

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 ЛАБОРАТОРНОЙ РАБОТЕ №2.17**  
**дисциплины «Основы программной инженерии»**

Выполнил:  
Юрьев Илья Евгеньевич  
2 курс, группа ПИЖ-б-о-22-1,  
09.03.04 «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Руководитель практики:  
Богданов С.С., ассистент кафедры  
инфокоммуникаций

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2024 г.

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

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

**Ход выполнения работы:**

1. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python:

**Create a new repository**

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

**Repository template**

No template ▾

Start your repository with a template repository's contents.

**Owner \*** daxstrong ▾ / **Repository name \*** lr2\_17

✔ lr2\_17 is available.

Great repository names are short and memorable. Need inspiration? How about [upgraded-octo-potato](#) ?

**Description** (optional)

Public ☒ Anyone on the internet can see this repository. You choose who can commit.

Private ☐ You choose who can see and commit to this repository.

**Initialize this repository with:**

☒ Add a README file  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

**Add .gitignore**

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

**Choose a license**

License: MIT License ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

This will set `main` as the default branch. Change the default name in your [settings](#).

Рисунок 1 – Создание репозитория с заданными настройками

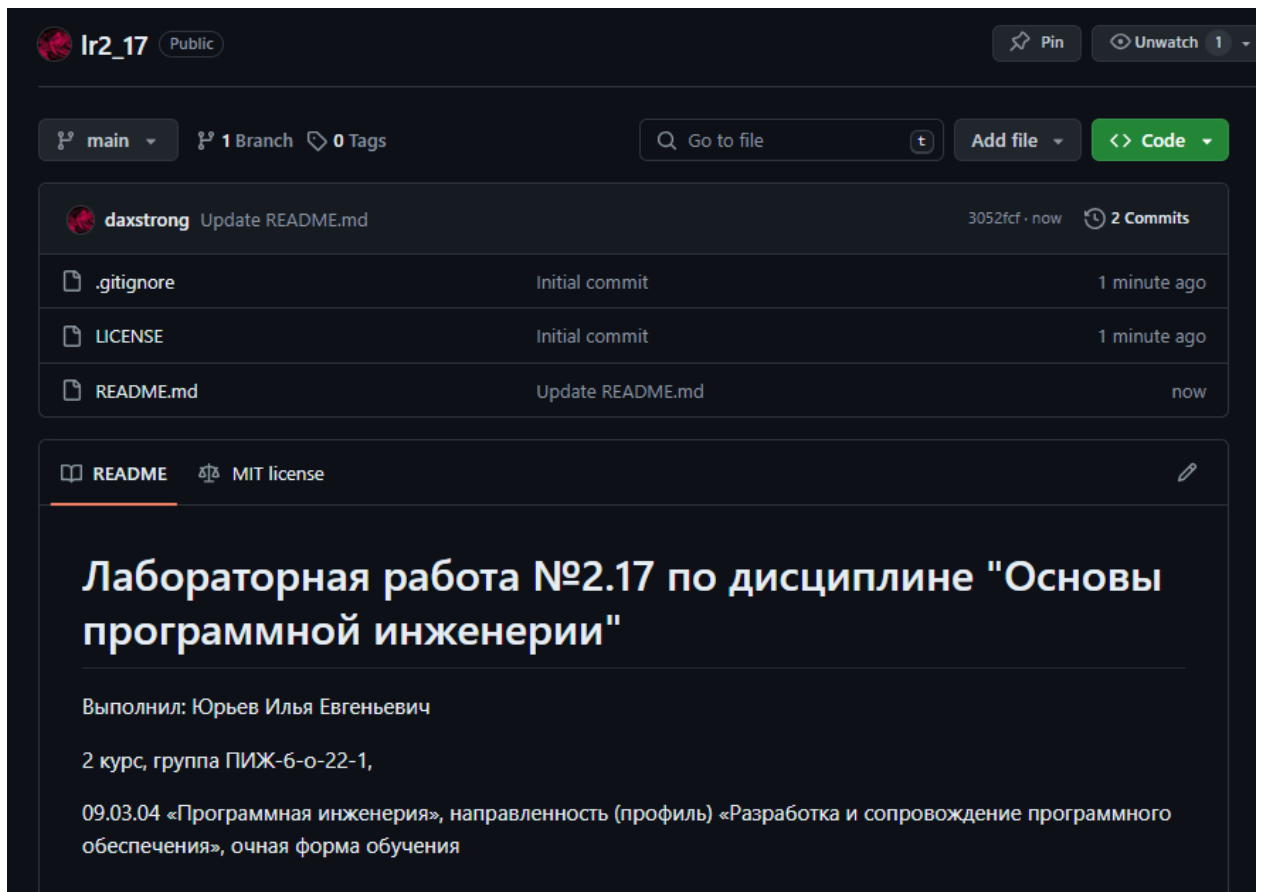


Рисунок 2 – Созданный репозиторий

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/4 семестр/Основы программн
ой инженерии
$ git clone https://github.com/daxstrong/lr2_17.git
Cloning into 'lr2_17'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (8/8), 4.13 KiB | 1.38 MiB/s, done.
Resolving deltas: 100% (1/1), done.
```

Рисунок 3 – Клонирование репозитория

```
ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/4 семестр/Основы программн
ой инженерии/lr2_17 (main)
$ git checkout -b develop
Switched to a new branch 'develop'
```

Рисунок 4 – Создание ветки develop

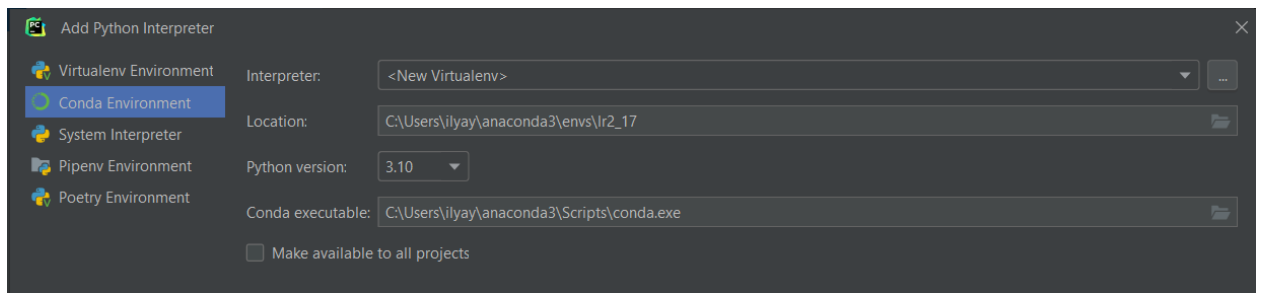


Рисунок 5 – Создание виртуального окружения с помощью Anaconda

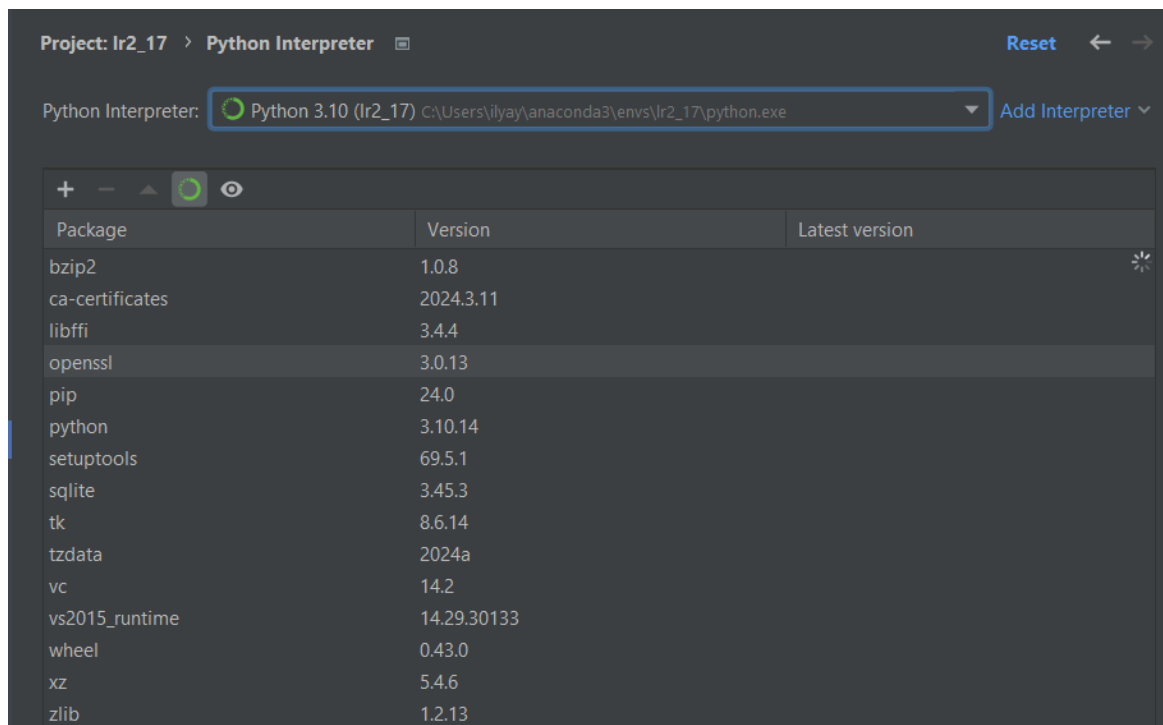


Рисунок 6 – Создание виртуального окружения с помощью Anaconda

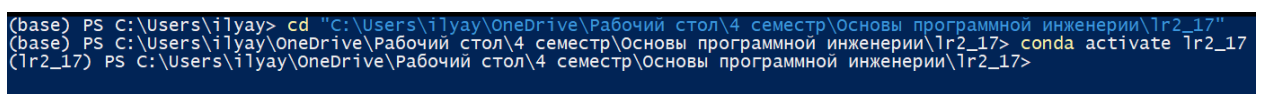


Рисунок 7 – Активация виртуального окружения

2. Проработать пример лабораторной работы. Для примера №1 лабораторной работы №2.16 разработать интерфейс командной строки:

```
#!/usr/bin/env python3
# -*- 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:
        # Заголовки 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 кириллицы 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="%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 работников.
    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 – Добавление интерфейса командной строки к учебному примеру

```

PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17> py ex1.py add data.json --name="Yurev Ilya" --post="Student" --year="2022"
PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17>

```

Рисунок 8 – Добавление нового работника, используя команду add

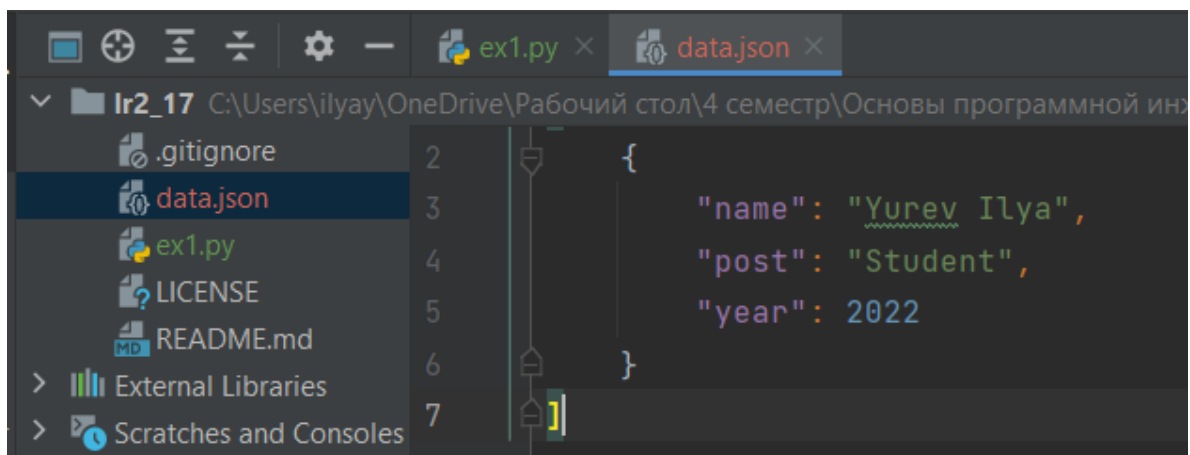


Рисунок 9 – Созданный файл data.json

```

PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17> py ex1.py display data.json
+-----+-----+-----+-----+
| № |          Ф.И.О.          |      Должность      |      Год      |
+-----+-----+-----+-----+
|  1 | Yurev Ilya                | Student             | 2022          |
+-----+-----+-----+-----+

```

Рисунок 10 – Отображение работников, используя команду display



```
PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17> py ex1.py select data.json --period=1
```

№	Ф.И.О.	Должность	Год
1	Yurev Ilya	Student	2022

Рисунок 11 – Выбор сотрудников с указанным периодом работы, используя команду select

### 3. Выполним индивидуальные задания:

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
import sys
from datetime import datetime

def exit_program():
    """
    Выход из программы.
    """
    sys.exit()

def add_person(people, last_name, first_name, phone_number,
birthdate_str):
    """
    Добавление информации о человеке.
    """
    birthdate = datetime.strptime(birthdate_str, "%d.%m.%Y")

    person = {
        'фамилия': last_name,
        'имя': first_name,
        'номер телефона': phone_number,
        'дата рождения': birthdate,
    }

    people.append(person)
    people.sort(key=lambda x: x['фамилия'])
    return people

def list_people(people):
    """
    Вывод списка всех людей.
    """
```

```

line = f'+-{"-" * 25}+-{"-" * 15}+-{"-" * 25}+-'
print(line)
print(f"| {'Фамилия':^25} | {'Имя':^15} | {'Дата рождения':^25}
|")

for person in people:
    print(line)
    print(f"| {person['фамилия']:^25} | {person['имя']:^15} |
{person['дата рождения'].strftime('%d.%м.%Y'):^25} |")
    print(line)

def select_people_by_month(people, month_to_search):
    """
    Вывод людей с днем рождения в указанном месяце.
    """
    found = False

    print(f"Люди с днем рождения в месяце {month_to_search}:")
    for person in people:
        if person['дата рождения'].month == month_to_search:
            print(
                f"Фамилия: {person['фамилия']], Имя: {person['имя']],
                Номер телефона: {person['номер телефона']], Дата рождения:
                {person['дата рождения'].strftime('%d.%м.%Y')}}"
            )
            found = True

    if not found:
        print("Нет людей с днем рождения в указанном месяце.")

def save_people(file_name, people):
    """
    Сохранить всех людей в файл JSON.
    """
    # Преобразуем объекты datetime в строки
    for person in people:
        person['дата рождения'] = person['дата
рождения'].strftime('%d.%м.%Y')

    with open(file_name, "w", encoding="utf-8") as fout:
        json.dump(people, fout, ensure_ascii=False, indent=4)

def load_people(file_name):
    """
    Загрузить всех людей из файла JSON.
    """
    try:
        with open(file_name, "r", encoding="utf-8") as fin:
            people_data = json.load(fin)
            for person in people_data:
                person['дата рождения'] =
datetime.strptime(person['дата рождения'], '%d.%м.%Y')
            return people_data
    except FileNotFoundError:
        return []

```

```

def main(command_line=None):
    """Основная функция управления программой."""

    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="Имя файла для данных"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("people")
    parser.add_argument(
        "--version", action="version", version="% (prog)s 1.0.0"
    )
    subparsers = parser.add_subparsers(dest="command")

    # Создать субпарсер для добавления человека.
    add = subparsers.add_parser(
        "add", parents=[file_parser], help="Добавить информацию о
человеке"
    )
    add.add_argument(
        "-l", "--last_name", action="store", required=True,
        help="Фамилия"
    )
    add.add_argument(
        "-f", "--first_name", action="store", required=True,
        help="Имя"
    )
    add.add_argument(
        "-p", "--phone_number", action="store", required=True,
        help="Номер телефона"
    )
    add.add_argument(
        "-b", "--birthdate", action="store", required=True, help="Дата
рождения (в формате ДД.ММ.ГГГГ)"
    )

    # Создать субпарсер для вывода всех людей.
    _ = subparsers.add_parser(
        "list",
        parents=[file_parser],
        help="Вывести список всех людей"
    )

    # Создать субпарсер для выбора людей по месяцу рождения.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Вывести людей с днем рождения в указанном месяце"
    )
    select.add_argument(
        "-m",

```

```

        "--month",
        action="store",
        type=int,
        required=True,
        help="Месяц (число от 1 до 12)"
    )

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

    # Загрузить всех людей из файла, если файл существует.
    changed_file = False
    if os.path.exists(args.filename):
        people = load_people(args.filename)
    else:
        people = []

    # Добавить человека.
    if args.command == "add":
        people = add_person(
            people,
            args.last_name,
            args.first_name,
            args.phone_number,
            args.birthdate
        )
        changed_file = True

    # Вывести всех людей.
    elif args.command == "list":
        list_people(people)

    # Выбрать людей по месяцу рождения.
    elif args.command == "select":
        select_people_by_month(people, args.month)

    else:
        print(f"Неизвестная команда {args.command}", file=sys.stderr)

    # Сохранить данные в файл, если список людей был изменен.
    if changed_file:
        save_people(args.filename, people)

if __name__ == '__main__':
    main()

```

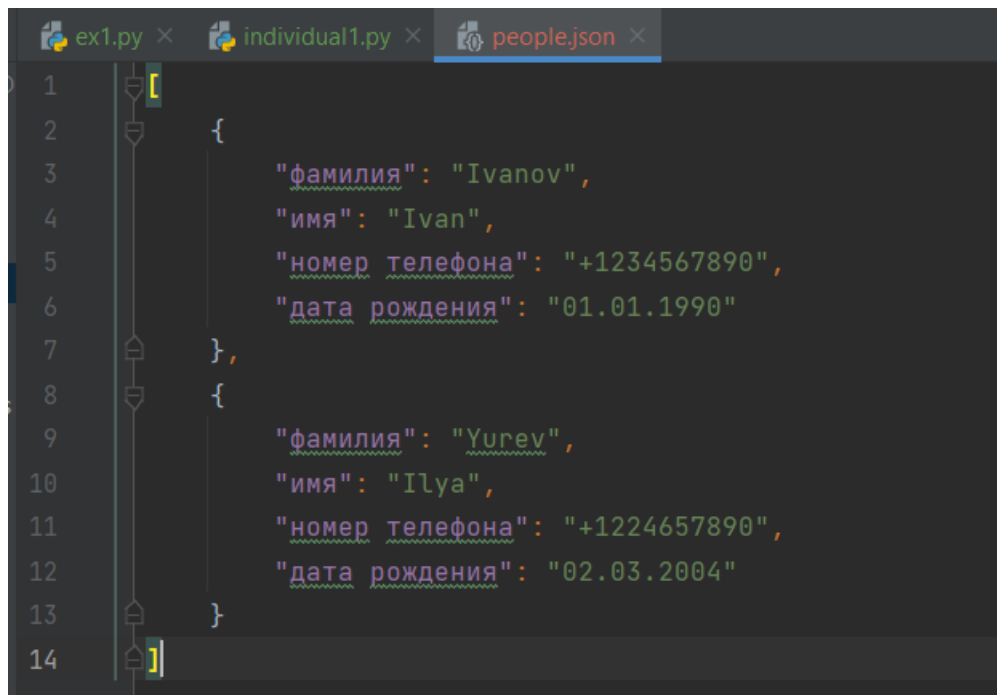
Листинг 2 – Добавление интерфейса командной строки к индивидуальному примеру из лабораторной работы №2.16

```

PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17> py individual1.py add people.json -f Ivanov -i Ivan -p +1234567890 -b 01.01.1990
PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17> py individual1.py add people.json -f Yurev -i Ilya -p +1224657890 -b 02.03.2004
PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17> py individual1.py list people.json
+-----+-----+-----+
| Фамилия | Имя | Дата рождения |
+-----+-----+-----+
| Ivanov | Ivan | 01.01.1990 |
+-----+-----+-----+
| Yurev | Ilya | 02.03.2004 |
+-----+-----+-----+
PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17> py individual1.py select people.json -m 1
Люди с днем рождения в месяце 1:
Фамилия: Ivanov, Имя: Ivan, Номер телефона: +1234567890, Дата рождения: 01.01.1990
PS C:\Users\ilyay\OneDrive\Рабочий стол\4 семестр\Основы программной инженерии\lr2_17>

```

Рисунок 12 – Выполнение команд



```

1  [
2      {
3          "фамилия": "Ivanov",
4          "имя": "Ivan",
5          "номер телефона": "+1234567890",
6          "дата рождения": "01.01.1990"
7      },
8      {
9          "фамилия": "Yurev",
10         "имя": "Ilya",
11         "номер телефона": "+1224657890",
12         "дата рождения": "02.03.2004"
13     }
14 ]

```

Рисунок 13 – Сохраненные данные в people.json

4. Выполним задание повышенной сложности:

Необходимо ознакомиться с пакетом click для построения интерфейса командной строки и реализовать его для своего варианта лабораторной работы №2.16:

```

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

import click
import json
import os
from datetime import datetime

```

```

def add_person(people, last_name, first_name, phone_number,
birthdate_str):
    """
    Добавление информации о человеке.
    """
    birthdate = datetime.strptime(birthdate_str, "%d.%m.%Y")

    person = {
        'фамилия': last_name,
        'имя': first_name,
        'номер телефона': phone_number,
        'дата рождения': birthdate,
    }

    people.append(person)
    people.sort(key=lambda x: x['фамилия'])
    return people

def list_people(people):
    """
    Вывод списка всех людей.
    """
    line = f'+-{"-" * 25}+-{"-" * 15}+-{"-" * 25}+-'
    click.echo(line)
    click.echo(f"| {'Фамилия':^25} | {'Имя':^15} | {'Дата рождения':^25} |")

    for person in people:
        click.echo(line)
        click.echo(
            f"| {person['фамилия']:^25} | {person['имя']:^15} | {person['дата рождения'].strftime('%d.%m.%Y'):^25} |")
        click.echo(line)

def select_people_by_month(people, month_to_search):
    """
    Вывод людей с днем рождения в указанном месяце.
    """
    found = False

    click.echo(f"Люди с днем рождения в месяце {month_to_search}:")
    for person in people:
        if person['дата рождения'].month == month_to_search:
            click.echo(
                f"Фамилия: {person['фамилия']}, Имя: {person['имя']}, Номер телефона: {person['номер телефона']}, Дата рождения: {person['дата рождения'].strftime('%d.%m.%Y')}")
            found = True

    if not found:
        click.echo("Нет людей с днем рождения в указанном месяце.")

def save_people(file_name, people):
    """

```

```

        Сохранить всех людей в файл JSON.
        """
        # Преобразуем объекты datetime в строки
        for person in people:
            person['дата рождения'] = person['дата
рождения'].strftime('%d.%m.%Y')

        with open(file_name, "w", encoding="utf-8") as fout:
            json.dump(people, fout, ensure_ascii=False, indent=4)

def load_people(file_name):
    """
    Загрузить всех людей из файла JSON.
    """
    try:
        with open(file_name, "r", encoding="utf-8") as fin:
            people_data = json.load(fin)
            for person in people_data:
                person['дата рождения'] =
datetime.strptime(person['дата рождения'], '%d.%m.%Y')
            return people_data
    except FileNotFoundError:
        return []

@click.group()
def cli():
    """Список команд для управления информацией о людях."""
    pass

@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-l', '--last_name', prompt='Фамилия', help='Фамилия')
@click.option('-f', '--first_name', prompt='Имя', help='Имя')
@click.option('-p', '--phone_number', prompt='Номер телефона',
help='Номер телефона')
@click.option('-b', '--birthdate', prompt='Дата рождения
(ДД.ММ.ГГГГ)', help='Дата рождения (в формате ДД.ММ.ГГГГ)')
def add(filename, last_name, first_name, phone_number, birthdate):
    """Добавить информацию о новом человеке."""
    if os.path.exists(filename):
        people = load_people(filename)
    else:
        people = []

    people = add_person(people, last_name, first_name, phone_number,
birthdate)
    save_people(filename, people)

@cli.command()
@click.argument('filename', type=click.Path())
def list(filename):
    """Вывести список всех людей."""
    if os.path.exists(filename):

```

```

        people = load_people(filename)
        list_people(people)
    else:
        click.echo("Файл не найден.")

@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-m', '--month', prompt='Месяц', type=int, help='Месяц
(число от 1 до 12)')
def select(filename, month):
    """Вывести людей с днем рождения в указанном месяце."""
    if os.path.exists(filename):
        people = load_people(filename)
        select_people_by_month(people, month)
    else:
        click.echo("Файл не найден.")

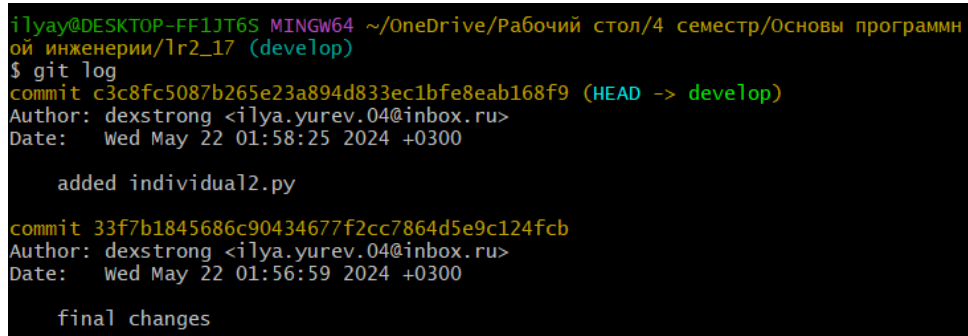
if __name__ == '__main__':
    cli()

```

### Листинг 3 – Добавление интерфейса командной строки к индивидуальному примеру из лабораторной работы №2.16 с помощью пакета click

Пакет click предоставляет простой и интуитивно понятный способ создания интерфейсов командной строки (CLI). Он упрощает обработку аргументов и опций командной строки, обеспечивает ввод-вывод и валидацию данных, а также позволяет организовать код более структурированным и читаемым образом.

5. Зафиксируем проделанные изменения, сольем ветки и отправим на удаленный репозиторий:



```

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/4 семестр/Основы программ
ой инженерии/lr2_17 (develop)
$ git log
commit c3c8fc5087b265e23a894d833ec1bfe8eab168f9 (HEAD -> develop)
Author: dexstrong <ilya.yurev.04@inbox.ru>
Date: Wed May 22 01:58:25 2024 +0300

    added individual2.py

commit 33f7b1845686c90434677f2cc7864d5e9c124fcb
Author: dexstrong <ilya.yurev.04@inbox.ru>
Date: Wed May 22 01:56:59 2024 +0300

    final changes

```

Рисунок 14 – Коммиты ветки develop во время выполнения лабораторной работы



```

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/4 семестр/Основы программн
ой инженерии/lr2_17 (develop)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/4 семестр/Основы программн
ой инженерии/lr2_17 (main)
$ git merge develop
Updating 3052fcf..c3c8fc5
Fast-forward
 .idea/.gitignore | 8 +
 .idea/inspectionProfiles/Project_Default.xml | 51 ++++++
 .idea/inspectionProfiles/profiles_settings.xml | 6 +
 .idea/lr2_17.iml | 8 +
 .idea/misc.xml | 4 +
 .idea/modules.xml | 8 +
 .idea/vcs.xml | 6 +
 ex1.py | 193 +++++++++++++++++++++++++++++++++++++
 individual1.py | 188 +++++++++++++++++++++++++++++++++++++
 individual2.py | 133 +++++++++++++++++++++++++++++++++
 10 files changed, 605 insertions(+)
 create mode 100644 .idea/.gitignore
 create mode 100644 .idea/inspectionProfiles/Project_Default.xml
 create mode 100644 .idea/inspectionProfiles/profiles_settings.xml
 create mode 100644 .idea/lr2_17.iml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/modules.xml
 create mode 100644 .idea/vcs.xml
 create mode 100644 ex1.py
 create mode 100644 individual1.py
 create mode 100644 individual2.py

```

Рисунок 15 – Слияние веток main и develop

```

ilyay@DESKTOP-FF1JT6S MINGW64 ~/OneDrive/Рабочий стол/4 семестр/Основы программн
ой инженерии/lr2_17 (main)
$ git push origin main
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 12 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (17/17), 6.83 KiB | 6.83 MiB/s, done.
Total 17 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/daxstrong/lr2_17.git
 3052fcf..c3c8fc5 main -> main

```

Рисунок 16 – Отправка изменений на удаленный репозиторий

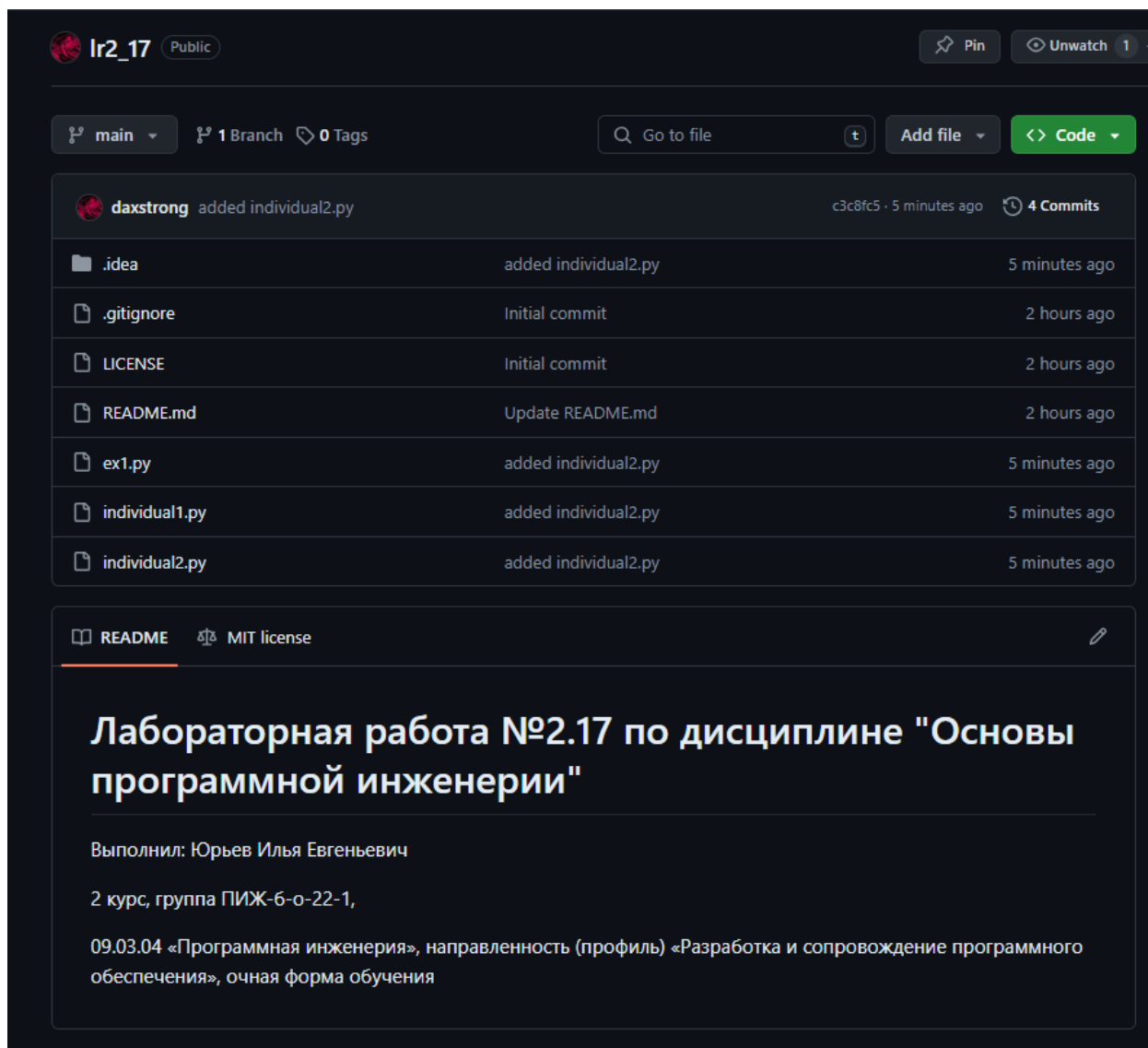


Рисунок 17 – Изменения удаленного репозитория

Ссылка на репозиторий **GitHub**: [https://github.com/daxstrong/lr2\\_17](https://github.com/daxstrong/lr2_17)

## **Ответы на контрольные вопросы:**

### **1. В чем отличие терминала и консоли?**

Терминал – 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 – 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 – 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. У каждого из этих способов есть свои плюсы и минусы, поэтому стоит оценить каждый, чтобы увидеть, какой из них лучше всего соответствует вашим потребностям.

### **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`.

#### 5. Какие особенности построение CLI с использованием модуля `getopt`?

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.

#### 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.

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.

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`);

- `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 отправляет к другому (субпарсеру), в зависимости от аргументов на входе.