

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИТМО
ITMO University

ДОМАШНЯЯ РАБОТА

По дисциплине Программирование

Тема работы Реализация программной модели инфокоммуникационной системы

Обучающийся Сакулин Иван Михайлович

Факультет Факультет прикладной информатики

Группа К3121

Направление подготовки 11.03.02 Инфокоммуникационные технологии и системы связи

Образовательная программа Программирование в инфокоммуникационных системах

Обучающийся	_____	_____	Сакулин И.М.
	(дата)	(подпись)	(Ф.И.О.)
Руководитель	_____	_____	Казанова П.П.
	(дата)	(подпись)	(Ф.И.О.)

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ	4
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	5
1.1 Функциональные требования	5
1.2 Нефункциональные требования	5
1.3 Хранение записей	6
2 РАБОТА С ДАННЫМИ	7
2.1 Представление записей	7
2.2 Индексация	7
2.3 Добавление записи	8
2.4 Загрузка из файла	8
2.5 Сохранение в файл	9
2.6 Удаление записи.....	9
2.7 Сортировка записей.....	9
2.8 Получение записей по дате и категории	9
2.9 Работа с датами	10
3 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС	11
3.1 Цикл работы.....	11
3.2 Рисование страницы	11
3.3 Ввод значений	12
4 РЕАЛИЗАЦИЯ.....	13
4.1 Алгоритм работы	13
4.2 Загрузка коллекции.....	15
4.3 Добавление продукта в коллекцию.....	15
4.4 Просмотр всего записанного в программу.....	15
4.5 Просмотр покупок по дате и категории	15
4.6 Распределение по стоимости	15
4.7 Удаление требуемых записей	16
4.8 Сохранение и выход из приложения	16
4.9 Справка	16
ЗАКЛЮЧЕНИЕ	17

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	18
ПРИЛОЖЕНИЕ А Исходный код файла data.py	19
ПРИЛОЖЕНИЕ Б Пример файла хранения base.csv	26
ПРИЛОЖЕНИЕ В Исходный код файла dtf.py	27
ПРИЛОЖЕНИЕ Г Исходный код файла tui.py	28
ПРИЛОЖЕНИЕ Д Исходный код файла main.py	35
ПРИЛОЖЕНИЕ Е Интерфейс	42

ВВЕДЕНИЕ

В отчёте представлено решение домашней работы: программный код на python для реализации технического задания и объяснение его работы. Отчёт, как и программа, состоит из трёх частей: работа с данными, пользовательский интерфейс, реализация.

Техническое задание

Создать программное обеспечение системы обработки данных: «Программа для контроля собственных денежных средств».

Необходимо реализовать следующие функции, позволяющие:

1. Добавлять продукт в коллекцию (тип коллекции на ваш выбор).
2. Просматривать все записанное в программу.
3. Просматривать покупки по дате и категории.
4. Распределять их по стоимости от минимальной к максимальной или наоборот.
5. Удалять требуемые записи и выходить из программы.

Дополнительные указания: интерфейс программы реализовать в отдельной функции на ваше усмотрение. Рекомендуется реализовать возможность сохранения данных в файл.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Цель такого вида программ – анализировать, на что тратится больше всего денег. Целевой аудиторией обычно являются люди, следящие за бюджетом, планирующие покупки или ведущие малый бизнес. Для них нужна простота использования и возможность быстро добавлять покупки. У приложения, конечно, есть очень много аналогов, например, "Дзен-мани" или "Money Manager". Они имеют графический интерфейс и удобны в использовании, их пользователям нравятся быстрый ввод данных и доступность. У всех приложений есть общие проблемы в этой области: пользователи забывают вносить покупки, а также тот факт, что некоторые магазины не дают чеки.

1.1 Функциональные требования

Информационная система должна уметь.

- Загружать коллекцию из файла.
- Проверять корректность коллекции.
- Корректно обрабатывать ввод пользователя.
- Добавлять продукт в коллекцию.
- Отображать коллекцию с фильтрами по дате и категории.
- Сортировать коллекцию по стоимости.
- Удалять записи из коллекции.
- Сохранять коллекцию в файл.

1.2 Нефункциональные требования

- Понятность для пользователя.
- Скорость работы.
- Обработка ошибок.

1.3 Хранение записей

Запись храниться в виде кортежа из пяти элементов:

1. уникальный номер,
2. название продукта,
3. цена продукта,
4. категория продукта,
5. дата.

2 РАБОТА С ДАННЫМИ

В этой главе будет рассказано о части программы, которая отвечает за корректную работу с данными, то есть обеспечивает их целостность и корректность при любых взаимодействиях. Эта часть хранится в файле «data.py».

2.1 Представление записей

На время работы программе коллекция хранится в массиве `data` (приложение А, рисунок 2, строка 16), каждый продукт представлен в виде кортежа из пяти элементов:

- уникальный номер (`id`, натуральное число),
- название продукта (`name`, строка до 100 символов),
- цена продукта (`cost`, натуральное число, цена умноженная на 100, то есть последние 2 знака - это копейки),
- категория продукта (`type`, строка до 40 символов),
- дата (`date`, натуральное число «timestamp utc», формат времени в секундах, прошедшее с 1 января 1970 года).

Константами заданы значения максимальных размеров и разделителей в файлах (приложение А, рисунок 2, строки 7-11).

Функция «available» возвращает, загружены ли данные, «display_cost» приводит цену к нормальному виду с точкой, «display_row» приводит запись о продукте в нормальный вид для отображения (рисунок 2, строки 18-24).

2.2 Индексация

Для того, чтобы сделать поиск записей за минимальное время и быстрее сортировать записи, были введены словари, хранящие индексацию. Массив «index_eq» для каждого ID (уникального номера) хранит индекс записи в массиве `data`. Остальные для каждого значения (дата, категория) хранят ID (приложение А, рисунок 3, строки 2-4).

Таким образом, чтобы найти продукт по дате, можно просто взять по дате из словаря все ID с нужной датой, затем посмотреть индекс продукта с таким ID в массиве, после просто получить запись. Сложность поиска становится почти $O(1)$, вместо $O(n)$.

2.3 Добавление записи

Запись можно добавить как от пользователя (через «add_product» - приложение А, рисунок 7, строки 1-7), так и из записи в файла («_add_row_as_product» - приложение А, рисунок 4, строки 14-21).

Перед добавлением записи о продукте, данные проходят несколько этапов.

1. Валидация («_validate_product» - приложение А, рисунок 3, строки 7-35) - проверяется правильность формата значений, преобразует строки в число, проверяет дату.
2. Добавление («_add_product» - приложение А, рисунок 4, строки 1-11) - все корректные записи добавляет в файл, назначает им id.
3. Индексация - добавляет в ID в индексаторы.

2.4 Загрузка из файла

Все записи хранятся в табличном виде в формате csv. Это текстовый файл, где записи о продукте разделяются переносом строки, значения в записи - запятой (пример файла - приложение Б, рисунок 9). В питоне есть встроенный модуль для бесперебойной работы с такими таблицами.

Сначала происходит попытка открытия файла, если его не существует - создаётся пустой. Затем очищаются все данные о текущей коллекции. После, с помощью модуля csv построчно читаются, а затем добавляются корректные записи (приложение А, рисунок 5).

2.5 Сохранение в файл

Все кортежи продуктов преобразуются в список и с помощью модуля csv загружаются в файл (приложение А, рисунок 6). В случае неожиданной ошибки, только проблемная запись не сохраняется (но ошибок, способных появиться на этом этапе не выявлено).

2.6 Удаление записи

Функция «remove_product» (приложение А, рисунок 7, строки 10-29) заменяет запись на значение None и удаляет её ID из всех индексаторов. В последствии все None продукты игнорируются. Это сделано не через «pop», чтобы при удалении значения индексаторов не приходилось восстанавливать из-за сдвига на 1.

2.7 Сортировка записей

Функция «sort_cost» (приложение А, рисунок 8, строки 25-36) сортирует продукты по цене стандартной сортировкой python, затем восстанавливает «index_eq»: сопоставляет ID новый индекс. Сложность получается $O(n \cdot \log n + n)$. Это минимальная сложность алгоритма, при которой поиск, удаление и добавление остаются константными.

2.8 Получение записей по дате и категории

Реализована возможность получать:

- все продукты (приложение А, рисунок 7, строки 32-39),
- продукты с определённой датой (приложение А, рисунок 8, строки 1-10),
- продукты из определённой категории (приложение А, рисунок 8, строки 13-22).

Для этого написаны функции генераторы (позволяют сэкономить память), которые при каждом обращении выдают элемент. Также реализованы вспомогательные функции чтобы предупредить, если записей по запросу нет.

2.9 Работа с датами

Для работы с датой на основе модуля `datetime` был написан файл «`dtf.py`» (приложение В, рисунок 10), в котором собраны нужные функции для работы с датами в формате объекта `datetime` и числа `timestamp`.

3 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

В этой главе будет представлена реализация текстового пользовательского интерфейса (TUI) в файле «tui.py» для красивой и перебойной работы с пользователем в консоли.

3.1 Цикл работы

Каждая функция - это отдельное состояние («state»), на некоторых состояниях рисуется окно, на некоторых информация выводится в текущее окно.

Цикл продолжается, пока программа не будет завершена соответствующей командой перехода в состояние выключения, и состоит из следующих этапов (приложение Г, рисунок 14 и строки 1-4 на рисунке 15).

1. Если нет планируемого состояния, запросить ввод пользователя.
2. Если планируемое состояние появилось или уже было, запускаем.
3. Если есть, подготавливаем новое запрошенное состояние.

Таким образом, состояния могут запускать пользователь и другие состояния.

Пользователь вводит команду и аргументы при наличии, затем если введенная команда доступна, она приводится в действие.

Доступные команды хранятся в виде словарей, где ключ - команда, значение - кортеж из функции-состояния, массива возможных аргументов и пояснения к команде. Проверяются на корректность и устанавливаются команды с помощью функции «set_commands» (приложение Г, рисунок 11, строки 18-33).

3.2 Рисование страницы

Для цветного отображения текста была использована библиотека «colorama», очистка консоли происходит за счёт вызова консольной функции соответствующей ОС (приложение Г, рисунок 11, строки 6-12).

Функции «draw_state» (приложение Г, рисунок 13) и «draw_substate» (приложение Г, рисунок 12, строки 28-37) очищают экран и рисуют шапку страницы с заголовком. Первая также пишет все доступные на данный момент для исполнения команды.

Также предусмотрено рисование таблицы (приложение Г, рисунок 12, строки 1-26), оно нужно, чтобы красиво отображать все продукты.

3.3 Ввод значений

Для ввода значений и обработки их корректности написаны 4 функции (приложение Г, рисунки 15, 16 и 17):

1. «input_bool» – ввод бинарных значений, где у, д, + или 1 - истина, н, - или 0 - ложь;
2. «input_str» – ввод строки (с ограничением длины);
3. «input_float» – ввод числа (с ограниченным числом знаков после запятой и максимальным значением);
4. «input_date» – ввод даты (с проверкой корректности), при пустом вводе возвращает текущую дату.

4 РЕАЛИЗАЦИЯ

В предыдущих главах было рассказано о работе с пользователем и с данными. В этой главе результаты предыдущих объединятся, чтобы сделать программу соответствующую техническому заданию.

4.1 Алгоритм работы

Цикл программы запускается (приложение Д, рисунок ??, строки 1-26) и начинается с основной страницы, где доступно большинство команд (приложение Д, рисунок ??, строки 29-36). Интерфейс с основными командами представлен в приложении Е, на рисунке 25.

Блок-схема алгоритма работы приложения представлена на рисунке 1.

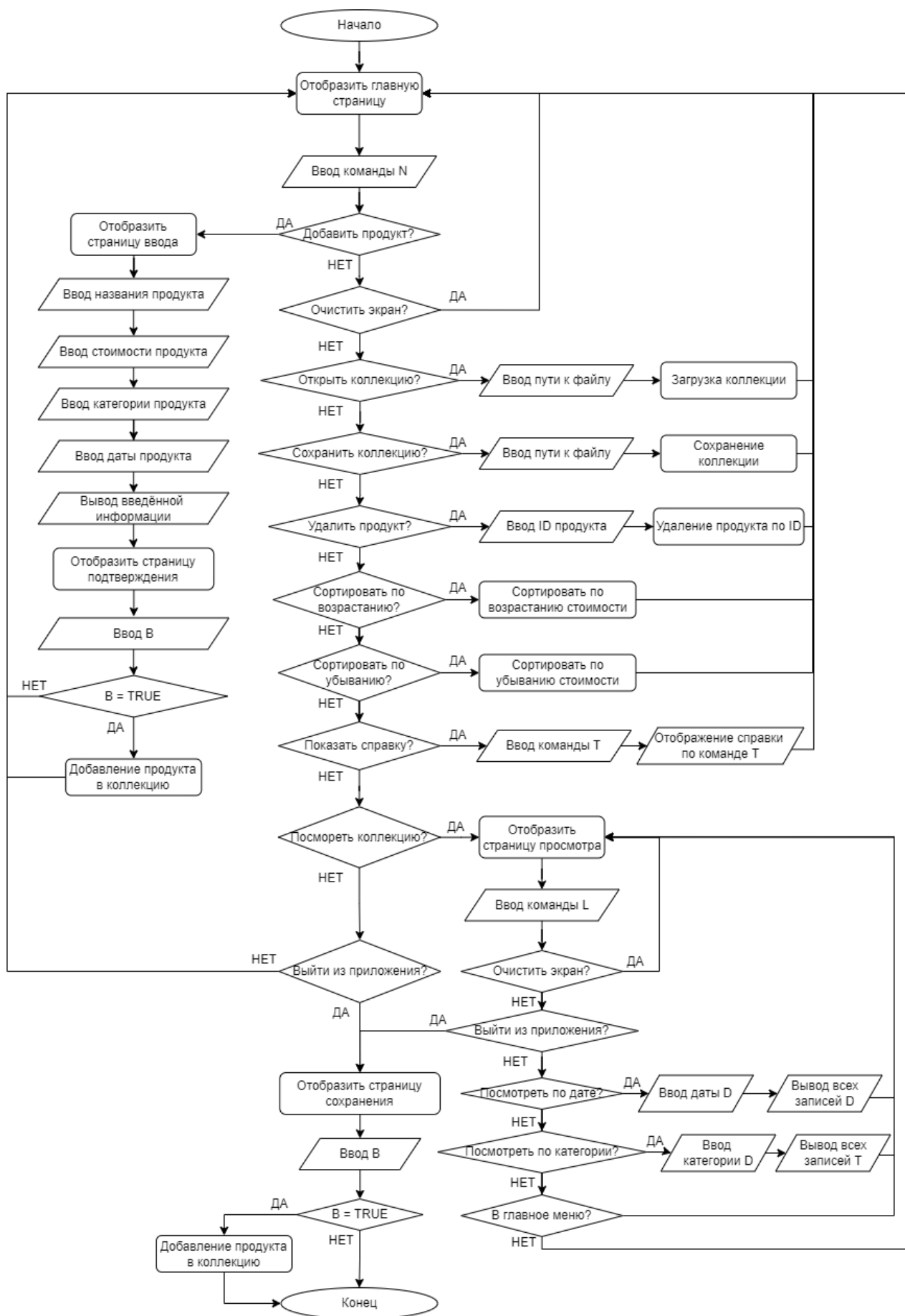


Рисунок 1 — Блок-схема работы приложения

4.2 Загрузка коллекции

С помощью команды можно загрузить файл (приложение Д, рисунок 19, строки 16-27). Коллекция, которая уже была загружена, по решению пользователя, сохраняется (приложение Д, рисунок 19, строки 1-7).

4.3 Добавление продукта в коллекцию

Отображается страница, где поочередно запрашиваются значения (название, цена, категория, дата), а затем подтверждение добавления (приложение Д, рисунок 24). Процесс добавления продукта можно увидеть на рисунках 26 и 27 в приложении Е.

4.4 Просмотр всего записанного в программу

Отображается страница (приложение Е, рисунок 28), где рисуется таблица со всеми записями (приложение Д, рисунок 23).

4.5 Просмотр покупок по дате и категории

На странице просмотра можно ввести команды фильтров (приложение Е, рисунок 29) по дате и категории (приложение Д, рисунок 21).

4.6 Распределение по стоимости

На главной странице можно отсортировать коллекцию по возрастанию или убыванию стоимости соответствующими командами (приложение Д, рисунок 20, строки 22-29).

Описание алгоритма сортировки. Сначала производится встроенная в python сортировка по значению цены. TimSort - это гибридный алгоритм сортировки $O(n \cdot \log n)$, который комбинирует идеи из двух других сортировок: сортировки слиянием и сортировки вставками.

1. Входной массив разбивается на подмассивы фиксированного размера (обычно размер подмассива выбирается в зависимости от количества элементов в исходном массиве).
2. Каждый подмассив сортируется с помощью сортировки вставками.
3. Отсортированные подмассивы объединяются вместе с помощью сортировки слиянием.

Затем восстанавливается индексация: заново сопоставляются ID и индексы в массиве data за один проход $O(n)$. Общая сложность алгоритма получается $O(n \cdot \log n + n)$.

4.7 Удаление требуемых записей

На главной странице можно удалить запись по уникальному номеру – ID (приложение Д, рисунок 20, строки 16-19).

4.8 Сохранение и выход из приложения

Сохранение данных в файл происходит из команд загрузки и закрытия приложения, а также из отдельной команды (приложение Д, рисунок 20, строки 1-13).

Закрытие предлагает сохранить коллекцию, затем закрывает процесс исполнения с помощью «exit(0)» (приложение Д, рисунок 19, строки 1-13).

4.9 Справка

По каждой команде можно получить справку, которая записана в «helps.py» (приложение Д, рисунок 18, строки 18-34).

ЗАКЛЮЧЕНИЕ

Цель практической работы достигнута. Автор представил решения задачи с выполненными пунктами технического задания: загрузка и сохранение данных в файл, добавление и удаление записей, сортировка по убыванию и возрастанию цены, просмотр записей, в том числе по дате и категории. Применил навыки поиска информации в интернете и программирования на python.

Исходный код программы можно найти в репозитории [1].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Исходный код программы – репозиторий github [Электронный ресурс]: [сайт]. - URL: <https://github.com/OneTwoZzzPlus/programming-homework> (дата обращения: 06.11.2024)

ПРИЛОЖЕНИЕ А

Исходный код файла data.py

```
1 import csv
2 import dtf
3 import logging
4
5
6 # Размеры и формат данных
7 PRODUCT_NAME_LEN = 100
8 PRODUCT_TYPE_LEN = 40
9 PRODUCT_COST_MAX = 4_294_967_296 # COST.00 x100
10 DELIMITER = ','
11 LINETERMINATOR = '\r'
12
13 # Хранилище данных
14 current_path = ''
15 # ID | NAME | COST*100 | TYPE | UTC_TIME
16 data: list[tuple[int, str, int, str, int]] = None
17
18 # Доступность
19 available = lambda: data is not None
20
21 # Для отображения
22 display_cost = lambda x: f"{x/100:.2f}"
23 display_row = lambda x: (str(x[0]), x[1], display_cost(x[2]),
24                          x[3], dtf.display_utc(x[4]))
25
26 # Для рисования таблиц
27 table_head = ["ID", "Название", "Цена", "Категория", "Дата"]
28 id_length = lambda: len(str(len(data))) if available() else 7
29 cost_length = len(str(PRODUCT_COST_MAX)) + 1
30 table_width_min = lambda: [id_length(), 10, 6, 10, 8]
```

Рисунок 2 — Код файла data.py

```

1  # Индексаторы
2  index_eq: list[int] = []  # Сопоставление индексаторов и ID
3  index_type: dict[str, list[int]] = {}  # Индексатор по категории
4  index_date: dict[int, list[int]] = {}  # Индексатор по дате
5
6
7  def _validate_product(index: int, pname, pcost, ptype, pdate):
8      """ Преобразование корректных данных в продукт O(1) """
9      try:
10         # Название
11         product_name = str(pname)
12         if len(product_name) >= PRODUCT_NAME_LEN:
13             raise TypeError(
14                 f"длина названия превышает {PRODUCT_NAME_LEN}")
15         # Цена
16         product_cost = int(pcost)
17         if product_cost <= 0:
18             raise TypeError(f"цена не положительная!")
19         if product_cost >= PRODUCT_COST_MAX:
20             raise TypeError(f"цена превышает {PRODUCT_COST_MAX}!")
21         # Категория
22         product_type = str(ptype)
23         if len(product_type) >= PRODUCT_TYPE_LEN:
24             raise ValueError(
25                 f"длина названия превышает {PRODUCT_NAME_LEN}")
26         # Дата
27         product_date = dtf.correct(int(pdate))
28         if product_date <= 0 or product_date >= dtf.MAX_DATE:
29             raise TypeError
30         return (index, product_name, product_cost,
31                 product_type, product_date)
32     except (TypeError, ValueError) as e:
33         logging.info(f"Проигнорировано, т.к. {e}: {(
34             pname, pcost, ptype, pdate)})")
35     return None

```

Рисунок 3 — Код файла data.py

```

1 def _add_product(pname, pcost, ptype, pdate):
2     """ Добавление продукта из данных O(1) """
3     global data
4     index_id = len(data)
5     xv = _validate_product(index_id, pname, pcost, ptype, pdate)
6     if xv is not None:
7         data.append(xv)
8         # Индексируем
9         index_eq.append(index_id)
10        for value, indexer in zip(xv[3:5], [index_type, index_date]):
11            indexer.setdefault(value, []).append(index_id)
12
13
14 def _add_row_as_product(x: list):
15     """ Добавление продукта из строки файла O(1) """
16     if len(x) == 4:
17         _add_product(*x)
18     else:
19         logging.info(
20             f"Строка проигнорировано, т.к. разделителей '{DELIMITER}'\
21             f"слишком {"МНОГО" if len(x) > 4 else "МАЛО"}: {x}")

```

Рисунок 4 — Код файла data.py

```
1 def load_file(path: str='base.csv') -> bool:
2     """ Загрузить коллекцию из файла O(n) """
3     global data, current_path, index_type, index_date, index_eq
4     try:
5         # Читаем файл
6         file = open(path, 'r', encoding='utf-8')
7         file_reader = csv.reader(file, delimiter=DELIMITER,
8                                 lineterminator=LINETERMINATOR)
9         current_path = path
10        # Очищаем место хранения данных и индексы
11        data = []
12        index_eq = []
13        index_type, index_date = {}, {}
14        # Построчно читаем записи
15        for x in file_reader:
16            _add_row_as_product(x)
17        # Закрываем файл
18        file.close()
19        return True
20    except FileNotFoundError:
21        # Создаём файл, если его не существует
22        try:
23            logging.info(f"Создаём файл {path}")
24            open(path, 'w', encoding='utf-8')
25            return load_file(path)
26        except (PermissionError, FileNotFoundError):
27            return False
28    except PermissionError:
29        return False
```

Рисунок 5 — Код файла data.py

```

1 def save_file(path: str=None) -> bool:
2     """ Сохранить коллекцию в файл O(n) """
3     global data, current_path
4     if path is None:
5         path = current_path
6         # Не сохранять, если база не загружена
7         if not available():
8             return False
9     try:
10        # Открываем файл для записи
11        file = open(path, 'w', encoding='utf-8')
12        file_writer = csv.writer(file, delimiter=DELIMITER,
13                                lineterminator=LINETERMINATOR)
14        current_path = path
15        # Построчно записываем
16        for x in data:
17            # Если запись не удалена
18            if x is not None:
19                try:
20                    file_writer.writerow(list(x[1:]))
21                except Exception as e:
22                    logging.warning(
23                        f"Неожиданная ошибка '{e}' при записи {x}"
24                    )
25        # Закрываем файл
26        file.close()
27        return True
28    except PermissionError:
29        return False

```

Рисунок 6 — Код файла data.py

```

1  def add_product(product_name: str, product_cost: float,
2                  product_type: str, product_date: dtf.date):
3      """ Добавление продукта O(1) """
4      _add_product(
5          product_name, int(product_cost * 100),
6          product_type, dtf.datetime_to_utc(product_date)
7      )
8
9
10 def remove_product(index: int) -> bool:
11     global data, index_type, index_date, index_eq
12     # Проверяем существование записи
13     if not 0 <= index < len(data):
14         return False
15     if data[index_eq[index]] is None:
16         return False
17     # Получаем запись
18     i, _, _, pt, pd = data[index_eq[index]]
19     # Удаляем (заменяем на None) запись
20     data[index_eq[index]] = None
21     # Удаляем индекс из индексаторов
22     for value, indexer in zip([pt, pd], [index_type, index_date]):
23         try:
24             indexer[value].remove(index)
25             if not indexer[value]:
26                 del indexer[value]
27         except:
28             pass
29     return True
30
31
32 def get_list(count: int=None, start: int=0):
33     """ Получение списка продуктов (iterator по O(1)) """
34     if count is None:
35         count = len(data)
36     if start < len(data):
37         for x in data[start:min(start + count + 1, len(data))]:
38             if x is not None:
39                 yield display_row(x)

```

Рисунок 7 — Код файла data.py

```

1 def empty_list_date(dt: dtf.date):
2     """ Проверка на пустоту результата поиска по дате O(1) """
3     return dtf.datetime_to_utc(dt) not in index_date
4
5
6 def get_list_date(dt: dtf.date):
7     """ Поиск по дате (iterator по O(1)) """
8     for i in index_date[dtf.datetime_to_utc(dt)]:
9         if data[index_eq[i]] is not None:
10             yield display_row(data[index_eq[i]])
11
12
13 def empty_list_type(t: str):
14     """ Проверка на пустоту результата поиска по категории O(1) """
15     return t not in index_type
16
17
18 def get_list_type(t: str):
19     """ Поиск по категории (iterator по O(1)) """
20     for i in index_type[t]:
21         if data[index_eq[i]] is not None:
22             yield display_row(data[index_eq[i]])
23
24
25 def sort_cost(inc: bool):
26     """
27     Сортировка по возрастанию(inc=True) /
28     убыванию(inc=False) стоимости O(n*logn + n)
29     """
30     data.sort(key=lambda x: 0 if x is None else x[2], reverse=not(inc))
31     # Восстанавливаем индексацию через index_eq
32     i = 0
33     for p in data:
34         if p is not None:
35             index_eq[p[0]] = i
36             i += 1
37

```

Рисунок 8 — Код файла data.py

ПРИЛОЖЕНИЕ Б

Пример файла хранения base.csv

1	Молоко 2.5%,6598,молоко,1730754000
2	Молоко 3.2%,7065,молоко,1731099600
3	хлеб ржаной,4500,хлеб,1730754000
4	Булочка с сосиской,5245,хлеб,1730754000
5	Хлеб пшеничный,3900,хлеб,1731099600
6	Колбаса,30564,колбаса,1731099600

Рисунок 9 — Пример файла хранения base.csv

ПРИЛОЖЕНИЕ В

Исходный код файла dtf.py

```
1  import datetime
2
3  date = datetime.date
4  MAX_DATE = 32535205199
5
6  display_utc = lambda x: utc_to_datetime(x).strftime('%d.%m.%y')
7  display_data = lambda x: x.strftime('%d.%m.%y')
8
9  def utc_to_datetime(x: int):
10     return datetime.date.fromtimestamp(x)
11
12  def datetime_to_utc(x: datetime.date):
13     return int(datetime.datetime(x.year, x.month, x.day).timestamp())
14
15  correct = lambda x: datetime_to_utc(utc_to_datetime(x))
16
17  now = lambda: datetime.date(
18     datetime.datetime.now().year,
19     datetime.datetime.now().month,
20     datetime.datetime.now().day
21 )
22
23  def validate_point_date(r: str):
24     try:
25         rs = r.split('.')
26         year = int(rs[2])
27
28         if year < 100:
29             year += 2000
30         elif 100 <= year <= 1970 or year > 3000:
31             raise ValueError
32
33         return datetime.date(year, int(rs[1]), int(rs[0]))
34     except (ValueError, TypeError, IndexError) as e:
35         return None
```

Рисунок 10 — Код файла dtf.py

ПРИЛОЖЕНИЕ Г

Исходный код файла tui.py

```
1  import os
2  import shutil
3  import dtf
4  from typing import Callable
5
6  from colorama import init as colorama_init, Fore, Back, Style
7  CR = Style.RESET_ALL
8  colorama_init()
9
10
11  # Очистка терминала в зависимости от ОС
12  cls = lambda: os.system('cls' if os.name=='nt' else 'clear')
13
14  # Таблицы
15  _table_small: bool = False
16
17  # Текущие доступные команды
18  _comm: dict[str, tuple[Callable, list, str]] = {}
19
20
21  def set_commands(commands: dict[str, tuple[Callable, list, str]]):
22      """ Проверяем на корректность переданные доступные команды """
23      if not (isinstance(commands, dict)
24              and all(isinstance(k, str) and isinstance(v, tuple)
25                      for k, v in commands.items())
26              and all(isinstance(v[0], Callable) and
27                      isinstance(v[1], list) and isinstance(v[2], str)
28                      for v in commands.values())):
29          ):
30          raise TypeError(
31              'COMMANDS type not dict[str, tuple[Callable, str]]')
32  global _comm
33  _comm = commands
```

Рисунок 11 — Код файла tui.py

```

1 def draw_table_head(head: list[str], width_min: list[str]):
2     global _table_small
3     if not ( len(head) == len(width_min) ):
4         raise TypeError("Different list sizes!")
5     x, _ = shutil.get_terminal_size((80, 20))
6     count = len(head)
7     min_w = sum(width_min) + 3 * (count - 1)
8     if min_w > x:
9         _table_small = None
10    else:
11        print(Fore.GREEN, " | ".join(head), CR, sep='')
12    if _table_small:
13        print(f"{Fore.GREEN}{'', ' '.join(head)}{CR}")
14        # Вертикальная линия
15        print(f'{Fore.GREEN}{'-' * (x - 1)}'\
16              f'{'-' * int(not(x % 2))}{CR}')
17
18 def draw_table_row(row):
19     global _table_small
20     x, _ = shutil.get_terminal_size((80, 20))
21     if _table_small:
22         print(f"{'', ' '.join(row)}")
23         # Вертикальная линия
24         print(f'{'-' * (x - 1)}{'-' * int(not(x % 2))}')
25     else:
26         print(" | ".join(row))
27
28 def draw_substate(title: str):
29     """ Отрисовка временной страницы TUI """
30     cls() # Очистка экрана
31     x, _ = shutil.get_terminal_size((80, 20))
32     # Размеры линий
33     equ = (x - len(title) - 3) // 2
34     eqc = int(not(x % 2))
35     # Линия с подписью
36     print(f'{Fore.GREEN}{'=' * equ} {title} '\
37           f'{'=' * equ}{'=' * eqc}{CR}')

```

Рисунок 12 — Код файла tui.py

```

1 def draw_state(title: str,
2               caption_up: str='',
3               caption_down: str=''):
4     global _comm
5     """ Отрисовка страницы TUI """
6     cls() # Очистка экрана
7     x, _ = shutil.get_terminal_size((80, 20))
8     # Размеры линий
9     equ = (x - len(title) - 3) // 2
10    eqf = 2 * equ + len(title) + 2
11    eqc = int(not(x % 2))
12    # Линия с подписью
13    print(f'{Fore.GREEN}{'=' * equ} {title} '\
14          f'{'=' * equ}{'=' * eqc}{CR}')
15    # Подпись сверху при наличии
16    if caption_up != '':
17        print(caption_up)
18    # Список команд
19    com = [f'{Fore.CYAN}{key}{' ' if _comm else ''}'\
20           f'{' ' .join(_comm[key][1])}{CR} - {_comm[key][2]} '
21           for key in _comm]
22    # Расчёт размера колонн
23    com_len = [(len(key) + len(_comm[key][2]) + bool(_comm)
24                + len(' ' .join(_comm[key][1])) + 4)
25               for key in _comm]
26    count_columns = x // max(com_len)
27    width_columns = x // count_columns
28    # Отображение списка команд в табличном виде
29    for i in range(0, len(com), count_columns):
30        out = ''
31        for j in range(i, min(i + count_columns, len(com))):
32            out += com[j] + (' ' * (width_columns - com_len[j]))
33        print(out)
34    # Подпись снизу при наличии
35    if caption_down != '':
36        print(caption_down)
37    # Вертикальная линия
38    print(f'{Fore.GREEN}{'=' * eqf}{'=' * eqc}{CR}')
```

Рисунок 13 — Код файла tui.py

```

1 def run(start: Callable):
2     """ Приём команд и переход в другое состояние """
3     global _comm
4     ret: tuple[Callable, tuple] = start, ()
5     while True:
6         try:
7             # Если нет состояния, запросить у пользователя
8             if ret is None:
9                 r = input('>>> ').split()
10                if r:
11                    if r[0] in _comm:
12                        if len(r) == 1:
13                            ret = (_comm[r[0]][0], ())
14                        else:
15                            ret = (_comm[r[0]][0], tuple(r[1:]))
16                    else:
17                        raise KeyError
18                if ret is None:
19                    raise KeyError
20                # Запустить состояние
21                raw_ret = ret[0](*ret[1])
22                # Обработать запрос на другое состояние
23                if raw_ret is None:
24                    ret = None
25                elif isinstance(raw_ret, Callable):
26                    ret = (raw_ret, tuple())
27                elif isinstance(raw_ret, tuple):
28                    if (len(raw_ret) == 0 or
29                        not isinstance(raw_ret[0], Callable)):
30                        ret = None
31                    else:
32                        if len(raw_ret) == 1:
33                            ret = (raw_ret[0], tuple())
34                        else:
35                            ret = (raw_ret[0], tuple(raw_ret[1:]))
36                else:
37                    ret = None

```

Рисунок 14 — Код файла tui.py

```

1         except (EOFError, KeyError) as e:
2             pass
3         except (KeyboardInterrupt) as e:
4             exit(0)
5
6     def input_bool() -> bool:
7         """ Ввод bool """
8         while True:
9             try:
10                 r = input('[д/н] > ')
11                 if r in 'уд1+':
12                     return True
13                 elif r in 'нн0-':
14                     return False
15             except (EOFError, ValueError, TypeError) as e:
16                 pass
17             except (KeyboardInterrupt) as e:
18                 exit(0)
19
20
21     def input_str(max_lenght: int, s: str=" ") -> str:
22         """ Ввод str """
23         while True:
24             try:
25                 # Считываем строку, убирая пустые символы
26                 r = input(f'{s} > ')
27                 # Проверяем длину, корректность символов и пустоту
28                 if not 0 < len(r) <= max_lenght:
29                     raise ValueError
30                 if any(x in '\r\n\t,;' for x in r):
31                     raise ValueError
32                 if r.isspace():
33                     raise ValueError
34                 return r
35             except (EOFError, ValueError, TypeError) as e:
36                 pass
37             except (KeyboardInterrupt) as e:
38                 exit(0)

```

Рисунок 15 — Код файла tui.py

```
1 def input_float(maximum: int,
2                 decimal_places: int,
3                 s: str=" ") -> float:
4     """ Beod float """
5
6     while True:
7         try:
8             # Считываем строку, убирая пустые символы
9             r = input(f'{s} > ').replace(' ', '')
10            f = float(r)
11            n = int(f * (10**decimal_places))
12            if not 0 < n <= maximum:
13                raise ValueError
14
15            return f
16        except (EOFError, ValueError,
17                TypeError, OverflowError) as e:
18            pass
19        except (KeyboardInterrupt) as e:
20            exit(0)
```

Рисунок 16 — Код файла tui.py

```

1 def input_date() -> dtf.date:
2     """ Ввод date """
3     # Текущая дата
4
5     while True:
6         try:
7             # Считываем строку, убирая пустые символы
8             print(f"Нажмите {Fore.CYAN}enter{CR} "\
9                   f"или введите {Fore.GREEN}дату{CR}")
10            r = input(f'{{dtf.now().strftime('%d.%m.%y')}} > ')
11            r = r.replace(' ', '')
12            if r == '':
13                # Текущее время
14                return dtf.now()
15            else:
16                # Проверяем дату
17                date = dtf.validate_point_date(r)
18                if date is not None:
19                    return date
20
21        except (EOFError, ValueError, TypeError) as e:
22            print(e)
23        except (KeyboardInterrupt) as e:
24            exit(0)
25

```

Рисунок 17 — Код файла tui.py

ПРИЛОЖЕНИЕ Д

Исходный код файла main.py

```
1 import tui
2 import data
3 import dtf
4 import helps
5
6 from colorama import Fore, Back, Style
7 CR = Style.RESET_ALL
8
9 # [DEVELOP] Настраиваем логирование
10 import logging
11 logging.basicConfig(
12     filename="log.txt", filemode='a',
13     encoding='utf-8', level=logging.DEBUG)
14 # [DEVELOP] Выключаем логирование
15 # logging.disable(level=logging.CRITICAL)
16
17
18 def state_help(*args):
19     """ Справка """
20     commands = {
21         '1': (state_main, [], 'главное меню'),
22         '9': (state_help, ["[команда]"], 'справка'),
23         '0': (state_exit, [], 'выход из приложения')
24     }
25     tui.set_commands(commands)
26     tui.draw_state("Справка")
27     if len(args) == 1:
28         if args[0] in helps.commands:
29             print(f"Информация по команде {Fore.CYAN}{args[0]}{CR}")
30             print(helps.commands[args[0]])
31         else:
32             print("Нет такой команды:", args[0])
33     else:
34         print(helps.main)
```

Рисунок 18 — Код файла main.py

```
1 def substate_qsave():
2     """ Запрашивает подтверждение сохранения """
3     tui.draw_substate("Сохранение")
4     if data.available():
5         print(f"Сохранить изменения в: {data.current_path}?")
6         if tui.input_bool():
7             data.save_file()
8
9
10 def state_exit(*args):
11     """ Выход из приложения """
12     substate_qsave()
13     exit(0)
14
15
16 def state_open_base(*args):
17     """ Выгружает данные из файла """
18     substate_qsave()
19
20     if len(args) == 0:
21         data.load_file()
22     else:
23         data.load_file(' '.join(args))
24
25     if not data.available():
26         print(f"Нет доступа к {data.current_path}")
27     return state_main, data.available()
```

Рисунок 19 — Код файла main.py

```

1 def state_save_base(*args):
2     """ Сохраняет данные в файл """
3     if len(args) == 0:
4         print("Сохранено!" if data.save_file()
5             else "Нет доступа к записи!")
6         return state_main, False
7     else:
8         print(f"Сохранить изменения в новый файл:"\
9             f"{' '}.join(args)}?")
10        if tui.input_bool():
11            print("Сохранено!" if data.save_file(' '.join(args))
12                else "Нет доступа к записи!")
13            return state_main
14
15
16 def substate_remove(*args):
17     if len(args) == 1 and args[0].isnumeric():
18         res = data.remove_product(int(args[0]))
19         print("Удалено" if res else "Не удалено")
20
21
22 def substate_inc():
23     data.sort_cost(True)
24     print("Отсортировано по возрастанию цены")
25
26
27 def substate_dec():
28     data.sort_cost(False)
29     print("Отсортировано по убыванию цены")

```

Рисунок 20 — Код файла main.py

```

1 def state_list_date(*args):
2     if len(args) == 0:
3         date = dtf.now()
4     elif len(args) == 1:
5         date = dtf.validate_point_date(args[0])
6         if date is None:
7             print("Неправильная дата!")
8             return state_list, False
9     else:
10        print("Лишние аргументы!")
11        return state_list, False
12
13    if data.empty_list_date(date):
14        print("На эту дату ничего нет!")
15        return state_list, False
16
17    tui.draw_state("Коллекция")
18    tui.draw_table_head(data.table_head, data.table_width_min())
19    for x in data.get_list_date(date):
20        tui.draw_table_row(x)
21
22
23 def state_list_type(*args):
24     if len(args) != 1:
25         print("Должен быть ровно один аргумент")
26         return state_list, False
27
28     if data.empty_list_type(args[0]):
29         print("В этой категории ничего нет!")
30         return state_list, False
31
32     tui.draw_state("Коллекция")
33     tui.draw_table_head(data.table_head, data.table_width_min())
34     for x in data.get_list_type(args[0]):
35         tui.draw_table_row(x)

```

Рисунок 21 — Код файла main.py

```
1 def state_list(clear: bool=True, *args):
2     if len(data.data) == 0:
3         print("Коллекция пуста! Добавьте туда что-нибудь")
4         return state_main, False
5
6     commands = {
7         '1': (state_main, [], 'главное меню'),
8         '4': (state_list, [], 'просмотреть всю коллекцию'),
9         '41': (state_list_date, ['[ДД.ММ.ГГ]'],
10              'просмотреть по дате'),
11         '42': (state_list_type, ['<type>'],
12              'просмотреть по категории')
13     }
14     tui.set_commands(commands)
15     if clear:
16         tui.draw_state('Коллекция')
17         tui.draw_table_head(data.table_head, data.table_width_min())
18         for x in data.get_list():
19             tui.draw_table_row(x)
```

Рисунок 22 — Код файла main.py

```

1 def state_add(*args):
2     tui.draw_substate('Добавление')
3
4     print(f"Введите {Fore.GREEN}название{CR} продукта (до "\
5           f"{Fore.CYAN}{data.PRODUCT_NAME_LEN}{CR} символов)")
6     product_name = tui.input_str(data.PRODUCT_NAME_LEN)
7
8     print(f"Введите {Fore.GREEN}стоимость{CR} продукта (до "\
9           f"{Fore.CYAN}2{CR} знаков после точки)")
10    product_cost = tui.input_float(data.PRODUCT_COST_MAX, 2)
11
12    print(f"Введите {Fore.GREEN}категорию{CR} продукта (до "\
13          f"{Fore.CYAN}{data.PRODUCT_TYPE_LEN}{CR} символов)")
14    product_type = tui.input_str(data.PRODUCT_TYPE_LEN)
15
16    product_date = tui.input_date()
17
18    tui.draw_substate('Подтверждение')
19    print(f"Сохранить продукт?")
20    print(f"Название:\t{Fore.CYAN}{product_name}{CR}")
21    print(f"Стоимость:\t{Fore.CYAN}{product_cost}{CR}")
22    print(f"Категория:\t{Fore.CYAN}{product_type}{CR}")
23    print(f"Дата:\t\t{Fore.CYAN}{dtf.display_data(product_date)}{CR}")
24
25    if tui.input_bool():
26        data.add_product(product_name, product_cost,
27                          product_type, product_date)
28    return state_main

```

Рисунок 23 — Код файла main.py

```

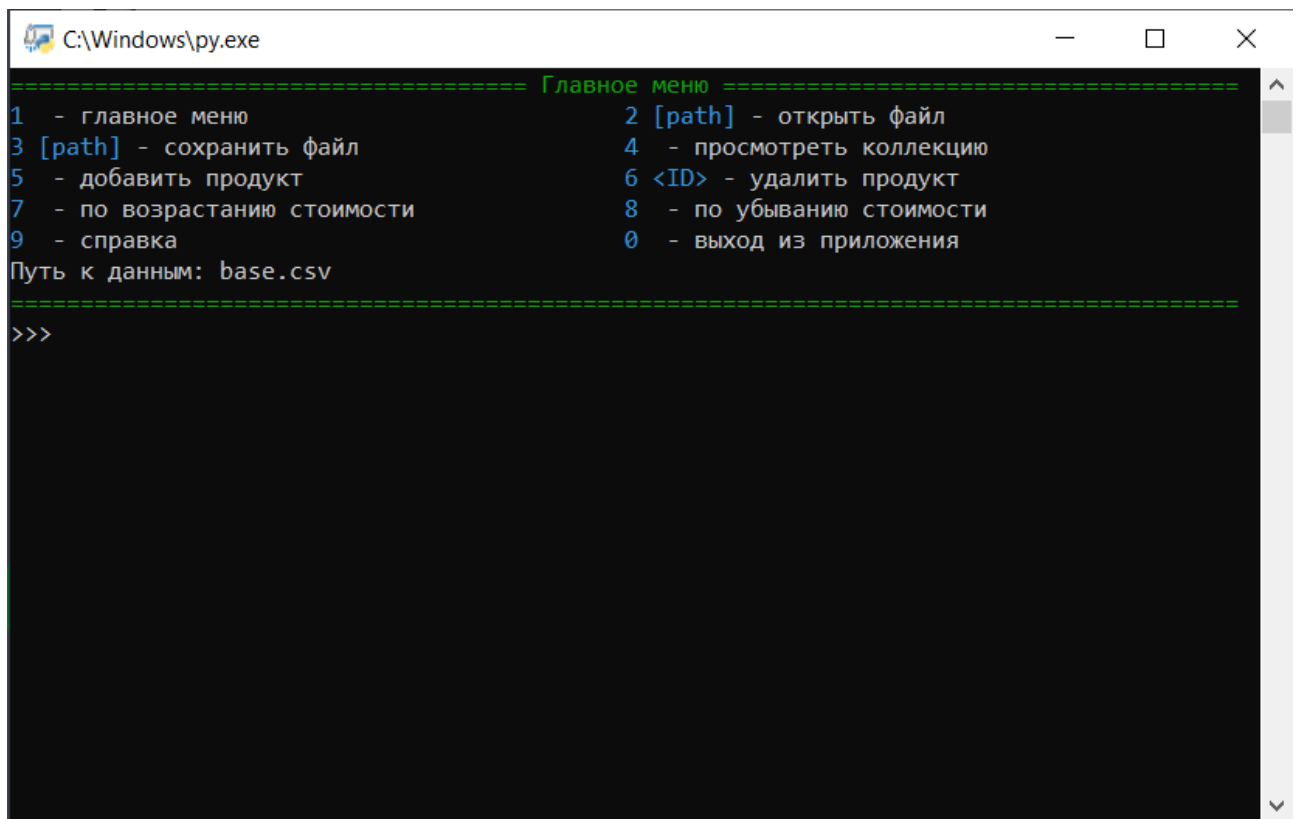
1 def state_main(clear: bool=True, *args):
2     if data.available():
3         caption = f'Путь к данным: {data.current_path}'
4         commands = {
5             '1': (state_main, [], 'главное меню'),
6             '2': (state_open_base, ['[path]'], 'открыть файл'),
7             '3': (state_save_base, ['[path]'], 'сохранить файл'),
8             '4': (state_list, [], 'просмотреть коллекцию'),
9             '5': (state_add, [], 'добавить продукт'),
10            '6': (substate_remove, ['<ID>'], 'удалить продукт'),
11            '7': (substate_inc, [], 'по возрастанию стоимости'),
12            '8': (substate_dec, [], 'по убыванию стоимости'),
13            '9': (state_help, [], 'справка'),
14            '0': (state_exit, [], 'выход из приложения')
15        }
16     else:
17         caption = "Откройте файл"
18         commands = {
19             '1': (state_main, [], 'главное меню'),
20             '2': (state_open_base, ['[path]'], 'открыть файл'),
21             '9': (state_help, [], 'справка'),
22             '0': (state_exit, [], 'выход из приложения')
23        }
24     tui.set_commands(commands)
25     if clear:
26         tui.draw_state('Главное меню', caption_down=caption)
27
28
29 if __name__ == "__main__":
30     tui.cls()
31     print(helps.main)
32     try:
33         input('Нажмите enter > ')
34     except (EOFError, KeyboardInterrupt) as e:
35         exit(0)
36     tui.run(state_main)

```

Рисунок 24 — Код файла main.py

ПРИЛОЖЕНИЕ Е

Интерфейс



```
C:\Windows\py.exe
===== Главное меню =====
1  - главное меню                2 [path] - открыть файл
3 [path] - сохранить файл        4  - просмотреть коллекцию
5  - добавить продукт           6 <ID> - удалить продукт
7  - по возрастанию стоимости   8  - по убыванию стоимости
9  - справка                    0  - выход из приложения
Путь к данным: base.csv
=====
>>>
```

Рисунок 25 — Интерфейс главного меню

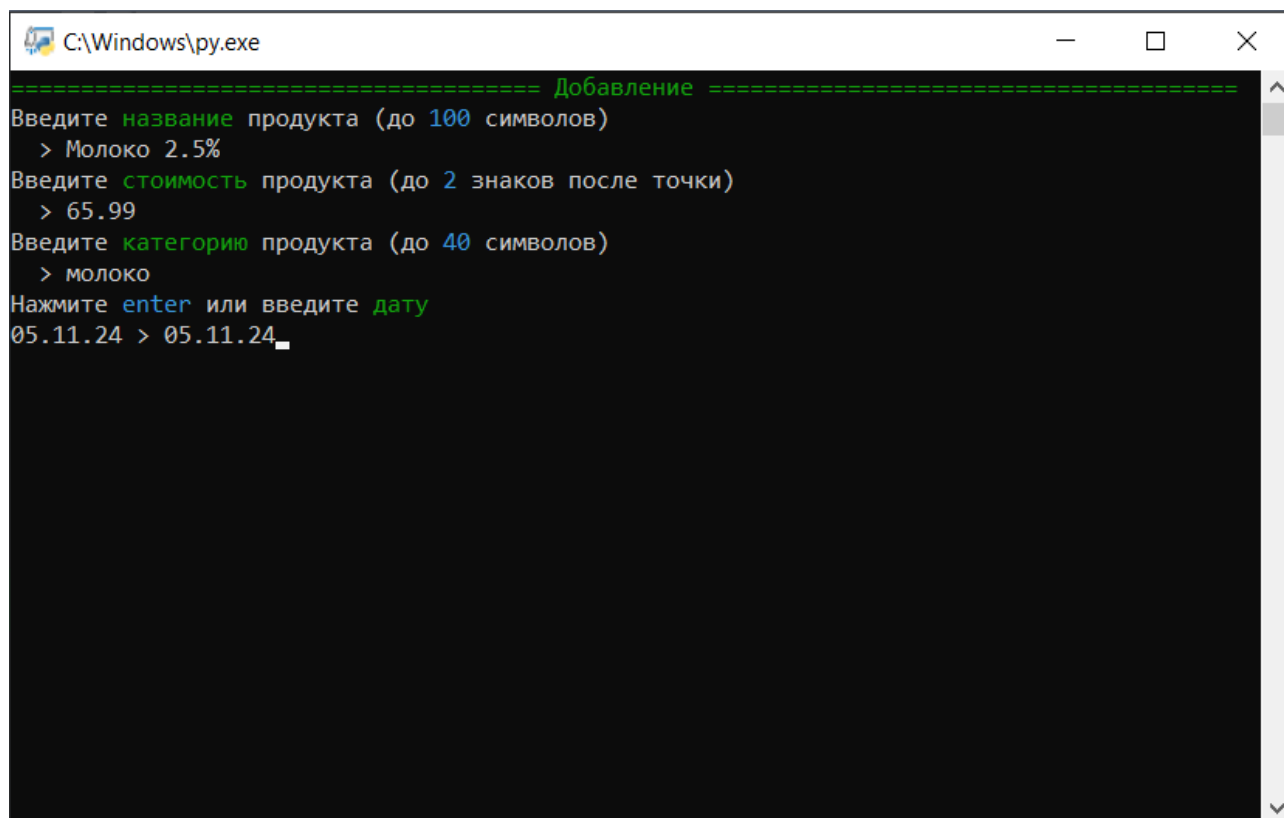


Рисунок 26 — Добавление продукта

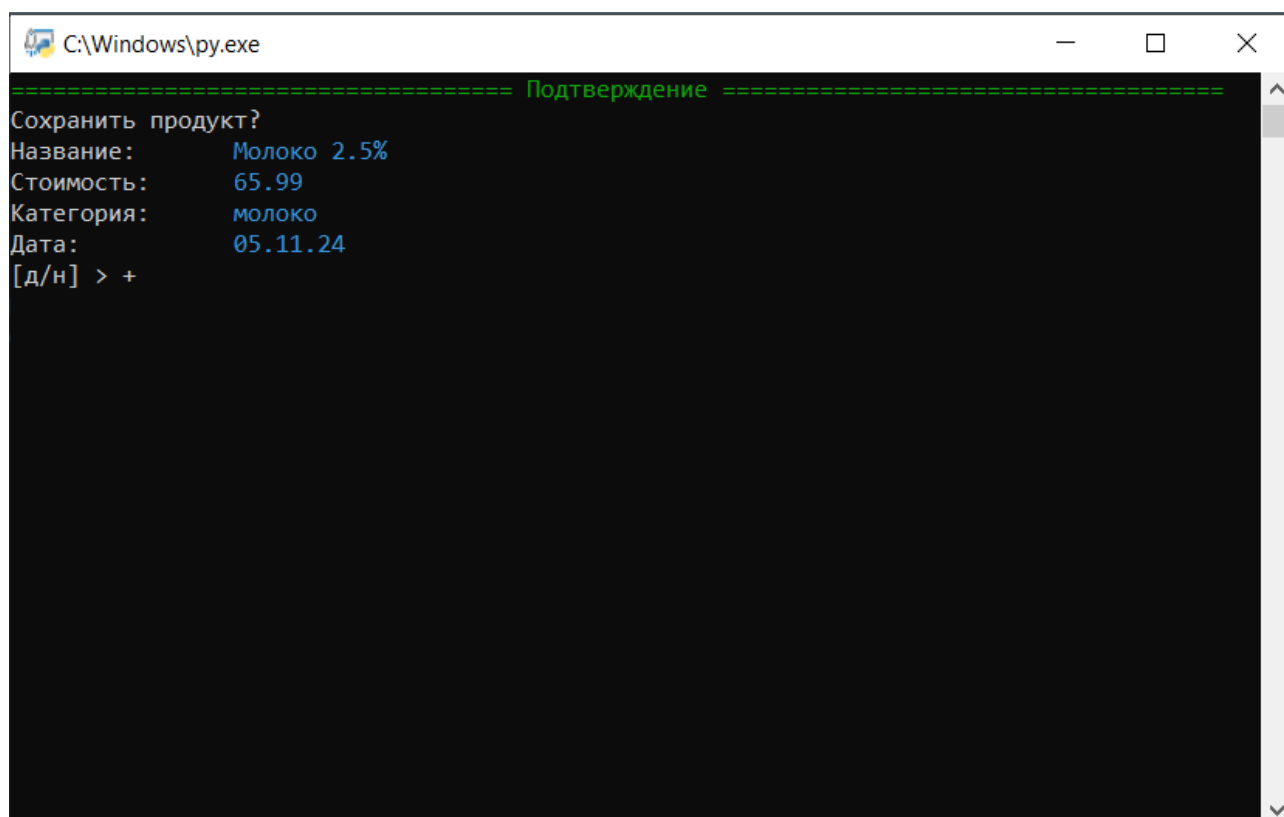


Рисунок 27 — Подтверждение добавления продукта

```
C:\Windows\py.exe

===== Коллекция =====
1 - главное меню                                4 - посмотреть всю коллекцию
41 [ДД.ММ.ГГ] - посмотреть по дате              42 <type> - посмотреть по категории
=====
ID | Название | Цена | Категория | Дата
0 | Молоко 2.5% | 65.98 | молоко | 05.11.24
1 | Молоко 3.2% | 70.65 | молоко | 05.11.24
2 | Хлеб ржаной | 45.00 | хлеб | 05.11.24
3 | Булочка с сосиской | 52.45 | хлеб | 05.11.24
4 | Хлеб пшеничный | 39.00 | хлеб | 05.11.24
5 | Колбаса | 305.64 | колбаса | 05.11.24
>>> _
```

Рисунок 28 — Просмотр продуктов

```
C:\Windows\py.exe

===== Коллекция =====
1 - главное меню                                4 - посмотреть всю коллекцию
41 [ДД.ММ.ГГ] - посмотреть по дате              42 <type> - посмотреть по категории
=====
ID | Название | Цена | Категория | Дата
2 | Хлеб ржаной | 45.00 | хлеб | 05.11.24
3 | Булочка с сосиской | 52.45 | хлеб | 05.11.24
4 | Хлеб пшеничный | 39.00 | хлеб | 05.11.24
>>> bl_
```

Рисунок 29 — Просмотр продуктов по категории