МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций Институт цифрового развития

ОТЧЁТ

по лабораторной работе № 12

Дисциплина: «Программирование на Python»

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

Выполнил: студент 2 курса

группы ИВТ-б-о-21-1

Толубаев Рамиль Ахметович

Ставрополь 2022

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

Порядок выполнения работы:

1.Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.

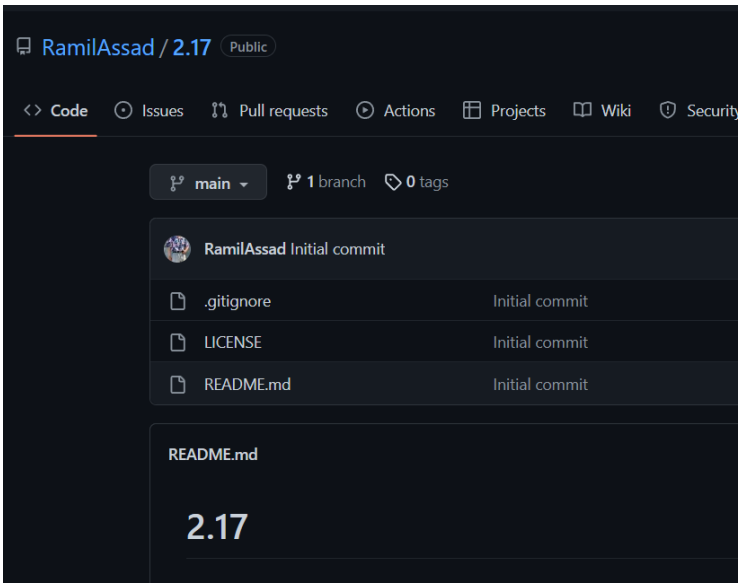


Рисунок 1. Создание репозитория

```
C:\Users\User>cd C:\Users\User\Desktop\2 курс Python\lab 20

C:\Users\User\Desktop\2 курс Python\lab 20>git clone https://github.com/aikanyshkaukanbekova/2.17.git
Cloning into '2.17'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 4.51 KiB | 769.00 KiB/s, done.
Resolving deltas: 100% (1/1), done.

C:\Users\User\Desktop\2 курс Python\lab 20>
```

Рисунок 2. Клонирование репозитория

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

import ...


def add_worker(staff, name, post, year):
    """
    Добавить данные о работнике.
    """
    staff.append(
        {
            "name": name,
            "post": post,
            "year": year
        }
    )
    return staff


def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
```

```python
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "No",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )
        print(line)

    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 рааботников.
    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)


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

Рисунок 3. Примеры лабораторной работы

Индивидуальное задание

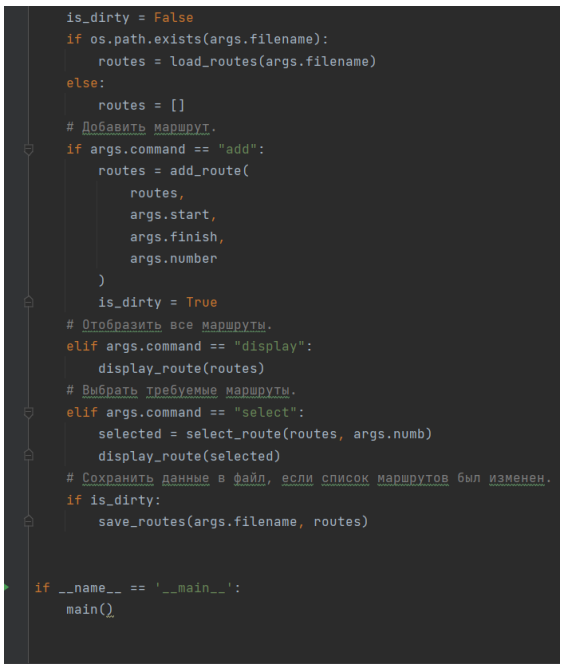
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import argparse
import json
import os.path


def add_route(routes, start, finish, number):
    """
    Добавить данные о маршруте
    """

    routes.append(
        {
            'start': start,
            'finish': finish,
            'number': number
        }
    )
    return routes


def display_route(routes):
    """
    Отобразить списко маршрутов
    """
    if routes:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
```



```
    is_dirty = False
    if os.path.exists(args.filename):
        routes = load_routes(args.filename)
    else:
        routes = []
    # Добавить маршрут.
    if args.command == "add":
        routes = add_route(
            routes,
            args.start,
            args.finish,
            args.number
        )
        is_dirty = True
    # Отобразить все маршруты.
    elif args.command == "display":
        display_route(routes)
    # Выбрать требуемые маршруты.
    elif args.command == "select":
        selected = select_route(routes, args.numb)
        display_route(selected)
    # Сохранить данные в файл, если список маршрутов был изменен.
    if is_dirty:
        save_routes(args.filename, routes)


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

Рисунок 4. Индивидуальное задание

Задача повышенной сложности

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

import ...


@click.group()
def cl():
    pass


@cl.command()
@click.argument("routes")
@click.option("-s", "--start")
@click.option("-f", "--finish")
@click.option("-n", "--number")
def add_route(routes, start, finish, number):
    """
    Добавить данные о маршруте
    """
    routes = load_routes(routes)
    routes.append(
        {
            'start': start,
            'finish': finish,
            'number': number
        }
    )
    with open(routes, "w", encoding="utf-8") as file_out:
        json.dump(routes, file_out, ensure_ascii=False, indent=4)
```

```
        print(line)
    else:
        print("Список маршрутов пуст.")


@cl.command()
@click.argument('routes')
@click.option("-N", "--numb")
def select_route(routes, period):
    """
    Выбрать маршрут
    """
    result = []
    for employee in routes:
        if employee.get('number') == period:
            result.append(employee)

    return result


def load_routes(routes):
    with open(routes, "r", encoding="utf-8") as f_in:
        return json.load(f_in)


def main():
    cl()
```

Рисунок 5. Задача повышенной сложности

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().

5. Какие особенности построение CLI с использованием модуля getopt?

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.

Аргументы в списке аргументов теперь можно анализировать с помощью метода getopts() . Но перед этим нам нужно сообщить getopts() о том, какие параметры допустимы.

Для метода getopt() необходимо настроить три параметра – список фактических аргументов из argv, а также допустимые короткие и длинные параметры.

Сам вызов метода хранится в инструкции try - catch , чтобы скрыть ошибки во время оценки. Исключение возникает, если обнаруживается аргумент, который не является частью списка, как определено ранее. Скрипт в Python выведет сообщение об ошибке на экран и выйдет с кодом ошибки 2.

Наконец, аргументы с соответствующими значениями сохраняются в двух переменных с именами arguments и values. Теперь вы можете легко оценить эти переменные в своем коде. Мы можем использовать цикл for для перебора списка распознанных аргументов, одна запись за другой.

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.

Для начала работы с argparse необходимо задать парсер. Далее, парсеру стоит указать, какие объекты Вы от него ждете.

Если действие (action) для данного аргумента не задано, то по умолчанию он будет сохраняться (store) в namespace, причем мы также можем указать тип этого аргумента (int, boolean и тд). Если имя возвращаемого аргумента (dest) задано, его значение будет сохранено в соответствующем атрибуте namespace.

Остановимся на действиях (actions). Они могут быть следующими: store: возвращает в пространство имен значение (после необязательного

приведения типа). Как уже говорилось, store — действие по умолчанию; store_const: в основном используется для флагов. Либо вернет Вам

значение, указанное в const, либо (если ничего не указано), None.

store_true / store_false: аналог store_const , но для булевых True и False ; append: возвращает список путем добавления в него значений

агрументов.

append_const: возвращение значения, определенного в спецификации аргумента, в список.

count: как следует из названия, считает, сколько раз встречается значение данного аргумента.

Вывод: были приобретены навыки по работе с данными формата JSON при написании программ с помощью языка программирования Python версии 3.x.