

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ  
АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО  
ОБРАЗОВАНИЯ «СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»  
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет по лабораторной работе № 1  
«Установка пакетов в Python»**

**по дисциплине «Основы программной инженерии»**

Выполнила:  
Образцова Мария Дмитриевна,  
2 курс, группа ПИЖ-б-о-21-1,  
Проверил:  
Доцент кафедры инфокоммуникаций,  
Воронкин Р.А.

Ставрополь, 2023 г.

## Методика и порядок выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.


### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository? [Import a repository.](#)

---

Owner \*

Repository name \*

 obrMaria ▾


 / 

OPI\_2.14 ✓


Great repository names are short and memorable. Need inspiration? How about [super-lamp?](#)

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 ▾

This will set  **main** as the default branch. Change the default name in your [settings](#).

---

 You are creating a public repository in your personal account.

---

Creating repository...

3. Выполните клонирование созданного репозитория. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.

```

$ git clone https://github.com/obrMaria/OPI_2.14.git
Cloning into 'OPI_2.14'...
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.

M@DESKTOP-UVM9NOL MINGW64 ~/desktop (master)
$ cd OPI_2.14

M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_2.14 (main)
$ 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:/Users/M/desktop/OPI_2.14/.git/hooks]

M@DESKTOP-UVM9NOL MINGW64 ~/desktop/OPI_2.14 (develop)
$

```

Рисунок – клонирование созданного репозитория

#### 4. Установка виртуального окружений и работа с ними.

```

C:\Users\M>pip --version
pip 23.0 from D:\anaconda\lib\site-packages\pip (python 3.9)

```

Рисунок – проверка версии pip (проверка на наличие pip)

```
python -m venv env
```

Рисунок – Создание виртуального окружения

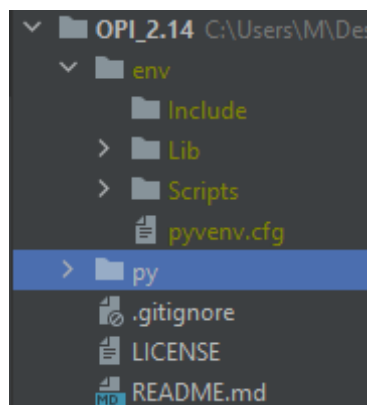


Рисунок – Созданная папка вирт окружения

```
C:\Users\M\Desktop\OPI_2.14>.\env\Scripts\activate  
(env) C:\Users\M\Desktop\OPI_2.14>
```

Рисунок – Активация вирт. окружения

```
(env) C:\Users\M\Desktop\OPI_2.14>pip install black  
Collecting black  
  Downloading black-23.1.0-cp39-cp39-win_amd64.whl (1.2 MB)  
    ----- 1.2/1.2 MB 1.2 MB/s eta 0:00:00  
Collecting click>=8.0.0  
  Downloading click-8.1.3-py3-none-any.whl (96 kB)  
    ----- 96.6/96.6 KB 1.1 MB/s eta 0:00:00  
Collecting pathspec>=0.9.0  
  Downloading pathspec-0.11.0-py3-none-any.whl (29 kB)  
Collecting platformdirs>=2  
  Downloading platformdirs-3.0.0-py3-none-any.whl (14 kB)  
Collecting packaging>=22.0  
  Downloading packaging-23.0-py3-none-any.whl (42 kB)  
    ----- 42.7/42.7 KB 514.9 kB/s eta 0:00:00  
Collecting mypy-extensions>=0.4.3  
  Downloading mypy_extensions-1.0.0-py3-none-any.whl (4.7 kB)  
Collecting typing-extensions>=3.10.0.0  
  Downloading typing_extensions-4.4.0-py3-none-any.whl (26 kB)  
Collecting tomli>=1.1.0
```

Рисунок – Установка пакета black

```
(env) C:\Users\M\Desktop\OPI_2.14>.\env\Scripts\deactivate  
C:\Users\M\Desktop\OPI_2.14>_
```

Рисунок – Деактивация вирт окружения

### вирт окр virtualenv

```
(base) PS C:\Users\M> python -m pip install virtualenv  
Collecting virtualenv  
  Downloading virtualenv-20.19.0-py3-none-any.whl (8.7 MB)  
    ----- 8.7/8.7 MB 1.1 MB/s eta 0:00:00  
Collecting distlib<1,>=0.3.6  
  Downloading distlib-0.3.6-py2.py3-none-any.whl (468 kB)  
    ----- 468.5/468.5 kB 977.2 kB/s eta 0:00:00  
Requirement already satisfied: platformdirs<4,>=2.4 in d:\anaconda\lib\site-packages (from virtualenv) (2.5.2)  
Requirement already satisfied: filelock<4,>=3.4.1 in d:\anaconda\lib\site-packages (from virtualenv) (3.6.0)  
Installing collected packages: distlib, virtualenv  
Successfully installed distlib-0.3.6 virtualenv-20.19.0
```

Рисунок – Установка вирт окр virtualenv

```
(base) PS C:\Users\M\desktop\opi_2.14> virtualenv -p python env
created virtual environment CPython3.9.13.final.0-64 in 22483ms
  creator CPython3Windows(dest=C:\Users\M\Desktop\OPI_2.14\env, clear=False, no
_vcs_ignore=False, global=False)
  seeder FromAppData(download=False, pip=bundle, setuptools=bundle, wheel=bundl
e, via=copy, app_data_dir=C:\Users\M\AppData\Local\pypa\virtualenv)
    added seed packages: black==23.1.0, click==8.1.3, colorama==0.4.6, mypy_ext
ensions==1.0.0, packaging==23.0, pathspec==0.11.0, pip==23.0, platformdirs==3.0
.0, setuptools==67.1.0, tomli==2.0.1, typing_extensions==4.4.0, wheel==0.38.4
  activators BashActivator,BatchActivator,FishActivator,NushellActivator,PowerS
hellActivator,PythonActivator
(base) PS C:\Users\M\desktop\opi_2.14> _
```

Создание вирт окр

```
PS C:\Users\M\desktop\opi_2.14> env\scripts\activate
(env) PS C:\Users\M\desktop\opi_2.14> deactivate
```

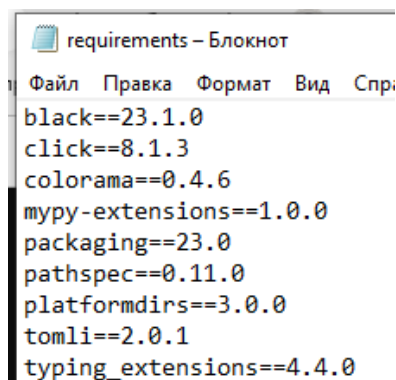
Его активизация и деактивизация

Перенос вирт окр

```
(base) (env) PS C:\Users\M\desktop\opi_2.14> pip freeze
black==23.1.0
click==8.1.3
colorama==0.4.6
mypy-extensions==1.0.0
packaging==23.0
pathspec==0.11.0
platformdirs==3.0.0
tomli==2.0.1
typing_extensions==4.4.0
(base) (env) PS C:\Users\M\desktop\opi_2.14> pip freeze > requirements.txt
(base) (env) PS C:\Users\M\desktop\opi_2.14> _
```

Рисунок – Список пакетных зависимостей

Перенаправление вывод команд в файл(сохранение)



```
requirements - Блокнот
Файл  Правка  Формат  Вид  Спр:
black==23.1.0
click==8.1.3
colorama==0.4.6
mypy-extensions==1.0.0
packaging==23.0
pathspec==0.11.0
platformdirs==3.0.0
tomli==2.0.1
typing_extensions==4.4.0
```

Рисунок – Содержимое файла

5. Создайте виртуальное окружение Anaconda с именем репозитория.

### Управление пакетами с помощью Conda

```
(base) PS C:\Users\student-09-525\desktop> mkdir %my%

Каталог: C:\Users\student-09-525\desktop

Mode                LastWriteTime         Length Name
----                -
d-----          10.02.2023         23:06         %my%

(base) PS C:\Users\student-09-525\desktop> cd %my%
(base) PS C:\Users\student-09-525\desktop\%my%> copy NUL > main.py
```

Рисунок – Создание чистого виртуального окружения с conda

```
(base) PS C:\Users\student-09-525\desktop\1> conda create -n %my% python=3.10
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
current version: 4.8.3
latest version: 23.1.0

Please update conda by running

$ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\student-09-525\.conda\envs\%my%

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
tk-8.6.12                  | h2bbff1b_0           | 3.1 MB
tzdata-2022g               | h04d1e81_0           | 114 KB
vc-14.2                    | h21ff451_1           | 8 KB
vs2015_runtime-14.27.29016 | h5e58377_2           | 1007 KB
wheel-0.37.1               | pyhd3eb1b0_0         | 33 KB
wincertstore-0.2           | py310haa95532_2      | 15 KB
xz-5.2.10                  | h8cc25b3_1           | 520 KB
zlib-1.2.13                | h8cc25b3_0           | 113 KB
```

```
(base) PS C:\Users\student-09-525\desktop\1> conda activate %my%
```

Рисунок – Его активация

```

(<%my%> PS C:\Users\student-09-525\desktop\1> conda install django, Pandas
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.8.3
  latest version: 23.1.0

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\student-09-525\.conda\envs\%my%

  added / updated specs:
    - django
    - pandas

The following packages will be downloaded:



| package               | build           |          |
|-----------------------|-----------------|----------|
| asgiref-3.5.2         | py310haa95532_0 | 39 KB    |
| bottleneck-1.3.5      | py310h9128911_0 | 106 KB   |
| django-4.1            | py310haa95532_0 | 4.2 MB   |
| intel-openmp-2021.4.0 | haa95532_3556   | 2.2 MB   |
| mkl-2021.4.0          | haa95532_640    | 114.9 MB |
| 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  |
| python-dateutil-2.8.2 | pyhd3eb1b0_0    | 233 KB   |
| python-tzdata-2021.1  | pyhd3eb1b0_0    | 137 KB   |
| pytz-2022.7           | py310haa95532_0 | 210 KB   |
| six-1.16.0            | pyhd3eb1b0_1    | 18 KB    |
| sqlparse-0.4.3        | py310haa95532_0 | 94 KB    |
| Total:                |                 | 139.2 MB |



The following NEW packages will be INSTALLED:

```

Рисунок – Установка пакетов Django и pandas

```

(<%my%> PS C:\Users\student-09-525\desktop\1> conda env export > enviroment.yml

```

Создание файла конфигурации для быстрого развертывания вирт окр

```
enviroment.yml x
1  name: '%my%'
2  channels:
3    - defaults
4  dependencies:
5    - asgiref=3.5.2=py310haa95532_0
6    - blas=1.0=mkl
7    - bottleneck=1.3.5=py310h9128911_0
8    - bzip2=1.0.8=he774522_0
9    - ca-certificates=2023.01.10=haa95532_0
10   - certifi=2022.12.7=py310haa95532_0
11   - django=4.1=py310haa95532_0
12   - intel-openmp=2021.4.0=haa95532_3556
13   - libffi=3.4.2=hd77b12b_6
14   - mkl=2021.4.0=haa95532_640
15   - mkl-service=2.4.0=py310h2bbff1b_0
16   - mkl_fft=1.3.1=py310ha0764ea_0
17   - mkl_random=1.2.2=py310h4ed8f06_0
18   - numexpr=2.8.4=py310hd213c9f_0
19   - numpy=1.23.5=py310h60c9a35_0
20   - numpy-base=1.23.5=py310h04254f7_0
21   - openssl=1.1.1s=h2bbff1b_0
22   - packaging=22.0=py310haa95532_0
23   - pandas=1.5.2=py310h4ed8f06_0
24   - pip=22.3.1=py310haa95532_0
25   - python=3.10.9=h966fe2a_0
26   - python-dateutil=2.8.2=pyhd3eb1b0_0
27   - python-tzdata=2021.1=pyhd3eb1b0_0
28   - pytz=2022.7=py310haa95532_0
29   - setuptools=65.6.3=py310haa95532_0
30   - six=1.16.0=pyhd3eb1b0_1
31   - sqlite=3.40.1=h2bbff1b_0
32   - sqlparse=0.4.3=py310haa95532_0
33   - tk=8.6.12=h2bbff1b_0
34   - tzdata=2022g=h04d1e81_0
35   - vc=14.2=h21ff451_1
```



Установите в виртуальное окружение следующие пакеты: pip, NumPy, Pandas, SciPy.

```
(%my%) PS C:\Users\student-09-525\desktop\1> conda install pip,NumPy, SciPy
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.8.3
  latest version: 23.1.0

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\student-09-525\.conda\envs\%my%

  added / updated specs:
    - numpy
    - pip
    - scipy

The following packages will be downloaded:

  package                                     build                                     12 KB
  appdirs-1.4.4                             pyhd3eb1b0_0                             335 KB
  brotli-1.0.7                              py310h2bbff1b_1002                       239 KB
  cffi-1.15.1                               py310h2bbff1b_3                           35 KB
  charset-normalizer-2.0.4                  pyhd3eb1b0_0                             1.0 MB
  cryptography-38.0.4                       py310h21b164f_0                           672 KB
  fftw-3.3.9                                h2bbff1b_1                               6.5 MB
  icc_rt-2022.1.0                           h6049295_2                               97 KB
  idna-3.4                                   py310haa95532_0                            41 KB
  pooch-1.4.0                               pyhd3eb1b0_0                            94 KB
  pycparser-2.21                            pyhd3eb1b0_0                            50 KB
  pyopenssl-22.0.0                          pyhd3eb1b0_0                            28 KB
  pysocks-1.7.1                             py310haa95532_0                           101 KB
  requests-2.28.1                           py310haa95532_0                           18.8 MB
  scipy-1.10.0                              py310hb9afe5d_0                           195 KB
  urllib3-1.26.14                           py310haa95532_0                             9 KB
  win_inet_pton-1.1.0                       py310haa95532_0

  Total:                                     28.2 MB

The following NEW packages will be INSTALLED:
```

Рисунок – Установка необходимых пакетов

Попробуйте установить менеджером пакетов **conda** пакет **TensorFlow**. Возникает ли при этом ошибка? Попробуйте выявить и укажите причину этой ошибки.

Попробуйте установить пакет **TensorFlow** с помощью менеджера пакетов **pip**. Сформируйте файлы `requirements.txt` и `environment.yml`. Проанализируйте содержимое этих файлов.

```
Executing transaction: done
(%my%) PS C:\Users\student-09-525\desktop\1> 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

==> WARNING: A newer version of conda exists. <==
current version: 4.8.3
latest version: 23.1.0

Please update conda by running

$ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\student-09-525\.conda\envs\%my%

added / updated specs:
- tensorflow

The following packages will be downloaded:
```

package	build	
_tflow_select-2.3.0	mkl	3 KB
absl-py-1.3.0	py310haa95532_0	172 KB
aiohttp-3.8.3	py310h2bbff1b_0	418 KB
aiosignal-1.2.0	pyhd3eb1b0_0	12 KB
astunparse-1.6.3	py_0	17 KB
async-timeout-4.0.2	py310haa95532_0	12 KB
attrs-22.1.0	py310haa95532_0	85 KB
blinker-1.4	py310haa95532_0	22 KB
cachetools-4.2.2	pyhd3eb1b0_0	13 KB
click-8.0.4	py310haa95532_0	157 KB
colorama-0.4.6	py310haa95532_0	32 KB
flatbuffers-2.0.0	h6c2663c_0	1.4 MB
flit-core-3.6.0	pyhd3eb1b0_0	42 KB
frozenlist-1.3.3	py310h2bbff1b_0	40 KB
gast-0.4.0	pyhd3eb1b0_0	13 KB
giflib-5.2.1	h8cc25b3_1	81 KB
google-auth-2.6.0	pyhd3eb1b0_0	83 KB
google-auth-oauthlib-0.4.4	pyhd3eb1b0_0	18 KB
google-pasta-0.2.0	pyhd3eb1b0_0	46 KB

Рисунок – Установка Tensorflow

```

(%my%) PS C:\Users\student-09-525\desktop\1> pip install TensorFlow
Requirement already satisfied: TensorFlow in c:\users\student-09-525\.conda\envs\
\my%\lib\site-packages (2.10.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (2.1.0)
Collecting protobuf<3.20,>=3.9.2
  Downloading protobuf-3.19.6-cp310-cp310-win_amd64.whl (895 kB)
----- 895.7/895.7 kB 3.2 MB/s eta 0:00:00
Requirement already satisfied: keras<2.11,>=2.10.0 in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (2.10.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (1.6.3)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (2.0)
Requirement already satisfied: packaging in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (22.0)
Requirement already satisfied: keras-preprocessing>=1.1.1 in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (1.1.2)
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 2.3 MB/s eta 0:00:00
Requirement already satisfied: tensorflow-estimator<2.11,>=2.10.0 in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (2.10.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (1.3.0)
Requirement already satisfied: setuptools in c:\users\student-09-525\.conda\envs\my%\lib\site-packages (from TensorFlow) (65.6.3)
Collecting libclang>=13.0.0
  Downloading libclang-15.0.6.1-py2.py3-none-win_amd64.whl (23.2 MB)
----- 13.9/23.2 MB 1.9 MB/s eta 0:00:06

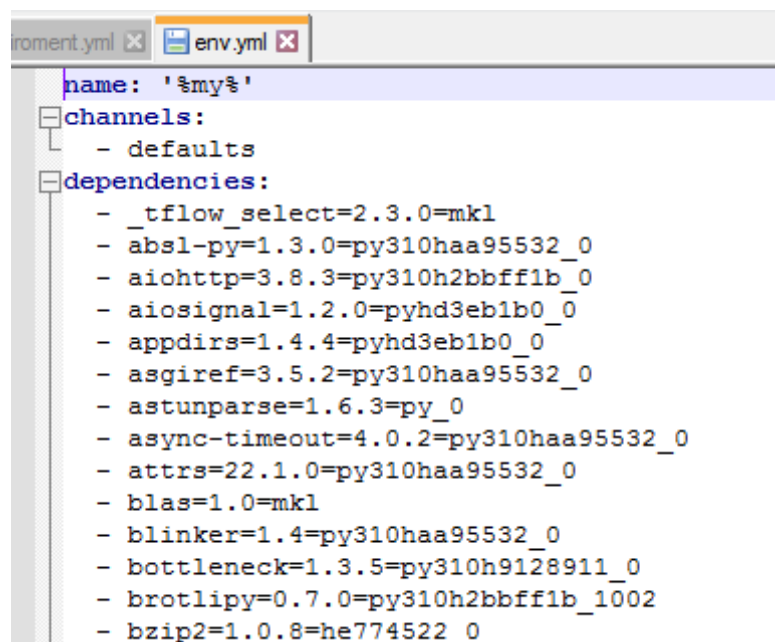
```

Рисунок – Установка tensorflow с помощью менеджера пакетов pip

```

(%my%) PS C:\Users\student-09-525\desktop\1> conda env export > env.yml
(%my%) PS C:\Users\student-09-525\desktop\1> pip freeze
(%my%) PS C:\Users\student-09-525\desktop\1> pip freeze > req.txt

```



```

name: '%my%'
channels:
  - defaults
dependencies:
  - _tflow_select=2.3.0=mk1
  - absl-py=1.3.0=py310haa95532_0
  - aiohttp=3.8.3=py310h2bbff1b_0
  - aiosignal=1.2.0=pyhd3eb1b0_0
  - appdirs=1.4.4=pyhd3eb1b0_0
  - asgiref=3.5.2=py310haa95532_0
  - astunparse=1.6.3=py_0
  - async-timeout=4.0.2=py310haa95532_0
  - attrs=22.1.0=py310haa95532_0
  - blas=1.0=mk1
  - blinker=1.4=py310haa95532_0
  - bottleneck=1.3.5=py310h9128911_0
  - brotli=0.7.0=py310h2bbff1b_1002
  - bzip2=1.0.8=he774522_0

```

Рисунок – Сформированные файлы requirements.txt и environment.yml

## ВОПРОСЫ

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

Существует Python Package Index (PyPI) – это репозиторий, открытый для всех разработчиков, в нём можно найти пакеты для решения практических задач.

**2. Как осуществить установку менеджера пакетов pip?**

Pip – это консольная утилита (без графического интерфейса). После того, как вы её скачаете и установите, она пропишется в PATH и будет доступна для использования.

Чтобы установить утилиту pip, нужно скачать скрипт get-pip.py

**3. Откуда менеджер пакетов pip по умолчанию устанавливает пакеты?**

По умолчанию в Linux Pip устанавливает пакеты в

/usr/local/lib/python2.7/dist-packages. Использование virtualenv или --user во время установки изменит это местоположение по умолчанию. Важный момент: по умолчанию pip устанавливает пакеты глобально. Это может привести к конфликтам между версиями пакетов.

```
$ pip install ProjectName
```

**4. Как установить последнюю версию пакета с помощью pip?**

**5. Как установить заданную версию пакета с помощью pip?**

```
$ pip install ProjectName==3.2
```

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

```
$ pip install -e git+https://gitrepo.com/ProjectName.git
```

```
$ pip install ./dist/ProjectName.tar.gz
```

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

8. Как удалить установленный пакет с помощью pip?

```
$ pip uninstall ProjectName
```

```
$ pip install --upgrade ProjectName
```

9. Как обновить установленный пакет с помощью pip?

10. Как отобразить список установленных пакетов с помощью pip?

```
$ pip list
```

11. Каковы причины появления виртуальных окружений в языке Python? 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.

Идея виртуального окружения родилась раньше, чем была реализована стандартными средствами Python. Попыток было несколько, но в основу PEP 405 легла утилита virtualenv Яна Бикинга. Были проанализированы возникающие при работе с ней проблемы. После этого в работу интерпретатора Python версии 3.3 добавили их решения. Так был создан встроенный в Python модуль venv, а утилита virtualenv теперь дополнительно

использует в своей работе и его.

Как работает виртуальное окружение? Ничего сверхъестественного. В отдельной папке создаётся неполная копия выбранной установки Python. Это копия является просто набором файлов (например, интерпретатора или ссылки на него), утилит для работы с собой и нескольких пакетов (в том числе pip).

Стандартные пакеты при этом не копируются.

## **12. Каковы основные этапы работы с виртуальными окружениями?**

- 1) Создаём через утилиту новое виртуальное окружение в отдельной папке для выбранной версии интерпретатора Python.
- 2) Активируем ранее созданное виртуальное окружение для работы.
- 3) Работаем в виртуальном окружении, а именно управляем пакетами используя pip и запускаем выполнение кода.
- 4) Деактивируем после окончания работы виртуальное окружение.
- 5) Удаляем папку с виртуальным окружением, если оно нам больше не нужно.

## **13. Как осуществляется работа с виртуальными окружениями с помощьюvenv?**

```
python3 -m venv <путь к папке виртуального окружения>
```

Для создания виртуального окружения достаточно дать команду в формате:

Создадим виртуальное окружение в папке проекта. Для этого перейдём в корень любого проекта

на Python  $\geq 3.3$  и дадим команду:

```
$ python3 -m venv env
```

После её выполнения создастся папка env с виртуальным окружением.

Чтобы активировать виртуальное окружение под Windows нужно дать команду:

```
> env\Scripts\activate
```

После активации приглашение консоли изменится. В его начале в круглых скобках будет отображаться имя папки с виртуальным окружением.

При размещении виртуального окружения в папке проекта стоит позаботиться об его исключении из репозитория системы управления версиями. Для этого, например, при использовании Git нужно добавить папку в файл .gitignore. Это делается для того, чтобы не засорять проект разными вариантами виртуального окружения.

```
$ python3 -m venv /home/user/envs/project1_env
```

Чтобы переключиться с одного окружения на другое нам нужно выполнить команду деактивации и команду активации другого виртуального окружения.

```
$ deactivate  
$ source /home/user/envs/project1_env2/bin/activate
```

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

```
# Для python 3  
python3 -m pip install virtualenv  
  
# Для единственного python  
python -m pip install virtualenv
```

Для начала пакет нужно установить. Установку можно выполнить командой:

Создание виртуального окружения с утилитой virtualenv отличается от стандартного. Например, создание в текущей папке виртуального окружения для интерпретатора доступного через команду python3 с названием папки окружения env:

```
virtualenv -p python3 env
```

```
> env\\Scripts\\activate
```

```
(env) > deactivate
```

Активация и деактивация такая же, как у стандартной утилиты Python.

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

Грубо говоря, `pipenv` можно рассматривать как симбиоз утилит `pip` и `venv` (или `virtualenv`), которые работают вместе, пряча многие неудобные детали от конечного пользователя.

Помимо этого `pipenv` ещё умеет вот такое:

- автоматически находить интерпретатор Python нужной версии (находит даже интерпретаторы, установленные через `pyenv` и `asdf`!);
- запускать вспомогательные скрипты для разработки;
- загружать переменные окружения из файла `.env`;
- проверять зависимости на наличие известных уязвимостей.

Стоит сразу оговориться, что если вы разрабатываете библиотеку (или что-то, что устанавливается через `pip`, и должно работать на нескольких версиях интерпретатора), то `pipenv` — не ваш путь. Этот инструмент создан в первую очередь для разработчиков конечных приложений (консольных утилит, микросервисов, веб-сервисов). Формат хранения зависимостей подразумевает работу только на одной конкретной версии интерпретатора (это имеет смысл для конечных приложений, но для библиотек это, как



правило, не приемлемо).

Для разработчиков библиотек существует другой прекрасный инструмент —poetry.

Установка на Windows, самый простой способ — это установка в домашнююдиректорию пользователя:

```
$ pip install --user pipenv Теперь проверим установку:
```

```
$ pipenv --version
```

```
pipenv, version 2018.11.26
```

Если вы получили похожий вывод, значит, всё в порядке.  
Инициализация проекта

Давайте создадим простой проект под управлением pipenv. Подготовка:

```
$ mkdir pipenv_demo
```

```
$ cd pipenv_demo
```

Создать новый проект, использующий конкретную версию Python можно воттакой командой:

```
$ pipenv --python 3.8
```

Если же вам не нужно указывать версию так конкретно, то есть шорткаты:# Создает проект с Python 3, версию выберет автоматически.

```
$ pipenv --three
```

# Аналогично с Python 2.

# В 2020 году эта опция противопоказана.

```
$ pipenv --two
```

После выполнения одной из этих команд, `pipenv` создал файл `Pipfile` и виртуальное окружение где-то в заранее определенной директории (по умолчанию вне директории проекта).

```
$ cat Pipfile [[source]] name = "pypi"
```

```
url = "https://pypi.org/simple" verify_ssl = true[dev-packages]
```

```
[packages] [requires] python_version = "3.8"
```

Это минимальный образец `Pipfile`. В секции `[[source]]` перечисляются индексы пакетов — сейчас тут только PyPI, но может быть и ваш собственный индекс пакетов. В секциях `[packages]` и `[dev-packages]` перечисляются зависимости приложения — те, которые нужны для непосредственной работы приложения (минимум), и те, которые нужны для разработки (запуск тестов, линтеры и прочее). В секции `[requires]` указана версия интерпретатора, на которой данное приложение может работать.

Если вам нужно узнать, где именно `pipenv` создал виртуальное окружение (например, для настройки IDE), то сделать это можно вот так:

```
$ pipenv --py
```

```
/Users/and-semakin/.local/share/virtualenvs/pipenv_demo-1dgGUSFy/bin/python Управление зависимостями через pipenv
```

Теперь давайте установим в проект первую зависимость. Делается это при помощи команды `pipenv install`:

```
$ pipenv install requests
```

Давайте посмотрим, что поменялось в `Pipfile` (здесь и дальше я буду сокращать вывод команд или содержимое файлов при помощи ...):

```
$ cat Pipfile
```

```
...
```

```
[packages] requests = "*"
```

```
...
```

В секцию `[packages]` добавилась зависимость `requests` с версией `*` (версия нефиксирована).

А теперь давайте установим зависимость, которая нужна для разработки, например, восхитительный линтер `flake8`, передав флаг `--dev` в ту же команду `install`:

```
$ pipenv install --dev flake8
```

```
$ cat Pipfile
```

```
...
```

```
[dev-packages] flake8 = "*"
```

...

Теперь можно увидеть всё дерево зависимостей проекта при помощи команды `pipenv graph`:

```
$ pipenv graph flake8==3.7.9
```

- entrypoints [required: >=0.3.0,<0.4.0, installed: 0.3]
- mccabe [required: >=0.6.0,<0.7.0, installed: 0.6.1]
- pycodestyle [required: >=2.5.0,<2.6.0, installed: 2.5.0]
- pyflakes [required: >=2.1.0,<2.2.0, installed: 2.1.1] requests==2.23.0
- certifi [required: >=2017.4.17, installed: 2020.4.5.1]
- chardet [required: >=3.0.2,<4, installed: 3.0.4]
- idna [required: >=2.5,<3, installed: 2.9]
- urllib3 [required: >=1.21.1,<1.26,!1.25.1,!1.25.0, installed: 1.25.9]

Это бывает полезно, чтобы узнать, что от чего зависит, или почему в вашем виртуальном окружении есть определённый пакет.

Также, пока мы устанавливали пакеты, `pipenv` создал `Pipfile.lock`, но этот файл длинный и не интересный, поэтому показывать содержимое я не буду.

Удаление и обновление зависимостей происходит при помощи команд

`pipenv uninstall` и `pipenv update` соответственно. Работают они довольно интуитивно, но если возникают вопросы, то вы всегда можете получить справку при помощи флага `--help`:

```
$ pipenv uninstall --help
```

```
$ pipenv update --help
```

Управление виртуальными окружениями

Давайте удалим созданное виртуальное окружение:

```
$ pipenv --rm
```

И представим себя в роли другого разработчика, который только присоединился к вашему проекту. Чтобы создать виртуальное окружение и установить в него зависимости нужно выполнить следующую команду:

```
$ pipenv sync --dev
```

Эта команда на основе `Pipfile.lock` воссоздаст точно то же самое виртуальное окружение, что и у других разработчиков проекта.

Если же вам не нужны dev-зависимости (например, вы разворачиваете ваш проект на продакшн), то можно не передавать флаг `--dev`:

```
$ pipenv sync
```

Чтобы "войти" внутрь виртуального окружения, нужно выполнить:

```
$ pipenv shell (pipenv_demo) $
```

В этом режиме будут доступны все установленные пакеты, а имена `python` и `pip` будут указывать на соответствующие программы внутри виртуального окружения.

Есть и другой способ запускать что-то внутри виртуального окружения безсоздания нового шелла:

```
# это запустит REPL внутри виртуального окружения
```

```
$ pipenv run python
```

```
# а вот так можно запустить какой-нибудь файл
```

```
$ pipenv run python script.py
```

```
# а так можно получить список пакетов внутри виртуального окружения
```

```
$ pipenv run pip freeze
```

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

Просмотреть список зависимостей мы можем командой: `pip freeze > requirements.txt`

Имя файла хранения зависимостей `requirements.txt` выбрано не зря. Оно является стандартной договоренностью и используется некоторыми

утилитами автоматически.

Установка пакетов из файла зависимостей в новом виртуальном окружении также выполняется одной командой:

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

Все пакеты, которые вы установили перед выполнением команды и предположительно использовали в каком-либо проекте, будут перечислены в файле с именем «requirements.txt». Кроме того, будут указаны их точные версии. Расширение: .txt

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

Можно вручную создать этот файл и наполнить его названиями и версиями нужных пакетов, а также можно использовать команду `pip freeze > requirements.txt`. Которая создаст requirements.txt наполнив его названиями и версиями тех пакетов, что используются в текущем окружении.

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

Основная проблема заключается в том, что `pip`, `easy_install` и `virtualenv` ориентированы на Python. Эти инструменты игнорируют библиотеки зависимостей, реализованные с использованием других языков. Например, XSLT, HDF5, MKL и другие, которые не имеют `setup.py` в исходном коде и не устанавливают файлы в директорию `site-packages`.

Conda же 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`).

Существуют также некоторые различия, если вы заинтересованы в создании собственных пакетов. Например, `pip` создан на основе `setuptools`,

тогда как conda использует свой собственный формат, который имеет некоторые преимущества (например, статическая компиляция пакета).

## **18. В какие дистрибутивы Python входит пакетный менеджер conda?**

Anaconda, miniconda и PyCharm.

## **19. Как создать виртуальное окружение conda?**

1. Начиная проект, создайте чистую директорию и дайте ей понятное короткое имя.

Для Windows, если используется дистрибутив Anaconda, то необходимо вначале запустить консоль Anaconda Powershell Prompt. Делается

это из системного меню, посредством выбора следующих пунктов: Пуск Anaconda3 (64-bit) Anaconda Powershell Prompt (Anaconda3). В результате будет отображено окно консоли, показанное на рисунке.

Обратите на имя виртуального окружения по умолчанию, которым в данном случае является base. В этом окне необходимо ввести следующую последовательность команд:

```
mkdir %PROJ_NAME% cd %PROJ_NAME% copy NUL > main.py
```

Здесь PROJ\_NAME - это переменная окружения, в которую записано имя проекта. Допускается не использовать переменные окружения, а использовать имя проекта вместо \$PROJ\_NAME или

```
%PROJ_NAME% .
```

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

```
conda create -n %PROJ_NAME% python=3.7 conda activate  
%PROJ_NAME%
```



Установите пакеты, необходимые для реализации проекта. `conda install django,pandas`

## 21. Как деактивировать и удалить виртуальное окружение conda?

Для Windows необходимо использовать следующую команду:

```
conda deactivate
```

Если вы хотите удалить только что созданное окружение, выполните:

```
conda remove -n $PROJ_NAME
```

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

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

Достаточно набрать:

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

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

```
conda env export > enviromant.yml
```

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

Создавайте отдельное окружение Conda и устанавливайте только нужные библиотеки для каждого проекта. PyCharm позволяет легко создавать и выбирать правильное окружение.

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

Предоставляет доступ другим пользователям к файлам.