Лекция 1: Начало

Сергей Лебедев

sergei.a.lebedev@gmail.com

7 сентября 2015 г.

- $n \approx 10$ лекций;
- $n - 1$ домашних заданий
  - несколько задач, которые нужно решить на Python,
  - решения можно сдавать в течение недели после лекции **без штрафа** и в течение второй недели **со штрафом** 50%;
  - каждая задача оценивается максимум 4 баллами: по два за корректность и стилистику;
- $n - 1$ тестов
  - несколько вопросов по материалам предыдущей лекции,
  - полностью правильный тест оценивается 4 баллами.

Скучающий компьютерный ученый

«[...] in December 1989, I was looking for a “hobby” programming project that would keep me occupied during the week around Christmas. My office [...] would be closed, but I had a home computer, and not much else on my hands.»

Foreword for “Programming Python” (1st ed.)

ABC

```
HOW TO RETURN words document:
    PUT {} IN collection
    FOR line IN document:
        FOR word IN split line:
            IF word not.in collection:
                INSERT word IN collection
    RETURN collection
```

Modula-3

```
TRY
  DO.Something()
EXCEPT
| IO.Error => IO.Put("An I/O error occurred.")
END;
```

- Хотелось простой, понятный, удобный и полезный язык с открытым исходным кодом.
- Получилось

```
def magic(dir):
    acc = []
    for root, dirs, files in os.walk(dir):
        acc.extend(os.path.join(root, file)
                   for file in files)

    return acc
```

- Что делает функция `magic`?

[1] https://www.python.org/doc/essays/cp4e

```
>>> def add(x, y):
...     return x + y
...
>>> def bar(x):
...     add(x, "1", "2")  # не ошибка
...
>>> add.__code__ = bar.__code__
>>> add(42)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in bar
TypeError: bar() takes 1 positional argument but 3 [...]
```

```
$ cat ./hello.py
message = "Hello, world!"
$ python -m dis ./hello.py
  0 LOAD_CONST               0 ('Hello, world!')
  3 STORE_NAME               0 (message)
  6 LOAD_CONST               1 (None)
  9 RETURN_VALUE
```

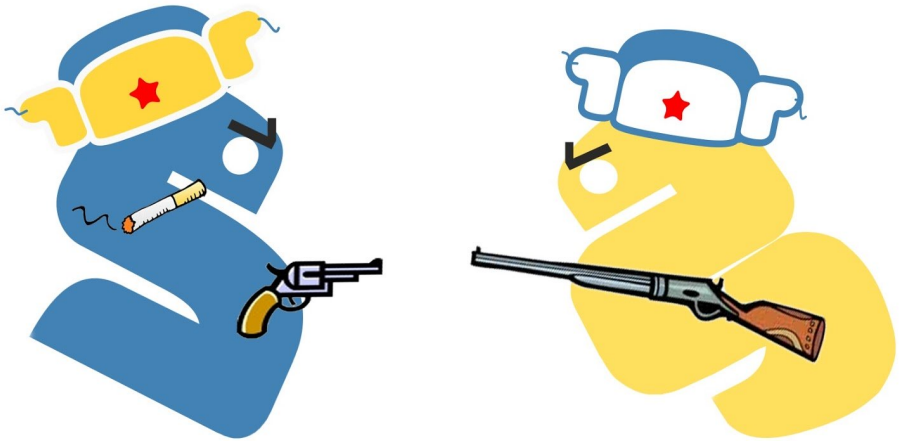

python™


Jython


python
stackless


PyPy

Python 2000 и Python 3000[2]



- Две несовместимых ветки языка: Python 2.X и Python 3.X.
- Официально ветка 2.X в режиме поддержки, а 3.X — в режиме развития.
- Мы будем использовать ветку 3.X.

[2] http://vitaliypodoba.com/2014/05/python2-or-python3

Идеи

Основные идеи языка: файл, модуль, пространство имён

```
>>> import hello
>>> hello
<module 'hello' from '[...]/hello.py'>
>>> dir(hello)
['__builtins__', '__cached__', '__doc__', '__file__',
 '__loader__', '__name__', '__package__', '__spec__',
 'message']
>>> hello.__name__, hello.__file__
('hello', '[...]/hello.py')
>>> print(hello.message)
"Hello, world!"
```

```
>>> while True:
... print(42)
  File "<stdin>", line 2
    print(42)
        ^
IndentationError: expected an indented block
```

```
>>> import hello
>>> type(hello)
<class 'module'>
>>> type(type(hello))
<class 'type'>
>>> type(type(type(hello)))
<class 'type'>
```

Типы

Странные

```
>>> None
>>> None == None
True
>>> None is None
True
```

Логические

```
>>> to_be = False
>>> to_be or not to_be
True
>>> True or print(None)  # ленивый оператор
True
>>> 42 + True
43
```

```
>>> 42       # int
42
>>> .42      # float
0.42
>>> 42j      # complex
42j
>>> 42**24   # приведение к длинному числу
907784931546351634835748413459499319296
>>> 16 / 3   # приведение к числу с плавающей точкой
5.333333333333333
>>> 16 // 3
5
>>> 16 % 3
1
```

```
>>> b"foo"
b'foo'
>>> b"foo".decode("utf-8")
'foo'
>>> "bar"
'bar'
>>> bar = "bar"
>>> len(bar)
3
>>> bar[0]
'b'
>>> s = "." * 10
>>> s
'..........'
>>> "<>" + s + "<"
'<>..........<'
>>> "  foo bar   ".strip()
'foo bar'
```

```
>>> []  # или list()
[]
>>> [0] * 4
[0, 0, 0, 0]
>>>
>>> xs = [1, 2, 3, 4]
>>> len(xs)
4
>>> xs[0]
1
>>> xs[0] = -1
>>> xs
[-1, 2, 3, 4]
>>> xs.append(42)
[-1, 2, 3, 4, 42]
>>> del xs[0]  # или xs.pop(0)
[2, 3, 4, 42]
```

Конкатенация

```
>>> xs = [1, 2, 3, 4]
>>> xs + [5, 6]
[1, 2, 3, 4, 5, 6]
```

```
>>> ", ".join(["foo", "bar"])
'foo, bar'
```

Слайсы

```
>>> xs = [1, 2, 3, 4]
>>> xs[:2]
[1, 2]
>>> xs[2:]
[3, 4]
>>> xs[1:3]
[2, 3]
>>> xs[0:4:2]
[1, 3]
>>> xs[:]
[1, 2, 3, 4]
```

```
>>> s = "foobar"
>>> s[:2]
'fo'
>>> s[2:]
'obar'
>>> s[1:3]
'oo'
>>> s[0:4:2]
'fo'
>>> s[:]
'foobar'
```

```
>>> tuple()  # неизменяемый
()
>>> date = ("year", 2015)
('year', 2015)
>>> len(date)
2
>>> date[1] = 2016
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> del date[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

```
>>> set()  # хеш-сет
set()
>>> xs = {1, 2, 3, 4}
>>> 42 in xs
False
>>> 42 not in xs
True
>>> xs.add(42)       # {1, 2, 3, 4, 42}
>>> xs.discard(42)   # {1, 2, 3, 4}
```

Капитан сообщает

Операторы `in` и `not in` работают для всех контейнеров:

```
>>> 42 in [1, 2, 3, 4]
False
>>> "month" not in ("year", 2015)
True
```

Основные типы языка: операции над множествами

```
>>> xs = {1, 2, 3, 4}
>>> ys = {4, 5}
>>> xs.intersection(ys)
{4}
>>> xs & ys
{4}
>>> xs.union(ys)
{1, 2, 3, 4, 5}
>>> xs | ys
{1, 2, 3, 4, 5}
>>> xs.difference(ys)
{1, 2, 3}
>>> xs - ys
{1, 2, 3}
```

```
>>> {}  # или dict(), хеш-таблица
{}
>>> date = {"year": 2015, "month": "September"}
>>> len(date)
2
>>> date["year"]
2015
>>> date.get("day", 7)
7
>>> date["year"] = 2016
>>> date
{'month': 'September', 'year': 2016}
>>> del date["year"]  # или date.pop("year")
{'month': 'Semptember'}
```

Вопрос

Как проверить наличие элемента в словаре?

```
>>> date = {"year": 2015, "month": "September"}
>>> date.keys()
dict_keys(['month', 'year'])
>>> date.values()
dict_values(['September', 2015])
>>> date.items()
dict_items([('month', 'September'), ('year', 2015)])
>>> other_date = {"month": "October", "day": 24}
>>> date.keys() + other_date.keys()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: [...]
>>> date.keys() | other_date.keys()
{'date', 'month', 'year'}
```

- Базовых типов не очень много:
  - **`None`**;
  - логические **`True`**, **`False`**;
  - числовые `int`, `float`, `complex`;
  - строковые `bytes`, `str`;
  - изменяемые коллекции `list`, `set`, `dict`,
  - неизменяемые коллекции `tuple`.
- Чаще всего над объектами из одной группы можно совершить конкретное действие одним очевидным способом:

```
>>> 1 + 1
>>> 1. + 1.
>>> 1j + 1j
```

```
>>> b"p" + b"y"
b'py'
>>> "p" + "y"
'py
```

```
>>> 42 in {42}
>>> 42 in [42]
>>> 42 in {42: "..."}
```

Управляющие конструкции

```
>>> x = 42
>>> if x % 5 == 0:
...     print("fizz")
... elif x % 3 == 0:
...     print("buzz")
... else:
...     pass  # необязательная ветка
...
buzz
```

Тернарный оператор

```
>>> "even" if x % 2 == 0 else "odd"
'even'
```

Мотивацию для такого решения Гвидо можно прочитать в архивах рассылки python-dev[3].

[3] http://bit.ly/python-dev-ifthen

```
>>> i = 0
>>> while i < 10:
...     i += 1
...
>>> i
10
```

```
>>> sum = 0
>>> for i in range(1, 5):
...     sum += i * i
...
>>> sum
30
```

Капитан сообщает

В Python есть операторы **break** и **continue**, которые работают ≈ также как и в других императивных языках.

```
>>> while False:
...     pass
... else:
...     print("What on earth is this?")
...
What on earth is this?
```

range

Принимает три аргумента: начало полуинтервала, конец полуинтервала и опциональный аргумент — шаг, который может быть отрицательным.

```
>>> range(0, 5, 2)
range(0, 5, 2)
>>> list(range(0, 5, 2))
[0, 2, 4]
>>> list(range(4, -1, -2))
[4, 2, 0]
```

reversed

Перечисляет элементы переданной ей последовательности в обратном порядке.

```
>>> list(reversed([1, 2, 3]))
[3, 2, 1]
```

for можно использовать для итерации по коллекциям, файлам и вообще много чему.

```
>>> for x in [0, 1, 1, 2, 3]:
...     pass
...
>>> for x in reversed([0, 1, 1, 2, 3]):
...     pass
...
>>> for line in open("./HBA1.txt"):
...     pass
...
>>> for ch in "abracadabra":
...     pass
```

- В Python есть многое из того, что так дорого программисту на императивном языке: `if`, `for`, `while`, тернарный оператор.
- В Python нет
  - фигурных скобок для обозначения области видимости,

```
>>> from __future__ import braces
  File "<stdin>", line 1
SyntaxError: not a chance
```

  - циклов с пост-условием, потому что `while` вполне достаточно,
  - операторов `switch` и `for` с явным счетчиком, потому что они имеют нетривиальную семантику.

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

IPython

- IPython — более лучшая интерактивная оболочка для языка Python.
- Она поддерживает:
  - интроспекцию и дополнение имён модулей, классов, методов, переменных,
  - работу с историей,
  - использование стандартных команд UNIX, например, `ls`,
  - систему “магических” команд `%%magic`.
- Установить можно с помощью менеджера пакетов `pip`

```
$ pip install "ipython[all]"
```

```
In [1]: from builtins import di<TAB>
dict    dir     divmod

In [2]: d = {"foo": "bar"}

In [3]: d.c<TAB>
d.clear  d.copy

In [4]: def f():
   ...:     "I do nothing and return 42."
   ...:     return 42
   ...:

In [5]: f?  # вывод ?? также содержит исходный код.
Type:           function
String form:    <function f at 0x103d68ae8>
File:           ...
Definition:     f()
Docstring:      I do nothing and return 42.
```

```
#  номер ячейки
#    |
#    v
In [1]: [1, 2, 3]
Out[1]: [1, 2, 3]

In [2]: _[1:]                  # ссылка на предыдущую ячейку
Out[2]: [2, 3]

In [3]: __[1:]                 # ссылка на две ячейки назад
Out[3]: [2, 3]

In [4]: Out[2] == Out[3]  # ссылка на ячейку по номеру
Out[4]: True
```

```
In [1]: pwd
Out[1]: './cscenter/python'

In [2]: ls lecture*.tex
lecture-base.tex  lecture01.tex

In [3]: cd /tmp
/tmp

In [4]: !date
Wed  2 Sep 2015 23:26:54 MSK

In [5]: output = !date

In [6]: output
Out[6]: ['Wed  2 Sep 2015 23:27:08 MSK']
```

- “Магические” команды отвечают за магию.
- Однострочные “магические” команды начинаются с символа %, многострочные — с %%.
- Пример однострочной “магической” команды, которая позволяет отредактировать и вычислить содержимое ячейки с указанным номером:

```
In [1]: def f():
   ...:     pass
   ...:

In [2]: %edit 1
IPython will make a temporary file named: [...]
done. Executing edited code...
Out[2]: 'def f():\n    return 42\n'

In [3]: f()
Out[3]: 42
```

- По умолчанию IPython работает в режиме `%automagic`, то есть префикс % у однострочных команд можно не писать.
- Команды UNIX, которыми мы уже пользовались, — это “магические” команды IPython, вызванные без префикса %.
- Пример многострочной “магической” команды, которая сохраняет содержимое ячейки в указанный файл:

```
In [1]: %%writefile /tmp/example.py
   ...: def f():
   ...:     return 42
   ...:
Writing /tmp/example.py

In [2]: %load /tmp/example.py

In [4]: f()
Out[4]: 42
```

IPython: больше “магических” команд

```
In [1]: env EDITOR
Out[1]: 'nano'

In [2]: env EDITOR=emacs
env: EDITOR=emacs

In [3]: %cpaste
Pasting code; enter '--' alone on the line to stop
:   def f():
:         return "I'm badly indented!"
:--

In [4]: f()
Out[4]: "I'm badly indented!"

In [5]: %timeit sum(range(1000))
10000 loops, best of 3: 25.1 µs per loop

In [6]: %lsmagic  # DIY.
```

- IPython — удобная и полезная альтернатива стандартной интерактивной оболочке Python.
- Мы поговорили только про основы её использования. Больше информации можно найти на сайте: `http://ipython.org`.