Институт цифрового развития
Кафедра инфокоммуникаций

«Разработка приложений с интерфейсом командной строки (CLI) в Python3»

**ОТЧЕТ по лабораторной работе
№4 дисциплины «Основы
программной инженерии»**

Выполнил:
Сотников Андрей Александрович
2 курс, группа ПИЖ-б-о-21-1,
011.03.04 «Программная инженерия»,
направленность (профиль) «Разработка
и сопровождение программного
обеспечения», очная форма обучения

_____

(подпись)

Проверил:

_____

(подпись)

Отчет защищен с оценкой_____Дата защиты _____

Ставрополь, 2023 г.

Лабораторная работа 2.17. 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:

Ниже представлен код индивидуального задания:

```
def main(command_line=None):

    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")

    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new acc"
    )
    add.add_argument(
        "-s",
        "--s_b_a",
        action="store"
```

```python
@click.command()
@click.argument('command')
@click.argument('filename')
@click.option('--s_b_a', help='The person`s name')
@click.option('--b_a', help='The zodiac_sign')
@click.option('--t_a', help='The birth')
def main(command, filename, s_b_a, b_a, t_a):
    is_dirty = False
    if os.path.exists(filename):
        requisites = load_workers(filename)
    else:
        requisites = []

    if command == "add":
        s_b_a = click.prompt("Введите счёт отправителя: ")
        b_a = click.prompt("Введите счёт отправителя: ")
        t_a = click.prompt("Введите сумму трансфера: ")
        requisites = add_bank_acc(
            requisites,
            s_b_a,
            b_a,
            t_a
        )
        is_dirty = True

    elif command == "display":
        display_accs(requisites)
```

Вывод: были приобретены навики построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.

Контрольные вопросы

1. В чем отличие терминала и консоли?

Терминал (от лат. terminus — граница) — устройство или ПО, выступающее посредником между человеком и вычислительной системой. Обычно данный термин используется, когда точка доступа к системе вынесена в отдельное

физическое устройство и предоставляет свой пользовательский интерфейс на основе внутреннего интерфейса (например, сетевых протоколов).

Консоль console — исторически реализация терминала с клавиатурой и текстовым дисплеем. В настоящее время это слово часто используется как синоним сеанса работы или окна оболочки командной строки. В том же смысле иногда применяется и слово “терминал”.

2. Что такое консольное приложение?

Консольное приложение console application — вид ПО, разработанный с расчётом на работу внутри оболочки командной строки, т.е. опирающийся на текстовый ввод-вывод.

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

Python 3 поддерживает несколько различных способов обработки аргументов командной строки. Встроенный способ – использовать модуль sys . С точки зрения имен и использования, он имеет прямое отношение к библиотеке C (libc). Второй способ – это модуль getopt , который обрабатывает как короткие, так и длинные параметры, включая оценку значений параметров. Кроме того, существуют два других общих метода. Это модуль argparse , производный от модуля optparse, доступного до Python 2.7. Другой метод – использование модуля docopt , доступного на GitHub.

4. Какие особенности построение CLI с использованием модуля sys ? Это базовый модуль, который с самого начала поставлялся с Python. Он использует подход, очень похожий на библиотеку C, с использованием argc

и argv для доступа к аргументам. Модуль sys реализует аргументы командной строки в простой структуре списка с именем sys.argv .

Каждый элемент списка представляет собой единственный аргумент. Первый элемент в списке sys.argv [0] – это имя скрипта Python. Остальные элементы списка, от sys.argv [1] до sys.argv [n] , являются аргументами командной строки с 2 по n. В качестве разделителя между аргументами используется пробел. Значения аргументов, содержащие пробел, должны быть заключены в кавычки, чтобы их правильно проанализировал sys .

Эквивалент argc – это просто количество элементов в списке. Чтобы получить это значение, используйте оператор len() . Позже мы покажем это на примере кода.

5. Какие особенности построение CLI с использованием модуля getopt ? Как вы могли заметить ранее, модуль sys разбивает строку командной строки только на отдельные фасеты. Модуль getopt в Python идет немного дальше и расширяет разделение входной строки проверкой параметров. Основанный на функции C getopt , он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений.

На практике для правильной обработки входных данных требуется модуль sys . Для этого необходимо заранее загрузить как модуль sys , так и модуль getopt . Затем из списка входных параметров мы удаляем первый элемент списка (см. код ниже) и сохраняем оставшийся список аргументов командной строки в переменной с именем arguments_list .

6. Какие особенности построение CLI с использованием модуля argparse? Начиная с версий Python 2.7 и Python 3.2, в набор стандартных библиотек

была включена библиотека argparse для обработки аргументов (параметров, ключей) командной строки. Хотелось бы остановить на ней Ваше внимание. Для начала рассмотрим, что интересного предлагает argparse :

- анализ аргументов sys.argv ;
- конвертирование строковых аргументов в объекты Вашей программы и работа с ними;
- форматирование и вывод информативных подсказок.

Одним из аргументов противников включения argparse в Python был довод о том, что в стандартных модулях и без этого содержится две библиотеки для семантической обработки (парсинга) параметров командной строки. Однако, как заявляют разработчики argparse , библиотеки getopt и optparse уступают argparse по нескольким причинам:

- обладая всей полнотой действий с обычными параметрами командной строки, они не умеют обрабатывать позиционные аргументы (positional arguments). Позиционные аргументы — это аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в эту программу передаются. Простейший пример — программа cp, имеющая минимум 2 таких аргумента («cp source destination»).
- argparse дает на выходе более качественные сообщения о подсказке при минимуме затрат (в этом плане при работе с optparse часто можно наблюдать некоторую избыточность кода);
- argparse дает возможность программисту устанавливать для себя, какие символы являются параметрами, а какие нет. В отличие от него, optparse считает опции с синтаксисом наподобие "-pf, -file, +rgb, /f и т.п. «внутренне противоречивыми» и «не поддерживается optpars 'ом и никогда не будет»;

- argparse даст Вам возможность использовать несколько значений переменных у одного аргумента командной строки (nargs);
- argparse поддерживает субкоманды (subcommands). Это когда основной парсер отсылает к другому (субпарсеру), в зависимости от аргументов на входе.