Python2018

compscicenter.ru

aleksey.kladov@gmail.com

В предыдущей серии

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
```

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")
```

break/continue

- нет меток
- внутри функци работает return

Лекция 2

Функции

```
~/python-2018
λ cat def.py
def foo():
    """I do nothing and return 92."""
    return 92
foo()

~/python-2018
λ python3 -m dis def.py
λ python3 -m dis def.py
  1           0 LOAD_CONST               0 (<code object ...>)
              2 LOAD_CONST               1 ('foo')
              4 MAKE_FUNCTION            0
              6 STORE_NAME               0 (foo)

  4           8 LOAD_NAME                0 (foo)
             10 CALL_FUNCTION            0
             12 POP_TOP
             14 LOAD_CONST               2 (None)
             16 RETURN_VALUE
```

```
>>> def foo():
...     """I do nothing and return 92."""
...     return 92
...
>>> foo.__name__
'foo'
>>> foo.__doc__
'I do nothing and return 92.'
>>> help(foo)
```

```
def min(x, y):
    return x if x < y else y
```

```
def min(x, y):
    return x if x < y else y

min(1, 2)
min(1, y=2)
min(x=1, y=2)
min(y=2, x=1)
```

```
def min(*args):
    # type(args) => <class 'tuple'>

    res = float('inf')
    for x in args:
        res = x if x < res else res # Monoid!
    return res

min(92, 10, 62)
min()
xs = [1, 2, 3]
min(*xs)
```

```
def min(first, *rest):
    res = first
    for x in rest:
        res = x if x < res else res
    return res


>> min("hello", ",", " ", "world")
' '
>>> min()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: min() missing 1 required
    positional argument: 'first'
```

```
def min(*args, default=None):
    if not args:
        return default

    res, *rest = args
    for x in rest:
        res = x if x < res else res
    return res


min(xs, default=0) # must use a name!
```

```
~/python-2018
λ cat default.py
def foo(x=[], y=92):
    pass

~/python-2018
λ python3 -m dis default.py
  1           0 BUILD_LIST               0
              2 LOAD_CONST               0 (92)
              4 BUILD_TUPLE              2
              6 LOAD_CONST               1 (<code object ...>)
              8 LOAD_CONST               2 ('foo')
             10 MAKE_FUNCTION            1
             12 STORE_NAME               0 (foo)
             14 LOAD_CONST               3 (None)
             16 RETURN_VALUE
```

```
>>> def foo(x=[], y=92):
...     pass
...
>>> foo.__defaults__
([], 92)
```

```
def unique(xs, seen=set()):
    res = []
    for x in xs:
        if x in seen:
            continue
        seen.add(x)
        res.append(x)
    return res

>>> unique([1, 2, 3])
[1, 2, 3]
>>> unique([1, 2, 3])
[]
>>> unique.__defaults__
({1, 2, 3},)
>>> print(":-(")
:-(
```

```
def unique(xs, seen=None):
    seen = seen or set()
    res = []
    for x in xs:
        if x in seen:
            continue
        seen.add(x)
        res.append(x)
    return res

>>> unique([1, 2, 3, 2])
[1, 2, 3]
>>> unique([1, 2, 3, 2])
[1, 2, 3]
```

```
flatten([0, 1, [2, [3]], 4], 1)
```

```
flatten([0, 1, [2, [3]], 4], depth=1)
```

```
def flatten(iterable, *, depth=None):
    """I flatten a given iterable up to a fixed depth."""
    ...
```

```
>>> def call_me(*args, **kwargs):
...     return (args, kwargs)
...
>>> (args, kwargs) = call_me(1, 2, a=3, b=4)
>>> args
(1, 2)
>>> kwargs
{'a': 3, 'b': 4}
>>> type(kwargs)
<class 'dict'>
>>> kwargs.pop('b')
4
```

```
>>> def call_me(*args, **kwargs):
...     return (args, kwargs)
...
>>> call_me({"a": 92})
```

```
>>> def call_me(*args, **kwargs):
...     return (args, kwargs)
...
>>> call_me({"a": 92})
(({'a': 92},), {})
```

```
>>> def call_me(*args, **kwargs):
...     return (args, kwargs)
...
>>> call_me(**{"a": 92})
```

```
>>> def call_me(*args, **kwargs):
...     return (args, kwargs)
...
>>> call_me(**{"a": 92})
((), {'a': 92})
```

```
def forwarder(*args, **kwargs):
    return f(*args, **kwargs)
```

- позиционные/именнованые аргументы
- только именнованные аргументы
- упаковка: def foo(*args, **kwargs)
- распакова: foo(*[1, 2, 3], **{"foo": 92})

```
>>> x, *xs = [1, 2, 3]
>>> x, xs
(1, [2, 3])
```

```
>>> x, *xs = "123"
>>> x, xs
('1', ['2', '3'])
```

```
>>> first, *rest, last = [1, 2, 3]
>>> first, rest, last
(1, [2], 3)
```

```
>>> rectangle = ((0, 0), (2, 3))
>>> (x1, y1), (x2, y2) = rectangle
>>> y2
3
```

```
"""
name,price,quantity
spam,8,92
"""

d = {}
for line in text.splitlines():
    cells = line.split(',')
    d[cells[0]] = cells[1]
```

```
"""
name,price,quantity
spam,8,92
"""

d = {}
for line in text.splitlines():
    name, price, _ = line.split(',') # checks format!
    d[name] = price
```

```
"""
name,price,quantity
spam,8,92
"""

d = {}
for line in text.splitlines():
    name, price, *_ = line.split(',') # ignore explicitly!
    d[name] = price
```

```
>>> print(*[1], *[2], 3)
1 2 3
>>> dict(**{'x': 1}, y=2, **{'z': 3})
{'x': 1, 'y': 2, 'z': 3}
```

```
>>> *range(4), 4
(0, 1, 2, 3, 4)
>>> [*range(4), 4]
[0, 1, 2, 3, 4]
>>> {*range(4), 4}
{0, 1, 2, 3, 4}
>>> {'x': 1, **{'y': 2}}
{'x': 1, 'y': 2}
```

```
>>> {'x': 1, **{'x': 2}}
{'x': 2}
>>> {**{'x': 2}, 'x': 1}
{'x': 1}
```

Области видимости

```
def is_even(n):
    return n == 0 if n <= 1 else is_odd(n - 1)


def is_odd(n):
    return n == 1 if n <= 1 else is_even(n - 1)


assert is_even(92)
```

```
def is_even(n):
    return n == 0 if n <= 1 else is_odd(n - 1)


assert is_even(92)


def is_odd(n):
    return n == 1 if n <= 1 else is_even(n - 1)
```

```
def is_even(n):
    return n == 0 if n <= 1 else is_odd(n - 1)


assert is_even(92)


def is_odd(n):
    return n == 1 if n <= 1 else is_even(n - 1)


Traceback (most recent call last):
  File "scopes.py", line 5, in <module>
    assert is_even(92)
  File "scopes.py", line 2, in is_even
    return n == 0 if n <= 1 else is_odd(n - 1)
NameError: name 'is_odd' is not defined
```

```
x = 1


def foo():
    y = 2
    print(globals(), type(globals()))
    print(locals(), type(locals()))


foo()

# {'x': 1, 'foo': <function ...>, ... } <class 'dict'>
# {'y': 2} <class 'dict'>
```

```
def foo():

    x = 92
    def bar():
        return x
    return bar

bar = foo()
assert bar() == 92
```

```
def foo():
    blob = [None] * 10**8
    x = 92
    def bar():
        return x
    return bar
```

```
def foo():
    def bar():
        return x
    print(bar.__closure__)
    print(locals())
    x = 92
    print(bar.__closure__)
    print(locals())


# (<cell at 0x7f9d3ef75c18: empty>,)
# {'bar': <function ... >}

# (<cell at 0x7f9d3ef75c18: int object ... >,)
# {'bar': <function ... >, 'x': 92}
```

```
x = 1
def foo():
    print(x)
    # x = 2

foo()
# prints(1)
```

```
x = 1
def foo():
    print(x) # --\ at function compile-to-bytecode time
    x = 2    # <-/

foo()
# UnboundLocalError:
# local variable 'x' referenced before assignment
```

LEGB

- local, locals() -- по индексу
- enclosing, foo.__closure__ -- по индексу
- global, foo.__globals__, globals() -- поиск в словаре
- builtins

https://docs.python.org/3.7/reference/executionmodel.html#naming-and-binding

```
x = 1
def foo():
    global x
    x = 2
    y = 1
    def bar():
        nonlocal y
        y = 2
```

```
def foo():
    res = []
    for i in range(3):
        def bar():
            return i

        res.append(bar)
    return res

for f in foo():
    print(f(), end=" ") # prints 2 2 2
```

```
def foo():
    def is_even(n):
        return n == 0 if n <= 1 else is_odd(n - 1)

    def is_odd(n):
        return return n == 1 if n <= 1 else is_even(n - 1)
```

```
def foo():
    res = []
    for i in range(3):
        def bar(i=i):
            return i

        res.append(bar)
    return res
```

```
def foo():
    def make_bar(i):
        def bar():
            return i
        return bar

    res = []
    for i in range(3):
        res.append(make_bar(i))
    return res
```

- статическое разрешение локальных имён
- globals()
- locals() (это не правда)
- замыкания

```
lambda arguments: expression
```

```
def _(arguments):
    return expression
```

```
lambda a, *args, b=1, **kwargs: 92
```

```
f = lambda x: expr # не делайте так

def f(x):
    return expr
```

```
>>> range(3)
range(0, 3)
>>> list(range(3))
[0, 1, 2]
>>> map(lambda x: x + 1, [0, 1, 2])
<map object at 0x7fc54d060da0>
>>> list(map(lambda x: x + 1, [0, 1, 2]))
[1, 2, 3]
```

```
>>> list(map(lambda x, y: x + y, [0, 1, 2], [3, 4, 5, 6]))
[3, 5, 7]
```

```
>>> list(filter(lambda x: x % 2 == 0, range(10)))
[0, 2, 4, 6, 8]
```

```
>>> list(filter(None, [0, 1, True, False, [], {None}]))
[1, True, {None}]
```

```
>>> list(zip("hello", range(10)))
[('h', 0), ('e', 1), ('l', 2), ('l', 3), ('o', 4)]
```

```
assert len(xs) == len(ys)
for x, y in zip(xs, ys):
    ...
```

```
>>> [x**2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
```

```
>>> [x**2 for x in range(10) if x % 2 == 0]
[0, 4, 16, 36, 64]
>>> list(map(
...     lambda x: x**2,
...     filter(lambda x: x % 2 == 0, range(10)),
... ))
[0, 4, 16, 36, 64]
```

```
>>> [(x, y)
...  for x in range(5)
...  if x % 2 == 0
...  for y in range(x)
...  if y % 2 == 1]
[(2, 1), (4, 1), (4, 3)]
```

```
res = [
    (x, y)
    for x in range(5)
    if x % 2 == 0
    for y in range(x)
    if y % 2 == 1
]

res = []
for x in range(5):
    if x % 2 == 0:
        for y in range(x):
            if y % 2 == 1:
                res.append((x, y))
```

```
>>> {x**2 % 5 for x in range(5)}
{0, 1, 4}
>>> {x: x**2 for x in range(5)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
>>> (x**2 for x in range(5))
<generator object <genexpr> at 0x7fcb9b4dac50>
>>> map(lambda x: x**2, range(5))
<map object at 0x7fcb9b4d6e80>
```

```
x, y, z = map(int, input().split())
```

Что читать в транспорте

https://www.python.org/dev/peps/pep-0008/