

**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** Разработка приложения “Программа для контроля  
собственных денежных средств”

**Обучающийся** Шарыпов Егор Антонович

**Факультет** факультет инфокоммуникационных технологий

**Группа** K3123

**Направление подготовки** 11.03.02 Инфокоммуникационные технологии и  
системы связи

**Образовательная программа** Программирование в  
инфокоммуникационных системах

**Обучающийся**

\_\_\_\_\_  
(дата)

\_\_\_\_\_  
(подпись)

Шарыпов Е.А.  
(Ф.И.О.)

**Руководитель**

\_\_\_\_\_  
(дата)

\_\_\_\_\_  
(подпись)

Казанова П.П.  
(Ф.И.О.)

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

**ЗАДАНИЕ НА ВЫПОЛНЕНИЕ ДОМАШНЕЙ РАБОТЫ**

**По дисциплине** Программирование  
**Обучающийся** Шарыпов Егор Антонович  
**Факультет** факультет инфокоммуникационных технологий  
**Группа** К3123  
**Направление подготовки** 11.03.02 Инфокоммуникационные технологии и системы связи  
**Образовательная программа** Программирование в инфокоммуникационных системах  
**Тема домашней работы** Разработка приложения “Программа для контроля собственных денежных средств”  
**Руководитель домашней работы** Казанова Полина Петровна, Университет ИТМО, факультет инфокоммуникационных технологий, тьютор.  
**Основные вопросы, подлежащие разработке** В рамках домашней работы требуется разработать консольное приложение “Программа для контроля собственных денежных средств”

**Дата выдачи задания:** 01.09.2023

**Срок предоставления готовой курсовой работы:** 30.09.2023

<b>Руководитель</b>	_____	_____	<b>Казанова П.П.</b>
	(дата)	(подпись)	(Ф.И.О.)
<b>Задание принял к исполнению</b>	_____	_____	<b>Шарыпов Е.А.</b>
	(дата)	(подпись)	(Ф.И.О.)

# СОДЕРЖАНИЕ

Стр.

<b>1</b>	<b>Анализ предметной области и требований .....</b>	<b>4</b>
<b>2</b>	<b>Разработка приложения .....</b>	<b>5</b>
2.1	Проектирование интерфейса .....	5
2.2	Разработка общей структуры кода приложения .....	5
2.3	Разработка классов данных .....	7
2.4	Разработка пользовательского интерфейса .....	8
2.4.1	Разработка системы команд .....	9
2.4.2	Разработка функций очистки данных .....	10
2.4.3	Разработка системы вопросов к пользователю .....	11
2.4.4	Разработка команды 'новая покупка' .....	11
2.4.5	Разработка команды 'редактировать покупку' .....	12
2.4.6	Разработка команды 'удалить покупку' .....	13
2.4.7	Разработка команд 'вывести последние покупки', 'вывести дорогие покупки', 'вывести дешевые покупки' ..	13
2.4.8	Разработка команд 'группировать по категории', 'группировать по дате' .....	15
2.4.9	Разработка команды 'справка' .....	16
	<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>17</b>
	<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....</b>	<b>18</b>

## 1 Анализ предметной области и требований

Цель работы — создать приложение для контроля собственных денежных средств.

Задачи:

1. реализовать функцию добавления покупок в коллекцию и сохранения их в файл
2. реализовать функцию просмотра всех покупок
3. реализовать функцию просматривать покупки по дате и категории
4. реализовать функцию распределения покупок по стоимости
5. реализовать функции удаления покупок и выхода из приложения

По своей сути разрабатываемое приложение - простое консольное приложение, и для его разработки лучше всего подойдет язык программирования Python по следующим причинам:

- его простота
- тот факт, что его преподают на курсе "Программирование"

Также было принято решение использовать git как систему контроля версий.

## **2 Разработка приложения**

### **2.1 Проектирование интерфейса**

Было принято решение сделать консольный интерфейс приложения, похожий на интерфейс утилиты GNU Parted. Такой интерфейс подразумевает интерактивную оболочку, в которую пользователь вводит однобуквенные команды. Данные для команд могут передаваться как с помощью аргументов, так и с помощью ответов на вопросы. Был сформирован следующий список команд для реализации:

- н — новая покупка
- р — редактировать покупку
- у — удалить покупку
- п — вывести последние покупки
- ц, Ц — вывести покупки отсортированные по цене
- д — вывести покупки по датам
- к — вывести покупки по категориям
- в — выйти из приложения
- с — вывести справку

### **2.2 Разработка общей структуры кода приложения**

Было решено, что код приложения будет условно разбит на три части: часть, которая работает с данными, пользовательский интерфейс и входной файл, который выступает связующим звеном между другими частями. В общем алгоритм работы приложения может быть представлен следующей блок-схемой (рисунок 2.1):

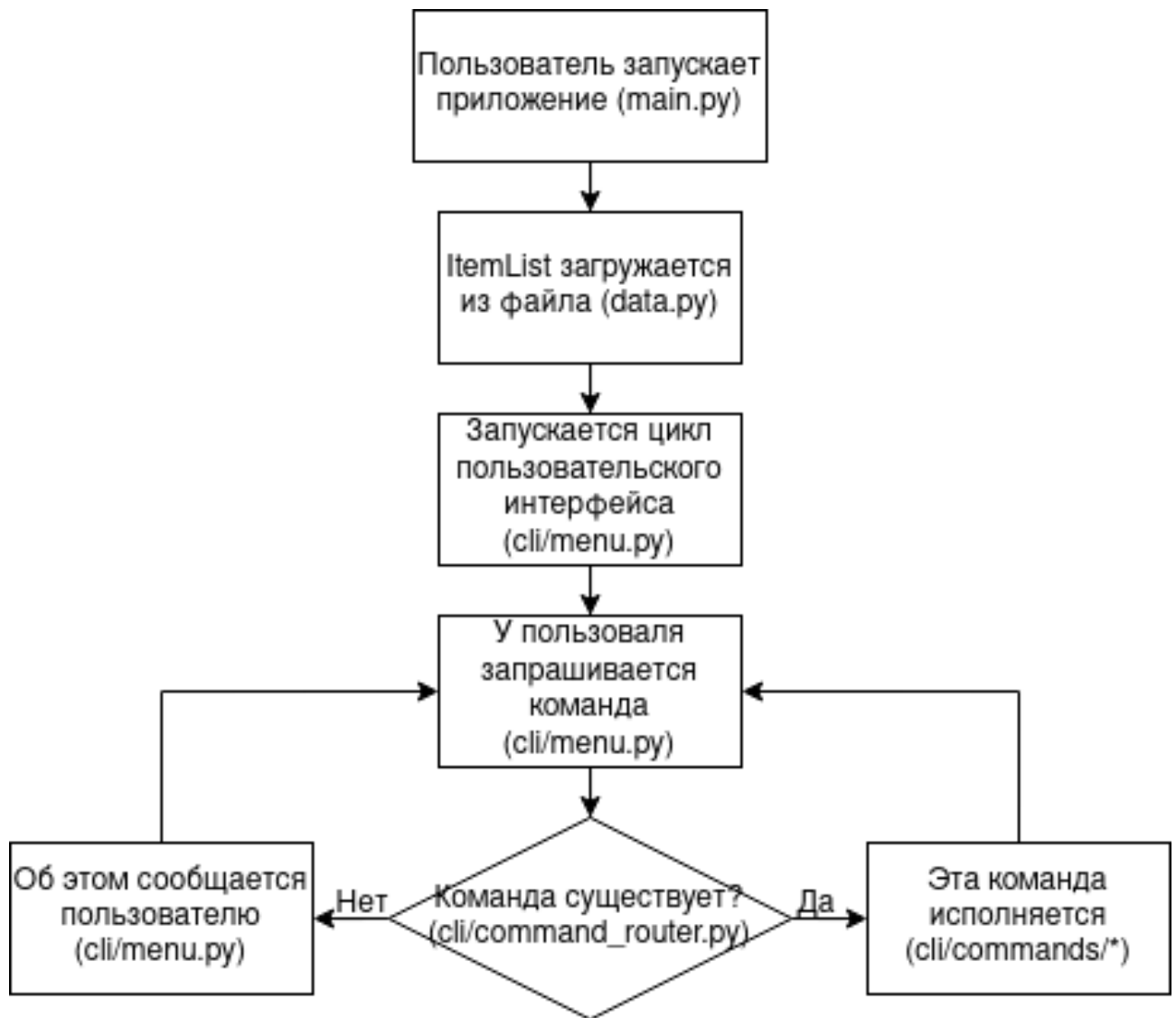


Рисунок 2.1 — Алгоритм работы приложения

Входной файл называется main.py, его код приведен ниже (рисунок 2.2):

```

1 from data import FileStorage, ItemList
2 from cli.menu import start_cli
3 import os
4 import platform
5
6 if __name__ == '__main__':
7     file_name = 'pokupki.txt'
8     file_path = f"{os.getenv('HOME')}/{file_name}" \
9         if platform.system() == 'Linux' \
10         else f"{os.getenv('APPDATA')}\\{file_name}"
11
12     storage = FileStorage(file_path)
13     item_list = ItemList.load(storage)
14
15     start_cli(item_list)

```

Рисунок 2.2 — Исходный код main.py

Этот код определяет путь до файла с покупками в зависимости от ОС, инициализирует ItemList и запускает пользовательский интерфейс.

## 2.3 Разработка классов данных

Для хранения и обработки данных было создано 3 класса — Item, ItemList и FileStorage, — которые были помещены в файл data.py.

Класс Item представляет из себя класс покупок. У него есть такие поля, как name (строка), id (целое число), category (строка), date (Datetime), cost (целое число). В нем реализованы методы для перевода данных в строку и обратно для хранения в файле, а также метод to\_str для генерации строкового представления покупки для пользовательского интерфейса.

Класс ItemList представляет из себя класс коллекций покупок. Внутренне хранит покупки в списке. Этот класс ответственен за назначение id покупкам внутри него. Для создания списка требуется указать некоторое хранилище (storage), с данными из которого будет автоматически синхронизываться

содержание списка, где хранилище — это что угодно, что может хранить строку. У `ItemList` есть следующие методы для внешнего использования (таблица 2.1):

Таблица 2.1 — Методы `ItemList`

Метод	Описание
<code>static load(storage)</code>	Загрузить данные из хранилища. Хранилище при этом — любой объект с методом <code>read</code> , который читает строку, и методом <code>write</code> , который записывает строку.
<code>get_items()</code>	Возвращает покупки в виде питоновского списка.
<code>add(new_item)</code>	Добавляет новую покупку в список, автоматически назначая ему <code>id</code> .
<code>get_categories()</code>	Возвращает множество категорий списка.
<code>get_by_id(id)</code>	Пытается найти и вернуть покупку с заданным <code>id</code> . В случае неудачи создает <code>KeyError</code> .
<code>delete(id)</code>	Пытается найти и удалить покупку с заданным <code>id</code> . В случае неудачи создает <code>KeyError</code> .

Класс `FileStorage` — файловое хранилище, используемое в коде для хранения в нем `ItemList`. Для создания такого хранилища нужно указать только строку — путь до файла.

## 2.4 Разработка пользовательского интерфейса

Пользовательский интерфейс — самая обширная часть кодовой базы, и весь код, связанный с ним, находится в папке `cli/`. Входная точка пользовательского интерфейса — файл `cli/menu.py`. Ниже приведен его код (рисунок 2.3):



```

1 from cli.command_router import process_command, UnknownCommandException
2
3 def start_cli(item_list):
4     while True:
5         command_text = input('Введите команду(с - справка): ')
6         if command_text == '':
7             continue
8
9         try:
10            process_command(command_text, item_list)
11        except UnknownCommandException:
12            print(f"Некорректная команда {command_text}!")
13        except Exception:
14            print(f"Неизвестная ошибка")

```

Рисунок 2.3 — Пример работы команды ‘редактировать покупку’

Функция `start_cli(item_list)` вызывается из `main.py` и в бесконечном цикле предлагает пользователю ввести команду, обработка которой передается в модуль `cli/command_router.py`.

### 2.4.1 Разработка системы команд

Было принято решение разработать систему команд для более удобной организации кодовой базы. Команда — это объект класса `Command` из `cli/command.py`. У любой команды есть: поле `name` (строка) — то, что нужно набрать для выполнения команды, поле `help` (строка) — то, что выводится в справке о команде, метод `run(args, item_list)` — метод, который вызывается при исполнении команды, где `args` — это список строк, аргументов команды, а `item_list` — объект класса `ItemList`. Для файлов с командами была создана папка `cli/commands/`. Ниже приведен простейший пример команды (рисунок 2.4):

```

1 from cli.command import Command
2
3
4 def run_exit(args, item_list):
5     quit()
6
7
8 exit_command = Command('в', 'в - выход из приложения', run_exit)

```

Рисунок 2.4 — Исходный код cli/commands/exit.py

Код в файле cli/command\_router.py ответственен за выбор нужной команды в зависимости от того, что ввел пользователь. Он импортирует все команды и составляет из них список commands. Ниже представлен код функции process\_command из этого файла (рисунок 2.5):

```

def process_command(command_text, item_list):
    command_name, *args = command_text.split(' ')
    try:
        command = next(cmd for cmd in commands if cmd.name == command_name)
        command.run(args, item_list)
    except StopIteration:
        raise UnknownCommandException

```

Рисунок 2.5 — Исходный код функции process\_command из файла cli/command\_router.py

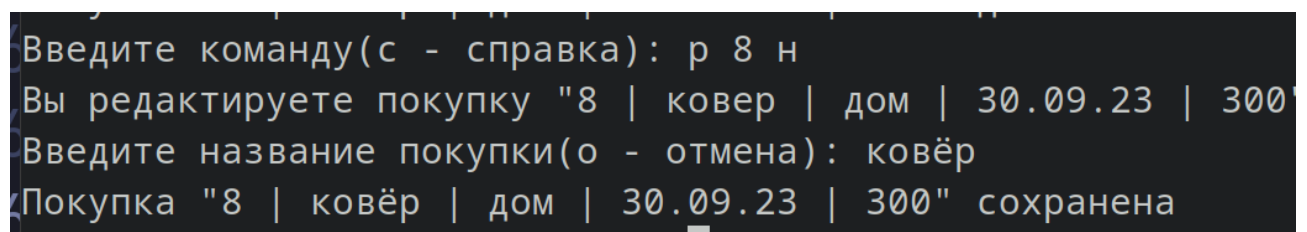
## 2.4.2 Разработка функций очистки данных

Исполнение некоторых команд требует очистки данных, введенных пользователем. Для такой очистки были разработаны специальные функции sanitize\_int, sanitize\_date, sanitize\_str, sanitize\_bool, которые были помещены в файл cli/sanitize.py. Они берут в качестве аргумента ввод пользователя и пытаются привести его к нужному формату. В случае некорректных данных эти функции создают ValueError. Функция sanitize\_int переводит строку в число, убеждаясь в том, что это число целое и положительное. Функция sanitize\_str удаляет символы '\t' и

'|', так как эти символы используются как разделители при строковом отображении покупок. Функция `sanitize_bool` переводит слова 'да' и 'нет' в `True` и `False`. Функция `sanitize_date` генерирует объекты типа `datetime` из пользовательского ввода, при этом даты 29.09.2023, 29.09.23 и 29.09 эквивалентны.

### 2.4.3 Разработка системы вопросов к пользователю

Так как при редактировании и при создании покупки пользователю задаются одинаковые вопросы, было принято решение вынести эту логику в отдельный файл `cli/questions.py`. Этот файл содержит такие функции как `ask_name`, `ask_category`, `ask_date`, `ask_cost`. При этом если пользователь отвечает на любой вопрос буквой 'о', это означает отмену действия, и в таком случае из функций возвращается `None`. В случае ввода некорректных данных приложение сообщит об этом и задаст вопрос повторно. Рассмотрим подробнее функцию `ask_category`. Ее особенность заключается в том, что если пользователь вводит категорию, которая раньше не встречалась, то приложение уточнит, не опечатка ли это, и укажет уже существующие категории. Если пользователь ответит, что хочет создать новую категорию, то `ask_category` вернет новую категорию. Пример работы `ask_category` можно увидеть на скриншоте ниже (рисунок 2.6):



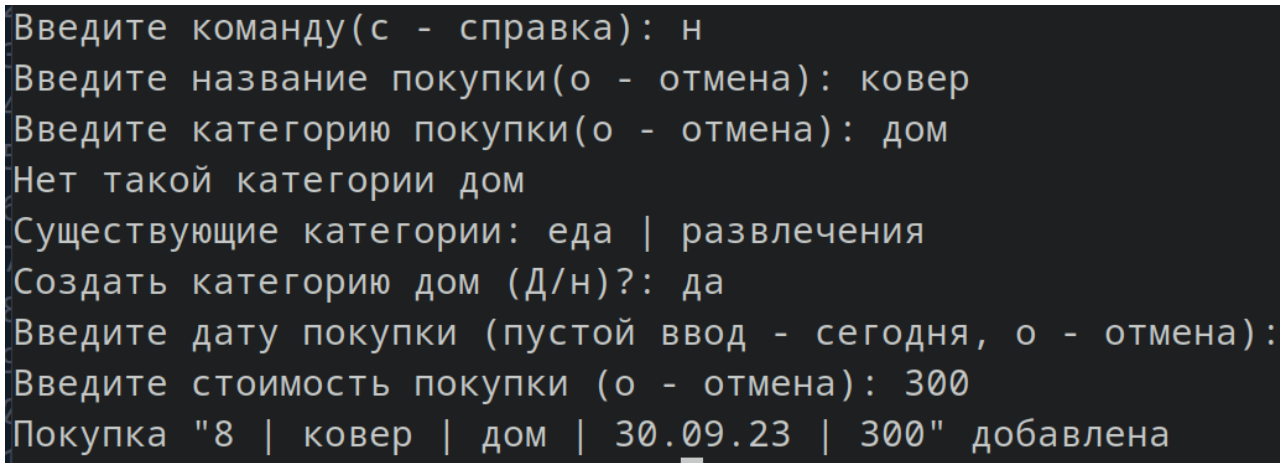
```
Введите команду(с - справка): р 8 н
Вы редактируете покупку "8 | ковер | дом | 30.09.23 | 300"
Введите название покупки(о - отмена): ковёр
Покупка "8 | ковёр | дом | 30.09.23 | 300" сохранена
```

Рисунок 2.6 — Пример работы команды 'редактировать покупку'

### 2.4.4 Разработка команды 'новая покупка'

Для реализации функции добавления покупок была разработана команда "новая покупка" код которой лежит в файле `cli/commands/create.py`.

Эта команда не принимает никаких аргументов, последовательно задает вопросы о названии, категории, дате и цене покупки с помощью модуля `cli/questions.py`, создает новую покупку и добавляет ее в список. Пример работы этой команды можно увидеть на скриншоте ниже (рисунок 2.7):



```
Введите команду(с - справка): н
Введите название покупки(о - отмена): ковер
Введите категорию покупки(о - отмена): дом
Нет такой категории дом
Существующие категории: еда | развлечения
Создать категорию дом (Д/н)? : да
Введите дату покупки (пустой ввод - сегодня, о - отмена):
Введите стоимость покупки (о - отмена): 300
Покупка "8 | ковер | дом | 30.09.23 | 300" добавлена
```

Рисунок 2.7 — Пример работы команды ‘новая покупка’

#### 2.4.5 Разработка команды ‘редактировать покупку’

Для реализации функции редактирования покупок была разработана команда "редактировать покупку" код которой лежит в файле `cli/commands/modify.py`. Эта команда принимает такие опциональные аргументы как `id` покупки, поле для редактирования и новое значение. В случае, если не все аргументы заданы, команда задаст пользователю необходимые вопросы. Информация о поле для редактирования кодируется буквами н, к, д, ц для названия, категории, даты и цены соответственно. Для третьего вопроса о новом значении используются вопросы из модуля `cli/questions.py`. После получения всех необходимых данных команда редактирует нужную покупку и завершает работу. Пример работы этой команды можно увидеть на скриншоте ниже (рисунок 2.8):

```
Введите команду(с - справка): р 8 н
Вы редактируете покупку "8 | ковер | дом | 30.09.23 | 300"
Введите название покупки(о - отмена): ковёр
Покупка "8 | ковёр | дом | 30.09.23 | 300" сохранена
```

Рисунок 2.8 — Пример работы команды ‘редактировать покупку’

#### 2.4.6 Разработка команды ‘удалить покупку’

Для реализации функции удаления покупок была разработана команда ‘удалить покупку’, код которой лежит в файле `cli/commands/delete.py`. Для удаления нужно ввести `id` покупки, которую требуется удалить, как аргумент команды либо как ответ на вопрос команды. После того, как покупка найдена, команда просит пользователя подтвердить удаление. В случае подтверждения команда удаляет покупку. Пример работы этой команды можно увидеть на скриншоте ниже (рисунок 2.9):

```
Введите команду(с - справка): у
Введите ID покупки, которую хотите удалить(о - отмена): 1
Вы точно хотите удалить покупку "1 | бипки) | еда | 21.09.23 | 10" (Д/н)?: да
Покупка "1 | бипки) | еда | 21.09.23 | 10" удалена
```

Рисунок 2.9 — Пример работы команды ‘редактировать покупку’

#### 2.4.7 Разработка команд ‘вывести последние покупки’, ‘вывести дорогие покупки’, ‘вывести дешевые покупки’

Для реализации функции просмотра последних, дешевых, дорогих покупок были разработаны команды ‘вывести последние покупки’, ‘вывести дорогие покупки’, ‘вывести дешевые покупки’, код которых лежит в файлах `cli/commands/recent.py` и `cli/commands/cost.py`. Эти команды работают схожим образом, поэтому при описании они выделены в одну подсекцию. В качестве опционального аргумента эти команды принимают на вход число покупок, которые будут отображены. Значение по умолчанию для этого

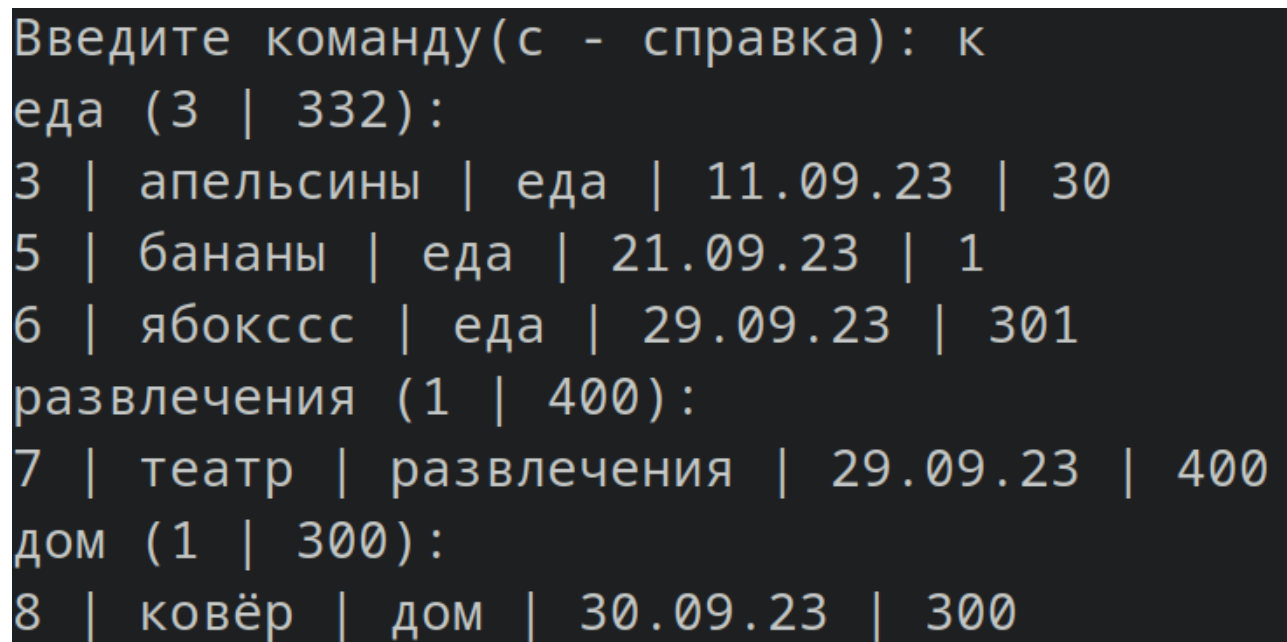
параметра равно 10. Ниже приведен исходный код для команд 'вывести дорогие покупки' и 'вывести дешевые покупки' (рисунок 2.10):

```
1 from cli.command import Command
2 from cli.sanitize import sanitize_int
3
4 n_default = 10
5 def run_cost_command(args, item_list, descending=True):
6     try:
7         n = sanitize_int(args[0])
8     except IndexError:
9         n = n_default
10    except ValueError:
11        print('N должно быть целым положительным числом')
12        return
13
14    for item in sorted(
15        item_list.get_items(),
16        key=lambda item: item.cost,
17        reverse=descending)[:n]:
18
19        print(item.to_str())
20
21
22 cost_desc_command = Command(
23     'ц',
24     f"ц [N = {n_default}] - выводит N самых дорогих покупок",
25     lambda args, item_list: run_cost_command(args, item_list, True)
26 )
27
28 cost_asc_command = Command(
29     'ц',
30     f"ц [N = {n_default}] - выводит N самых дешевых покупок",
31     lambda args, item_list: run_cost_command(args, item_list, False)
32 )
```

Рисунок 2.10 — Исходный код cli/commands/cost.py

#### 2.4.8 Разработка команд ‘группировать по категории’, ‘группировать по дате’

Для реализации функции просмотра покупок, сгруппированных по группам и по датам были разработаны команды ‘группировать по категории’и ‘группировать по дате’, код которых лежит в файлах `cli/commands/categories.py` и `cli/commands/date.py`. Эти команды работают схожим образом, поэтому при описании они выделены в одну подсекцию. В качестве опционального аргумента эти команды принимают на категорию или дату, покупки из которой нужно вывести. Если этот аргумент не задан, тогда команды выводят все возможные категории или даты. Также для каждой категории или даты в скобках после названия выводится количество покупок и суммарная стоимость покупок. Ниже приведен скриншот примера работы команды ‘группировать по категории’(рисунок 2.11):

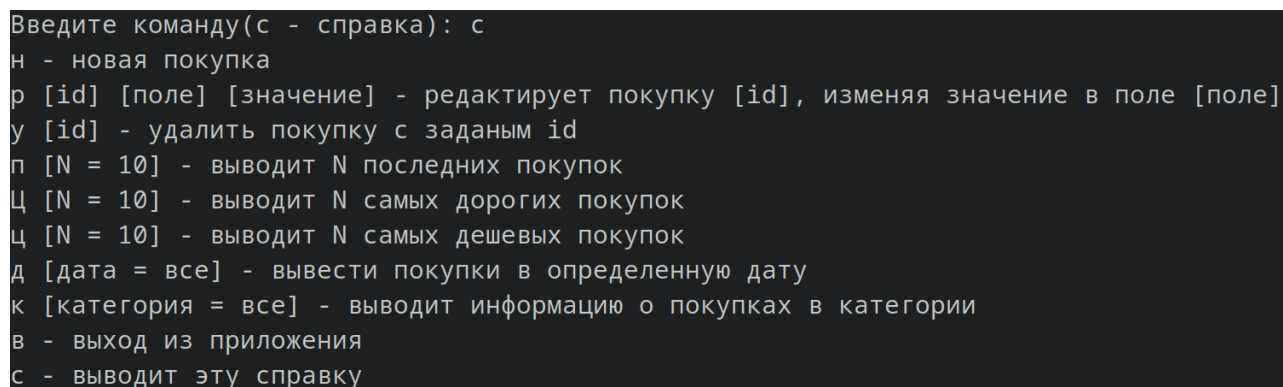


```
Введите команду(с - справка): к
еда (3 | 332):
3 | апельсины | еда | 11.09.23 | 30
5 | бананы | еда | 21.09.23 | 1
6 | ябокссс | еда | 29.09.23 | 301
развлечения (1 | 400):
7 | театр | развлечения | 29.09.23 | 400
дом (1 | 300):
8 | ковёр | дом | 30.09.23 | 300
```

Рисунок 2.11 — Пример работы команды ‘сгруппировать по дате’

### 2.4.9 Разработка команды ‘справка’

Для того, чтобы пользователь мог ознакомиться с функционалом приложения было решено создать команду ‘справка’. Код этой команды лежит в `cli/command_router.py`. Чтобы пользователь мог воспользоваться справкой при первом использовании приложения, в приглашении главного меню указано, что команда ‘с’ вызывает справку. Для генерации справки эта команда итерирует по всем командам и для каждой команды выводит сообщение для справки, указанное при создании этих команд. Выводимая информация представлена на скриншоте ниже (рисунок 2.12):

A screenshot of a terminal window showing the output of the 'с' (help) command. The text is as follows:

```
Введите команду(с - справка): с
н - новая покупка
р [id] [поле] [значение] - редактирует покупку [id], изменяя значение в поле [поле]
у [id] - удалить покупку с заданным id
п [N = 10] - выводит N последних покупок
Ц [N = 10] - выводит N самых дорогих покупок
ц [N = 10] - выводит N самых дешевых покупок
д [дата = все] - вывести покупки в определенную дату
к [категория = все] - выводит информацию о покупках в категории
в - выход из приложения
с - выводит эту справку
```

Рисунок 2.12 — Выводимая информация команды ‘справка’



## ЗАКЛЮЧЕНИЕ

В ходе этой работы получилось выполнить все поставленные задачи и выполнить цель работы. Было создано приложение для контроля собственных денежных средств, его исходный код можно скачать из репозитория [1]. В процессе разработки были использованы такие механизмы языка Python как классы, функции, списки, условия, модуль datetime.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Исходный код приложения: [Электронный ресурс]. URL: <https://github.com/koldun256/pokupki> (Дата обращения 30.09.2023).
2. Форум для поиска ответов на технические вопросы: [Электронный ресурс]. URL: <https://stackoverflow.com> (Дата обращения 30.09.2023).
3. Документация к языку программирования Python: [Электронный ресурс]. URL: <https://docs.python.org/3/> (Дата обращения 30.09.2023).