Практика 4. Виртуальное окружение Python

В Python у нас есть возможность создавать виртуальные окружения, например при помощи инструмента `virtualenv` (https://virtualenv.pypa.io/)
Начиная с версии Python 3.3 подмножество `virtualenv` интегрировано в стандартную библиотеку `venv`, что помимо прочего позволило ей отрабатывать до загрузки системных модулей.
Виртуальное окружение представляет собой директорию, похожую на системное окружение.

- В директории `Scripts` (`bin` для Liunx) расположена копия интерпретатора `python.exe` и копия исполняемого файла `pip.exe` (точка входа)
- В директории `Lib\site-packages` хранятся библиотеки установленные в окружении. Только что созданное окружение содержит пакеты `pip` и `setuptools`

При разработке в виртуальном окружении следует запускать исполняемые файлы из директории `Scripts`, а не системные.
Интерпретатор Python будет искать пакеты по относительному пути `..\Lib\`, что позволяет создать изолированную среду.

Установка для Ubuntu:

```
sudo apt install python3-venv
```

Чтобы проверить наличие модуля venv можно выполнить команду

```
python -m venv --help
```

Создадим виртуальное окружение и установим модуль requests:

```
python -m venv venv
.\venv\Scripts\pip.exe install requests
```

Теперь у нас появились новые модули в директории `Lib\site-packages`, кроме самого `requests` `pip` загрузил зависимости, такие как `urllib3`

Чтобы удобно работать в окружении мы можем его активировать, для этого есть сценарий оболочки `Scripts\activate` (`bin/activate` для Linux)

```
.\venv\Scripts\activate
```

Теперь мы можем не указывать пусть до исполняемых файлов виртуального кружения

```
pip install fastapi
```

Данные библиотеки были выбраны не случайно и будут использоваться в следующей практической работе.

Установив несколько библиотек мы загрузили последние доступные версии и будем разрабатывать проект на них, однако со временем версии обновятся и что-то может перестать работать так, как раньше.
Для избежания данной проблемы мы можем заморозить версии библиотек, для этого используется команда `pip freeze`

```
> pip freeze
annotated-types==0.5.0
anyio==3.7.1
certifi==2023.7.22
charset-normalizer==3.2.0
fastapi==0.103.1
idna==3.4
numpy==1.26.0
pandas==2.1.1
pydantic==2.3.0
pydantic_core==2.6.3
python-dateutil==2.8.2
pytz==2023.3.post1
requests==2.31.0
six==1.16.0
sniffio==1.3.0
starlette==0.27.0
typing_extensions==4.8.0
tzdata==2023.3
urllib3==2.0.5
```

Принято записывать используемые для разработки версии в файл requirements.txt в корне проекта.

```
pip freeze > requirements.txt
```

При создании репозитория, например на GitHub, мы не загружаем самое окружение, а только файл requirements.txt, библиотеки из которого при развёртывании можно установить одной командой

```
pip install -r requirements.txt
```