Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №10
дисциплины
«Объектно-ориентированное программирование»
Вариант № 18

Выполнил:
Текеева Мадина Азрет-Алиевна
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии Воронкин Р.А

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Разработка приложений с интерфейсом командной строки (CLI) в Python3.

Цель работы: приобретение навыков построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Репозиторий: https://github.com/bickrosss/TM_oop_lab10

Порядок выполнения работы:

1. Пример использования getopt.

```
import getopt
import sys

full_cmd_arguments = sys.argv
argument_list = full_cmd_arguments[1:]

short_options = "ho:v"
long_options = ["help", "output=", "verbose"]

try:
    arguments, values = getopt.getopt(argument_list, short_options, long_options)
except getopt.error as err:
    print(str(err))
    sys.exit(2)

for current_argument, current_value in arguments:
    if current_argument in ("-v", "--verbose"):
        print("Enabling verbose mode")
    elif current_argument in ("-h", "--help"):
        print("Displaying help")
    elif current_argument in ("-o", "--output"):
        print(f"Enabling special output mode ({current_value})")
```

2. Простой пример argparse.

```
import argparse

parser = argparse.ArgumentParser()
parser.parse_args()
```

3. Позиционные аргументы.

```
import argparse

parser = argparse.ArgumentParser()
```

```
parser.add_argument("echo", help="echo the string you use here")

args = parser.parse_args()
print(args.echo)
```

4. Возведение в квадрат с помощью argparse.

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("square", help="display a square of a given number",
type=int)

args = parser.parse_args()
print(args.square**2)
```

5. Опциональные аргументы.

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("--verbose", help="increase output verbosity",
action="store_true")

args = parser.parse_args()
if args.verbose:
    print("verbosity turned on")
```

6. Квадрат с уровнем детализации.

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("square", type=int, help="display a square of a given
number")
parser.add_argument("-v", "--verbose", action="store_true", help="increase output
verbosity")

args = parser.parse_args()
answer = args.square ** 2
if args.verbose:
    print(f"the square of {args.square} equals {answer}")
else:
    print(answer)
```

7. Аргументы с выбором из списка.

```
import argparse

parser = argparse.ArgumentParser()
```

```
parser.add_argument("square", type=int, help="display a square of a given
number")
parser.add_argument("-v", "--verbosity", type=int, choices=[0, 1, 2],
help="increase output verbosity")

args = parser.parse_args()
answer = args.square ** 2
if args.verbosity == 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity == 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

8. Счетчик аргументов.

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("square", type=int, help="display a square of a given
number")
parser.add_argument("-v", "--verbosity", action="count", default=0,
help="increase output verbosity")

args = parser.parse_args()
answer = args.square ** 2
if args.verbosity >= 2:
    print(f"the square of {args.square} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.square}^2 == {answer}")
else:
    print(answer)
```

9. Возведение в степень.

```
import argparse

parser = argparse.ArgumentParser()
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")
parser.add_argument("-v", "--verbosity", action="count", default=0)

args = parser.parse_args()
answer = args.x ** args.y
if args.verbosity >= 2:
    print(f"{args.x} to the power {args.y} equals {answer}")
elif args.verbosity >= 1:
    print(f"{args.x}^{args.y} == {answer}")
else:
    print(answer)
```

10. Взаимоисключающие аргументы.

```
import argparse

parser = argparse.ArgumentParser(description="calculate X to the power of Y")
group = parser.add_mutually_exclusive_group()
group.add_argument("-v", "--verbose", action="store_true")
group.add_argument("-q", "--quiet", action="store_true")
parser.add_argument("x", type=int, help="the base")
parser.add_argument("y", type=int, help="the exponent")

args = parser.parse_args()
answer = args.x ** args.y

if args.quiet:
    print(answer)
elif args.verbose:
    print(f"{args.x} to the power {args.y} equals {answer}")
else:
    print(f"{args.x}^{args.y} == {answer}")
```

11. Пример с субпарсерами.

```
import argparse

parser = argparse.ArgumentParser()
subparsers = parser.add_subparsers(help='List of commands')

list_parser = subparsers.add_parser('list', help='List contents')
list_parser.add_argument('dirname', action='store', help='Directory to list')

create_parser = subparsers.add_parser('create', help='Create a directory')
create_parser.add_argument('dirname', action='store', help='New directory to
create')
create_parser.add_argument('--read-only', default=False, action='store_true',
help='Set permissions to prevent writing to the directory')
```

12. Родительские парсеры.

```
import argparse

parent_parser = argparse.ArgumentParser(add_help=False)
parent_parser.add_argument('--user', action='store')
parent_parser.add_argument('--password', action='store')

child_parser = argparse.ArgumentParser(parents=[parent_parser])
child_parser.add_argument('--show_all', action='store_true')

args = child_parser.parse_args()
```

```
print(args)
```

13. Выполнили задание 1.

Во всех предложенных задачах требуется разработать полноценное консольное приложение (CLI), предназначенное для управления коллекцией однотипных объектов. Каждая программа должна быть реализована на Python и использовать декорограф @dataclass для описания сущности, которая выступает основной единицей данных (книга, студент, сотрудник, заказ, рецепт, автомобиль и т. д.). Все такие объекты должны храниться внутри отдельного датакласса-контейнера, содержащего список элементов и методы для добавления записей, отображения содержимого, поиска, сортировки или фильтрации по заданным критериям.

Интерфейс командной строки необходимо построить с помощью модуля argparse, используя подкоманды для реализации различных операций: добавления новых записей, вывода списка, выполнения выборки, загрузки данных из файла и сохранения данных в файл. Каждая команда должна принимать необходимые параметры через флага(-n – name, --year, --category и др.), а также обеспечивать корректную обработку ошибок и вывод помощи (--help).

Для задач, предполагающих работу с JSON-файлами, требуется использовать модуль json. Чтение данных должно приводить к созданию объектов соответствующих датаклассов, а сохранение – к сериализации списка объектов в корректный JSON-формат. Аналогично, для задач, где требуется использовать XML, необходимо применять стандартный модуль xml.etree.ElementTree. Во всех решениях важно соблюдать единую архитектуру:

– датчаклассы описывают структуру данных;

– контейнер управляет коллекцией;

– CLI обеспечивает взаимодействие с пользователем;

– функции загрузки/сохранения отдельно от логики фильтрации и вывода.

Каждая программа должна демонстрировать добавление объектов, отображение текущего списка, выборку по заданному условию и сохранение/загрузку данных, обеспечивая цельный рабочий цикл. Задачи являются практическими и направлены на формирование навыков структурирования данных, разработки консольных интерфейсов и работы с файлами в формате JSON или XML.

Вариант 18: Учёт питомцев

Питомец: кличка, вид, возраст. CLI: добавление, вывод, выбор по виду животного, сохранение JSON.

Ядро системы:

```
# -*- coding: utf-8 -*-
"""
Ядро приложения "Учёт питомцев".
Содержит классы данных и логику работы с коллекцией питомцев.
"""

import json
from dataclasses import dataclass, asdict, field
from typing import List, Optional
from pathlib import Path
from enum import Enum


class Species(Enum):
    """Перечисление видов животных."""
    CAT = "кот"
    DOG = "собака"
    BIRD = "птица"
    FISH = "рыба"
    RODENT = "грызун"
    OTHER = "другое"

    def __str__(self):
        return self.value


@dataclass
class Pet:
    """Класс для представления питомца."""
    name: str
    species: Species
    age: int
```

```python
    def __post_init__(self):
        """Валидация данных после инициализации."""
        if not self.name.strip():
            raise ValueError("Имя питомца не может быть пустым")
        if self.age < 0:
            raise ValueError("Возраст не может быть отрицательным")
        if self.age > 100:
            raise ValueError("Возраст слишком большой")


@dataclass
class PetContainer:
    """Контейнер для управления коллекцией питомцев."""
    pets: List[Pet] = field(default_factory=list)

    def add_pet(self, name: str, species_str: str, age: int) -> Pet:
        """
        Добавить нового питомца.

        Args:
            name: Кличка питомца
            species_str: Вид животного
            age: Возраст в годах

        Returns:
            Pet: Созданный объект питомца

        Raises:
            ValueError: При некорректных данных
        """
        try:
            # Конвертируем строку в Species Enum
            species = Species[species_str.upper().replace(" ", "_")]
        except KeyError:
            # Если вид не найден в Enum, используем OTHER
            species = Species.OTHER

        pet = Pet(name=name, species=species, age=age)
        self.pets.append(pet)
        return pet

    def show_all(self) -> str:
        """
        Получить форматированное представление всех питомцев.

        Returns:
            str: Отформатированная строка со списком питомцев
        """
        if not self.pets:
            return "Список питомцев пуст."
```

```
        result = ["\nСписок всех питомцев:", "-" * 40]
        for i, pet in enumerate(self.pets, 1):
            result.append(f"{i}. {pet.name} - {pet.species}, {pet.age} лет")
        result.extend(["-" * 40, f"Всего: {len(self.pets)} питомцев"])
        return "\n".join(result)

    def find_by_species(self, species_str: str) -> List[Pet]:
        """
        Найти питомцев по виду.

        Args:
            species_str: Вид животного для поиска

        Returns:
            List[Pet]: Список найденных питомцев
        """
        try:
            species = Species[species_str.upper().replace(" ", "_")]
        except KeyError:
            # Если вид не найден, возвращаем пустой список
            return []

        return [pet for pet in self.pets if pet.species == species]

    def sort_by_age(self, reverse: bool = False) -> None:
        """Сортировать питомцев по возрасту."""
        self.pets.sort(key=lambda pet: pet.age, reverse=reverse)

    def sort_by_name(self) -> None:
        """Сортировать питомцев по имени."""
        self.pets.sort(key=lambda pet: pet.name)

    def save_to_file(self, filename: str) -> None:
        """
        Сохранить данные в JSON файл.

        Args:
            filename: Имя файла для сохранения

        Raises:
            IOError: При ошибке записи в файл
        """
        # Конвертируем Enum в строки для сериализации
        data = []
        for pet in self.pets:
            pet_dict = asdict(pet)
            pet_dict['species'] = pet.species.name  # Сохраняем имя Enum
            data.append(pet_dict)

        with open(filename, 'w', encoding='utf-8') as f:
            json.dump(data, f, ensure_ascii=False, indent=2)
```

```python
def load_from_file(self, filename: str) -> None:
    """
    Загрузить данные из JSON файла.

    Args:
        filename: Имя файла для загрузки

    Raises:
        FileNotFoundError: Если файл не существует
        JSONDecodeError: Если файл содержит некорректный JSON
    """
    if not Path(filename).exists():
        raise FileNotFoundError(f"Файл {filename} не существует")

    with open(filename, 'r', encoding='utf-8') as f:
        data = json.load(f)

    self.pets = []
    for pet_data in data:
        # Конвертируем строку обратно в Species Enum
        species_name = pet_data.get('species', 'OTHER')
        try:
            species = Species[species_name]
        except KeyError:
            species = Species.OTHER

        self.pets.append(Pet(
            name=pet_data['name'],
            species=species,
            age=pet_data['age']
        ))

def get_statistics(self) -> dict:
    """
    Получить статистику по питомцам.

    Returns:
        dict: Статистика (общее количество, по видам, средний возраст)
    """
    if not self.pets:
        return {"total": 0}

    stats = {
        "total": len(self.pets),
        "by_species": {},
        "average_age": sum(pet.age for pet in self.pets) / len(self.pets)
    }

    for pet in self.pets:
        species_name = pet.species.name
```

```
            stats["by_species"][species_name] =
stats["by_species"].get(species_name, 0) + 1

        return stats
```

Интерактивный интерфейс:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Интерактивная версия приложения "Учёт питомцев".
Предоставляет меню для взаимодействия с пользователем.
"""

import os
import sys
from pets_core import PetContainer, Species


class InteractivePetCLI:
    """Интерактивный интерфейс для управления питомцами."""

    def __init__(self):
        self.container = PetContainer()
        self.default_file = "pets.json"

    def load_default_file(self):
        """Загрузить данные из файла по умолчанию, если он существует."""
        if os.path.exists(self.default_file):
            try:
                self.container.load_from_file(self.default_file)
                print(f"✓ Загружено {len(self.container.pets)} питомцев из
{self.default_file}")
            except Exception as e:
                print(f"✗ Ошибка при загрузке: {e}")

    def print_menu(self):
        """Вывести меню команд."""
        print("\n" + "=" * 50)
        print("СИСТЕМА УЧЁТА ПИТОМЦЕВ")
        print("=" * 50)
        print("1. Добавить питомца")
        print("2. Показать всех питомцев")
        print("3. Найти питомцев по виду")
        print("4. Отсортировать питомцев")
        print("5. Показать статистику")
        print("6. Сохранить данные")
        print("7. Загрузить данные")
        print("8. Выйти")
        print("=" * 50)

    def add_pet_interactive(self):
```

```
        """Интерактивное добавление питомца."""
        print("\n--- Добавление питомца ---")

        name = input("Кличка питомца: ").strip()
        if not name:
            print("✗ Кличка не может быть пустой")
            return

        print("Доступные виды:", ", ".join([s.name.lower() for s in Species]))
        species_input = input("Вид животного: ").strip().upper()

        try:
            age = int(input("Возраст (лет): "))
        except ValueError:
            print("✗ Возраст должен быть числом")
            return

        try:
            pet = self.container.add_pet(name, species_input, age)
            print(f"✓ Добавлен питомец: {pet.name} ({pet.species}, {pet.age}
лет)")
        except Exception as e:
            print(f"✗ Ошибка: {e}")

    def find_pets_interactive(self):
        """Интерактивный поиск питомцев по виду."""
        print("\n--- Поиск питомцев ---")
        print("Доступные виды:", ", ".join([s.name.lower() for s in Species]))

        species_input = input("Введите вид для поиска: ").strip().upper()
        found = self.container.find_by_species(species_input)

        if not found:
            print(f"✗ Питомцы вида '{species_input.lower()}' не найдены")
            return

        print(f"\nНайдено {len(found)} питомцев:")
        print("-" * 40)
        for i, pet in enumerate(found, 1):
            print(f"{i}. {pet.name}, {pet.age} лет")
        print("-" * 40)

    def sort_pets_interactive(self):
        """Интерактивная сортировка питомцев."""
        print("\n--- Сортировка питомцев ---")
        print("1. По имени (А-Я)")
        print("2. По возрасту (младшие сначала)")
        print("3. По возрасту (старшие сначала)")

        choice = input("Выберите вариант сортировки: ").strip()
```

```
        if choice == "1":
            self.container.sort_by_name()
            print("✓ Питомцы отсортированы по имени")
        elif choice == "2":
            self.container.sort_by_age(reverse=False)
            print("✓ Питомцы отсортированы по возрасту (младшие сначала)")
        elif choice == "3":
            self.container.sort_by_age(reverse=True)
            print("✓ Питомцы отсортированы по возрасту (старшие сначала)")
        else:
            print("✗ Неверный выбор")

    def show_statistics(self):
        """Показать статистику по питомцам."""
        stats = self.container.get_statistics()

        print("\n--- Статистика ---")
        print(f"Всего питомцев: {stats['total']}")

        if stats['total'] > 0:
            print(f"Средний возраст: {stats['average_age']:.1f} лет")
            print("\nРаспределение по видам:")
            for species, count in stats['by_species'].items():
                species_name = Species[species].value
                print(f"  {species_name}: {count}")

    def save_data_interactive(self):
        """Интерактивное сохранение данных."""
        filename = input(f"Имя файла [{self.default_file}]: ").strip() or
self.default_file

        try:
            self.container.save_to_file(filename)
            print(f"✓ Данные сохранены в {filename}")
        except Exception as e:
            print(f"✗ Ошибка при сохранении: {e}")

    def load_data_interactive(self):
        """Интерактивная загрузка данных."""
        filename = input(f"Имя файла [{self.default_file}]: ").strip() or
self.default_file

        try:
            self.container.load_from_file(filename)
            print(f"✓ Загружено {len(self.container.pets)} питомцев из
{filename}")
        except FileNotFoundError:
            print(f"✗ Файл {filename} не существует")
        except Exception as e:
            print(f"✗ Ошибка при загрузке: {e}")
```

```
    def run(self):
        """Запустить интерактивный интерфейс."""
        self.load_default_file()

        while True:
            self.print_menu()

            try:
                choice = input("\nВыберите действие (1-8): ").strip()

                if choice == "1":
                    self.add_pet_interactive()
                elif choice == "2":
                    print(self.container.show_all())
                elif choice == "3":
                    self.find_pets_interactive()
                elif choice == "4":
                    self.sort_pets_interactive()
                elif choice == "5":
                    self.show_statistics()
                elif choice == "6":
                    self.save_data_interactive()
                elif choice == "7":
                    self.load_data_interactive()
                elif choice == "8":
                    # Автосохранение перед выходом
                    if self.container.pets:
                        save_choice = input("Сохранить данные перед выходом?
(y/n): ").lower()
                        if save_choice == 'y':
                            self.container.save_to_file(self.default_file)
                            print(f"✓ Данные сохранены в {self.default_file}")
                    print("До свидания!")
                    break
                else:
                    print("✗ Неверный выбор. Введите число от 1 до 8.")

            except KeyboardInterrupt:
                print("\n\nПрограмма прервана пользователем.")
                break
            except Exception as e:
                print(f"✗ Неожиданная ошибка: {e}")


def main():
    """Основная функция запуска интерактивной версии."""
    cli = InteractivePetCLI()
    cli.run()


if __name__ == "__main__":
```

```
main()
```

```
✓ Загружено 1 питомцев из pets.json

==================================================
СИСТЕМА УЧЁТА ПИТОМЦЕВ
==================================================
1. Добавить питомца
2. Показать всех питомцев
3. Найти питомцев по виду
4. Отсортировать питомцев
5. Показать статистику
6. Сохранить данные
7. Загрузить данные
8. Выйти
==================================================

Выберите действие (1-8):
```

Рисунок 1. Результат выполнения программы

14. Решили задание 2.

Самостоятельно изучите работу с пакетом click для построения интерфейса командной строки (CLI). Для своего варианта задания 1 необходимо реализовать интерфейс командной строки с использованием пакета click.

Листинг программы:

```
# -*- coding: utf-8 -*-
"""
Ядро приложения "Учёт питомцев".
Содержит классы данных и логику работы с коллекцией питомцев.
"""

import json
from dataclasses import dataclass, asdict, field
from typing import List, Optional
from pathlib import Path
from enum import Enum


class Species(Enum):
    """Перечисление видов животных."""
```

```
    CAT = "кот"
    DOG = "собака"
    BIRD = "птица"
    FISH = "рыба"
    RODENT = "грызун"
    OTHER = "другое"

    def __str__(self):
        return self.value


@dataclass
class Pet:
    """Класс для представления питомца."""
    name: str
    species: Species
    age: int

    def __post_init__(self):
        """Валидация данных после инициализации."""
        if not self.name.strip():
            raise ValueError("Имя питомца не может быть пустым")
        if self.age < 0:
            raise ValueError("Возраст не может быть отрицательным")
        if self.age > 100:
            raise ValueError("Возраст слишком большой")


@dataclass
class PetContainer:
    """Контейнер для управления коллекцией питомцев."""
    pets: List[Pet] = field(default_factory=list)

    def add_pet(self, name: str, species_str: str, age: int) -> Pet:
        """
        Добавить нового питомца.

        Args:
            name: Кличка питомца
            species_str: Вид животного
            age: Возраст в годах

        Returns:
            Pet: Созданный объект питомца

        Raises:
            ValueError: При некорректных данных
        """
        try:
            # Конвертируем строку в Species Enum
            species = Species[species_str.upper().replace(" ", "_")]
```

```
        except KeyError:
            # Если вид не найден в Enum, используем OTHER
            species = Species.OTHER

        pet = Pet(name=name, species=species, age=age)
        self.pets.append(pet)
        return pet

    def show_all(self) -> str:
        """
        Получить форматированное представление всех питомцев.

        Returns:
            str: Отформатированная строка со списком питомцев
        """
        if not self.pets:
            return "Список питомцев пуст."

        result = ["\nСписок всех питомцев:", "-" * 40]
        for i, pet in enumerate(self.pets, 1):
            result.append(f"{i}. {pet.name} - {pet.species}, {pet.age} лет")
        result.extend(["-" * 40, f"Всего: {len(self.pets)} питомцев"])
        return "\n".join(result)

    def find_by_species(self, species_str: str) -> List[Pet]:
        """
        Найти питомцев по виду.

        Args:
            species_str: Вид животного для поиска

        Returns:
            List[Pet]: Список найденных питомцев
        """
        try:
            species = Species[species_str.upper().replace(" ", "_")]
        except KeyError:
            # Если вид не найден, возвращаем пустой список
            return []

        return [pet for pet in self.pets if pet.species == species]

    def sort_by_age(self, reverse: bool = False) -> None:
        """Сортировать питомцев по возрасту."""
        self.pets.sort(key=lambda pet: pet.age, reverse=reverse)

    def sort_by_name(self) -> None:
        """Сортировать питомцев по имени."""
        self.pets.sort(key=lambda pet: pet.name)

    def save_to_file(self, filename: str) -> None:
```

```
        """
        Сохранить данные в JSON файл.

        Args:
            filename: Имя файла для сохранения

        Raises:
            IOError: При ошибке записи в файл
        """
        # Конвертируем Enum в строки для сериализации
        data = []
        for pet in self.pets:
            pet_dict = asdict(pet)
            pet_dict['species'] = pet.species.name  # Сохраняем имя Enum
            data.append(pet_dict)

        with open(filename, 'w', encoding='utf-8') as f:
            json.dump(data, f, ensure_ascii=False, indent=2)

    def load_from_file(self, filename: str) -> None:
        """
        Загрузить данные из JSON файла.

        Args:
            filename: Имя файла для загрузки

        Raises:
            FileNotFoundError: Если файл не существует
            JSONDecodeError: Если файл содержит некорректный JSON
        """
        if not Path(filename).exists():
            raise FileNotFoundError(f"Файл {filename} не существует")

        with open(filename, 'r', encoding='utf-8') as f:
            data = json.load(f)

        self.pets = []
        for pet_data in data:
            # Конвертируем строку обратно в Species Enum
            species_name = pet_data.get('species', 'OTHER')
            try:
                species = Species[species_name]
            except KeyError:
                species = Species.OTHER

            self.pets.append(Pet(
                name=pet_data['name'],
                species=species,
                age=pet_data['age']
            ))
```

```python
    def get_statistics(self) -> dict:
        """
        Получить статистику по питомцам.

        Returns:
            dict: Статистика (общее количество, по видам, средний возраст)
        """
        if not self.pets:
            return {"total": 0}

        stats = {
            "total": len(self.pets),
            "by_species": {},
            "average_age": sum(pet.age for pet in self.pets) / len(self.pets)
        }

        for pet in self.pets:
            species_name = pet.species.name
            stats["by_species"][species_name] =
stats["by_species"].get(species_name, 0) + 1

        return stats
```

```
usage: pets_cli.py [-h] {add,list,find,stats,save,load,sort} ...

Учёт питомцев - CLI приложение для управления коллекцией питомцев

options:
  -h, --help            show this help message and exit

доступные команды:
  выберите одну из следующих команд:

  {add,list,find,stats,save,load,sort}
                        дополнительная справка по командам
    add                 добавить нового питомца
    list                показать всех питомцев
    find                найти питомцев по виду
    stats               показать статистику по питомцам
    save                сохранить данные в файл
    load                загрузить данные из файла
    sort                отсортировать питомцев

Примеры использования:
  pets_cli.py add -n Барсик -s CAT -a 3
  pets_cli.py list
  pets_cli.py find -s DOG
  pets_cli.py save -f my_pets.json
  pets_cli.py load -f my_pets.json
```

Рисунок 2. Результат

Вывод: в ходе выполнения лабораторной работы были приобретены навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.