

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

Институт цифрового развития
Кафедра инфокоммуникаций

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

ОТЧЕТ
по лабораторной работе №19
дисциплины
«Основы программной инженерии»

Выполнила:

Кувшин Ирина Анатольевна

2 курс, группа ПИЖ-б-о-21-1,

011.03.04 «Программная инженерия»,

направленность (профиль) «Разработка

и сопровождение программного

обеспечения», очная форма обучения

(подпись)

Проверил:

(подпись)

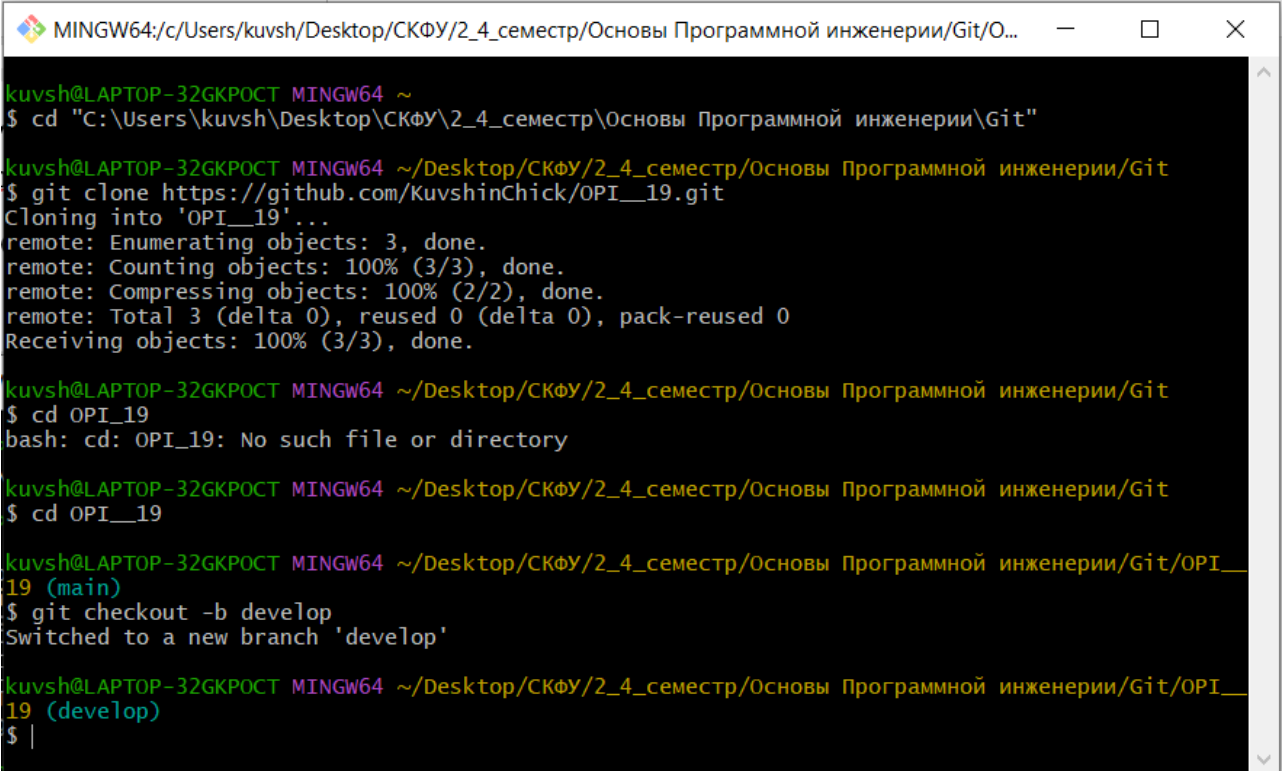
Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2023 г.

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 MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл `.gitignore` необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления `git-flow`.



```
MINGW64;C:/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/O...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~
$ cd "C:\Users\kuvsh\Desktop\СКФУ\2_4_семестр\Основы Программной инженерии\Git"

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ git clone https://github.com/KuvshinChick/OPI__19.git
Cloning into 'OPI__19'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.

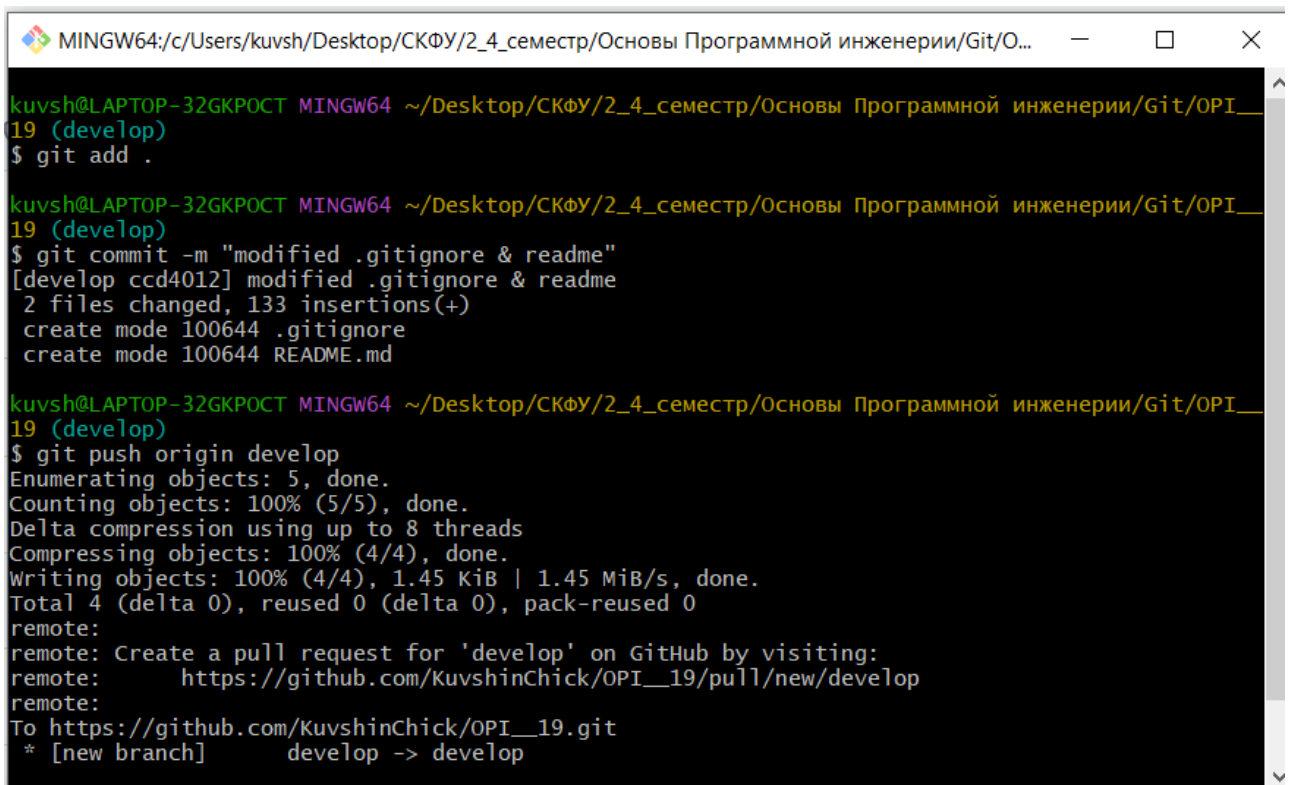
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd OPI__19
bash: cd: OPI__19: No such file or directory

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git
$ cd OPI__19

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__
19 (main)
$ git checkout -b develop
Switched to a new branch 'develop'

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__
19 (develop)
$ |
```

Рисунок 19.1 – Клонирование репозитория и создание ветки `develop`



```
MINGW64/c/Users/kuvsh/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/O...
kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__19 (develop)
$ git add .

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__19 (develop)
$ git commit -m "modified .gitignore & readme"
[develop ccd4012] modified .gitignore & readme
2 files changed, 133 insertions(+)
create mode 100644 .gitignore
create mode 100644 README.md

kuvsh@LAPTOP-32GKPOCT MINGW64 ~/Desktop/СКФУ/2_4_семестр/Основы Программной инженерии/Git/OPI__19 (develop)
$ git push origin develop
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 1.45 KiB | 1.45 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/KuvshinChick/OPI__19/pull/new/develop
remote:
To https://github.com/KuvshinChick/OPI__19.git
* [new branch]      develop -> develop
```

Рисунок 19.2 – Обновление .gitignore и readme

6. Создайте проект PyCharm в папке репозитория.

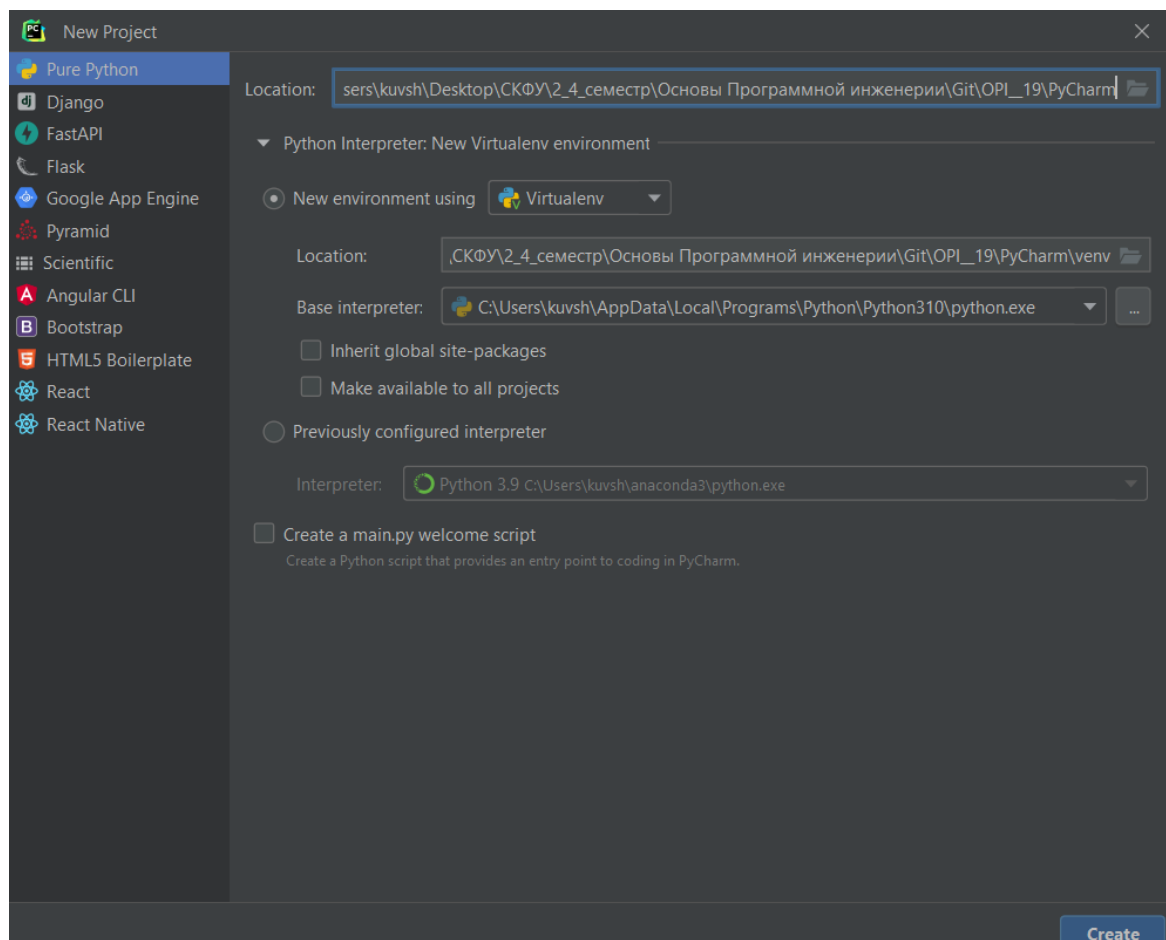


Рисунок 19.3 – Создание проекта и виртуального окружения

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

8. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных вводимых с клавиатуры.

ex_1: Подсчет файлов

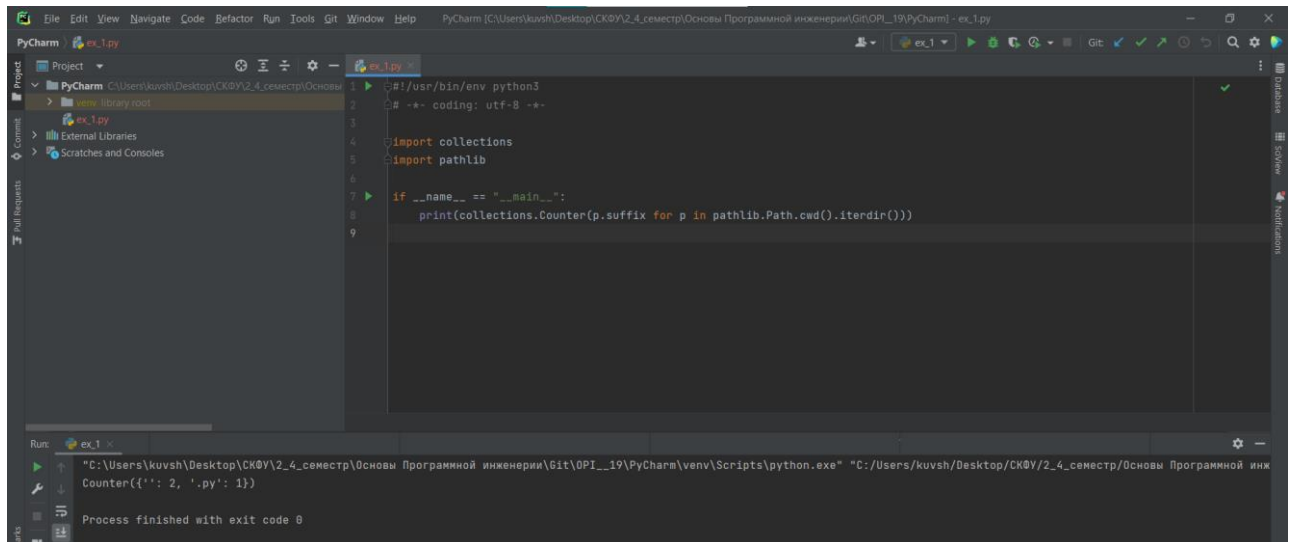


Рисунок 19.4 – Проработка и результат программы

ex_2: Показать дерево каталогов

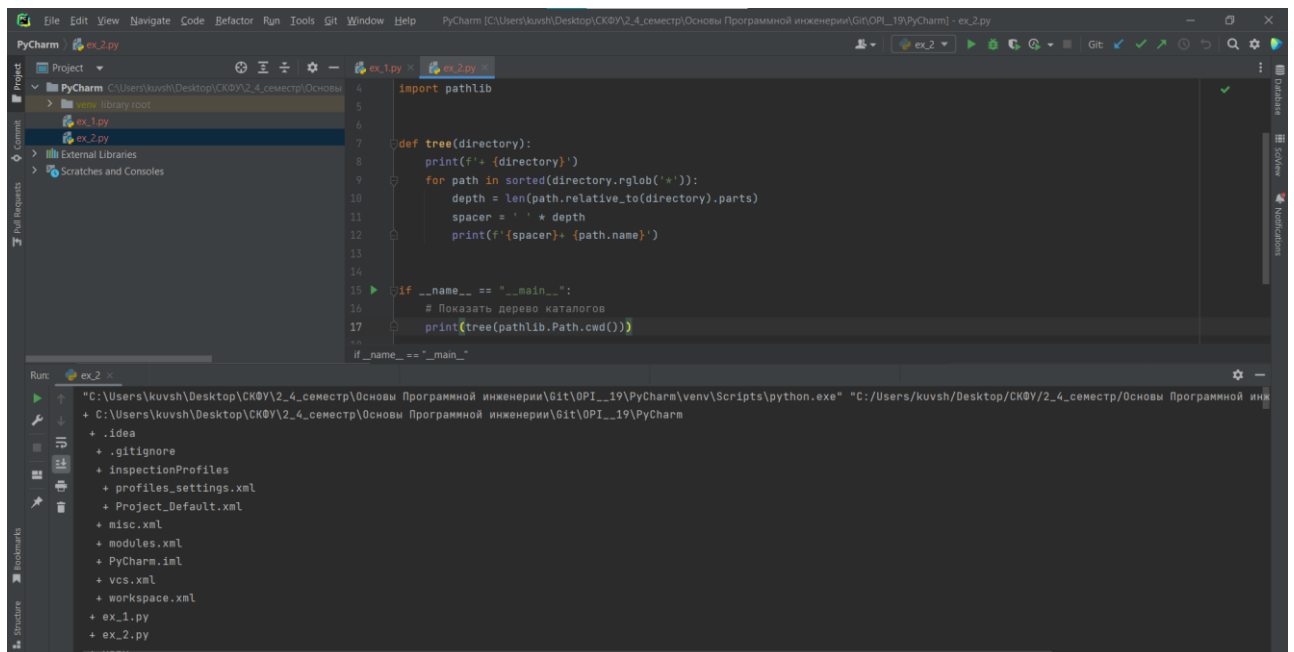


Рисунок 19.5 – Проработка и результат программы

ex_3: Найти последний измененный файл

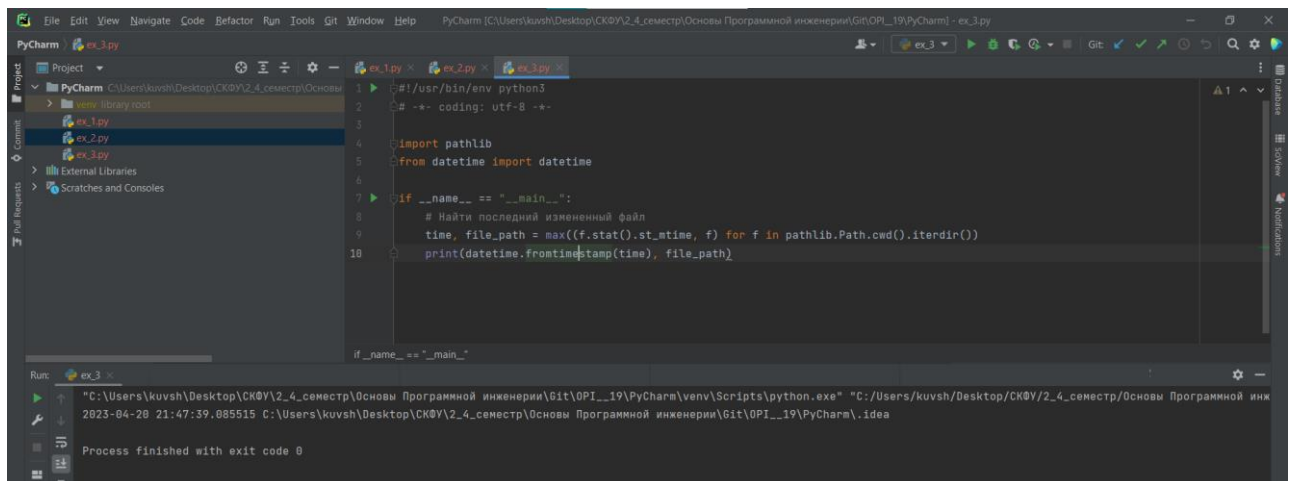


Рисунок 19.6 – Проработка и результат программы

ex_4: Создать уникальное имя файла

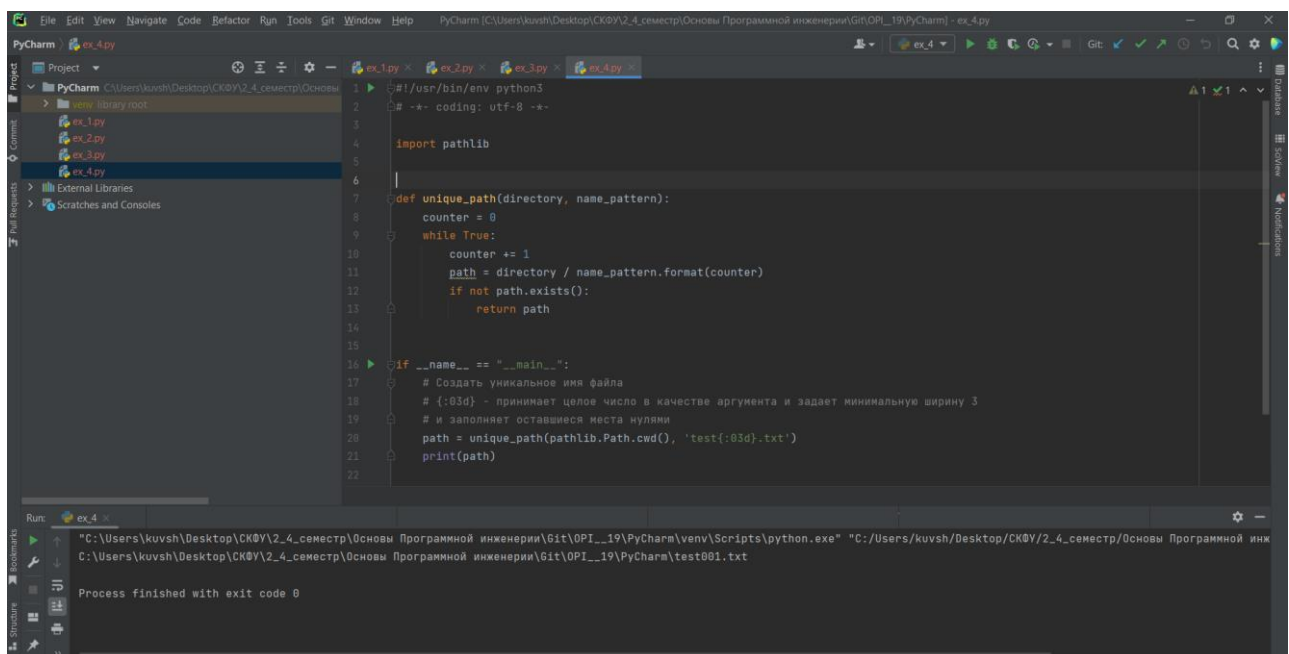


Рисунок 19.7 – Проработка и результат программы

Индивидуальное задание.

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`.

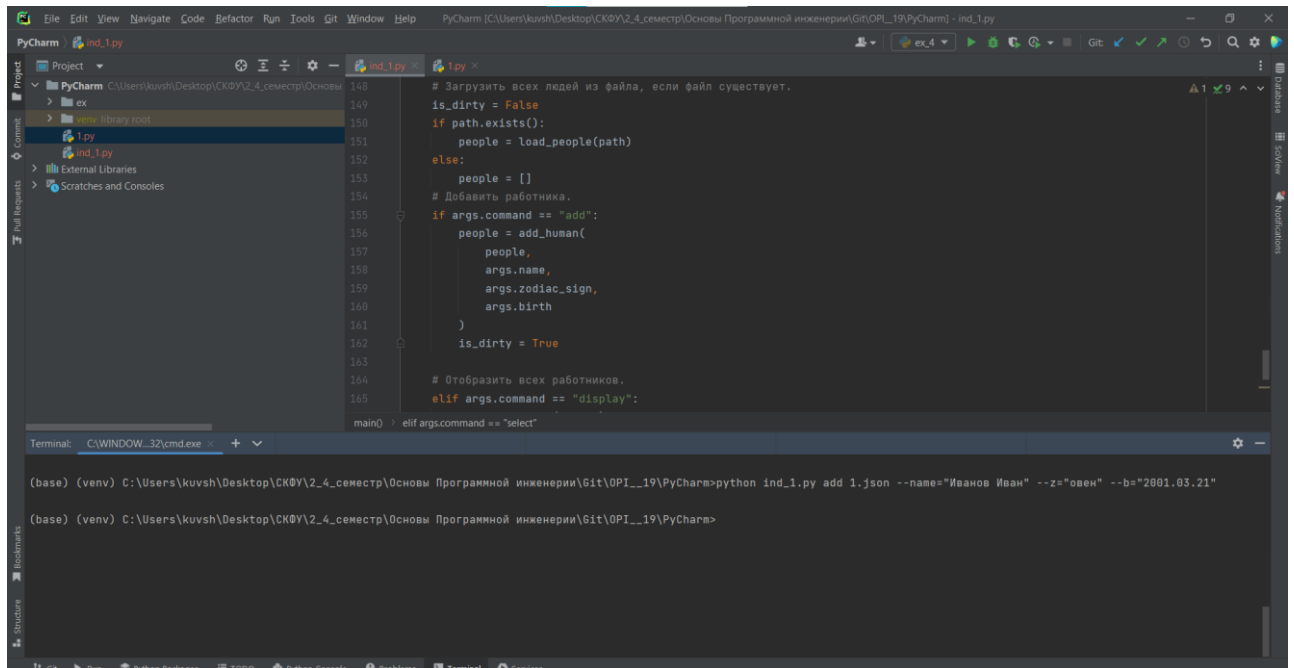


Рисунок 19.8 – Проработка индивидуального задания

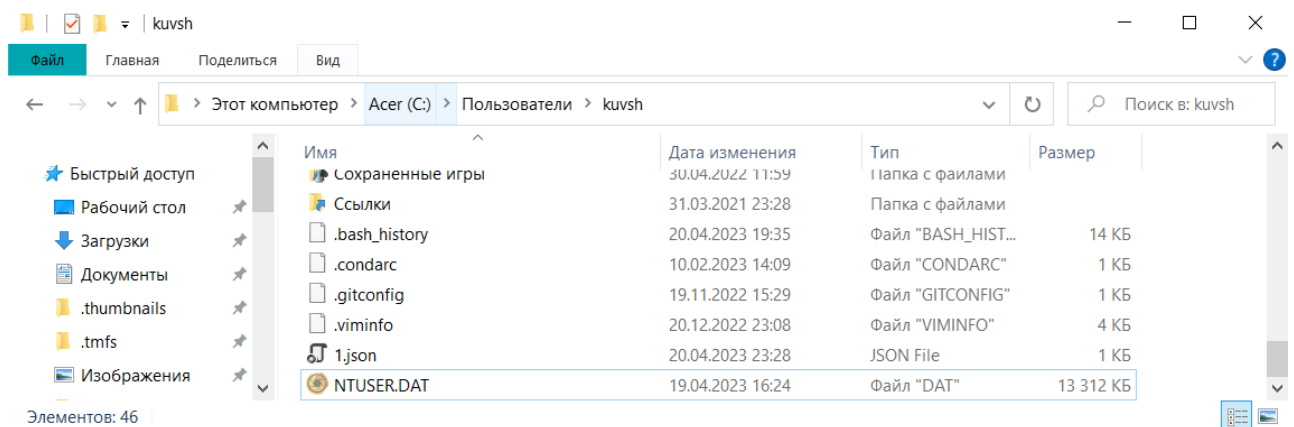


Рисунок 19.9 – Результат проработки программы

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.

9. Зафиксируйте сделанные изменения в репозитории.
10. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.
11. Зафиксируйте сделанные изменения в репозитории.
12. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксируйте изменения.
13. Выполните слияние ветки для разработки с веткой master/main.
14. Отправьте сделанные изменения на сервер GitHub.
15. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

Контрольные вопросы

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.rspllit('\\', 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`.

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):

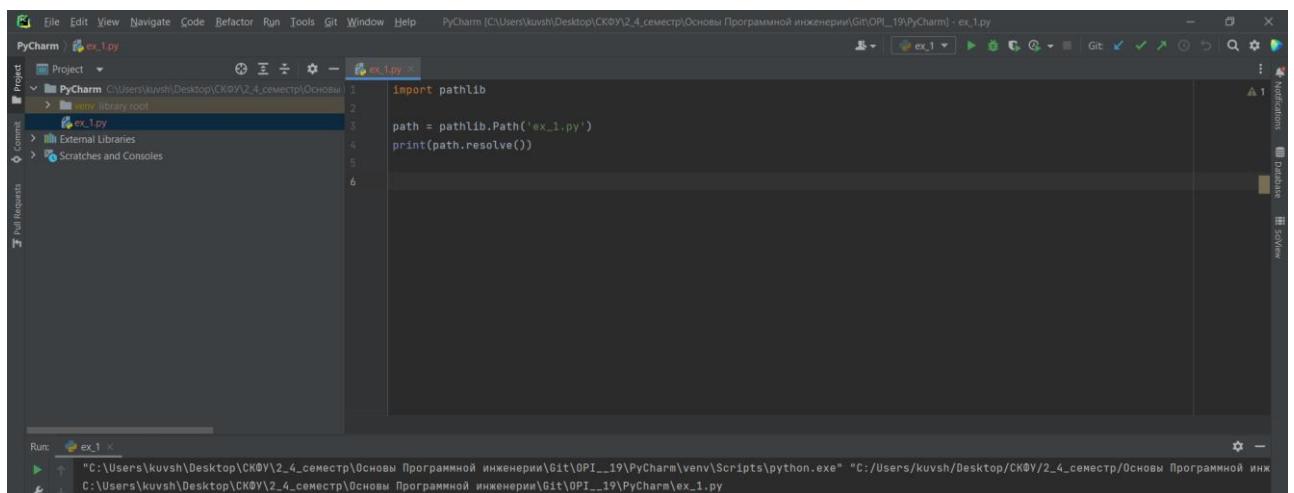
```
>>> import pathlib
>>> pathlib.Path.cwd()
PosixPath('/home/gahjelle/realpython/')
```


Третий способ построения пути - это соединение частей пути с помощью специального оператора `/`. Оператор прямой косой черты используется независимо от фактического разделителя пути на платформе:

```
>>> pathlib.Path.home()/'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()
```



5. Как получить путь к родительским элементам 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

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))
```

Эквивалентной альтернативой является вызов `.open()` для объекта `Path`:

```
with path.open(mode='r') as fid:
    ...
```

Фактически, `Path.open()` вызывает встроенную функцию `open()` за кулисами. Какой вариант вы используете, это в основном дело вкуса.

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

Вот эти свойства в действии:

```
>>> path
PosixPath('/home/gahjelle/realpython/test.md')
>>> path.name
'test.md'
```

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

Через `pathlib` вы также получаете доступ к базовым операциям на уровне файловой системы, таким как перемещение, обновление и даже удаление файлов. По большей части эти методы не выдают предупреждение и не ждут подтверждения, прежде чем информация или файлы будут потеряны. Будьте осторожны при использовании этих методов.

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)
```

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 `collections.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?

В следующем примере определяется функция `tree()`, которая будет печатать визуальное дерево, представляющее иерархию файлов, с корнем в данном каталоге. Здесь мы также хотим перечислить подкаталоги, поэтому мы используем метод `.rglob()`:

```
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?

Последний пример покажет, как создать уникальное нумерованное имя файла на основе шаблона. 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')
```

Если каталог уже содержит файлы `test001.txt` и `test002.txt`, приведенный выше код установит для `path` значение `test003.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
```