

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

Институт цифрового развития  
Кафедра инфокоммуникаций

**ОТЧЕТ**  
**По лабораторной работе №5**  
**Дисциплины «Анализ данных»**

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и  
вычислительная техника (профиль)  
«Программное обеспечение средств  
вычислительной  
техники и автоматизированных  
систем», очная форма обучения

---

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических  
наук, доцент, доцент кафедры  
инфокоммуникаций

---

(подпись)

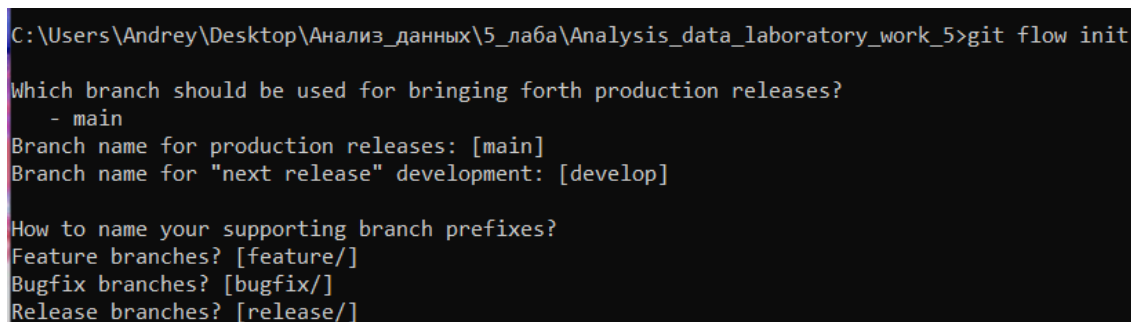
Ставрополь, 2024 г.

Тема: Работа с файловой системой в Python3 с использованием модуля pathlib.

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

Ход работы:

Создание общедоступного репозитория на «GitHub», клонирование репозитория, редактирование файла «.gitignore», организация репозитория согласно модели ветвления «git-flow» (рис. 1).



```
C:\Users\Andrey\Desktop\Анализ_данных\5_лаба\Analysis_data_laboratory_work_5>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/]
```

Рисунок 1 – Организация модели ветвления «git-flow»

Выполнение индивидуальных заданий:

Задание 1.

Необходимо для своего варианта лабораторной работы 2.17 добавить возможность хранения файла данных в домашнем каталоге пользователя. Для выполнения операции с файлами необходимо использовать модуль «pathlib».

Необходимо использовать словарь, содержащий следующие ключи: название пункта назначения; номер поезда; время отправления. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение (Вариант 26 (7), работа 2.8). Код программы индивидуального задания 1 (рис. 2).

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

import argparse
import json
import os.path
import pathlib

"""
Необходимо для своего варианта лабораторной работы 2.17 добавить возможность
хранения файла данных в домашнем каталоге пользователя. Для выполнения
операции
с файлами необходимо использовать модуль «pathlib».
"""

"""
Необходимо использовать словарь, содержащий следующие ключи: название пункта
назначения; номер поезда; время отправления. Написать программу, выполняющую
следующие действия: ввод с клавиатуры данных в список, состоящий из словарей
заданной структуры; записи должны быть упорядочены по времени отправления
поезда;
вывод на экран информации о поездах, направляющихся в пункт, название
которого
введено с клавиатуры; если таких поездов нет, выдать на дисплей
соответствующее
сообщение (Вариант 26 (7), работа 2.8).
"""

def add_train(trains, departure_point, number_train, time_departure,
destination):
    """
    Добавить данные о поезде.
    """
    trains.append(
        {
            "departure_point": departure_point,
            "number_train": number_train,
            "time_departure": time_departure,
            "destination": destination
        }
    )

    return trains

def display_trains(trains):
    """
    Отобразить список поездов со станциями.
    """
    # Проверить, что список поездов не пуст.
    if trains:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+-{}-+-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 13,
            '-' * 18,
            '-' * 14
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^13} | {:^18} | {:^14} |'.format(

```

```

        "№",
        "Пункт отправления",
        "Номер поезда",
        "Время отправления",
        "Пункт назначения"
    )
)
print(line)

# Вывести данные о всех поездах со станциями.
for idx, train in enumerate(trains, 1):
    print(
        '| {:>4} | {:<30} | {:<13} | {:>18} | {:^16} |'.format(
            idx, train.get('departure_point', ''),
            train.get('number_train', ''),
            train.get('time_departure', ''),
            train.get('destination', '')
        )
    )
print(line)

else:
    print("Список поездов пуст.")

def select_trains(trains, point_user):
    """
    Выбрать поезда по пункту назначения.
    """
    # Сформировать список поездов.
    result = []
    for train in trains:
        if point_user == str.lower(train['destination']):
            result.append(train)

    # Возвратить список выбранных поездов, направляющихся в пункт.
    return result

def save_trains(file_name, trains):
    """
    Сохранить все поезда со станциями 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(trains, fout, ensure_ascii=False, indent=4)

def load_trains(file_name):
    """
    Загрузить все поезда со станциями 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"
    )
    file_parser.add_argument(
        "--own",
        action="store_true",
        help="Save data file in own directory.",
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("trains")
    parser.add_argument(
        "--version",
        action="version",
        version="% (prog)s 0.1.0"
    )

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

    # Создать субпарсер для добавления поезда.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new train"
    )
    add.add_argument(
        "-dep",
        "--departure_point",
        action="store",
        required=True,
        help="The train's departure point"
    )
    add.add_argument(
        "-n",
        "--number_train",
        action="store",
        required=True,
        help="The train's number"
    )
    add.add_argument(
        "-t",
        "--time_departure",
        action="store",
        required=True,
        help="The time departure of train"
    )
    add.add_argument(
        "-des",
        "--destination",
        action="store",
        required=True,
        help="The destination of train"
    )

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

    # Создать субпарсер для выбора поездов по пунктам назначения.
    select = subparsers.add_parser(
        "select",

```

```

        parents=[file_parser],
        help="Select the trains"
    )
    select.add_argument(
        "-P",
        "--point_user",
        action="store",
        required=True,
        help="The required point"
    )

# Выполнить разбор аргументов командной строки.
args = parser.parse_args(command_line)

# Загрузить все поезда со станциями из файла, если файл существует.
is_dirty = False
if args.own:
    filepath = pathlib.Path.home() / args.filename
else:
    filepath = pathlib.Path(args.filename)
if os.path.exists(filepath):
    trains = load_trains(filepath)
else:
    trains = []

# Добавить поезд со станциями.
if args.command == "add":
    trains = add_train(
        trains,
        args.departure_point,
        args.number_train,
        args.time_departure,
        args.destination
    )
    is_dirty = True

# Отобразить все поезда со станциями.
elif args.command == "display":
    display_trains(trains)

# Выбрать требуемые поезда.
elif args.command == "select":
    selected = select_trains(trains, args.point_user)
    display_trains(selected)

# Сохранить данные в файл, если список поездов был изменен.
if is_dirty:
    save_trains(filepath, trains)

if __name__ == "__main__":
    main()

```

Рисунок 2 – Код программы индивидуального задания 1

Результаты работы программы при использовании соответствующего параметра в командной строке для сохранения файла в домашнем каталоге пользователя «Andrey» («python individual\_1.py add --own --filename data.json -

--departure\_point "Москва" --number\_train "123" --time\_departure "12:30" --destination "Санкт-Петербург"») (рис. 3).

```
C:\Users\Andrey\Desktop\Анализ_данных\5_лаба\Analysis_data_laboratory_work_5\individual>python individual_1.py add --own --filename data.json --departure_point "Москва" --number_train "123" --time_departure "12:30" --destination "Санкт-Петербург"
```

```
C:\Users\Andrey\Desktop\Анализ_данных\5_лаба\Analysis_data_laboratory_work_5\individual>python individual_1.py display --own --filename data.json
```

№	Пункт отправления	Номер поезда	Время отправления	Пункт назначения
1	Москва	123	12:30	Санкт-Петербург

Рисунок 3 – Результаты работы программы индивидуального задания 1

Созданный файл «data.json», который создался в домашнем каталоге (рис. 4).

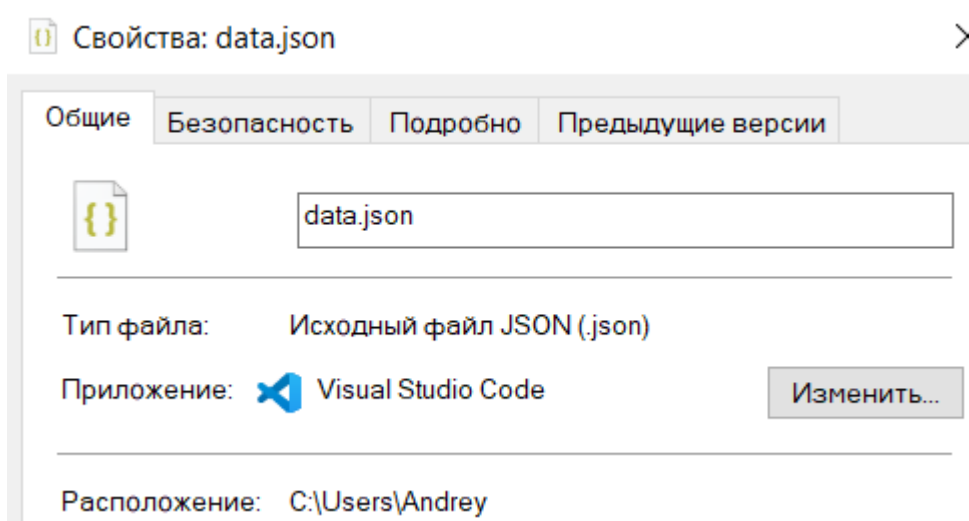


Рисунок 4 – Файл «data.json»

## Задание 2.

Необходимо разработать аналог утилиты «three» в Linux. Необходимо использовать возможности модуля «argparse» для управления отображением дерева каталогов файловой системы и добавить дополнительные уникальные возможности в программу.

Программа должна выводить дерево каталогов и файлов при указании в качестве параметра пути к каталогу.

Программа имеет ключи: «--file» и «--directory» для вывода только файлов или только каталогов соответственно.

Код программы для решения данной задачи (рис. 5).

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

import argparse
```

```

import sys
import pathlib

"""
Необходимо разработать аналог утилиты «three» в Linux.
Необходимо использовать возможности модуля «argparse» для управления
отображением
дерева каталогов файловой системы и добавить дополнительные уникальные
возможности в программу.
Программа должна выводить дерево каталогов и файлов при указании в качестве
параметра пути к каталогу.
"""

def tree(directory, args, prefix="", level=0):
    """
    Функция, которая рекурсивно выводит содержимое каталога.
    """

    # Получение содержимого текущего каталога
    contents = list(directory.iterdir())

    # Если задан параметр --directory, то выводятся только каталоги
    if args.directory:
        filtered_contents = []
        for file in contents:
            if file.is_dir(): # Проверка текущего объекта
                filtered_contents.append(file)
        contents = filtered_contents

    # Если задан --file, выводятся только файлы.
    if args.file:
        filtered_contents = []
        for file in contents:
            if file.is_file(): # Проверка объекта на то, является ли он
                # файлом
                filtered_contents.append(file)
        contents = filtered_contents

    """
    Команда tree обычно использовала декорации, чтобы показать дерево
    каталогов.
    Подсчитывается количество файлов в текущем каталоге.
    Перед последним ставится декорация └─.
    """

    decoration = ["└─ "] * (len(contents) - 1) + ["└─ "]

    # Анализ и вывод полученных данных
    for pointer, path in zip(decoration, contents):
        print(prefix + pointer + path.name)
        # Если текущий элемент - каталог, то вызывается функция tree для
        # этого каталога
        if path.is_dir():
            # Определение украшения для вложенных элементов (| или отступ)
            if pointer == "└─ ":
                extension = "|  "
            else:
                extension = "    "
            tree(path, args, prefix=prefix + extension, level=level + 1)

def main(command_line=None):
    """

```



```

Главная функция программы.
"""
# Создать основной парсер командной строки
parser = argparse.ArgumentParser()
parser.add_argument(
    "directory",
    type=str,
    help="The directory to list."
)

# Необходимо запретить одновременное использование --file и --directory
choose = parser.add_mutually_exclusive_group()
# Выводятся только каталоги
choose.add_argument(
    "--directory",
    action="store_true",
    help="List directories only."
)
# Выводятся только файлы
choose.add_argument(
    "--file",
    action="store_true",
    help="List files only."
)
parser.add_argument(
    "--version",
    action="version",
    version="% (prog)s 0.0.1"
)

# Разбор аргументов командной строки
args = parser.parse_args(command_line)
try:
    directory = pathlib.Path(args.directory).resolve(strict=True)
except FileNotFoundError:
    print("Этот файл не был найден!", file=sys.stderr)
    sys.exit(1)
except Exception as e:
    print(f"Error: {e}", file=sys.stderr)
    sys.exit(1)
tree(directory, args)

if __name__ == "__main__":
    main()

```

Рисунок 5 – Код программы индивидуального задания 2

Результаты работы программы с параметрами и без них (рис. 6).

```
C:\Users\Andrey\Desktop\Анализ_данных\5_лаба\Analysis_data_laboratory_work_5\individual>python individual_3.py "C:\Users\Andrey\Desktop\Анализ_данных\5_лаба"
Analysis_data_laboratory_work_5
├── .git
│   ├── hooks
│   ├── info
│   ├── logs
│   └── refs
│       ├── heads
│       │   └── feature
│       ├── remotes
│       └── origin
├── objects
│   ├── e6
│   ├── info
│   └── pack
├── refs
│   ├── heads
│   │   └── feature
│   ├── remotes
│   └── origin
├── tags
├── .idea
│   └── inspectionProfiles
├── doc
└── individual
```

```
C:\Users\Andrey\Desktop\Анализ_данных\5_лаба\Analysis_data_laboratory_work_5\individual>python individual_3.py --directory "C:\Users\Andrey\Desktop\Анализ_данных\5_лаба"
Analysis_data_laboratory_work_5
├── .git
│   ├── hooks
│   ├── info
│   ├── logs
│   └── refs
│       ├── heads
│       │   └── feature
│       ├── remotes
│       └── origin
├── objects
│   ├── e6
│   ├── info
│   └── pack
├── refs
│   ├── heads
│   │   └── feature
│   ├── remotes
│   └── origin
├── tags
├── .idea
│   └── inspectionProfiles
├── doc
└── individual
```

Рисунок 6 – Результаты работы индивидуального задания 2

Ответы на контрольные вопросы:

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

До Python 3.4 использовались модули `os` и `os.path` для работы с файловой системой.

2. Что регламентирует PEP 428?

PEP 428 регламентирует "Представление дерева каталогов в стандартной библиотеке Python".

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

Для создания путей средствами модуля `pathlib` используется метод `Path()` с указанием нужного пути.

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

Для получения пути дочернего элемента файловой системы с помощью модуля `pathlib` используется метод `resolve()` или оператор `/`.

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` используется атрибут `parent`.

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

Операции с файлами с помощью модуля `pathlib` выполняются путём создания объектов `Path`, которые можно использовать для навигации по файловой системе, проверки существования файлов/директорий, создания/удаления файлов и директорий и т.д.

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

Для выделения компонентов пути файловой системы с помощью модуля `pathlib` можно использовать различные атрибуты объектов `Path`, такие как: `name`, `suffix`, `parent` и т.д., чтобы получить имя файла, его расширение, родительский каталог и прочее.

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

Для перемещения файлов с помощью модуля `pathlib` можно использовать метод `rename()` или `replace()`, а для удаления – метод `unlink()`.

9. Как выполнить подсчет файлов в файловой системе?

Для подсчета файлов можно использовать рекурсивную функцию, которая пройдет по всем каталогам и файлам.

10. Как отобразить дерево каталогов файловой системы?

Для отображения дерева каталогов файловой системы можно использовать рекурсивную функцию, которая пройдет по всем каталогам и файлам, выводя их структуру.

11. Как создать уникальное имя файла?

Для создания уникального имени файла можно использовать функции модуля `tempfile` или генерировать уникальные имена на основе времени, случайных чисел или других уникальных идентификаторов.

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

Основное отличие в использовании модуля `pathlib` для различных ОС заключается в разделителе каталогов: для Windows используется обратный слеш `\`, а для Unix-подобных систем – `/`. Кроме того, есть различия в поддерживаемых атрибутах файловых систем и их названиях, но в большинстве случаев модуль `pathlib` абстрагируется от этих различий, обеспечивая удобный интерфейс для работы с путями вне зависимости от ОС.

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с файловой системой с помощью библиотеки «`pathlib`» языка программирования Python версии 3.x, способы работы с файловой системой с использованием данного модуля.