

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

Отчет о лабораторной работе № 2 по дисциплине
«Основы программной инженерии»

Выполнил студент
2 курса, группы ПИЖ-б-о-20-1
Тотубалина С.С.

Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г.

1. Установка пакетов в Python. Виртуальные окружения

```
(base) C:\Users\admin>conda create -n lab.2.14
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.10.3
  latest version: 4.12.0

Please update conda by running

    $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\admin\anaconda3\envs\lab.2.14

Proceed ([y]/n)? y
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
#
# To activate this environment, use
#
#     $ conda activate lab.2.14
#
# To deactivate an active environment, use
#
#     $ conda deactivate
```

Рис. 1 – Создано виртуальное окружение с названием репозитория

```
(base) C:\Users\admin>conda activate lab.2.14
```

Рис. 2 – Активация виртуального окружения

```
(lab.2.14) C:\Users\admin>conda install -n lab.2.14 pip
Collecting package metadata (current_repodata.json): done
Solving environment: done

==> WARNING: A newer version of conda exists. <==
  current version: 4.10.3
  latest version: 4.12.0

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

  environment location: C:\Users\admin\anaconda3\envs\lab.2.14

  added / updated specs:
    - pip

The following packages will be downloaded:



| package                   | build          |         |
|---------------------------|----------------|---------|
| ca-certificates-2022.3.29 | haa95532_0     | 122 KB  |
| certifi-2021.10.8         | py39haa95532_2 | 152 KB  |
| openssl-1.1.1n            | h2bbff1b_0     | 4.8 MB  |
| python-3.9.12             | h6244533_0     | 17.1 MB |
| sqlite-3.38.2             | h2bbff1b_0     | 807 KB  |
| tzdata-2022a              | hda174b7_0     | 109 KB  |
| wheel-0.37.1              | pyhd3eb1b0_0   | 33 KB   |
| Total:                    |                | 23.1 MB |



The following NEW packages will be INSTALLED:



|                 |                                                         |
|-----------------|---------------------------------------------------------|
| ca-certificates | pkgs/main/win-64::ca-certificates-2022.3.29-haa95532_0  |
| certifi         | pkgs/main/win-64::certifi-2021.10.8-py39haa95532_2      |
| openssl         | pkgs/main/win-64::openssl-1.1.1n-h2bbff1b_0             |
| pip             | pkgs/main/win-64::pip-21.2.4-py39haa95532_0             |
| python          | pkgs/main/win-64::python-3.9.12-h6244533_0              |
| setuptools      | pkgs/main/win-64::setuptools-58.0.4-py39haa95532_0      |
| sqlite          | pkgs/main/win-64::sqlite-3.38.2-h2bbff1b_0              |
| tzdata          | pkgs/main/noarch::tzdata-2022a-hda174b7_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-py39haa95532_2       |



Proceed ([y]/n)? y
```

Рис. 3 – Установка пакетов на примере pip

```
Downloading and Extracting Packages
openssl-1.1.1n | 4.8 MB | ##### | 100%
wheel-0.37.1 | 33 KB | ##### | 100%
sqlite-3.38.2 | 807 KB | ##### | 100%
certifi-2021.10.8 | 152 KB | ##### | 100%
tzdata-2022a | 109 KB | ##### | 100%
ca-certificates-2022 | 122 KB | ##### | 100%
python-3.9.12 | 17.1 MB | ##### | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Рис. 4 – Установка пакетов на примере pip

```
(lab.2.14) C:\Users\admin>conda list
# packages in environment at C:\Users\admin\anaconda3\envs\lab.2.14:
#
# Name                          Version                      Build      Channel
blas                            1.0                          mkl
bottleneck                      1.3.4                        py39h080aedc_0
ca-certificates                 2022.3.29                    haa95532_0
certifi                         2021.10.8                    py39haa95532_2
icc_rt                          2019.0.0                     h0cc432a_1
intel-openmp                    2021.4.0                     haa95532_3556
mkl                             2021.4.0                     haa95532_640
mkl-service                     2.4.0                        py39h2bbff1b_0
mkl_fft                         1.3.1                        py39h277e83a_0
mkl_random                      1.2.2                        py39hf11a4ad_0
numexpr                         2.8.1                        py39hb80d3ca_0
numpy                           1.21.5                       py39ha4e8547_0
numpy-base                      1.21.5                       py39hc2deb75_0
openssl                         1.1.1n                       h2bbff1b_0
packaging                       21.3                          pyhd3eb1b0_0
pandas                          1.4.1                        py39hd77b12b_1
pip                             21.2.4                       py39haa95532_0
pyparsing                       3.0.4                        pyhd3eb1b0_0
python                          3.9.12                       h6244533_0
python-dateutil                 2.8.2                        pyhd3eb1b0_0
pytz                            2021.3                       pyhd3eb1b0_0
scipy                           1.7.3                        py39h0a974cb_0
setuptools                      58.0.4                       py39haa95532_0
six                             1.16.0                       pyhd3eb1b0_1
sqlite                          3.38.2                       h2bbff1b_0
tzdata                          2022a                        hda174b7_0
vc                              14.2                         h21ff451_1
vs2015_runtime                  14.27.29016                  h5e58377_2
wheel                           0.37.1                       pyhd3eb1b0_0
wincertstore                     0.2                          py39haa95532_2
```

Рис. 5 – Список пакетов после установки

```
(lab.2.14) C:\Users\admin>conda install -n lab.2.14 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: 4.10.3
  latest version: 4.12.0

Please update conda by running

  $ conda update -n base -c defaults conda

## Package Plan ##

environment location: C:\Users\admin\anaconda3\envs\lab.2.14

added / updated specs:
- tensorflow

The following packages will be downloaded:
```

package	build	
-----	-----	
_tflow_select-2.3.0	mk1	3 KB
abseil-cpp-20210324.2	hd77b12b_0	1.6 MB
absl-py-0.15.0	pyhd3eb1b0_0	103 KB
aiohttp-3.8.1	py39h2bbff1b_1	496 KB
aiohttp-3.8.1	pyhd3eb1b0_0	12 KB
aiosignal-1.2.0	py39h2bbff1b_1	47 KB
astor-0.8.1	py39h2bbff1b_1	17 KB
astunparse-1.6.3	py_0	10 KB
async-timeout-4.0.1	pyhd3eb1b0_0	51 KB
attrs-21.4.0	pyhd3eb1b0_0	23 KB
blinker-1.4	py39h2bbff1b_1	13 KB
cachetools-4.2.2	py39h2bbff1b_1	224 KB
cffi-1.15.0	py39h2bbff1b_1	155 KB
click-8.0.4	py39h2bbff1b_1	1.4 MB
flatbuffers-2.0.0	h6c2663c_0	77 KB
frozenset-1.2.0	py39h2bbff1b_1	13 KB
gast-0.4.0	pyhd3eb1b0_0	83 KB
google-auth-2.6.0	pyhd3eb1b0_0	20 KB
google-auth-oauthlib-0.4.1	py_2	46 KB
google-pasta-0.2.0	pyhd3eb1b0_0	1.9 MB
grpcio-1.42.0	py39hc60d5dd_0	879 KB
h5py-3.6.0	py39h3de5c98_0	11.0 MB
icu-68.1	h6c2663c_0	49 KB
idna-3.3	pyhd3eb1b0_0	41 KB
importlib-metadata-4.11.3	py39h2bbff1b_1	35 KB
keras-preprocessing-1.1.2	pyhd3eb1b0_0	295 KB
libcurl-7.80.0	h86230a5_0	1.9 MB
libprotobuf-3.14.0	h23ce68f_0	144 KB
markdown-3.3.4	py39h2bbff1b_1	

Рис. 6 – Ошибка при установке пакета TensorFlow отсутствует

```
tensorflow 2.6.0 mkl_py39h31650da_0
```

Рис. 7 – Версия tensorflow в списке пакетов

```
(lab.2.14) C:\Users\admin>python
Python 3.9.12 (main, Apr 4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
```

Рис. 8 – Запуск Python в окружении и импорт пакета tensorflow

```
>>> print(tensorflow.__version__)
2.6.0
```

Рис. 9 – Версия импортированного tensorflow

```
environment.yml
1 name: lab.2.14
2 channels:
3   - defaults
4 dependencies:
5   - _tflow_select=2.3.0=mk1
6   - abseil-cpp=20210324.2=hd77b12b_0
7   - absl-py=0.15.0=pyhd3eb1b0_0
8   - aiohttp=3.8.1=py39h2bbff1b_1
9   - aiosignal=1.2.0=pyhd3eb1b0_0
10  - astor=0.8.1=py39haa95532_0
11  - astunparse=1.6.3=py_0
12  - async-timeout=4.0.1=pyhd3eb1b0_0
13  - attrs=21.4.0=pyhd3eb1b0_0
14  - blas=1.0=mk1
15  - blinker=1.4=py39haa95532_0
16  - bottleneck=1.3.4=py39h080aedc_0
17  - brotliipy=0.7.0=py39h2bbff1b_1003
18  - ca-certificates=2022.3.29=haa95532_0
19  - cachetools=4.2.2=pyhd3eb1b0_0
20  - certifi=2021.10.8=py39haa95532_2
21  - cffi=1.15.0=py39h2bbff1b_1
22  - charset-normalizer=2.0.4=pyhd3eb1b0_0
23  - click=8.0.4=py39haa95532_0
24  - colorama=0.4.4=pyhd3eb1b0_0
25  - cryptography=3.4.8=py39h71e12ea_0
26  - dataclasses=0.8=pyh6d0b6a4_7
27  - flatbuffers=2.0.0=h6c2663c_0
28  - frozenlist=1.2.0=py39h2bbff1b_0
29  - gast=0.4.0=pyhd3eb1b0_0
30  - giflib=5.2.1=h62dcd97_0
31  - google-auth=2.6.0=pyhd3eb1b0_0
32  - google-auth-oauthlib=0.4.1=py_2
33  - google-pasta=0.2.0=pyhd3eb1b0_0
34  - grpcio=1.42.0=py39hc60d5dd_0
35  - h5py=3.6.0=py39h3de5c98_0
36  - hdf5=1.10.6=h7ebc959_0
37  - icc_rt=2019.0.0=h0cc432a_1
38  - icu=68.1=h6c2663c_0
39  - idna=3.3=pyhd3eb1b0_0
40  - importlib-metadata=4.11.3=py39haa95532_0
41  - intel-openmp=2021.4.0=haa95532_3556
42  - jpeg=9d=h2bbff1b_0
43  - keras-preprocessing=1.1.2=pyhd3eb1b0_0
44  - libcurl=7.80.0=h86230a5_0
45  - libpng=1.6.37=h2a8f88b_0
46  - libprotobuf=3.14.0=h23ce68f_0
47  - libssh2=1.9.0=h7aldbcl_1
48  - markdown=3.3.4=py39haa95532_0
49  - mkl=2021.4.0=haa95532_640
50  - mkl-service=2.4.0=py39h2bbff1b_0
51  - mkl_fft=1.3.1=py39h277e83a_0
52  - mkl_random=1.2.2=py39hf11a4ad_0
53  - multidict=5.1.0=py39h2bbff1b_2
54  - numexpr=2.8.1=py39hb80d3ca_0
55  - numpy=1.21.5=py39ha4e8547_0
56  - numpy-base=1.21.5=py39hc2deb75_0
57  - oauthlib=3.2.0=pyhd3eb1b0_0
58  - openssl=1.1.1n=h2bbff1b_0
59  - opt_einsum=3.3.0=pyhd3eb1b0_1
60  - packaging=21.3=pyhd3eb1b0_0
```

Рис. 10 – Содержимое файла environment.yml

Ответы на вопросы:

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. Там также есть \ возможность выкладывать свои пакеты. Для скачивания и установки используется специальная утилита, которая называется `pip`.

2. Как осуществить установку менеджера пакетов `pip`?

При развертывании современной версии Python (начиная с Python 2.7.9 и Python 3.4), `pip` устанавливается автоматически. Но если, по какой-то причине, `pip` не установлен на вашем ПК, то сделать это можно вручную.

Существует несколько способов.

1. Универсальный способ

Будем считать, что Python у вас уже установлен, теперь необходимо установить `pip`. Для того, чтобы это сделать, скачайте скрипт `get-pip.py`

```
$ curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
```

```
$ python get-pip.py
```

При этом, вместе с `pip` будут установлены `setuptools` и `wheels`. `Setuptools` – это набор инструментов для построения пакетов Python. `Wheels` – это формат дистрибутива для пакета Python.

2. Способ для Linux

Если вы используете Linux, то для установки `pip` можно воспользоваться имеющимся в вашем дистрибутиве пакетным менеджером. Ниже будут перечислены команды для ряда Linux систем, запускающие установку `pip` (будем рассматривать только Python 3, т.к. Python 2 уже морально устарел).

Fedora

```
$ sudo dnf install python3 python3-wheel openSUSE
```

```
$ sudo zypper install python3-pip python3-setuptools python3-wheel
```

Debian/Ubuntu

```
$ sudo apt install python3-venv python3-pip Arch Linux
```

```
$ sudo pacman -S python-pip
```

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

По умолчанию это Python Package Index (PyPI) – репозиторий, открытый для всех Python разработчиков.

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

```
$ pip install ProjectName
```

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

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

```
$ pip install ProjectName>=3.1
```

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

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

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

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

8. Как удалить установленный пакет с помощью `pip`?

```
$ pip uninstall ProjectName
```

9. Как обновить установленный пакет с помощью `pip`?

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

10. Как отобразить список установленных пакетов с помощью `pip`?

```
$ pip list
```

```
$ pip show ProjectName
```

11. Каковы причины появления виртуальных окружений в языке Python?

В системе для интерпретатора Python может быть установлена глобально только одна версия пакета. Это порождает ряд проблем:

1. 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. Чем опасно обновление пакетов или версий интерпретатора? В новой версии пакета могут измениться названия функций или методов объектов и число и/или

порядок передаваемых в них параметров. В следующей версии интерпретатора могут появиться новые ключевые слова, которые совпадают с именами переменных уже существующих приложений.

Эта же проблема затрагивает и процесс разработки. Обычно программист работает со множеством проектов, так как приходится поддерживать созданные ранее и не редко даже унаследованные от других: когда-то веб приложения создавались для одной версии фреймворка, а сегодня уже вышла новая, но переписывать все долго и дорого, поэтому проект и дальше поддерживается для старой.

2. 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 работы с виртуальным окружением:

1. Создаём через утилиту новое виртуальное окружение в отдельной папке для выбранной версии интерпретатора Python.
2. Активируем ранее созданное виртуального окружения 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 помощью `venv`?

Для создания виртуального окружения достаточно дать команду в формате: `python3 -m venv <путь к папке виртуального окружения>`

Создадим виртуальное окружение в папке проекта. Для этого перейдём в корень любого проекта на Python `>= 3.3` и дадим команду:

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

После её выполнения создастся папка `env` с виртуальным окружением. Чтобы активировать виртуальное окружение под Windows команда выглядит иначе: `> env\\Scripts\\activate`

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.

Чтобы переключиться с одного окружения на другое нам нужно выполнить команду деактивации и команду активации другого виртуального окружения, например, так:

```
$ deactivate
```

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

Команда `deactivate` всегда выполняется из контекста текущего виртуального окружения. По этой причине для неё не нужно указывать полный путь. Ранее мы работали с интерпретаторами доступными через стандартные вызовы `python`. Если в системе установлена другая версия интерпретатора, которая не добавлена в переменную `PATH`, то для создания виртуального окружения нужно явно задать путь до исполняемого файла интерпретатора. Например, у нас Windows и ранее мы установили Python версии 3.7.6. Сегодня мы решили ознакомиться с особенностями Python версии 3.8.2, так как нужно расти в качестве разработчика. Для этого скачали и установили новую версию в папку `C:\\Python38`, но при этом не добавили в `PATH` (не установили соответствующую галочку в программе установки).

Для создания виртуального окружения для Python 3.8.2 нам придется дать в папке проекта команду с указанием полного пути до интерпретатора, например: `> C:\\Python38\\python -m venv env`

После выполнения команды мы получим виртуальное окружение для Python 3.8.2. Активация же будет происходить стандартно для Windows:

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

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

Для начала пакет нужно установить. Установку можно выполнить командой:

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

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`

Активация и деактивация: `> env\\Scripts\\activate (env) > deactivate`

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 `ryenv` 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 домашнюю директорию 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 вот такой командой:

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

Если же вам не нужно указывать версию так конкретно, то есть шорткаты:

```
$ pipenv --three
```

После выполнения 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 не фиксирована).

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 без создания нового шелла:

```
$ pipenv run python
```

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

```
$ 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 так же 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

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%
```

21. Как деактивировать и удалить виртуальное окружение conda?

Деактивация - `conda deactivate`
`conda remove -n $PROJ_NAME`

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

Файл environment.yml позволит воссоздать окружение в любой нужный момент. `conda env export > environment.yml`

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

```
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.

Метод первый:

Используйте анаконду для создания новой среды

1. Убедитесь, что Anaconda или Miniconda загружена и установлена на вашем компьютере, и вам известен путь к ее исполняемому файлу.

Более подробную информацию см. в инструкции по установке.

1. Выполните одно из следующих действий:

- Добавьте новый интерпретатор.

- Нажмите **Ctrl+Alt+S**, чтобы открыть проект **Настройки/Преференции** и перейдите в раздел **Проект <имя проекта> | Python Interpreter**. Затем щелкните значок и выберите «Добавить».

В левой части диалогового окна **Добавить интерпретатор Python** выберите **Среда Conda**. Следующие действия зависят от того, существовала ли среда Conda ранее.

Если выбрано Новое окружение:

1. Укажите расположение новой среды Conda в текстовом поле или щелкните и найдите расположение в вашей файловой системе. Обратите внимание, что каталог, в котором должно быть расположено новое окружение Conda, должен быть пустым!

2. Выберите версию Python из списка.

3. Укажите расположение исполняемого файла Conda в текстовом поле, или нажмите и найдите расположение в каталоге установки Conda. В основном вы ищете путь, который вы использовали при установке Conda на вашу машину.

4. Установите флажок Сделать доступным для всех проектов, если вы хотите повторно использовать эту среду при создании интерпретаторов Python в PyCharm.

Если выбрано существующее окружение:

1. Раскройте список Интерпретатор и выберите любой из существующих интерпретаторов. Или нажмите и укажите путь к исполняемому файлу Conda в вашей файловой системе, например, C:\Users\jetbrains\Anaconda3\python.exe.

2. Установите флажок Сделать доступным для всех проектов, если вы хотите повторно использовать эту среду при создании интерпретаторов Python в PyCharm.

Нажмите ОК, чтобы завершить задачу.

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

Эти файлы важны так же, как и сам код. Не рекомендуется загружать всё виртуальное окружение проекта в репозиторий, так как всё окружение можно восстановить на другом компьютере благодаря этим файлам.