

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ
ФЕДЕРАЦИИ**

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

Кафедра инфокоммуникаций

«Работа с файловой системе в Python3 с использованием модуля pathlib»

Отчет по лабораторной работе № 2.19

по дисциплине «Основы программной инженерии»

Выполнил студент группы ПИЖ-б-о-21-1

Халимендик Я. Д. « » 2023г.

Подпись студента _____

Работа защищена « » _____ 20__ г.

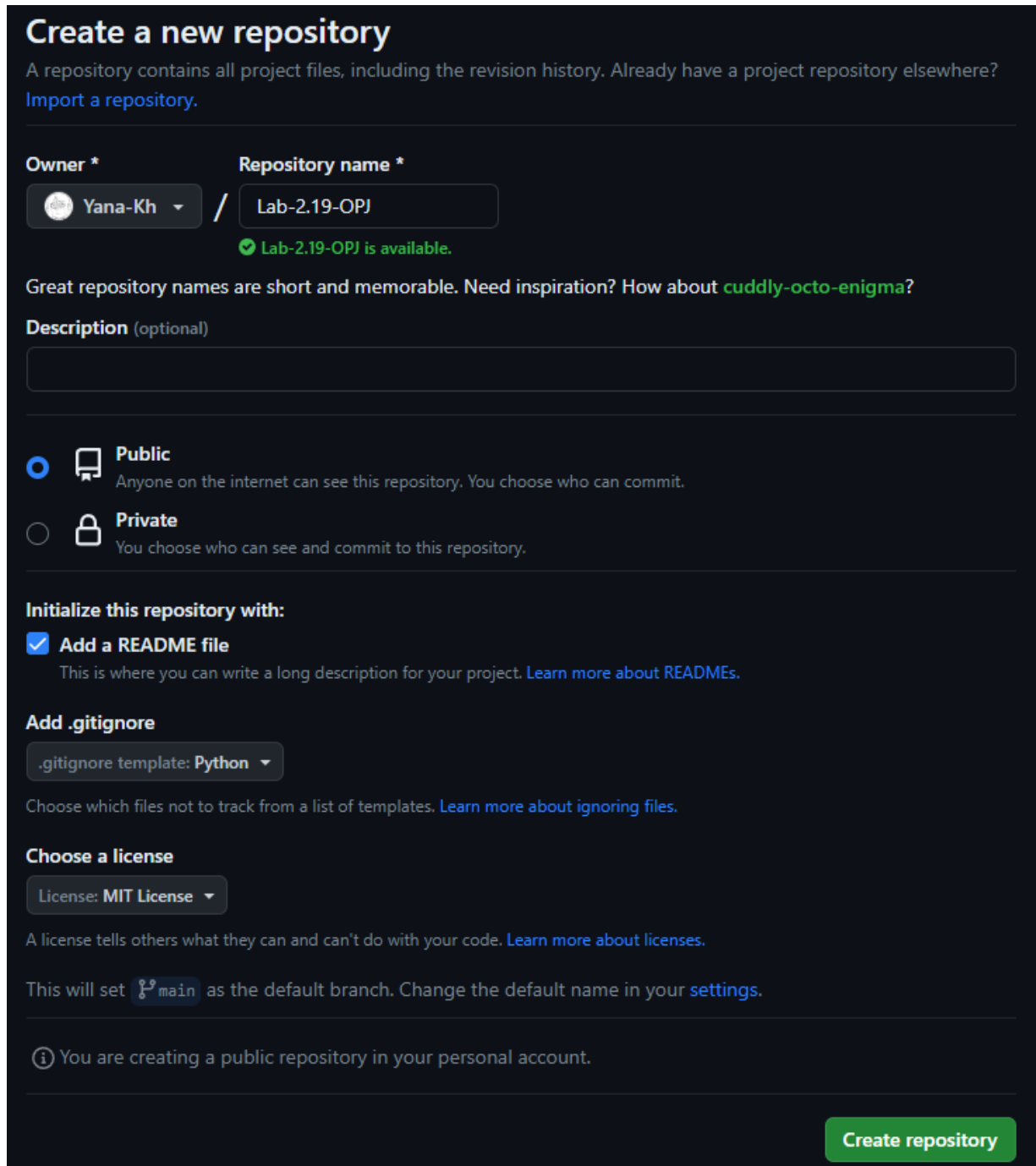
Проверил Воронкин Р.А. _____
(подпись)

Ставрополь 2023

Цель работы: приобретение навыков по работе с файловой системой с помощью библиотеки `pathlib` языка программирования Python версии 3.x.

Ход работы:

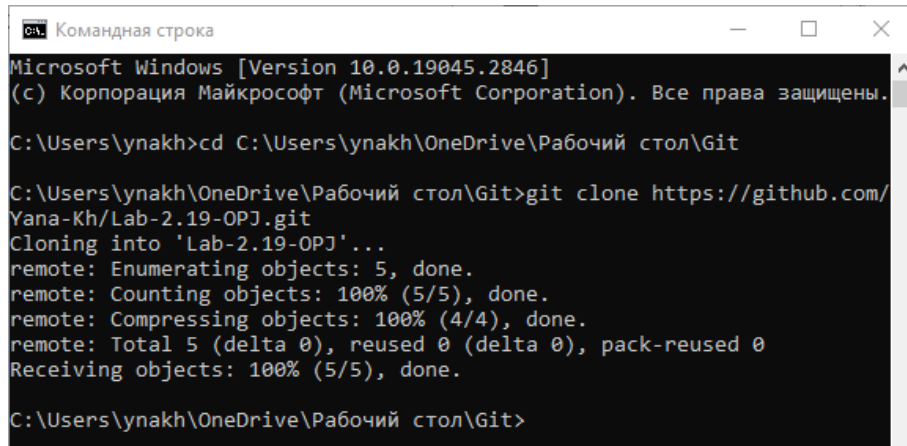
1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия IT и язык программирования Python.



The screenshot 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-2.19-OPJ'. A green checkmark indicates that the repository name is available. Below these fields, there is a section for 'Description' (optional) with a text input area. Further down, there are two radio button options for repository visibility: 'Public' (selected) and 'Private'. Below this, there is a section for 'Initialize this repository with:' which includes a checked checkbox for 'Add a README file'. There is also a section for 'Add .gitignore' with a dropdown menu set to 'Python'. Below that, there is a section for 'Choose a license' with a dropdown menu set to 'MIT License'. At the bottom right, there is a green button labeled 'Create repository'. A small note at the bottom left states: 'You are creating a public repository in your personal account.'

Рисунок 1 – Создание репозитория

3. Выполните клонирование созданного репозитория.



```
Командная строка
Microsoft Windows [Version 10.0.19045.2846]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

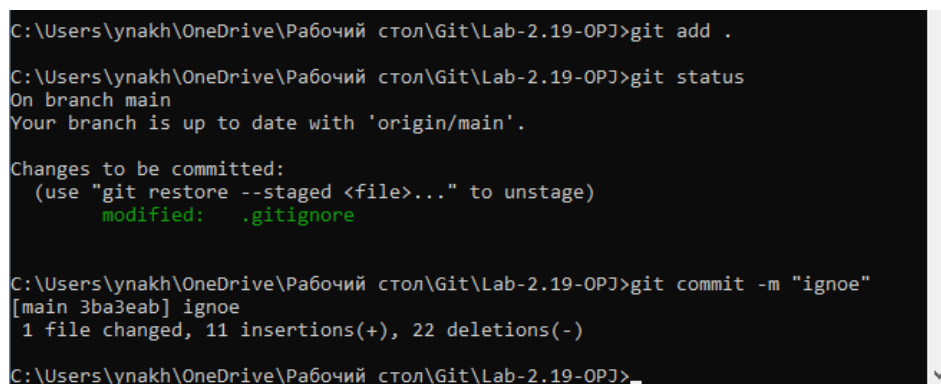
C:\Users\ynakh>cd C:\Users\ynakh\OneDrive\Рабочий стол\Git

C:\Users\ynakh\OneDrive\Рабочий стол\Git>git clone https://github.com/
Yana-Kh/Lab-2.19-OPJ.git
Cloning into 'Lab-2.19-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-2.19-OPJ>git add .

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ>git status
On branch main
Your branch is up to date with 'origin/main'.

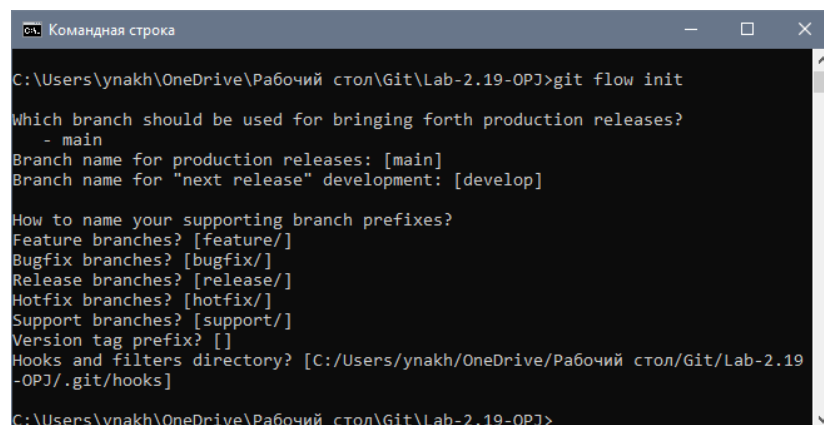
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   .gitignore

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ>git commit -m "ignoe"
[main 3ba3eab] ignoe
 1 file changed, 11 insertions(+), 22 deletions(-)

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ>
```

Рисунок 3 – Дополнение файла .gitignore

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



```
Командная строка

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-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-2.19-OPJ/.git/hooks]

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ>
```

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

6. Создайте проект PyCharm в папке репозитория.

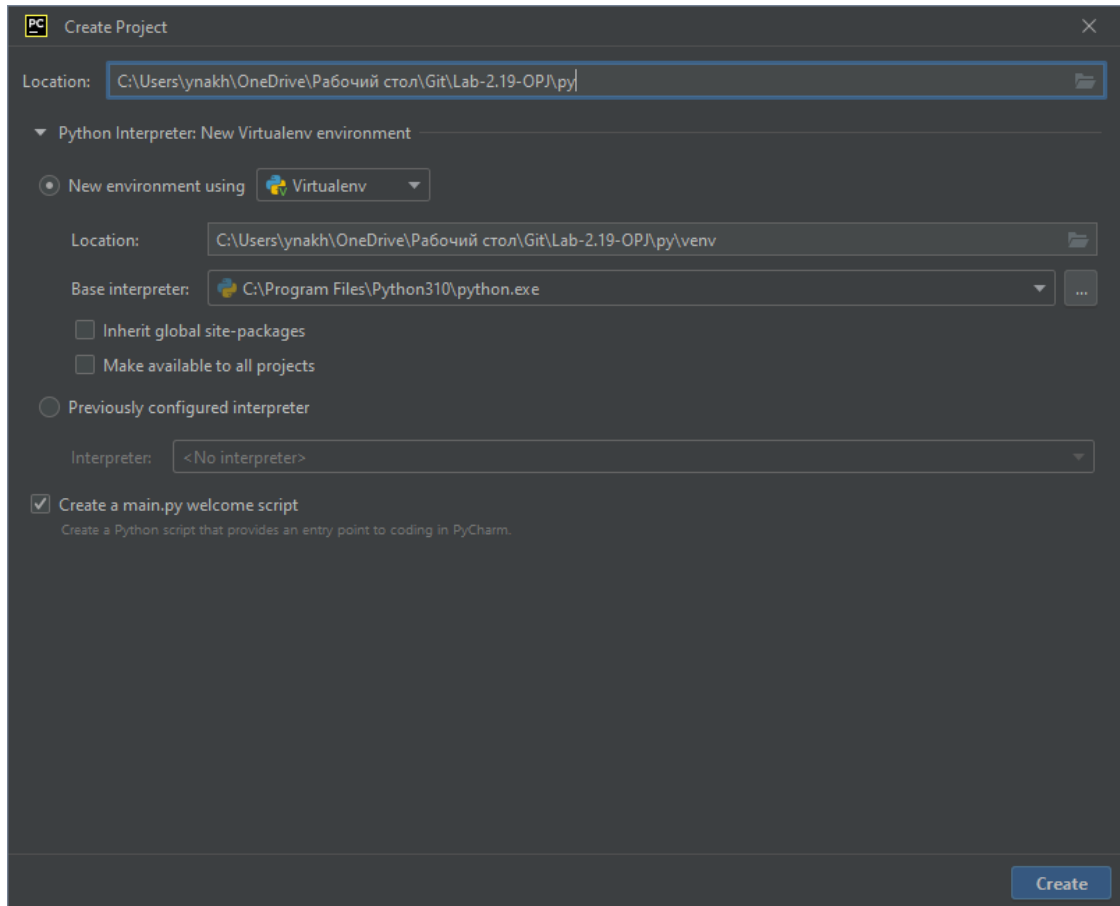


Рисунок 5 – Создание проекта PyCharm

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

Пример 1. Подсчет файлов

Код:

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

import pathlib
import collections

# Пример 1. Подсчет файлов
if __name__ == "__main__":
    print(collections.Counter(p.suffix for p in
        pathlib.Path.cwd().iterdir()))
```

```
Counter({' .py': 3, '': 2})
```

```
Process finished with exit code 0
```

Рисунок 6 – Результат работы программы

Пример 2. Показать дерево каталогов

Код:

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

import pathlib
import collections

# Пример 2. Подсчет файлов
if __name__ == "__main__":
    print(collections.Counter(p.suffix for p in pathlib.Path.cwd().glob('*.*')))
```

```
Counter({'*.py': 5})
```

```
Process finished with exit code 0
```

Рисунок 7 – Результат работы программы

Пример 3. Показать дерево каталогов

Код:

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

import pathlib

def tree(directory):
    print(f'+ {directory}')
    for path in sorted(directory.rglob('*')):
        depth = len(path.relative_to(directory).parts)
        spacer = ' ' * depth
        print(f'{spacer}+ {path.name}')

# Пример 3. Показать дерево каталогов
if __name__ == "__main__":
    print(tree(pathlib.Path.cwd()))
```

```
+ C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-0PJ\py
+ .idea
+ .gitignore
+ inspectionProfiles
+ profiles_settings.xml
+ Project_Default.xml
+ misc.xml
+ modules.xml
+ py.iml
+ vcs.xml
+ workspace.xml
+ ex1.py
+ ex2.py
+ ex3.py
+ ex4.py
+ ex5.py
+ venv
+ .gitignore
```

Рисунок 8 – Результат работы программы

Пример 4. Найти последний измененный файл

Код:

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

from datetime import datetime
import pathlib

# Пример 4. Найти последний измененный файл
if __name__ == "__main__":
    time, file_path = max((f.stat().st_mtime, f) for f in pathlib.Path.cwd().iterdir())
    print(datetime.fromtimestamp(time), file_path)
```

```
2023-05-04 13:21:55.314894 C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-0PJ\py\ex4.py
Process finished with exit code 0
```

Рисунок 9 – Результат работы программы

Пример 5. Создать уникальное имя файла

Код:

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

import pathlib

def unique_path(directory, name_pattern):
    counter = 0
    while True:
        counter += 1
        path = directory/name_pattern.format(counter)
        if not path.exists():
            return path

# Пример 5. Создать уникальное имя файла
if __name__ == "__main__":
    path = unique_path(pathlib.Path.cwd(), 'test{:03d}.txt')
    print(path)
```

```
C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\test001.txt
Process finished with exit code 0
```

Рисунок 10 – Результат работы программы

Пример 6.

Код:

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

import pathlib

if __name__ == "__main__":
    path = pathlib.Path(r"D:\lr6_test.txt")
    print(path.name)
    print(path.parent)
    print(path.exists())
```

```
lr6_test.txt
D:\
False
Process finished with exit code 0
```

Рисунок 11 – Результат работы программы







 ex1.py	example
 ex2.py	example
 ex3.py	example
 ex4.py	example
 ex5.py	example
 ex6.py	example

Рисунок 12 – Фиксирование изменений в репозитории

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

Задание 1:

Для своего варианта лабораторной работы 2.17 добавьте возможность хранения файла данных в домашнем каталоге пользователя. Для выполнения операций с файлами необходимо использовать модуль `pathlib`.

Код:

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

import json
import pathlib
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 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)

    path = pathlib.Path.home() / args.filename

    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if path.exists():
        people = load_humans(path)
    else:
        people = []

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

```

```

# Отобразить всех людей.
case "display":
    display_human(people)
# Выбрать требуемых людей
case "find":
    selected = find_human(people, args.surname)
    display_human(selected)

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

if __name__ == "__main__":
    main()

```

Командная строка

```

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\id>id1.py display data.json
+-----+-----+-----+-----+
| № | Фамилия и имя | Телефон | День рождения |
+-----+-----+-----+-----+
| 1 | Иванов Иван | 9188823454 | 2010-01-01 |
+-----+-----+-----+-----+
| 2 | Петров Петр | 9164537789 | 2012-12-12 |
+-----+-----+-----+-----+

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\id>id1.py add -n="Тулин Тихон" -p=9712345490 -bd="09.12.2011" data.json

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\id>id1.py display data.json
+-----+-----+-----+-----+
| № | Фамилия и имя | Телефон | День рождения |
+-----+-----+-----+-----+
| 1 | Иванов Иван | 9188823454 | 2010-01-01 |
+-----+-----+-----+-----+
| 2 | Петров Петр | 9164537789 | 2012-12-12 |
+-----+-----+-----+-----+
| 3 | Тулин Тихон | 9712345490 | 2011-12-09 |
+-----+-----+-----+-----+

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\id>id1.py find -sn="Петров" data.json
+-----+-----+-----+-----+
| № | Фамилия и имя | Телефон | День рождения |
+-----+-----+-----+-----+
| 1 | Петров Петр | 9164537789 | 2012-12-12 |
+-----+-----+-----+-----+

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\id>id1.py find -sn="Ушаков" data.json
Запись не найдена
Список пуст.

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\id>_

```

Рисунок 13 – Результат работы программы

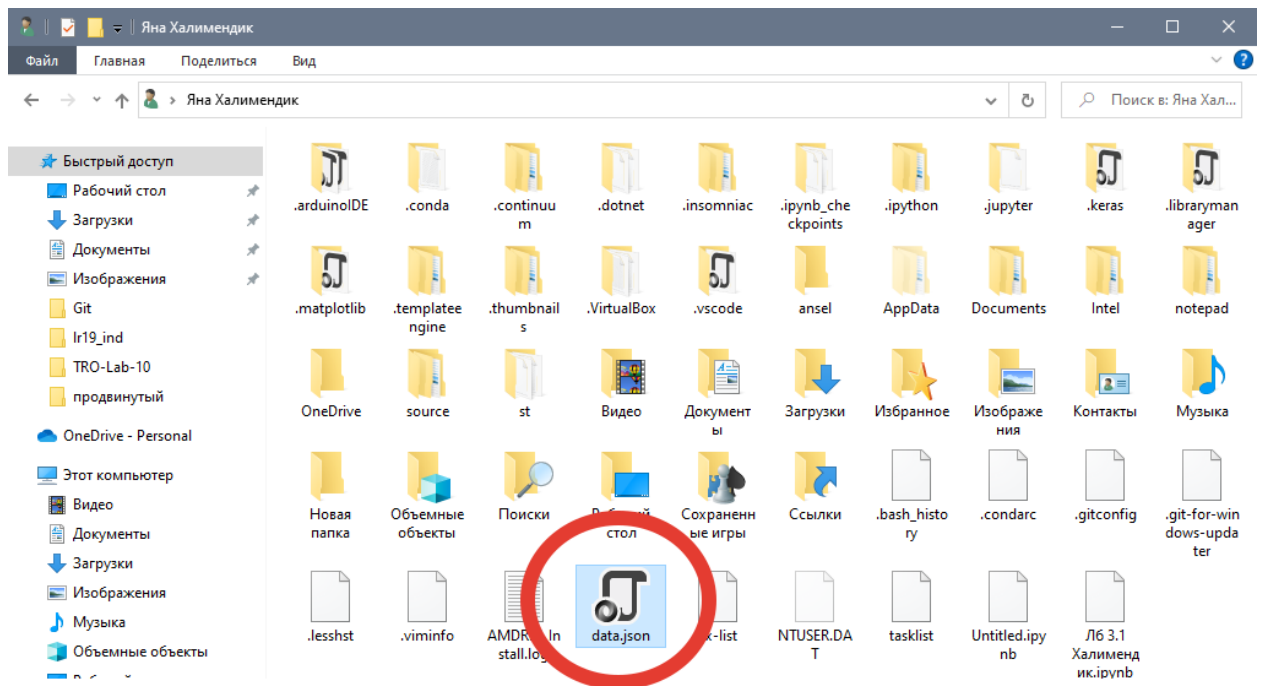


Рисунок 14 – Результат работы программы

Задание 2: Разработайте аналог утилиты tree в Linux. Используйте возможности модуля argparse для управления отображением дерева каталогов файловой системы. Добавьте дополнительные уникальные возможности в данный программный продукт.

Код:

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

import argparse
import pathlib
from colorama import init, Fore, Style

init() # Инициализация библиотеки colorama

def tree(directory, head='', tail='', level=0, colors=None):
    path = pathlib.Path(directory)

    if colors is None:
        colors = [Fore.LIGHTYELLOW_EX, Fore.YELLOW,
Fore.MAGENTA,Fore.LIGHTBLUE_EX]
        # Определение цветов для раскраски

    if path.is_dir():
        print(f"{colors[level]}{head}{path.name}{Style.RESET_ALL}")

        entries = sorted(path.iterdir())
        for i, entry in enumerate(entries):
            if level == 3:
                level = -1
            if i < len(entries) - 1:
                tree(entry, f"{tail} |—", f"{tail} | ", level=level + 1,
```

```

colors=colors)
    else:
        tree(entry, f"{tail} └─", f"{tail}   ", level=level + 1,
colors=colors)

def main():
    parser = argparse.ArgumentParser(add_help=False)
    # Добавление аргумента для пути к каталогу, который не является
    # обязательным
    # Если он не указан, то выводится текущий рабочий каталог
    parser.add_argument(
        "directory",
        type=pathlib.Path,
        default=pathlib.Path.cwd(),
        help="Каталог для отображения (по умолчанию: текущий каталог)")

    args = parser.parse_args()
    tree(args.directory, '', '', level=0)

if __name__ == "__main__":
    main()

```

```

C:\Users\ynakh\OneDrive\Рабочий стол\Git\Lab-2.19-OPJ\py\id>id2.py "D:\Виртуалка"
Виртуалка
├── Linux
│   ├── Linux
│   └── Logs
└── Win10
    └── Logs

```

Рисунок 9 – Результат работы программы

10. Зафиксируйте сделанные изменения в репозитории.

Рисунок 10 – Фиксирование изменений в репозитории

Вопросы для защиты работы:

1. Какие существовали средства для работы с файловой системой до Python 3.4?

До Python 3.4 работа с путями файловой системы осуществлялась либо с помощью методов строк:

```
path.split('\\', maxsplit=1)[0]
```

либо с помощью модуля `os.path` :

```
os.path.isfile(os.path.join(os.path.expanduser('~'), 'realpython.txt'))
```

2. Что регламентирует PEP 428?

PEP 428 - "The pathlib module - representing file system paths as objects" регламентирует использование модуля `pathlib` в Python.

3. Как осуществляется создание путей средствами модуля `pathlib`?

Создание путей средствами модуля `pathlib` осуществляется с помощью класса `pathlib.Path`. Прежде всего, существуют classmethods наподобие `.cwd()` (текущий рабочий каталог) и `.home()` (домашний каталог вашего пользователя). Путь также может быть явно создан из его строкового представления.

4. Как получить путь дочернего элемента файловой системы с помощью модуля `pathlib`?

Чтобы получить путь дочернего элемента файловой системы с помощью модуля `pathlib` в Python, можно использовать метод `joinpath()`.

Допустим, у вас есть объект `Path`, представляющий путь к родительскому каталогу, и вы хотите получить путь к дочернему элементу `child_dir` в этом каталоге. Для этого можно вызвать метод `joinpath()` на объекте `Path`, передав в него имя дочернего элемента в качестве аргумента.

5. Как получить путь к родительским элементам файловой системы с помощью модуля `pathlib`?

Для получения пути к родительским элементам файловой системы с помощью модуля `pathlib` в Python, можно использовать атрибут `parent` объекта `Path`. Этот атрибут возвращает объект `Path`, представляющий родительский каталог текущего элемента.

6. Как выполняются операции с файлами с помощью модуля pathlib?

Он позволяет выполнять операции с файлами с помощью объектов Path, которые представляют пути файловой системы.

```
from pathlib import Path
```

```
# создание файла
```

```
file_path = Path('/path/to/myfile.txt')
```

```
file_path.touch()
```

```
# чтение содержимого файла
```

```
file_path = Path('/path/to/myfile.txt')
```

```
with file_path.open() as f:
```

```
    contents = f.read()
```

Запись в файл:

```
# запись в файл
```

```
file_path = Path('/path/to/myfile.txt')
```

```
with file_path.open(mode='w') as f:
```

```
    f.write('Hello, world!')
```

Переименование файла:

```
# переименование файла
```

```
file_path = Path('/path/to/myfile.txt')
```

```
new_file_path = Path('/path/to/newfile.txt')
```

```
file_path.rename(new_file_path)
```

```
# удаление файла
```

```
file_path = Path('/path/to/myfile.txt')
```

```
file_path.unlink()
```


7. Как можно выделить компоненты пути файловой системы с помощью модуля pathlib?

Модуль `pathlib` в Python позволяет выделить различные компоненты пути файловой системы, такие как имя файла, расширение файла, родительский каталог и т.д.

```
from pathlib import Path
```

```
# получение имени файла
```

```
file_path = Path('/path/to/myfile.txt')
```

```
file_name = file_path.name
```

```
print(file_name) # 'myfile.txt'
```

```
# получение расширения файла
```

```
file_path = Path('/path/to/myfile.txt')
```

```
file_ext = file_path.suffix
```

```
print(file_ext) # '.txt'
```

```
# получение родительского каталога
```

```
file_path = Path('/path/to/myfile.txt')
```

```
parent_dir = file_path.parent
```

```
print(parent_dir) # '/path/to'
```

```
# получение всех компонентов пути
```

```
file_path = Path('/path/to/myfile.txt')
```

```
components = file_path.parts
```

```
print(components) # ('/', 'path', 'to', 'myfile.txt')
```

8. Как выполнить перемещение и удаление файлов с помощью модуля `pathlib`?

Модуль `pathlib` также предоставляет методы для перемещения и удаления каталогов, такие как `Path.rename()` и `Path.rmdir()`. Чтобы переместить файл, используйте `replace()`. Обратите внимание, что если место назначения уже существует, `replace()` перезапишет его. К сожалению, `pathlib` явно не поддерживает безопасное перемещение файлов. Чтобы избежать возможной перезаписи пути назначения, проще всего проверить, существует ли место назначения перед заменой.

9. Как выполнить подсчет файлов в файловой системе?

Есть несколько разных способов перечислить много файлов. Самым простым является метод `iterdir()`, который перебирает все файлы в данном каталоге. Более гибкие списки файлов могут быть созданы с помощью методов `.glob()` и `.rglob()` (рекурсивный глоб).

10. Как отобразить дерево каталогов файловой системы?

Для отображения дерева каталогов файловой системы в Python можно использовать модуль `pathlib` и рекурсивную функцию.

```
def tree(directory):  
    print(f' {directory}')  
    for path in sorted(directory.rglob('*')):  
        depth = len(path.relative_to(directory).parts)  
        spacer = ' ' * depth  
        print(f'{spacer} {path.name}')
```

11. Как создать уникальное имя файла?

Сначала укажите шаблон для имени файла с местом для счетчика. Затем проверьте существование пути к файлу, созданного путем соединения каталога и имени файла (со значением счетчика). Если он уже существует, увеличьте счетчик и попробуйте снова:

```
def unique_path(directory, name_pattern):  
    counter = 0  
    while True:  
        counter += 1  
        path = directory/name_pattern.format(counter)  
        if not path.exists():  
            return path
```

```
path = unique_path(pathlib.Path.cwd(), 'test{:03d}.txt')
```

12. Каковы отличия в использовании модуля `pathlib` для различных операционных систем?

Ранее мы отмечали, что когда мы создавали экземпляр `pathlib.Path`, возвращался либо объект `WindowsPath`, либо `PosixPath`. Тип объекта будет зависеть от операционной системы, которую вы используете. Эта функция позволяет довольно легко писать кроссплатформенный код. Можно явно запросить `WindowsPath` или `PosixPath`, но вы будете ограничивать свой код только этой системой без каких-либо преимуществ. Такой конкретный путь не может быть использован в другой системе.

В некоторых случаях может потребоваться представление пути без доступа к базовой файловой системе (в этом случае также может иметь смысл представлять путь `Windows` в системе, отличной от `Windows`, или наоборот). Это можно сделать с помощью объектов `PurePath`.

Вы можете напрямую создать экземпляр `PureWindowsPath` или `PurePosixPath` во всех системах. Создание экземпляра `PurePath` вернет один из этих объектов в зависимости от используемой операционной системы.