

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

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

**Кафедра инфокоммуникаций
«Основы цифровой обработки изображений в OpenCV»**

**Отчет по лабораторной работе № 2.21(8)
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: изучение взаимодействия с базами данных SQLite3 с помощью языка программирования Python.

Ход работы:

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT 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.](#)

Owner * Yana-Kh / Repository name * Lab-2.21-OPJ

✔ Lab-2.21-OPJ is available.

Great repository names are short and memorable. Need inspiration? How about **solid-fiesta**?

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.](#)

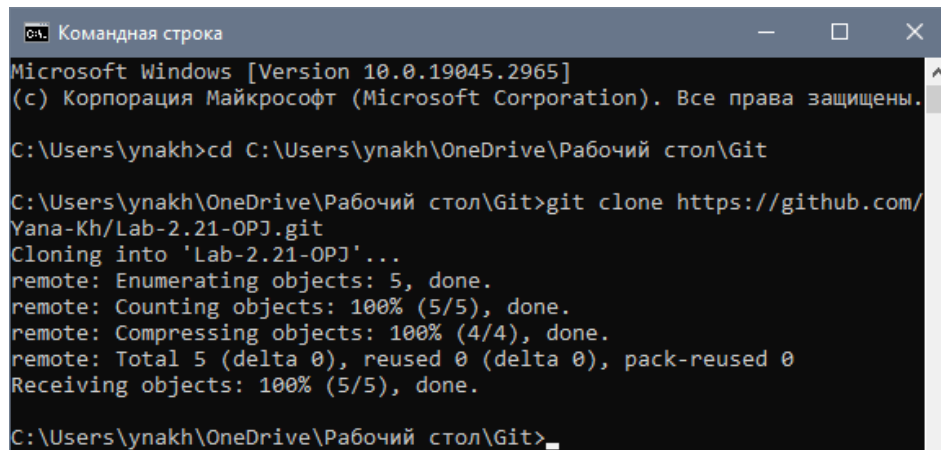
This will set **main** as the default branch. Change the default name in your [settings](#).

You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.



```
Командная строка
Microsoft Windows [Version 10.0.19045.2965]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

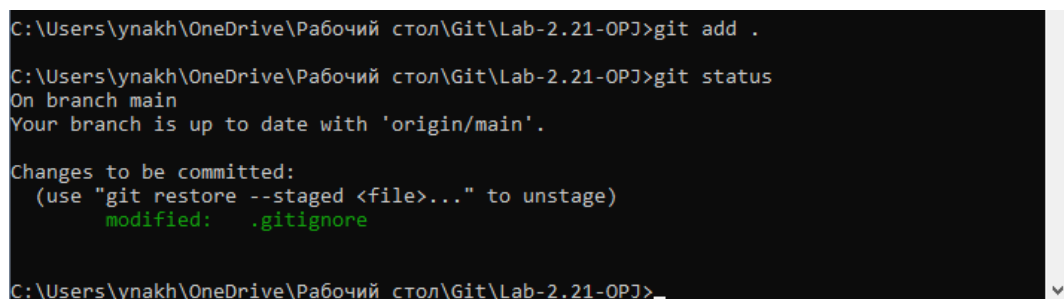
C:\Users\ynakh>cd C:\Users\ynakh\OneDrive\Рабочий стол\Git

C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/
Yana-Kh/Lab-2.21-OPJ.git
Cloning into 'Lab-2.21-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.



```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.21-OPJ>git add .

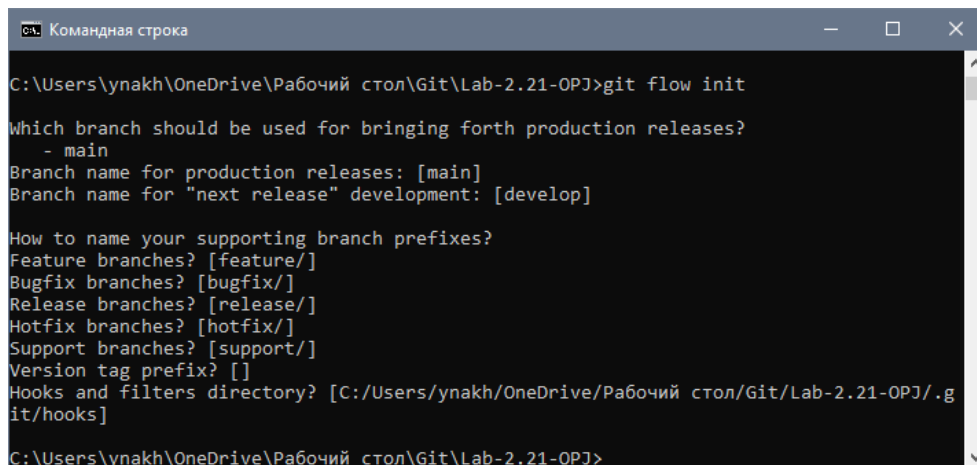
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.21-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

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

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.21-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-2.21-OPJ>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-2.21-OPJ/.git/hooks]

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.21-OPJ>
```

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

6. Создайте проект PyCharm в папке репозитория.

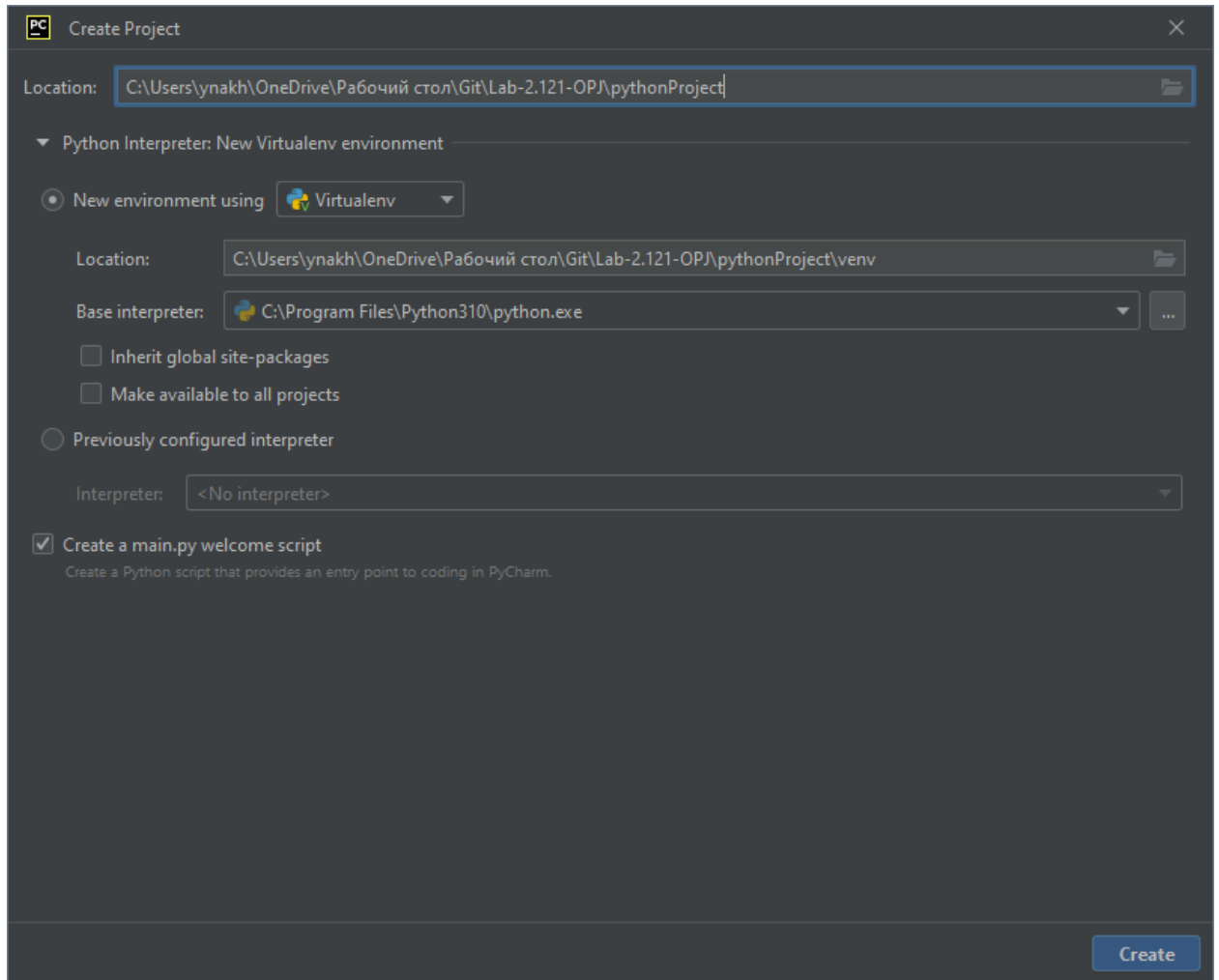


Рисунок 5 – Создание проекта PyCharm

7. Проработать примеры лабораторной работы. Создайте для них отдельные модули языка. Зафиксируйте изменения в репозитории.

Пример 1. Для примера 1 лабораторной работы 2.17 реализуйте возможность хранения данных в базе данных SQLite3.

Код:

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

import argparse
import sqlite3
import typing as t
from pathlib import Path
```

```

def display_workers(staff: t.List[t.Dict[str, t.Any]]) -> None:
    """
    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 create_db(database_path: Path) -> None:
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    # Создать таблицу с информацией о должностях.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS posts (
            post_id INTEGER PRIMARY KEY AUTOINCREMENT,
            post_title TEXT NOT NULL
        )
        """
    )
    # Создать таблицу с информацией о работниках.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS workers (
            worker_id INTEGER PRIMARY KEY AUTOINCREMENT,
            worker_name TEXT NOT NULL,
            post_id INTEGER NOT NULL,
            worker_year INTEGER NOT NULL,
            FOREIGN KEY(post_id) REFERENCES posts(post_id)
        )
        """
    )

```

```

    )
    conn.close()

def add_worker(
    database_path: Path,
    name: str,
    post: str,
    year: int
) -> None:
    """
    Добавить работника в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    # Получить идентификатор должности в базе данных.
    # Если такой записи нет, то добавить информацию о новой должности.
    cursor.execute(
        """
        SELECT post_id FROM posts WHERE post_title = ?
        """,
        (post,)
    )
    row = cursor.fetchone()
    if row is None:
        cursor.execute(
            """
            INSERT INTO posts (post_title) VALUES (?)
            """,
            (post,)
        )
        post_id = cursor.lastrowid
    else:
        post_id = row[0]
    # Добавить информацию о новом работнике.
    cursor.execute(
        """
        INSERT INTO workers (worker_name, post_id, worker_year)
        VALUES (?, ?, ?)
        """,
        (name, post_id, year)
    )
    conn.commit()
    conn.close()

def select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех работников.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT workers.worker_name, posts.post_title, workers.worker_year
        FROM workers
        INNER JOIN posts ON posts.post_id = workers.post_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "name": row[0],

```

```

        "post": row[1],
        "year": row[2],
    }
    for row in rows
]

def select_by_period(
    database_path: Path, period: int
) -> t.List[t.Dict[str, t.Any]]:
    """
    Выбрать всех работников с периодом работы больше заданного.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT workers.worker_name, posts.post_title, workers.worker_year
        FROM workers
        INNER JOIN posts ON posts.post_id = workers.post_id
        WHERE (strftime('%Y', date('now')) - workers.worker_year) >= ?
        """,
        (period,)
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "name": row[0],
            "post": row[1],
            "year": row[2],
        }
        for row in rows
    ]

def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "--db",
        action="store",
        required=False,
        ##### !!!!!!!!!!!!!!!!!!!!! Path.home() / "workers.db"
        default=str(Path.cwd() / "workers.db"),
        help="The database file name"
    )
    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version=f"%(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)
    # Получить путь к файлу базы данных.
    db_path = Path(args.db)
    create_db(db_path)
    # Добавить работника.
    if args.command == "add":
        add_worker(db_path, args.name, args.post, args.year)
    # Отобразить всех работников.
    elif args.command == "display":
        display_workers(select_all(db_path))
    # Выбрать требуемых работников.
    elif args.command == "select":
        display_workers(select_by_period(db_path, args.period))
    pass

if name == "main":
    main()

```



```
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>ex.py add --db="workers.db" --name="Сидоров Кирилл" --post="Programist" --year=2011
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>ex.py add --db="workers.db" --name="Петров Петр" --post="Programist" --year=2002
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>ex.py add --db="workers.db" --name="Иванов Иван" --post="Director" --year=2020
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>ex.py display
```

No	Ф.И.О.	Должность	Год
1	Сидоров Кирилл	Programist	2011
2	Петров Петр	Programist	2002
3	Иванов Иван	Director	2020

```
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>ex.py select --db="workers.db" --period=5
```

No	Ф.И.О.	Должность	Год
1	Сидоров Кирилл	Programist	2011
2	Петров Петр	Programist	2002

```
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>
```

Рисунок 6 – Результат работы программы

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

Задание: Для своего варианта лабораторной работы 2.17 необходимо реализовать хранение данных в базе данных SQLite3. Информация в базе данных должна храниться не менее чем в двух таблицах.

Код:

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

import sys
import argparse
import sqlite3
import typing as t
from pathlib import Path

def create_db(database_path: Path) -> None:
    """
    Создать базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    # Создать таблицу с информацией о должностях.
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS numbers (
            human_id INTEGER PRIMARY KEY AUTOINCREMENT,
            phone_number INTEGER NOT NULL
        )
        """
    )
    # Создать таблицу с информацией о работниках.
    cursor.execute(
        """
        """
    )
```

```

CREATE TABLE IF NOT EXISTS people (
    human_id INTEGER PRIMARY KEY AUTOINCREMENT,
    human_name TEXT NOT NULL,
    human_bd TEXT NOT NULL,
    FOREIGN KEY(human_id) REFERENCES numbers(human_id)
)
"""
)
conn.close()

def add_human(
    database_path: Path,
    name: str,
    phone: int,
    date_bday: str
) -> None:
    """
    Добавить человека в базу данных.
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()

    cursor.execute(
        """
        SELECT human_id FROM people WHERE human_name = ?
        """,
        (name,)
    )
    row = cursor.fetchone()

    if row is None:
        cursor.execute(
            """
            INSERT INTO people (human_name, human_bd) VALUES (?, ?)
            """,
            (name, date_bday)
        )
        human_id = cursor.lastrowid
    else:
        human_id = row[0]
        # Добавить информацию о новом человеке.
        cursor.execute(
            """
            INSERT INTO numbers (human_id, phone_numer)
            VALUES (?, ?)
            """,
            (human_id, phone)
        )
        conn.commit()
        conn.close()

def display_human(people: t.List[t.Dict[str, t.Any]]) -> None:
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if people:
        # Заголовок таблицы.
        line = "+-{}-+-{}-+-{}-+-{}-+".format(
            "_" * 4,
            "_" * 30,

```

```

        "-" * 15,
        "-" * 15
    )
    print(line)
    print(
        "| {:^4} | {:^30} | {:^15} | {:^15} |".format(
            "№",
            "Фамилия и имя",
            "День рождения",
            "Телефон"
        )
    )
    print(line)

    # Вывести данные о всех людях.
    for idx, human in enumerate(people, 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 select_all(database_path: Path) -> t.List[t.Dict[str, t.Any]]:
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT people.human_name, people.human_bd, numbers.phone_numer
        FROM numbers
        INNER JOIN people ON people.human_id = numbers.human_id
        """
    )
    rows = cursor.fetchall()
    conn.close()
    return [
        {
            "name": row[0],
            "phone": row[1],
            "birthday": row[2],
        }
        for row in rows
    ]

def find_human(
    database_path: Path, bd: str
) -> t.List[t.Dict[str, t.Any]]:
    """
    Вывод на экран информации о человека по фамилии
    """
    conn = sqlite3.connect(database_path)
    cursor = conn.cursor()
    cursor.execute(
        """
        SELECT people.human_name, people.human_bd, numbers.phone_numer
        FROM numbers
        INNER JOIN people ON people.human_id = numbers.human_id
        """
    )

```

```

        WHERE people.human_name LIKE ? || '%'
        """
        (bd,)
    )
    rows = cursor.fetchall()
    conn.close()
    if len(rows) == 0:
        return []

    return [
        {
            "name": row[0],
            "birthday": row[1],
            "phone": row[2],
        }
        for row in rows
    ]

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(
        "--db",
        action="store",
        required=False,
        default=str(Path.cwd() / "data_ph.db"),
        help="The database 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 people"
    )

    # 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)

    # Получить путь к файлу базы данных.
    db_path = Path(args.db)
    create_db(db_path)

    match args.command:
        # Добавить челооека.
        case "add":
            add_human(db_path, args.name, args.phone, args.bday)
        # Отобразить всех людей.
        case "display":
            display_human(select_all(db_path))
        # Выбрать требуемых людей
        case "find":
            display_human(find_human(db_path, args.surname))

if __name__ == "__main__":
    main()

```

```
C:\WINDOWS\system32\cmd.exe
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>id.py add -n="Романов Владимир" -p=1234567890 -bd=19.11.2013
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>id.py display
+-----+-----+-----+-----+
| № | Фамилия и имя | День рождения | Телефон |
+-----+-----+-----+-----+
| 1 | Иванов Петр | 10.09.2001 | 9345627132 |
+-----+-----+-----+-----+
| 2 | Сидоров Кирилл | 12.12.2010 | 1234229089 |
+-----+-----+-----+-----+
| 3 | Романов Владимир | 19.11.2013 | 1234567890 |
+-----+-----+-----+-----+
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>id.py find --surname="Иванов"
+-----+-----+-----+-----+
| № | Фамилия и имя | День рождения | Телефон |
+-----+-----+-----+-----+
| 1 | Иванов Петр | 10.09.2001 | 9345627132 |
+-----+-----+-----+-----+
C:\Users\антон\Desktop\Git\Lab-2.21-OPJ\py>
```

Рисунок 8 – Результат работы программы

9. Зафиксируйте сделанные изменения в репозитории.

..		
ex.py	ex	now
id.py	id	1 minute ago

Рисунок 9 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

1. Каково назначение модуля sqlite3?

Непосредственно модуль `sqlite3` – это API к СУБД SQLite. Своего рода адаптер, который переводит команды, написанные на Питоне, в команды, которые понимает SQLite. Как и наоборот, доставляет ответы от SQLite в python-программу.

2. Как выполняется соединение с базой данных SQLite3? Что такое курсор базы данных?

Чтобы использовать SQLite3 в Python, прежде всего, вам нужно будет импортировать модуль `sqlite3`, а затем создать объект соединения, который соединит нас с базой данных и позволит нам выполнять операторы SQL. Объект соединения создается с помощью функции `connect()`. Вызов функции `connect()` приводит к созданию объекта-экземпляра от класса `Connection`. Этот

объект обеспечивает связь с файлом базы данных, представляет конкретную БД в программе.

Для взаимодействия с базой данных SQLite3 в Python необходимо создать объект cursor. Вы можете создать его с помощью метода cursor() . Курсор SQLite3 – это метод объекта соединения. Для выполнения инструкций SQLite3 сначала устанавливается соединение, а затем создается объект курсора с использованием объекта соединения.

3. Как подключиться к базе данных SQLite3, находящейся в оперативной памяти компьютера?

Создать базу данных в оперативной памяти с помощью функции :memory: with the connect. Такая база данных называется базой данных в памяти.

4. Как корректно завершить работу с базой данных SQLite3?

con.close()

5. Как осуществляется вставка данных в таблицу базы данных SQLite3?

Чтобы вставить данные в таблицу, используется оператор INSERT INTO. Мы также можем передавать значения/аргументы оператору INSERT в методе execute ().

6. Как осуществляется обновление данных таблицы базы данных SQLite3?

Чтобы обновить данные в таблице, просто создайте соединение, затем создайте объект курсора с помощью соединения и, наконец, используйте оператор UPDATE в методе execute ().

7. Как осуществляется выборка данных из базы данных SQLite3?

Оператор `SELECT` используется для выбора данных из определенной таблицы. Если вы хотите выбрать все столбцы данных из таблицы, вы можете использовать звездочку (*). Синтаксис для этого будет следующим. В SQLite3 оператор `SELECT` выполняется в методе `execute` объекта `cursor`.

8. Каково назначение метода `rowcount`?

SQLite3 `rowcount` используется для возврата количества строк, которые были затронуты или выбраны последним выполненным SQL-запросом.

9. Как получить список всех таблиц базы данных SQLite3?

Чтобы перечислить все таблицы в базе данных SQLite3, вы должны запросить данные из таблицы `sqlite_master`, а затем использовать `fetchall()` для получения результатов из инструкции `SELECT`.

`sqlite_master` – это главная таблица в SQLite3, которая хранит все таблицы.

10. Как выполнить проверку существования таблицы как при ее добавлении, так и при ее удалении?

Чтобы проверить, не существует ли таблица уже, мы используем `IF NOT EXISTS` с оператором `CREATE TABLE`:

```
CREATE TABLE IF NOT EXISTS table_name (column1, column2, ..., columnN)
```

11. Как выполнить массовую вставку данных в базу данных SQLite3?

Метод `executemany` можно использовать для вставки нескольких строк одновременно.

12. Как осуществляется работа с датой и временем при работе с базами данных SQLite3?

В базе данных Python SQLite3 мы можем легко хранить дату или время, импортируя модуль `datetime`