

**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 №2.14

Дисциплина: «Основы программной инженерии»

Тема: «Установка пакетов в Python. Виртуальные окружения»

Выполнила:

студентка 2 курса

группы Пиж-б-о-21-1

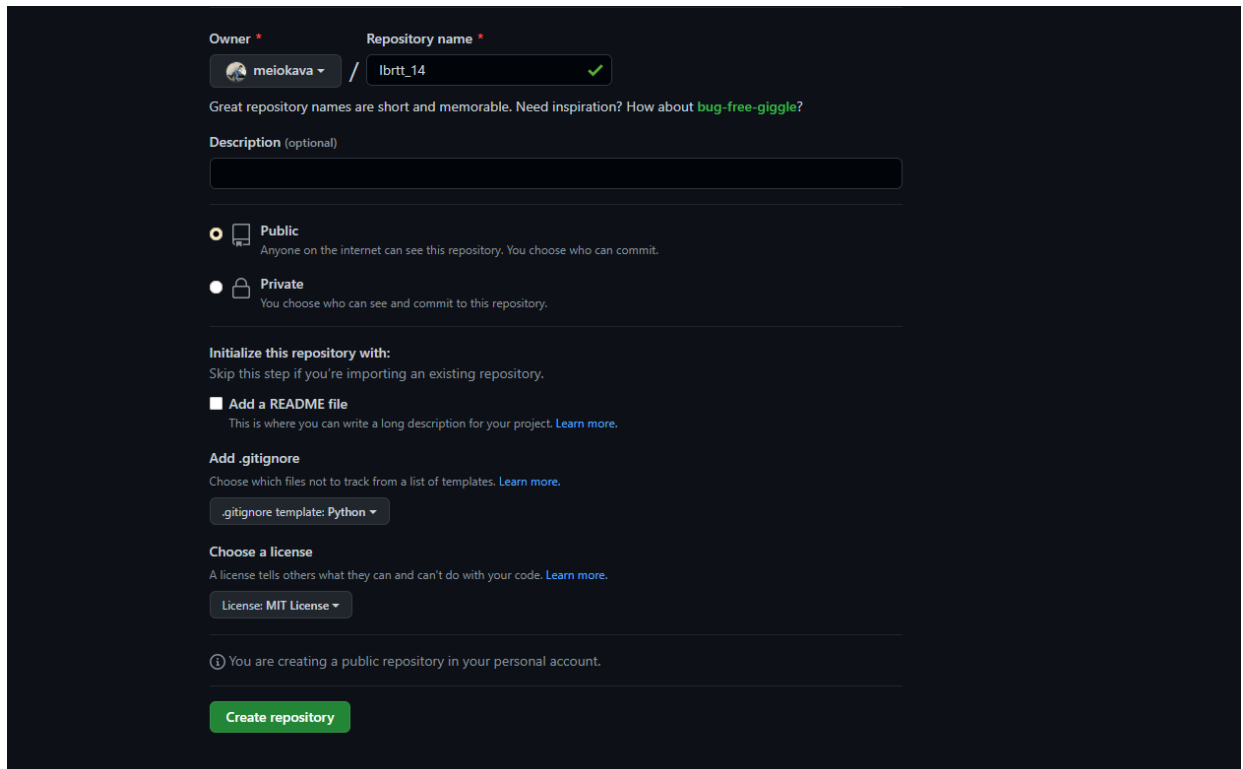
Джолдошова Мээрим

Бекболотовна

Ставрополь 2022

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

1. Был создан репозиторий в Github в который были добавлены правила gitignore для работы IDE PyCharm, была выбрана лицензия MIT, сам репозиторий был клонирован на локальный сервер и был организован в соответствии с моделью ветвления git-flow.



Owner * Repository name *

meiokava / lbrtt_14 ✓

Great repository names are short and memorable. Need inspiration? How about [bug-free-giggle?](#)

Description (optional)

☐ Public
Anyone on the internet can see this repository. You choose who can commit.

☐ Private
You choose who can see and commit to this repository.

Initialize this repository with:
Skip this step if you're importing an existing repository.

☒ Add a README file
This is where you can write a long description for your project. [Learn more.](#)

Add .gitignore
Choose which files not to track from a list of templates. [Learn more.](#)

.gitignore template: Python

Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: MIT License

ⓘ You are creating a public repository in your personal account.

Create repository

Рисунок 1 – Создание репозитория

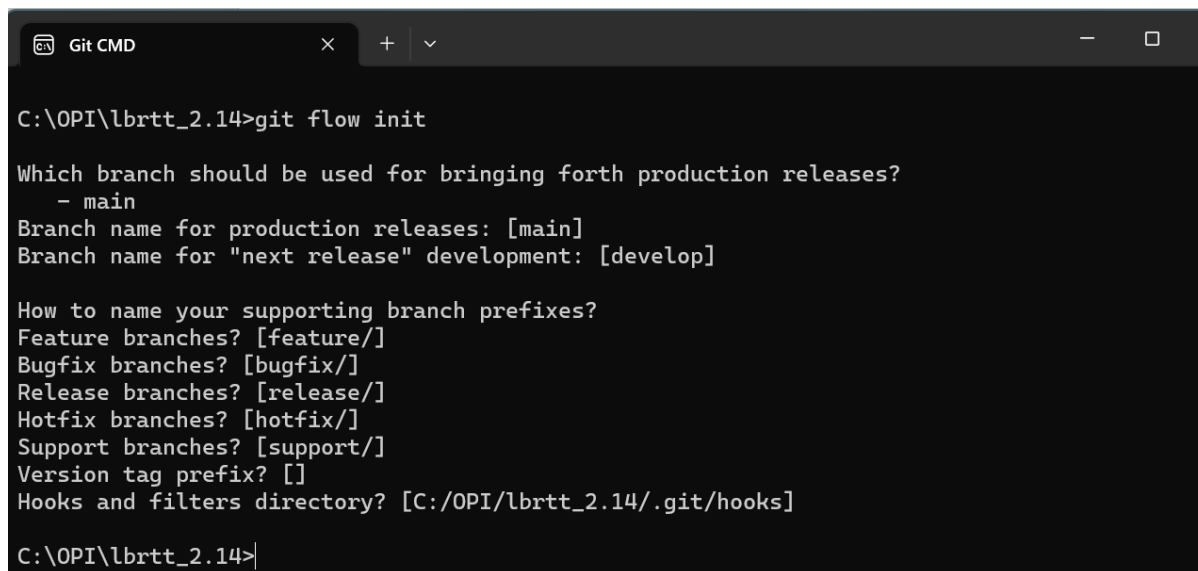
```
C:\Users\mвидео>cd/d C:\OPI

C:\OPI>git clone https://github.com/meiokava/lbrtt_2.14.git
Cloning into 'lbrtt_2.14'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (4/4), done.

C:\OPI>cd/d C:\OPI\lbrtt_2.14

C:\OPI\lbrtt_2.14>
```

Рисунок 2 – Клонирование репозитория



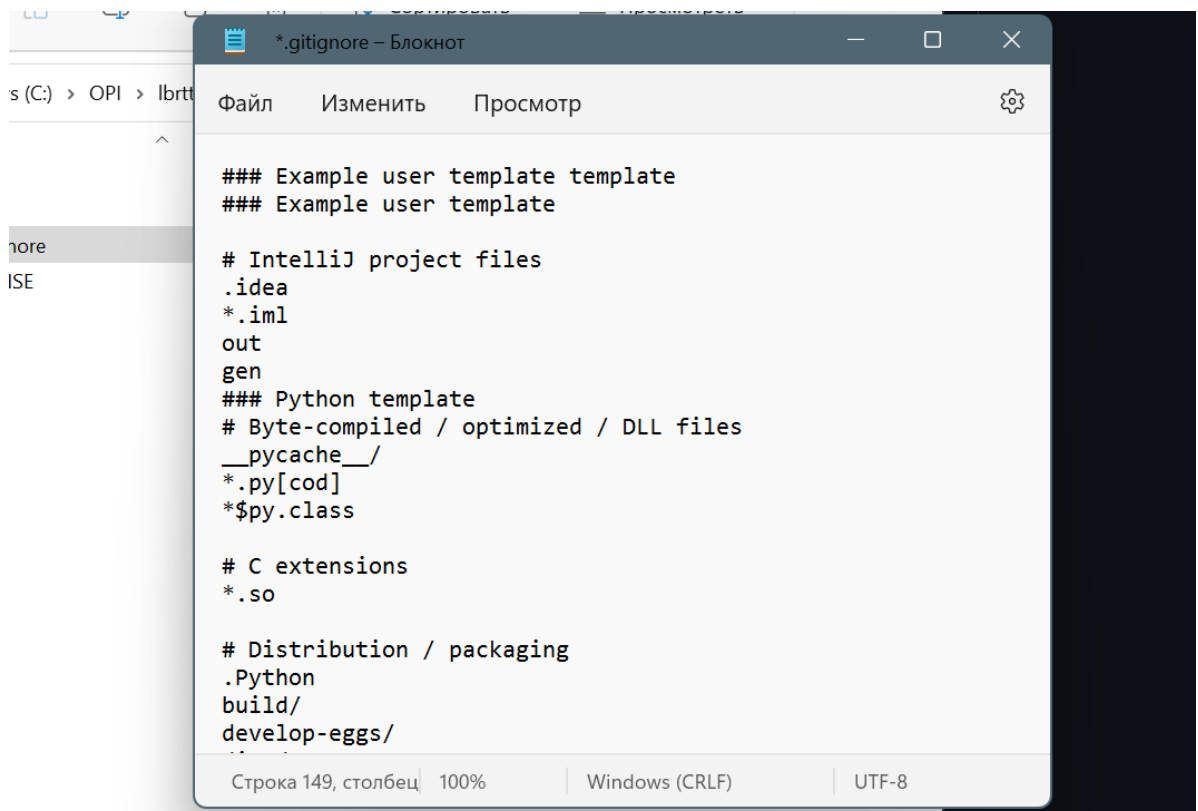
```
C:\OPI\lbrtt_2.14>git flow init

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:/OPI/lbrtt_2.14/.git/hooks]

C:\OPI\lbrtt_2.14>
```

Рисунок 3 – Организация модели ветвления git-flow



```
*.gitignore – Блокнот
Файл  Изменить  Просмотр

### Example user template template
### Example user template

# IntelliJ project files
.idea
*.iml
out
gen
### Python template
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/

Строка 149, столбец 100% | Windows (CRLF) | UTF-8
```

Рисунок 4 – Дополнение файла .gitignore

```
C:\OPI\lbrtt_2.14>python -m venv env

C:\OPI\lbrtt_2.14>
```

Рисунок 5 – Создание виртуального окружения

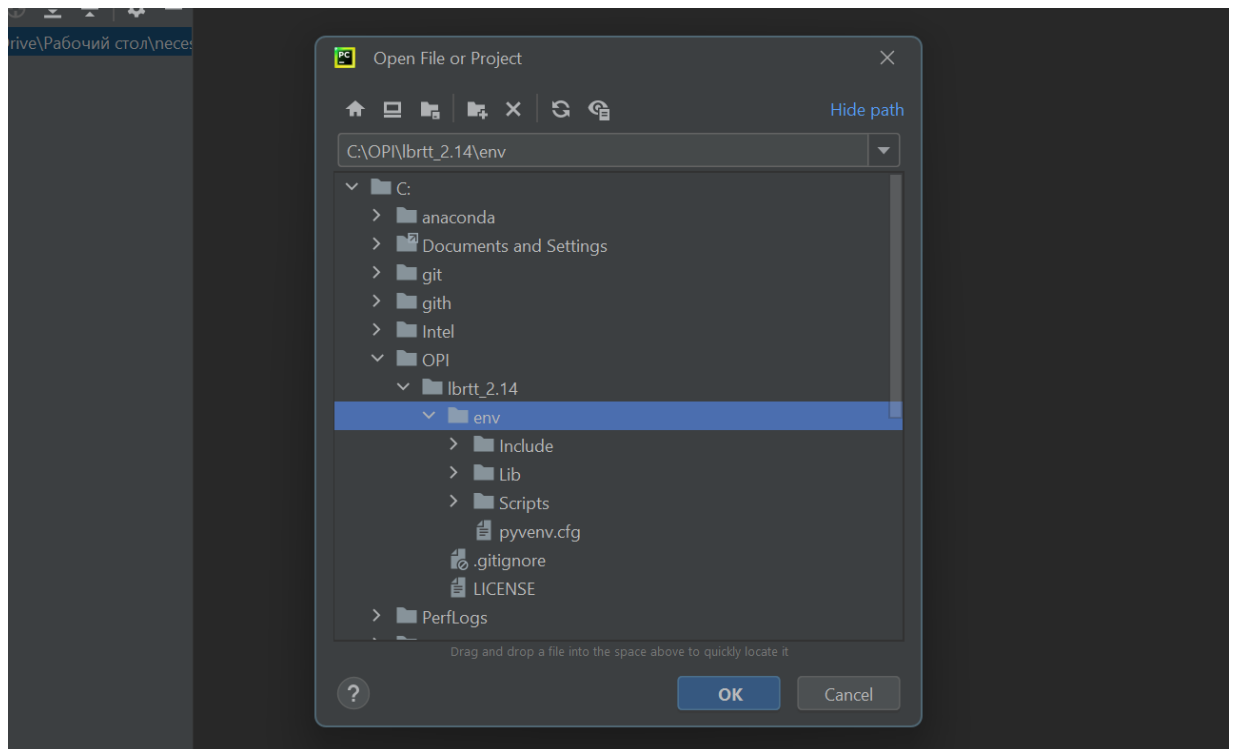


Рисунок 6 – Содержимое папки env

```
C:\OPI>.\venv\Scripts\activate
```

Рисунок 7 – Активация виртуального окружения

```
(venv) C:\OPI>pip install pygame
Collecting pygame
  Downloading pygame-2.1.2-cp310-cp310-win_amd64.whl (8.4 MB)
    8.4/8.4 MB 6.7 MB/s eta 0:00:00
Installing collected packages: pygame
Successfully installed pygame-2.1.2

[notice] A new release of pip available: 22.2.2 -> 23.0
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Рисунок 8 – Установка пакета pygame

```
(venv) C:\OPI>pip freeze
pygame==2.1.2

(venv) C:\OPI>pip freeze > requirements.txt

(venv) C:\OPI>
```

Рисунок 9 – Перенос виртуального окружения и перенаправление в файл

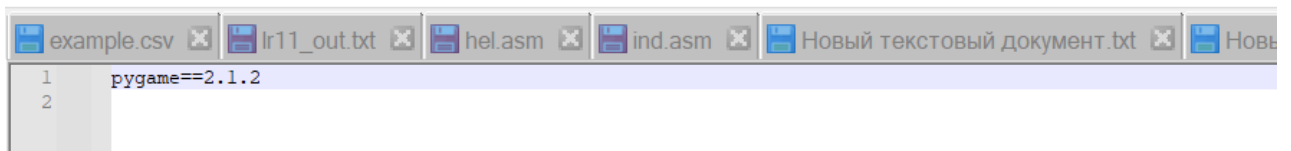


Рисунок 10 – Содержимое файла

1. Создание виртуального окружения Anaconda с именем репозитория.

```
(base) PS C:\OPI> mkdir lb_2.14
```

Каталог: C:\OPI

Mode	LastWriteTime	Length	Name
d-----	09.02.2023 20:11		lb_2.14

```
(base) PS C:\OPI> |
```

```
(base) PS C:\OPI\lb_2.14> copy NUL > main.py
```

Рисунок 11 – Создание чистого виртуального окружения с conda

```
(base) PS C:\OPI\lb_2.14> cd ../
(base) PS C:\OPI> conda create -n lb_2.14 python=3.10
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\anaconda\envs\lb_2.14

added / updated specs:
  - python=3.10

The following packages will be downloaded:
```

package	build	
ca-certificates-2023.01.10	haa95532_0	121 KB
certifi-2022.12.7	py310haa95532_0	149 KB
libffi-3.4.2	hd77b12b_6	109 KB
openssl-1.1.1s	h2bbff1b_0	5.5 MB
pip-22.3.1	py310haa95532_0	2.8 MB
python-3.10.9	h966fe2a_0	15.8 MB
setuptools-65.6.3	py310haa95532_0	1.2 MB
sqlite-3.40.1	h2bbff1b_0	889 KB
tzdata-2022g	h04d1e81_0	114 KB
wincertstore-0.2	py310haa95532_2	15 KB
xz-5.2.10	h8cc25b3_1	520 KB
zlib-1.2.13	h8cc25b3_0	113 KB
Total:		27.2 MB

```

The following NEW packages will be INSTALLED:

bzip2                pkgs/main/win-64::bzip2-1.0.8-he774522_0
ca-certificates      pkgs/main/win-64::ca-certificates-2023.01.10-haa95532_0
certifi              pkgs/main/win-64::certifi-2022.12.7-py310haa95532_0
libffi               pkgs/main/win-64::libffi-3.4.2-hd77b12b_6
openssl              pkgs/main/win-64::openssl-1.1.1s-h2bbff1b_0
pip                  pkgs/main/win-64::pip-22.3.1-py310haa95532_0
python               pkgs/main/win-64::python-3.10.9-h966fe2a_0
setuptools           pkgs/main/win-64::setuptools-65.6.3-py310haa95532_0
sqlite               pkgs/main/win-64::sqlite-3.40.1-h2bbff1b_0
tk                   pkgs/main/win-64::tk-8.6.12-h2bbff1b_0
tzdata               pkgs/main/noarch::tzdata-2022g-h04d1e81_0
vc                   pkgs/main/win-64::vc-14.2-h21ff451_1
vs2015_runtime       pkgs/main/win-64::vs2015_runtime-14.27.29016-h5e58377_2
wheel                pkgs/main/noarch::wheel-0.37.1-pyhd3eb1b0_0
wincertstore         pkgs/main/win-64::wincertstore-0.2-py310haa95532_2
xz                   pkgs/main/win-64::xz-5.2.10-h8cc25b3_1
zlib                 pkgs/main/win-64::zlib-1.2.13-h8cc25b3_0

Proceed ([y]/n)? y

```

Downloading and Extracting Packages

```

Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#   $ conda activate lb_2.14
#
# To deactivate an active environment, use
#
#   $ conda deactivate

(base) PS C:\OPI> conda activate lb_2.14
(lb_2.14) PS C:\OPI>

```

Рисунок 12 – его активация

2. Установка в виртуальное окружение следующие пакеты: `pip`, `NumPy`, `Pandas`, `SciPy`.

```
(lb_2.14) PS C:\OPI> conda install pip, NumPy, Pandas, SciPy
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

environment location: C:\anaconda\envs\lb_2.14

added / updated specs:
- numpy
- pandas
- pip
- scipy

The following packages will be downloaded:
```

package	build	
bottleneck-1.3.5	py310h9128911_0	106 KB
brotlipy-0.7.0	py310h2bbff1b_1002	335 KB
cffi-1.15.1	py310h2bbff1b_3	239 KB
cryptography-38.0.4	py310h21b164f_0	1.0 MB
idna-3.4	py310haa95532_0	97 KB
mkl-service-2.4.0	py310h2bbff1b_0	48 KB
mkl_fft-1.3.1	py310ha0764ea_0	136 KB
mkl_random-1.2.2	py310h4ed8f06_0	221 KB
numexpr-2.8.4	py310hd213c9f_0	128 KB
numpy-1.23.5	py310h60c9a35_0	11 KB
numpy-base-1.23.5	py310h04254f7_0	6.0 MB
packaging-22.0	py310haa95532_0	68 KB
pandas-1.5.2	py310h4ed8f06_0	10.5 MB

Рисунок 13 – установка пакетов в виртуальное окружение

3. Установка менеджером пакетов conda пакет TensorFlow.

```
(lb_2.14) PS C:\OPI> conda install TensorFlow
Collecting package metadata (current_repodata.json): done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##
```

Downloading and Extracting Packages

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(lb_2.14) PS C:\OPI> |
```

Рисунок 14 – Установка TensorFlow

4. Установка пакета TensorFlow с помощью менеджера пакетов pip.

```
Anaconda Powershell Prompt
(lb_2.14) PS C:\OPI> pip install TensorFlow
Requirement already satisfied: TensorFlow in c:\anaconda\envs\lb_2.14\lib\site-packages (2.10.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (1.42.0)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (4.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (0.2.0)
Requirement already satisfied: numpy>=1.20 in c:\users\мвдео\appdata\roaming\python\python310\site-packages (from TensorFlow) (1.24.2)
Collecting libclang>=13.0.0
  Downloading libclang-15.0.6.1-py2.py3-none-win_amd64.whl (23.2 MB)
    23.2/23.2 MB 4.6 MB/s eta 0:00:00
Requirement already satisfied: wrapt>=1.11.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (1.14.1)
Collecting protobuf<3.20,>=3.9.2
  Downloading protobuf-3.19.6-cp310-cp310-win_amd64.whl (895 kB)
    895.7/895.7 kB 5.1 MB/s eta 0:00:00
Requirement already satisfied: keras-preprocessing>=1.1.1 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (1.1.2)
Requirement already satisfied: tensorflow-estimator<2.11,>=2.10.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (2.10.0)
Requirement already satisfied: keras<2.11,>=2.10.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (2.10.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (2.1.0)
Requirement already satisfied: packaging in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (22.0)
Requirement already satisfied: flatbuffers>=2.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (2.0)
Requirement already satisfied: setuptools in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (65.6.3)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (0.4.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (3.3.0)
Requirement already satisfied: six>=1.12.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (1.16.0)
Requirement already satisfied: tensorboard<2.11,>=2.10 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (2.10.0)
Collecting tensorflow-io-gcs-filesystem>=0.23.1
  Downloading tensorflow_io_gcs_filesystem-0.30.0-cp310-cp310-win_amd64.whl (1.5 MB)
    1.5/1.5 MB 5.5 MB/s eta 0:00:00
Requirement already satisfied: astunparse>=1.6.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (1.6.3)
Requirement already satisfied: absl-py>=1.0.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (1.3.0)
Requirement already satisfied: h5py>=2.9.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from TensorFlow) (3.7.0)
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in c:\anaconda\envs\lb_2.14\lib\site-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.11,>=2.10->TensorFlow) (0.4.8)
Requirement already satisfied: oauthlib>=3.0.0 in c:\anaconda\envs\lb_2.14\lib\site-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<0.5,>=0.4.1->tensorboard<2.11,>=2.10->TensorFlow) (3.2.1)
Installing collected packages: libclang, tensorflow-io-gcs-filesystem, protobuf
  Attempting uninstall: protobuf
    Found existing installation: protobuf 3.20.3
    Uninstalling protobuf-3.20.3:
      Successfully uninstalled protobuf-3.20.3
Successfully installed libclang-15.0.6.1 protobuf-3.19.6 tensorflow-io-gcs-filesystem-0.30.0
(lb_2.14) PS C:\OPI>
```

Рисунок 15 – Установка TensorFlow с помощью менеджера пакетов pip

5. Сформировать файлы requirements.txt и environment.yml.


	environment.yml	09.02.2023 20:34	Исходны
	requirements.txt	09.02.2023 19:49	Файл "ТХ"

Рисунок 16 – Файлы requirements.txt и environment.yml

```
C:\OPI\lbrtt_2.14>git add .
C:\OPI\lbrtt_2.14>git commit -m "completed tasks"
[develop 58e8469] completed tasks
3 files changed, 23 insertions(+), 2 deletions(-)
create mode 100644 environment.yml
create mode 100644 requirements.txt
```

Рисунок 17 – Коммит изменений


```
C:\OPI\lbrtt_2.14>git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

```
C:\OPI\lbrtt_2.14>git push
Everything up-to-date

C:\OPI\lbrtt_2.14>|
```

Рисунок 18 – слияние и пуш на удаленный сервер

Контр. вопросы и ответы на них:

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

Существует так называемый Python Package Index (PyPI) – это репозиторий, открытый для всех Python разработчиков, в нем вы можете найти пакеты для решения практически любых задач.

2. Как осуществить установку менеджера пакетов pip?

При развертывании современной версии Python, pip устанавливается автоматически. Но если, по какой-то причине, pip не установлен на вашем ПК, то сделать это можно вручную. Чтобы установить pip, нужно скачать скрипт get-pip.py и выполнить его.

3. Откуда менеджер пакетов pip по умолчанию устанавливает пакеты?

По умолчанию менеджер пакетов pip скачивает пакеты из Python Package Index (PyPI).

4. Как установить последнюю версию пакета с помощью pip?

С помощью команды \$ pip install ProjectName.

5. Как установить заданную версию пакета с помощью pip?

С помощью команды \$ pip install ProjectName==3.2, где вместо 3.2 необходимо указать нужную версию пакета.

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

С помощью команды `$ pip install e git+https://gitrepo.com/ProjectName.git`

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

С помощью команды `$ pip install ./dist/ProjectName.tar.gz`

8. Как удалить установленный пакет с помощью pip?

С помощью команды `$ pip uninstall ProjectName` можно удалить установленный пакет.

9. Как обновить установленный пакет с помощью pip?

С помощью команды `$ pip install --upgrade ProjectName` можно обновить необходимый пакет.

10. Как отобразить список установленных пакетов с помощью pip?

Командой `$ pip list` можно отобразить список установленных пакетов.

11. Каковы причины появления виртуальных окружений в языке Python?

Существует несколько причин появления виртуальных окружений в языке Python - проблема обратной совместимости и проблема коллективной разработки. Проблема обратной совместимости - некоторые операционные системы, например, Linux и MacOS используют содержащиеся в них предустановленные интерпретаторы Python. Обновив или изменив самостоятельно версию какого-то установленного глобально пакета, мы можем непреднамеренно сломать работу утилит и приложений из дистрибутива операционной системы. Проблема коллективной разработки - Если разработчик работает над проектом не один, а с командой, ему нужно передавать и получать список зависимостей, а также обновлять их на своем компьютере таким образом, чтобы не нарушалась работа других его проектов. Значит нам нужен механизм, который вместе с обменом проектами быстро устанавливал бы локально и все необходимые для них пакеты, при этом не мешая работе других проектов.

12. Каковы основные этапы работы с виртуальными окружениями?

Основные этапы:

Создаём через утилиту новое виртуальное окружение в отдельной папке для выбранной версии интерпретатора Python.

Активируем ранее созданное виртуального окружения для работы.

Работаем в виртуальном окружении, а именно управляем пакетами используя `pip` и запускаем выполнение кода.

Деактивируем после окончания работы виртуальное окружение.

Удаляем папку с виртуальным окружением, если оно нам больше не нужно.

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

С его помощью можно создать виртуальную среду, в которую можно устанавливать пакеты независимо от основной среды или других виртуальных окружений. Основные действия с виртуальными окружениями с помощью `venv`: создание виртуального окружения, его активация и деактивация.

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

Для начала пакет нужно установить. Установку можно выполнить командой: `python3 -m pip install virtualenv` `Virtualenv` позволяет создать абсолютно изолированное виртуальное окружение для каждой из программ. Окружением является обычная директория, которая содержит копию всего необходимого для запуска определенной программы, включая копию самого интерпретатора, полной стандартной библиотеки, `pip`, и, что самое главное, копии всех необходимых пакетов.

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

Для формирования и развертывания пакетных зависимостей используется утилита `pip`.

Основные возможности `pipenv`:

- Создание и управление виртуальным окружением
- Синхронизация пакетов в `Pipfile` при установке и удалении пакетов
- Автоматическая подгрузка переменных окружения из `.env` файла

После установки `pipenv` начинается работа с окружением. Его можно

создать в любой папке. Достаточно установить любой пакет внутри папки. Используем `requests`, он автоматически установит окружение и создаст `Pipfile` и `Pipfile.lock`.

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

Установить пакеты можно с помощью команды: `pip install -r requirements.txt`. Также можно использовать команду `pip freeze > requirements.txt`, которая создаст `requirements.txt` наполнив его названиями и версиями тех пакетов что используются вами в текущем окружении. Это удобно если вы разработали проект и в текущем окружении все работает, но вы хотите перенести проект в иное окружение (например, заказчику или на сервер). С помощью закрепления зависимостей мы можем быть уверены, что пакеты, установленные в нашей производственной среде, будут точно соответствовать пакетам в нашей среде разработки, чтобы ваш проект неожиданно не ломался.

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

`Conda` способна управлять пакетами как для `Python`, так и для `C/ C++`, `R`, `Ruby`, `Lua`, `Scala` и других. `Conda` устанавливает двоичные файлы, поэтому работу по компиляции пакета самостоятельно выполнять не требуется (по сравнению с `pip`).

18. В какие дистрибутивы `Python` входит пакетный менеджер `conda`?

Все чаще среди `Python`-разработчиков заходит речь о менеджере пакетов `conda`, включенный в состав дистрибутивов `Anaconda` и `Miniconda`. `JetBrains` включил этот инструмент в состав `PyCharm`.

19. Как создать виртуальное окружение `conda`?

С помощью команды: `conda create -n %PROJ_NAME% python=3.7`

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

Чтобы установить пакеты, необходимо воспользоваться командой: —

conda install A для активации: conda activate %PROJ_NAME%

21. Как деактивировать и удалить виртуальное окружение conda?

Для деактивации использовать команду: conda deactivate, а для удаления: conda remove -n \$PROJ_NAME.

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

Создание файла: conda env export > environment.yml

Файл environment.yml позволит воссоздать окружение в любой нужный момент.

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

Достаточно набрать: conda env create -f environment.yml

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

Работа с виртуальными окружениями в PyCharm зависит от способа взаимодействия с виртуальным окружением:

Создаём проект со своим собственным виртуальным окружением, куда затем будут устанавливаться необходимые библиотеки.

Предварительно создаём виртуальное окружение, куда установим нужные библиотеки. И затем при создании проекта в PyCharm можно будет его выбирать, т.е. использовать для нескольких проектов. Для первого способа ход работы следующий: запускаем PyCharm и в окне приветствия выбираем Create New Project. В мастере создания проекта, указываем в поле Location путь расположения создаваемого проекта. Имя конечной директории также является именем проекта. Далее разворачиваем параметры окружения, щелкая по Project Interpreter. И выбираем New environment using Virtualenv. Путь расположения окружения генерируется автоматически. И нажимаем на Create. Теперь установим библиотеки, которые будем использовать в программе. С помощью главного меню переходим в настройки

File → Settings. Где переходим в Project: project_name → Project

Interpreter. Выходим из настроек. Для запуска программы, необходимо создать профиль с конфигурацией. Для этого в верхнем правом углу нажимаем на кнопку Add Configuration. Откроется окно Run/Debug Configurations, где нажимаем на кнопку с плюсом (Add New Configuration) в правом верхнем углу и выбираем Python. Далее указываем в поле Name имя конфигурации и в поле Script path расположение Python файла с кодом программы. В завершение нажимаем на Apply, затем на ОК. Для второго способа необходимо сделать следующее: на экране приветствия в нижнем правом углу через Configure → Settings переходим в настройки. Затем переходим в раздел Project Interpreter.

В верхнем правом углу есть кнопка с шестерёнкой, нажимаем на неё и выбираем Add, создавая новое окружение. И указываем расположение для нового окружения. Нажимаем на ОК. Далее в созданном окружении устанавливаем нужные пакеты. И выходим из настроек. В окне приветствия выбираем Create New Project. В мастере создания проекта, указываем имя расположения проекта в поле Location. Разворачиваем параметры окружения, щелкая по Project Interpreter, где выбираем Existing interpreter и указываем нужное нам окружение. Далее создаем конфигурацию запуска программы, также как создавали для раннее. После чего можно выполнить программу.

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

Чтобы пользователи, которые скачивают какие-либо программы, скрипты, модули могли без проблем посмотреть, какие пакеты им нужно установить дополнительно для корректной работы. За описание о наличии каких-либо пакетов в среде как раз и отвечают файлы requirements.txt и environment.yml.

