Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЁТ

по лабораторной работе №2.19

Дисциплина: «Анализ данных»
Тема: «Работа с файловой системе в Python3 с использованием модуля pathlib»

Выполнил:
Епифанов Алексей Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль)
«Программное обеспечение средств вычислительной
техники и автоматизированных систем
», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

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

Порядок выполнения работы:

1. Создал новый репозиторий, клонировал его, в нем создал ветку developer и перешел на нее.

2. Выполнил индивидуальное 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 .

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


import argparse
import bisect
import json
import os
from pathlib import Path

from jsonschema import ValidationError, validate


def add_route(routes, start, end, count):
    """
    Добавить данные о маршруте.
    """
    route = {
        "начальный пункт": start.lower(),
        "конечный пункт": end.lower(),
        "номер маршрута": count,
    }
    if route not in routes:
        bisect.insort(
            routes,
            route,
            key=lambda item: item.get("номер маршрута"),
        )
    else:
        print("Данный маршрут уже добавлен.")
    return routes
```

```
def display_routes(routes):
    """
    Отобразить список маршрутов.
    """
    if routes:
        line = "+-{}-+-{}-+-{}-+".format("-" * 30, "-" * 20, "-" * 8)
        print(line)
        print("| {:^30} | {:^20} | {:^8} |".format("Начало", "Конец", "Номер"))
        print(line)
        for route in routes:
            print(
                "| {:<30} | {:<20} | {:>8} |".format(
                    route.get("начальный пункт", ""),
                    route.get("конечный пункт", ""),
                    route.get("номер маршрута", ""),
                )
            )
        print(line)
    else:
        print("Список маршрутов пуст.")


def select_routes(routes, name_point):
    """
    Выбрать маршруты с заданным пунктом отправления или прибытия.
    """
    selected = []
    for route in routes:
        if (
            route["начальный пункт"] == name_point
            or route["конечный пункт"] == name_point
        ):
            selected.append(route)

    return selected


def save_routes(file_path, routes):
    """
    Сохранить все маршруты в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with file_path.open("w") as file_out:
```

```
        # Записать данные из словаря в формат JSON и сохранить их
        # в открытый файл.
        json.dump(routes, file_out, ensure_ascii=False, indent=4)


def load_routes(file_path):
    """
    Загрузить все маршруты из файла JSON.
    """
    schema = {
        "type": "array",
        "items": {
            "type": "object",
            "properties": {
                "начальный пункт": {"type": "string"},
                "конечный пункт": {"type": "string"},
                "номер маршрута": {"type": "integer"},
            },
            "required": [
                "начальный пункт",
                "конечный пункт",
                "номер маршрута",
            ],
        },
    }
    # Открыть файл с заданным именем и прочитать его содержимое.
    with file_path.open("r") as file_in:
        data = json.load(file_in)  # Прочитать данные из файла

    try:
        # Валидация
        validate(instance=data, schema=schema)
        print("JSON валиден по схеме.")
        return data
    except ValidationError as e:
        print(f"Ошибка валидации: {e.message}")
        return []


def main(command_line=None):
    """
    Главная функция программы.
    """
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
```

```
    "--home",
    action="store_true",
    help="Save the file in the user's home directory",
)
file_parser.add_argument(
    "filename", action="store", help="The data file name"
)
parser = argparse.ArgumentParser("routes")
parser.add_argument(
    "--version", action="version", version="%(prog)s 0.1.0"
)
subparsers = parser.add_subparsers(dest="command")
add = subparsers.add_parser(
    "add", parents=[file_parser], help="Add a new route"
)
add.add_argument(
    "-s", "--start", action="store", required=True, help="The route start"
)
add.add_argument(
    "-e", "--end", action="store", required=True, help="The route endpoint"
)
add.add_argument(
    "-n",
    "--number",
    action="store",
    type=int,
    required=True,
    help="The number of route",
)

_ = subparsers.add_parser(
    "list", parents=[file_parser], help="Display all routes"
)

select = subparsers.add_parser(
    "select", parents=[file_parser], help="Select the routes"
)
select.add_argument(
    "-p",
    "--point",
    action="store",
    required=True,
    help="Routes starting or ending at this point",
)
```

```
    args = parser.parse_args(command_line)

    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if args.home:
        filepath = Path.home() / args.filename
    else:
        filepath = Path(args.filename)
    if os.path.exists(filepath):
        routes = load_routes(filepath)
    else:
        routes = []

    match args.command.lower():
        case "add":
            routes = add_route(routes, args.start, args.end, args.number)
            is_dirty = True

        case "list":
            display_routes(routes)

        case "select":
            name_point = args.point.lower()
            selected = select_routes(routes, name_point)
            display_routes(selected)
    if is_dirty:
        save_routes(filepath, routes)


if __name__ == "__main__":
    main()
```

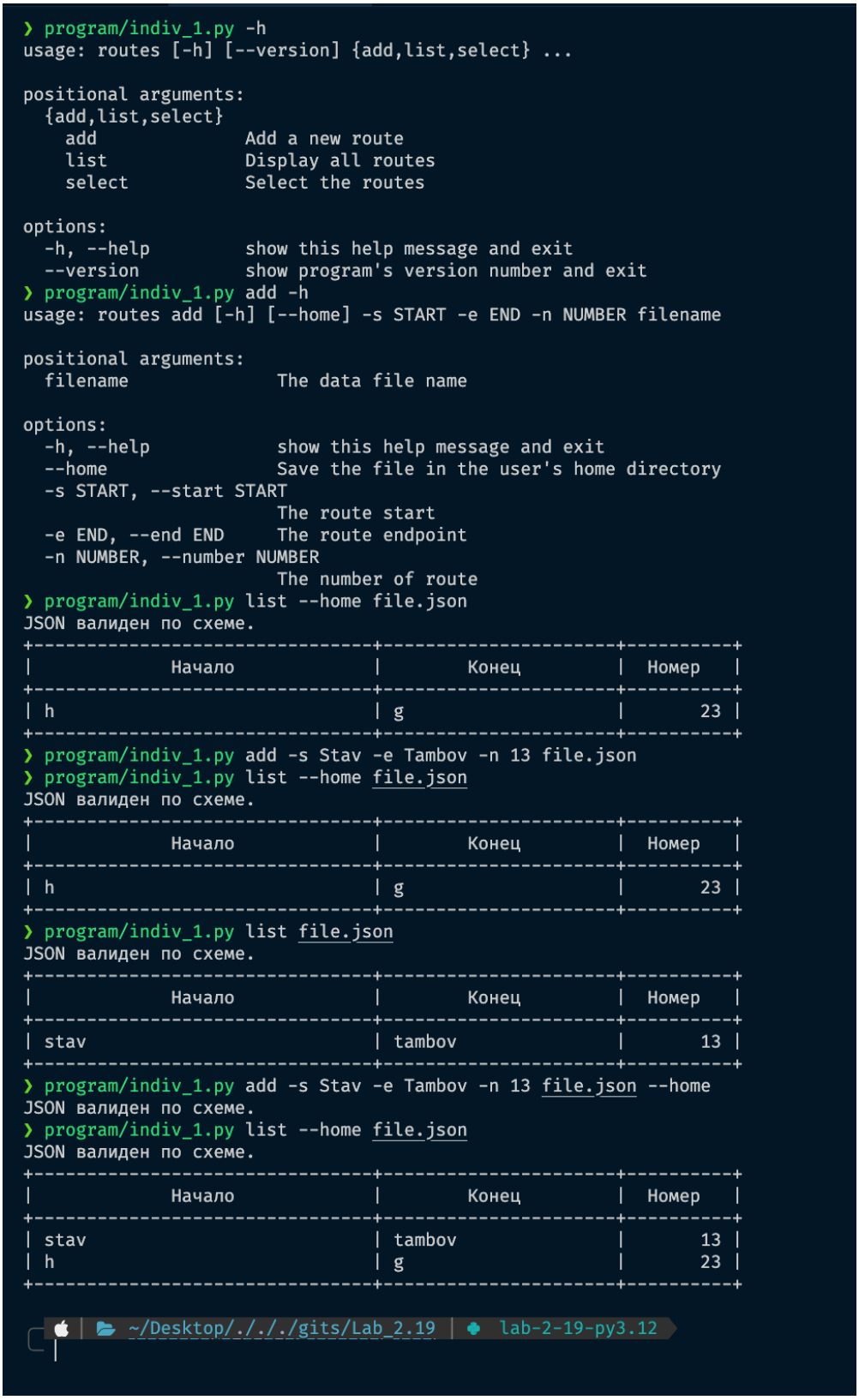

Рисунок 1. Вывод программы индивидуального задания 1

3. Выполнил индивидуальное 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.

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


import argparse
import sys
from pathlib import Path

from colorama import Fore, Style


def print_tree(tree, lines, level=0, levels=[]):
    """
    Отрисовка дерева каталогов в виде иерархической структуры.
    """
    if not tree:
        return

    for i, (node, child) in enumerate(tree.items()):
        if i == len(tree) - 1 and level != 0:
            levels[level - 1] = False
        if not lines:
            branch = "".join("│   " if lev else "    " for lev in levels[:-1])
            branch += " └── " if i == len(tree) - 1 else " ├── "
        else:
            branch = ""
        if level == 0:
            # Синий цвет для корневой папки
            print(Fore.BLUE + str(node) + Style.RESET_ALL)
        else:
            # Для файлов: зеленый цвет, для папок: желтый цвет
            color = Fore.GREEN if child is not None else Fore.YELLOW
            print(branch + color + str(node) + Style.RESET_ALL)

        print_tree(child, lines, level + 1, levels + [True])


def tree(directory, args):
    """
    Создание структуры дерева каталогов в виде словаря.
    """
```

```
sw = False
files = 0
folders = 0
folder_tree = {}
count = 0

path_list = []
all_files = args.a
max_depth = args.max_depth
only_dir = args.d
counter = 0
for path in directory.rglob("*"):
    try:
        if counter < 100000:
            counter += 1
        else:
            sw = True
            break
        if len(path_list) >= 1000:
            break
        if (
            max_depth
            and len(path.parts) - len(directory.parts) > max_depth
        ):
            continue
        if only_dir and not path.is_dir():
            continue
        if (not all_files) and (
            any(part.startswith(".") for part in path.parts)
        ):
            continue
        path_list.append(path)
    except PermissionError:
        pass
path_list.sort()

for path in path_list:
    count += 1
    relative_path = path.relative_to(directory)
    parts = relative_path.parts

    if args.f:
        path_work = relative_path
    else:
        path_work = Path(relative_path.name)
```

```
            current_level = folder_tree
            p = Path()
            for part in parts[:-1]:
                if args.f:
                    p = p / part
                else:
                    p = Path(part)
                current_level = current_level[p]

            if path.is_dir():
                current_level[path_work] = current_level.get(path_work, {})
                folders += 1
            else:
                current_level[path_work] = None
                files += 1
            if folders + files >= 1000:
                sw = True
                break

    print_tree({directory.name: folder_tree}, args.i)

    if sw:
        if folders + files < 1000:
            str_1 = "Вывод ограничен временем"
        else:
            str_1 = "Вывод ограничен по длине: 1000 элементов"
        print(Fore.RED, str_1, Style.RESET_ALL)
    print(
        Fore.YELLOW,
        files,
        Style.RESET_ALL,
        "files, ",
        Fore.GREEN,
        folders,
        Style.RESET_ALL,
        "folders.",
    )


def main(command_line=None):
    """
    Главная функция программы.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument(
```

```
        "-a", action="store_true", help="All files are printed."
    )
    parser.add_argument(
        "-d", action="store_true", help="Print directories only."
    )
    parser.add_argument("-f", action="store_true", help="Print relative path.")
    parser.add_argument(
        "-m",
        "--max_depth",
        type=int,
        default=None,
        help="Max depth of directories.",
    )
    parser.add_argument(
        "-i",
        action="store_true",
        help="Tree does not print the indentation lines."
        " Useful when used in conjunction with the -f option.",
    )
    parser.add_argument(
        "directory", nargs="?", default=".", help="Directory to scan."
    )
    args = parser.parse_args(command_line)

    try:
        directory = Path(args.directory).resolve(strict=True)
    except FileNotFoundError:
        print(f"Directory '{Path(args.directory).resolve()}' does not exist.")
        sys.exit(1)

    tree(directory, args)


if __name__ == "__main__":
    main()
```

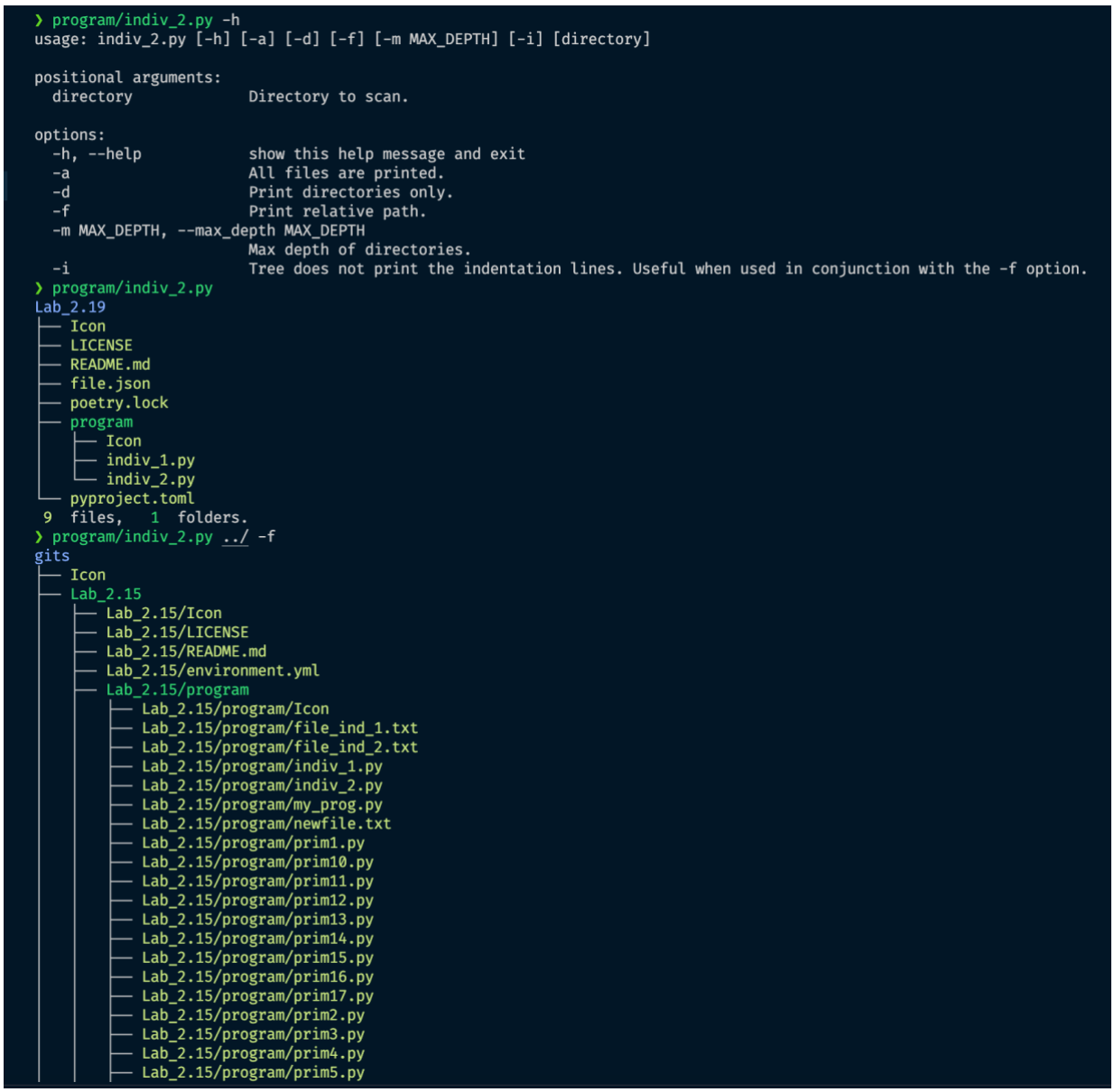

Рисунок 2. Вывод программы аналога tree

Ответы на контрольные вопросы:

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

– Методы строк, например path.rsplit('\\', maxsplit=1)[0]

– Модуль os.path

2. Что регламентирует PEP 428?

Модуль Pathlib – Объектно-ориентированные пути файловой системы

3. Как осуществляется создание путей средствами модуля pathlib?

Есть несколько разных способов создания пути. Прежде всего,

существуют classmethods наподобие .cwd() (текущий рабочий каталог) и .home() (домашний каталог вашего пользователя).

Также можно создать путь при помощи: pathlib.Path(«путь») или pathlib.Path.home()/'python'/'scripts'/'test.py'

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

При помощи метода resolve().

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

При помощи свойства parent.

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

source.replace(destination)

– перемещение;

file_to_delete.unlink()

– удаление файлов;

file_to_write.write_text('Привет, мир!')

content = file_to_read.read_text()

– чтение и запись файлов.

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

.name : имя файла без какого-либо каталога

.parent : каталог, содержащий файл, или родительский каталог, если путь является каталогом

.stem : имя файла без суффикса .suffix : расширение файла

.anchor : часть пути перед каталогами

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

.replace() – метод перемещения файлов

.unlink() – метод удаления файлов

9. Как выполнить подсчет файлов в файловой системе?

```
import collections
collections.Counter(p.suffix for p in pathlib.Path.cwd().iterdir())
Counter({'.md': 2, '.txt': 4, '.pdf': 2, '.py': 1})
```

10. Как отобразить дерево каталогов файловой системы?

```
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?

```
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.

Вывод: в результате выполнения лабораторной работы были приобретены навыки по работе с файловой системой с помощью библиотеки pathlib языка программирования Python версии 3.x.