

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:  
Соколов Михаил Романович  
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 .gitignore файл для языка 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 \*** PoTtaTto ▾ / **Repository name \*** fundamentals\_of\_software

✔ fundamentals\_of\_software\_engineering\_lab2\_17 is available.

Great repository names are short and memorable. Need inspiration? How about reimagined-waddle ?

**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](#).

ⓘ You are creating a public repository in your personal account.

**Create repository**

Рисунок 1 – Создание репозитория с заданными настройками

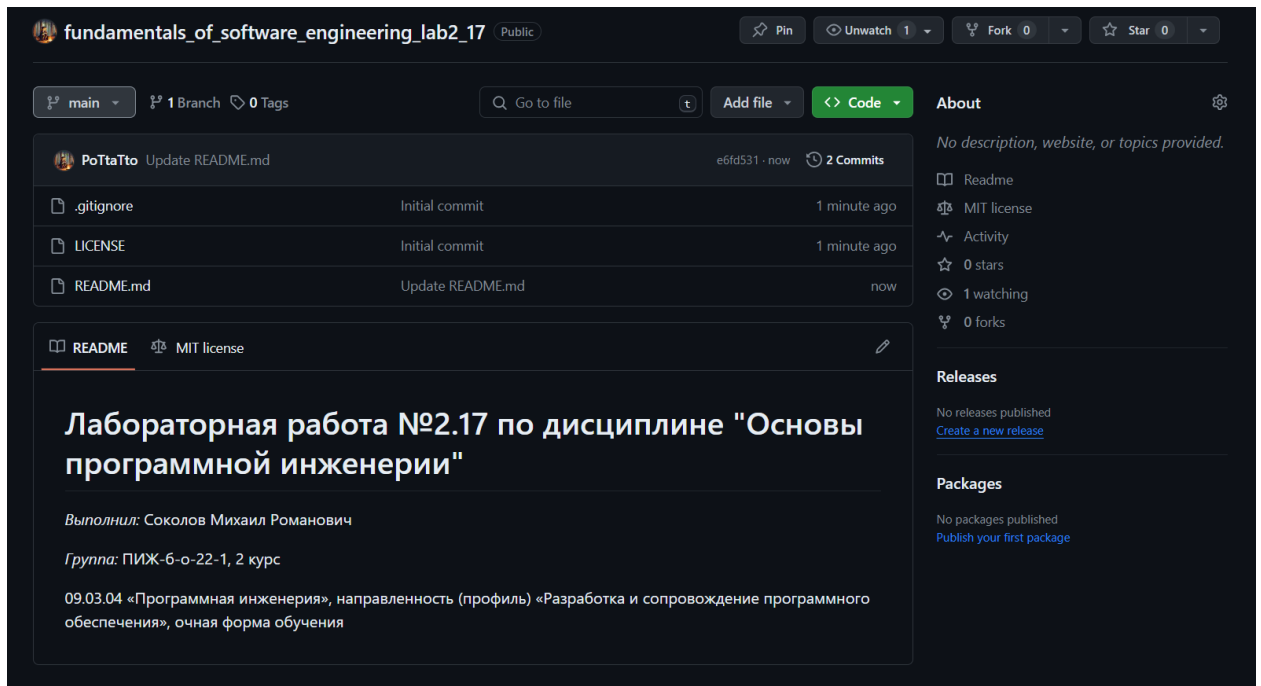


Рисунок 2 – Созданный репозиторий

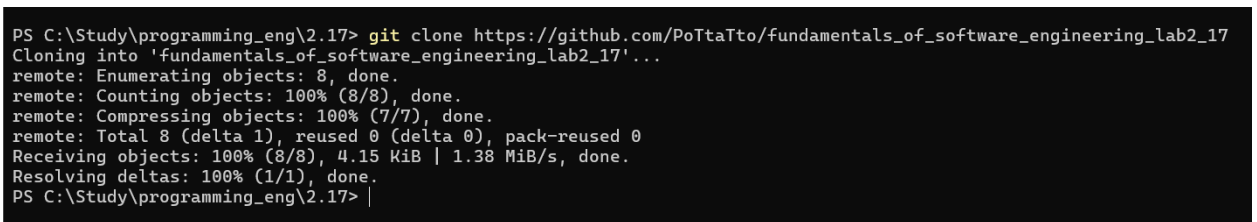


Рисунок 3 – Клонирование репозитория

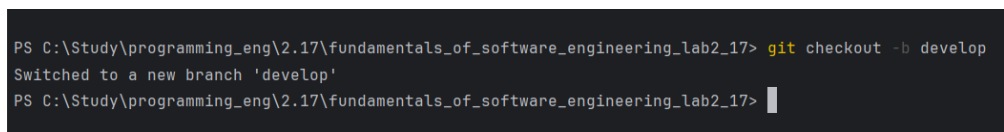


Рисунок 4 – Создание ветки develop, где будут происходить изменения проекта до его полного релиза

```

1 # Byte-compiled / optimized / DLL files
2 __pycache__ /
3 *.py[cod]
4 *$py.class
5
6 # C extensions
7 *.so
8
9 # Distribution / packaging
10 .Python
11 build /
12 develop-eggs /
13 dist /
14 downloads /
15 eggs /
16 .eggs /
17 lib /
18 lib64 /
19 parts /
20 sdist /
21 var /
22 wheels /
23 share/python-wheels /
24 *.egg-info /

```

Рисунок 5 – Часть .gitignore, созданного GitHub

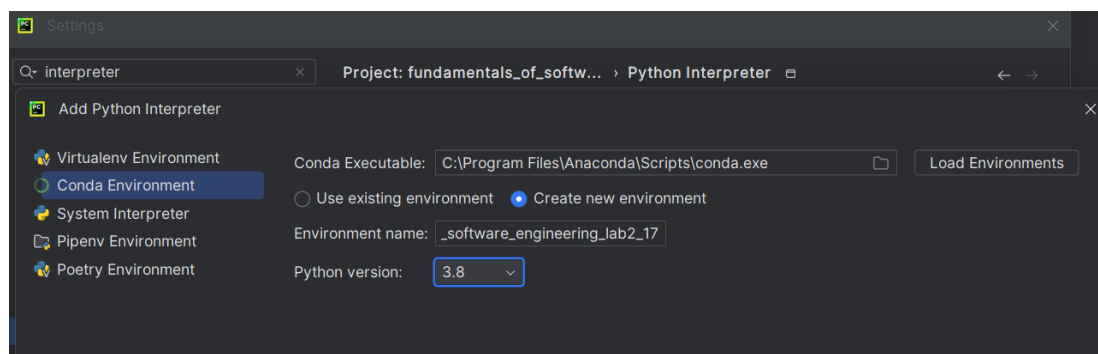


Рисунок 6 – Создание виртуального окружения с помощью Anaconda

Project: fundamentals\_of\_softw... Python Interpreter Reset

Python Interpreter: fundamentals\_of\_software\_engineering\_lab2\_17 C:\Use Add Interpreter

Try the redesigned packaging support in Python Packages tool window. Go to tool window

Package	Version	Latest version
ca-certificates	2023.12.12	
libffi	3.4.4	
openssl	3.0.13	
pip	23.3.1	
python	3.8.18	
setuptools	68.2.2	
sqlite	3.41.2	
vc	14.2	
vs2015_runtime	14.27.29016	
wheel	0.41.2	

Рисунок 7 – Созданное окружение

2. Проработаем 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:
        # Заголовок 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="% (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 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()

```

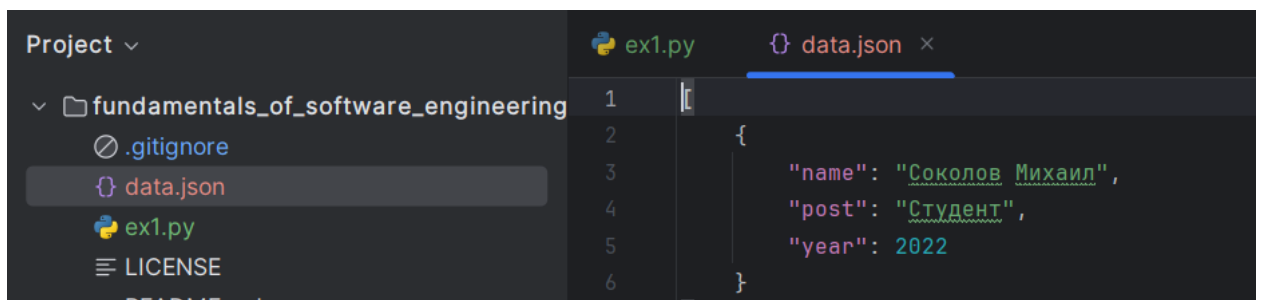
Листинг 1 – Добавление интерфейса командной строки к учебному примеру

```

workers 0.1.0
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py ex1.py add data.json --name="Соколов Михаил" --post="Студент" --year=2022
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17>

```

Рисунок 8 – Добавление нового работника, используя команду add



The screenshot shows a code editor with a file named `data.json` open. The file contains a JSON object with the following fields: `"name": "Соколов Михаил", "post": "Студент", "year": 2022`. The left sidebar shows the project structure with `data.json` selected.

Рисунок 9 – Созданный файл data.json

```

PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py ex1.py display data.json
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Соколов Михаил | Студент | 2022 |
+-----+-----+-----+-----+
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17>

```

Рисунок 10 – Отображение работников, используя команду display

```

PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py ex1.py select data.json --period=1
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Соколов Михаил | Студент | 2022 |
+-----+-----+-----+-----+
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17>

```

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

3. Выполним индивидуальное задание. Для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI):



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

# Standard
import os
import argparse
from random import randint
import json
import jsonschema

def add_train(trains, num, destination, start_time):
    """
    Добавляет информацию о поезде в список trains.

    Args:
    - trains (list): Список поездов.
    - num (int): Номер поезда.
    - destination (str): Пункт назначения.
    - start_time (str): Время отправки

    Returns:
    - trains (list): Список поездов.
    """
    trains.append({
        'num': num,
        'destination': destination,
        'start_time': start_time
    })
    if len(trains) > 1:
        trains.sort(key=lambda item: item['start_time'])

    return trains

def save_trains(file_name, trains):
    """
    Сохраняет список поездов в файл в формате JSON.

    Args:
    - file_name (str): Имя файла.
    - trains (list): Список поездов.

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

def load_trains(file_name):
    """
    Загружает список поездов из файла в формате JSON.

    Args:
    - file_name (str): Имя файла.

    Returns:
    - trains (list): Список поездов.

    """
    with open(file_name, "r", encoding="utf-8") as fin:
        loaded_data = json.load(fin)

    with open('scheme.json', 'r', encoding='utf-8') as scheme_file:
```

```

        scheme = json.load(scheme_file)

    try:
        jsonschema.validate(loader_data, scheme)
        return loader_data
    except jsonschema.exceptions.ValidationError as e:
        print('Ошибка валидации данных:', e)
        return None

def display_trains(trains):
    """
    Выводит список поездов на экран.

    Args:
    - trains (list): Список поездов.

    """
    line = f'+-{"-" * 15}-+{"-" * 30}-+{"-" * 25}-+'
    print(line)
    header = f"| {'№ поезда':^15} | {'Пункт назначения':^30} | {'Время отъезда':^25} |"
    print(header)
    print(line)
    for train in trains:
        num = train.get('num', randint(1000, 10000))
        destination = train.get('destination', 'None')
        start_time = train.get('start_time', 'None')
        recording = f"| {num:^15} | {destination:^30} | {start_time:^25} |"
        print(recording)
    print(line)

def select_trains(trains, destination):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - destination (list): Пункт назначения.

    Returns:
    - trains (list): Список поездов.

    """

    return [train for train in trains if train['destination'].strip() == destination]

def main(command_line=None):
    # Creating file parser
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "filename",
        action="store",
        help="The data file name"
    )

    # Main parser of command line
    parser = argparse.ArgumentParser("trains")
    parser.add_argument(
        "--version",
        action="version",

```

```

        version="% (prog)s 1.0.0"
    )

    # Subparsers
    subparsers = parser.add_subparsers(dest="command")

    # Subparser for add command
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new train"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        required=True,
        type=int,
        help="The number of a train"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination point"
    )
    add.add_argument(
        "-st",
        "--start_time",
        action="store",
        required=True,
        help="Depart time"
    )

    # Subparser for display command
    display = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all trains"
    )

    # Subparser for select command
    select = subparsers.add_parser(
        'select',
        parents=[file_parser],
        help='Select trains by destination'
    )

    select.add_argument(
        "-D",
        "--dest",
        action="store",
        required=True,
        help="The required destination"
    )

    args = parser.parse_args(command_line)
    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if os.path.exists(args.filename):
        trains = load_trains(args.filename)
    else:
        trains = []

```

```

match args.command:
    case 'add':
        trains = add_train(
            trains,
            args.number,
            args.destination,
            args.start_time
        )
        is_dirty = True
    case 'display':
        display_trains(trains)
    case 'select':
        selected = select_trains(trains, args.dest)
        display_trains(selected)

# Save changes in file if data is changed
if is_dirty:
    save_trains(args.filename, trains)

if __name__ == '__main__':
    main()

```

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

```

PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py -h
usage: trains [-h] [--version] {add,display,select} ...

positional arguments:
  {add,display,select}
    add                Add a new train
    display            Display all trains
    select            Select trains by destination

options:
  -h, --help            show this help message and exit
  --version            show program's version number and exit

```

Рисунок 12 – Отображения подсказок пользования программой

```

PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py --version
trains 1.0.0

```

Рисунок 13 – Отображение текущей версии программы

```

PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py trains.json
usage: trains [-h] [--version] {add,display,select} ...
trains: error: argument command: invalid choice: 'trains.json' (choose from 'add', 'display', 'select')
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17>

```

Рисунок 14 – Запуск программы без позиционных аргументов

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py add
usage: trains add [-h] -n NUMBER -d DESTINATION -st START_TIME filename
trains add: error: the following arguments are required: filename, -n/--number, -d/--destination, -st/--start_time
```

Рисунок 15 – Запуск программы без опциональных, но требуемых аргументов

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py add trains.json --number 2 -d Москва -st "18:40"
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py add trains.json -n 1 --destination Ставрополь --start_time "17:00"
```

Рисунок 16 – Добавление информации о поездах

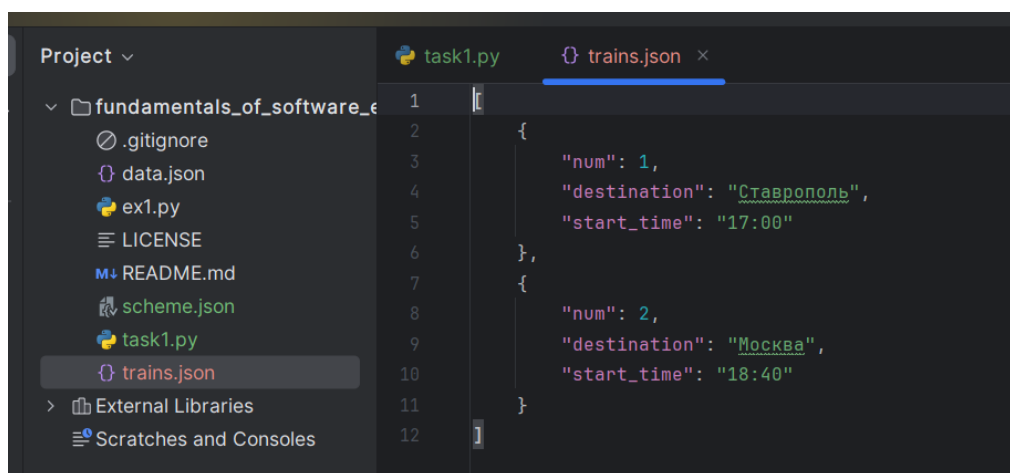


Рисунок 17 – Полученный файл trains.json

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py display trains.json
```

№ поезда	Пункт назначения	Время отъезда
1	Ставрополь	17:00
2	Москва	18:40

Рисунок 18 – Отображение данных в файле

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py select trains.json --dest Бишкек
```

№ поезда	Пункт назначения	Время отъезда
----------	------------------	---------------

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17>
```

Рисунок 19 – Выбор поездов, у которых пункт назначения Бишкек (таких нет)

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> py task1.py select trains.json --dest Ставрополь
```

№ поезда	Пункт назначения	Время отъезда
1	Ставрополь	17:00

Рисунок 20 – Выбор поездов, у которых пункт назначения Ставрополь

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:

```
(base) PS C:\Users\MrPot> activate fundamentals_of_software_engineering_lab_2_17
(base) PS C:\Users\MrPot> pip install click
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: click in c:\program files\anaconda\lib\site-packages (8.0.4)
Requirement already satisfied: colorama in c:\program files\anaconda\lib\site-packages (from click) (0.4.6)
(base) PS C:\Users\MrPot> |
```

Рисунок 21 – Установка пакета `click`

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

# Standard
import os
import click
from random import randint
import json
import jsonschema

def save_trains(file_name, trains):
    """
    Сохраняет список поездов в файл в формате JSON.

    Args:
    - file_name (str): Имя файла.
    - trains (list): Список поездов.

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

def load_trains(file_name):
    """
    Загружает список поездов из файла в формате JSON.

    Args:
    - file_name (str): Имя файла.

    Returns:
    - trains (list): Список поездов.

    """
```

```

    if os.path.exists(file_name):
        with open(file_name, "r", encoding="utf-8") as fin:
            loaded_data = json.load(fin)

        with open('scheme.json', 'r', encoding='utf-8') as scheme_file:
            scheme = json.load(scheme_file)

        try:
            jsonschema.validate(loaded_data, scheme)
            return loaded_data
        except jsonschema.exceptions.ValidationError as e:
            print('Ошибка валидации данных:', e)
            return None
    else:
        return []

@click.group()
@click.version_option('1.0.0')
def cli():
    pass

@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-n', '--number', type=int, prompt=True, help='The number of a train')
@click.option('-d', '--destination', prompt=True, help='Destination point')
@click.option('-st', '--start_time', prompt=True, help='Depart time')
def add(filename, num, destination, start_time):
    """
    Добавляет информацию о поезде в список trains.

    Args:
    - trains (list): Список поездов.
    - num (int): Номер поезда.
    - destination (str): Пункт назначения.
    - start_time (str): Время отправки

    Returns:
    - trains (list): Список поездов.
    """
    trains = load_trains(file_name=filename)

    trains.append({
        'num': num,
        'destination': destination,
        'start_time': start_time
    })
    if len(trains) > 1:
        trains.sort(key=lambda item: item['start_time'])

    save_trains(file_name=filename, trains=trains)

@cli.command()
@click.argument('filename', type=click.Path())
def display(filename):
    """
    Выводит список поездов на экран.

    Args:
    - trains (list): Список поездов.

```

```

"""
line = f'+-{"-" * 15}-+{"-" * 30}-+{"-" * 25}-+'
print(line)
header = f"| {'№ поезда':^15} | {'Пункт назначения':^30} | {'Время
отъезда':^25} |"
print(header)
print(line)
for train in load_trains(file_name=filename):
    num = train.get('num', randint(1000, 10000))
    destination = train.get('destination', 'None')
    start_time = train.get('start_time', 'None')
    recording = f"| {num:^15} | {destination:^30} | {start_time:^25} |"
    print(recording)
print(line)

@cli.command()
@click.argument('filename', type=click.Path())
@click.option('-D', '--destination', prompt=True, help='The required
destination')
def select(filename, destination):
    """
    Выводит информацию о поездах, направляющихся в указанный пункт.

    Args:
    - trains (list): Список поездов.
    - destination (list): Пункт назначения.

    Returns:
    - trains (list): Список поездов.

    """

    return [train for train in load_trains(file_name=filename) if
train['destination'].strip() == destination]

def main(command_line=None):
    cli()

if __name__ == '__main__':
    main()

```

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

Click - это пакет Python, который упрощает создание интерфейсов командной строки (CLI). В предоставленном коде Click используется для определения команд и параметров CLI с использованием декораторов, таких как `@click.command()` и `@click.option()`. Определены такие команды, как `add`, `display` и `select`, каждая со своим собственным набором аргументов и функциональных возможностей. Click обрабатывает синтаксический анализ



аргументов и параметров командной строки и выполняет соответствующие функции при вызове команд.

Отображение в консоли не изменилось и вывод аналогичен как при использовании argparse (см. пункт 3).

5. Сошьем ветки develop и main/master и отправим изменения на удаленный репозиторий:

Ссылка: [https://github.com/PoTtaTto/fundamentals\\_of\\_software\\_engineering\\_lab2\\_17](https://github.com/PoTtaTto/fundamentals_of_software_engineering_lab2_17)

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> git log --oneline
eeb7376 (HEAD -> develop) hard task is completed
a1b0360 task1 is completed
7be0ced lab task 1 is completed
e6fd531 (origin/main, origin/HEAD, main) Update README.md
dee414d Initial commit
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> |
```

Рисунок 22 – История коммитов

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> git merge develop
Updating e6fd531..eeb7376
Fast-forward
 .gitignore | 2 +-
 data.json | 7 +++
 ex1.py | 193 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 scheme.json | 13 +++
 task1.py | 139 +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 trains.json | 12 +++
 6 files changed, 365 insertions(+), 1 deletion(-)
 create mode 100644 data.json
 create mode 100644 ex1.py
 create mode 100644 scheme.json
 create mode 100644 task1.py
```

Рисунок 23 – Слияние веток main и develop

```
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17> git push origin main
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 12 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 5.15 KiB | 5.15 MiB/s, done.
Total 13 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/PoTtaIto/fundamentals\_of\_software\_engineering\_lab2\_17
 e6fd531..eeb7376  main -> main
PS C:\Study\programming_eng\2.17\fundamentals_of_software_engineering_lab2_17>
```

Рисунок 24 – Отправка изменений на удаленный репозиторий

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.