Лекция 1: Начало

Дмитрий Коргун

dmitry.tbb@ya.ru

24 сентября 2018г.

Создатель

«[…] 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 «Programing 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.');
```

Цели

- Хотелось простой, понятный 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?

Python - **динамический** интерпретируемый язык

```
In [1]: def add(x, y):
   ...:     return x + y
   ...:

In [2]: def bar(x):
   ...:     add(x, '1', '2') # Ошибки нет
   ...:

In [3]: add.__code__ = bar.__code__

In [4]: add(42)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-0c5efbbcdaa1> in <module>()
----> 1 add(42)

<ipython-input-2-4ad24f5f646f> in bar(x)
      1 def bar(x):
----> 2     add(x, '1', '2') # Ошибки нет
      3

TypeError: bar() takes 1 positional argument but 3 were given
```

Python - динамический **интерпретируемый** язык

```
tbb$ cat hello.py
message = 'Hello'
name = 'Gvido'
print(message, name)

tbb$ python -m dis ./hello.py
  1           0 LOAD_CONST               0 ('Hello')
              3 STORE_NAME               0 (message)

  2           6 LOAD_CONST               1 ('Gvido')
              9 STORE_NAME               1 (name)

  3          12 LOAD_NAME                0 (message)
             15 LOAD_NAME                1 (name)
             18 BUILD_TUPLE              2
             21 PRINT_ITEM
             22 PRINT_NEWLINE
             23 LOAD_CONST               2 (None)
             26 RETURN_VALUE
```

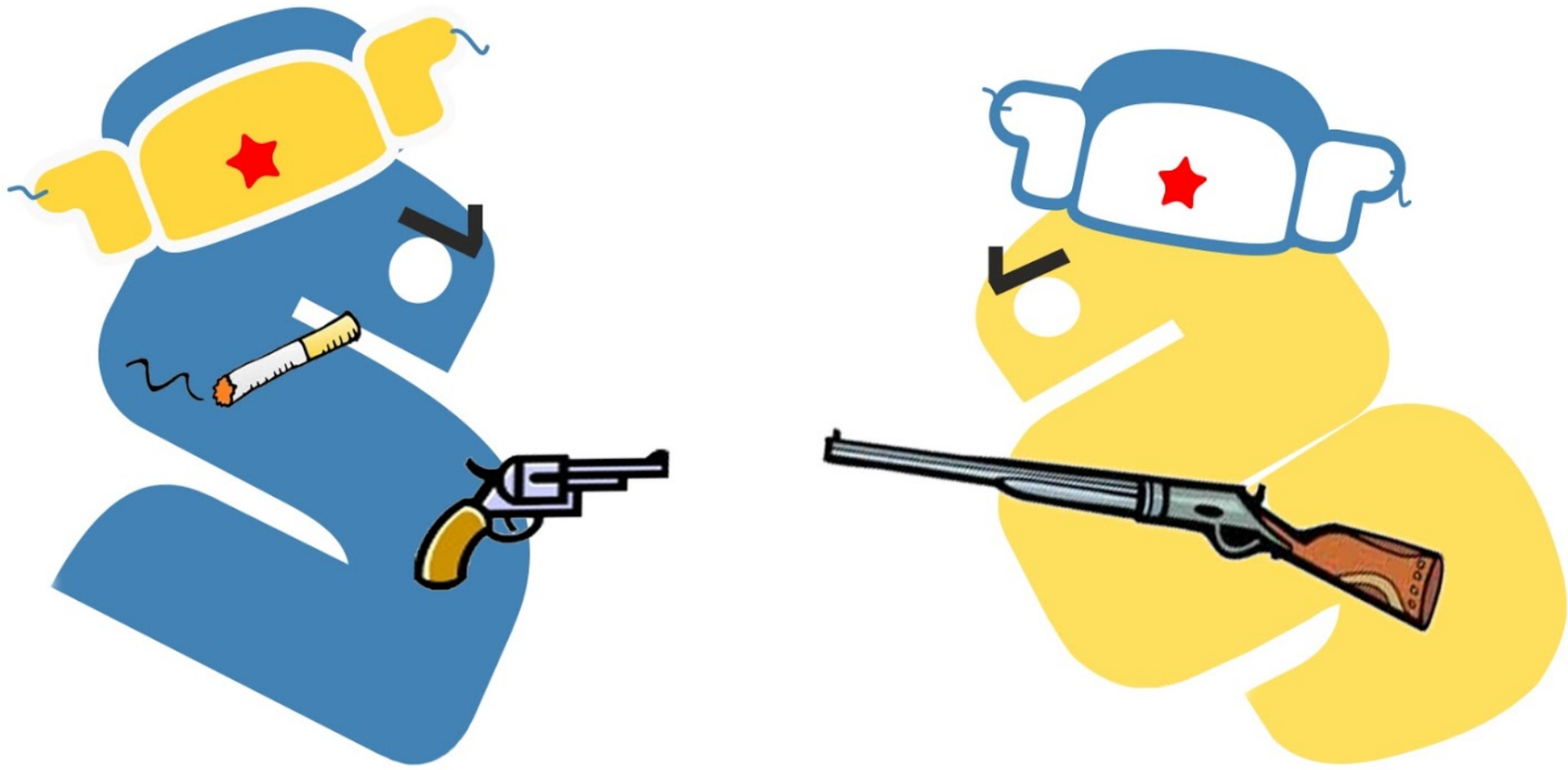
python™

PyPy

Jython

IronPython

Py2 vs Py3



- Две несовместимые ветки языка: 2.X и 3.X
- Официально вторая версия в режиме поддержки до 2020 года
- Третья версия в режиме развития

Основные идеи

```
In [5]: import hello
Hello Gvido

In [6]: hello
Out[6]: <module 'hello' from '/Users/tbb/hello.py'>

In [7]: dir(hello)
Out[7]:
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'message',
 'name']

In [8]: hello.__name__, hello.__file__
Out[8]: ('hello', '/Users/tbb/hello.py')
```

Основные идеи языка: отступы вместо скобочек

```
In [9]: while True:
   ...: print(42)
  File "<ipython-input-9-8f798d798a21>", line 2
    print(42)
        ^
IndentationError: expected an indented block
```

```
In [10]: import hello

In [11]: type(hello)
Out[11]: module

In [12]: type(type(hello))
Out[12]: type

In [13]: type(type(type(hello)))
Out[13]: type
```

Типы

None

```
In [14]: None

In [15]: None == None
Out[15]: True

In [16]: None is None
Out[16]: True
```

Логические

```
In [17]: to_be = False

In [18]: to_be or not to_be
Out[18]: True

In [19]: True or print('Hello') # Сокращенные вычисления (short-circuiting)
Out[19]: True

In [20]: 42 + True
Out[20]: 43
```

Основные типы данных: числовые

```
In [27]: 42    #int
Out[27]: 42

In [28]: .42   #float
Out[28]: 0.42

In [29]: 42j   #comlex
Out[29]: 42j

In [30]: 2 ** 128 #поддержка длинных чисел
Out[30]: 340282366920938463463374607431768211456
```

```
In [31]: 16 / 3
Out[31]: 5.333333333333333

In [32]: 16 //3
Out[32]: 5

In [33]: 16 % 3
Out[33]: 1
```

Основные типы данных: строковые

```
In [34]: 'bar'
Out[34]: 'bar'

In [35]: bar = 'bar'

In [36]: len(bar)
Out[36]: 3

In [37]: bar[0]
Out[37]: 'b'

In [38]: bar * 5
Out[38]: 'barbarbarbarbar'

In [39]: 'mala' + bar
Out[39]: 'malabar'

In [40]: '   foo bar     '.strip()
Out[40]: 'foo bar'
```

Основные типы данных: списки

```
In [41]: [] # или list()
Out[41]: []

In [42]: [0] * 4
Out[42]: [0, 0, 0, 0]

In [43]: xs = [1, 2, 3, 4]

In [44]: len(xs)
Out[44]: 4
```

```
In [45]: xs[0]
Out[45]: 1

In [46]: xs[0] = -1

In [47]: xs
Out[47]: [-1, 2, 3, 4]

In [48]: xs.append(42)

In [49]: del xs[0] # или xs.pos(0)
```

Срезы (или слайсы, от Slice)

```
In [53]: xs = [1, 2, 3, 4]

In [54]: xs[:2]
Out[54]: [1, 2]

In [55]: xs[2:]
Out[55]: [3, 4]

In [56]: xs[1:3]
Out[56]: [2, 3]

In [57]: xs[0:4:2]
Out[57]: [1, 3]

In [58]: xs[:]
Out[58]: [1, 2, 3, 4]
```

```
In [59]: s = 'foobar'

In [60]: s[:2]
Out[60]: 'fo'

In [61]: s[2:]
Out[61]: 'obar'

In [62]: s[1:3]
Out[62]: 'oo'

In [63]: s[0:4:2]
Out[63]: 'fo'

In [64]: s[:]
Out[64]: 'foobar'
```

Конкатенация

```
In [50]: xs = [1, 2, 3, 4]

In [51]: xs + [5, 6]
Out[51]: [1, 2, 3, 4, 5, 6]
```

```
In [52]: ', '.join(['foo', 'bar'])
Out[52]: 'foo, bar'
```

Основные типы данных: кортеж

```
In [65]: tuple() # Неизменяемый
Out[65]: ()

In [66]: date = ('year', 2015)

In [67]: len(date)
Out[67]: 2

In [68]: date[1] = 2016
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-8eb16e3d7459> in <module>()
----> 1 date[1] = 2016

TypeError: 'tuple' object does not support item assignment
```

```
In [82]: set() # множество, хеш-сет
Out[82]: set()

In [83]: xs = {1, 2, 3, 4}

In [84]: 42 in xs
Out[84]: False

In [85]: 42 not in xs
Out[85]: True

In [86]: xs.add(42)         # {1, 2, 3, 4, 42}

In [87]: xs.discard(42)     # {1, 2, 3, 4}
```

Капитан сообщает

Операторы in и not in работают для всех контейнеров:

```
In [88]: 42 in [1, 2, 3, 4]
Out[88]: False

In [89]: 'month' not in ('year', 2015)
Out[89]: True
```

Основные типы данных: операции над множествами

```
In [90]: xs = {1, 2, 3, 4}

In [91]: ys = {4, 5}

In [92]: xs.intersection(ys)
Out[92]: {4}

In [93]: xs & ys
Out[93]: {4}

In [94]: xs.union(ys)
Out[94]: {1, 2, 3, 4, 5}

In [95]: xs | ys
Out[95]: {1, 2, 3, 4, 5}

In [96]: xs - ys
Out[96]: {1, 2, 3}
```

Основные типы данных: словарь

```
In [97]: {} # или dict(), хэш-таблица
Out[97]: {}

In [98]: date = {'year': 2018, 'month': 'September'}

In [99]: date['year']
Out[99]: 2018

In [100]: date.get('day', 24)
Out[100]: 24

In [101]: date['year'] = 2019
```

```
In [102]: date
Out[102]: {'month': 'September', 'year': 2019}

In [103]: del date['year']
```

Вопрос

Как проверить наличие элемента в словаре?

Основные типы данных: ключи и значения

```
In [104]: date = {'year': 2018, 'month': 'September'}

In [105]: date.keys()
Out[105]: dict_keys(['year', 'month'])

In [106]: date.values()
Out[106]: dict_values([2018, 'September'])

In [107]: date.items()
Out[107]: dict_items([('year', 2018), ('month', 'September')])

In [108]: other_date = {'month': 'October', 'day': 24}

In [109]: date.keys() + other_date.keys()
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-109-74b5c6f292df> in <module>()
----> 1 date.keys() + other_date.keys()

TypeError: unsupported operand type(s) for +: 'dict_keys' and 'dict_keys'

In [110]: date.keys() | other_date.keys() # как множества
Out[110]: {'day', 'month', 'year'}
```

Основные типы данных: резюме

Базовых типов не так уж много:

- **None**
- Логические bool (**True** и **False**)
- Числовые int, float, complex
- Строковые bytes, str
- Изменяемые коллекции list, set, dict
- Неизменяемые коллекции tuple (сюда же можно отнести bytes и str)

Управляющие конструкции

Условный оператор

```
x = 42
if x % 5 == 0:
    print('fizz')
elif x % 3 == 0:
    print('buzz')
else:
    pass
```

Тернарный оператор

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

While

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

i # i = 10
```

For

```
i = 0
for i in range(1, 5):
    i += i * i

i # i = 30
```

Капитан сообщает

В Python есть операторы break и continue, которые работают так же как в других императивных языках.

```
for x in range(5):
    pass
else:
    print('For without break')
```

range

Принимает три аргумента: начало и конец полуинтервала, шаг.

```
In [113]: range(0, 5, 2)
Out[113]: range(0, 5, 2)

In [114]: list(range(0, 5, 2))
Out[114]: [0, 2, 4]

In [115]: list(range(4, -1, -2))
Out[115]: [4, 2, 0]
```

reversed

Перечисляет элементы переданной ей последовательности в обратном порядке

```
In [116]: list(reversed([1, 2, 3]))
Out[116]: [3, 2, 1]
```

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

```
for x in [0, 1, 2, 3]:
    pass

for x in reversed([0, 1, 2, 3]):
    pass
```

```
for line in open('./file.txt'):
    pass

for ch in 'abracadabra':
    pass
```

Основные конструкции языка: резюме

- В Python есть всё необходимое любому программисту: if, for, while и т.д.
- В Python нет
  - фигурных скобок для обозначения логических блоков и областей видимости, но…

```
In [117]: from __future__ import braces
  File "<ipython-input-117-6d5c5b2f0daf>", line 1
    from __future__ import braces
                                  ^
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.

```
In [119]: 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!
```