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 ПРАКТИЧЕСКОЙ РАБОТЕ №3
дисциплины «Анализ данных»
Вариант 23

Выполнил:
Мотовилов Вадим Борисович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»,
направленность (профиль)
«Информатика и вычислительная
техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 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.

Порядок выполнения работы:

1. Создал репозиторий и скопировал его

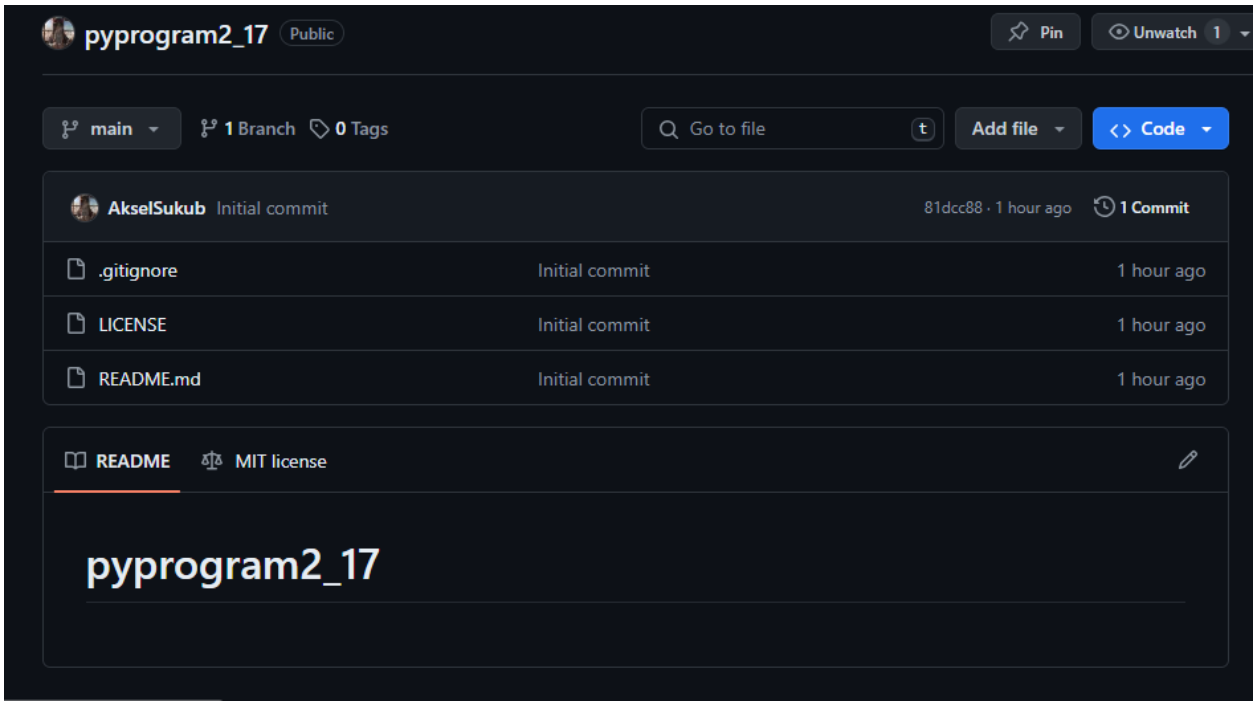

Рисунок 1. Созданный репозиторий

2. Изменил файл .gitignore и README.rm и добавил git flow

```
/site

# mypy
.mypy_cache/
.dmypy.json
dmypy.json

# Pyre type checker
.pyre/

# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

# PyCharm
#  JetBrains specific template is maintained in a separate JetBrains.gitignore that can
#  be found at https://github.com/github/gitignore/blob/main/Global/JetBrains.gitignore
#  and can be added to the global gitignore or merged into this file.  For a more nuclear
#  option (not recommended) you can uncomment the following to ignore the entire idea folder.
.idea/
**/.DS_Store
.vscode
*.dSYM
```

Рисунок 2. Измененный файл .gitignore

3. Выполнил задания

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

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

from datetime import date
import json
import argparse
import os.path
from jsonschema import validate, ValidationError
import sys

def get_worker(staff, point, number, type):
    """
    Запросить данные о рейсе.
    """
    # Создать словарь.
    staff.append(
        {
            'point': point,
```

```
            'number': number,
            'type': type
        }
    )
    return staff

def display_workers(staff):
    """
    Отобразить список рейсов.
    """
    # Проверить, что список рейсов не пуст.
    if staff:
        # Заголовок таблицы.
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4, "-" * 30, "-" * 10, "-" * 20
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^10} | {:^20} |".format(
                "No", "Пункт назначения", "No рейса", "Тип самолета"
            )
        )
        print(line)
        # Вывести данные о всех рейсах.
        for idx, worker in enumerate(staff, 1):
            print(
                "| {:>4} | {:<30} | {:<10} | {:>20} |".format(
                    idx,
                    worker.get("point", ""),
                    worker.get("number", 0),
                    worker.get("type", ""),
                )
            )
        print(line)
    else:
        print("Список рейсов пуст.")

def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
    today = date.today()
    # Сформировать список рейсов.
    result = []
    for employee in staff:
        if today.year - employee.get("year", today.year) >= period:
            result.append(employee)
    # Возвратить список выбранных рейсов.
    return result

```

```python
def save_workers(file_name, staff):
    """
    Сохранить все рейсы 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 все рейсы из файла JSON.
    """
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "point": {"type": "string"},
                "number": {"type": "integer"},
                "type": {"type": "string"},
            },
            "required": [
                "point",
                "number",
                "type",
                ],
            },
        }
        # Проверить, существует ли файл
        if os.path.exists(file_name):
            # Открыть файл с заданным именем для чтения.
            with open(file_name, "r", encoding="utf-8") as fin:
                data = json.load(fin)

            try:
                # Валидация
                validate(instance=data, schema=schema)
                print("JSON валиден по схеме.")
            except ValidationError as e:
                print(f"Ошибка валидации: {e.message}")
            return data
        else:
            print(f"Файл {file_name} не найден.")
            sys.exit(1)

  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("staff")
    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 staff"
    )
    add.add_argument(
        "-p",
        "--point",
        action="store",
        required=True,
        help="The start of the staff"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="The finish of the staff"
    )
    add.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="The number of the staff"
    )
    # Создать субпарсер для отображения всех маршрутов.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all staff"
    )
    # Создать субпарсер для выбора маршрута.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the staff"
```

```
    )
    select.add_argument(
        "-p",
        "--period",
        action="store",
        type=int,
        required=True,
        help="The staff"
    )

    # 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):
        staff = load_workers(args.filename)
    else:
        staff = []

    # Добавить маршрут.
    if args.command == "add":
        staff = get_worker(
            staff,
            args.point,
            args.number,
            args.type
        )
        is_dirty = True

    # Отобразить все маршруты.
    elif args.command == "display":
        display_workers(staff)

    # Выбрать требуемые маршруты.
    elif args.command == "select":
        selected = select_workers(staff, args.period)
        display_workers(selected)

    # Сохранить данные в файл, если список маршрутов был изменен.
    if is_dirty:
        save_workers(args.filename, staff)

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

```
(base) C:\otkat>python C:\otkat\prog3\ind1.py add -p Moscow -n 43 -t Mig-2 C:\otkat\lab3ind.json
JSON валиден по схеме.

(base) C:\otkat>python C:\otkat\prog3\ind1.py load C:\otkat\lab3ind.json
usage: staff [-h] [--version] {add,display,select} ...
staff: error: argument command: invalid choice: 'load' (choose from 'add', 'display', 'select')

(base) C:\otkat>python C:\otkat\prog3\ind1.py display C:\otkat\lab3ind.json
JSON валиден по схеме.
+------+--------------------------------+------------+----------------------+
|  No  |       Пункт назначения         |  No рейса  |     Тип самолета     |
+------+--------------------------------+------------+----------------------+
|    1 | Moscow                         | 123        |          Boeing 737 |
|    2 | SPb                            | 456        |          Airbus A320 |
|    3 | Sochi                          | 789        |  Sukhoi Superjet 100 |
|    4 | Moscow                         | 43         |               Mig-2 |
+------+--------------------------------+------------+----------------------+
```

Рисунок 3 . Выполнение 1 задания

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`.

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

from datetime import date
import json
import click
import os.path
from jsonschema import validate, ValidationError
import sys


def display_workers(staff):
    """
    Отобразить список рейсов.
    """
    # Проверить, что список рейсов не пуст.
    if staff:
        # Заголовок таблицы.
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4, "-" * 30, "-" * 10, "-" * 20
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^10} | {:^20} |".format(
                "No", "Пункт назначения", "No рейса", "Тип самолета"
            )
        )
        print(line)
        # Вывести данные о всех рейсах.
        for idx, worker in enumerate(staff, 1):
```

```
            print(
                "| {:>4} | {:<30} | {:<10} | {:>20} |".format(
                    idx,
                    worker.get("point", ""),
                    worker.get("number", 0),
                    worker.get("type", ""),
                )
            )
        print(line)
    else:
        print("Список рейсов пуст.")


def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
    today = date.today()
    # Сформировать список рейсов.
    result = []
    for employee in staff:
        if today.year - employee.get("year", today.year) >= period:
            result.append(employee)
    # Возвратить список выбранных рейсов.
    return result


def save_workers(file_name, staff):
    """
    Сохранить все рейсы 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 все рейсы из файла JSON.
    """
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "point": {"type": "string"},
                "number": {"type": "integer"},
                "type": {"type": "string"},
```

```
            },
            "required": [
                "point",
                "number",
                "type",
            ],
        },
    }
    # Проверить, существует ли файл
    if os.path.exists(file_name):
        # Открыть файл с заданным именем для чтения.
        with open(file_name, "r", encoding="utf-8") as fin:
            data = json.load(fin)

        try:
            # Валидация
            validate(instance=data, schema=schema)
            print("JSON валиден по схеме.")
        except ValidationError as e:
            print(f"Ошибка валидации: {e.message}")
        return data
    else:
        print(f"Файл {file_name} не найден.")
        sys.exit(1)

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


@commands.command("add")
@click.argument("filename")
@click.option("--point", help="point")
@click.option("--number", help="number")
@click.option("--type", help="type")
def add(filename, point, number, type):
    """
    Добавить данные о маршруте
    """
    staff = load_workers(filename)
    route = {
        "point": point,
        "number": number,
        "type": type,
    }
    staff.append(route)
    save_workers(filename, staff)


@commands.command("display")
@click.argument("filename")
```

```python
def display(filename):
    """
    Отобразить список маршрутов
    """
    staff = load_workers(filename)
    display_workers(staff)


@commands.command("select")
@click.argument("number")
@click.argument("filename")
def select(filename, number):
    """
    Выбрать маршрут с заданным номером
    """
    staff = load_workers(filename)
    result = []
    for staff in staff:
        if staff.get("number") == number:
            result.append(staff)

    display_workers(result)


def main():
    commands()


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

```
(base) C:\otkat>python C:\otkat\prog3\indsl1.py add --point Madagascar --number 999 --type PA-54 C:\otkat\lab3ind.json
JSON валиден по схеме.

(base) C:\otkat>python C:\otkat\prog3\indsl1.py display C:\otkat\lab3ind.json
Ошибка валидации: '999' is not of type 'integer'
+------+--------------------------------+------------+----------------------+
|  No  |       Пункт назначения         |  No рейса  |     Тип самолета     |
+------+--------------------------------+------------+----------------------+
|    1 | Moscow                         | 123        |          Boeing 737 |
|    2 | SPb                            | 456        |         Airbus A320 |
|    3 | Sochi                          | 789        |  Sukhoi Superjet 100 |
|    4 | Madagascar                     | 999        |               PA-54 |
+------+--------------------------------+------------+----------------------+
```

Рисунок 4 . Выполнение 2 задания

Ответы на контрольные вопросы:

1. Чем отличаются терминал и консоль?

Ответ: терминал – программа-оболочка, запускающая оболочку и позволяющая вводить команды. Консоль – разновидность терминала, это окно, в котором активны программы текстового режима.

2. Что такое консольное приложение?

Ответ: консольное приложение – программа, не имеющая графического интерфейса (окон), и которая работает в текстовом режиме в консоли. Команды в такой программе нужно вводить с клавиатуры, результаты работы консольные приложения также выводят на экран в текстовом виде.

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

Ответ: модуль sys (предоставляет доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором Python) и модуль argparse (Позволяет создавать красивые и гибкие интерфейсы командной строки с автоматической генерацией справки и поддержкой нескольких параметров командной строки).

4. Какие особенности построения CLI с использованием модуля sys?

Ответ: sys.argv – позволяет получить список аргументов командной строки. Эквивалент argc – количество элементов в списке (Получается от len()).

5. Какие особенности построения CLI с использованием модуля getopt?

Ответ: Модуль getopt в Python расширяет разделение входной строки проверкой параметров. Основанный на функции C getopt, он позволяет использовать как короткие, так и длинные варианты, включая присвоение значений. Удобен для простых CLI, но может быть не так гибок и мощен, как argparse.

6. Какие особенности построения CLI с использованием модуля argparse?

Ответ: особенности построения CLI с использованием модуля argparse: Поддержка создания позиционных аргументов и флагов. a) Возможность создания подкоманд для более сложных CLI. b) Автоматическая генерация справки. c) Поддержка типизации аргументов и их ограничений. d) Гибкая конфигурация для обработки различных сценариев использования. e) Часто используется для создания профессиональных и гибких CLI-интерфейсов.