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 №2.14
дисциплины «Основы кроссплатформенного программирования»
Вариант___

Выполнил:
Ермолович Иван Денисович
2 курс, группа ИТС-б-0-22-1,
11.03.02 «Инфокоммуникационные технологии и системы связи»,
направленность (профиль) «Инфокоммуникационные системы и сети», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А., доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой_____Дата защиты_____

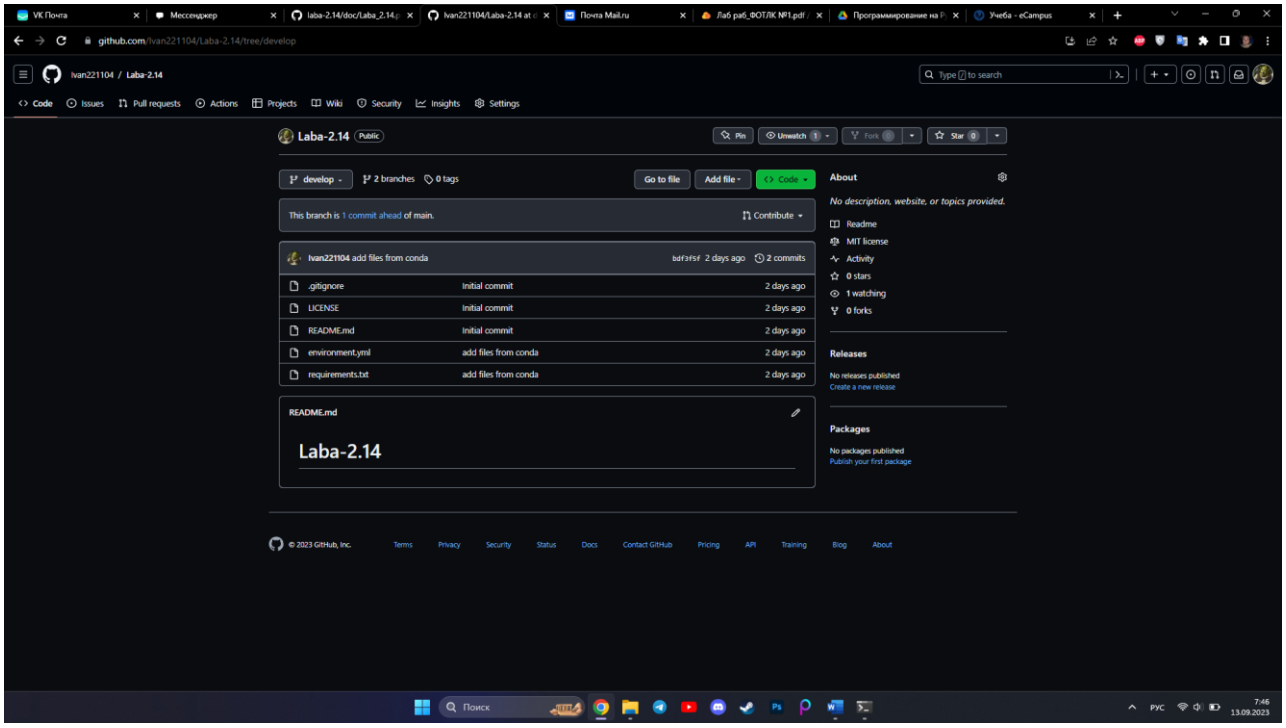
Ставрополь, 2023 г.

Тема: Виртуальные окружения

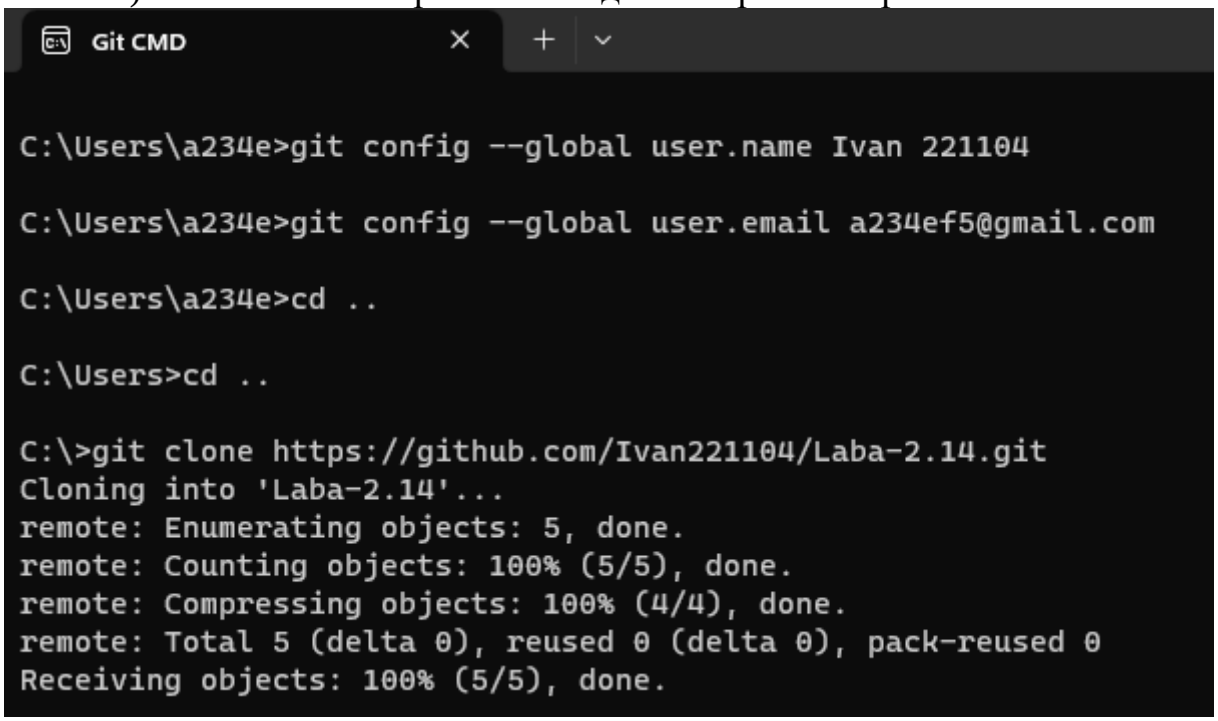
Цель: приобретение навыков по работе с менеджером пакетов pip и виртуальными окружениями с помощью языка программирования Python версии 3.x.

Ход работы

1) Создал общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



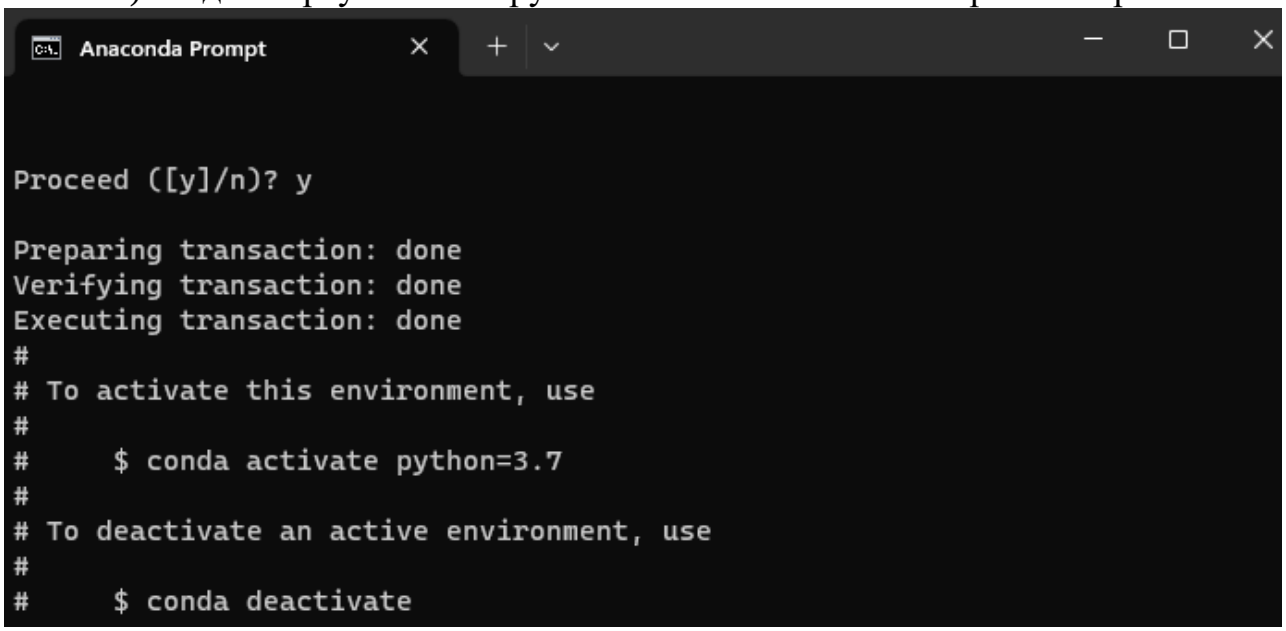
2) Выполнил клонирование созданного репозитория.



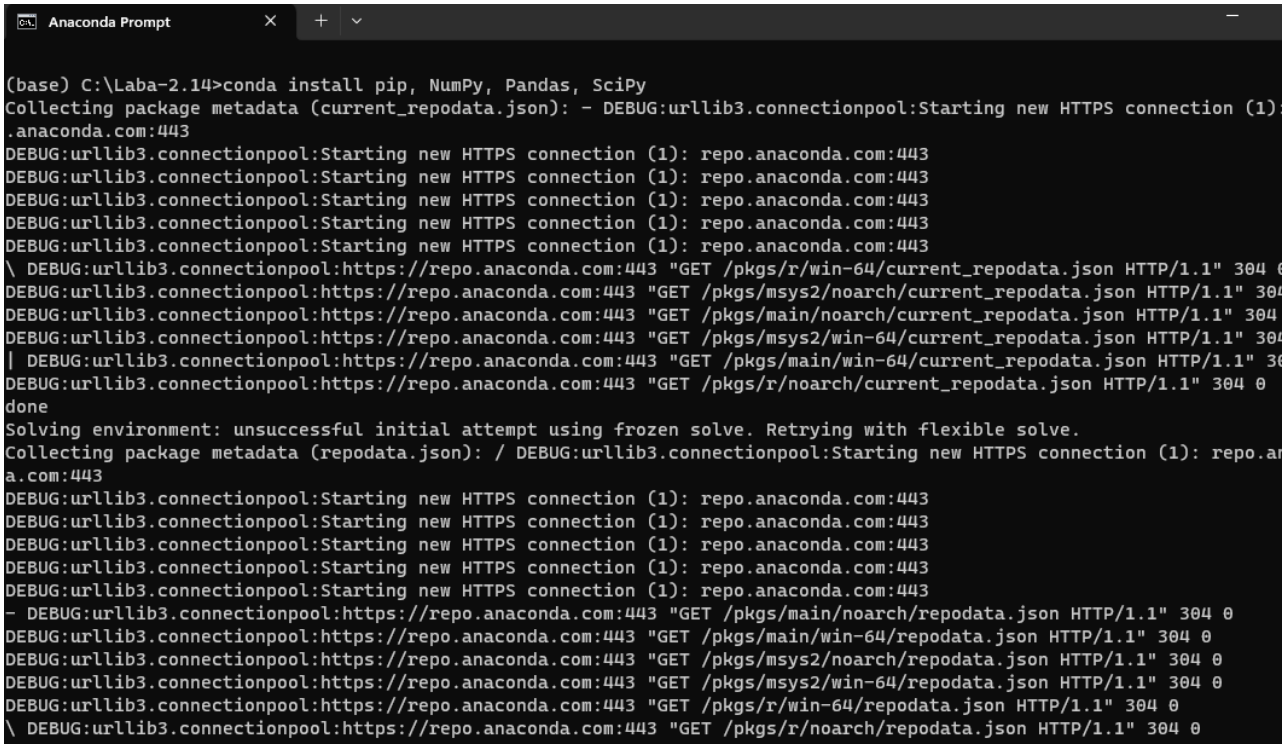
3)Сделал модель ветвления git flow

```
C:\Laba-2.14>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/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? []
Hooks and filters directory? [C:/Laba-2.14/.git/hooks]
```

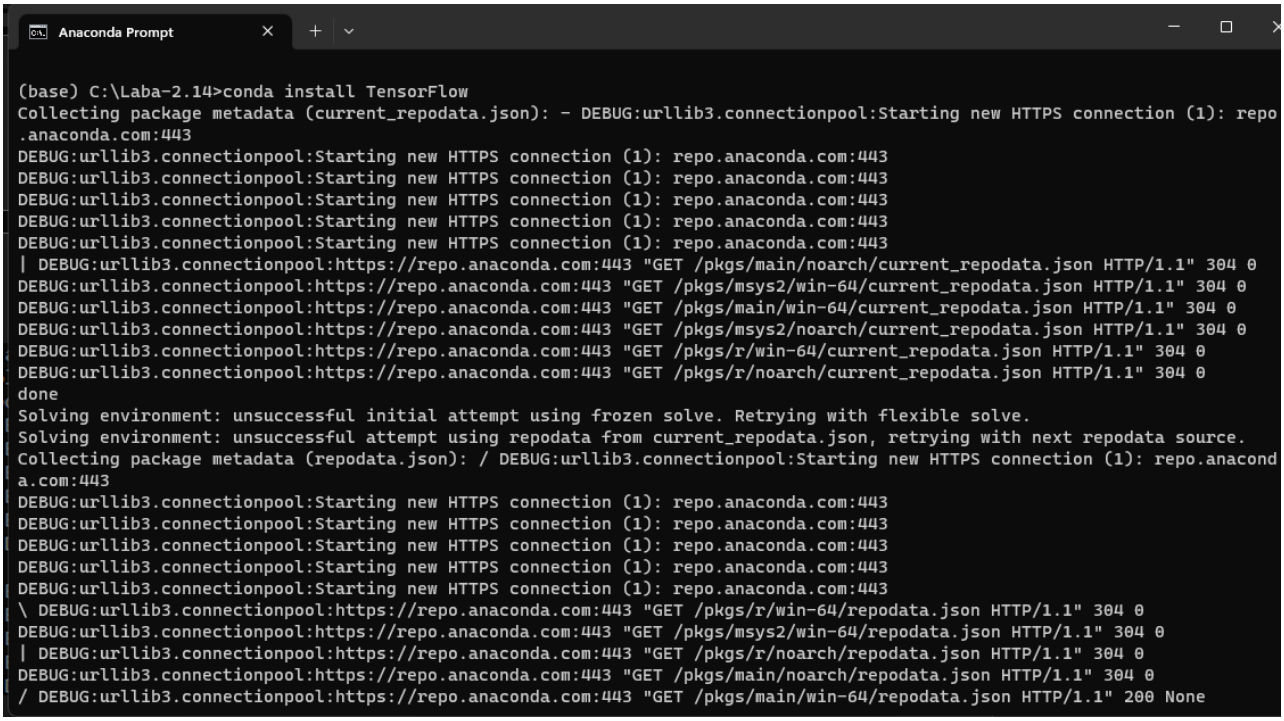
4)Создал виртуальное окружение Anaconda с именем репозитория.



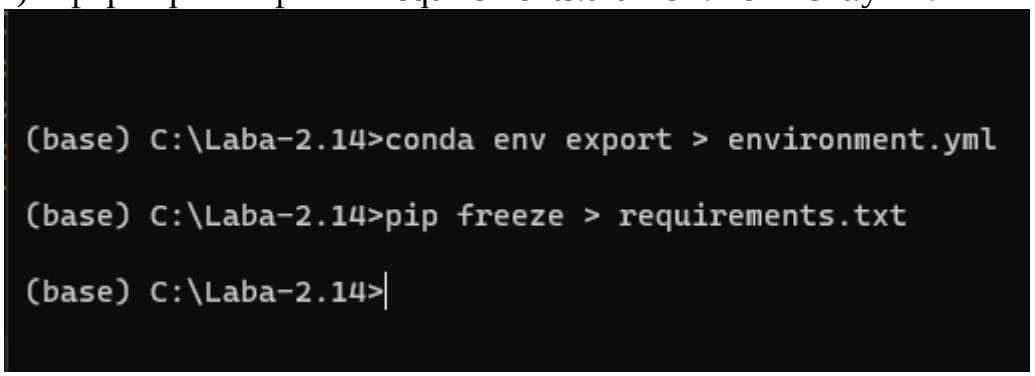
5) Установил в виртуальное окружение следующие пакеты: pip, NumPy, Pandas, SciPy.



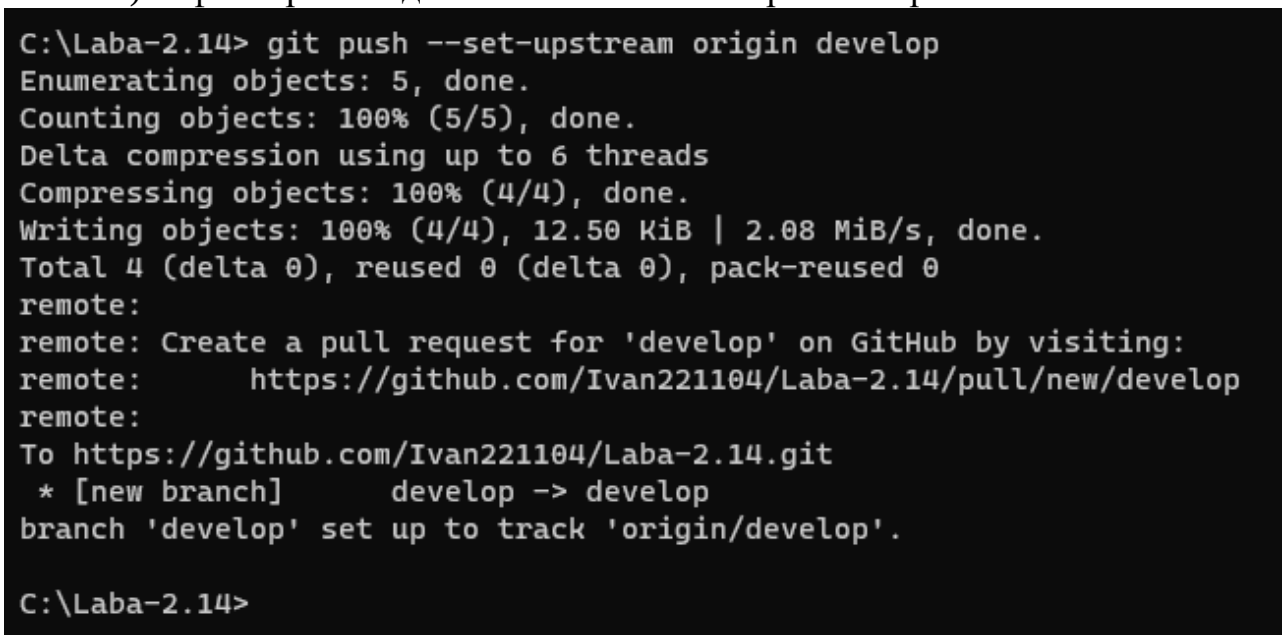
6) Установил пакет TensorFlow с помощью менеджера пакетов pip.



7) Сформировал файлы requirements.txt и environment.yml .



8) Зафиксировал сделанные изменения в репозитории.



Ответы на контрольные вопросы

1. Каким способом можно установить пакет Python, не входящий в стандартную библиотеку?

В Python есть несколько способов установить пакеты, не входящие в стандартную библиотеку. Наиболее распространенными способами являются использование менеджера пакетов pip и ручная установка из исходных кодов.

1. Установка с помощью pip:

- Убедитесь, что у вас установлен pip. Если нет, установите его следующей командой в командной строке: `python -m ensurepip --upgrade` (для Python 2: `python -m pip install --upgrade pip`).

- Откройте командную строку (терминал) и выполните команду `pip install название_пакета`, где название_пакета - это имя пакета, который вы хотите установить. Например, `pip install requests`.

2. Ручная установка из исходных кодов:

- Скачайте исходные коды пакета с официального сайта разработчика. - Распакуйте скачанный архив.

- Откройте командную строку (терминал) и перейдите в папку с распакованными исходными кодами. - Запустите команду `python setup.py install` для установки пакета.

2. Как осуществить установку менеджера пакетов pip?

Для установки менеджера пакетов pip необходимо выполнить следующие шаги:

1. Убедитесь, что у вас установлен Python. В большинстве операционных систем Python по умолчанию установлен. Вы можете проверить это, выполнив команду `python --version` или `python3 --version` в командной строке. Если Python не установлен, вам нужно будет установить его.

2. Загрузите `get-pip.py` скрипт. Для этого перейдите на страницу https://bootstrap.pypa.io/get-pip.py в браузере и сохраните файл на вашем компьютере.

3. Откройте командную строку (в Windows можно использовать команду `cmd`, а в macOS и Linux - `Terminal`).

4. Перейдите в каталог, где вы сохранили `get-pip.py` скрипт, с помощью команды `cd ПУТЬ_К_ФАЙЛУ` (например, `cd C:\Users\Имя_Пользователя\Downloads`, если файл был сохранен в папке «Загрузки»). Обратите внимание, что вам нужно будет заменить `Имя_Пользователя` на ваше реальное имя пользователя.

5. Установите pip, выполнив следующую команду:

python **get**-pip.py

6. Дождитесь завершения установки pip. После этого можно будет использовать pip для установки и управления пакетами Python.

3. Откуда менеджер пакетов pip по умолчанию устанавливает пакеты?

По умолчанию менеджер пакетов pip устанавливает пакеты в системную директорию Python. В операционных системах Linux и macOS это обычно /usr/local/lib/python3.X/dist-packages, где X - версия Python (например, 3.7). В операционной системе Windows по умолчанию используется C:\Python\PythonXX\Lib\site-packages, где XX - версия Python (например, 37 для Python 3.7).

4. Как установить последнюю версию пакета с помощью pip?

Для установки последней версии пакета с помощью pip, можно использовать команду `pip install --upgrade <название_пакета>`. Эта команда обновит пакет до последней доступной версии. Если пакет еще не установлен, она установит последнюю доступную версию.

5. Как установить заданную версию пакета с помощью pip?

Для установки заданной версии пакета с помощью pip вам нужно выполнить следующую команду в командной строке или терминале:

pip install package_name==version_number

Замените `package_name` на имя пакета, который вы хотите установить, и `version_number` на версию пакета, которую вы хотите установить. Например, если вы хотите установить версию 2.3 пакета requests, выполните следующую команду:

pip install requests==2.3

После выполнения команды pip установит указанную версию пакета. Если указанная версия не найдена или не совместима с вашей системой, pip выдаст ошибку.

6. Как установить пакет из git репозитория (в том числе GitHub) с помощью pip?

Для установки пакета из git репозитория с помощью pip, вы можете использовать следующий синтаксис команды:

pip install git+<**URL_репозитория**>

где `<URL_репозитория>` - это URL адрес git репозитория, откуда вы хотите установить пакет. Например, если вы хотите установить пакет из github репозитория, вы можете использовать команду:

pip install git+https://github.com/имя_пользователя/репозиторий.git

Если репозиторий находится на другом хосте, вы можете заменить https://github.com на URL этого хоста. Вы также можете добавить опции `-e` (или `--editable`) для создания ссылки на репозиторий, что позволит вам вносить изменения в код пакета прямо из репозитория, без необходимости повторной установки пакета.

pip install -e git+https://github.com/имя_пользователя/репозиторий.git

После выполнения команды, pip скачает код из git репозитория и выполнит установку пакета в вашем виртуальном окружении или глобально, в зависимости от настроек уровня пользователя.

7. Как установить пакет из локальной директории с помощью pip?

Чтобы установить пакет из локальной директории с помощью pip, нужно выполнить следующую команду в командной строке:

pip install /path/to/package

где `/path/to/package` - путь к директории, где находится пакет, который вы хотите установить. При этом путь может быть как абсолютным, так и относительным. После выполнения этой команды, пакет будет установлен из указанной директории.

8. Как удалить установленный пакет с помощью pip?

Для удаления установленного пакета с помощью pip вам нужно выполнить следующую команду в командной строке:

pip uninstall <**название-пакета**>

Например, если вы хотите удалить пакет "numpy", вы можете выполнить следующую команду:

pip uninstall numpy

После выполнения этой команды pip удалит установленный пакет с вашей системы.

9. Как обновить установленный пакет с помощью pip?

Для обновления установленного пакета с помощью pip, выполните следующую команду:

pip install --upgrade <package_name>

Здесь `<package_name>` - это имя пакета, который вы хотите обновить. Убедитесь, что вы вводите правильное имя пакета. После выполнения этой команды pip обновит пакет на последнюю доступную версию. Если пакет уже является последней версией, pip выдаст сообщение об этом.

10. Как отобразить список установленных пакетов с помощью pip?

Чтобы отобразить список установленных пакетов с помощью pip, вам нужно выполнить следующую команду в командной строке:

pip list

Эта команда отобразит список всех установленных пакетов вместе с их версиями. Если вы хотите сохранить список в файл для дальнейшего использования, можно использовать редирект оператора ">". Например:

pip list > installed_packages.txt

Эта команда сохранит список установленных пакетов в файле с именем "installed_packages.txt" в текущем рабочем каталоге.

11. Каковы причины появления виртуальных окружений в языке Python?

Существует несколько причин появления виртуальных окружений в языке Python:

1. Изоляция проектов: Виртуальные окружения позволяют изолировать зависимости и установленные пакеты для каждого проекта. Это позволяет иметь разные версии пакетов для разных проектов, избегая конфликтов и обеспечивая надежность и стабильность.

2. Управление зависимостями: Виртуальные окружения предоставляют удобный способ управления зависимостями проекта. Они позволяют

устанавливать, обновлять и удалять пакеты локально без влияния на другие проекты.

3. Переносимость проектов: Виртуальные окружения обеспечивают переносимость проектов между разными системами. Вы можете создать виртуальное окружение на одной системе и передать его на другую без необходимости устанавливать все зависимости заново.

4. Удобство работы в командной строке: Виртуальные окружения позволяют удобно работать с проектом из командной строки. Вы можете активировать нужное виртуальное окружение и использовать его пакеты и команды без конфликтов с другими проектами или системными настройками.

12. Каковы основные этапы работы с виртуальными окружениями?

Основные этапы работы с виртуальными окружениями:

1. Установка и настройка менеджера виртуальных сред (например, virtualenv или Anaconda). Сначала необходимо установить менеджер виртуальных сред, который позволит создавать и управлять виртуальными окружениями.

2. Создание виртуального окружения. После установки менеджера виртуальных сред, можно создать новое виртуальное окружение для проекта. Это можно сделать с помощью команды или специального интерфейса.

3. Активация виртуального окружения. После создания виртуального окружения его необходимо активировать, чтобы использовать его для работы. Активация может быть выполнена с помощью команды или специальных скриптов, предоставляемых менеджером виртуальных сред.

4. Установка зависимостей. После активации виртуального окружения можно установить необходимые зависимости проекта с помощью пакетного менеджера, такого как pip или conda. Установка зависимостей в виртуальное окружение позволяет изолировать их от других проектов и обеспечивает чистоту окружения.

5. Работа с виртуальным окружением. После установки зависимостей можно начать разработку и проводить все необходимые операции с

виртуальным окружением, такие как запуск скриптов, установка дополнительных пакетов, тестирование и отладка.

6. Деактивация виртуального окружения. По окончании работы с виртуальным окружением его можно деактивировать, чтобы не занимать лишние ресурсы. Деактивация может быть выполнена с помощью команды или специального скрипта.

7. Удаление виртуального окружения (при необходимости). Если виртуальное окружение больше не нужно, его можно удалить с помощью команды или интерфейса, предоставляемого менеджером виртуальных сред. Это позволяет освободить место на диске и убрать все связанные с окружением файлы и пакеты.

13. Как осуществляется работа с виртуальными окружениями с помощью venv?

Работа с виртуальными окружениями в Python может быть осуществлена с помощью модуля `venv`, который является стандартной библиотекой Python. Вот некоторые шаги, которые необходимо выполнить для создания и использования виртуального окружения с помощью `venv`:

1. Убедитесь, что у вас установлена версия Python 3.3 или выше, так как `venv` был добавлен в стандартную библиотеку начиная с версии 3.3.

2. Откройте командную строку и перейдите в каталог, где вы хотите создать виртуальное окружение.

3. Создайте виртуальное окружение с помощью команды:

python3 -m venv имя_окружения

Здесь `имя_окружения` - это имя, которое вы хотите присвоить вашему виртуальному окружению. Вы можете выбрать любое удобное для вас имя. **4.** Активируйте виртуальное окружение: - В операционной системе Windows:

имя_окружения\Scripts\activate.bat

После активации виртуального окружения ваша командная строка должна показывать имя окружения перед путь к текущему каталогу. **5.** Теперь

вы можете устанавливать и использовать пакеты, не влияя на глобальную установку пакетов Python. Используйте `pip` для установки пакетов:

pip install пакет

6. Когда вы закончите работать в виртуальном окружении, вы можете его деактивировать с помощью команды:

deactivate

Виртуальное окружение, созданное с использованием `venv`, будет содержать отдельные установленные пакеты и библиотеки, относящиеся только к этому окружению. Это позволяет легко изолировать и управлять зависимостями для различных проектов Python.

14. Как осуществляется работа с виртуальными окружениями с помощью virtualenv?

Работа с виртуальными окружениями с помощью virtualenv осуществляется следующим образом:

1. Установка virtualenv: Если у вас нет virtualenv, установите его с помощью команды:

pip install virtualenv

2. Создание виртуального окружения: В папке вашего проекта выполните следующую команду:

virtualenv <имя окружения>

где `<имя окружения>` - имя виртуального окружения, которое вы хотите создать.

3. Активация виртуального окружения: Чтобы активировать виртуальное окружение на Windows, выполните команду:

<имя окружения>\Scripts\activate

После активации виртуального окружения вы увидите его имя в начале командной строки.

4. Установка зависимостей: В активированном виртуальном окружении вы можете установить все необходимые зависимости, используя команду pip. Например:

pip install <название пакета>

5. Выход из виртуального окружения: Чтобы выйти из активированного виртуального окружения, выполните команду:

deactivate

Теперь вы можете работать в вашем виртуальном окружении, отдельно от системной установки пакетов и зависимостей. Это помогает изолировать проект и обеспечить его независимую среду.

15. Изучите работу с виртуальными окружениями pipenv. Как осуществляется работа с виртуальными окружениями pipenv?

Работа с виртуальными окружениями с использованием pipenv включает несколько шагов:

1. Установка pipenv: для этого можно использовать pip, инструмент установки пакетов Python. В командной строке выполните:

pip install pipenv

2. Создание нового виртуального окружения: перейдите в директорию вашего проекта и выполните команду:

pipenv install

Эта команда создаст новое виртуальное окружение и установит пакеты, указанные в файле Pipfile.

3. Активация виртуального окружения: выполните команду:

pipenv shell

Она активирует виртуальное окружение и переключит вашу командную строку в контекст этого окружения.

4. Установка пакетов: вы можете устанавливать пакеты, используя команду `pipenv install`. Например, для установки пакета requests выполните:

pipenv install requests

5. Запуск скриптов: чтобы запустить скрипт, использующий установленные пакеты, выполните команду `pipenv run`. Например, для запуска скрипта `my_script.py` выполните:

pipenv run python my_script.py

6. Деактивация виртуального окружения: чтобы выйти из виртуального окружения, выполните команду `exit` или `Ctrl+D`. Кроме того, pipenv предоставляет другие полезные команды, такие как `pipenv lock` для создания файла `Pipfile.lock`, фиксирующего версии установленных пакетов, и `pipenv sync` для установки пакетов из `Pipfile.lock`.

В целом, pipenv обеспечивает удобное управление зависимостями и виртуальными окружениями, упрощая разработку и управление проектами на языке Python.

16. Каково назначение файла requirements.txt ? Как создать этот файл? Какой он имеет формат?

Файл requirements.txt в основном используется в проектах Python для описания зависимостей проекта. Он содержит список всех пакетов и их версий, необходимых для правильной работы проекта.

Для создания файла requirements.txt можно использовать команду pip freeze, которая создаст список всех установленных пакетов и их версий в текущей среде разработки Python. Для создания файла можно выполнить следующую команду:

pip freeze > requirements.txt

Эта команда создаст файл requirements.txt и запишет в него список пакетов и их версий. Формат файла requirements.txt очень простой. Каждая строка файла содержит имя пакета и его версию, разделенные знаком ==. Например:

requests==2.24.0

numpy==1.18.5

Также можно использовать другие операторы версий, такие как >=, <=, >, <, !=, чтобы указать диапазон версий, которые будут установлены. Файл requirements.txt может быть передан в другую среду разработки или другому разработчику, чтобы установить все зависимости проекта одной командой. Это также полезно при работе с виртуальными средами разработки или при развертывании проекта на сервере.

17. В чем преимущества пакетного менеджера conda по сравнению с пакетным менеджером pip?

Conda и pip - это два основных пакетных менеджера для языка программирования Python, и у каждого из них есть свои преимущества. Преимущества пакетного менеджера conda по сравнению с pip:

1. Управление зависимостями: Conda обрабатывает зависимости не только для языка Python, но и для других языков, таких как R, C++, Java и других. Он предоставляет среду управления пакетами, которая позволяет установить и управлять зависимостями, включая необходимые библиотеки для других языков программирования. В то время как pip устанавливает только Python-зависимости.

2. Воспроизводимость среды: Conda позволяет создавать изолированные среды (с помощью виртуальных окружений), которые содержат все необходимые зависимости для выполнения конкретного проекта или эксперимента. Это помогает обеспечить консистентность и воспроизводимость среды, что важно, особенно когда вы работаете с большим проектом или коллаборативно. Pip также предоставляет виртуальные окружения, но они могут быть менее надежными и более сложными в использовании.

3. Более широкий выбор пакетов: Conda предлагает широкий выбор пакетов, включая как Python-специфичные пакеты, так и множество пакетов для других языков программирования и научных вычислений. Pip в основном ориентирован на установку Python-пакетов.

4. Устранение проблем совместимости: Conda может управлять различными версиями одной библиотеки и заменять модули, которые могут конфликтовать между пакетами. Это позволяет избежать проблем совместимости между разными пакетами и обеспечить гладкую работу всех зависимостей. Pip решает проблемы совместимости, но иногда может возникнуть конфликт между зависимостями.

5. Управление дистрибутивами: Conda также предлагает возможность установки не только пакетов из Python Package Index (PyPI), но и из других дистрибутивов пакетов, таких как Anaconda Cloud, Conda-Forge и других. Это дает более широкий выбор и большую гибкость при установке пакетов.

18. В какие дистрибутивы Python входит пакетный менеджер conda?

Пакетный менеджер conda входит в следующие дистрибутивы Python:

1. Anaconda: Anaconda является дистрибутивом Python, который включает в себя конду, а также множество пакетов и инструментов для анализа данных, визуализации и разработки.

2. Miniconda: Miniconda - это минимальная версия Anaconda, которая включает в себя только конду и некоторые базовые пакеты. Он предоставляет пользователям возможность настраивать свою среду Python в соответствии с их потребностями, установив только необходимые пакеты.

Оба дистрибутива (Anaconda и Miniconda) предлагают конду, что позволяет легко управлять пакетами и создавать изолированные среды для разработки в Python.

19. Как создать виртуальное окружение conda?

Чтобы создать виртуальное окружение в conda, выполните следующие шаги:

1. Установите Anaconda, если еще не сделали это, скачав и запустив установщик Anaconda с официального сайта Anaconda (https://www.anaconda.com/products/individual).

2. Откройте терминал или командную строку.

3. Введите следующую команду, чтобы создать новое виртуальное окружение conda с именем "myenv" (вместо "myenv" вы можете выбрать любое другое имя окружения):

```
conda create --name myenv
```

4. При выполнении команды conda попросит подтверждение. Введите 'y' и нажмите Enter, чтобы продолжить создание окружения.

5. Conda начнет загрузку и установку необходимых пакетов для нового окружения.

6. После завершения установки можно активировать новое окружение с помощью команды: - Для Windows:

conda activate myenv

После активации окружение будет изменяться на "myenv", и вы сможете установить и использовать пакеты Python, специфичные для этого окружения. Вы также можете указать конкретную версию Python для нового окружения, добавив аргумент `python=x.x` в команду создания окружения. Например, `conda create --name myenv python=3.8` создаст новое окружение с Python версии 3.8.

20. Как активировать и установить пакеты в виртуальное окружение conda?

Для активации и установки пакетов в виртуальное окружение conda, следуйте следующим шагам:

1. Откройте командную строку или терминал, в зависимости от вашей операционной системы.

2. Активируйте виртуальное окружение conda с помощью следующей команды:

conda activate <**название_виртуального_окружения**>

3. После активации виртуального окружения, вы можете установить необходимые пакеты с помощью команды `conda install`. Например, чтобы установить пакет numpy, выполните следующую команду:

conda install numpy

4. Если вы хотите установить пакет из пакетного репозитория Anaconda, вы можете использовать команду `conda search` для поиска доступных версий пакета. Например, чтобы найти версии пакета numpy, выполните следующую команду:

conda **search** numpy

Затем выберите нужную версию пакета и установите его с помощью команды `conda install`, указав версию пакета. Например:

conda install numpy=1.18.1

5. Если вы хотите установить пакет, который не является частью пакетного репозитория Anaconda, вы можете использовать команду `conda-forge`. Например, чтобы установить пакет matplotlib из репозитория conda-forge, выполните следующую команду:

conda install -c conda-forge matplotlib

6. После установки всех необходимых пакетов, вы можете проверить список установленных пакетов в вашем виртуальном окружении с помощью команды `conda list`. **7.** Чтобы выйти из виртуального окружения conda, выполните команду:

conda deactivate

21. Как деактивировать и удалить виртуальное окружение conda?

Для деактивации и удаления виртуального окружения conda выполните следующие шаги:

1. Деактивация окружения:

- На Windows: Откройте командную строку (Command Prompt) и выполните `conda deactivate`.

- На macOS и Linux: Откройте терминал и выполните `conda deactivate`.

2. Удаление окружения:

- На Windows: Откройте командную строку (Command Prompt) и выполните `conda env remove --name <название_окружения>`.

- На macOS и Linux: Откройте терминал и выполните `conda env remove --name <название_окружения>`.

Замените `<название_окружения>` на название вашего виртуального окружения, которое вы хотите удалить.

22. Каково назначение файла environment.yml ? Как создать этот файл?

Файл environment.yml используется в среде разработки Anaconda для создания и организации виртуальной среды (env) и управления зависимостями в проекте.

Назначение файла environment.yml состоит в следующем:

1. Задание списка пакетов и их версий, необходимых для проекта.

2. Создание виртуальной среды с заданными пакетами и их версиями.

3. Обеспечение воспроизводимости окружения проекта для других пользователей.

Для создания файла environment.yml:

1. Откройте командную строку или терминал.

2. Перейдите в директорию проекта или ту директорию, в которой вы хотите создать файл environment.yml.

3. Запустите команду conda env export > environment.yml. Эта команда экспортирует текущее окружение conda в файл environment.yml.

4. Файл environment.yml будет создан в текущей директории проекта. Файл environment.yml может быть редактирован вручную для настройки пакетов и их версий перед сборкой окружения с помощью команды conda env create -f environment.yml.

23. Как создать виртуальное окружение conda с помощью файла environment.yml ?

Чтобы создать виртуальное окружение conda с использованием файла environment.yml, выполните следующие шаги:

1. Откройте командную строку или терминал.

2. Перейдите в папку, где находится файл environment.yml. Для этого можно использовать команду `cd`:

```
cd путь_к_папке
```

3. Создайте виртуальное окружение с помощью команды `conda env create`:

```
conda env create -f environment.yml
```

Команда `conda env create` создает новое виртуальное окружение, а флаг `-f` указывает на файл environment.yml, который содержит список пакетов и их зависимостей.

4. Дождитесь завершения процесса создания виртуального окружения. Затем можно активировать его с помощью команды `conda activate`:

conda activate название_окружения

24. Самостоятельно изучите средства IDE PyCharm для работы с виртуальными окружениями conda. Опишите порядок работы с виртуальными окружениями conda в IDE PyCharm.

PyCharm предоставляет несколько удобных средств для работы с виртуальными окружениями. Вот некоторые из них:

1. Создание виртуального окружения: Вы можете создать новое виртуальное окружение с помощью PyCharm, следуя простым шагам. Для этого перейдите в "Settings" (или "Preferences" на macOS), выберите "Project: <имя проекта>" и "Python interpreter". Затем щелкните на значке шестеренки рядом с выпадающим списком интерпретатора Python и выберите "Create VirtualEnv". Укажите имя и путь к новому виртуальному окружению.

2. Активация виртуального окружения: После создания виртуального окружения, вы можете активировать его, чтобы использовать его в своем проекте. Для активации виртуального окружения в PyCharm, щелкните на значке шестеренки рядом с выпадающим списком интерпретатора Python и выберите уже созданное вами виртуальное окружение.

3. Использование pip: PyCharm обеспечивает легкий доступ к инструменту pip для установки пакетов в ваше виртуальное окружение. Вы можете установить новый пакет, выбрав "Python interpreter" в настройках проекта и щелкнув по значку плюса для установки дополнительных пакетов.

4. Редактирование файлов конфигурации: Подробный контроль над виртуальными окружениями в PyCharm может быть достигнут с помощью файла конфигурации "pyvenv.cfg". Вы можете добавлять, удалять и изменять виртуальные окружения через этот файл.

5. Автообнаружение виртуальных окружений: PyCharm автоматически обнаруживает виртуальные окружения в вашем проекте и предлагает их использовать без необходимости вручную указывать путь к ним.

25. Почему файлы requirements.txt и environment.yml должны храниться в репозитории git?

Файлы requirements.txt и environment.yml содержат информацию о необходимых зависимостях и настройках для работы проекта. Эти файлы хранятся в репозитории git по нескольким причинам:

1. Воспроизводимость: Файлы requirements.txt и environment.yml позволяют другим разработчикам воспроизвести окружение проекта, включая все зависимости. Это способствует консистентности разработки и упрощает развертывание проекта на других машинах.

2. Управление зависимостями: Хранение этих файлов в репозитории позволяет контролировать версию зависимостей проекта. Если разработчикам понадобится вернуться к определенной версии зависимости, они смогут найти ее в истории репозитория.

3. Команда или комьюнити: Если несколько разработчиков работают над проектом, то файлы requirements.txt и environment.yml помогут всем участникам использовать одинаковые версии зависимостей и окружения. Это позволяет упростить процесс совместной работы и избежать потенциальных конфликтов из-за несовместимых версий зависимостей.

4. Развертывание: Файлы requirements.txt и environment.yml могут быть использованы для автоматического развертывания проекта на сервере. Наличие этих файлов в репозитории упрощает процесс развертывания и минимизирует возможные проблемы с зависимостями на сервере.

В целом, хранение файлов requirements.txt и environment.yml в репозитории git помогает обеспечить консистентность, контроль версий и упрощенное совместное использование проекта и его зависимостей.

Вывод: приобрел навыки по работе с менеджером пакетов pip и виртуальными окружениями с помощью языка программирования Python версии 3.x.