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 №4
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

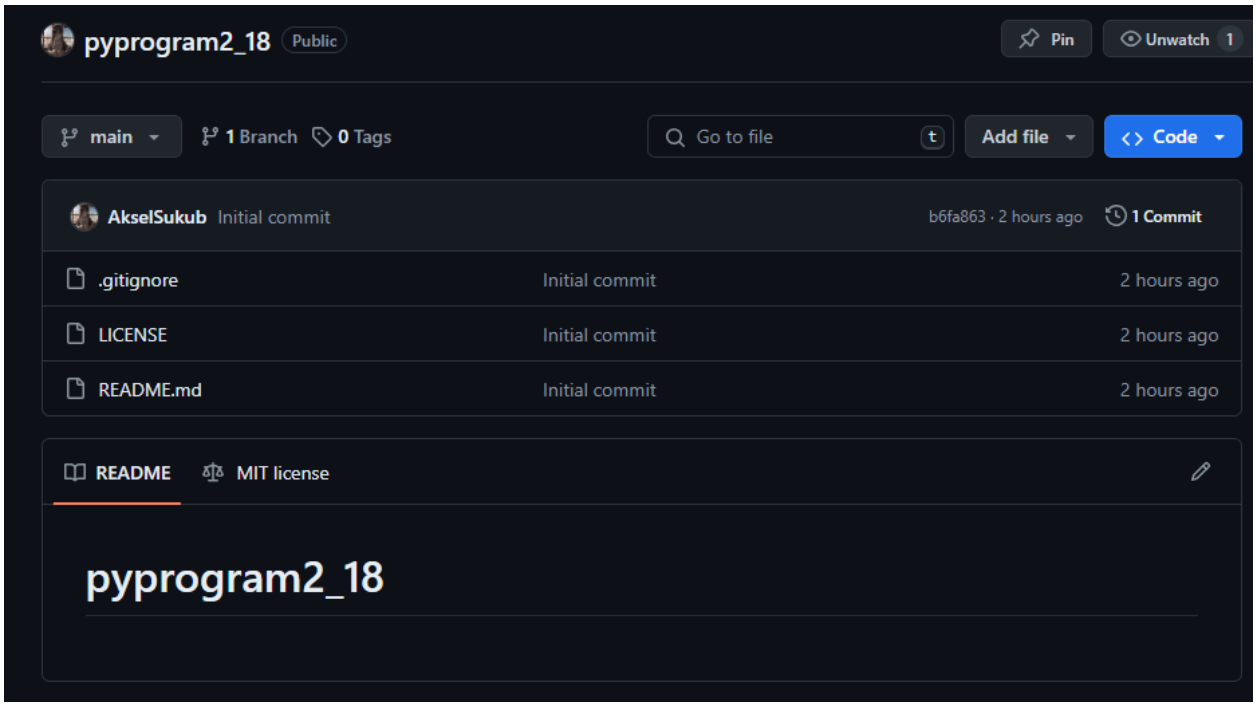

Рисунок 1. Созданный репозиторий

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 2.17 добавьте возможность получения имени файла данных, используя соответстсвующую переменную окружения.

```
#!/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.
    display = subparsers.add_parser(
        "display",
        parents=[file_parser], # Добавьте file_parser как родительский парсер
        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\prog4\ind1.py add -p Kazahstan -n 76 -t Kaz-25 C:\otkat\staff.json
JSON валиден по схеме.

(base) C:\otkat>python C:\otkat\prog4\ind1.py display C:\otkat\staff.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 | Kazahstan                      | 76         |               Kaz-25 |
+------+--------------------------------+------------+----------------------+
```

Рисунок 3 . Выполнение 1 задания

Самостоятельно изучите работу с пакетом `python-dotenv` . Модифицируйте программу задания 1 таким образом, чтобы значения необходимых переменных окружения считывались из файла `.env` .

```python
#!/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
from dotenv import load_dotenv

load_dotenv()  # Загружаем переменные окружения из .env

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


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)
```

```python
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
    data_file = os.getenv("DATA_FILE")
    if data_file:
        try:
            staff = load_workers(data_file)
        except FileNotFoundError:
            print(f"Файл данных {data_file} не найден.")
            staff = []
    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()
```

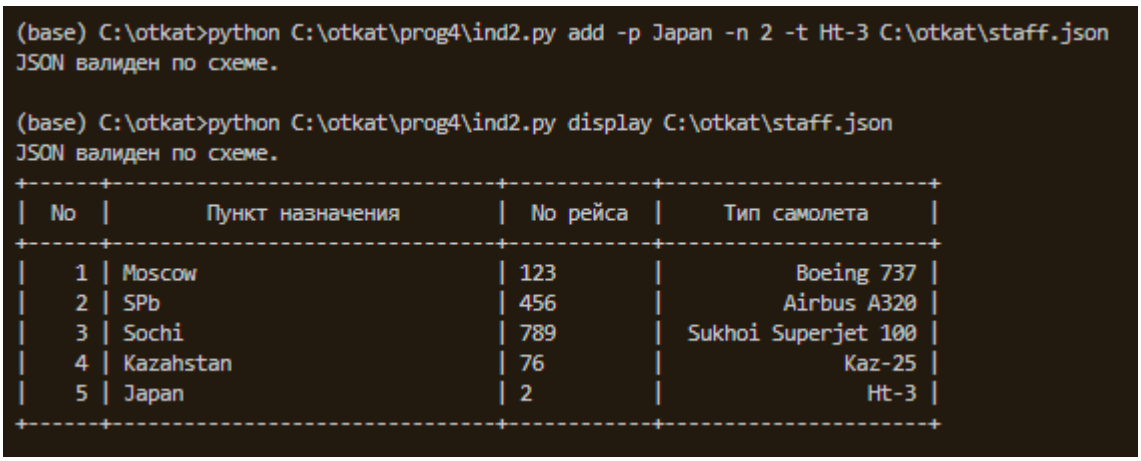

Рисунок 4 . Выполнение 2 задания

Ответы на контрольные вопросы:

1. Каково назначение переменных окружения?

Ответ: Переменные окружения используются для передачи информации процессам, которые запущены в оболочке.

2. Какая информация может храниться в переменных окружения? Переменные среды хранят информацию о среде операционной системы.

Ответ: Эта информация включает такие сведения, как путь к операционной системе, количество процессоров, используемых операционной системой, и расположение временных папок.

3. Как получить доступ к переменным окружения в ОС Windows?

Ответ: Нужно открыть окно свойства системы и нажать на кнопку “Переменные среды”.

4. Каково назначение переменных PATH и PATHEXT?

Ответ: PATH позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. PATHEXT дает возможность не указывать даже расширение файла, если оно прописано в ее значениях.

5. Как создать или изменить переменную окружения в Windows?

Ответ: В окне “Переменные среды” нужно нажать на кнопку “Создать”, затем ввести имя переменной и путь.

6. Что представляют собой переменные окружения в ОС Linux?

Ответ: Переменные окружения в Linux представляют собой набор именованных значений, используемых другими приложениями.

7. В чем отличие переменных окружения от переменных оболочки?

Ответ: Переменные окружения (или «переменные среды») – это переменные, доступные в масштабах всей системы и наследуемые всеми дочерними процессами и оболочками. Ответ: Переменные оболочки − это переменные, которые применяются только к текущему экземпляру оболочки. Каждая оболочка, например, bash или zsh, имеет свой собственный набор внутренних переменных.

8. Как вывести значение переменной окружения в Linux?

Ответ: Наиболее часто используемая команда для вывода переменных окружения – printenv.

9. Какие переменные окружения Linux Вам известны? USER − текущий пользователь.

Ответ: PWD – текущая директория; HOME – домашняя директория текущего пользователя. SHELL – путь к оболочке текущего пользователя; EDITOR – заданный по умолчанию редактор. Этот редактор будет вызываться в ответ на команду edit; LOGNAME – имя пользователя, используемое для входа в систему; PATH – пути к каталогам, в которых будет производиться поиск вызываемых команд. При выполнении команды система будет проходить по данным каталогам в указанном порядке и выберет первый из них, в котором будет находиться исполняемый файл искомой команды; LANG – текущие настройки языка и кодировки. TERM – тип текущего эмулятора терминала; MAIL – место хранения почты текущего пользователя. LS_COLORS задает цвета, используемые для выделения объектов.

10. Какие переменные оболочки Linux Вам известны?

Ответ: BASHOPTS – список задействованных параметров оболочки, разделенных двоеточием; BASH_VERSION – версия запущенной оболочки bash; COLUMNS – количество столбцов, которые используются для

отображения выходных данных; HISTFILESIZE – максимальное количество строк для файла истории команд. HISTSIZE – количество строк из файла истории команд, которые можно хранить в памяти. HOSTNAME – имя текущего хоста. IFS – внутренний разделитель поля в командной строке. PS1 – определяет внешний вид строки приглашения ввода новых команд. PS2 – вторичная строка приглашения. UID – идентификатор текущего пользователя.