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 РАБОТЕ №2.17
дисциплины «Анализ данных»

Выполнила:
Мурашко Анастасия Юрьевна
2 курс, группа ИВТ-б-о-22-1,
09.03.02 «Информатика и вычислительная техника», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р.А., канд. тех. наук, доцент,
доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____Дата защиты_____

Ставрополь, 2023 г.

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

Цель работы: приобретение навыков написания многопоточных приложений на языке программирования Python версии 3.x.

Порядок выполнения работы:

Задание 1.

Изучил теоретический материал работы, создал общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавил файл .gitignore с необходимыми правилами.

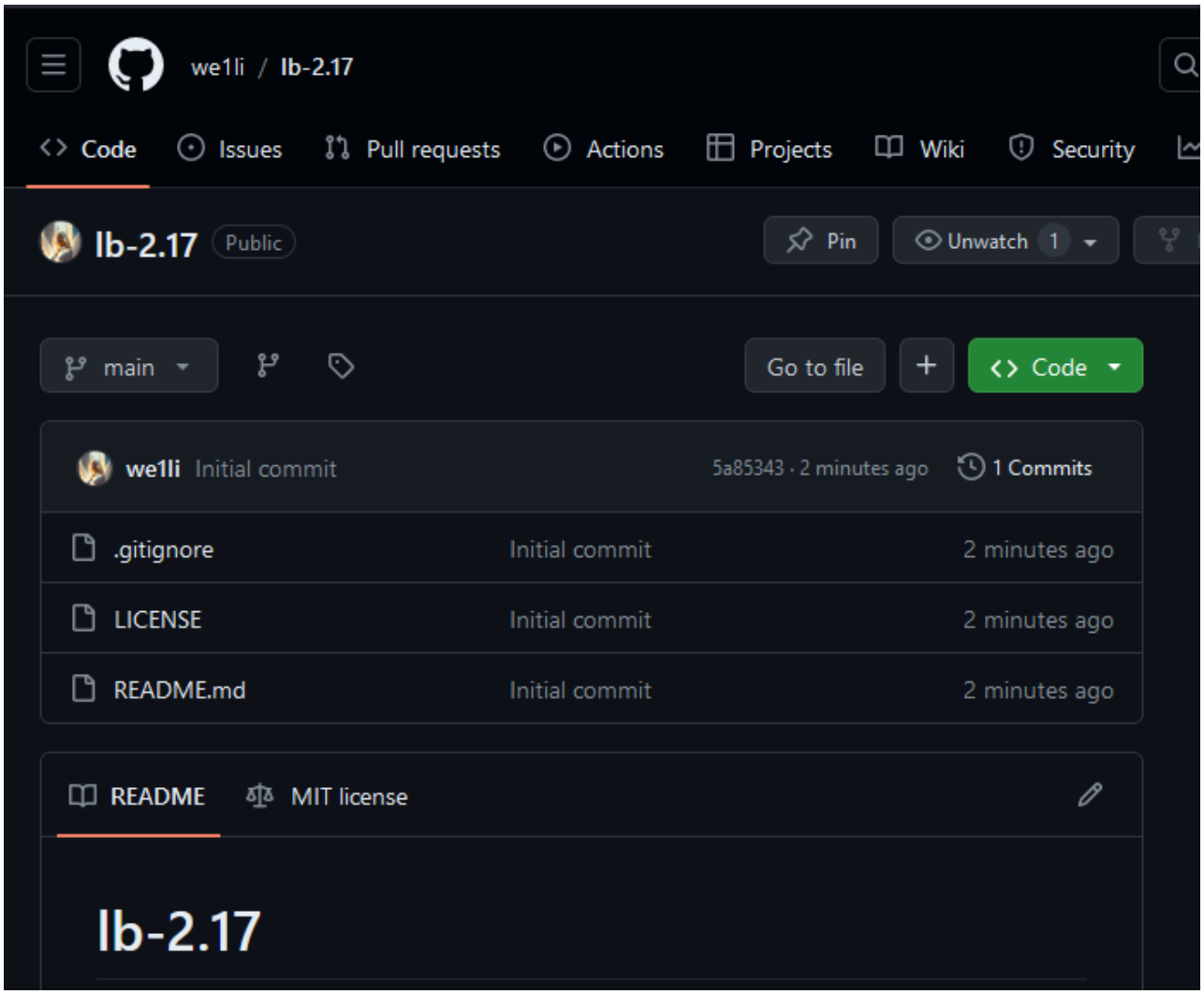

Рисунок 1. Новый репозиторий

Задание 2.

Проклонировала свой репозиторий на свой компьютер.

Организовала свой репозиторий в соответствие с моделью ветвления git-flow, появилась новая ветка develop.

```
C:\Users\student-09-510>git clone https://github.com/we1li/lb-2.17.git
Cloning into 'lb-2.17'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (5/5), done.

C:\Users\student-09-510>cd C:\Users\student-09-510\lb-2.17

C:\Users\student-09-510\lb-2.17>git flow init

Which branch should be used for bringing forth production releases?
   - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/]
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Users/student-09-510/lb-2.17/.git/hooks]

C:\Users\student-09-510\lb-2.17>
```

Рисунок 2. Клонирование и модель ветвления git-flow

Реализовывала примеры и индивидуальные задания на основе ветки develop, без создания дополнительной ветки feature/(название ветки) по указанию преподавателя.

Задание 3.

Проработала пример лабораторной работы.

Задание: Для примера 1 лабораторной работы 2.16 разработайте интерфейс командной строки.

При построении интерфейса командной строки мы не можем постоянно держать данные в оперативной памяти как это было сделано в предыдущей лабораторной работе. Поэтому имя файла JSON с данными программы должно быть одним из обязательных позиционных аргументов командной строки.

Добавим также следующие подкоманды:

- add – добавление рабочего, имя которого задано в аргументе с параметром --name , должность – в аргументе с параметром --post , а год поступления – в аргументе gitc параметром --year .
- display – отображение списка всех работников.
- select – выбор и отображение требуемых работников, у которых заданный период передается через аргумент с параметром --period .

Напишем программу для решения поставленной задачи.

```
#!/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(
                "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()
```

```
C:\Users\user\Desktop\ИВТ\Анализ данных\2.17\YPlr17>python lr7prim.py add data.json --name="Ведьмаков Геральт" --post="Помощник" --year=2003

C:\Users\user\Desktop\ИВТ\Анализ данных\2.17\YPlr17>python lr7prim.py display data.json
+------+------------------------------+----------------------+----------+
|  No  |           Ф.И.О.             |      Должность       |   Год    |
+------+------------------------------+----------------------+----------+
|    1 | Шишкин Шишка                 | Начальник            |     1990 |
|    2 | Шульман Ксения               | Водитель             |     2000 |
|    3 | Ведьмаков Геральт            | Помощник             |     2003 |
+------+------------------------------+----------------------+----------+
```

Рисунок 3. Результат работы примера

Задание 3.

Выполнение индивидуального задания.

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

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import argparse
import json
import os.path
import sys
```

```python
def add_shop(list_shop, name_shop, name_product, prise):
    """
    Добавить данные магазина.
    """
    list_shop.append(
        {
            "name_shop": name_shop,
            "name_product": name_product,
            "prise": prise
        }
    )
    return list_shop


def display_shop(list_shop):
    """
    Отобразить список магазинов.
    """
    if list shop:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 6,
            '-' * 20,
            '-' * 30,
            '-' * 20
        )
        print(line)
        print(
            '| {:^6} | {:^20} | {:^30} | {:^20} |'.format(
                "No",
                "Название магазина",
                "Название продукта",
                "Стоимость товара"
            )
        )
        print(line)
        for idx, listshop in enumerate(list_shop, 1):
            print(
                '| {:>6} | {:<20} | {:<30} | {:>20} |'.format(
                    idx,
                    listshop.get('name_shop', ''),
                    listshop.get('name_product', ''),
                    listshop.get('prise', 0)
                )
            )
        print(line)
    else:
        print("Список магазинов пуст.")


def select_product(list_shop, shop_sear):
    """
    Выбрать продукт.
    """
    search_shop = []
    for shop_sear_itme in list_shop:
        if shop_sear == shop_sear_itme['name_product']:
            search_shop.append(shop_sear_itme)
    return search_shop


def save_shop(file_name, list_shop):
    """
    Сохранить всех работников в файл JSON.
    """
    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(list_shop, fout, ensure_ascii=False, indent=4)
```

```
def load_list_shop(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)


def main(command_line=None):
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )
    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%(prog)s 0.1.0"
    )
    subparsers = parser.add_subparsers(dest="command")

    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new shop"
    )
    add.add_argument(
        "-ns",
        "--name_shop",
        action="store",
        required=True,
        help="The shop's name"
    )
    add.add_argument(
        "-np",
        "--name_product",
        action="store",
        help="The product's name"
    )
    add.add_argument(
        "-ps",
        "--prise",
        action="store",
        type=int,
        required=True,
        help="Cost of goods"
    )
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all shops"
    )
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the product"
    )
    select.add_argument(
        "-SS",
        "--shop_sear",
        action="store",
        type=str,
        required=True,
        help="The name product"
    )
```

```
    args = parser.parse_args(command_line)
    is_dirty = False
    if os.path.exists(args.filename):
        shop = load_list_shop(args.filename)
    else:
        shop = []
    if args.command == "add":
        shop = add_shop(
            shop,
            args.name_shop,
            args.name_product,
            args.prise
        )
        is_dirty = True
    elif args.command == "display":
        display_shop(shop)

    elif args.command == "select":
        selected = select product(shop, args.shop sear)
        display_shop(selected)
    if is_dirty:
        save_shop(args.filename, shop)


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

```
C:\Users\user\Desktop\ИВТ\Анализ данных\2.17\YPlr17>python lr7iz.py add data.json -ns="Пятерочка" -np="Огурец" -ps=12

C:\Users\user\Desktop\ИВТ\Анализ данных\2.17\YPlr17>python lr7iz.py add data.json -ns="Пятерочка" -np="Печенье" -ps=54

C:\Users\user\Desktop\ИВТ\Анализ данных\2.17\YPlr17>python lr7iz.py add data.json -ns="Пятерочка" -np="Сыр" -ps=104

C:\Users\user\Desktop\ИВТ\Анализ данных\2.17\YPlr17>python lr7iz.py display data.json
+--------+----------------------+--------------------------------+----------------------+
|   No   |  Название магазина   |       Название продукта        |   Стоимость товара   |
+--------+----------------------+--------------------------------+----------------------+
|      1 | Пятерочка            | Огурец                         |                   12 |
|      2 | Пятерочка            | Печенье                        |                   54 |
|      3 | Пятерочка            | Сыр                            |                  104 |
+--------+----------------------+--------------------------------+----------------------+
```

Рисунок 7. Результат индивидуального задания

1. Функция `add_shop`:

- Добавляет новый магазин в список магазинов с указанным названием, названием продукта и ценой.

2. Функция `display_shop`:

- Отображает список всех магазинов с их товарами и ценами в виде таблицы.

3. Функция `select_product`:

- Позволяет выбрать товары из списка магазинов по заданному названию продукта.

4. Функция `save_shop`:

- Сохраняет данные о магазинах и их товарах в файл JSON.

5. Функция `load_list_shop`:

- Загружает данные о магазинах и их товарах из файла JSON.

6. Функция `main`:

Основная функция, которая обрабатывает аргументы командной строки и управляет всеми остальными функциями в зависимости от команды.

Поддерживает следующие команды:

- `add`: Добавляет новый магазин с указанными данными.
- `display`: Отображает список всех магазинов.
- `select`: Выбирает товары из магазинов по указанному названию продукта.

Читает и сохраняет данные о магазинах из/в файл JSON.

Этот код позволяет управлять данными о магазинах и их товарах через командную строку, добавлять новые магазины, отображать существующие магазины и выбирать товары по их названию.

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?

Sys, getopt, argparse, click

4. Какие особенности построение CLI с использованием модуля sys

Модуль sys в Python предоставляет простые функции, которые позволяют нам напрямую взаимодействовать с интерпретатором. Функции, предоставляемые модулем sys, позволяют нам работать с базовым интерпретатором, независимо

от того, является ли он платформой Windows, Macintosh или Linux

5. Какие особенности построение CLI с использованием модуля getopt ?

Модуль getopt в Python – это анализатор, используемый для параметров командной строки, которые основаны на соглашении, организованном функцией UNIX getopt(). Он в основном используется для анализа последовательности аргументов, например sys.argv. Мы также можем истолковать этот модуль как помощника сценариям анализировать аргументы командной строки в sys.argv

6. Какие особенности построение CLI с использованием модуля argparse ?

Модуль argparse является рекомендуемым к использованию модулем стандартной библиотеки Python, предназначенным для работы с аргументами командной строки

Вывод: в ходе лабораторной работы приобретены навыки построения приложений с интерфейсом командной строки с помощью языка программирования Python версии 3.x.