МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРОКАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Кафедра

инфокоммуникаций

Институт цифрового

развития

ОТЧЁТ

по лабораторной работе №2.19

Дисциплина: «Основы программной инженерии»

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

Выполнила:

студентка 2 курса

группы Пиж-б-о-21-1

Джолдошова Мээрим

Бекболотовна

Ставрополь 2023

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

1. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python. Зафиксируйте изменения в репозитории.

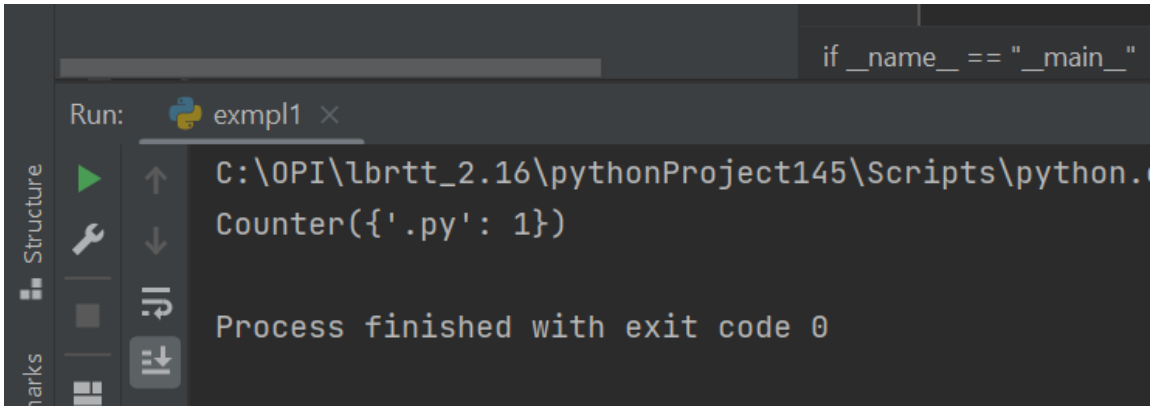

Рисунок 1 – Результат работы 1-го примера

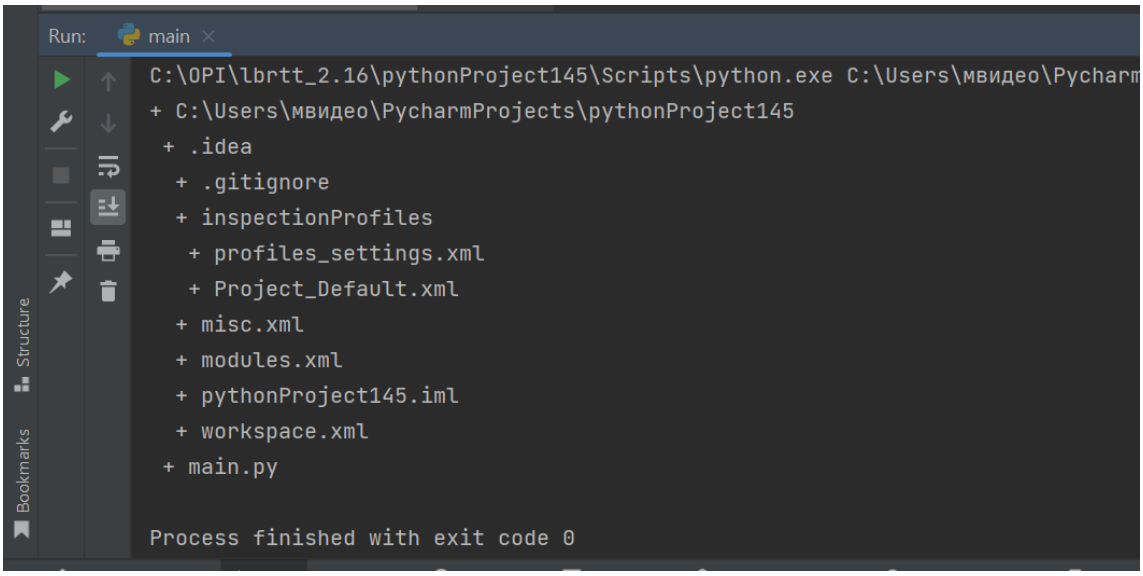

Рисунок 2 – Результат работы 2-го примера

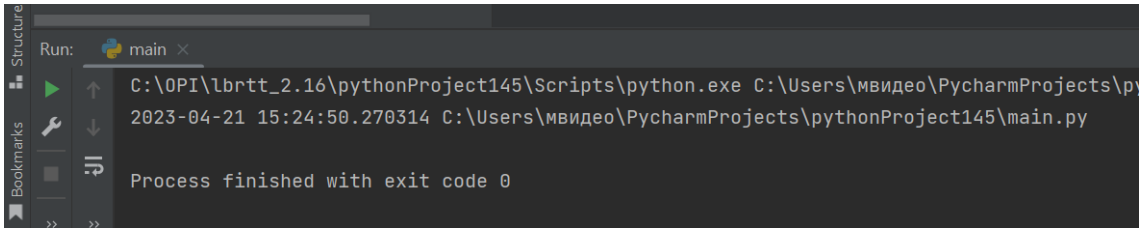

Рисунок 3 – Результат работы 3-го примера

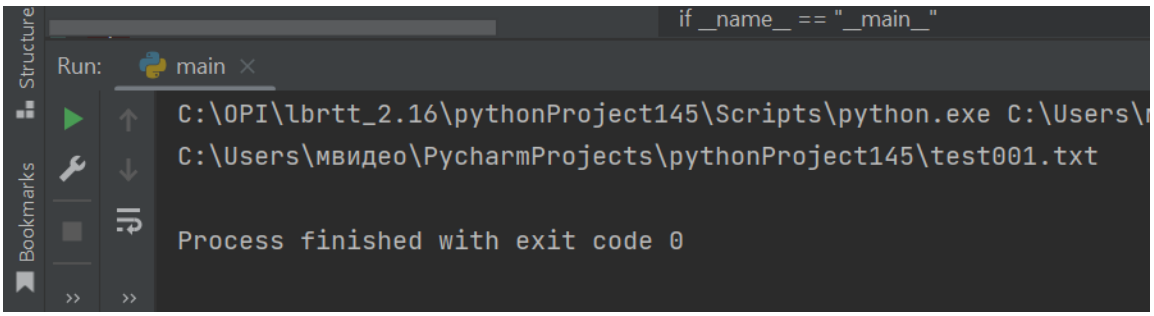

Рисунок 4 – Результат работы 4-го примера

Рисунок 5 – Результат работы 5-го примера

2. 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`.

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.

Задание 1

Код программы:

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

import json
import argparse
import os.path
import pathlib
```

```
def add_flight(flights, dst, nmb, tpe):
    flights.append(
        {
            "destination": dst,
            "number_flight": nmb,
            "type_plane": tpe
        }
    )
    return flights


def display_flights(flights):
    """
    displaying the given information
    """
    if flights:
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 18
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^18} |'.format(
                "№",
                "Destination",
                "NumberOfTheFlight",
                "TypeOfThePlane"
            )
        )
        print(line)

        for idx, flight in enumerate(flights, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>18} |'.format(
                    idx,
                    flight.get('destination', ''),
                    flight.get('number_flight', ''),
                    flight.get('type_plane', 0)
                )
            )
        print(line)
    else:
        print('list is empty')


def select_flights(flights, t):
    result = []
    for flight in flights:
        if t in str(flight.values()):
            result.append(flight)
    return result


def save_flights(file_name, flights):
    with open(file_name, "w", encoding="utf-8", errors="ignore") as fout:
        json.dump(flights, fout, ensure_ascii=False, indent=4)


def load_flights(file_name):
    with open(file_name, "r", encoding="utf-8", errors="ignore") 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("flights")
    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 flight"
    )
    add.add_argument(
        "-d",
        "--destination",
        action="store",
        required=True,
        help="Destination of the flight"
    )
    add.add_argument(
        "-n",
        "--number",
        action="store",
        type=int,
        required=True,
        help="Number of the flight"
    )
    add.add_argument(
        "-t",
        "--type",
        action="store",
        required=True,
        help="Type of the plane"
    )

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

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the flights"
    )
    select.add_argument(
```

```
        "-s",
        "--select",
        action="store",
        required=True,
        help="The required select"
    )

    args = parser.parse_args(command_line)
    dst = pathlib.Path.home() / args.filename
    is_dirty = False
    if dst.exists():
        flights = list(load_flights(dst))
    else:
        flights = []


    # Добавить работника.
    if args.command == "add":
        flights = add_flight(
            flights,
            args.destination,
            args.number,
            args.type
        )
        is_dirty = True

    # Отобразить всех работников.
    elif args.command == "display":
        display_flights(flights)

    elif args.command == "select":
        selected = select_flights(flights, args.select)
        display_flights(selected)

    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_flights(dst, flights)


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

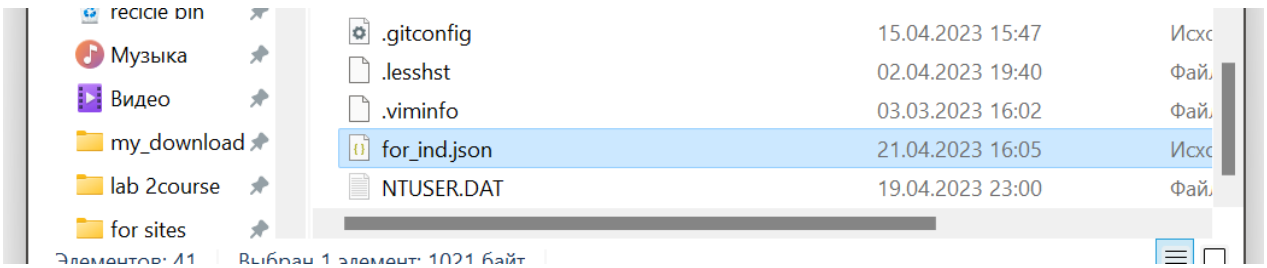

Рисунок 6 – Наличие файла в домашнем каталоге

```
C:\OPI\lbrtt_2.19\indiv>python indiv1.py display for_ind.json
+------+--------------------------------+----------------------+--------------------+
|  №   |          Destination           |  NumberOfTheFlight   |   TypeOfThePlane   |
+------+--------------------------------+----------------------+--------------------+
|    1 | japan                          | 2                    |         passangers |
|    2 | china                          | 7                    |         passangers |
|    3 | korea                          | 2                    |           official |
|    4 | canada                         | 3                    |           official |
|    5 | canada                         | 3                    |           official |
|    6 | america                        | 1                    |           official |
|    7 | france                         | 11                   |         passangers |
|    8 | Indonesia                      | 9                    |         passangers |
+------+--------------------------------+----------------------+--------------------+
```

Рисунок 7 – Результат выполнения индивидуального задания

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 программы:

```python
import os
import argparse

def build_tree(dir_path, level, show_files, show_dirs_only):
    if not os.path.isdir(dir_path):
        return

    # Prepare the indent string
    indent = ""
    if level > 0:
        indent = "|   " * (level - 1)
        indent += "|-- "

    # Print the current directory name
    print(indent + os.path.basename(dir_path) + "/")

    # Recursively print the contents of subdirectories
    if level >= 0:
        for item in sorted(os.listdir(dir_path)):
            item_path = os.path.join(dir_path, item)
            if os.path.isdir(item_path):
                build_tree(item_path, level + 1, show_files, show_dirs_only)
            elif show_files and not show_dirs_only:
                print(indent + "|   " * level + "|-- " + item)

# Set up command line arguments
parser = argparse.ArgumentParser(description='Print the directory tree')
parser.add_argument('dir_path', metavar='dir_path', type=str, nargs='?',
                    default='.', help='the directory path')
parser.add_argument('-l', '--level', type=int, default=-1,
                    help='the maximum recursion depth')
parser.add_argument('-f', '--show-files', action='store_true',
                    help='show files in addition to directories')
parser.add_argument('-d', '--show-dirs-only', action='store_true',
```

```
                    help='show only directories, no files')

# Parse the command line arguments
args = parser.parse_args()

# Build and print the directory tree
build_tree(args.dir_path, 0, args.show_files, args.show_dirs_only)
```

```
C:\OPI\lbrtt_2.19\indiv>python indivvv2.py -l 1 -f "C:\OPI\lbrtt_2.17"
lbrtt_2.17/
|-- .git/
|-- |   |-- COMMIT_EDITMSG
|-- |   |-- HEAD
|-- |   |-- ORIG_HEAD
|-- |   |-- config
|-- |   |-- description
|   |-- hooks/
|   |-- |   |   |-- applypatch-msg.sample
|   |-- |   |   |-- commit-msg.sample
|   |-- |   |   |-- fsmonitor-watchman.sample
|   |-- |   |   |-- post-update.sample
|   |-- |   |   |-- pre-applypatch.sample
|   |-- |   |   |-- pre-commit.sample
|   |-- |   |   |-- pre-merge-commit.sample
|   |-- |   |   |-- pre-push.sample
|   |-- |   |   |-- pre-rebase.sample
|   |-- |   |   |-- pre-receive.sample
|   |-- |   |   |-- prepare-commit-msg.sample
|   |-- |   |   |-- push-to-checkout.sample
|   |-- |   |   |-- update.sample
```

Рисунок 8 – Результат работы программы

Контрольные вопросы

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

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

path.rsplit('\\', 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?

Данный PEP предлагает включить в стандартную библиотеку модуль стороннего разработчика – pathlib. Включение предлагается под предварительной меткой, как описано в PEP 411. Поэтому изменения в API могут быть сделаны либо в рамках процесса PEP, либо после принятия в стандартную библиотеку (и до тех пор, пока предварительная метка не будет снята).

Цель этой библиотеки - предоставить простую иерархию классов для работы с путями файловой системы и обычными операциями, которые пользователи выполняют над ними.

3. Как осуществляется создание путей средствами модуля pathlib?

Все, что вам действительно нужно знать, это класс pathlib.Path . Есть несколько разных способов создания пути. Прежде всего, существуют classmethods наподобие .cwd() (текущий рабочий каталог) и .home() (домашний каталог вашего пользователя):

import pathlib pathlib.Path.cwd()

Вывод: PosixPath('/home/gahjelle/realpython/')

Путь также может быть явно создан из его строкового представления: pathlib.Path(r'C:\Users\gahjelle\realpython\file.txt')

Вывод: WindowsPath('C:/Users/gahjelle/realpython/file.txt') Объединение путей: с помощью «\» или .joinpath() pathlib.Path.home().joinpath('python',

'scripts', 'test.py') PosixPath('/home/gahjelle/python/scripts/test.py')

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

path = pathlib.Path('test.md') path.resolve()

PosixPath('/home/gahjelle/realpython/test.md')

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

path.parent

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

Чтение и запись файлов

Традиционно для чтения или записи файла в Python использовалась встроенная функция open() . Это все еще верно, поскольку функция open() может напрямую использовать объекты Path . Следующий пример находит все заголовки в файле Markdown и печатает их:

path = pathlib.Path.cwd() / 'test.md' with open(path, mode='r') as fid:

headers = [line.strip() for line in fid if line.startswith('#')]

print('\n'.join(headers))

Для простого чтения и записи файлов в библиотеке pathlib есть несколько удобных методов:

.read_text() : открыть путь в текстовом режиме и вернуть содержимое в виде строки.

.read_bytes() : открыть путь в двоичном/байтовом режиме и вернуть содержимое в виде строки байтов.

.write_text() : открыть путь и записать в него строковые данные.

.write_bytes() : открыть путь в двоичном/байтовом режиме и записать в него данные.

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

Различные части пути удобно доступны как свойства. Основные

примеры включают в себя:

.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?

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:

if not destination.exists(): source.replace(destination)

Тем не менее, это оставляет дверь открытой для возможного состояния гонки. Другой процесс может добавить файл по пути destination между выполнением оператора if и метода .replace() . Если это вызывает озабоченность, более безопасный способ - открыть путь назначения для создания exclusive и явно скопировать исходные данные:

with destination.open(mode='xb') as fid:

fid.write(source.read_bytes())

Приведенный выше код вызовет FileExistsError , если destination уже существует. Технически это копирует файл. Чтобы выполнить перемещение, просто удалите source после завершения копирования.

Когда вы переименовываете файлы, полезными методами могут быть

.with_name() и .with_suffix() . Они оба возвращают исходный путь, но с замененным именем или суффиксом соответственно.

path PosixPath('/home/gahjelle/realpython/test001.txt')

path.with_suffix('.py') PosixPath('/home/gahjelle/realpython/test001.py') path.replace(path.with_suffix('.py'))

Каталоги и файлы могут быть удалены с помощью .rmdir() и .unlink() соответственно.

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. В следующем примере комбинируется .iterdir() с классом collection.Counter для подсчета количества файлов каждого типа в текущем каталоге:

import collections

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

Более гибкие списки файлов могут быть созданы с помощью методов

.glob() и .rglob() (рекурсивный глоб). Например, pathlib.Path.cwd().glob('*.txt') возвращает все файлы с суффиксом .txt в текущем каталоге. Следующее только подсчитывает типы файлов, начинающиеся с p :

import collections

collections.Counter(p.suffix for p in pathlib.Path.cwd().glob('*.p*')) Counter({'.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?

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:

pathlib.WindowsPath('test.md')

NotImplementedError: cannot instantiate 'WindowsPath' on your system 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 .

path = pathlib.PureWindowsPath(r'C:\Users\gahjelle\realpython\file.txt') path.name

'file.txt' path.parent

PureWindowsPath('C:/Users/gahjelle/realpython') path.exists()

AttributeError: 'PureWindowsPath' object has no attribute 'exists'

Windows использует «\» , а Mac и Linux используют «/» в качестве разделителя. Это различие может привести к трудно обнаруживаемым ошибкам.