Программирование на языке Python

Орлов Илья Евгеньевич

Лекция №1

Введение в Python

- Положение среди других языков программирования
- Преимущества
- Недостатки
- Области применения
- Инструментарий
- PEP20 – дзен Питона
- Как выполняется код
- Основы синтаксиса
- Виртуальное окружение
- PEP8 – кодинг стайл
- Практика

Определение

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций (из Википедии).

От машинного кода к языкам высокого уровня

Преимущества Python

- Простота и понятность кода
- Простота разработки и поддержки
- Поддержка динамической типизации
- Автоматическое управление памятью (в частности, сборка мусора)
- Мощная стандартная библиотека («батарейки») и набор пакетов расширений
- Мультиплатформенность
- Мультипарадигменность

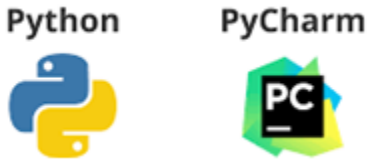
Недостатки Python

- Низкая производительность
- Ограничение распараллеливания из-за GIL (Global Interpreter Lock)
- Проблемы совместимости версий 2.x и 3.x
- Проблема безопасности из-за открытости кода

Области применения

- Прототипирование, создание POC (Proof of Concept)
- Научные расчеты (пакеты NumPy и SciPy)
- Автоматизация тестирования (Robot Framework)
- Скрипты и cli (command line interface)
- Машинное обучение и нейросети (PyTorch, TensorFlow, Keras)
- Веб-программирование (Django)

Инструментарий



- Интерпретатор со стандартной библиотекой (CPython)
- pip – система управления пакетами (начиная с версии Python 2.7.9 и Python 3.4, стандартная библиотека включает пакет pip по умолчанию)
- Среда разработки (IDE - Integrated Development Environment) – PyCharm Community
- Интернет, откуда собственно скачиваются все необходимые программы и пакеты, а также документация и примеры решения тех или иных задач

PEP20 - Дзен Питона (The Zen of Python)

- Красивое лучше, чем уродливое.
- Явное лучше, чем неявное.
- Простое лучше, чем сложное.
- Сложное лучше, чем запутанное.
- Плоское лучше, чем вложенное.
- Разреженное лучше, чем плотное.
- Читаемость имеет значение.
- Особые случаи не настолько особые, чтобы нарушать правила.
- При этом практичность важнее безупречности.
- Ошибки никогда не должны замалчиваться.
- Если не замалчиваются явно.
- Встретив двусмысленность, отбрось искушение угадать.
- Должен существовать один — и, желательно, только один — очевидный способ сделать это.
- Хотя он поначалу может быть и не очевиден, если вы не голландец.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.
- Пространства имён — отличная вещь! Давайте будем делать их больше!

Как выполняется код

test.py:

```
print('Hello, {}! '.format('user') * 3)
```

Запускаем:

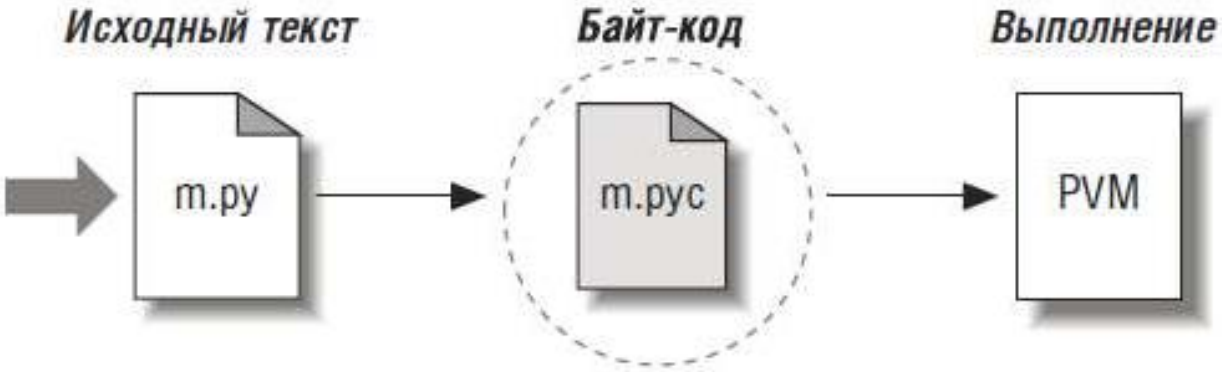
```
linuxuser@VBU:~/PyProjects$ python3 test.py
Hello, user! Hello, user! Hello, user!! Hello, user! Hello, user!
```

Также можно выполнить код через консоль.

```
linuxuser@VBU:~/PyProjects$ python3
Python 3.6.6 (default, Sep 12 2018, 18:26:19)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello, {}! '.format('user') * 3)
Hello, user! Hello, user! Hello, user!
>>>
```

Как выполняется код

В первом случае скрипт был преобразован в байт-код и выполнен виртуальной машиной Python:



Во втором случае было выполнено аналогичное преобразование, но без создания отдельного .pyc файла

Как выполняется код

Если в системе установлена только одна версия Python, интерпретатор можно вызвать командой:

```
linuxuser@VBU:~/PyProjects$ python
```

При наличии нескольких версий Python (как правило, двух: 2.x и 3.x) ссылки на исполняемый файл интерпретатора, а значит, и сами команды запуска можно настраивать. По умолчанию, предлагаются следующие варианты. В Linux:

```
linuxuser@VBU:~/PyProjects$ python3
Python 3.6.6 (default, Sep 12 2018, 18:26:19)
[GCC 8.0.1 20180414 (experimental) [trunk revision 259383]] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
linuxuser@VBU:~/PyProjects$ python2
Python 2.7.15rc1 (default, Nov 12 2018, 14:31:15)
[GCC 7.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Как выполняется код

В Windows (в командной строке):

```
F:\PyProjects>py -3
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

F:\PyProjects>py -2
Python 2.7.15 (v2.7.15:ca079a3ea3, Apr 30 2018, 16:22:17) [MSC v.1500 32 bit
(Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Вызов **python** (в Linux) и/или **py** (в Windows) запускает интерпретатор, заданный по умолчанию.

Для выхода из интерпретатора надо набрать команду **exit()**, либо нажать **Ctrl-D** (в Linux), либо **Ctrl-Z** (в Windows).

Основы синтаксиса Python

- Блоки инструкций выделяются отступами
- Конец строки завершает инструкцию
- Инструкции на одной строке разделяются точкой с запятой «;» (однако, оставлять несколько инструкций на одной строке не рекомендуется)
- После объявления функции, класса, условного оператора или цикла ставится двоеточие «:»

```
for item in items:
    item += 1; print item
```

Виртуальное окружение (virtualenv)

virtualenv – это пакет расширения, который можно установить, используя pip.

```
linuxuser@VBU:~/PyProjects$ pip3 install virtualenv
```

Для работы в виртуальном окружении его нужно сначала создать:

```
linuxuser@VBU:~/PyProjects$ virtualenv testenv
Using base prefix '/usr'
New python executable in /home/linuxuser/PyProjects/testenv/bin/python3
Also creating executable in /home/linuxuser/PyProjects/testenv/bin/python
Installing setuptools, pip, wheel...
done.
```

а когда понадобится его использовать - активировать:

```
linuxuser@VBU:~/PyProjects$ source testenv/bin/activate
(testenv) linuxuser@VBU:~/PyProjects$
```

а при необходимости вернуться в основное окружение - деактивировать:

```
(testenv) linuxuser@VBU:~/PyProjects$ deactivate
linuxuser@VBU:~/PyProjects$
```

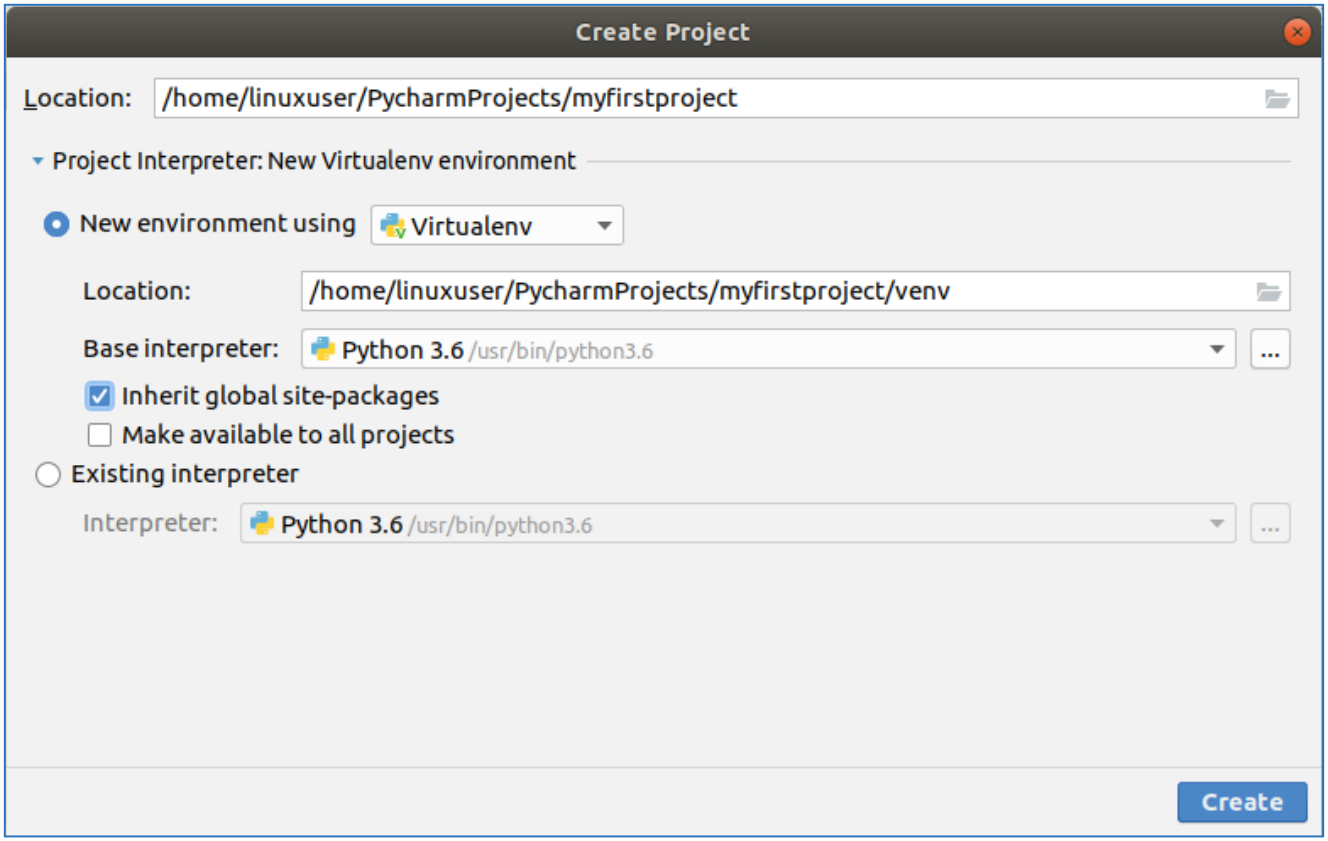
Виртуальное окружение (virtualenv)

В Windows работа с виртуальным окружением выполняется через командную строку практически аналогично Linux. Единственное отличие – в способе активации виртуального окружения:

```
F:\PyProjects> testenv\Scripts\activate
(testenv) F:\PyProjects>
```

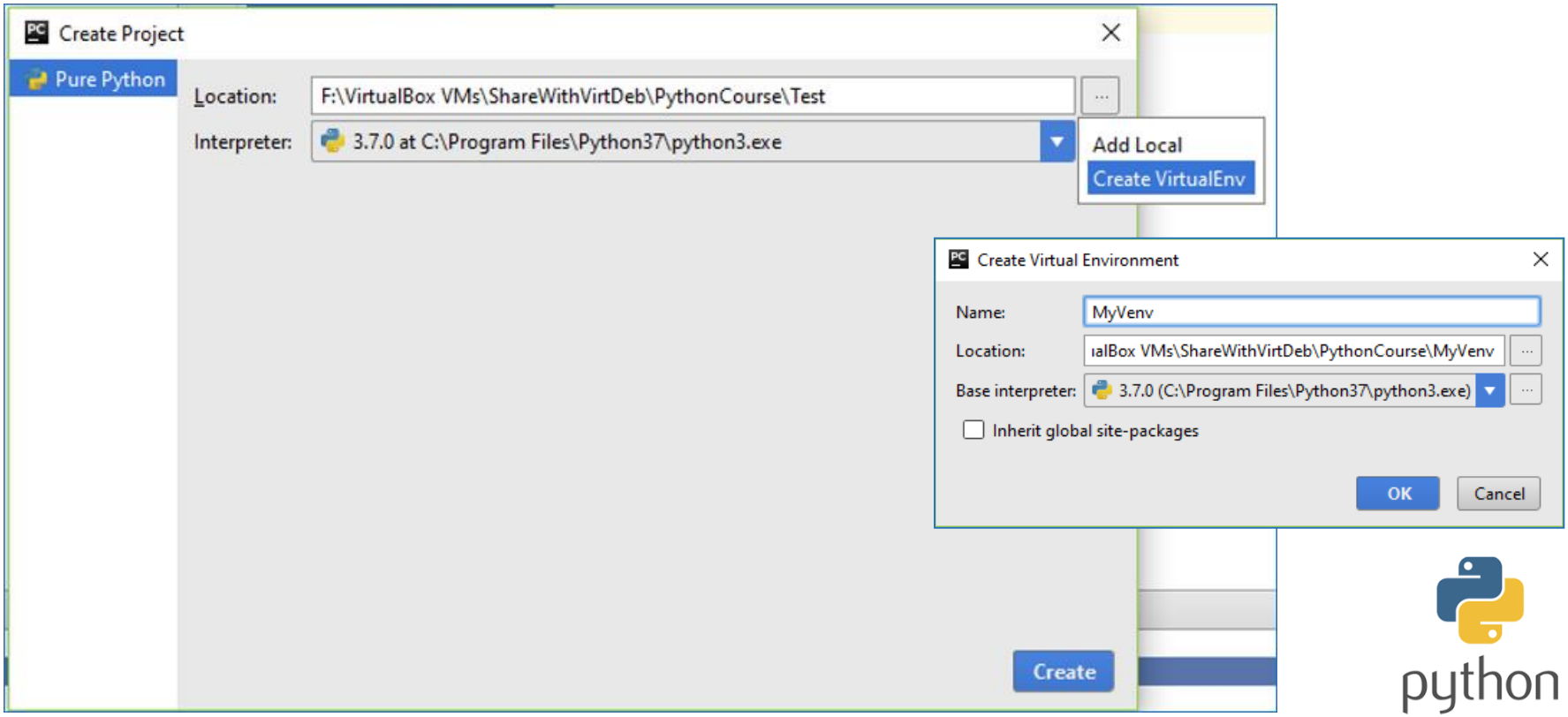
Виртуальное окружение (virtualenv)

В PyCharm виртуальное окружение можно создать при добавлении нового проекта (File->New Project…). В Linux это выглядит так:



Виртуальное окружение (virtualenv)

В версии PyCharm для Windows виртуальное окружение создается практически аналогично:



virtualenvwrapper

Часто дополнительно используется пакет virtualenvwrapper, который слегка упрощает (и без того, кстати, весьма простую) работу с virtualenv. Пакет virtualenvwrapper (для Windows - virtualenvwrapper-win) устанавливается так же при помощи pip:

```
linuxuser@VBU:~/PyProjects$ pip3 install virtualenvwrapper
```

Для работы с virtualenvwrapper в Linux нужно прописать необходимые переменные и запуск соответствующего скрипта в профиль пользователя (~/.bashrc) и активировать этот профиль:

```
linuxuser@VBU:~/PyProjects$ mkdir ~/test_pro
linuxuser@VBU:~/PyProjects$ which virtualenvwrapper.sh
/home/linuxuser/.local/bin/virtualenvwrapper.sh
linuxuser@VBU:~/PyProjects$ gedit ~/.bashrc
export WORKON_HOME=$HOME/.virtualenvs
export PROJECT_HOME=$HOME/test_pro
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source <строчка, которая появилась от команды which virtualenvwrapper.sh>
linuxuser@VBU:~/PyProjects$ source ~/.bashrc
```

virtualenvwrapper

В Windows необходимо добавить переменную окружения WORKON_HOME со значением %USERPROFILE%\Envs.
Используется virtualenvwrapper следующим образом:

```
linuxuser@VBU:~/PyProjects$ mkvirtualenv -p $(which python3) testenv

(testenv) linuxuser@VBU:~/PyProjects$ workon testenv # активация
(testenv) linuxuser@VBU:~/PyProjects$ python # запуск (python уже без тройки!)
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print('hello')
hello
>>>
(testenv) linuxuser@VBU:~/PyProjects$ deactivate # деактивация
linuxuser@VBU:~/PyProjects$ rmvirtualenv testenv # удаление testenv
```

PEP8 – кодинг стайл

1. Используйте 4 пробела для обозначения очередного уровня вложенности.
2. В многострочных выражениях элементы в скобках должны выравниваться либо вертикально по воображаемой линии внутри скобок (круглых, квадратных или фигурных), либо с использованием висячего отступа. При использовании висячего отступа на первой линии не должно быть аргументов, а остальные строки должны четко восприниматься как продолжение одного выражения.

```
# Правильно:
# Выровнено по открывающему разделителю
foo = long_function_name(var_one, var_two,
                                              var_three, var_four)

# Добавлено больше отступов, чтоб отличать части одного выражения
# от остального кода
def long_function_name(
            var_one, var_two, var_three,
            var_four):
    print(var_one)
```

PEP8 – кодинг стайл

```
# Неправильно:
# Аргументы на первой линии запрещены, если не используется
# вертикальное выравнивание
foo = long_function_name(var_one, var_two,
      var_three, var_four)

# Требуется больше отступов, чтоб отличать части одного
# выражения от остального кода
def long_function_name(
      var_one, var_two, var_three,
      var_four):
      print(var_one)
```

PEP8 – кодинг стайл

```
# Закрывающие круглые/квадратные/фигурные скобки
# в многострочных конструкциях могут находиться под
# первым непробельным символом последней строки списка,
my_list = [
        1, 2, 3,
        4, 5, 6,
        ]
result = some_function_that_takes_arguments(
        'a', 'b', 'c',
        'd', 'e', 'f',
        )

# либо быть под первым символом строки, начинающей
# многострочную конструкцию:
my_list = [
        1, 2, 3,
        4, 5, 6,
]
result = some_function_that_takes_arguments(
        'a', 'b', 'c',
        'd', 'e', 'f',
)
```

PEP8 – кодинг стайл

```
# Длинные строки могут быть разбиты на несколько строк,
# обернутых в скобки.
# Это предпочтительнее использования обратной косой черты
# для продолжения строки.
# Обратная косая черта все еще может быть использована время от времени.
# Например, длинная конструкция with не может использовать неявные
# продолжения так что обратная косая черта является приемлемой:
with open('/path/to/some/file/you/want/to/read') as file_1, \
           open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
```

PEP8 – кодинг стайл

3. Лучше пробелы, чем табуляция.
4. Длина строки кода не должна превышать 79 символов, документации и комментариев – 72 символа.
5. Отделяйте функции верхнего уровня и определения классов двумя пустыми строками.
6. Определения методов внутри класса разделяются одной пустой строкой.
7. Дополнительные пустые строки возможно использовать для разделения различных групп похожих функций. Пустые строки могут быть опущены между несколькими связанными однострочными определениями (например, набор фиктивных реализаций).
8. Используйте пустые строки в функциях, чтобы указать логические разделы.
9. В Python 3 не нужно указывать кодировку в файлах с кодом.
10. Каждый импорт, как правило, должен быть на отдельной строке.
11. Рекомендуется абсолютное импортирование.

PEP8 – кодинг стайл

12. Импорты должны быть сгруппированы в следующем порядке:
    1) импорты из стандартной библиотеки
    2) импорты сторонних библиотек
    3) импорты модулей текущего проекта

```
# Правильно:
import os
import sys
import mypkg.sibling  # абсолютный импорт

# Неправильно:
import sys, os
from . import sibling # относительный

# В то же время, можно писать так:
from subprocess import Popen, PIPE
```

PEP8 – кодинг стайл

13. Соглашения по именованию:

1) **_inner_var**: Нижнее подчеркивание перед первым символом говорит о слабой скрытности переменной.
2) **class_**: так пишется, когда слово уже является baseword'ом питона.
3) **__double_leading_underscore**: изменяет имя атрибута класса, то есть в классе **FooBar** поле **__boo** становится **_FooBar__boo** (атрибуты класса можно посмотреть, используя **MyClass.__dict_**).
4) **__double_leading_and_trailing_underscore__** (двойное подчеркивание в начале и в конце имени): магические методы или атрибуты.
5) Модули должны иметь короткие имена, состоящие из маленьких букв.
6) Имена классов должны обычно следовать соглашению CapWords.
7) Имена функций должны состоять из маленьких букв, а слова разделяться символами подчеркивания.
8) Всегда используйте self в качестве первого аргумента метода экземпляра объекта.
9) Всегда используйте cls в качестве первого аргумента метода класса.

Практика

1. Скачать и поставить Python 3.7 и 2.7.
2. Скачать и запустить PyCharm Community. Попробовать добавить новый проект с виртуальным окружением.
3. Создать виртуальное окружение в командной строке Windows и в Linux, как без virtualenvwrapper, так и с ним.
4. Попробовать попрограммировать: создать питоновский файл с парой команд (например, сложить два числа и вывести их на экран). Для этого не нужно ничего знать, но такие действия проделать необходимо.
5. Прочитать про систему управления версиями GIT.