

Python2018

compscicenter.ru

aleksey.kladov@gmail.com

Acknowledgements

- основа материала -- курс Сергея Лебедева
- <https://github.com/superbobry/>
- 💖

Лекция 1

Введение, история, ликбез

Зачем учить питон?

- Язык по умолчанию для "написать что-нибудь"
- Один из самых популярных языков

Зачем учить питон?

- Язык по умолчанию для "написать что-нибудь"
- Один из самых популярных языков
- Фокус на коротком и лёгком коде "for humans"

Как писать хороший код?

Как писать хороший код?

РЕФЛЕКСИЯ

История

- 1989 — первый код
- 1994 — Python 1.0 (→)
- 1996 — Java 1.0, Ruby 1.0

Окружение для ОС Амоеба



Source:

Early years of Python

Влияния

- ABC
- Modula-3
- C

подход "worse is better"

```
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
```

<http://python-history.blogspot.com/>

- 2000 -- Python 2, PEP process
- 2008 -- Python 3
- 2018 -- Гвидо больше не BDLF



-
- 2000 -- Python 2, PEP process
 - 2008 -- Python 3
 - 2018 -- Гвидо больше не BDLF

Что получилось?

```
def magic(top):  
    acc = []  
    for entry in os.scandir(top):  
        if entry.is_file() and entry.name.endswith(".py"):  
            acc.append(entry.path)  
    return acc
```

- Интерпретируемый язык
- CPython -- reference implementation



IronPython



```
~/python-2018
λ cat arith.py
x = 30
y = 62
z = x + y
```

```
~/python-2018
λ python3 -m dis arith.py
1          0 LOAD_CONST           0 (30)
          2 STORE_NAME           0 (x)

2          4 LOAD_CONST           1 (62)
          6 STORE_NAME           1 (y)

3          8 LOAD_NAME             0 (x)
         10 LOAD_NAME             1 (y)
         12 BINARY_ADD
         14 STORE_NAME           2 (z)
         16 LOAD_CONST           2 (None)
         18 RETURN_VALUE
```

```
def eval(code):
    env = {}
    stack = []
    for line in code.strip().splitlines():
        op, *args = line.split()
        print(f"env : {env}\nstack: {stack}\n")
        if op == "LOAD_CONST":
            stack.append(int(args[0]))
        elif op == "STORE_NAME":
            env[args[0]] = stack.pop()
        elif op == "LOAD_NAME":
            stack.append(env[args[0]])
        elif op == "BINARY_ADD":
            stack.append(stack.pop() + stack.pop())
        else:
            assert False, f"unknown op: {op[0]}"
    print(f"env : {env}\nstack: {stack}\n")
```

```
eval("""
LOAD_CONST    30
STORE_NAME    x
LOAD_CONST    62
STORE_NAME    y
LOAD_NAME     x
LOAD_NAME     y
BINARY_ADD
STORE_NAME    z
""")
```

```
env : {}
stack: []

env : {}
stack: [30]

env : {'x': 30}
stack: []

env : {'x': 30}
stack: [62]

env : {'x': 30, 'y': 62}
stack: []

env : {'x': 30, 'y': 62}
stack: [30]

env : {'x': 30, 'y': 62}
stack: [30, 62]

env : {'x': 30, 'y': 62}
stack: [92]

env : {'x': 30, 'y': 62, 'z': 92}
stack: []
```

- Динамический язык
- Проверка типов во время исполнения
- read/write интроспекция
- everything is an object
- everything is a statement (!)

```
def add(a, b):  
    return a + b  
  
def f():  
    return add(1, "92", [])  
  
f.__code__ = add.__code__  
f(62, 30)
```


- Динамический язык
- Проверка типов во время исполнения
- read/write интроспекция
- everything is an object
- everything is a statement (!)

```
>>> type(92)
<class 'int'>
>>> type(type(92))
<class 'type'>
>>> type(type(type(92)))
<class 'type'>
```

- Объектно ориентированный?
- Если вы настаиваете
- Функциональный?
- Скорее нет

-- Быстрый?

-- Скорее всего, это не важно

```
>>> import timeit
>>> timeit.timeit('sum(range(10**8))', number=1)
1.6271799539972562
>>> timeit.timeit(
...     'urlretrieve("http://python.org")',
...     'from urllib.request import urlretrieve',
...     number=1,
... )
1.1692394479978248
```

Встроенные типы

None

```
λ python3
Python 3.6.6 (default, Jun 27 2018, 05:47:41)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> None           # аналог null, но полноценный объект
>>> res = print(None) # любая функция возвращает значение
None
>>> res == None     # не делайте так
True
>>> res is None     # делайте лучше так
True
>>> id(res)         # в CPython -- адресс
140503861261072
>>> id(None)
140503861261072
>>>
```

bool

```
>>> to_be = False
>>> to_be or not to_be      # слова вместо значков
True
>>> x = 1
>>> y = 2
>>> x**2 + y**2 < 5 == True  # True синглтон, не делайте так
False
>>> x**2 + y**2 < 5 is True  # но и так тоже не делайте
False
>>> x**2 + y**2 < 5
False
```

bool

```
>>> False and print('also')      # short-circuiting!
False
>>> res = True and print('also')
also
>>> assert res is None, "print should return None"
>>> False or 92                  # работает с любым значением
92
```

assert

```
>>> x = 100
>>> y = 3
>>> assert x % y == 0, (x, y)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError: (100, 3)
```


Number

```
>>> True + False + True    # bool тоже числа
2
>>> 0
0
>>> 2**128                  # сколько угодно знаков
340282366920938463463374607431768211456
>>> 1.0                     # double
1.0
>>> float('inf')
inf
>>> int('92')
92
>>> 90 + 2j                 # комплексные числа тоже есть
(90+2j)
```

Number

```
>>> 3 / 2      # вещественное деление
1.5
>>> 4 / 2
2.0
>>> 4 // 2     # деление с остатком
2
>>> 3 // 2
1
>>> -3 // 2    # как в алгебре!
-2
>>> -1 % 3     # C/Java/Rust скажут -1
2
```

Number

```
>>> x = 10
>>> 0 <= x and x < 100
True
>>> 0 <= x < 100
True
```

list

```
>>> []  
[]  
>>> xs = [1, 2, 3,]  
>>> len(xs)  
3  
>>> xs[0]  
1  
>>> xs[0] = 0  
>>> xs  
[0, 2, 3]
```

list

```
>>> xs = [1, 2] * 3
>>> xs
[1, 2, 1, 2, 1, 2]
>>> xs = [[0] * 3] * 3 # не делайте так
>>> xs[0][0] = 1
>>> xs
[[1, 0, 0], [1, 0, 0], [1, 0, 0]] # :-(
```

list

```
>>> [1] + [2, 3] + [4] # O(?)
[1, 2, 3, 4]
>>> xs = [1, 2, 3]
>>> xs.append(4)        # O(?)
>>> xs
[1, 2, 3, 4]
>>> xs.pop()           # O(?)
4
>>> xs
[1, 2, 3]
>>> xs.pop(0)          # O(?)
1
>>> xs
[2, 3]
>>> xs.insert(0, 92)    # O(?)
>>> xs
[92, 2, 3]
>>> xs += [1]           # так делать не стоит
>>> xs
[92, 2, 3, 1]
```

slices

```
>>> xs = list(range(5))
>>> xs
[0, 1, 2, 3, 4]
>>> xs[len(xs) - 1]
4
>>> xs[-1]
4
>>> xs[2:4]
[2, 3]
>>> xs[:-2]
[0, 1, 2]
>>> xs[:2]
[0, 1]
>>> xs[::2]
[0, 2, 4]
>>> xs[:]
[0, 1, 2, 3, 4]
```

slices

```
>>> y = xs[:] # или y = list(xs)
>>> y[0] = 92
>>> y
[92, 1, 2, 3, 4]
>>> xs
[0, 1, 2, 3, 4]
```


slices, weird edition

```
>>> xs[:100]                                # слайс может выходить за границу
[0, 1, 2, 3, 4]
>>> xs[100]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> xs[1:-1:2] = [None] * 2                  # для слайсов работает присваивание
>>> xs
[0, None, 2, None, 4]
>>> every_second = slice(None, None, 2)      # слайсы -- first-class объект
>>> list(range(10))[every_second]
[0, 2, 4, 6, 8]
```

str

```
>>> s = "hello" # 'hello'
>>> len(s)
5
>>> s[:-1]
'hell'
>>> s[0]
'h'
>>> s[1] = "i" # в отличие от списков, строки не изменяемы
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> "<" + "." * 8 + "<"
'<.....<'
```

str

```
>>> "hello \nworld".splitlines() # разбить на строки
['hello ', 'world']
>>> "a b c".split() # на слова
['a', 'b', 'c']
>>> "\thello ".strip() # убрать пробелы
'hello'
>>> ", ".join(["a", "b", "c"]) # соединить список строк через разделитель
'a, b, c'
>>> str(42)
'42'
>>> x = 92
>>> f"2 * x = {2 * x}" # интерполяция
'2 * x = 184'
```

tuple

```
>>> date = ("September", 2018)
>>> len(date)                # API, как у списка
2
>>> date[1] = 2019            # но менять нельзя
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> xs = ([], [])
>>> xs[0].extend([1, 2, 3])
>>> xs
([1, 2, 3], [])
```

tuple

```
>>> () # пустой кортеж
()
>>> 1
1
>>> (1, ) # кортеж из одного элемента
(1,)
>>> date = "September", 2018 # скобки опциональны
>>> date
('September', 2018)
```

tuple

```
>>> def div_mod(x, y):  
...     return x // y, x % y  
...  
>>> d, m = div_mod(10, 3)  
>>> assert (d, m) == (3, 1)
```

set

```
>>> xs = {1, 2, 3} # множество (hash set)
>>> 1 in xs
True
>>> 92 not in xs
True
>>> xs.add(1)
>>> xs.add(92)
>>> xs
{1, 2, 3, 92}      # в множестве нет повторений
>>> set()           # для пустого множества нет литерала
set()
```

set

```
>>> xs = {1, 2, 3} # множество (hash set)
>>> 1 in xs
True
>>> 92 not in xs
True
>>> xs.add(1)
>>> xs.add(92)
>>> xs
{1, 2, 3, 92}      # в множестве нет повторений
>>> set()           # для пустого множества нет литерала
set()

>>> 1 in [1, 2, 3]  # O(?)
True
>>> "world" in "hello, world"
True
```


set

```
>>> {1, 2, 3}.union({3, 4, 5}) # или |
{1, 2, 3, 4, 5}
>>> {1, 2, 3} & {3, 4, 5}      # или .intersection
{3}
>>> {1, 2, 3} ^ {3, 4, 5}      # или .symmetric_difference
{1, 2, 4, 5}
>>> xs = {1, 2, 3}
>>> xs.discard(2) # в реальной жизни операция удаления -- редкая
>>> xs
{1, 3}
```

set

```
>>> xs = set()
>>> xs.add([])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

dict

```
>>> date = {"year": 2018, "month": "September"}
>>> len(date)
2
>>> date["year"]           # KeyError если ключа нет, в отличие от Java
2018
>>> date.get("day", 14)    # Значение по умолчанию
14
>>> date["day"] = 14
>>> date.pop("year")
2018
```

dict

```
>>> date.keys()
dict_keys(['month', 'day'])      # set
>>> date.values()
dict_values(['September', 14])  # he set
>>> date.items()
dict_items([('month', 'September'), ('day', 14)])  # set!
```

dict

```
>>> date.items() | {1: 2}.items()
{('day', 14), ('month', 'September'), (1, 2)}
>>> date.items() | {1: []}.items()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
```

dict

```
>>> 'day' in date.keys() # плохо!  
True  
>>> 'day' in date  
True
```

dict

```
>>> d = {}
>>> d["a"] = 1
>>> d["b"] = 2
>>> d["c"] = 3
>>> list(d.keys())
['a', 'b', 'c']      # порядок гарантируется
```

Встроенные типы, итоги

- None;
 - True, False;
 - int, float, complex;
 - str;
 - list, set, dict и tuple.
-
- одинаковое API для похожих типов
 - могущественные операции
 - не надо ничего изобретать, можно брать и программировать

Документация:

<https://docs.python.org/3/library/stdtypes.html>

Выражения

if

```
if 0 <= n and n < len(xs):  
    print(xs[n])
```

ternary if

```
x = 50  
y = 25  
small = x if x < y else y  
# int small = x < y ? x : y;
```

if

```
# :( :( :(
if x[0] < 100 and x[1] > 100 and (is_full_moon() or not is_thursday()) and user.is_admin
    pass
```

if

```
# >:(  
if x[0] < 100 and x[1] > 100  
    and (is_full_moon() or not is_thursday())  
    and user.is_admin:  
    pass
```

if

```
# :( :( :(
if x[0] < 100 and x[1] > 100 \
    and (is_full_moon() or not is_thursday()) \
    and user.is_admin:
    pass
```

if

```
# :( :(
if (x[0] < 100 and x[1] > 100
    and (is_full_moon() or not is_thursday()))
    and user.is_admin):
    pass
```

if

```
# :|  
value_in_range = x[0] < 100 and x[1] > 100  
good_date = is_full_moon() or not is_thursday()  
if value_in_range and good_date and user.is_admin:  
    pass
```


while

```
i = 0  
while i < 4:  
    i += 1  
i
```

Truthy/Falsy

```
>>> bool(True)
True
>>> bool(0)
False
>>> bool(1)
True
>>> bool([])
False
>>> bool([0])
True
```

Truthy/Falsy

```
(False,          # Falsy!  
None,  
0, 0.0, 0j,  
'',  
[], (), set(), {})
```

Truthy/Falsy

```
if len(xs) == 0: # плохо!  
    pass
```

```
if xs:  
    pass
```

```
if not xs:  
    pass
```

for

```
for x in [1, 2, 3]:  
    print(x)
```

```
for line in open("./HBA1.txt"):  
    pass
```

как правильно -- в следующих сериях

```
for ch in "foobar":  
    pass
```

for

```
for i in range(10):  
    print(x)
```

```
for i in range(2, 10, 3):  
    print(x)
```

```
for i in range(9, -1, -1):    # :(  
    print(x)
```

```
for i in reversed(range(10)): # :)  
    print(x)
```

```
for i in reversed("hello"):  
    print(x)
```

```
for i in reversed([1, 2, 3]):  
    print(x)
```

break

```
target = 92
for item in items:
    if item == target:
        print("Found!", item)
        break
```

break

```
target = 92
for item in items:
    if item == target:
        print("Found!", item)
        break
print("Not found")  # :(
```


break

```
target = 92
found = False # :( :( :(
for item in items:
    if item == target:
        print("Found!", item)
        found = True
        break
if found:
    print("Not found")
```

break

```
target = 92
for item in items:
    if item == target:
        print("Found!", item)
        break
else:
    print("Not found")
```

continue

```
target = 92
res = []
for item in items:
    if item != target:
        continue
    res.append(item)
```

break/continue

- нет меток
- внутри функции работает return

Выражения, итоги

- Нет C-style for
- Итерироваться можно по всему
- else к for и while

Документация:

https://docs.python.org/3/reference/compound_stmts.html

```
def eval(code):
    env = {}
    stack = []
    for line in code.strip().splitlines():
        op, *args = line.split()
        print(f"env : {env}\nstack: {stack}\n")
        if op == "LOAD_CONST":
            stack.append(int(args[0]))
        elif op == "STORE_NAME":
            env[args[0]] = stack.pop()
        elif op == "LOAD_NAME":
            stack.append(env[args[0]])
        elif op == "BINARY_ADD":
            stack.append(stack.pop() + stack.pop())
        else:
            assert False, f"unknown op: {op[0]}"
    print(f"env : {env}\nstack: {stack}\n")
```

```
eval("""
LOAD_CONST    30
STORE_NAME    x
LOAD_CONST    62
STORE_NAME    y
LOAD_NAME     x
LOAD_NAME     y
BINARY_ADD
STORE_NAME    z
""")
```

```
env : {}
stack: []
```

```
env : {}
stack: [30]
```

```
env : {'x': 30}
stack: []
```

```
env : {'x': 30}
stack: [62]
```

```
env : {'x': 30, 'y': 62}
stack: []
```

```
env : {'x': 30, 'y': 62}
stack: [30]
```

```
env : {'x': 30, 'y': 62}
stack: [30, 62]
```

```
env : {'x': 30, 'y': 62}
stack: [92]
```

```
env : {'x': 30, 'y': 62, 'z': 92}
stack: []
```

Что читать

- Learning Python, М. Lutz -- учебник
- Python Cookbook, D. Beazly -- effective Python, рекомендуется

Документация:

<https://docs.python.org/3/reference/index.html>

Что читать в транспорте

<https://docs.python.org/3/library/stdtypes.html>