

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 №3
дисциплины «Анализ данных»

Выполнил:
Гайчук Дарья Дмитриевна
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 Р.А.-доцент кафедры
инфокоммуникаций

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: работа с переменными окружениями в python3

Цель работы: приобретение навыков по работе с переменными окружениями с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

1. Создала новый репозиторий и клонировала его на свой компьютер.

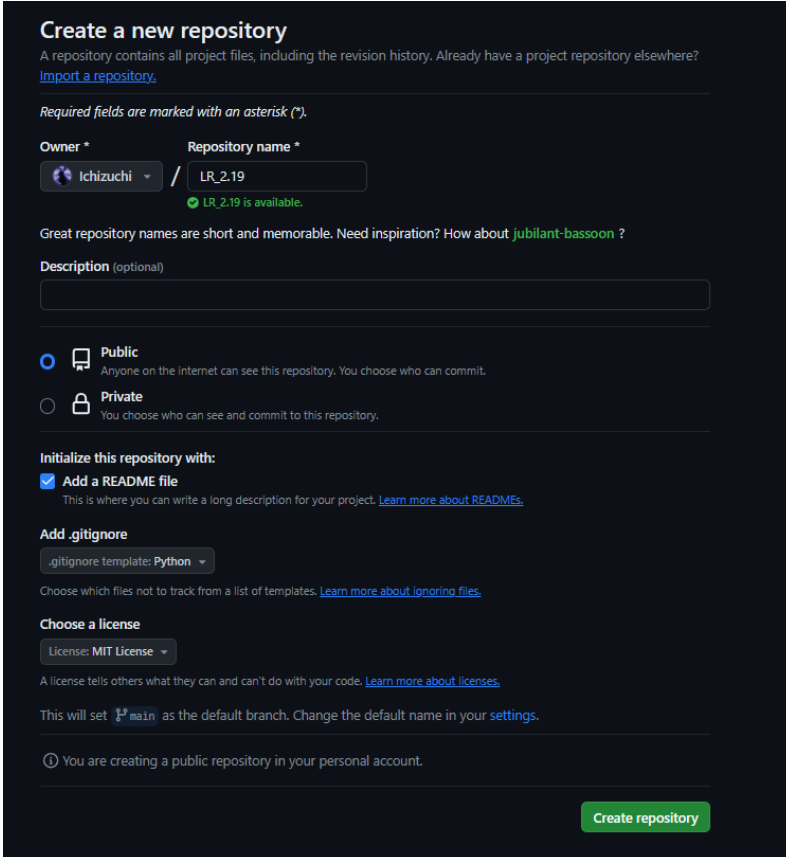
The image shows the GitHub 'Create a new repository' page. It includes fields for 'Owner' (set to 'Ichizuchi') and 'Repository name' (set to 'LR_2.19', with a note that it is available). There is an optional 'Description' field. Under 'Initialize this repository with:', the 'Add a README file' option is selected. The 'Add .gitignore' section shows a dropdown for 'Python'. The 'Choose a license' section shows 'MIT License' selected. A green 'Create repository' button is at the bottom right.

Рисунок 1. Создан новый репозиторий

2. Клонировала репозиторий на свой компьютер. В ходе данной лабораторной работы работала с моделью ветвления git-flow.

```
● @Ichizuchi →/workspaces/LR_2.19 (main) $ git clone https://github.com/Ichizuchi/LR_2.19
Cloning into 'LR_2.19'...
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.
```

Рисунок 2. Клонирование и модель ветвления git-flow

3. Создала виртуальное окружение Anaconda с именем репозитория.

```

@Ichizuchi →/workspaces/LR_2.19 (main) $ conda create -n myenv python=3.10
Channels:
  - defaults
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

environment location: /opt/conda/envs/myenv

added / updated specs:
  - python=3.10

The following packages will be downloaded:


```

package	build	size
_libgcc_mutex-0.1	main	3 KB
_openmp_mutex-5.1	1_gnu	21 KB
bzip2-1.0.8	h5eee18b_5	262 KB
ca-certificates-2024.3.11	h06a4308_0	127 KB
ld_impl_linux-64-2.38	h1181459_1	654 KB
libffi-3.4.4	h6a678d5_0	142 KB
libgcc-ng-11.2.0	h1234567_1	5.3 MB
libgomp-11.2.0	h1234567_1	474 KB
libstdcxx-ng-11.2.0	h1234567_1	4.7 MB
libuuid-1.41.5	h5eee18b_0	27 KB

Рисунок 3. Создание виртуального окружения

4. Выполнение индивидуального задания

```

main.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  import argparse
4  import json
5  import os
6  from pathlib import Path
7
8  # Имя файла данных по умолчанию
9  default_data_file = "flights.json"
10
11 def input_flights():
12     flights = []
13     n = int(input("Введите количество рейсов: "))
14
15     for i in range(n):
16         flight = {}
17         flight["город назначения"] = input("Введите город назначения: ")
18         flight["номер рейса"] = input("Введите номер рейса: ")
19         flight["тип самолета"] = input("Введите тип самолета: ")
20         flights.append(flight)
21
22     flights.sort(key=lambda x: x["город назначения"])
23     return flights
24
25 def print_flights_with_plane_type(flights):
26     plane_type = input("Введите тип самолета: ")

```

PROBLEMY Выходные данные КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ ПОРТЫ КОММЕНТАРИИ

```

/home/codespace/.python/current/bin/python3 /workspaces/LR_2.19/main.py
@Ichizuchi →/workspaces/LR_2.19 (main) $ /home/codespace/.python/current/bin/python3 /workspaces/LR_2.19/main.py
@Ichizuchi →/workspaces/LR_2.19 (main) $ python3 main.py --input
Введите количество рейсов: 3
Введите город назначения: Москва
Введите номер рейса: 635521
Введите тип самолета: Почтовый
Введите город назначения: Казань
Введите номер рейса: 736251
Введите город назначения: Специального назначения
Введите город назначения: Ставрополь
Введите номер рейса: 627381
Введите тип самолета: Аэробус
@Ichizuchi →/workspaces/LR_2.19 (main) $ python3 main.py --print_plane_type
Введите тип самолета: Почтовый
Город назначения: Москва, Номер рейса: 635521
@Ichizuchi →/workspaces/LR_2.19 (main) $

```

Рисунок 4. Выполнение индивидуального задания

5. Сформировала файлы environment.yml и requirements.txt

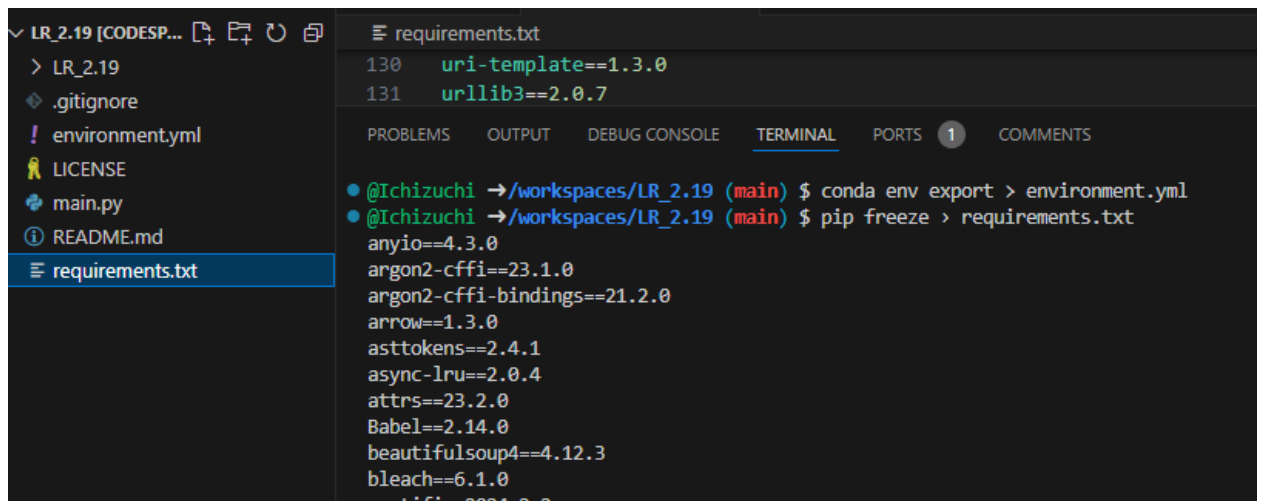


Рисунок 5. Файлы environment.yml и requirements.txt

6. Отправила на удаленный сервер.

Ответы на контрольные вопросы

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

До Python 3.4 основными средствами для работы с файловой системой были модули:

- 1) `os` - для взаимодействия с операционной системой.
- 2) `os.path` - для операций с путями и файлами.

Эти модули предоставляли функции для работы с директориями, файлами, проверки существования файлов и директорий, а также другие операции с файловой системой.

2. Что регламентирует PEP 428?

PEP 428 регламентирует введение стандартного модуля pathlib в Python 3.4. Этот PEP (Python Enhancement Proposal) был предложен Гвидо ван Россумом и осуществлен Армина Ронахера.

Основные цели PEP 428:

Предоставление более удобного и выразительного интерфейса для работы с путями и файловой системой.

Замена более старых и менее удобных способов, таких как использование строковых операций или модуля `os.path`.

Улучшение переносимости кода между различными операционными системами.

`pathlib` вводит классы `Path`, представляющие пути к файлам и директориям, и предоставляет методы и операторы для выполнения различных операций с ними. Внедрение этого модуля значительно улучшило удобство и читаемость кода, связанного с работой с файловой системой.

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

Для создания путей с помощью модуля `pathlib` в Python, вы используете класс `Path`. Вот несколько примеров создания путей:

```
# Создание объекта Path для файла file_path = Path('/путь/к/файлу.txt')  
  
# Создание объекта Path для директории dir_path =  
Path('/путь/к/директории')  
  
# Склеивание путей new_path = dir_path / 'файл.txt'
```

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

Для получения пути дочернего элемента файловой системы с помощью модуля `pathlib` вы можете использовать оператор `/` или метод `.joinpath()`. Вот примеры обоих способов:

Оператор `/`: pythonCopy code

```
from pathlib import Path # Создание объекта Path для директории  
dir_path =
```

```
Path('/путь/к/родительской_директории') # Получение пути дочернего  
элемента child_path = dir_path / 'дочерняя_директория' / 'файл.txt' Метод  
.joinpath(): pythonCopy code
```

```
from pathlib import Path # Создание объекта Path для директории
dir_path =
```

```
Path('/путь/к/родительской_директории') # Получение пути дочернего
элемента child_path = dir_path.joinpath('дочерняя_директория', 'файл.txt')
```

Оба способа создают объект Path, представляющий путь к дочернему элементу файловой системы.

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`. Вот пример:

pythonCopy code

```
from pathlib import Path # Создание объекта Path для файла или
директории file_path = Path('/путь/к/файлу_или_директории') # Получение
пути к родительской директории parent_path = file_path.parent
```

В данном примере, `parent_path` станет объектом Path, представляющим родительскую директорию файла или директории, указанных в `file_path`.

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

Модуль `pathlib` предоставляет удобные методы для выполнения операций с файлами. Вот примеры некоторых операций:

Чтение содержимого файла:

```
from pathlib import Path file_path = Path('/путь/к/файлу.txt') # Чтение
содержимого файла content = file_path.read_text()
```

Запись в файл:

```
from pathlib import Path file_path = Path('/путь/к/файлу.txt') # Запись в
файл text_to_write = 'Пример текста для записи в файл.'
file_path.write_text(text_to_write)
```

Добавление текста в конец файла:

```
from pathlib import Path file_path = Path('/путь/к/файлу.txt') #  
Добавление текста в конец файла text_to_append = 'Этот текст будет добавлен  
в конец файла.' file_path.write_text(file_path.read_text() + text_to_append)
```

Чтение и запись бинарных данных:

```
from pathlib import Path file_path = Path('/путь/к/файлу.bin') # Чтение  
бинарных данных из файла binary_content = file_path.read_bytes() # Запись  
бинарных данных в файл new_binary_content = b'Новые бинарные данные.'  
file_path.write_bytes(new_binary_content)
```

Открытие файла в контекстном менеджере:

```
from pathlib import Path file_path = Path('/путь/к/файлу.txt') # Открытие  
файла в контекстном менеджере (автоматическое закрытие файла) with  
file_path.open() as file: content = file.read()
```

Эти методы делают работу с файлами более удобной и читаемой в сравнении с использованием старых методов из модуля `open` и `os`.

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

С помощью модуля `pathlib` можно легко выделять компоненты пути файловой системы. Вот примеры выделения различных компонентов:

Имя файла:

```
from pathlib import Path file_path = Path('/путь/к/директории/файл.txt') #  
Получение имени файла file_name = file_path.name print(f'Имя файла:  
{file_name}')
```

Имя директории:

```
from pathlib import Path file_path = Path('/путь/к/директории/файл.txt') #  
Получение имени директории dir_name = file_path.parent.name print(f'Имя
```

директории: {dir_name}")

Расширение файла:

```
from pathlib import Path file_path = Path('/путь/к/директории/файл.txt') #
```

```
Получение расширения файла file_extension = file_path.suffix  
print(f'Расширение файла: {file_extension}')
```

Без расширения (базовое имя):

```
from pathlib import Path file_path = Path('/путь/к/директории/файл.txt') #
```

```
Получение базового имени файла без расширения base_name =  
file_path.stem print(f'Базовое имя файла: {base_name}')
```

Эти методы предоставляют удобные способы получения различных компонентов пути, что делает код более читаемым и легко поддерживаемым.

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

Перемещение файла:

```
from pathlib import Path # Исходный путь файла source_path =
```

```
Path('/путь/к/исходному_файлу.txt') # Путь для перемещения файла  
destination_path = Path('/путь/к/целевой_директории/новое_имя_файла.txt') #
```

```
Перемещение файла source_path.rename(destination_path)
```

```
Удаление файла: from pathlib import Path # Путь к файлу для удаления  
file_path = Path('/путь/к/удаляемому_файлу.txt') # Удаление файла  
file_path.unlink()
```

Эти примеры показывают, как с использованием модуля `pathlib` можно легко перемещать и удалять файлы. Важно отметить, что при перемещении файла метод `rename` также может использоваться для переименования файла, если новое имя указано в целевом пути.

9. Как выполнить подсчет файлов в файловой системе?

Для выполнения подсчета файлов в файловой системе с помощью модуля `pathlib` вы можете использовать методы `rglob` (рекурсивный поиск) или `glob`. Вот пример подсчета файлов в текущей директории и ее поддиректориях:

```
from pathlib import Path # Путь к директории, для которой мы хотим
подсчитать файлы directory_path = Path('/путь/к/директории') # Рекурсивный
подсчет файлов file_count = sum(1 for _ in directory_path.rglob('*') if _.is_file())

print(f"Общее количество файлов в директории: {file_count}")
```

В этом примере `rglob('*')` рекурсивно ищет все файлы в текущей директории и ее поддиректориях, а `is_file()` проверяет, является ли каждый найденный путь файлом.

Если вы хотите подсчитать только файлы в текущей директории (без рекурсии), используйте `glob`:

```
from pathlib import Path # Путь к текущей директории current_directory
= Path() # Подсчет файлов в текущей директории file_count = sum(1 for _ in
current_directory.glob('*') if _.is_file()) print(f"Общее количество файлов в
текущей директории: {file_count}")
```

Оба эти примера помогут вам выполнить подсчет файлов в файловой системе с использованием модуля `pathlib`.

11. Как отобразить дерево каталогов файловой системы?

Для отображения дерева каталогов файловой системы с помощью модуля `'pathlib'` можно использовать рекурсивную функцию. Вот пример, который покажет, как это сделать:

```
python from pathlib import Path

def display_directory_tree(directory_path, indent=""):

    current_dir = Path(directory_path)

    print(f"{indent}+-- {current_dir.name}")
```

```

        # Рекурсивный обход поддиректорий    for item in current_dir.iterdir():
if item.is_dir():

    display_directory_tree(item, indent + '  ')

    else:

        print(f"{indent} |-- {item.name}")

# Путь к директории, для которой мы хотим отобразить дерево
root_directory = Path('/путь/к/директории')

# Вызываем функцию для отображения дерева
display_directory_tree(root_directory)

```

Этот код создает функцию `display_directory_tree`, которая рекурсивно обходит директории, начиная с указанной. Для каждого элемента она выводит его имя, а для поддиректорий вызывает сама себя.

Замените `'/путь/к/директории'` на путь к той директории, для которой вы хотите отобразить дерево.

12. Как создать уникальное имя файла?

Для создания уникального имени файла вам часто приходится добавлять к основному имени какие-то уникальные метки, такие как текущее время, случайное число или другие параметры. Модуль `pathlib` предоставляет удобные средства для создания уникальных имен файлов. Вот несколько примеров:

Используя текущее время:

```

from pathlib import Path from datetime import datetime # Определение
основного имени файла base_name = "file" # Получение текущей даты и
времени в строковом формате timestamp =
datetime.now().strftime("%Y%m%d%H%M%S")

# Создание уникального имени файла unique_name =
f"{base_name}_{timestamp}.txt" # Путь к файлу file_path =

```

```
Path('/путь/к/директории') / unique_name
```

Используя модуль uuid (универсальный уникальный идентификатор):

```
from pathlib import Path import uuid # Определение основного имени
файла base_name = "file" # Генерация уникального идентификатора unique_id
= str(uuid.uuid4()) # Создание уникального имени файла unique_name =
f"{base_name}_{unique_id}.txt" # Путь к файлу file_path =
```

```
Path('/путь/к/директории') / unique_name
```

Выбор метода зависит от ваших конкретных требований и предпочтений. Оба этих примера создадут уникальные имена файлов, которые могут быть использованы для создания файлов в файловой системе.

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

Модуль pathlib в Python создан с целью обеспечения переносимости кода между различными операционными системами. В основном, отличия в использовании pathlib для различных ОС сводятся к различиям в символах разделителей пути (/ или \), которые используются в путях файловой системы.

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