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 13

Выполнил:
Рябинин Егор Алексеевич
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 Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г

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

Цель: Приобретение построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.13.3.

Порядок выполнения работы:

Ссылка на репозиторий:

https://github.com/bohemiaaaaa/Lab10_Object-oriented-programming

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 декоратор @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 флаги (--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:

– датаклассы описывают структуру данных;

– контейнер управляет коллекцией;

– CLI обеспечивает взаимодействие с пользователем;

– функции загрузки/сохранения отделены 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.

13. Каталог рецептов.

Рецепт: название, кухня, время приготовления. CLI: добавление рецепта, вывод всех рецептов по кухне, сортировка по времени, JSON.

Листинг программы 1:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import argparse
import json
from dataclasses import asdict, dataclass, field
from pathlib import Path
from typing import List


@dataclass(frozen=True)
class Recipe:
    name: str
    cuisine: str
    time: int

    def __post_init__(self):
        if not isinstance(self.time, int):
            raise TypeError("time must be int")


@dataclass
class RecipeCatalog:
    recipes: List[Recipe] = field(default_factory=list)

    def add(self, recipe: Recipe) -> None:
        self.recipes.append(recipe)
        self.recipes.sort(key=lambda r: r.time)

    def select_by_cuisine(self, cuisine: str) -> List[Recipe]:
        return [r for r in self.recipes if r.cuisine.lower() ==
```

```
cuisine.lower()]

    def load(self, filename: str) -> None:
        if not Path(filename).exists():
            return

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

        self.recipes = [Recipe(**item) for item in data]
        self.recipes.sort(key=lambda r: r.time)

    def save(self, filename: str) -> None:
        with open(filename, "w", encoding="utf-8") as f:
            json.dump(
                [asdict(r) for r in self.recipes], f, ensure_ascii=False,
indent=2
            )

    def __str__(self) -> str:
        if not self.recipes:
            return "Каталог рецептов пуст."

        lines = [
            "+----+------------------------------+--------------------+------
--+",
            "| №  | Название                     | Кухня              | Время
|",
            "+----+------------------------------+--------------------+------
--+",
        ]

        for i, r in enumerate(self.recipes, 1):
            lines.append(f"| {i:>2} | {r.name:<28} | {r.cuisine:<18} |
{r.time:>6} |")

        lines.append(
            "+----+------------------------------+--------------------+------
--+"
        )
        return "\n".join(lines)


# ====== CLI ======


def build_parser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(
        description="Каталог рецептов (CLI, dataclass, JSON)"
    )

    parser.add_argument("--file", default="recipes.json", help="JSON-файл с
данными")

    subparsers = parser.add_subparsers(dest="command", required=True)

    add = subparsers.add_parser("add", help="Добавить рецепт")
    add.add_argument("--name", required=True)
    add.add_argument("--cuisine", required=True)
    add.add_argument("--time", required=True, type=int)

    subparsers.add_parser("list", help="Показать все рецепты")

    select = subparsers.add_parser("select", help="Выборка по кухне")
```

```
    select.add_argument("--cuisine", required=True)

    subparsers.add_parser("save", help="Сохранить данные")

    subparsers.add_parser("load", help="Загрузить данные")

    return parser


def main() -> None:
    parser = build_parser()
    args = parser.parse_args()

    catalog = RecipeCatalog()
    catalog.load(args.file)

    if args.command == "add":
        catalog.add(Recipe(args.name, args.cuisine, args.time))
        catalog.save(args.file)
        print("Рецепт добавлен.")

    elif args.command == "list":
        print(catalog)

    elif args.command == "select":
        result = catalog.select_by_cuisine(args.cuisine)
        if not result:
            print("Ничего не найдено.")
        else:
            for r in result:
                print(f"{r.name} - {r.time} мин")

    elif args.command == "save":
        catalog.save(args.file)
        print("Данные сохранены.")

    elif args.command == "load":
        print("Данные загружены.")
        print(catalog)


if __name__ == "__main__":
    main()
```

Результат работы программы:

```
PS C:\Users\4isto\OOP_lab10> python tasks/task1.py --help
usage: task1.py [-h] [--file FILE] {add,list,select,save,load} ...

Каталог рецептов (CLI, dataclass, JSON)

positional arguments:
  {add,list,select,save,load}
    add                 Добавить рецепт
    list                Показать все рецепты
    select              Выборка по кухне
    save                Сохранить данные
    load                Загрузить данные

options:
  -h, --help            show this help message and exit
  --file FILE           JSON-файл с данными
```

Рисунок 1 – Подробный вывод списка команд

```
PS C:\Users\4isto\OOP_lab10> python tasks/task1.py add --name Пельмени --cuisine Русская --time 60
Рецепт добавлен.
```

Рисунок 2 – Добавление рецепта

```
PS C:\Users\4isto\OOP_lab10> python tasks/task1.py list
+----+------------------------------+--------------------+--------+
| №  | Название                     | Кухня              | Время  |
+----+------------------------------+--------------------+--------+
|  1 | Пельмени                     | Русская            |     60 |
|  2 | Пицца                        | Итальянская        |     60 |
+----+------------------------------+--------------------+--------+
```

Рисунок 3 – Вывод всех рецептов

```
PS C:\Users\4isto\OOP_lab10> python tasks/task1.py select --cuisine Итальянская
Пицца — 60 мин
```

Рисунок 4 – Выборка по кухне (стране)

```
PS C:\Users\4isto\OOP_lab10> python tasks/task1.py save
Данные сохранены.
```

Рисунок 5 – Сохранение данных

```
PS C:\Users\4isto\OOP_lab10> python tasks/task1.py load
Данные загружены.
+----+------------------------------+--------------------+--------+
| №  | Название                     | Кухня              | Время  |
+----+------------------------------+--------------------+--------+
|  1 | Пельмени                     | Русская            |     60 |
|  2 | Пицца                        | Итальянская        |     60 |
+----+------------------------------+--------------------+--------+
```

Рисунок 6 – Загрузка данных

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 2:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


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

import click


@dataclass(frozen=True)
```

```python
class Recipe:
    name: str
    cuisine: str
    time: int

    def __post_init__(self):
        if not isinstance(self.time, int):
            raise TypeError("time must be int")


@dataclass
class RecipeCatalog:
    recipes: List[Recipe] = field(default_factory=list)

    def add(self, recipe: Recipe) -> None:
        self.recipes.append(recipe)
        self.recipes.sort(key=lambda r: r.time)

    def select_by_cuisine(self, cuisine: str) -> List[Recipe]:
        return [r for r in self.recipes if r.cuisine.lower() ==
cuisine.lower()]

    def load(self, filename: str) -> None:
        if not Path(filename).exists():
            return

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

        self.recipes = [Recipe(**item) for item in data]
        self.recipes.sort(key=lambda r: r.time)

    def save(self, filename: str) -> None:
        with open(filename, "w", encoding="utf-8") as f:
            json.dump(
                [asdict(r) for r in self.recipes], f, ensure_ascii=False,
indent=2
            )

    def __str__(self) -> str:
        if not self.recipes:
            return "Каталог рецептов пуст."

        lines = [
            "+----+------------------------------+--------------------+------
--+",
            "| №  | Название                     | Кухня              | Время
|",
            "+----+------------------------------+--------------------+------
--+",
        ]

        for i, r in enumerate(self.recipes, 1):
            lines.append(f"| {i:>2} | {r.name:<28} | {r.cuisine:<18} |
{r.time:>6} |")

        lines.append(
            "+----+------------------------------+--------------------+------
--+"
        )
        return "\n".join(lines)


# ====== CLI (CLICK) ======
```

```python
@click.group()
@click.option("--file", default="recipes2.json", help="JSON-файл с данными")
@click.pass_context
def cli(ctx, file):
    """Каталог рецептов (CLI на click)"""
    catalog = RecipeCatalog()
    catalog.load(file)
    ctx.obj = {"catalog": catalog, "file": file}


@cli.command()
@click.option("--name", required=True, help="Название рецепта")
@click.option("--cuisine", required=True, help="Кухня")
@click.option("--time", required=True, type=int, help="Время приготовления")
@click.pass_context
def add(ctx, name, cuisine, time):
    """Добавить рецепт"""
    catalog = ctx.obj["catalog"]
    filename = ctx.obj["file"]

    catalog.add(Recipe(name, cuisine, time))
    catalog.save(filename)

    click.echo("Рецепт добавлен.")


@cli.command(name="list")
@click.pass_context
def list_recipes(ctx):
    """Показать все рецепты"""
    catalog = ctx.obj["catalog"]
    click.echo(catalog)


@cli.command()
@click.option("--cuisine", required=True, help="Кухня для выборки")
@click.pass_context
def select(ctx, cuisine):
    """Выборка рецептов по кухне"""
    catalog = ctx.obj["catalog"]
    result = catalog.select_by_cuisine(cuisine)

    if not result:
        click.echo("Ничего не найдено.")
    else:
        for r in result:
            click.echo(f"{r.name} - {r.time} мин")


@cli.command()
@click.pass_context
def save(ctx):
    """Сохранить данные"""
    catalog = ctx.obj["catalog"]
    filename = ctx.obj["file"]

    catalog.save(filename)
    click.echo("Данные сохранены.")


@cli.command()
@click.pass_context
```

```python
def load(ctx):
    """Загрузить и показать данные"""
    catalog = ctx.obj["catalog"]
    click.echo("Данные загружены.")
    click.echo(catalog)


if __name__ == "__main__":
    cli()
```

Результат работы программы:

```
PS C:\Users\4isto\OOP_lab10> python tasks/task2.py --help
Usage: task2.py [OPTIONS] COMMAND [ARGS]...

  Каталог рецептов (CLI на click)

Options:
  --file TEXT  JSON-файл с данными
  --help       Show this message and exit.

Commands:
  add     Добавить рецепт
  list    Показать все рецепты
  load    Загрузить и показать данные
  save    Сохранить данные
  select  Выборка рецептов по кухне
```

Рисунок 7 – Подробный вывод списка команд

```
PS C:\Users\4isto\OOP_lab10> python tasks/task2.py add --name Пицца --cuisine Итальянская --time 60
Рецепт добавлен.
PS C:\Users\4isto\OOP_lab10> python tasks/task2.py add --name Пельмени --cuisine Русская --time 60
Рецепт добавлен.
```

Рисунок 8 – Добавление рецепта

```
PS C:\Users\4isto\OOP_lab10> python tasks/task2.py list
+----+-----------------------------+--------------------+--------+
| №  | Название                    | Кухня              | Время  |
+----+-----------------------------+--------------------+--------+
|  1 | Пицца                       | Итальянская        |     60 |
|  2 | Пельмени                    | Русская            |     60 |
+----+-----------------------------+--------------------+--------+
```

Рисунок 9 – Вывод всех рецептов

```
PS C:\Users\4isto\OOP_lab10> python tasks/task2.py select --cuisine Русская
Пельмени — 60 мин
```

Рисунок 10 – Выборка по кухне (стране)

```
Пельмени — 60 мин
PS C:\Users\4isto\OOP_lab10> python tasks/task2.py save
Данные сохранены.
```

Рисунок 11 – Сохранение данных

```
PS C:\Users\4isto\OOP_lab10> python tasks/task2.py load
Данные загружены.
+----+------------------------------+--------------------+--------+
| №  | Название                     | Кухня              | Время  |
+----+------------------------------+--------------------+--------+
|  1 | Пицца                        | Итальянская        |     60 |
|  2 | Пельмени                     | Русская            |     60 |
+----+------------------------------+--------------------+--------+
```

Рисунок 12 – Загрузка данных

Тесты для написанных программ.

Листинг программы 3:

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from pathlib import Path

import pytest
from task1 import Recipe, RecipeCatalog


@pytest.fixture
def catalog():
    return RecipeCatalog()


def test_recipe_creation():
    recipe = Recipe("Борщ", "Украинская", 60)
    assert recipe.name == "Борщ"
    assert recipe.cuisine == "Украинская"
    assert recipe.time == 60


def test_recipe_time_must_be_int():
    with pytest.raises(TypeError):
        Recipe("Борщ", "Украинская", "час")


def test_add_and_sort(catalog):
    catalog.add(Recipe("Долма", "Кавказская", 90))
    catalog.add(Recipe("Паста", "Итальянская", 20))

    assert len(catalog.recipes) == 2
    assert catalog.recipes[0].name == "Паста"
    assert catalog.recipes[1].name == "Долма"


@pytest.mark.parametrize(
    "cuisine, expected",
    [
        ("японская", 2),
        ("Украинская", 1),
        ("Французская", 0),
    ],
)
def test_select_by_cuisine(cuisine, expected):
```

```
    catalog = RecipeCatalog()
    catalog.add(Recipe("Суши", "Японская", 50))
    catalog.add(Recipe("Рамен", "Японская", 40))
    catalog.add(Recipe("Борщ", "Украинская", 60))

    result = catalog.select_by_cuisine(cuisine)
    assert len(result) == expected


def test_save_and_load_json(tmp_path: Path):
    file = tmp_path / "recipes.json"

    catalog = RecipeCatalog()
    catalog.add(Recipe("Пицца", "Итальянская", 30))
    catalog.save(file)

    new_catalog = RecipeCatalog()
    new_catalog.load(file)

    assert len(new_catalog.recipes) == 1
    assert new_catalog.recipes[0].name == "Пицца"


def test_load_nonexistent_file_does_not_fail():
    catalog = RecipeCatalog()
    catalog.load("no_such_file.json")
    assert catalog.recipes == []
```

Листинг программы 4:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from pathlib import Path

import pytest
from task2 import Recipe, RecipeCatalog


@pytest.fixture
def filled_catalog():
    catalog = RecipeCatalog()
    catalog.add(Recipe("Рамен", "Японская", 40))
    catalog.add(Recipe("Суши", "Японская", 50))
    catalog.add(Recipe("Борщ", "Украинская", 60))
    return catalog


def test_recipe_fields():
    recipe = Recipe("Тако", "Мексиканская", 20)
    assert recipe.name == "Тако"
    assert recipe.cuisine == "Мексиканская"
    assert recipe.time == 20


def test_invalid_time_type():
    with pytest.raises(TypeError):
        Recipe("Паста", "Итальянская", None)


def test_filter_japanese_recipes(filled_catalog):
    result = filled_catalog.select_by_cuisine("японская")
```

```
    assert len(result) == 2


def test_sorting_by_time():
    catalog = RecipeCatalog()
    catalog.add(Recipe("Долгое", "Тест", 100))
    catalog.add(Recipe("Быстрое", "Тест", 5))

    assert catalog.recipes[0].time == 5


def test_json_roundtrip(tmp_path: Path):
    file = tmp_path / "data.json"

    catalog = RecipeCatalog()
    catalog.add(Recipe("Лапша", "Китайская", 25))
    catalog.save(file)

    loaded = RecipeCatalog()
    loaded.load(file)

    assert len(loaded.recipes) == 1
    assert loaded.recipes[0].cuisine == "Китайская"


def test_empty_catalog_str():
    catalog = RecipeCatalog()
    assert "пуст" in str(catalog).lower()
```

Результат работы тестов:

```
PS C:\Users\4isto\OOP_lab10> pytest
============================= test session starts ==============
platform win32 -- Python 3.11.0, pytest-8.3.5, pluggy-1.6.0 -- C:\Program Files\Python311\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\4isto\OOP_lab10
configfile: pyproject.toml
testpaths: tests
plugins: anyio-4.9.0
collected 14 items

tests/test_task1.py::test_recipe_creation PASSED
tests/test_task1.py::test_recipe_time_must_be_int PASSED
tests/test_task1.py::test_add_and_sort PASSED
tests/test_task1.py::test_select_by_cuisine[\u044f\u043f\u043e\u043d\u0441\u043a\u0430\u044f-2] PASSED
tests/test_task1.py::test_select_by_cuisine[\u0423\u043a\u0440\u0430\u0438\u043d\u0441\u043a\u0430\u044f-1] PASSED
tests/test_task1.py::test_select_by_cuisine[\u0424\u0440\u0430\u043d\u0446\u0443\u0437\u0441\u043a\u0430\u044f-0] PASSED
tests/test_task1.py::test_save_and_load_json PASSED
tests/test_task1.py::test_load_nonexistent_file_does_not_fail PASSED
tests/test_task2.py::test_recipe_fields PASSED
tests/test_task2.py::test_invalid_time_type PASSED
tests/test_task2.py::test_filter_japanese_recipes PASSED
tests/test_task2.py::test_sorting_by_time PASSED
tests/test_task2.py::test_json_roundtrip PASSED
tests/test_task2.py::test_empty_catalog_str PASSED

============================== 14 passed in 0.09s ==============
```

Рисунок 13 – Результат работы тестов

Контрольные вопросы:

1. В чем отличие терминала и консоли?

Терминал – это программа-эмулятор или физическое устройство для ввода-вывода текста. Консоль – это специализированный терминал, напрямую подключённый к системе (аппаратная консоль). В современном обиходе термины часто взаимозаменяемы.

2. Что такое консольное приложение?

Программа, работающая в текстовом режиме (командной строке), без графического интерфейса, взаимодействующая с пользователем через ввод команд и текстовый вывод.

3. Какие существуют средства языка программирования Python для построения приложений командной строки?

Стандартные модули: sys (используя sys.argv), getopt, argparse. Популярные сторонние библиотеки: click, docopt, fire.

4. Какие особенности построение CLI с использованием модуля sys?

Используется список sys.argv. Обработка примитивная, требуется ручной разбор аргументов, проверка типов и количества, нет автоматической генерации справки.

5. Какие особенности построение CLI с использованием модуля getopt?

Предоставляет функции для разбора аргументов в стиле C (getopt.getopt). Более структурированный разбор, чем у sys.argv, но всё ещё ручной, нет автоматической справки, сложен для сложных интерфейсов.

6. Для чего используется модуль argparse в Python и какие задачи он решает?

Для создания продвинутых интерфейсов командной строки. Решает задачи: объявление аргументов, их типов, значений по умолчанию; автоматическая генерация справки (--help); обработка ошибок ввода; поддержка подкоманд.

7. Что такое позиционные аргументы и чем они отличаются от опциональных?

Позиционные аргументы обязательны и их порядок важен. Опциональные аргументы (флаги, например --file) не обязательны, их порядок не важен.

8. Как создать объект парсера командной строки?

parser = argparse.ArgumentParser(description='Описание программы')

9. Как добавить к программе параметр вида --verbose с помощью argparse?

parser.add_argument('--verbose', action='store_true', help='Включить подробный вывод')

10. Как задать тип аргумента, чтобы значение автоматически преобразовывалось, например, в int?

Использовать параметр type: parser.add_argument('--number', type=int)

11. Что делает параметр required=True при добавлении аргумента?

Делает опциональный аргумент обязательным для указания пользователем.

12. Как в argparse создать справку (--help) и нужно ли добавлять её вручную?

Справка создаётся автоматически при добавлении параметра help к каждому аргументу. Добавлять её вручную не нужно, --help генерируется по умолчанию.

13. Как сделать параметр по умолчанию, например --limit 10?

Использовать параметр default: parser.add_argument('--limit', type=int, default=10)

14. Что такое подкоманды (subparsers) и для чего они используются?

Это механизм для создания вложенных команд в CLI (например, git commit, git push). Используются для организации сложной логики приложения с разными наборами аргументов для разных действий.

15. Как добавить подкоманду, например add, имеющую собственный набор аргументов?

subparsers = parser.add_subparsers()

parser_add = subparsers.add_parser('add')

parser_add.add_argument('--name', required=True)

16. Как получить результат разборки аргументов и где хранится значение каждого флага?

args = parser.parse_args(). Значения хранятся как атрибуты объекта args (например, args.name).

17. Что делает параметр action="store_true" и когда он бывает полезен?

Создаёт булевый флаг. Если флаг указан, значение становится True, иначе False. Полезен для включения/выключения режимов (например, --verbose).

18. Как задать ограниченный набор допустимых значений для аргумента (choices=[...])?

Использовать параметр choices: parser.add_argument('--color', choices=['red', 'green', 'blue'])

19. Можно ли использовать аргументы одновременно как короткие (-v) и длинные (--verbose) версии?

Да, передав оба варианта в add_argument: parser.add_argument('-v', '--verbose', action='store_true')

20. Как вывести справку по отдельной подкоманде, например prog.py add --help?

Справка для подкоманды генерируется автоматически при вызове prog.py add --help, если подкоманда была добавлена через add_subparsers.

21. Как указать пользовательское сообщение описания программы (description=)

При создании парсера: argparse.ArgumentParser(description='Моё описание программы')

22. Как реализовать печать версии программы через --version?

parser.add_argument('--version', action='version', version='%(prog)s 1.0')

23. Что будет, если пользователь введёт неизвестный аргумент, и как это обработать?

По умолчанию argparse завершит программу с ошибкой. Можно изменить поведение, задав argument_default=argparse.SUPPRESS или обрабатывая sys.argv вручную до парсинга.

24. Как сделать аргумент, который может встречаться несколько раз (nargs='+' или '*') и что означают эти режимы?

nargs='+' – один или более аргументов (список). nargs='*' – ноль или более аргументов (список). Пример: parser.add_argument('files', nargs='+').

25. Как отключить автоматическое форматирование help-сообщений и управлять форматированием вручную (RawTextHelpFormatter)?

Использовать formatter_class: parser = argparse.ArgumentParser(formatter_class=argparse.RawTextHelpFormatter)

26. Для чего используется библиотека click, и чем она отличается от стандартного модуля argparse?

click – сторонняя библиотека для создания CLI с декларативным синтаксисом на декораторах. Отличается более простым и кратким кодом, автоматической генерацией справки, лучшей поддержкой вложенных команд, проверкой типов.

27. Что делает декоратор @click.command() и когда он применяется?

Объявляет функцию как команду CLI. Применяется для создания корневой команды. Функция становится точкой входа для парсера click.

28. Как объявить аргумент командной строки с помощью @click.argument() и чем он отличается от @click.option()?

@click.argument() – для позиционных аргументов. @click.option() – для опциональных аргументов (флагов). Аргументы обязательны и идут в определённом порядке, опции – нет.

29. Для чего нужен параметр prompt= в click.option() и в каких ситуациях он удобен?

Если опция не указана в командной строке, click запросит её значение у пользователя интерактивно. Удобно для конфиденциальных данных (пароли) или для упрощения интерфейса.

30. Как сделать опцию булевой (--verbose / --no-verbose) с помощью click?

Использовать is_flag=True: @click.option('--verbose', is_flag=True, default=False, help='Включить подробный вывод'). Для пары --flag/--no-flag можно использовать @click.option('--flag/--no-flag', default=True).

31. Как создать подкоманды в click и что делает декоратор @click.group()?

@click.group() создаёт группу команд (аналог подкоманд). Функция с этим декоратором становится родительской. Подкоманды объявляются с помощью @<группа>.command().

32. Каким образом можно задать выбор из фиксированного набора значений (type=click.Choice([...]))?

@click.option('--color', type=click.Choice(['red', 'green', 'blue']))

33. Что делает параметр default= в click.option() и как задать значение по умолчанию?

Устанавливает значение по умолчанию для опции, если пользователь её не указал. Пример: @c lick.option('--limit', default=10).

34. Как организовать вывод ошибок и корректное завершение программы через click.echo() и click.ClickException?

click.echo() для вывода текста (лучше, чем print). Для ошибок – вызвать исключение click.ClickException('Сообщение об ошибке'), оно будет корректно обработано и выведено.

35. Как работает механизм контекстов (click.Context) и для чего может понадобиться передача общего состояния между командами?

Контекст (ctx) хранит состояние CLI (аргументы, объекты) и передаётся между командами одной группы. Полезен для передачи общего конфигурационного объекта, подключения к БД и т.п.

Вывод: в ходе лабораторной работы были приобретены навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.13.3.