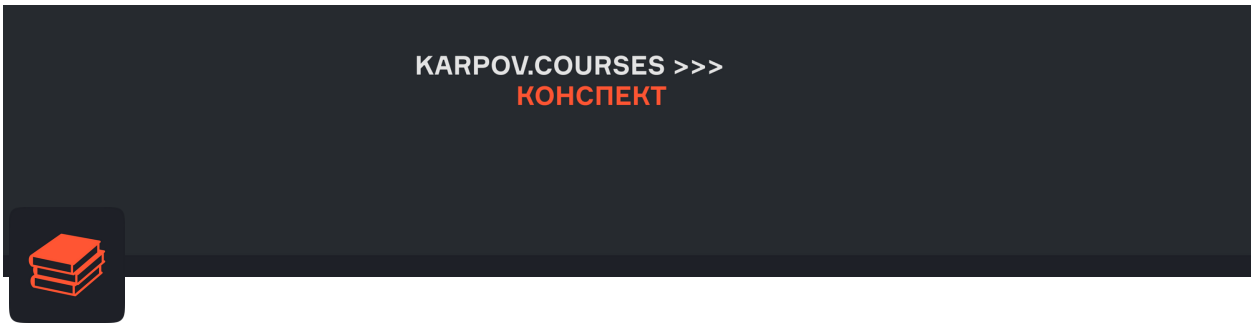


> Конспект > 3 урок > Внешние модули. Экосистема PyPi. Установка пакетов в виртуальные окружения

>Оглавление 3 урока

>Библиотеки

Библиотека(модуль, пакет) это набор готовых функций и переменных, которые можно использовать в своем коде, используются, дабы облегчить работу. Они

бывают встроенными и дополнительными. Библиотеки также могут называться пакетами

Чтобы использовать библиотеки, необходимо их импортировать. Синтаксис несложный:

```
import datetime as dt
#import - ключевое слово, дающее python понять, что мы ипортируем библиотеку
#datetime - сама библиотека
#as - ключевое слово, означающее, что далее мы будем использовать alias
#dt - псевдоним для конкретной библиотеки(alias)
```

Более простой вариант импорта без “псевдонима”:

```
import json
#import - ключевое слово, дающее python понять, что мы ипортируем библиотеку
#datetime - сама библиотека
```

Также, возможно импортировать определенные атрибуты библиотеки, и не импортировать всю библиотеку целиком. Синтаксис тоже не сложный:

```
from collections import defaultdict, Counter
#import - ключевое слово, дающее python понять, что мы ипортируем определенные атрибуты
#collections - сама библиотека
#import - ключевое слово, дающее python понять, что мы собираемся импортировать
#defaultdict, Counter - те самые объекты, которые мы хотим импортировать

#Далее не нужно обозначать все названиями модуля
```

>Библиотека datetime

Библиотека `datetime` используется для работы со временем и датами.

Теперь можно обращаться к библиотеке через её название и точку

```
christmas_day = dt.date(year=2022, month=1, day=7)
print(christmas_day)

2022-01-07
```

В datetime можно написать точную дату и время, с точностью до секунды. для этого надо вызвать `datetime()` из библиотеки datetime(она у нас под элиасом dt), синтаксис такой:

```
datetime.datetime(year, month, day, hour, minute, second)
```

```
some_time = dt.datetime(2022, 3, 4, 13, 55, 34)
print(some_time)
print(f'Год: {some_time.year}')
print(f'Месяц: {some_time.month}')
print(f'День: {some_time.day}')
print(f'Час: {some_time.hour}')
print(f'Минута: {some_time.minute}')
print(f'Секунда: {some_time.second}')

2022-03-04 13:55:34
Год 2022
Месяц 3
День 4
Час 13
Минута 55
Секунда 34
```

Библиотека datetime даёт нам возможность проводить арифметические вычисления над датами. `timedelta()` представляет собой разницу между двумя датами или временем.

```
delta = dt.timedelta(days=0,
                     seconds=0,
                     microseconds=0,
                     milliseconds=0,
                     minutes=0,
                     hours=0,
                     weeks=0)
```

Аргументы необязательны, а их значения по умолчанию равны 0. Также, аргументы могут быть целыми числами(int), числами с плавающей точкой(float), положительными или отрицательными.

```
delta_1 = dt.timedelta(days=60)
#Прибавим 60 дней к дате 2021-03-12
print(dt.date(2021, 3, 12) + delta_1)

2021-05-11
```

Подробнее о datetime

>Библиотека collections

Модуль collections в Python содержит большой набор специализированных типов данных, которые могут быть использованы для замены встроенных типов данных Python, таких как dict, tuple, list, set.

```
from collections import defaultdict, Counter
```

`defaultdict()` - это обычный словарь, но при обращении к несуществующему элементу, он не выбрасывает ошибку, а создает его.

```
d = defaultdict(int) #При отсутствии элемента создаётся 0 (явно передаем название типа)
print(d)
print(d["this key does not exist"])
print(d)

defaultdict(<class 'int'>, {})
0
defaultdict(<class 'int'>, {'this key does not exist': 0}) #новый ключик!
```

Как мы видим, при обращении к элементу "this key does not exist" он создался в словаре со значением int(). `defaultdict` автоматически назначает значение ноль любому ключу, который вызван, но еще не имеет значения.

Подсчёт элементов последовательности (списка, кортежа, символов, строки и т.д) одна из распространенных задач в программировании. Часто количество элементов в последовательности считают с помощью цикла:

```
array = [1, 3, 'a', 'm', 1, None, 3, 3, ()]
```

```
c = {}
for i in array:
    c[i] = array.count(i)
print(c)

{1: 2, 3: 3, 'a': 1, 'm': 1, None: 1, (): 1
```

(`count()` — подсчитывает количество экземпляров элемента в списке)

`Counter()` - предназначен для подсчетов количества элементов в последовательности без использования цикла. `Counter()` принимает последовательность и возвращает словарь, где значения - это количества повторений элемента в последовательности, а ключи - сами элементы.

```
array = [1, 3, 'a', 'm', 1, None, 3, 3, ()]
c = Counter(array)
print(c)

Counter({3: 3, 1: 2, 'a': 1, 'm': 1, None: 1, (): 1})
```

Как мы видим, число 3 встретилось три раза, число 1 встретилось два раза, а остальные элементы по одному разу.

Обратите внимание, что Counter это такой же словарь, как и dict, а значит, ключами в словаре могут быть только неизменяемые объекты, поэтому на вход в Counter() могут поступать только списки с неизменяемыми объектами.

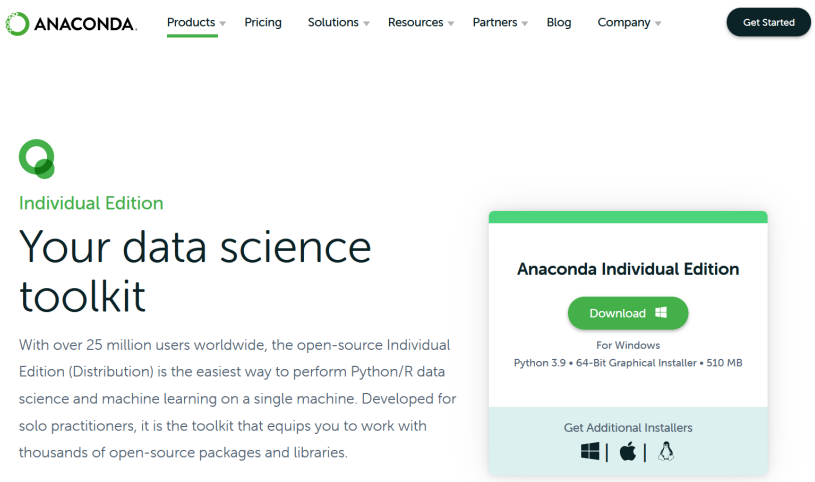
Подробнее о collections

>Windows 10: установка Anaconda

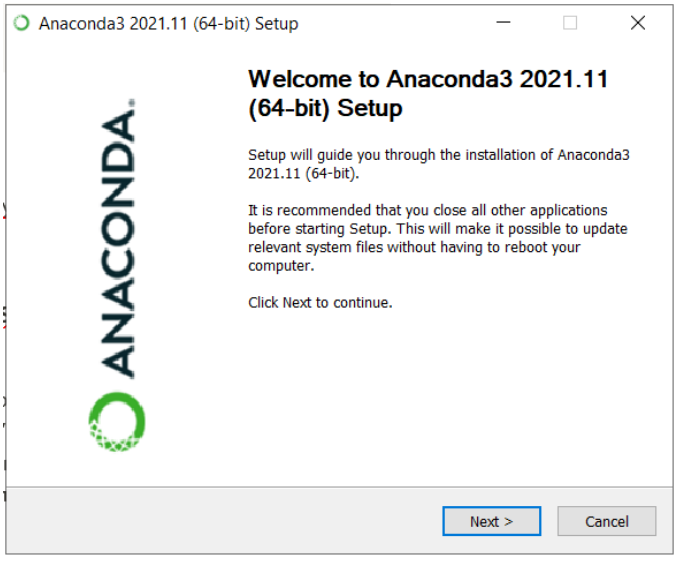
Anaconda — дистрибутив языков программирования Python и R, включающий набор популярных свободных библиотек

1. Перейдите на Anaconda.com
2. Перед скачиванием убедитесь в правильности разрядности. Чтобы посмотреть разрядность своей системы нажмите кнопку “**Пуск”**, затем выберите **Параметры,** затем **система** . Потом, в левом меню выберите пункт “**О программе”**. И справа, в разделе **Характеристики устройства**, посмотрите, какой **Тип системы** указан: 64-разрядный или 34-разрядный

3. Скачайте установщик Anaconda в соответствии с разрядностью вашей системы нажав на зеленую кнопку “Download”

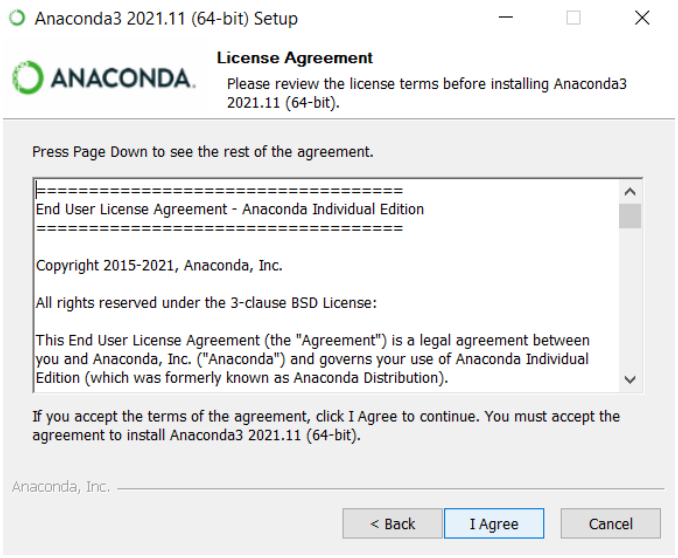


4. Кликните два раза по файлу с установочной программой.
5. Нажмите “Next” (Далее)

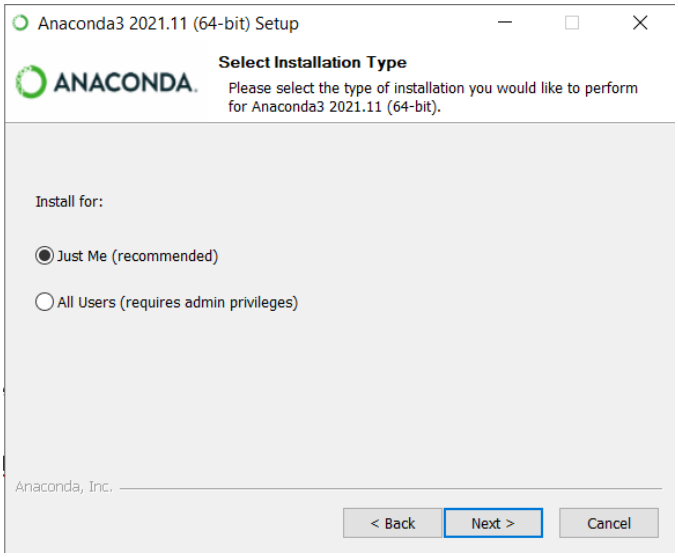


Если при установке появляются проблемы, попробуйте отключить антивирусное ПО на время установки Anaconda

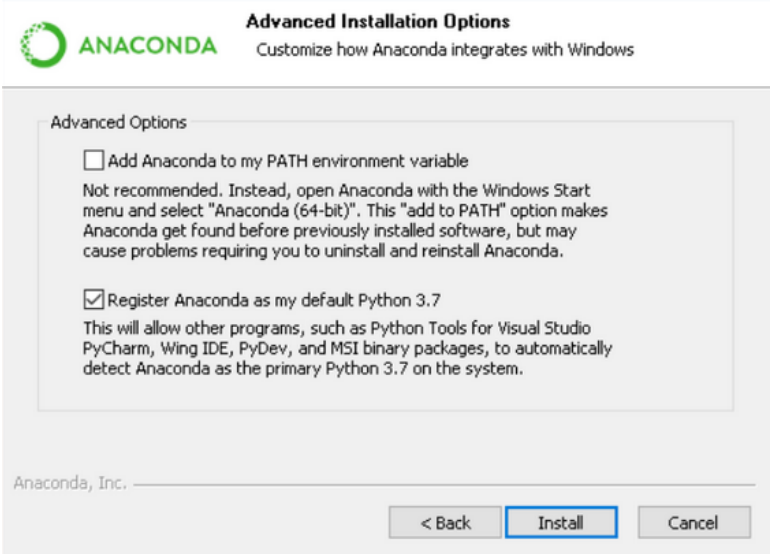
6. Как прочтете лицензионное соглашение нажмите “I agree” (Согласен).



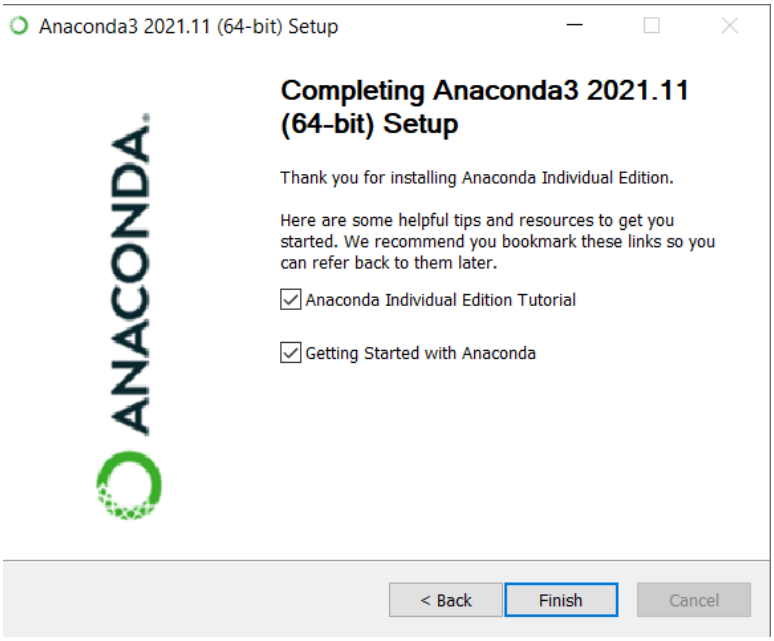
7. Выберите “Just Me” (Только я), если вы не устанавливаете программу для всех пользователей, нажмите “Next” (Далее).



8. Выберите папку установки и нажмите “Next” (Далее)
9. Выберите, нужно ли добавлять Anaconda в переменную окружения PATH. Сама Anaconda рекомендует этого не делать, потому что это может повлиять на работу других программ. Выбираем пункт 2 “Register Anaconda as my default Python 3.7”



10. Нажмите “Install” (Установить)
11. Загрузка займёт пару минут
12. Нажмите “Next” (Далее)
13. Для установки Anaconda без PyCharm нажмите кнопку “Next” (Далее)(PyCharm это такая среда разработки для Python)
14. Если хотите почитать об Anaconda Cloud и почитать туториал Anaconda, отметьте пункты «Learn more about Anaconda Cloud» и «Learn how to get started with Anaconda». Нажмите кнопку “Finish” (Завершить).



15. Чтобы проверить, успешно ли мы установили Anaconda, откройте консоль. Нажмите кнопку “**Пуск”,** найдите “Anaconda Powershell Prompt” и откройте его. В открывшейся консоли введите “python”(без кавычек).
16. Если Anaconda установлена правильно, то у вас откроется интерпретатор python

>Linux: Установка Anaconda

Anaconda — дистрибутив языков программирования Python и R, включающий набор популярных свободных библиотек

1. Перейдите на Anaconda.com.
2. Листаем вниз страницы, находим установщик Linux и сохраняем файл “Linux 64-Bit (x86) Installer”.
3. Открываем терминал нажав Ctrl+Alt+T.
4. Вводим в консоль команду bash Downloads/Anaconda3-2021.11-Linux-x86_64.
5. Читаем лицензионное соглашение(нажмите Q чтобы выйти из режима чтения) и принимаем его(пишем “yes” без кавычек).

6. Выбираем путь установки и жмём Enter.
7. На вопрос “Do you wish installer to initialize Anaconda3 by running Conda init?”(Вы хотите, чтобы установщик инициализировал Anaconda3, запустив Conda init?) пишем “yes” без кавычек.
8. Открываем новый терминал(см.п.3)
9. Если вы видите фразу “(base)” в начале строки, то Anaconda установлена правильно.

>MacOS: установка Anaconda

Anaconda — дистрибутив языков программирования Python и R, включающий набор популярных свободных библиотек

1. Перейдите на Anaconda.com.
2. Листаем вниз страницы, находим установщик MacOS и загружаем “64-Bit Command Line Installer”.
3. Находим терминал с помощью Spotlight, открываем его
4. Вводим в консоль команду bash Downloads/Anaconda3-2021.11-MacOSX-x86_64.sh
5. Читаем лицензионное соглашение(нажмите Q чтобы выйти из режима чтения) и принимаем его(пишем “yes” без кавычек).
6. Выбираем путь установки и жмём Enter.
7. На вопрос “Do you wish installer to initialize Anaconda3 by running Conda init?”(Вы хотите, чтобы установщик инициализировал Anaconda3, запустив Conda init?) пишем “yes” без кавычек.
8. Перезапускаем консоль
9. Вводим в консоль команду “which python”, которая покажет где сейчас установлен python

>Командная строка

Консоль — окно, где мы можем печатать команды, которые будут выполнены компьютером. Она широко используется для повседневной работы в системе, т.е работа в ней аналогична работе на рабочем столе. У командной строки есть главный минус - **высокий порог входа**, работать с терминалом значительно сложнее, чем с графическим интерфейсом. Но есть и плюсы:

- возможности автоматизации,
- меньшее потребление ресурсов,
- непосредственная возможность управлять файлами и данными(копирование, удаление, перемещение и т. д.)

Базовые команды консоли:

- pwd (print working directory) — выводит название директории в которой вы находитесь
- ls (list) — выводит файлы и директории в указанной папке
- cd (change directory) – изменяет директорию
- cat (concatenate) — выдаёт содержимое файла

Pip (система управления пакетами) — Это утилита командной строки, которая позволяет устанавливать, переустанавливать и удалять пакеты python

Установить пакет можно путём короткой команды в консоли:

```
pip install название_библиотеки
```

Иногда библиотека может внутри себя использовать другие сторонние библиотеки. Таким образом получается целая цепочка зависимостей. Обратите внимание, что с установкой **Anaconda** мы также установили много пакетов(чтобы посмотреть список пакетов установленных в активной среде, введите в консоль команду `pip list`)

Открыть **Jupyter Notebook** можно написав `jupyter notebook` в консоли. Локальный **Jupyter Notebook** полезен тем, что теперь возможно полностью контролировать какие библиотеки будут в **Jupyter Notebook**

Иногда нужно поставить конкретную версию библиотеки:

```
pip install название_библиотеки==0.0.7
```

>requirements.txt

В исходниках некоторых проектов на Python можно встретить текстовый файл requirements.txt. Это обычный текстовый файл, где указаны названия python библиотек, которые использовались в проекте, и их версии.

Вот пример такого файла:

```
numpy
scipy==2.23.0
pandas
loguru
```

Команда `pip install` умеет читать requirements.txt и устанавливать все зависимости из него, если передать специальный флаг `-r`:

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

То есть, если файл requirements.txt будет иметь такое же содержание, как в примере выше, то обе эти команды будут иметь одинаковое действие:

```
pip install -r requirements.txt

pip install numpy scipy==1.7.3 pandas loguru
```

PYPI(Python Package Index, пайпиай) — каталог пакетов Python

>Виртуальное окружение

Виртуальное окружение(среда) - это изолированное окружение среды, которое позволяет нам использовать определенные версии приложений.

Представим, ты сейчас работаешь над несколькими проектами одновременно, и для каждого из них есть свои определенные особености версий пакетов. При этом версии пакетов менять ну никак нельзя - придется же много чего переделывать и переписывать. Вот для таких ситуаций нам как раз и пригодится виртуальное окружение, позволяющее использовать разные версии пакетов.

Создаётся окружение командой

```
python -m venv имя_окружения
```

Через опцию `-m` передается имя модуля, в данном случае это `venv`.

Далее, активация окружение командой(windows)

```
.\имя_окружения\Scripts\activate
```

Далее, активация окружение командой(Linux и MacOS)

```
source имя_окружения/bin/activate
```

Все последующие команды будет выполнять не системный python, а python в виртуальном окружении. Python использует пакеты и настройки только активной среды.

Чтобы выйти из виртуального окружение нужно ввести команду

```
deactivate
```

Теперь вы вернулись в контекст “*system*”, а команда python ссылается на общую установку Python. Помните: это можно делать когда угодно, после закрытия виртуальной среды.

Будьте внимательны, что при вызове `jupyter notebook` Ноутбук будет открываться только с помощью системного python. Чтобы это обойти придётся запусть `jupyter notebook` как модуль(перед этим его установив, конечно же):

```
python -m jupyter notebook
```