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

Пуценко И. А. « » 2022г.

Подпись студента_____

Работа защищена « »_____20__г.

Проверил Воронкин Р.А. _____ (подпись)

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

Выполнение работы:

1. Изучить теоретический материал работы.

2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT 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. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python. Зафиксируйте изменения в репозитории.

8. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных вводимых с клавиатуры.

9. Зафиксируйте сделанные изменения в репозитории.

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

11. Зафиксируйте сделанные изменения в репозитории.

12. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксируйте изменения.

13. Выполните слияние ветки для разработки с веткой master/main.

14. Отправьте сделанные изменения на сервер GitHub.

15. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Проработка примера:

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

import argparse
import json
import os.path
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(
                "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:
        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()
```

Листинг 1 – Код для примера № 1 лабораторной работы

Самостоятельные задания

Задание 1.

Задание

Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

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

import json
import os
import jsonschema
from jsonschema import validate
import argparse


def load_data():
    data = []
    if os.path.exists(data_file):
        with open(data_file, "r") as file:
            data = json.load(file)
            validate(data, schema)
    return data


def save_data(data):
    validate(data, schema)
    with open(data_file, "w") as file:
        json.dump(data, file, indent=4)


def exit_to_program():
    print('всего доброго')
    save_data(lst_planes)
    return exit(1)


def help_program():
    print("add - добавление рейса\n"
          "help - помощь по командам\n"
          "select \"пункт назначения\" - вывод самолетов летящих в п.н.\n"
          "display_plane - вывод всех самолетов\n"
          "exit - выход из программы")


def add_program(planes):
    plane = dict()
    plane["destination"] = input("Пункт назначения:\n")
    plane["flight_number"] = int(input("Номер рейса:\n"))
    plane["type_plane"] = input("Тип самолета\n")
```

```
    planes.append(plane)
    planes.sort(key=lambda key_plane: key_plane.get("flight_number"))
    return planes


def select_program(planes):
    lst = list(map(lambda x: x.get("destination"), planes))
    point = input('выберите нужное вам место\n')
    print("результаты поиска")
    if point in lst:
        print('рейсы в эту точку')
        for i in planes:
            if point == i["destination"]:
                print(f"{i['flight_number']}........{i['type_plane']}")
    else:
        print("рейсов не найдено")


def error():
    print('неверная комманда')


def display_plane(staff):
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
                '-' * 4,
                '-' * 30,
                '-' * 20,
                '-' * 8
            )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Направление",
                "Тип самолета",
                "рейс"
            )
        )
        print(line)

        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('destination', ''),
                    worker.get('type_plane', ''),
                    worker.get('flight_number', 0)
                )
            )
            print(line)

    else:
        print("рейсов не найдено")


def menu(lst_plane):
    command = input('введите команду("help" - руководство по
командам)\n>>>').lower()
    if command == 'exit':
        exit_to_program()
    elif command == 'help':
        help_program()
    elif command == 'add':
```

```
        lst_plane = add_program(lst_plane)
    elif command == 'select':
        select_program(lst_plane)
    elif command == 'display_plane':
        display_plane(lst_plane)
    else:
        error()


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Программа управления
рейсами самолетов')
    parser.add_argument('--file', help='Путь к файлу JSON для сохранения и
чтения данных')
    args = parser.parse_args()

    data_file = args.file if args.file else input("введите расположение
файла: ")

    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "destination": {"type": "string"},
                "flight_number": {"type": "integer"},
                "type_plane": {"type": "string"}
            },
            "required": ["destination", "flight_number", "type_plane"]
        }
    }

    lst_planes = load_data()

    while True:
        menu(lst_planes)
```

Листинг 2 – Код задания для ЛР №2.17

Задание 2.

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 json
import os
import jsonschema
from jsonschema import validate
import click
```

```
@click.command()
@click.option('--file', help='Путь к файлу JSON для сохранения и чтения
данных')
def main(file):
    data_file = file if file else click.prompt('Введите расположение файла:',
type=str)
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "destination": {"type": "string"},
                "flight_number": {"type": "integer"},
                "type_plane": {"type": "string"}
            },
            "required": ["destination", "flight_number", "type_plane"]
        }
    }

    lst_planes = load_data(data_file, schema)

    while True:
        menu(lst_planes, data_file, schema)


def load_data(data_file, schema):
    data = []
    if os.path.exists(data_file):
        with open(data_file, "r") as file:
            data = json.load(file)
            validate(data, schema)
    return data


def save_data(data, data_file, schema):
    validate(data, schema)
    with open(data_file, "w") as file:
        json.dump(data, file, indent=4)


def exit_to_program(data, data_file, schema):
    print('Всего доброго!')
    save_data(data, data_file, schema)
    exit(1)


def help_program():
    print("add - добавление рейса\n"
          "help - помощь по командам\n"
          "select \"пункт назначения\" - вывод самолетов летящих в п.н.\n"
          "display_plane - вывод всех самолетов\n"
          "exit - выход из программы")


def add_program(planes):
    plane = dict()
    plane["destination"] = click.prompt("Пункт назначения:")
    plane["flight_number"] = int(click.prompt("Номер рейса:", type=int))
    plane["type_plane"] = click.prompt("Тип самолета:")
    planes.append(plane)
    planes.sort(key=lambda key_plane: key_plane.get("flight_number"))
    return planes
```

```python
def select_program(planes):
    lst = list(map(lambda x: x.get("destination"), planes))
    point = click.prompt('Выберите нужное вам место:')
    print("Результаты поиска")
    if point in lst:
        print('Рейсы в эту точку')
        for i in planes:
            if point == i["destination"]:
                print(f"{i['flight_number']}.........{i['type_plane']}")
    else:
        print("Рейсов не найдено")


def error():
    print('Неверная команда')


def display_plane(staff):
    if staff:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Направление",
                "Тип самолета",
                "Рейс"
            )
        )
        print(line)

        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('destination', ''),
                    worker.get('type_plane', ''),
                    worker.get('flight_number', 0)
                )
            )
            print(line)

    else:
        print("Рейсов не найдено")


def menu(lst_plane, data_file, schema):
    command = click.prompt('Введите команду ("help" - руководство по
командам):').lower()
    if command == 'exit':
        exit_to_program(lst_plane, data_file, schema)
    elif command == 'help':
        help_program()
    elif command == 'add':
        lst_plane = add_program(lst_plane)
    elif command == 'select':
        select_program(lst_plane)
    elif command == 'display_plane':
        display_plane(lst_plane)
```

```
    else:
        error()


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

Листинг 3 – Задание повышенной сложности для ЛР 2.17

Вопросы для защиты работы

1. В чём отличие терминала от 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?

   Модули: sys, getopt, argparse, click

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.
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.