МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ

Федеральное государственное автономное

образовательное учреждение высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра инфокоммуникаций

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

Отчет по лабораторной работе № 2.17

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Кучеренко С. Ю. _____« » 2023г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____

(подпись)

Ставрополь 2023

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствие с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных, вводимых с клавиатуры.
8.

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

import argparse
import json
import os.path
import sys
from datetime import date


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,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
            "№",
            "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:
        print("Список работников пуст.")


def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
    today = date.today()

    # Сформировать список работников.
    result = []
    for employee in staff:
        if today.year - employee.get('year', today.year) >= period:
            result.append(employee)

    # Возвратить список выбранных работников.
        return result


def save_workers(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """

    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кирилицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)
```

```
def load_workers(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)


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.
    _ = subparsers.add_parser(
        "display",
```

```
        parents=[file_parser],
        help="Display all workers"
    )

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-P",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )

    # 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 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()
```

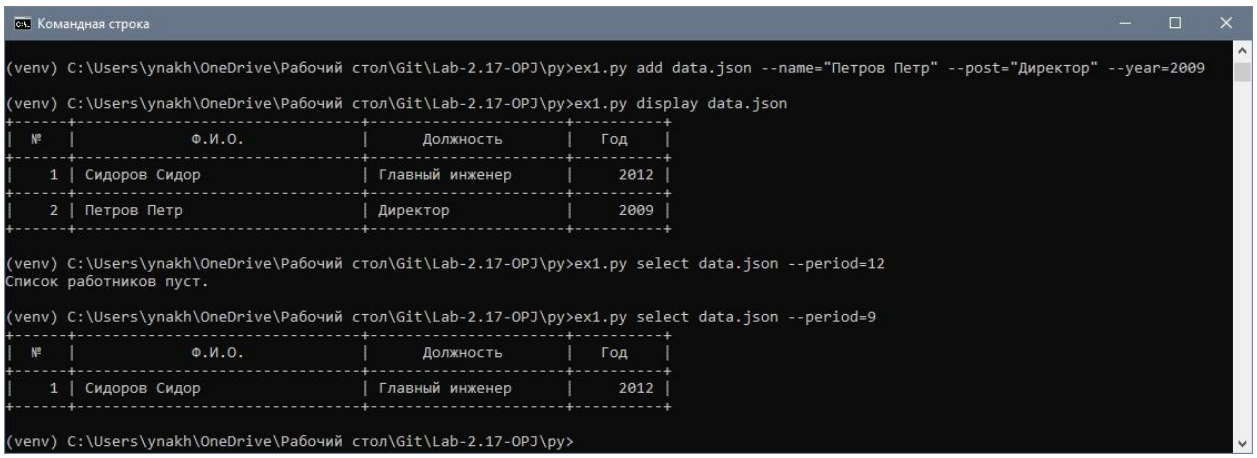

Рисунок 1 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.

10. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.

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

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

import argparse
import json
import os.path
import sys


def get_way(ways, start, finish, num):
    """
    Запросить данные о маршруте.
    """
    # Создать словарь.
    ways.append(
        {
            'start': start,
            'finish': finish,
            'num': num,
        }
    )
    return ways

def display_way(numbers):
    """
    Отобразить список маршрутов.
    """
    if numbers:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
```

```
            '-' * 30,
            '-' * 30,
            '-' * 15
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^30} | {:^15} |'.format(
                "No",
                "Название начального маршрута",
                "Название конечного маршрута",
                "Номер маршрута"
            )
        )
        print(line)

        # Вывести данные о всех маршрутах.
        for idx, way in enumerate(numbers, 1):
            print(
                '| {:>4} | {:<30} | {:<30} | {:>15} |'.format(
                    idx,
                    way.get('start', ''),
                    way.get('finish', ''),
                    way.get('num', 0)
                )
            )
        print(line)

    else:
        print("Список пуст.")


def find_way(numbers, nw):
    """
    Выбрать маршрут с данным номером.
    """
    # Список маршрутов
    result = []
    for h in numbers:
        if nw in str(h.values()):
            result.append(h)

    # Возвратить список выбранных маршрутов.
    return result


def save_ways(file_name, ways):
    """
    Сохранить номера всех маршрутов в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кирилицы установим ensure_ascii=False
        json.dump(ways, fout, ensure_ascii=False, indent=4)


def load_ways(file_name):
    """
    Загрузить все маршруты из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)
```

```
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("ways")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

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

    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new student"
    )

    add.add_argument(
        "-s",
        "--start",
        action="store",
        required=True,
        help="Start Route"
    )

    add.add_argument(
        "-f",
        "--finish",
        action="store",
        help="Final Route"
    )

    add.add_argument(
        "-n",
        "--num",
        action="store",
        required=True,
        help="Route number"
    )

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

    # Создать субпарсер для поиска маршрутов.
    find = subparsers.add_parser(
        "find",
        parents=[file_parser],
        help="find the ways"
    )
```

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

    # Загрузить все маршруты из файла, если файл существует.
    is_dirty = False
    if os.path.exists(args.filename):
        ways = load_ways(args.filename)
    else:
        ways = []

    # Добавить маршрут.
    if args.command == "add":
        ways = get_way(
            ways,
            args.start,
            args.finish,
            args.num
        )
        is_dirty = True

    # Отобразить всех студентов.
    elif args.command == "display":
        display_way(ways)

    # Выбрать требуемых студентов.
    elif args.command == "find":
        found = find_way(ways)
        display_way(found)

    # Сохранить данные в файл, если список студентов был изменен.
    if is_dirty:
        save_ways(args.filename, ways)


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

```
(venv) (base) svetik@MacBook-Air-Svetik PyCharm % python ind.py add data.json --start="lalalal"
--finish="djkdkkd" --num=55

(venv) (base) svetik@MacBook-Air-Svetik PyCharm % python ind.py display data.json
+------+--------------------------------+--------------------------------+-----------------+
|  No  |  Название начального маршрута  |  Название конечного маршрута   | Номер маршрута  |
+------+--------------------------------+--------------------------------+-----------------+
|    1 |                                |                                |               0 |
|    2 | lalalal                        | djkdkkd                        |              55 |
+------+--------------------------------+--------------------------------+-----------------+
(venv) (base) svetik@MacBook-Air-Svetik PyCharm %
```

Рисунок 2 – Результат работы программы

Задание 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.

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

import argparse
import json
import os.path
import sys


def get_way(ways, start, finish, num):
    """
    Запросить данные о маршруте.
    """
    # Создать словарь.
    ways.append(
        {
            'start': start,
            'finish': finish,
            'num': num,
        }
    )
    return ways


def display_way(numbers):
    """
    Отобразить список маршрутов.
    """
    if numbers:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 30,
            '-' * 15
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^30} | {:^15} |'.format(
                "No",
                "Название начального маршрута",
                "Название конечного маршрута",
                "Номер маршрута"
            )
        )
        print(line)

        # Вывести данные о всех маршрутах.
        for idx, way in enumerate(numbers, 1):
            print(
                '| {:>4} | {:<30} | {:<30} | {:>15} |'.format(
                    idx,
                    way.get('start', ''),
                    way.get('finish', ''),
                    way.get('num', 0)
                )
            )
        print(line)
```

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


def find_way(numbers, nw):
    """
    Выбрать маршрут с данным номером.
    """
    # Список маршрутов
    result = []
    for h in numbers:
        if nw in str(h.values()):
            result.append(h)

    # Возвратить список выбранных маршрутов.
    return result


def save_ways(file_name, ways):
    """
    Сохранить номера всех маршрутов в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кирилицы установим ensure_ascii=False
        json.dump(ways, fout, ensure_ascii=False, indent=4)


def load_ways(file_name):
    """
    Загрузить все маршруты из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)


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("ways")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )

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

    # Создать субпарсер для добавления маршрута.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new student"
    )
```

```
    add.add_argument(
        "-s",
        "--start",
        action="store",
        required=True,
        help="Start Route"
    )

    add.add_argument(
        "-f",
        "--finish",
        action="store",
        help="Final Route"
    )

    add.add_argument(
        "-n",
        "--num",
        action="store",
        required=True,
        help="Route number"
    )

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

    # Создать субпарсер для поиска маршрутов.
    find = subparsers.add_parser(
        "find",
        parents=[file_parser],
        help="find the ways"
    )

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

    # Загрузить все маршруты из файла, если файл существует.
    is_dirty = False
    if os.path.exists(args.filename):
        ways = load_ways(args.filename)
    else:
        ways = []

    # Добавить маршрут.
    if args.command == "add":
        ways = get_way(
            ways,
            args.start,
            args.finish,
            args.num
        )
        is_dirty = True

    # Отобразить всех студентов.
    elif args.command == "display":
        display_way(ways)

    # Выбрать требуемых студентов.
    elif args.command == "find":
        found = find_way(ways)
```

```
        display_way(found)

    # Сохранить данные в файл, если список студентов был изменен.
    if is_dirty:
        save_ways(args.filename, ways)


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

10. Зафиксируйте сделанные изменения в репозитории.

Вопросы для защиты работы:

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 – компьютер с клавиатурой и монитором.

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 ? 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 . # Include standard modules import getopt, sys

Get full command-line arguments full_cmd_arguments
= sys.argv

Keep all but the first argument_list =
full_cmd_arguments[1:] print(argument_list)

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

short_options = "ho:v"

long_options = ["help", "output=", "verbose"]

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. try:

arguments, values = getopt.getopt(argument_list,
short_options, long_options) except getopt.error as err:

Output error, and return with an error
code print(str(err)) sys.exit(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.

```
# Evaluate given options for current_argument,
current_value in arguments: if current_argument in ("-
v", "--verbose"):
print("Enabling verbose mode") elif
current_argument in ("-h", "--help"):
print("Displaying help") elif current_argument in ("-o",
"--output"): print(f"Enabling special output mode
({current_value})")
```

Ниже вы можете увидеть результат выполнения этого кода. Далее показано, как программа реагирует как на допустимые, так и на недопустимые программные аргументы:

```
$ python arguments-getopt.py -h
Displaying help
$ python arguments-getopt.py --help
Displaying help
$ python arguments-getopt.py --output=green --help -v
Enabling special output mode (green)
Displaying help
Enabling verbose mode
$ python arguments-getopt.py -verbose option
-e not recognized
```

Последний вызов нашей программы поначалу может показаться немного запутанным. Чтобы понять это, вам нужно знать, что сокращенные параметры (иногда также называемые флагами) могут использоваться вместе с одним тире. Это позволяет вашему инструменту легче воспринимать множество вариантов.

6. Какие особенности построение CLI с использованием модуля argparse?

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 зависимостиот аргументов на входе.