позволяет хранить информацию, полученную в результате работы программы. Большие объемы данных хранят не в списках или словарях, а в файлах и базах данных. На самом деле файл — это просто набор данных, сохраненный в виде последовательности битов на компьютере. Информация хранится в структуре данных и имеет название (имя файла). В высокоуровневом языке программирования Python предоставляются встроенные функции для создания, записи и чтения файлов двух типов файлов:

1. Текстовые
2. Бинарные

**Текстовые файлы**

Предполагается, что этот вид файлов с человекочитаемым содержимым, то есть в нём набор символов, который может прочитать человек. Как правило, такие файлы можно открыть для чтения или измениния любым текстовым редактором, например, блокнотом. Чаще всего текст хранится формате «.txt»

В файлах этого типа каждая строка текста заканчивается специальным символом, называемым EOL (end of line - конец строки), который по умолчанию является символом новой строки (‘\n’) в Python и многих других языках программирования.

**Бинарные файлы (двоичные)**

В файлах этого типа нет разделителя для строки, и данные сохраняются после преобразования их в понятный машине двоичный язык, в закодированной форме с использованием только нулей (0) и единиц (1) вместо простых символов. По большому счёту двоичный файл – это просто последовательность битов. Таким способом можно хранить любую информацию: изображения, аудио, видео и тем более текст. На информационных накопителях такие файлы хранятся в формате с расширением «bin».

При работе с файлами необходимо соблюдать три последовательных этапа:

* Открытие файла
* Выполнение операции (запись, чтение)
* Закрытие файла

В Python есть несколько способов прочитать текстовый файл. Для этого обычно используются функция open(), методы read(), readline(), readlines(), close() и ключевое слово with.

## 1.2 Открытие и закрытие файла

Чтобы начать работу с файлом, его надо открыть с помощью built-in функции open(), которая имеет следующее формальное определение: **open(file, mode)**. Эта функция имеет два аргумента. Первый аргумент функции представляет из себя строку, в которой указан путь к файлу. Путь файла может быть абсолютным, то есть начинаться с буквы диска, например, C://folder/file.txt. Либо можно быть относительным, например, folder/file.txt - в этом случае поиск файла будет идти относительно расположения запущенного скрипта Python. Второй аргумент, **mode**, устанавливает режим открытия файла в зависимости от того, что мы собираемся с ним делать. По умолчанию этот аргумент принимает значение «rt», с которым, и с некоторыми другими, можно ознакомиться в таблице 1. Если при вызове **open**() второй аргумент не указан, то файл открывается в режиме чтения как текстовый файл. Чтобы открыть файл как байтовый, дополнительно к букве режима чтения/записи добавляется символ 'b'. Буква 't' обозначает текстовый файл. Поскольку это тип файла по умолчанию, то обычно ее не указывают. Функция **open**() возвращает объект файлового типа. Его надо либо сразу связать с переменной, чтобы не потерять, либо сразу прочитать.

При открытии файла в текстовом режиме, также можно указать подходящую кодировку. Например, если в вашем файле содержится текст на русском в utf8, откройте его в этой кодировке:

russian\_file = open("russian.txt", "r", encoding="utf8")

Эти режимы могут быть скомбинированы. Например, «rb» открывает двоичный файл для чтения. Комбинируя «r+» или «w+» можно добиться открытия файла в режиме и чтения, и записи одновременно с одним отличием — первый режим вызовет исключение, если файла не существует, а работа во втором режиме в таком случае создаст его.

**Таблица 1. Список режимов открытия файла**

|  |  |
| --- | --- |
| **Режим** | **Обозначение** |
| 'r' | открытие на чтение (является значением по умолчанию). |
| 'w' | открытие на запись, содержимое файла удаляется, если файла не существует, создается новый. |
| 'x' | открытие на запись, если файла не существует, иначе исключение. |
| 'a' | открытие на дозапись, информация добавляется в конец файла. |
| 'b' | открытие в двоичном режиме. |
| 't' | открытие в текстовом режиме (является значением по умолчанию). |
| '+' | открытие на чтение и запись |

После открытия файла в Python его нужно закрыть. Таким образом освобождаются ресурсы и убирается мусор. Python автоматически закрывает файл, когда объект присваивается другому файлу. Существуют несколько способов реализации этого. После того как работа с файлом закончена, важно не забывать его закрыть, чтобы освободить место в памяти. Делается это с помощью файлового метода **close**(). Свойство файлового объекта closed позволяет проверить закрыт ли файл. После закрытия этот файл нельзя будет использовать до тех пор, пока заново его не открыть. Также можно написать try/finally, которое гарантирует, что если после открытия файла операции с ним приводят к исключениям, он закроется автоматически. Без него программа завершается некорректно. Этот метод гарантирует, что если операции над файлом вызовут исключения, то он закроется до того как программа остановится. Ну и наконец ещё один подход — использовать инструкцию with, которая упрощает обработку исключений с помощью инкапсуляции начальных операций, а также задач по закрытию и очистке. В таком случае инструкция close не нужна, потому что with автоматически закроет файл.

## 1.3 Чтение и запись файлов в Python

По умолчанию файл открывается в режиме rt - для чтения в текстовом формате. Стоит заметить, что файл можно открыть в двух разных форматах: текстовом и бинарном (двоичном). Файлы, открытые в текстовом формате (по умолчанию, или явно добавляя “t” к аргументу режима), обрабатываются Python-ом и возвращаются как строки. При открытии файла в бинарном формате никакой обработки содержимого не производится, содержимое возвращается побайтово. Обычно, файлы, в которых содержится текст, например, файлы txt, код вашей программы, файлы формата csv, открываются в текстовом формате, а файлы, которые нельзя проинтерпретировать как текст - в бинарном (например, картинки, музыку). Иногда файлы с текстом открывают в бинарном режиме, для более явного управления всеми спецсимволами (например табуляция ↹).

Обычно используются режимы чтения ('r') и записи ('w'). Если файл открыт в режиме чтения, то запись в него невозможна. Можно только считывать данные. Если файл открыт в режиме записи, то в него можно только записывать данные, считывать нельзя.

Если файл открывается в режиме 'w', то все данные, которые в нем были до этого, стираются. Файл становится пустым. Если не надо удалять существующие в файле данные, тогда следует использовать вместо режима записи, режим дозаписи ('a').

Если файл отсутствует, то открытие его в режиме 'w' создаст новый файл. Бывают ситуации, когда надо гарантировано создать новый файл, избежав случайной перезаписи данных существующего. В этом случае вместо режима 'w' используется режим 'x'. В нем всегда создается новый файл для записи. Если указано имя существующего файла, то будет выброшено исключение. Потери данных в уже имеющемся файле не произойдет.

В Python файлы можно читать или записывать информацию в них с помощью соответствующих режимов.

Функция **read**() используется для чтения содержимого файла после открытия его в режиме чтения (r).

Синтаксис file.read(size) где, file = объект файла, а size = количество символов, которые нужно прочитать. Если не указать, то файл прочитается целиком.

Функция **readline**() используется для построчного чтения содержимого файла. Она используется для крупных файлов. С ее помощью можно получать доступ к любой строке в любой момент.

Метод **readlines**() считывает сразу все строки и создает из них список.

Функция **write**() используется для записи в файлы Python, открытые в режиме записи. Синтаксис file.write(string). Если пытаться открыть файл, которого не существует, в этом режиме, тогда будет создан новый.

Объект файлового типа относится к итераторам. Из таких объектов происходит последовательное извлечение элементов. Элементами в данном случае являются строки-линии файла. Поэтому считывать данные из файла можно сразу в цикле без использования методов чтения. Запись в файл выполняется с помощью методов write() и writelines(). Во второй можно передать структуру данных. Метод write() возвращает количество записанных символов.

Конструкция **with** используется для того, чтобы гаранировать, что критические действия будут выполнены в любом случае, ее можно использовать и в некоторых других случаях, но в контексте открытия файлов она используется чаще всего. Я бы рекомендовал по возможности всегда открывать файлы, не зависимо от режима, с конструкцией with. Через конструкцию with можно открывать сразу несколько файлов. Чтобы считать из файла целую строку, используется метод readline(max\_len). Если указать параметр max\_len, то будут считаны максимум max\_len символов

**Таблица 2. Методы работы с файлами в Python**

|  |  |
| --- | --- |
| file.close() | закрывает открытый файл |
| file.fileno() | возвращает целочисленный дескриптор файла |
| file.flush() | очищает внутренний буфер |
| file.isatty() | возвращает True, если файл привязан к терминалу |
| file.next() | возвращает следующую строку файла |
| file.read(n) | чтение первых n символов файла |
| file.readline() | читает одну строчку строки или файла |
| file.readlines() | читает и возвращает список всех строк в файле |
| file.seek(offset[,whene]) | устанавливает текущую позицию в файле |
| file.seekable() | проверяет, поддерживает ли файл случайный доступ. Возвращает True, если да |
| file.tell() | возвращает текущую позицию в файле |
| file.truncate(n) | уменьшает размер файл. Если n указала, то файл обрезается до n байт, если нет — до текущей позиции |
| file.write(str) | добавляет строку str в файл |
| file.writelines(sequence) | добавляет последовательность строк в фай |

## 1.4 Подпрограммы: процедуры и функции в Python

**Подпрограмма** - это отдельная часть программы, имеющая имя и решающая свою отдельную задачу. Располагается подпрограмма в начале основной программы и может быть запущена (вызвана) из основной программы по указанию имени.

Подпрограммы:

* избавляют от необходимости многократно повторять в тексте программы

аналогичные фрагменты;

* улучшают структуру программы, облегчая ее понимание;
* позволяют распределять большие задачи между несколькими

разработчиками или стадиями проекта;

* повышают устойчивость к ошибкам программирования и непредвидимым

последствиям при модификациях программы.

Подпрограммы прежде всего необходимы в ситуации, когда в разных частях программы необходимо выполнять одни и те же действия несколько раз. В таком случае повторяемые операторы оформляются в виде функции или процедуры, к которой можно обращаться и вызывать ее выполнение из разных частей программы.

Использование подпрограмм позволяет избежать дублирования кода, в случае если необходимо один и тот же код писать в разных местах программы.

Библиотеки, которые импортируются в программу (например, математическая библиотека math) состоят из подпрограмм, которые уже кем-то составлены. Программистам не нужно думать о том, какие алгоритмы в них реализованы, а просто применяют их, задумываясь только о том, что именно они делают. Это большая экономия времени. Нет нужды писать алгоритм, который уже был кем-то написан.

Разбиение сложной программы на модули - основной способ борьбы со сложностью программ. Процесс такого разбиения называется декомпозицией программы. Он применяется на всех этапах разработки программ. Большие программные проекты начинаются с этапа проектирования, когда проект представляется в виде взаимосвязанной совокупности модулей. В объектно-ориентированном программировании роль таких модулей играют классы, в функциональном программировании - функции, в модульных языках модули так и называются - модулями. В языке Python для этих целей используются как модули, так и классы. Функции являются простейшим видом модуля. Научиться программировать в функциях - это непростая задача, которая ставится перед студентами первого курса, специализирующихся в информационных технологиях.

Функции - один из основных инструментов математиков - предмет математического (функционального) анализа. И для программистов функция - базисный элемент построения программ.

В программировании принято рассматривать два понятия - функция и процедура. В некоторых языках ООП используется термин метод класса, который может быть реализован либо как функция, либо как процедура.

В Python существует два вида реализации подпрограмм: функции в роли процедуры и функции в классическом понимании.

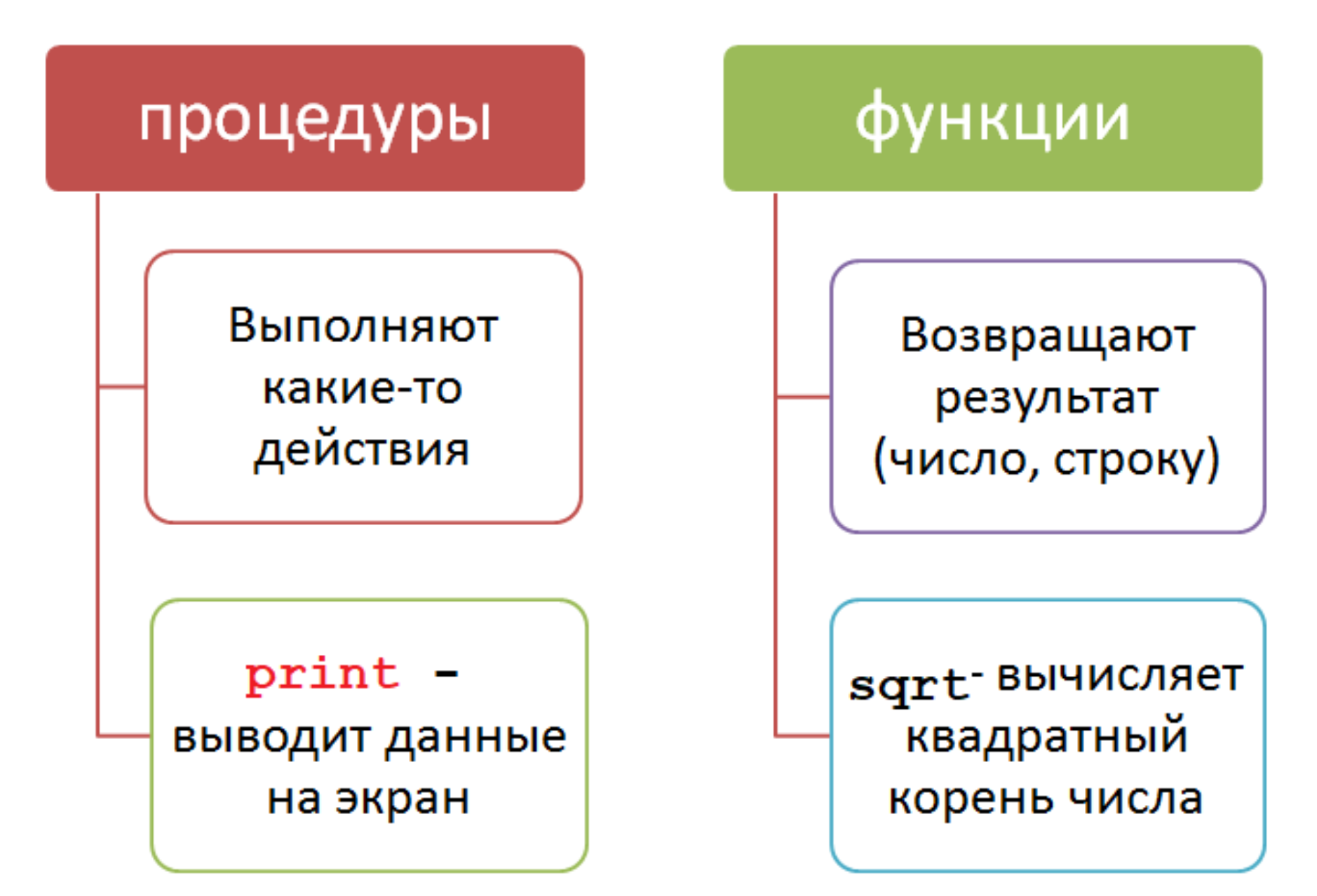
Функция в роли процедуры призвана не возвратить значение в основную программу, а вывести его, либо выполнить какие-либо действия с глобальными переменными, при этом не возвращая полученные значения основной программе (не используя ключевое слово return).

Каждая подпрограмма должна решать только одну задачу, либо только что-то вычислять, либо выводить какие-либо данные, либо делать что-то еще.

Подпрограммы-процедуры выполняют некоторые действия, например, выводят результат на экран в определенном виде. Подпрограммы-функции возвращают результат (число, символьную строчку и т.д.), который мы можем использовать в основной программе.

В Python нет формального разделения подпрограмм на функции и процедуры (как, например, в Паскале или Си), и процедурой можно считать функцию, возвращающую пустое значение – в основном используется единственный термин – функция.

Существует два типа подпрограмм:



**Рисунок 1. Классификация подпрограмм в Python**

Объявление классической функции, близкое к объявлению функции в математике, содержит заголовок и тело функции. В заголовке дается описание входных параметров, называемых также аргументами функции. Тело функции задает алгоритм вычисления значения функции на основе значений входных аргументов. Результат вычислений представляет значение, возвращаемое функцией при ее вызове.

Классическая функция возвращает одно значение - один результат. Однако, зачастую требуется создавать и возвращать несколько результатов. Для решения этой проблемы и были введены классические процедуры, у которых формальные параметры разделялись на две, а иногда и на три группы - входные, выходные и обновляемые.

У процедуры нет возвращаемого значения, но ее параметры делятся на группы. В теле процедуры по значениям входных параметров вычисляются значения выходных параметров.

Тем не менее и классическая функция позволяет фактически возвращать несколько результатов. Формально возвращается один результат - одно значение, но этот результат представляет сложную структуру данных, в которой упакованы все результаты, созданные функцией.

Функции вызываются в выражениях. Вызов процедуры - это оператор, такой же, как и оператор присваивания или цикла. Обычно, если результатом является одно простое значение, то соответствующий алгоритм вычислений реализуется как функция. Если результатов вычислений несколько, то предпочтительно создавать процедуру.

Существование в языке программирования одновременно процедур и функций избыточно, хотя позволяет более точно описать логику вычислений. Понятно, что любую функцию можно задать процедурой, рассматривая результат, возвращаемый функцией, как выходной параметр. Но и процедуру можно представить в виде функции. Один способ такого представления состоит в рассмотрении так называемых функций с побочным эффектом, такие функции имеют не только входные параметры, но и выходные. Один из результатов такая функция возвращает в качестве результата, остальные через выходные параметры. Так устроены функции в языке С, С++. Другой, более универсальный способ использует тот факт, что результат, возвращаемый функцией, может быть сложного типа, так что возвращаемый функцией объект содержит совокупность значений, представляющих выходные параметры.

Формально, в языке Python все, что задается определением **def**, является функцией (class function).

В мире Python принято использовать термин "метод", когда речь идет о методах класса, в остальных случаях использовать термин "функция". Термин "процедура", как правило не используется. Важно различать процедуры и функции, поэтому, нарушая традиции Python, использую термин "метод" как общее понятие процедур и функций и, если необходимо, уточняю, что метод реализован как функция или как процедура.

Все параметры метода - это входные параметры. Выходных параметров методы Python не имеют. Оператор **return** в теле метода, как и положено, возвращает результат вычисления функции. Методы Python могут использоваться как классические функции, так и как процедуры. Классическая функция Python обязательно содержит в теле функции оператор **return**. Вызов такой функции является первичным выражением и может быть сделан всюду, где допускаются по синтаксису выражения.

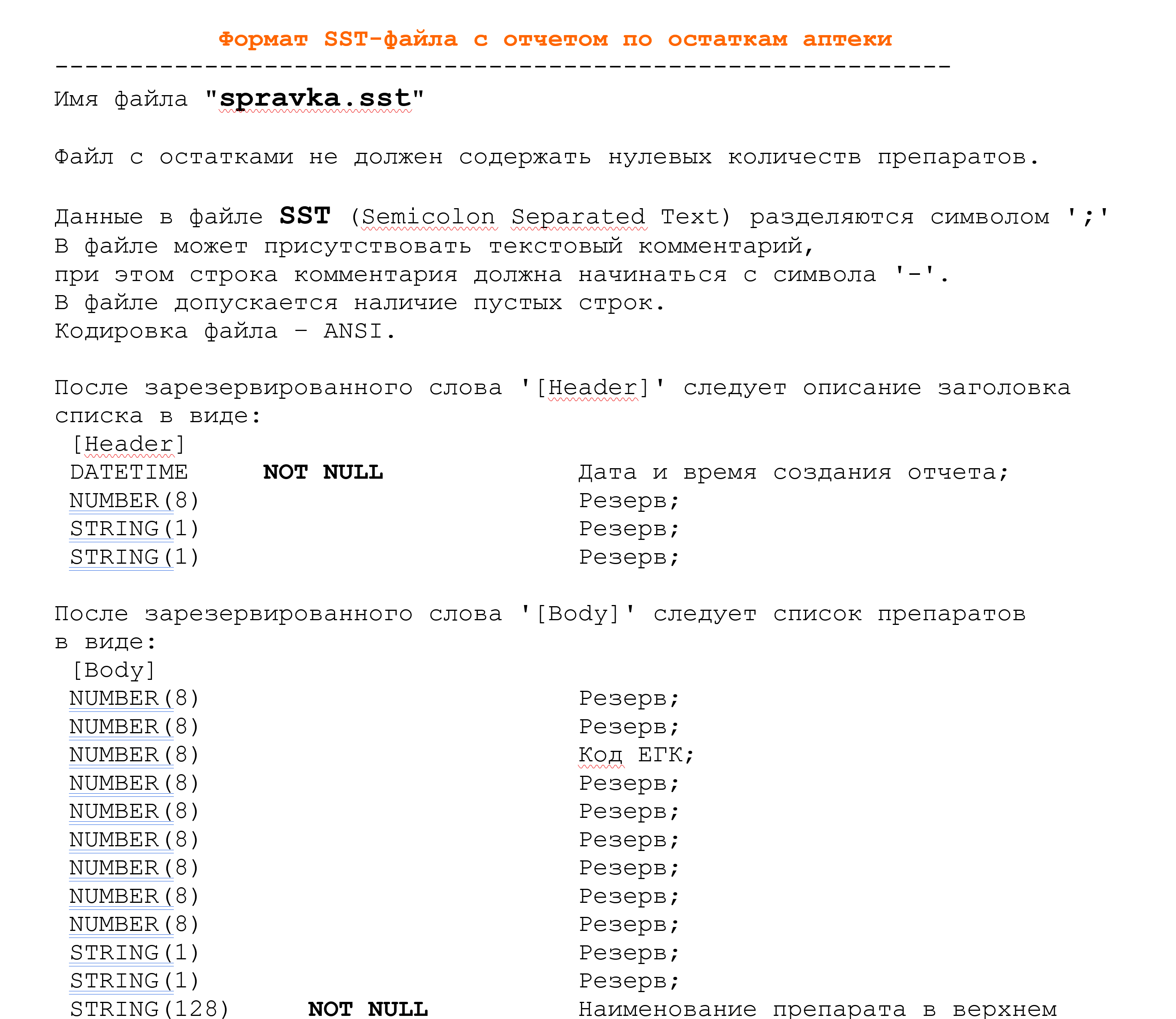
Процедура начинается со слова def (от англ. - define - определить). Ключевое слово def сообщает интерпретатору, что перед ним определение функции. За def следует имя функции. Оно может быть любым, также как и всякий идентификатор, например, переменная. В программировании весьма желательно давать всему осмысленные имена. После имени функции ставятся скобки. В приведенном примере они пустые. Это значит, что функция не принимает никакие данные из вызывающей ее программы. Однако она могла бы их принимать, и тогда в скобках были бы указаны так называемые параметры.

После двоеточия следует тело, содержащее инструкции, которые выполняются при вызове функции. Следует различать определение функции и ее вызов. В программном коде они не рядом и не вместе. Можно определить функцию, но ни разу ее не вызвать. Нельзя вызвать функцию, которая не была определена. Определив функцию, но ни разу не вызвав ее, вы никогда не выполните ее тела.

Все операторы, которые выполняются в процедуре, записываются с отступом. Чтобы выполнить процедуру, в основной программе необходимо вызвать ее по имени и не забыть написать скобки. Вызывать процедуру в программе можно сколько угодно раз.

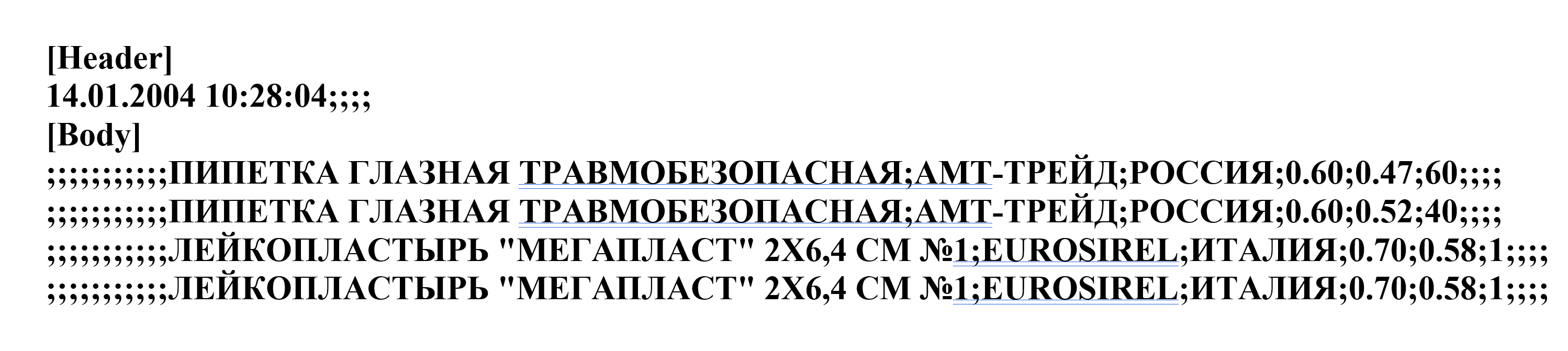
# 2. Практическая реализация алгоритмов в виде программного кода

Ознакомившись со справочной информацией по файлам формата Semicolon Separated Text и изучив предполагаемый функционал будущих функциональных модулей программного обеспечения по обработке базы данных Аптеки, была составлена предварительная структура из нескольких модулей. В дальнейшем мною были реализованы не только все необходимые функции с учётом желаемого функционала, но и возможность работы в меню для удобства пользователя.



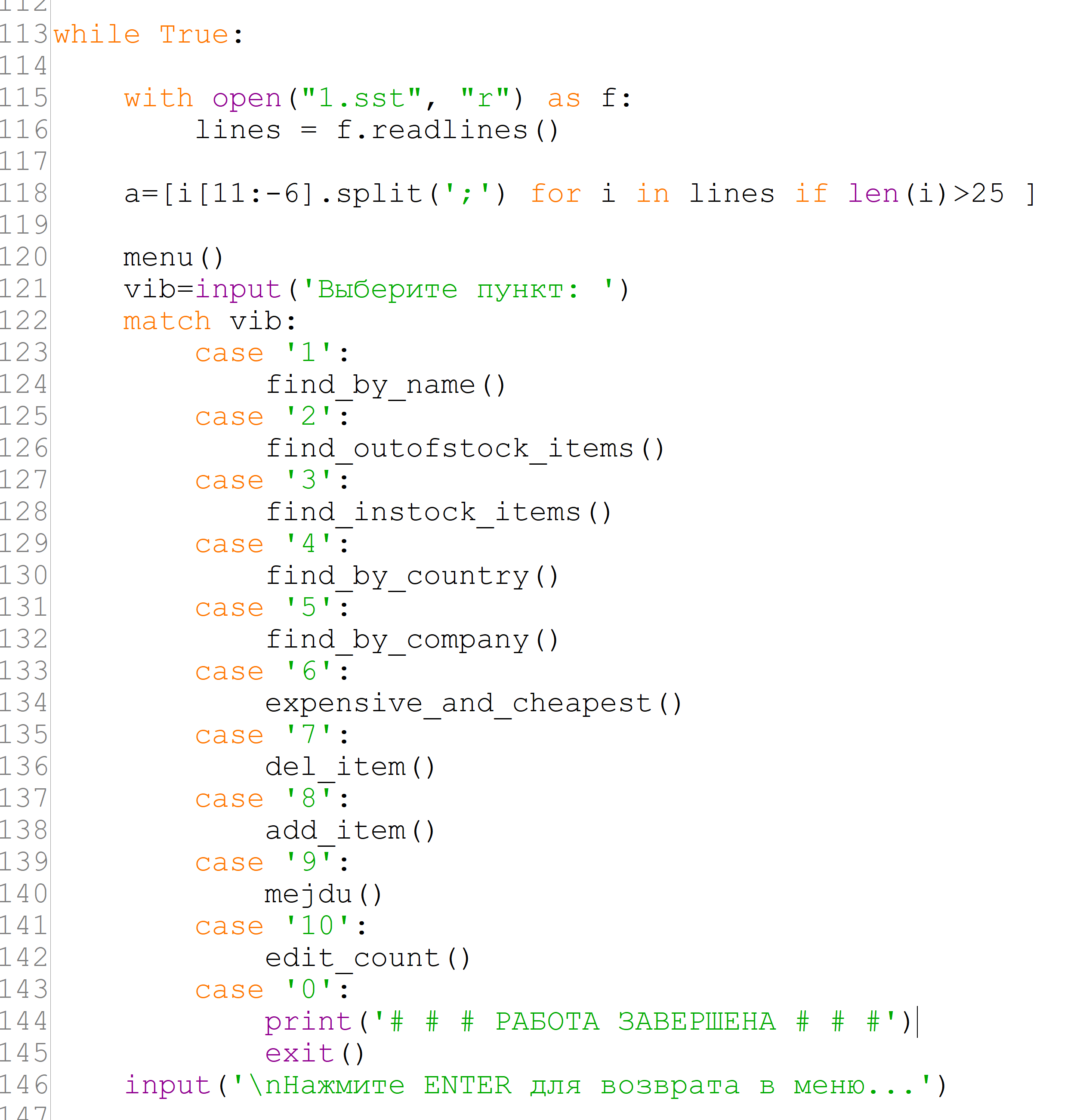
**Рисунок 2. Справочная информация по файлам SST**

При работе с файлом базы данных необходимо было учитывать особенности формата записей согласно структуре номенклатурных позиций. В связи с этим требовалось реализовать в программном обеспечении функционал отвечающий схожим стандартам формата при изменении файла, а не только получения информации.



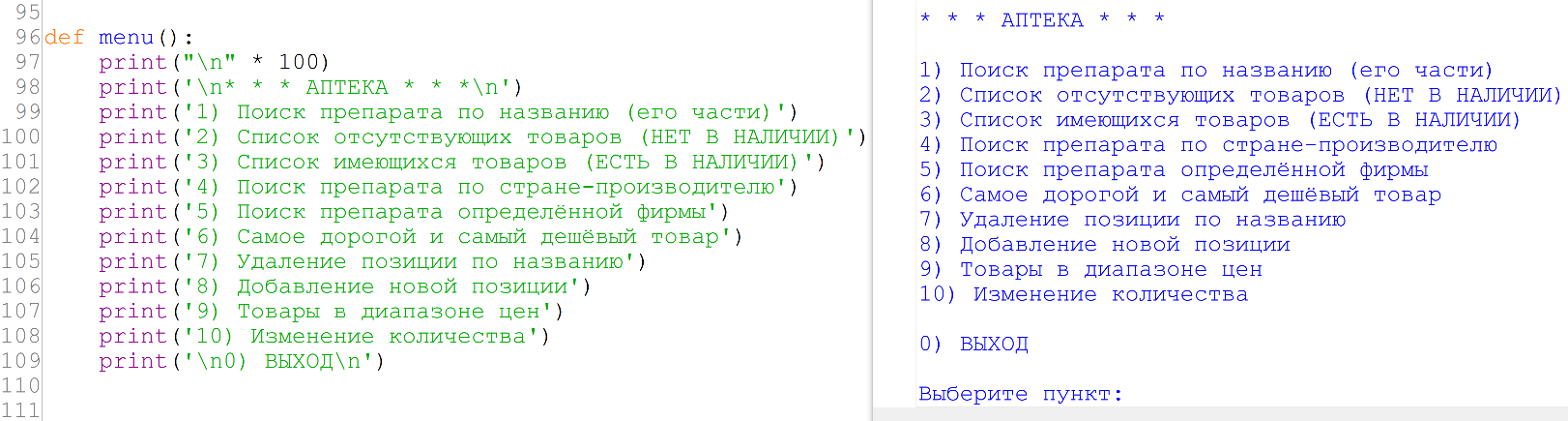
**Рисунок 3. Пример формата в файле базы данных**

В процессе реализации было написано основное тело программы, которое имеет функционал подключения базы данных из файла, чтение имеющейся информации и работа пользовательского меню.

****

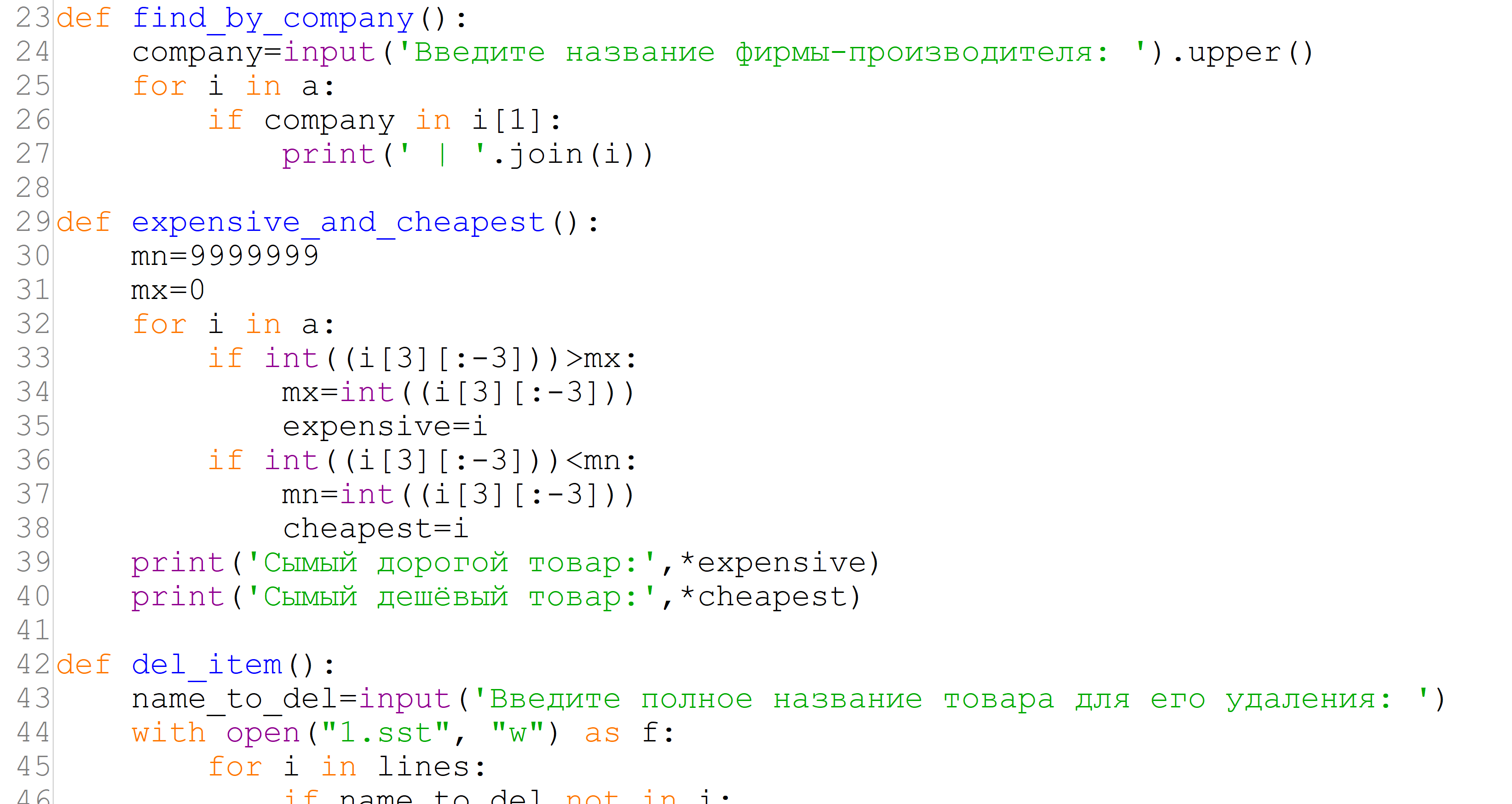
**Рисунок 4. Код Python тела основной программы**

При разработке программы были написаны на все необходимые подпрограммы для обеспечения необходимого функционала при работе с фалом базы данных. Данные функции вызываются пользователем посредством выбора в меню и соответствуют своим названиям.

****

**Рисунок 5. Программный код и вид пользовательского меню**

Во время работы с программой пользователь имеет расширенный функционал, позволяющий ему осуществлять поиск необходимого препарата по его названию (достаточно ввести даже часть), вывести информацию по отсутствующим позициям или наоборот имеющимся в наличии. Реализованы возможности поиска товаров по стране- производителю, по наименованию конкретной фирмы. Есть поиск информации о самом дорогом и самом дешёвом товаре или товарах в заданном ценовом диапазоне. Кроме того, имеется возможность удаления интересующих позиций, добавление новых и изменение количества определённого товара в файле базы данных.

**Рисунок 6. Пример нескольких функций**

# Заключение

При выполнении индивидуального проекта мною были решены все задачи для достижения поставленной цели. В частности я разобрался в новой для меня фармацевтической области, изучил основные принципы ведения электронного документооборота, ознакомился с техническим заданием и cпланировал основные этапы разработки программного обеспечения, исследовал способы обработки массива информации из файла базы данных. Итогом проделанной работы стала непосредственная реализация задуманных алгоритмов для обработки информации из базы данных в виде программного обеспечения.

В заключение проделанной работы хочу сделать вывод, что информационные технологии невероятно актуальны в нашей повседневной жизни и я считаю, что современный человек должен стремиться к тому, чтобы в них разбираться, так как грамотность в этой области очень важна в наши дни. Человек, который живёт в наш век огромного количества информации и тотальной цифровизации должен ориентироваться в том, как всё устроено вокруг него: программы, сайты, приложения и иные цифровые сервисы.

# Список литературы

1. Гниденко, И. Г. Технологии и методы программирования : учебное пособие для вузов / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. – Москва : Издательство Юрайт, 2022. – 235 с.
2. Копырин, А. С. Программирование на Python : учебное пособие / А. С. Копырин, Т. Л. Салова. – Сочи : СГУ, 2018. – 48 с.
3. Косицин, Д. Ю. Язык программирования Python : учебно-методическое пособие / Д. Ю. Косицин. – Минск : БГУ, 2019. – 136 с.
4. МакГрат, М. Программирование на PYTHON [Текст] / Майк МакГрат; [пер. с англ. М. А. Райтмана]. - Москва :Эксмо, 2015. - 192 с.
5. Мартелли, А. Python : справочник : полное описание языка / Алекс Мартелли, Анна Рейвенскрофт, Стив Холден ; [перевод с английского А. Г. Гузикевича]. – 3-е изд. – Москва ; Диалектика ; Санкт-Петербург : Диалектика, 2020. –892 с.
6. Пэйн, Б. Python для детей и родителей играй и программируй / Брайсон Пэйн; [пер. с англ. М.А. Райтмана]. – Москва: Издательство «Э», 2017. – 352 с.
7. Тарланов, А. Т. Основы языка программирования Python : учебно-методическое пособие / А. Т. Тарланов, Ш. Г. Магомедов. – Москва : РТУ МИРЭА, 2019. – 107 с.
8. Трофимов, В. В. Информатика в 2 т. Том 2 : учебник для среднего профессионального образования / В. В. Трофимов. – 3-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2022. – 406 с. .
9. Чернышев, С. А.  Основы программирования на Python : учебное пособие для вузов / С. А. Чернышев. – Москва : Издательство Юрайт, 2022. – 286 с.
10. Широбокова, С. Н. Программирование на языке Python для лабораторных занятий : учебное пособие / С. Н. Широбокова, А. А. Кацупеев, А. В. Сулыз. – Новочеркасск : ЮРГПУ, 2020. – 104 с.