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 № 3

Выполнил:
Левашев Тимур Рашидович
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 работы: “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.

Порядок выполнения работы:

Ссылка на Git репозиторий: https://github.com/mazy99/oop_pract_10

1. **Пример 1:** Использование модуля getopt для создания командного интерфейса.

Листинг программы 1 – Обработка параметров CLI через getopt

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

import getopt
import sys


def main() -> None:
    full_cmd_args: list[str] = sys.argv
    argument_list: list[str] = full_cmd_args[1:]
    short_options: str = "ho:v"
    long_options: list[str] = ["help", "output=", "verbose"]

    try:
        arguments: list[tuple[str, str]]
        values: list[str]
        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 ("-h", "--help"):
            print("Displaying help")
        elif current_argument in ("-o", "--output"):
            print(f"Output set to: {current_value}")
        elif current_argument in ("-v", "--verbose"):
            print("Verbose mode enabled")

    if values:
        print(f"Remaining arguments: {values}")
```

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

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run examples\getopt_example.py -o green --help -v

Output set to: green
Displaying help
Verbose mode enabled
```

Рисунок 1 – Результат выполнения программы

2. **Пример 2:** Использование модуля agrparse для создания командного интерфейса (вычисление квадрата).

Листинг программы 2 – Реализация CLI для вычисления квадрата числа

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


import argparse
from argparse import ArgumentParser, Namespace


def main() -> None:
    parser: ArgumentParser = argparse.ArgumentParser()
    parser.add_argument("square", type=int, help="Display the square of a given
number")

    parser.add_argument("--verbose", "-v", action="store_true", help="Increase
output verbosity")

    args: Namespace = parser.parse_args()
    result: int = args.square ** 2
    if args.verbose:
        print(f"The square of {args.square} is {result}")
    else:
        print(result)


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

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pra
The square of 5 is 25
```

Рисунок 2 – Результат выполнения программы

3. **Пример 3:** Создание многоуровневого интерфейса командной строки. Использование subparsers в agrparse для создания подкоманд.

Листинг 3 – Парсер командной строки с поддержкой нескольких команд.Многоуровневый интерфейс agreparse с подкомандами

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

import argparse
from argparse import ArgumentParser, _SubParsersAction, Namespace


def main() -> None:
    parser: ArgumentParser = argparse.ArgumentParser()
    subparsers: _SubParsersAction = parser.add_subparsers(
        help='List of commands'
    )

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

    create_parser: ArgumentParser = 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'
    )

    args: Namespace = parser.parse_args()
    print(args)
```

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

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run examples\agreparse_example_2.py -h
usage: agreparse_example_2.py [-h] {list,create} ...

positional arguments:
  {list,create}         List of commands
    list                List contents
    create              Create a directory

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

Рисунок 3 – Результат выполнения программы с ключом -h

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run python examples\agreparse_example_2.py list examples
Namespace(dirname='examples')
```

Рисунок 4 – Результат выполнения программы для аргумента list

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run python examples\agreparse_example_2.py create test_dir --read-o
nly
Namespace(dirname='test_dir', read_only=True)
```

Рисунок 5 – Результат выполнения программы для аргумента create

4. **Пример 4:** Работа с уровнями детализации вывода в agreparse. Вычисление квадрата с многоуровневой детализацией

Листинг 4 – Реализация парсера с многоуровневой детализацией

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

import argparse
from argparse import ArgumentParser, Namespace


def main() -> None:
    parser: ArgumentParser = argparse.ArgumentParser()
    parser.add_argument(
        "square",
        type=int,
        help="display the square of a given number"
    )
    parser.add_argument(
        "-v", "--verbosity",
        action="count",
```

```
        help="increase output verbosity"
    )

    args: Namespace = parser.parse_args()
    answer: int = 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)


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

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run python examples\agreparse_example_3.py 5 -v
5*2 = 25
```

Рисунок 6 – Результат выполнения программы

5. Пример 5:

Листинг 5 –

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

import argparse
import sys
import xml.etree.ElementTree as ET
from dataclasses import dataclass, field
from datetime import date


@dataclass(frozen=True)
class Worker:
    name: str
    post: str
    year: int


@dataclass
class Staff:
    workers: list[Worker] = field(default_factory=list)

    def add(self, name: str, post: str, year: int) -> None:
        self.workers.append(
            Worker(
                name=name,
```

```
                post=post,
                year=year
            )
        )
        self.workers.sort(key=lambda worker: worker.name)

    def __str__(self) -> str:
        table: list[str] = []
        line: str = f"+-{'-' * 4}-+-{'-' * 30}-+-{'-' * 20}-+-{'-' * 8}-+"
        table.append(line)
        table.append(f"| {'№':^4} | {'Ф.И.О.':^30} | {'Должность':^20} |
{'Год':^8} |")
        table.append(line)

        for idx, worker in enumerate(self.workers, 1):
            table.append(f"| {idx:>4} | {worker.name:<30} | {worker.post:<20} |
{worker.year:>8} |")

        table.append(line)
        return '\n'.join(table)

    def select(self, period: int) -> list[Worker]:
        today: date = date.today()
        result: list[Worker] = []

        for worker in self.workers:
            if today.year - worker.year >= period:
                result.append(worker)

        return result

    def load(self, filename: str) -> None:
        with open(filename, 'r', encoding='utf-8') as fin:
            xml: str = fin.read()

        parser: ET.XMLParser = ET.XMLParser(encoding='utf-8')
        tree: ET.ElementTree = ET.fromstring(xml, parser=parser)

        self.workers = []
        for worker_element in tree:
            name: str | None = None
            post: str | None = None
            year: int | None = None

            for element in worker_element:
                if element.tag == 'name':
                    name = element.text
                elif element.tag == 'post':
                    post = element.text
                elif element.tag == 'year':
                    year = int(element.text)
```

```
            if name is not None and post is not None and year is not None:
                self.workers.append(
                    Worker(
                        name=name,
                        post=post,
                        year=year
                    )
                )

    def save(self, filename: str) -> None:
        root: ET.Element = ET.Element('workers')

        for worker in self.workers:
            worker_element: ET.Element = ET.Element('worker')

            name_element: ET.Element = ET.SubElement(worker_element, 'name')
            name_element.text = worker.name

            post_element: ET.Element = ET.SubElement(worker_element, 'post')
            post_element.text = worker.post

            year_element: ET.Element = ET.SubElement(worker_element, 'year')
            year_element.text = str(worker.year)

            root.append(worker_element)

        tree: ET.ElementTree = ET.ElementTree(root)
        with open(filename, 'wb') as fout:
            tree.write(fout, encoding='utf-8', xml_declaration=True)


def build_parser() -> argparse.ArgumentParser:
    parser: argparse.ArgumentParser = argparse.ArgumentParser(
        description="Система учёта сотрудников (dataclass + XML + argparse)"
    )

    subparsers: argparse._SubParsersAction =
parser.add_subparsers(dest="command")

    add_parser: argparse.ArgumentParser = subparsers.add_parser(
        "add",
        help="Добавить сотрудника"
    )
    add_parser.add_argument(
        "--name",
        required=True,
        help="Фамилия и инициалы"
    )
    add_parser.add_argument(
        "--post",
```

```
        required=True,
        help="Должность"
    )
    add_parser.add_argument(
        "--year",
        required=True,
        type=int,
        help="Год поступления"
    )

    subparsers.add_parser(
        "list",
        help="Показать всех сотрудников"
    )

    select_parser: argparse.ArgumentParser = subparsers.add_parser(
        "select",
        help="Выбрать по стажу"
    )
    select_parser.add_argument(
        "--period",
        required=True,
        type=int,
        help="Стаж (годы)"
    )

    load_parser: argparse.ArgumentParser = subparsers.add_parser(
        "load",
        help="Загрузить из XML"
    )
    load_parser.add_argument(
        "filename",
        help="Имя XML файла"
    )

    save_parser: argparse.ArgumentParser = subparsers.add_parser(
        "save",
        help="Сохранить в XML"
    )
    save_parser.add_argument(
        "filename",
        help="Имя XML файла"
    )

    return parser


def main() -> None:
    staff: Staff = Staff()

    import os
```

```
    if os.path.exists("staff.xml"):
        staff.load("staff.xml")

    print("Система учёта сотрудников")
    print("Команды: add, list, select, load, save, exit")
    print()

    while True:
        try:
            command: str = input("Введите команду: ").strip()

            if command == "exit":
                print("До свидания!")
                break

            elif command == "add":
                name: str = input("Фамилия и инициалы: ").strip()
                post: str = input("Должность: ").strip()
                year: int = int(input("Год поступления: ").strip())
                staff.add(name, post, year)
                staff.save("staff.xml")
                print("✓ Сотрудник добавлен.")

            elif command == "list":
                print(staff)

            elif command == "select":
                period: int = int(input("Стаж (годы): ").strip())
                selected: list[Worker] = staff.select(period)
                if selected:
                    print(f"\nСотрудники со стажем >= {period} лет:")
                    for idx, worker in enumerate(selected, 1):
                        print(f"{idx:>4}: {worker.name} - {worker.post}
({worker.year})")
                else:
                    print("Работники с заданным стажем не найдены.")

            elif command == "load":
                filename: str = input("Имя XML файла: ").strip()
                staff.load(filename)
                print(f"✓ Данные загружены из {filename}")

            elif command == "save":
                filename: str = input("Имя XML файла: ").strip()
                staff.save(filename)
                print(f"✓ Данные сохранены в {filename}")

            else:
                print("Неизвестная команда. Доступные команды: add, list, select,
load, save, exit")
```

```
            print()

        except ValueError:
            print("Ошибка: неверное значение. Попробуйте снова.")
            print()
        except Exception as e:
            print(f"Ошибка: {e}")
            print()


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

```
Введите команду: add
Фамилия и инициалы: Иванов И.И
Должность: инженер
Год поступления: 2015
✓ Сотрудник добавлен.

Введите команду: add
Фамилия и инициалы: Петров П.П
Должность: менеджер
Год поступления: 2010
✓ Сотрудник добавлен.
```

Рисунок 7 – Результат добавления сотрудников

```
Введите команду: list
+------+--------------------------------+----------------------+----------+
|  №   |             Ф.И.О.             |      Должность       |   Год    |
+------+--------------------------------+----------------------+----------+
|    1 | Иванов И.И                     | инженер              |     2015 |
|    2 | Петров П.П                     | менеджер             |     2010 |
+------+--------------------------------+----------------------+----------+
```

Рисунок 8 – Результат вывода сотрудников

```
Стаж (годы): 15

Сотрудники со стажем >= 15 лет:
   1: Петров П.П - менеджер (2010)
```

Рисунок 9 – Результат вывода сотрудников со стажем 15 и более лет

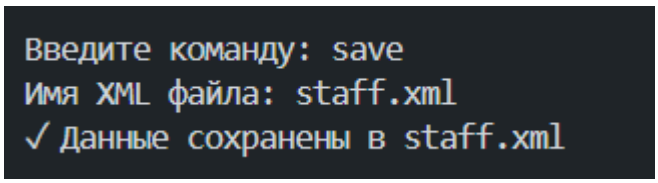

Рисунок 10 – Сохранение данных в xml файл

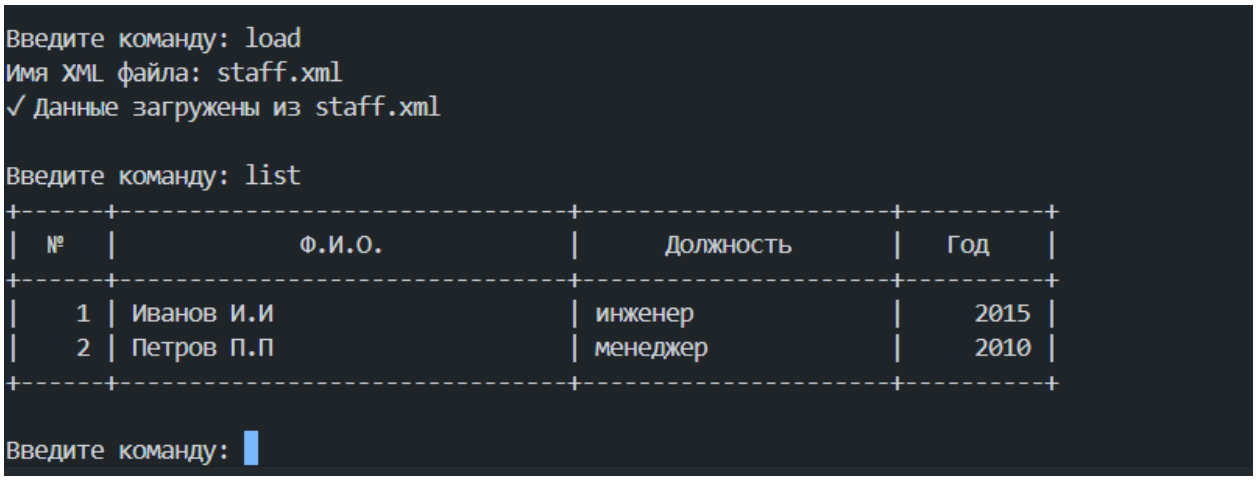

Рисунок 11 – Результат загрузки и вывода данных из сохраненного файла

6. **Задание 1:** создать CLI для списка задач: текст задачи, приоритет, статус. Реализовать добавление, вывод, выбор задач по статусу, сортировку по приоритету, загрузку и сохранение в XML файл.

Листинг 6 – CLI для списка задач с использованием agrparse

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

import argparse
import os
import xml.etree.ElementTree as ET
from dataclasses import dataclass, field
from enum import Enum


class Priority(Enum):
    LOW = 1
    MEDIUM = 2
    HIGH = 3

    def __str__(self) -> str:
        return self.name


class Status(Enum):
    NEW = "новая"
    IN_PROGRESS = "в работе"
    COMPLETED = "выполнена"
```

```python
    def __str__(self) -> str:
        return self.value


@dataclass(frozen=True)
class Task:
    text: str
    priority: Priority
    status: Status


@dataclass
class TodoList:
    tasks: list[Task] = field(default_factory=list)

    def add(self, text: str, priority: str, status: str = "новая") -> None:
        try:
            pri: Priority = Priority[priority.upper()]
        except KeyError:
            raise ValueError(f"Invalid priority: {priority}")

        try:
            st: Status = Status[status.upper().replace(" ", "_")]
        except KeyError:
            raise ValueError(f"Invalid status: {status}")

        self.tasks.append(Task(text=text, priority=pri, status=st))

    def __str__(self) -> str:
        if not self.tasks:
            return "Список задач пуст."

        table: list[str] = []
        line: str = f"+-{'-' * 3}-+-{'-' * 40}-+-{'-' * 10}-+-{'-' * 12}-+"
        table.append(line)
        table.append(
            f"| {'\u2116':^3} | {'\u0422\u0435\u043a\u0441\u0442':^40} | "
            f"{'\u041f\u0440\u0438\u043e\u0440\u0438\u0442\u0435\u0442':^10} | "
            f"{'\u0421\u0442\u0430\u0442\u0443\u0441':^12} |"
        )
        table.append(line)

        for idx, task in enumerate(self.tasks, 1):
            fmt_str: str = f"| {idx:>3} | {task.text:<40} | "
            fmt_str += f"{str(task.priority):<10} | {str(task.status):<12} |"
            table.append(fmt_str)

        table.append(line)
        return "\n".join(table)
```

```
    def select_by_status(self, status: str) -> list[Task]:
        try:
            st: Status = Status[status.upper().replace(" ", "_")]
        except KeyError:
            raise ValueError(f"Invalid status: {status}")

        return [task for task in self.tasks if task.status == st]

    def select_by_priority(self, priority: str) -> list[Task]:
        try:
            pri: Priority = Priority[priority.upper()]
        except KeyError:
            raise ValueError(f"Invalid priority: {priority}")

        return [task for task in self.tasks if task.priority == pri]

    def sort_by_priority(self) -> None:
        self.tasks.sort(key=lambda task: task.priority.value, reverse=True)

    def load(self, filename: str) -> None:
        with open(filename, "r", encoding="utf-8") as fin:
            xml: str = fin.read()

        parser: ET.XMLParser = ET.XMLParser(encoding="utf-8")
        tree: ET.Element = ET.fromstring(xml, parser=parser)

        self.tasks = []
        for task_element in tree:
            text: str | None = None
            priority: str | None = None
            status: str | None = None

            for element in task_element:
                if element.tag == "text":
                    text = element.text
                elif element.tag == "priority":
                    priority = element.text
                elif element.tag == "status":
                    status = element.text

            if text and priority and status:
                self.add(text, priority, status)

    def save(self, filename: str) -> None:
        root: ET.Element = ET.Element("tasks")

        for task in self.tasks:
            task_element: ET.Element = ET.Element("task")

            text_element: ET.Element = ET.SubElement(task_element, "text")
            text_element.text = task.text
```

```
            priority_element: ET.Element = ET.SubElement(
                task_element, "priority"
            )
            priority_element.text = task.priority.name

            status_element: ET.Element = ET.SubElement(task_element, "status")
            status_element.text = task.status.name

            root.append(task_element)

        tree: ET.ElementTree = ET.ElementTree(root)
        with open(filename, "wb") as fout:
            tree.write(fout, encoding="utf-8", xml_declaration=True)


def build_parser() -> argparse.ArgumentParser:
    parser: argparse.ArgumentParser = argparse.ArgumentParser(
        description="Система управления списком задач (dataclass + XML +
argparse)"
    )

    subparsers: argparse._SubParsersAction = parser.add_subparsers(
        dest="command"
    )

    add_parser: argparse.ArgumentParser = subparsers.add_parser(
        "add", help="Добавить задачу"
    )
    add_parser.add_argument(
        "--text", required=True, help="Текст задачи"
    )
    add_parser.add_argument(
        "--priority",
        required=True,
        choices=["low", "medium", "high"],
        help="Приоритет (low, medium, high)",
    )
    add_parser.add_argument(
        "--status",
        default="new",
        choices=["new", "in_progress", "completed"],
        help="Статус (new, in_progress, completed)",
    )

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

    select_parser: argparse.ArgumentParser = subparsers.add_parser(
        "select", help="Выбрать задачи"
    )
    select_group = select_parser.add_mutually_exclusive_group(required=True)
```

```
    select_group.add_argument(
        "--status",
        choices=["new", "in_progress", "completed"],
        help="Фильтр по статусу",
    )
    select_group.add_argument(
        "--priority",
        choices=["low", "medium", "high"],
        help="Фильтр по приоритету",
    )

    subparsers.add_parser(
        "sort", help="Отсортировать по приоритету"
    )

    load_parser: argparse.ArgumentParser = subparsers.add_parser(
        "load", help="Загрузить из XML"
    )
    load_parser.add_argument("filename", help="Имя XML файла")

    save_parser: argparse.ArgumentParser = subparsers.add_parser(
        "save", help="Сохранить в XML"
    )
    save_parser.add_argument("filename", help="Имя XML файла")

    return parser
```

Листинг 7 – Вызов программы

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

import os
from todolist import Task, TodoList


def main() -> None:
    todo_list: TodoList = TodoList()

    if os.path.exists("tasks.xml"):
        try:
            todo_list.load("tasks.xml")
        except Exception as e:
            print(f"Ошибка при загрузке: {e}")

    print("Система управления списком задач (TODO)")
    print("Команды: add, list, select, sort, load, save, exit")
    print()
```

```
    while True:
        try:
            command: str = input("Введите команду: ").strip().lower()

            if command == "exit":
                print("До свидания!")
                break

            elif command == "add":
                text: str = input("Текст задачи: ").strip()
                priority: str = input(
                    "Приоритет (low/medium/high): "
                ).strip().lower()
                status: str = input(
                    "Статус (new/in_progress/completed) [new]: "
                ).strip().lower() or "new"

                todo_list.add(text, priority, status)
                todo_list.save("tasks.xml")
                print("✓ Задача добавлена.\n")

            elif command == "list":
                print(todo_list)
                print()

            elif command == "select":
                filter_type: str = input(
                    "Выбрать по (status/priority): "
                ).strip().lower()

                if filter_type == "status":
                    status: str = input(
                        "Статус (new/in_progress/completed): "
                    ).strip().lower()
                    selected: list[Task] = (
                        todo_list.select_by_status(status)
                    )
                elif filter_type == "priority":
                    priority: str = input(
                        "Приоритет (low/medium/high): "
                    ).strip().lower()
                    selected = todo_list.select_by_priority(priority)
                else:
                    print("Неверный параметр.\n")
                    continue

                if selected:
                    print(f"\nНайдено задач: {len(selected)}\n")
                    for idx, task in enumerate(selected, 1):
                        status_str: str = str(task.status)
                        priority_str: str = str(task.priority)
```

```
                        print(
                            f"{idx}. {task.text} "
                            f"[{priority_str}, {status_str}]"
                        )
                else:
                    print("Задачи не найдены.")
                print()

            elif command == "sort":
                todo_list.sort_by_priority()
                todo_list.save("tasks.xml")
                print("✓ Задачи отсортированы по приоритету.\n")
                print(todo_list)
                print()

            elif command == "load":
                load_filename: str = input("Имя XML файла: ").strip()
                todo_list.load(load_filename)
                print(f"✓ Данные загружены из {load_filename}\n")

            elif command == "save":
                save_filename: str = input("Имя XML файла: ").strip()
                todo_list.save(save_filename)
                print(f"✓ Данные сохранены в {save_filename}\n")

            else:
                print(
                    "Неизвестная команда. "
                    "Доступные команды: add, list, select, "
                    "sort, load, save, exit\n"
                )

        except ValueError as e:
            print(f"Ошибка: {e}\n")
        except Exception as e:
            print(f"Ошибка: {e}\n")


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

```
Введите команду: add
Текст задачи: Купить продукты
Приоритет (low/medium/high): low
Статус (new/in_progress/completed) [new]: new
✓ Задача добавлена.

Введите команду: add
Текст задачи: Сделать уроки
Приоритет (low/medium/high): medium
Статус (new/in_progress/completed) [new]: in_progress
✓ Задача добавлена.

Введите команду: add
Текст задачи: Заплатить за квартиру
Приоритет (low/medium/high): high
Статус (new/in_progress/completed) [new]: new
✓ Задача добавлена.
```

Рисунок 12 – Добавление задач

```
Введите команду: list
+-----+-----------------------------------------+------------+--------------+
|  №  |                  Текст                  | Приоритет  |    Статус    |
+-----+-----------------------------------------+------------+--------------+
|   1 | Купить продукты                         | LOW        | новая        |
|   2 | Сделать уроки                           | MEDIUM     | в работе     |
|   3 | Заплатить за квартиру                   | HIGH       | новая        |
+-----+-----------------------------------------+------------+--------------+
```

Рисунок 13 – Вывод списка задач

```
Введите команду: sort
✓ Задачи отсортированы по приоритету.

+-----+-----------------------------------------+------------+--------------+
|  №  |                  Текст                  | Приоритет  |    Статус    |
+-----+-----------------------------------------+------------+--------------+
|   1 | Заплатить за квартиру                   | HIGH       | новая        |
|   2 | Сделать уроки                           | MEDIUM     | в работе     |
|   3 | Купить продукты                         | LOW        | новая        |
+-----+-----------------------------------------+------------+--------------+
```

Рисунок 14 – Сортировка задача по приоритету

```
Введите команду: save
Имя XML файла: tasks.xml
✓ Данные сохранены в tasks.xml
```

Рисунок 15 – Сохранение данных в файл

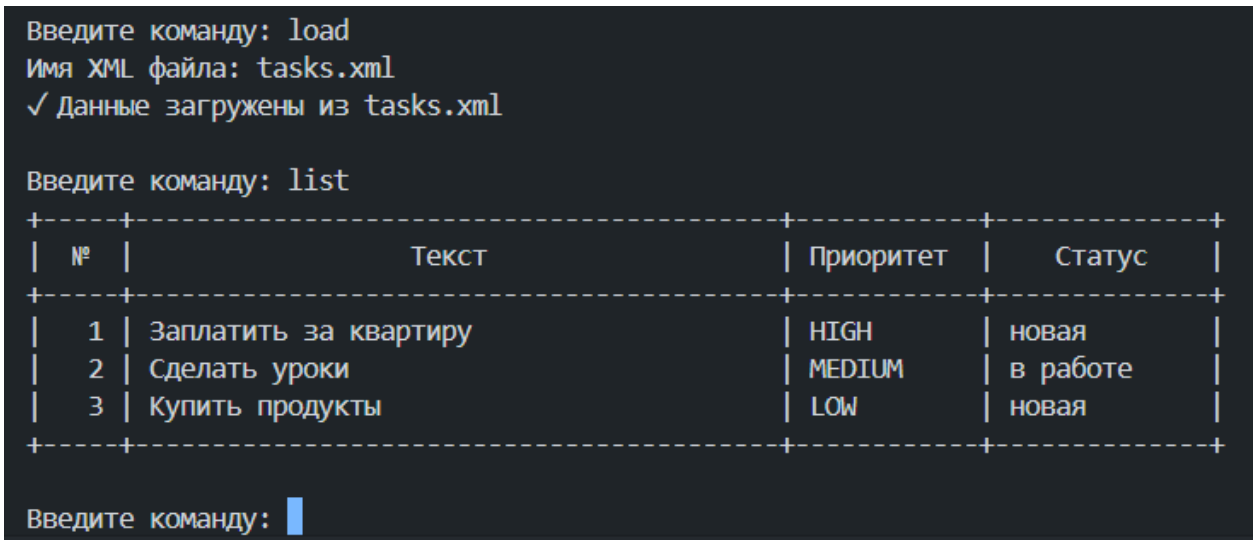

Рисунок 16 – Загрузка данных из файла

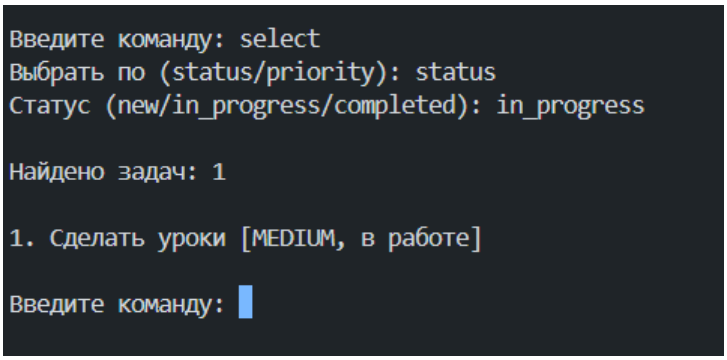

Рисунок 17 – Выбор задач по статусу

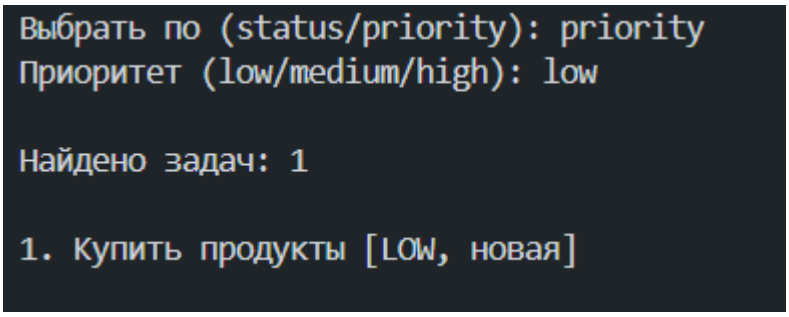

Рисунок 18 – Выбор задач по приоритету

7. **Задание 2:** Самостоятельно изучить работу с пакетом click для построения интерфейса командной строки (CLI). Для своего варианта задания 1 необходимо реализовать интерфейс командной строки с использованием пакта click.

Листинг 7 – Реализация CLI с использованием пакета click

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

import os
```

```python
from todolist import Task, TodoList

import click


@click.group()
def cli() -> None:
    pass


@cli.command()
@click.option(
    "--text", required=True, prompt="Текст задачи", help="Текст задачи"
)
@click.option(
    "--priority",
    type=click.Choice(["low", "medium", "high"]),
    required=True,
    prompt="Приоритет (low/medium/high)",
    help="Приоритет",
)
@click.option(
    "--status",
    type=click.Choice(["new", "in_progress", "completed"]),
    default="new",
    help="Статус",
)
def add(text: str, priority: str, status: str) -> None:
    todo_list: TodoList = TodoList()
    if os.path.exists("tasks.xml"):
        try:
            todo_list.load("tasks.xml")
        except Exception as e:
            click.echo(f"Ошибка при загрузке: {e}", err=True)

    try:
        todo_list.add(text, priority, status)
        todo_list.save("tasks.xml")
        click.echo("✓ Задача добавлена.")
    except ValueError as e:
        click.echo(f"Ошибка: {e}", err=True)


@cli.command()
def list() -> None:
    todo_list: TodoList = TodoList()
    if os.path.exists("tasks.xml"):
        try:
            todo_list.load("tasks.xml")
        except Exception as e:
            click.echo(f"Ошибка при загрузке: {e}", err=True)
```

```
            return

    click.echo(todo_list)


@cli.command()
@click.option(
    "--status",
    type=click.Choice(["new", "in_progress", "completed"]),
    help="Фильтр по статусу",
)
@click.option(
    "--priority",
    type=click.Choice(["low", "medium", "high"]),
    help="Фильтр по приоритету",
)
def select(status: str | None, priority: str | None) -> None:
    if not status and not priority:
        click.echo("Укажите --status или --priority", err=True)
        return

    if status and priority:
        click.echo(
            "Укажите только --status или только --priority", err=True
        )
        return

    todo_list: TodoList = TodoList()
    if os.path.exists("tasks.xml"):
        try:
            todo_list.load("tasks.xml")
        except Exception as e:
            click.echo(f"Ошибка при загрузке: {e}", err=True)
            return

    try:
        if status:
            selected: list[Task] = todo_list.select_by_status(status)
            filter_name: str = f"статусу '{status}'"
        else:
            selected = todo_list.select_by_priority(priority)
            filter_name = f"приоритету '{priority}'"

        if selected:
            click.echo(f"Найдено задач по {filter_name}: {len(selected)}\n")
            for idx, task in enumerate(selected, 1):
                status_str: str = str(task.status)
                priority_str: str = str(task.priority)
                click.echo(
                    f"{idx}. {task.text} "
                    f"[{priority_str}, {status_str}]"
```

```
                )
        else:
            click.echo(f"Задачи не найдены по {filter_name}.")
    except ValueError as e:
        click.echo(f"Ошибка: {e}", err=True)


@cli.command()
def sort() -> None:
    todo_list: TodoList = TodoList()
    if os.path.exists("tasks.xml"):
        try:
            todo_list.load("tasks.xml")
        except Exception as e:
            click.echo(f"Ошибка при загрузке: {e}", err=True)
            return

    try:
        todo_list.sort_by_priority()
        todo_list.save("tasks.xml")
        click.echo("✓ Задачи отсортированы по приоритету.")
        click.echo(todo_list)
    except Exception as e:
        click.echo(f"Ошибка: {e}", err=True)


@cli.command()
@click.argument("filename")
def load(filename: str) -> None:
    todo_list: TodoList = TodoList()
    try:
        todo_list.load(filename)
        click.echo(f"✓ Данные загружены из {filename}")
    except Exception as e:
        click.echo(f"Ошибка: {e}", err=True)


@cli.command()
@click.argument("filename")
def save(filename: str) -> None:
    todo_list: TodoList = TodoList()
    if os.path.exists("tasks.xml"):
        try:
            todo_list.load("tasks.xml")
        except Exception as e:
            click.echo(f"Ошибка при загрузке: {e}", err=True)
            return

    try:
        todo_list.save(filename)
        click.echo(f"✓ Данные сохранены в {filename}")
```

```
    except Exception as e:
        click.echo(f"Ошибка: {e}", err=True)


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

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run python tasks\task_2.py --help
Usage: task_2.py [OPTIONS] COMMAND [ARGS]...

Options:
  --help  Show this message and exit.

Commands:
  add
  list
  load
  save
  select
  sort
```

Рисунок 19 – Список возможных команд

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run python tasks\task_2.py add --text "Поменять розетку" --priority high
✓ Задача добавлена.
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run python tasks\task_2.py add --text "Выключить холодильник" --priority low
✓ Задача добавлена.
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run python tasks\task_2.py add --text "Поставить будильник" --priority medium
✓ Задача добавлена.
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10>
```

Рисунок 20 – Добавление новых задач

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run tasks\task_2.py save tasks.xml
✓ Данные сохранены в tasks.xml
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10>
```

Рисунок 21 – Сохранение данных в файл

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run tasks\task_2.py load tasks.xml
✓ Данные загружены из tasks.xml
```

Рисунок 22 – Загрузка данных из файла

```
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10> uv run tasks\task_2.py sort
✓ Задачи отсортированы по приоритету.
+-----+-----------------------------------------+------------+--------------+
|  №  |                  Текст                  | Приоритет  |    Статус    |
+-----+-----------------------------------------+------------+--------------+
|   1 | Заплатить за квартиру                   | HIGH       | новая        |
|   2 | Поменять розетку                        | HIGH       | новая        |
|   3 | Сделать уроки                           | MEDIUM     | в работе     |
|   4 | Поставить будильник                     | MEDIUM     | новая        |
|   5 | Купить продукты                         | LOW        | новая        |
|   6 | Выключить холодильник                   | LOW        | новая        |
+-----+-----------------------------------------+------------+--------------+
(oop-pract-10) PS C:\учеба\ООП\10 лаба\oop_pract_10>
```

Рисунок 23 – Сортировка задач по приоритету

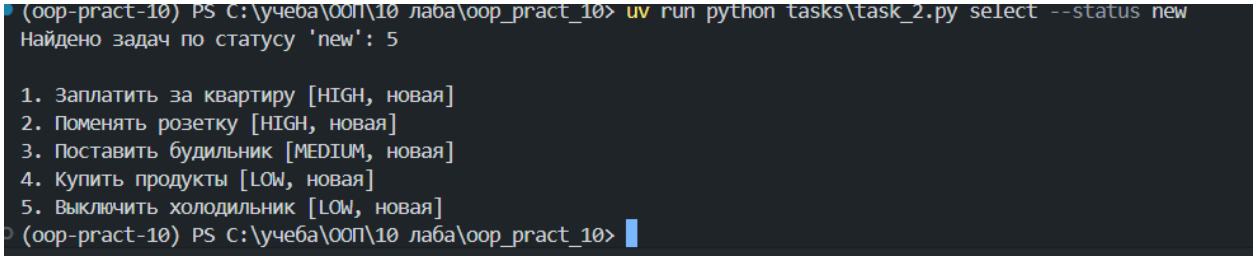

Рисунок 24 – Выбор задач по статусу

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