

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

**Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Кафедра инфокоммуникаций
«Разработка приложений с интерфейсом командной строки (CLI) в
Python3»**

**Отчет по лабораторной работе № 2.17
по дисциплине «Основы программной инженерии»**

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

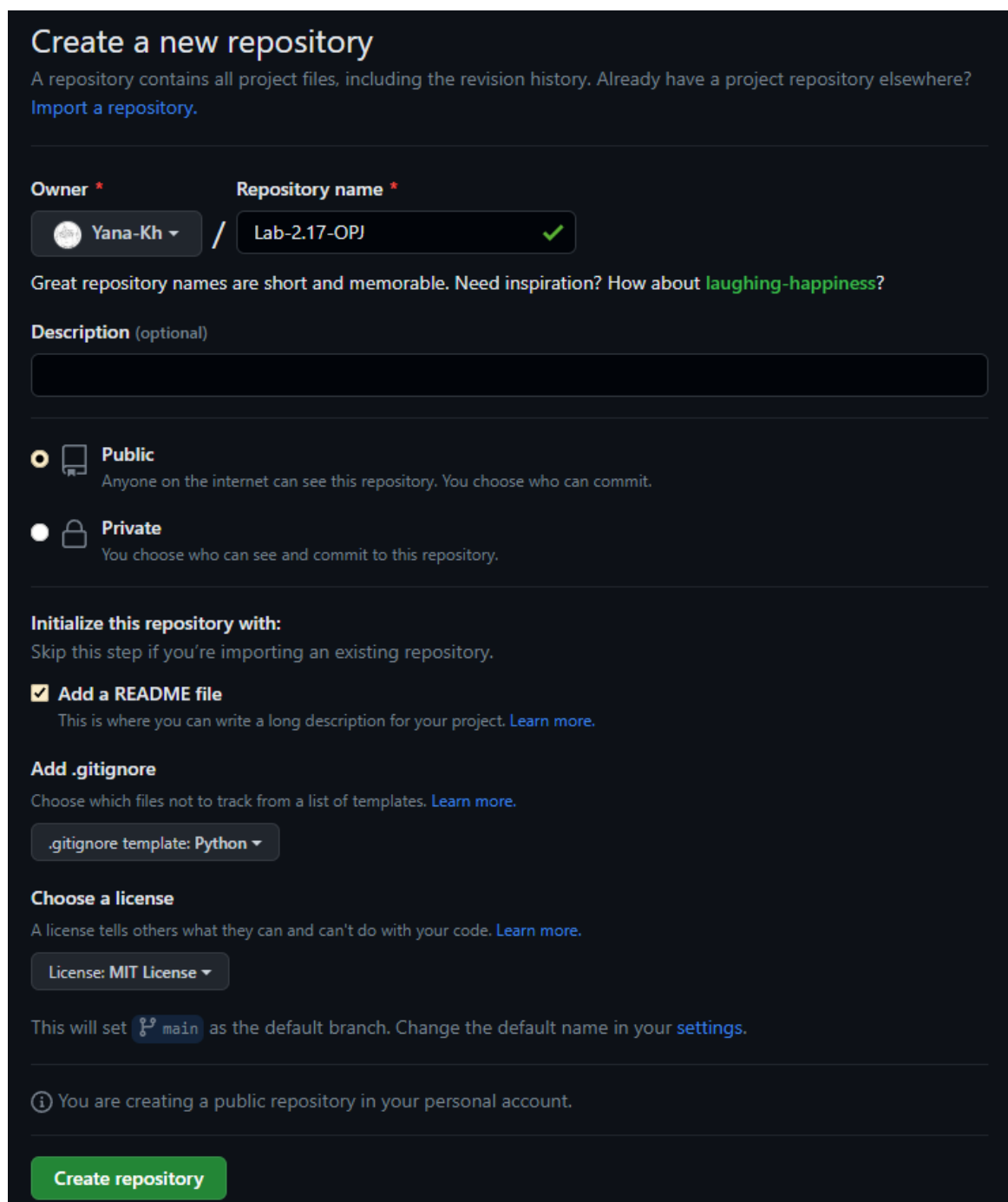
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

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

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.



The image shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and provides a brief explanation of what a repository is. Below this, there are two main input fields: 'Owner' and 'Repository name'. The 'Owner' field is set to 'Yana-Kh' and the 'Repository name' field is set to 'Lab-2.17-OPJ'. There is a green checkmark next to the repository name. Below these fields, there is a section for 'Description (optional)' with a text input area. Further down, there are two radio buttons for 'Public' and 'Private'. The 'Public' option is selected. Below this, there is a section for 'Initialize this repository with:' which includes a checkbox for 'Add a README file' (checked) and a dropdown for '.gitignore template: Python'. There is also a section for 'Choose a license' with a dropdown set to 'License: MIT License'. At the bottom, there is a green button labeled 'Create repository'.

Create a new repository

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

Owner * Repository name *

Yana-Kh / Lab-2.17-OPJ ✓

Great repository names are short and memorable. Need inspiration? How about [laughing-happiness?](#)

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:
Skip this step if you're importing an existing repository.

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

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

.gitignore template: Python

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

License: MIT License

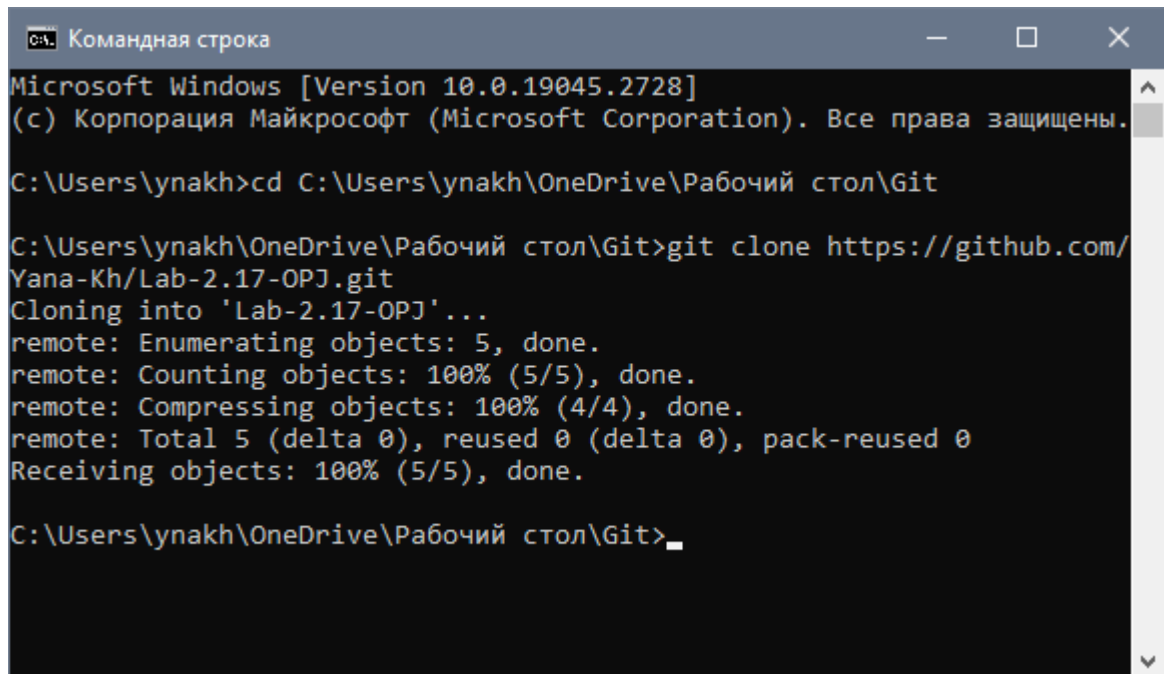
This will set `main` as the default branch. Change the default name in your [settings](#).

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

Create repository

Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.



```
Командная строка
Microsoft Windows [Version 10.0.19045.2728]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

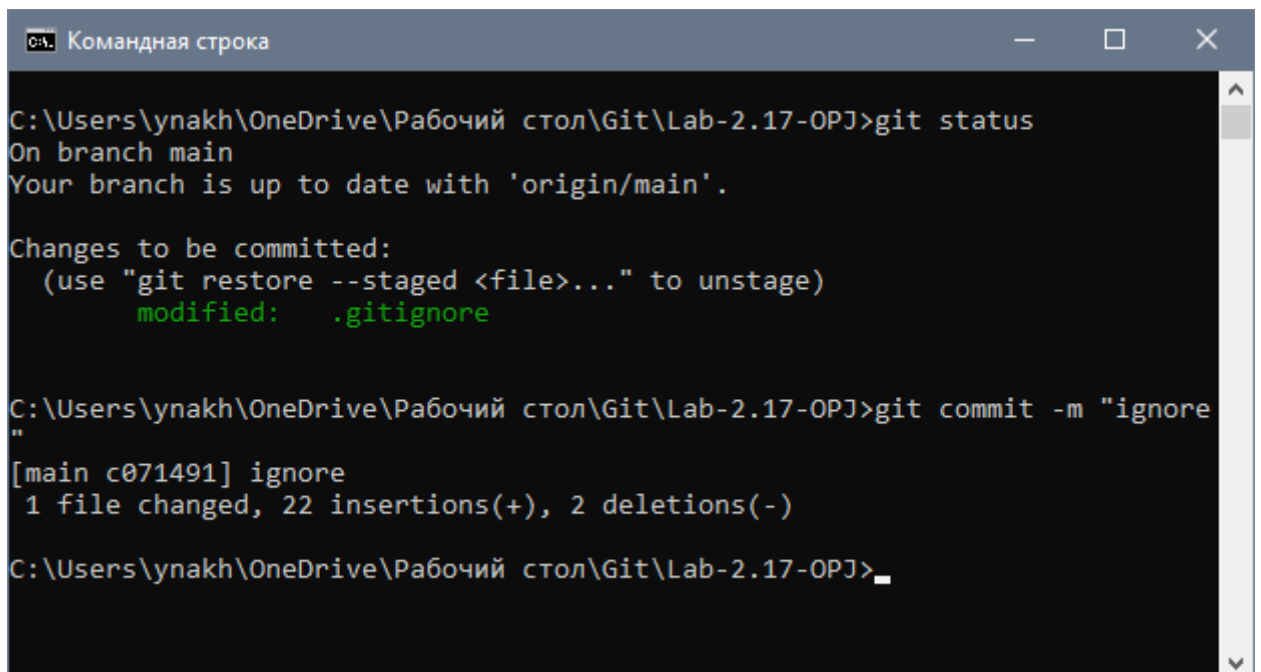
C:\Users\ynakh>cd C:\Users\ynakh\OneDrive\Рабочий стол\Git

C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/
Yana-Kh/Lab-2.17-OPJ.git
Cloning into 'Lab-2.17-OPJ'...
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\ynakh\OneDrive\Рабочий стол\Git>
```

Рисунок 2 – Клонирование репозитория

4. Дополните файл .gitignore необходимыми правилами для работы с IDE PyCharm.



```
Командная строка

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

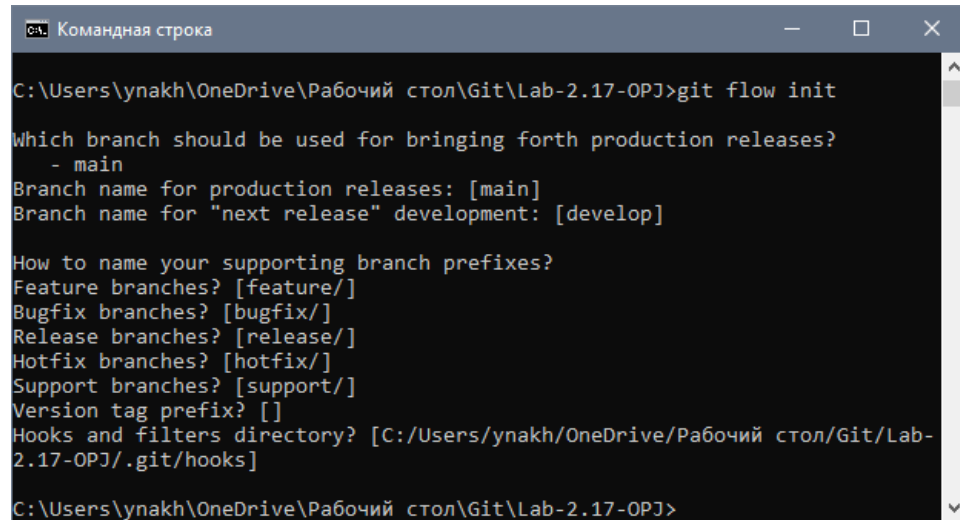
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ>git commit -m "ignore
"
[main c071491] ignore
1 file changed, 22 insertions(+), 2 deletions(-)

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.



```
Командная строка

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-ОПЈ>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/ynakh/OneDrive/Рабочий стол/Git/Lab-2.17-ОПЈ/.git/hooks]

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-ОПЈ>
```

Рисунок 4 – Организация репозитория в соответствии с моделью git-flow

6. Создайте проект PyCharm в папке репозитория.

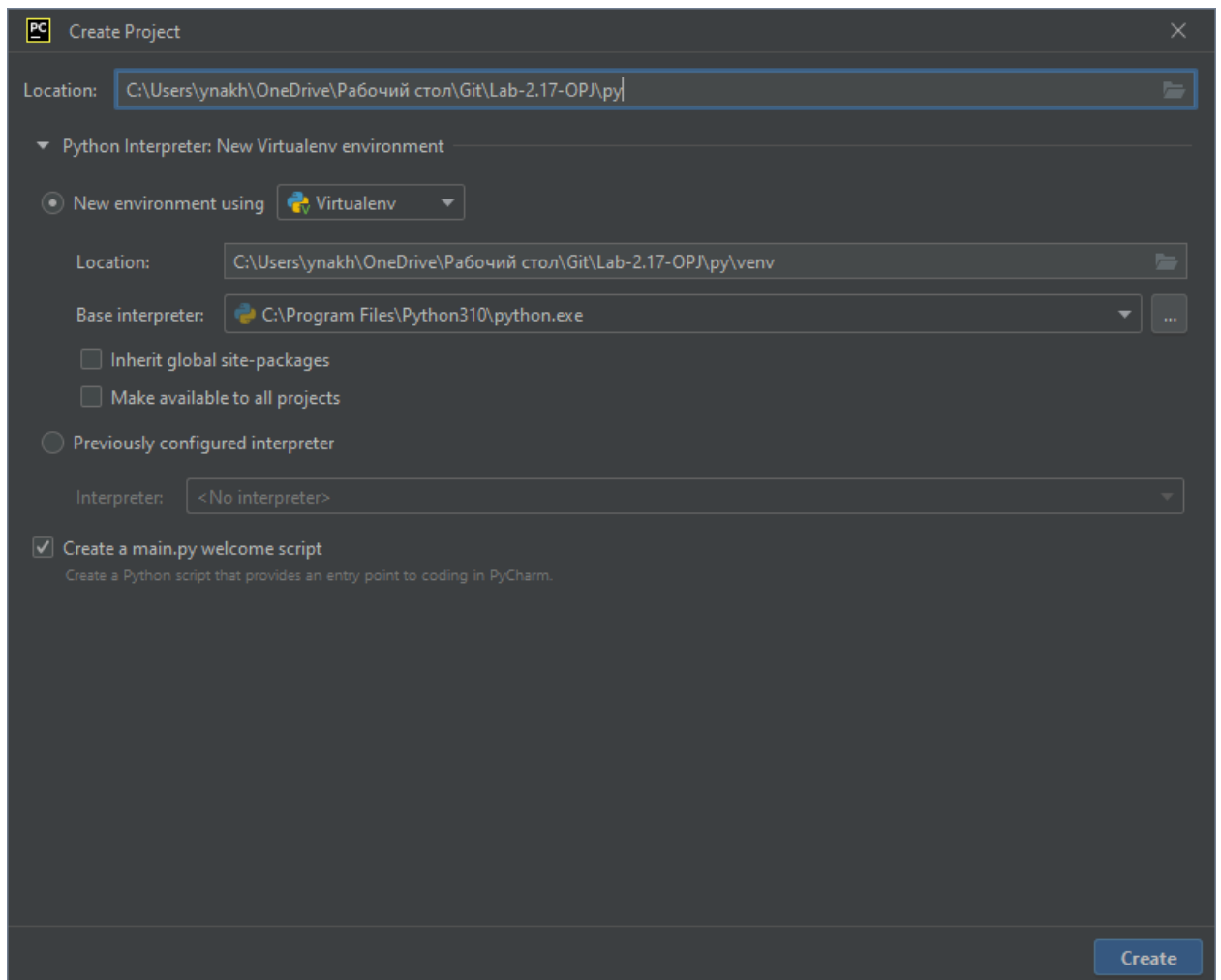


Рисунок 5 – Создание проекта PyCharm

7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных, вводимых с клавиатуры.

Пример 1.

Код:

```
#!/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(
                "№",
                "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()

```

```

(venv) C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>ex1.py add data.json --name="Петров Петр" --post="Директор" --year=2009
(venv) C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>ex1.py display data.json
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Сидоров Сидор | Главный инженер | 2012 |
+-----+-----+-----+-----+
| 2 | Петров Петр | Директор | 2009 |
+-----+-----+-----+-----+
(venv) C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>ex1.py select data.json --period=12
Список работников пуст.
(venv) C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>ex1.py select data.json --period=9
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Сидоров Сидор | Главный инженер | 2012 |
+-----+-----+-----+-----+
(venv) C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>

```

Рисунок 6 – Результат работы программы

8. Зафиксируйте сделанные изменения в репозитории.



Рисунок 7 – Фиксирование изменений в репозитории

9. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.

Задание: для своего варианта лабораторной работы 2.16 необходимо дополнительно реализовать интерфейс командной строки (CLI).

Код:

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

import json
import sys
import datetime
import os.path
import argparse

def add_human(staff, name, phone, str_bday):
    """
    Запросить данные о человеке.
    """
    b_day = list(map(int, str_bday.split(".")))
    date_bday = datetime.date(b_day[2], b_day[1], b_day[0])
    staff.append(
        {
            "name": name,
            "phone": phone,
            "birthday": date_bday
        }
    )

    return staff

def display_human(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "_" * 4,
            "_" * 30,
            "_" * 15,
            "_" * 15
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^15} | {:^15} |".format(
                "№",
                "Фамилия и имя",
                "Телефон",
                "День рождения"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
        for idx, human in enumerate(staff, 1):
```

```

        print(
            f"| {idx:>4} |"
            f' {human.get("name", ""):<30} |'
            f' {human.get("phone", 0):<15} |'
            f' {human.get("birthday")} |'
        )
        print(line)

    else:
        print("Список пуст.")

def find_human(staff, fname):
    """
    Выбрать работников с заданным стажем.
    """
    # Сформировать список людей.
    result = []
    for h in staff:
        if fname in str(h.values()):
            result.append(h)

    # Проверка на наличие записей
    if len(result) == 0:
        return print("Запись не найдена")

    # Возвратить список выбранных работников.
    return result

def json_deserial(obj):
    """
    Деериализация объектов datetime
    """
    for h in obj:
        if isinstance(h["birthday"], str):
            # print(datetime.strptime(h['birthday'], '%Y-%m-%d'))
            bday = list(map(int, h["birthday"].split("-")))
            h["birthday"] = datetime.date(bday[0], bday[1], bday[2])

def load_humans(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 json_serial(obj):
    """Сериализация объектов datetime"""

    if isinstance(obj, (datetime.datetime, datetime.date)):
        return obj.isoformat()

def save_humans(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,
default=json_serial)

def main(command_line=None):
    """
    Главная функция программы.
    """
    # 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("people")
    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",
        "--phone",
        type=int,
        action="store",
        help="The worker's post"
    )

    add.add_argument(
        "-bd",
        "--bday",
        action="store",
        required=True,
        help="The year of hiring"
    )

    # Создать субпарсер для отображения всех людей.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all workers"
    )

```

```

    )

    # Создать субпарсер для поиска людей по фамилии.
    find = subparsers.add_parser(
        "find",
        parents=[file_parser],
        help="Find the people"
    )

    find.add_argument(
        "-sn",
        "--surname",
        action="store",
        required=True,
        help="Required surname"
    )

    # 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):
        people = load_humans(args.filename)
    else:
        people = []

    # Добавить челоека.
    if args.command == "add":
        people = add_human(
            people,
            args.name,
            args.phone,
            args.bday
        )
        is_dirty = True

    # Отобразить всех людей.
    elif args.command == "display":
        display_human(people)

    # Выбрать требуемых рааботников.
    elif args.command == "find":
        selected = find_human(people, args.surname)
        display_human(selected)

    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_humans(args.filename, people)

if __name__ == "__main__":
    main()

```

```
Командная строка
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>id.py add data.json --name="Сидоров Петр" --phone=9086781234 --bday=06.09.2009
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>id.py add data.json --name="Иванов Даниил" --phone=9034567123 --bday=12.03.2000
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>id.py add data.json --name="Петрова Евгения" --phone=9097236757 --bday=30.11.2022
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>id.py display data.json
+-----+-----+-----+-----+
| № |      Фамилия и имя      |   Телефон   |   День рождения   |
+-----+-----+-----+-----+
| 1 | Сидоров Петр           | 9086781234  | 2009-09-06        |
+-----+-----+-----+-----+
| 2 | Иванов Даниил          | 9034567123  | 2000-03-12        |
+-----+-----+-----+-----+
| 3 | Петрова Евгения        | 9097236757  | 2022-11-30        |
+-----+-----+-----+-----+
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>id.py find data.json --surname="Петрова"
+-----+-----+-----+-----+
| № |      Фамилия и имя      |   Телефон   |   День рождения   |
+-----+-----+-----+-----+
| 1 | Петрова Евгения        | 9097236757  | 2022-11-30        |
+-----+-----+-----+-----+
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>id.py find data.json --surname="Тулин"
Запись не найдена
Список пуст.
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>
```

Рисунок 8 – Результат работы программы

Задание повышенной сложности

Самостоятельно изучите работу с пакетом `click` для построения интерфейса командной строки (CLI). Для своего варианта лабораторной работы 2.16 необходимо реализовать интерфейс командной строки с использованием пакета `click`.

Код:

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

import click
import json
import datetime
import os.path

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

@main.command('add')
@click.argument('filename')
@click.option('--name', help="The human's name")
@click.option('--phone', type=int, help="The human's phone")
@click.option('--bday', help="Birthday of a person")
def add_human(filename, name, phone, bday):
    """
    Запросить данные о человеке.
    """
    if os.path.exists(filename):
        people = load_humans(filename)
    else:
        people = []

    list_bday = list(map(int, bday.split(".")))
```

```

date_bday = datetime.date(list_bday[2], list_bday[1], list_bday[0])
people.append(
    {
        "name": name,
        "phone": phone,
        "birthday": date_bday
    }
)
save_humans(filename, people)
click.secho("Данные добавлены")

@main.command("display")
@click.argument('filename')
def display_CLI(filename):
    if os.path.exists(filename):
        people = load_humans(filename)
    else:
        people = []
    display_human(people)

@main.command("find")
@click.argument('filename')
@click.option('--surname', help="The human's name")
def find_human(filename, surname):
    """
    Выбрать работников с заданным стажем.
    """
    people = load_humans(filename)
    # Сформировать список людей.
    result = []
    for h in people:
        if surname in str(h.values()):
            result.append(h)

    # Проверка на наличие записей
    if len(result) == 0:
        return print("Запись не найдена")

    # Возвратить список выбранных работников.
    display_human(result)

def display_human(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "-" * 4,
            "-" * 30,
            "-" * 15,
            "-" * 15
        )
        print(line)
        print(
            "| {:^4} | {:^30} | {:^15} | {:^15} |".format(
                "№",
                "Фамилия и имя",
                "Телефон",
                "День рождения"
            )
        )

```

```

    )
    )
    print(line)

    # Вывести данные о всех сотрудниках.
    for idx, human in enumerate(staff, 1):
        print(
            f"| {idx:>4} |"
            f' {human.get("name", ""):<30} |'
            f' {human.get("phone", 0):<15} |'
            f' {human.get("birthday")} |'
        )
        print(line)

    else:
        print("Список пуст.")

def json_deserial(obj):
    """
    Деериализация объектов datetime
    """
    for h in obj:
        if isinstance(h["birthday"], str):
            # print(datetime.strptime(h['birthday'], '%Y-%m-%d'))
            bday = list(map(int, h["birthday"].split("-")))
            h["birthday"] = datetime.date(bday[0], bday[1], bday[2])

def load_humans(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 json_serial(obj):
    """Сериализация объектов datetime"""

    if isinstance(obj, (datetime.datetime, datetime.date)):
        return obj.isoformat()

def save_humans(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,
            default=json_serial)

if __name__ == "__main__":
    main()

```

```
Командная строка
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>hard.py add data.json --name="Попов Василий" --phone=9166578954 --bday=03.11.1995
Данные добавлены
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>hard.py display data.json
+-----+-----+-----+-----+
| № | Фамилия и имя | Телефон | День рождения |
+-----+-----+-----+-----+
| 1 | Сидоров Петр | 9086781234 | 2009-09-06 |
+-----+-----+-----+-----+
| 2 | Иванов Даниил | 9034567123 | 2000-03-12 |
+-----+-----+-----+-----+
| 3 | Петрова Евгения | 9097236757 | 2022-11-30 |
+-----+-----+-----+-----+
| 4 | Попов Василий | 9166578954 | 1995-11-03 |
+-----+-----+-----+-----+
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>hard.py find data.json --surname="Петрова"
+-----+-----+-----+-----+
| № | Фамилия и имя | Телефон | День рождения |
+-----+-----+-----+-----+
| 1 | Петрова Евгения | 9097236757 | 2022-11-30 |
+-----+-----+-----+-----+
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>hard.py find data.json --surname="Тулин"
Запись не найдена
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.17-OPJ\py>
```

Рисунок 9 – Результат работы программы

10. Зафиксируйте сделанные изменения в репозитории.

Рисунок 10 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

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 — компьютер с клавиатурой и монитором.

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?

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.

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`. Эквивалент `argc` – это просто количество элементов в списке. Чтобы получить это значение, используйте оператор `len()`.

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

Как вы могли заметить ранее, модуль `sys` разбивает строку командной строки только на отдельные фасы. 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. На практике для правильной обработки входных данных требуется модуль `sys`. Для этого необходимо заранее загрузить как модуль `sys`, так и модуль `getopt`. Затем из

списка входных параметров мы удаляем первый элемент списка (см. код ниже) и сохраняем оставшийся список аргументов командной строки в переменной с именем `arguments_list`.

```
# Include standard modules
import getopt, sys

# Get full command-line arguments
full_cmd_arguments = sys.argv

# Keep all but the first
argument_list = full_cmd_arguments[1:]

print(argument_list)
```

Аргументы в списке аргументов теперь можно анализировать с помощью метода `getopts()`. Но перед этим нам нужно сообщить `getopts()` о том, какие параметры допустимы. Они определены так:

```
short_options = "ho:v"
long_options = ["help", "output=", "verbose"]
```

Для метода `getopt()` необходимо настроить три параметра – список фактических аргументов из `argv`, а также допустимые короткие и длинные параметры. Сам вызов метода хранится в инструкции `try - catch`, чтобы скрыть ошибки во время оценки. Исключение возникает, если обнаруживается аргумент, который не является частью списка, как определено ранее. Скрипт в Python выведет сообщение об ошибке на экран и выйдет с кодом ошибки 2.

```
try:
    arguments, values = getopt.getopt(argument_list, short_options,
    long_options)
except getopt.error as err:
    # Output error, and return with an error code
    print(str(err))
    sys.exit(2)
```

Наконец, аргументы с соответствующими значениями сохраняются в двух переменных с именами `arguments` и `values`. Теперь вы можете легко

оценить эти переменные в своем коде. Мы можем использовать цикл `for` для перебора списка распознанных аргументов, одна запись за другой.

```
# Evaluate given options
for current_argument, current_value in arguments:
    if current_argument in ("-v", "--verbose"):
        print("Enabling verbose mode")
    elif current_argument in ("-h", "--help"):
        print("Displaying help")
    elif current_argument in ("-o", "--output"):
        print(f"Enabling special output mode ({current_value})")
```

Ниже вы можете увидеть результат выполнения этого кода. Далее показано, как программа реагирует как на допустимые, так и на недопустимые программные аргументы:

```
$ python arguments-getopt.py -h
Displaying help

$ python arguments-getopt.py --help
Displaying help

$ python arguments-getopt.py --output=green --help -v
Enabling special output mode (green)
Displaying help

Enabling verbose mode

$ python arguments-getopt.py -verbose
option -e not recognized
```

Последний вызов нашей программы поначалу может показаться немного запутанным. Чтобы понять это, вам нужно знать, что сокращенные параметры (иногда также называемые флагами) могут использоваться вместе с одним тире. Это позволяет вашему инструменту легче воспринимать множество вариантов.

6. Какие особенности построение CLI с использованием модуля `argparse`?

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.

Одним из аргументов противников включения argparse в Python был довод о том, что в стандартных модулях и без этого содержится две библиотеки для семантической обработки (парсинга) параметров командной строки. Однако, 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`). Позиционные аргументы — это аргументы, влияющие на работу программы, в зависимости от порядка, в котором они в эту программу передаются. Простейший пример — программа `cp`, имеющая минимум 2 таких аргумента («`cp source destination`»).

- argparse дает на выходе более качественные сообщения о подсказке при минимуме затрат (в этом плане при работе с `optparse` часто можно наблюдать некоторую избыточность кода);

- 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.