



Python и окружение для ML



Немного о Python

- Интерпретируемый язык.
- Модульный язык, каждый аспект которого можно кастомизировать.
- Можно использовать разные парадигмы: процедурную, ООП, функциональную.
- Читаемость кода важнее производительности.
- Высокой производительности можно достичь, написав библиотеку на C с биндингами в Python.



Python 2 vs Python 3

- Python 3 исправил проблемы в дизайне языка и сделал ядровые механизмы более логичными.
- Все новые проекты начинайте писать на Python 3.
- Поддержка Python 2 закончится 31 декабря 2019 г.
- Большинство крупных проектов перестанут поддерживать Python 2 в 2020 г.: <https://python3statement.org>
- Если вы столкнётесь с legacy, всегда проще дополнительно изучить нюансы Python 2, чем изучать сначала Python 2, а потом Python 3.



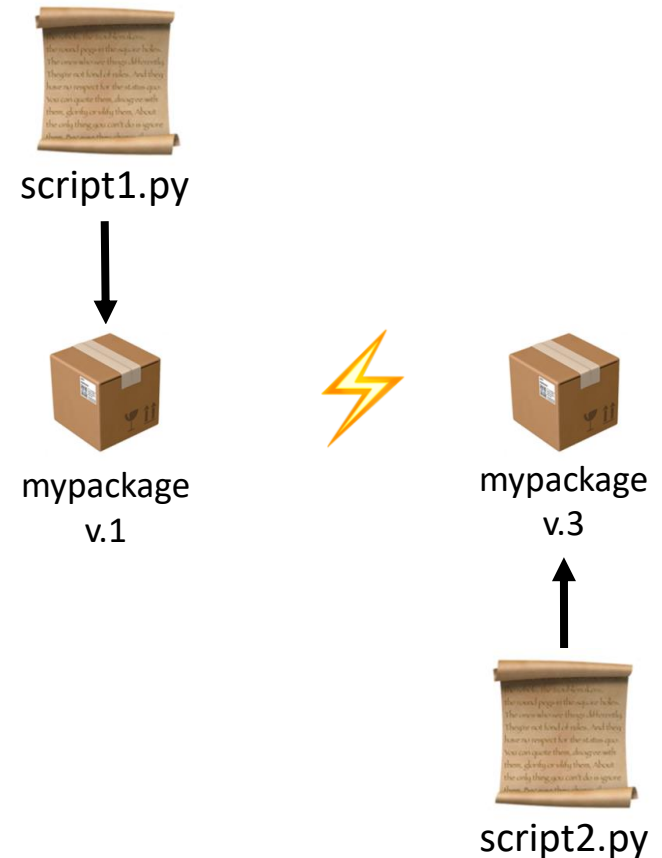
Почему Python, а не _____?

- Основная задача специалиста ML — быстро проверять гипотезы.
- У Python самая богатая экосистема для решения задач ML. Это позволяет максимально быстро проверять гипотезы и не изобретать велосипед.
- Python — действительно не лучший язык, чтобы писать на нём сложную систему для прода.
- Можно самому реализовать какой-нибудь алгоритм на любом языке — это отличный способ научиться новому.



pip и конфликт версий

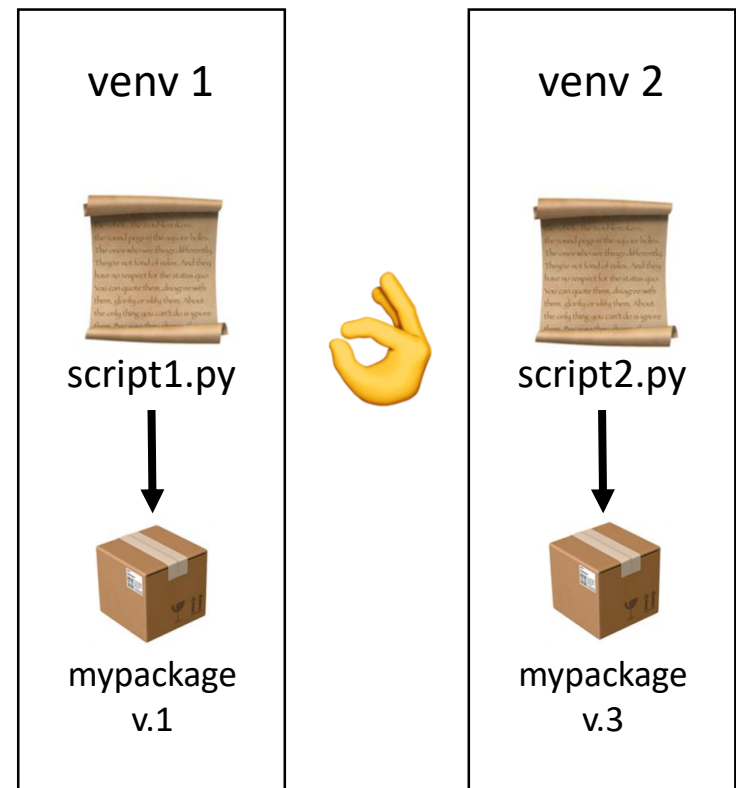
- Стандартный способ установки дополнительных библиотек — команда `pip install`. Библиотеки хранятся в центральном репозитории: <https://pypi.org>.
- Разные проекты могут использовать разные версии одной и той же библиотеки. Если более новая версия ломает обратную совместимость, у нас возникнет неразрешимая проблема.





Виртуальные окружения

- На помощь приходят виртуальные окружения — изолированные среды, в каждой из которых установлена своя версия Python и свой набор библиотек.
- Python видит только те библиотеки, которые установлены в его виртуальном окружении.
- С самого начала нужно привыкнуть к тому, что для каждого проекта создаётся новое независимое окружение.





Anaconda

- [Anaconda](#) — дистрибутив Python, система управления пакетами, средство для создания виртуальных сред.
- В отличие от `pip`, пакеты `conda` содержат и системные зависимости. Например, при установке `numpy` из `conda` также ставится библиотека `OpenBLAS`, которая значительно ускоряет матричные операции. При установке через `pip` за этим нужно следить вручную.
- Anaconda позволяет создавать свои виртуальные окружения. В этом курсе будем создавать окружения через `conda create` и ставить пакеты через `conda install`.



Пакеты conda и pip

`conda install numpy`



numpy



OpenBLAS



`pip install numpy`



numpy





Репозиторий [conda-forge](https://conda-forge.org/)



```
conda config --add channels conda-forge  
conda config --set channel_priority strict  
conda update --all -y
```



Создаём среду из консоли



```
conda create -n test_env python=3.7.3  
conda activate test_env
```



Устанавливаем пакеты

- Сначала пытаемся поставить пакет из conda-forge
- Если не нашли (что бывает редко), то используем `pip install`



```
conda install numpy
```



IDE

Можно писать код в блокноте и запускать из командной строки. Но удобнее пользоваться специальными IDE:

- [Visual Studio Code](#) с поддержкой [Python](#). Бесплатная IDE с отличной поддержкой Python и ноутбуков. Нужно лишь поставить плагин Python, и можно кодить.
- [PyCharm](#). Ничего не нужно настраивать, мощная IDE из коробки. Есть бесплатная Community версия, которой хватит для базовых задач.
- Есть ещё куча разных вариантов, но для начала обучения лучше использовать мейнстримные инструменты.



Jupyter notebooks

В Data Science часто используются интерактивные ноутбуки Jupyter. Они позволяют комбинировать код, визуализации и текст в одном документе.

В этом курсе будем использовать Jupyter Lab.



```
conda install jupyterlab  
jupyter lab
```



Google Colab

Можно вообще ничего не устанавливать локально и воспользоваться Google Colab — бесплатной облачной средой Jupyter Notebooks.

В этом курсе все домашки можно сделать в Colab, не устанавливая ничего локально.

Ищите кнопку

