

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

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №20
дисциплины «Основы программной инженерии»

Выполнил:
Плугатырев Владислав Алексеевич
2 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

(подпись)

Проверил Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

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

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

Ход работы.

1. Создание репозитория.



Рисунок 1.1 – Созданный репозиторий

2. Пример из лабораторной работы.

```
def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename", action="store", help="The data file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("workers")
    parser.add_argument("--version", action="version", version="% (prog)s 0.1.0")
    subparsers = parser.add_subparsers(dest="command")
    # Создать субпарсер для добавления работника.
    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Add a new worker"
    )
    add.add_argument(
        "-n", "--name", action="store", required=True, help="The worker's name"
    )
    add.add_argument("-p", "--post", action="store", help="The worker's post")
    add.add_argument(
        "-y",
        "--year",
        action="store",
        type=int,
        required=True,
        help="The year of hiring",
    )
    # Создать субпарсер для отображения всех работников.
    display = subparsers.add_parser(
        "display", parents=[file_parser], help="Display all workers"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)
    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if os.path.exists(args.filename):
        workers = load_workers(args.filename)
    else:
        workers = []
    # Добавить работника.
    if args.command == "add":
        workers = add_worker(workers, args.name, args.post, args.year)
        is_dirty = True
    # Отобразить всех работников.
    elif args.command == "display":
        display_workers(workers)
    # Выбрать требуемых работников.
    elif args.command == "select":
        selected = select_workers(workers, args.period)
        display_workers(selected)
    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_workers(args.filename, workers)
```

Рисунок 2.1 – Пример из лабораторной работы

3. Выполнение задания: для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

```
def main(command_line=None):
    """
    Главная функция программы.
    """
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename", action="store", help="The data file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("people")
    parser.add_argument("--version", action="version", version="%(prog)s 0.1.0")

    subparsers = parser.add_subparsers(dest="command")

    # Создать субпарсер для добавления человека.
    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Add a new person"
    )
    add.add_argument(
        "-s",
        "--surname",
        action="store",
        required=True,
        help="The person's surname",
    )
    add.add_argument(
        "-n", "--name", action="store", required=True, help="The person's name"
    )
    add.add_argument(
        "-z", "--zodiac", action="store", help="The person's zodiac"
    )
    add.add_argument(
        "-b",
        "--birthday",
        action="store",
        required=True,
        help="The person's birthday",
    )

    # Создать субпарсер для отображения всех людей.
    _ = subparsers.add_parser(
        "display", parents=[file_parser], help="Display people"
    )

    # Создать субпарсер для выбора людей по фамилии.
    select = subparsers.add_parser(
        "select", parents=[file_parser], help="Select people by surname"
    )
    select.add_argument(
        "-s",
        "--surname",
        action="store",
        type=str,
        required=True,
        help="The required surname",
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    is_dirty = False
    if os.path.exists(args.filename):
        people = load_people(args.filename)
    else:
        people = []
```

Рисунок 3.1 – Код программы

```
python ind.py add data.json -s Plugatyrev -n Vladislav -z Capricorn -b 12.01.2005
```

Рисунок 3.2 – Команда

```
[
    {
        "surname": "Plugatyrev",
        "name": "Vladislav",
        "zodiac": "Capricorn",
        "birthday": [
            "12",
            "01",
            "2005"
        ]
    }
]
```

Рисунок 3.3 – Результат

4. Выполнение задания повышенной сложности: самостоятельно изучите работу с пакетом click для построения интерфейса командной строки (CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс командной строки с использованием пакета click .

```
@cli.command()
@click.argument("filename")
@click.argument("surname")
@click.argument("name")
@click.argument("zodiac")
@click.argument("birthday")
def add(filename, surname, name, zodiac, birthday):
    if os.path.exists(filename):
        people = load_people(filename)
    else:
        people = []
    people = add_person(people, surname, name, zodiac, birthday)
    people.sort(
        key=lambda x: datetime.strptime(".".join(x["birthday"]), "%d.%m.%Y")
    )
    save_people(filename, people)

@cli.command()
@click.argument("filename")
def display(filename):
    people = load_people(filename)
    display_people(people)

@cli.command()
@click.argument("filename")
@click.argument("surname")
def select(filename, surname):
    people = load_people(filename)
    select_people(surname, people)
```

Рисунок 4.1 – Код программы

```
python ind_hard.py add data.json Plugatyrev Vladislav Koz erog 12.01.2005
```

Рисунок 4.2 – Команда

```
1  [
2      {
3          "surname": "Plugatyrev",
4          "name": "Vladislav",
5          "zodiac": "Kozerog",
6          "birthday": [
7              "12",
8              "01",
9              "2005"
10         ]
11     }
12 ]
```

Рисунок 4.3 – Результат

Ответы на контрольные вопросы

1. Отличие терминала и консоли:

Термины "терминал" и "консоль" часто используются как синонимы, но исторически они имеют немного разные значения. Консоль - это физическое устройство для ввода и вывода данных, которое использовалось для непосредственного управления компьютерной системой. Терминал - это аппаратное или программное устройство для ввода команд и отображения вывода системы. В современном использовании терминалом часто называют приложение, симулирующее работу консоли, предоставляя пользователю интерфейс командной строки (CLI).

2. Консольное приложение:

Консольное приложение - это программное приложение, предназначенное для использования в интерфейсе командной строки и не требующее графического пользовательского интерфейса (GUI). Такое приложение взаимодействует с пользователем через текстовые команды в консоли или терминале и возвращает результаты операций также в текстовом виде.

3. Средства Python для построения приложений командной строки:

В Python есть несколько библиотек и модулей для создания CLI приложений:

- «sys»: Для прямого доступа к аргументам командной строки, предоставляемым через «sys.argv».
- «argparse»: Для разработки более сложных CLI интерфейсов с автоматической генерацией справки, поддержкой типов аргументов и т.д.
- «click»: Для создания CLI приложений с использованием декораторов и автоматической генерации справки.
- «optparse»: Устаревший, но раньше использовался как альтернатива «argparse».
- «getopt»: Модуль, ориентированный на стиль работы с опциями командной строки, сходный с C-библиотекой «getopt».

4. Особенности построения CLI с использованием модуля sys:

Модуль «sys» предоставляет доступ к некоторым переменным и функциям, тесно связанным с интерпретатором. Аргументы командной строки доступны через «sys.argv» как список строк, где первый элемент - это путь до выполняемого скрипта, а остальные элементы - параметры командной строки. Это базовый способ получения аргументов, требующий ручного их разбора.

5. Особенности построения CLI с использованием модуля getopt:

Модуль «getopt» помогает парсить опции командной строки и параметры в стиле GNU «getopt()». Он анализирует порядок аргументов (короткие «-f», длинные «--file») и поддерживает проверку на валидность опций. Он не создает автоматической справки и является более низкоуровневым по сравнению с «argparse».

6. Особенности построения CLI с использованием модуля argparse:

Модуль «argparse» предлагает более продвинутые возможности для создания интерфейсов командной строки. Он поддерживает позиционные аргументы, опции, автоматическую генерацию справки, валидацию типов аргументов, подкоманды и другие расширенные функции, такие как действия («store_true», «store_false», «append» и т.д.), которые значительно упрощают разработку сложных CLI интерфейсов.