Немного о правилах хорошего тона

Truthy and Falsy values

refer to values which are evaluated to True or False

In [21]:
```
print(bool([]))

print(bool([1]))

print(bool(0))

print(bool(''))

print(bool(b''))

print(bool('hello'))
```

```
False
True
False
False
False
True
```

Falsy - None, False, 0, 0.0, 0j, пустые строки/байты, пустые коллекции.

Используйте эту семантику для проверки коллекции на пустоту

In [5]:
```
smth = []

# плохо
if smth == []:
    ...
if len(smth) != 0:
    ...


# лучше
if not smth:
    ...


some_counter = 0
if smth == 0:
    ...
```

А какие еще membership operators существуют в Python?

in

Evaluates to true if it finds a variable in the specified sequence and false otherwise.

`x in y` - here in results in a 1 if x is a member of sequence y.

not in

Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

`x not in y` - here not in results in a 1 if x is not a member of sequence y.

Не используйте dict.get и коллекцию dict.keys для проверки наличия ключа в словаре:

```
In [ ]:
pocket = {}

# Плохо
if key in pocket.keys():
    ...

if not pocket.get(key, False):
    ...

# Лучше
if key in pocket:
    ...

if key not in pocket:
    ...
```

Используйте литералы для создания пустых коллекций. Исключение: set, литералов пустого множества в Python нет.

In [ ]:
```
# Плохо
dict(), list(), tuple()

# Лучше
{}, [], ()
```

Все дело в скорости..

```
In [8]: from timeit import timeit as tm

        print(tm("d = dict()"))
        print(tm("d = {}"))
```

```
0.13073958399763796
0.028096617999835870
```

```
In [17]: from dis import dis

         dis('d = {}')
```

```
  1           0 BUILD_MAP                0
              2 STORE_NAME               0 (d)
              4 LOAD_CONST               0 (None)
              6 RETURN_VALUE
```

```
In [18]: dis('d = dict()')
```

```
  1           0 LOAD_NAME                0 (dict)
              2 CALL_FUNCTION            0
              4 STORE_NAME               1 (d)
              6 LOAD_CONST               0 (None)
              8 RETURN_VALUE
```

Оккам достает бритву. тут вам не Scala

In [ ]:
```
# Плохо
i = 0
while i < n:
    ...
    i += 1

# Лучше
for i in range(n):
    ...
```

```
In [ ]: # Плохо
        for i in range(len(xs)) :
            x = xs[i]

        # Лучше
        for x in xs:
            ...

        # Или
        for i, x in enumerate(xs):
            ...
```

In [ ]:

```
# Плохо
if condition:
    return True
else
    return False

# Лучше
return condition
```

Не итерируйтесь по файлу через методы `readline()` `readlines()`

```
In [ ]:
# Плохо
while True:
    line = file.readline()
    ...

for line in file.readlines():
    ...


# Лучше
for line in file:
    ...
```

Any & All in Python

Returns true if any of the items is True. It returns False if empty or all are false. Any can be thought of as a sequence of OR operations on the provided iterables. It short circuit the execution i.e. stop the execution as soon as the result is known.

Syntax : any(list of iterables)

Returns true if all of the items are True (or if the iterable is empty). All can be thought of as a sequence of AND operations on the provided iterables. It also short circuit the execution i.e. stop the execution as soon as the result is known.

Syntax : all(list of iterables)

In [ ]:
```
xs = [x for x in xs if predicate]
return True if xs else False

# Лучше
xs = [x for x in xs if predicate]
return bool(xs)

# супир-пупир
return any(map(predicate, xs))
return all(map(predicate, xs))
```

В любой непонятной ситуации используй методы встроенных структур данных

ну хотя бы Counter и defaultdict оставьте, молю 20:02

In [ ]:
```
# Плохо
s[:len(p)] == p
s.find(p) == len(s) - len(p)

# Лучше
s.startswith(p)
s.endswith(p)
```

Используй форматирование строк вместо явных вызовов str и конкатенации.

In [ ]:
```
# Плохо
"(+ " + str(expr1) + " " + str(expr2) + ")"

# Лучше
"(+ {} {})".format(expr1, expr2)
```

Исключение: приведение к строке одного объекта

In [ ]:
```
# Плохо
"{}".format(value)

# Лучше
str(value)
```

метод str.format **преобразует аргументы в строку.**

```
In [ ]: # Плохо
        "(+ {} {})".format(str(expr1), str(expr2))

        # Лучше
        "(+ {} {})".format(expr1, expr2)
```

Functions

```
In [11]: def funny_function():
             return 'to_the_blue_lagoon'
```

```
In [4]: funny_function()

Out[4]: 'to_the_blue_lagoon'
```

```
In [5]: funny_function

Out[5]: <function __main__.funny_function()>
```

Ограничение на выбор имени функции типичны

- буквы
- подчеркивание _
- цифры 0-9, **но не в начале!_**

```
In [6]: def 1foo():
            pass
```

```
  File "<ipython-input-6-735460777140>", line 1
    def 1foo():
        ^
SyntaxError: invalid syntax
```

return можно опустить - по умолчанию функция возвращает None

```
In [15]: def foo():
             'foo'
         print(foo())
```

```
None
```

```
In [16]: print(print(foo()))
```

```
None
None
```

return может быть несколько

```
In [26]: def never_gonna(what):
             if what == 1:
                 return 'give you up'
             if what == 2:
                 return 'let you down'
             return 'run around and desert you'
             print("You wouldn't get this from any other guy")

         print(never_gonna(1))
         print(never_gonna(10))
```

```
give you up
run around and desert you
```

Для документации функции используют строковые литералы:

```
In [7]: def creep():
            """I wish I was special"""
            return 'unreal'
```

Как их найти?

```
In [9]: creep.__doc__

Out[9]: 'I wish I was special'
```

```
In [10]: help(creep)

Help on function creep in module __main__:

creep()
    I wish I was special
```

Arguments

Positional arguments

```
In [28]: def avg(a, b):
             return (a+b)/2

         avg(10, 9)

Out[28]: 9.5
```

Keyword arguments

```
In [23]: def order_an_ice_cream(scoop, toping="syrup", flavor="chocolate"):
             return f"{scoop} scoop(s) with {flavor} and {toping} toping"


         print(order_an_ice_cream(10))
         print(order_an_ice_cream(3, "nut", "strawberries and bananas"))
         print(order_an_ice_cream(scoop=1, toping="KETCHUP", flavor="vanilla"))
```

```
10 scoop(s) with chocolate and syrup toping
3 scoop(s) with strawberries and bananas and nut toping
1 scoop(s) with vanilla and KETCHUP toping
```

```
In [24]: print(order_an_ice_cream(3, toping="nut", "strawberries and bananas"))
```

```
  File "<ipython-input-24-1eaced7ef92e>", line 1
    print(order_an_ice_cream(3, toping="nut", "strawberries and bananas"))
                                             ^
SyntaxError: positional argument follows keyword argument
```

Инициализация значений по умолчанию

In [35]:
```
def foo(a, lst=[]):
    lst.append(a)
    return lst

print(foo(1))
print(foo(2))
print(foo(3))
print(foo(1, ['q']))
```

```
[1]
[1, 2]
[1, 2, 3]
['q', 1]
```

```
In [36]: def foo(a, lst=None):
             lst = lst or []
             lst.append(a)
             return lst


         print(foo(1))
         print(foo(2))
         print(foo(3))
         print(foo(1, ['q']))
```

```
[1]
[2]
[3]
['q', 1]
```

Упаковка

```
In [37]: def avg(*args):
             return sum(args)/len(args)
```

```
In [38]: avg(1, 2, 3, 4, 5, 3.50)

Out[38]: 3.0833333333333335
```

```
In [39]: avg()
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-39-f5d909301850> in <module>
----> 1 avg()

<ipython-input-37-983fe2e3ee7a> in avg(*args)
      1 def avg(*args):
----> 2     return sum(args)/len(args)

ZeroDivisionError: division by zero
```

```
In [4]: def avg(first, *args):
            print(type(args))
            numbers = (first,) + args
            return sum(numbers)/len(numbers)

        avg(1, 2, 4)
        avg()
```

```
<class 'tuple'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-4-6e5fd68e1040> in <module>
      4     return sum(args)/len(args)
      5 avg(1, 2, 4)
----> 6 avg()

TypeError: avg() missing 1 required positional argument: 'first'
```

In [12]:
```python
def avg_with_kwargs(first, *args, **kwargs):
    numbers = (first,) + args
    res = sum(numbers)/len(numbers)
    if kwargs.get('do_print', False):
        print('Some very informative print telling us that return value is', res
)
    return numbers

print(avg_with_kwargs(1, 10, 100))

print(avg_with_kwargs(1, 10, 100, **{'do_print': True}))

settings = {'do_print': True}
print(avg_with_kwargs(1, 10, 100, **settings))
```

```
(1, 10, 100)
Some very informative print telling us that return value is 37.0
(1, 10, 100)
Some very informative print telling us that return value is 37.0
(1, 10, 100)
```

Function is a first-class-object (https://stackoverflow.com/questions/245192/what-are-first-class-objects)!

- может быть сохранен в переменной или структурах данных;
- может быть передан в функцию как аргумент;
- может быть возвращен из функции как результат;
- может быть создан во время выполнения программы;

In [25]:
```
def my_function():
    print('I am a function')
```

In [26]:
```
print(my_function)
print('Functions are objects -', isinstance(my_function, object))
```

```
<function my_function at 0x7f98947b37b8>
Functions are objects - True
```

Можно назначить переменную, хранящую ссылку на функцию

In [3]:
```
test = my_function
test()
```

```
I am a function
```

С функцией можно делать все, что и с обычным объектом

In [27]:
```
my_list = []
my_list.append(my_function)
print(my_list)
```

```
[<function my_function at 0x7f98947b37b8>]
```

Можно передать как параметр

```
In [ ]: def call_passed_function(incoming):
            print('Calling!')
            incoming()
            print('Called!')

        call_passed_function(my_function)
```

Можно вернуть функцию из функции

```
In [8]: def return_min_function():
            return min

        test = return_min_function()
        min_value = test(4, 5, -9, 12)
        print('Min values is', min_value)
```

```
Min values is -9
```

Можно создать аттрибут и положить туда что-то

```
In [13]: def foo():
             return 'moo'

         foo.attr = 'foo'
         foo.attr

Out[13]: 'foo'
```

И обратиться к нему

```
In [14]: def foo(): return foo.__name__

         foo()

Out[14]: 'foo'
```

Можно ли вызвать все что угодно?

In [6]:
```
try:
    d = 2
    d()  # but you can try
except TypeError as e:
    print('It is not a function', e)
```

```
It is not a function 'int' object is not callable
```

Callable

Вызвать функцию - вызвать метод `__call__` у объекта. Вызов типа `add(1, 2) == add.__call__(1, 2)`

Определяя функцию типа `def funcname(parameters):` вы в действительности создаете новый объект с определенным методом `__call__`

Проверить, что объект `callable`

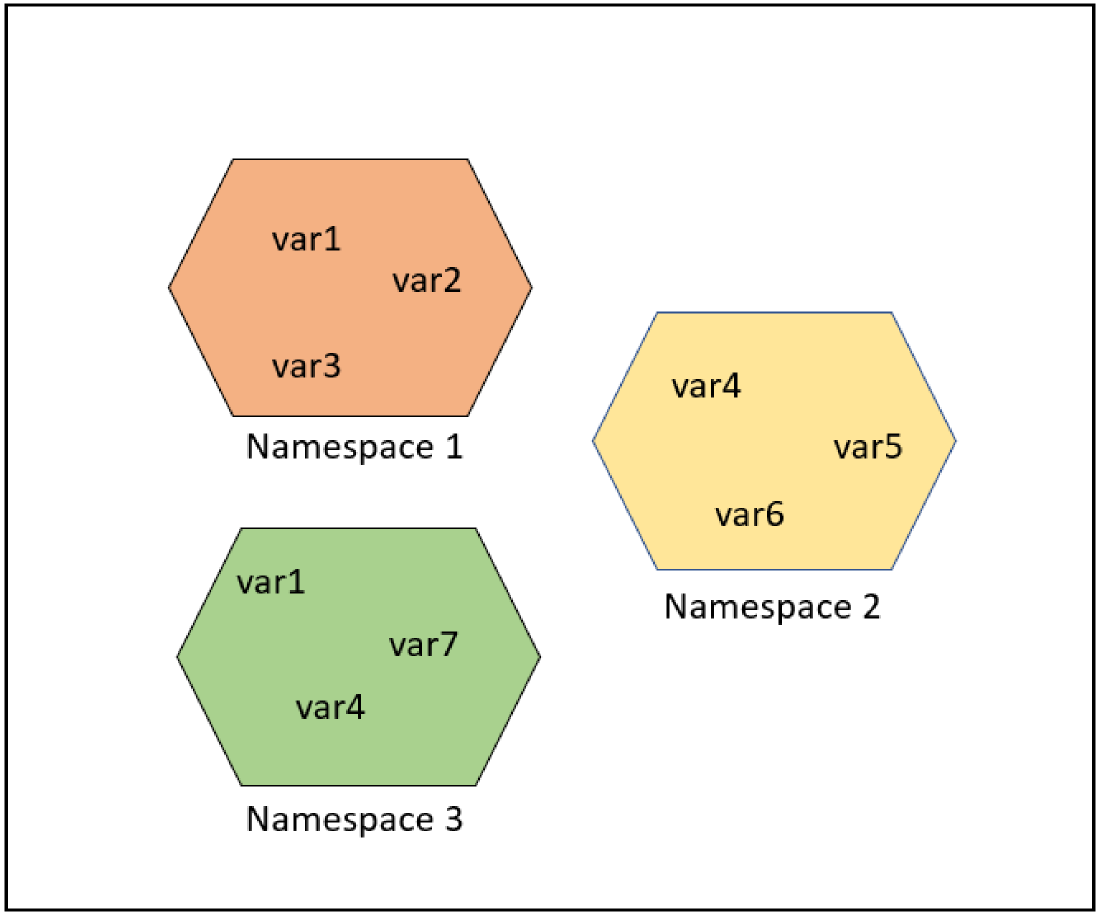
```
In [7]: print(callable(len), callable(45), callable(callable))
```

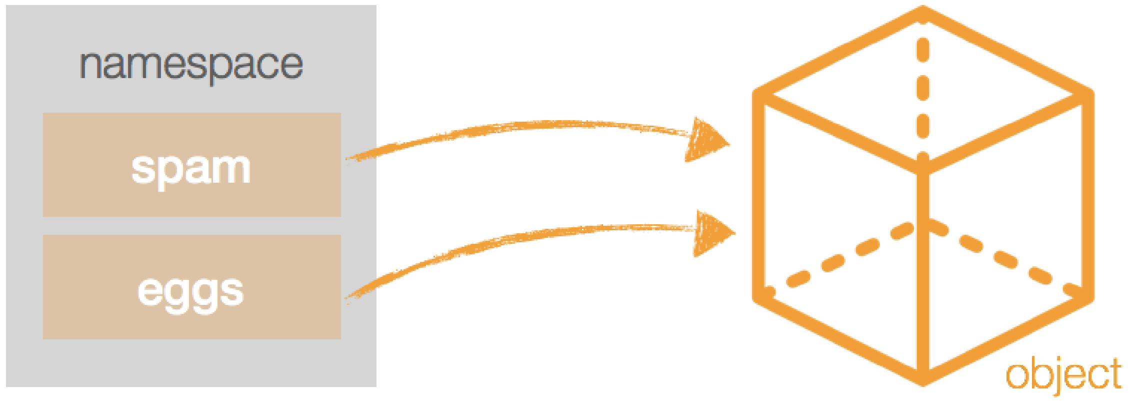
```
True False True
```

SCOPES AKA ОБЛАСТИ ВИДИМОСТИ

Пространство имен – это соотнесение имен с объектами, желательно без конфликтов.

Namespace ~= dict





```
In [2]: spam = 'spam and eggs'
        eggs = spam

        print(spam)  # spam and eggs
        print(eggs)  # spam and eggs

        print(id(spam))
        print(id(eggs))
```

```
spam and eggs
spam and eggs
140714621322480
140714621322480
```

Encapsulation and scoping

Замыкание – возможность функции использовать чужие переменные.

In [19]:
```
def spam():
    eggs = 'spam and eggs'
    def cantine():
        print(eggs)
    cantine()

spam()
```

```
spam and eggs
```

In [17]:
```
def spam():
    print(eggs)

eggs = 'spam and eggs'
spam()  # spam and eggs
```

```
spam and eggs
```

```
In [1]: def spam():
            eggs = 'spam and eggs'
            print(eggs)

        spam()        # spam and eggs
        print(eggs)   # raises a NameError exception
```

```
spam and eggs

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-ae7795babba4> in <module>
      4
      5 spam()        # spam and eggs
----> 6 print(eggs)   # raises a NameError exception

NameError: name 'eggs' is not defined
```

Инициализируя объект класса, мы также создаем новую область видимости.

```
In [22]: class Meal:
             def __init__(self):
                 self.eggs = 2


         my_meal = Meal()
         print(my_meal.eggs)    # 2
         print(eggs)            # raises a NameError exception
```

```
2

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-22-0462a63f2338> in <module>
      6 my_meal = Meal()
      7 print(my_meal.eggs)    # 2
----> 8 print(eggs)            # raises a NameError exception

NameError: name 'eggs' is not defined
```

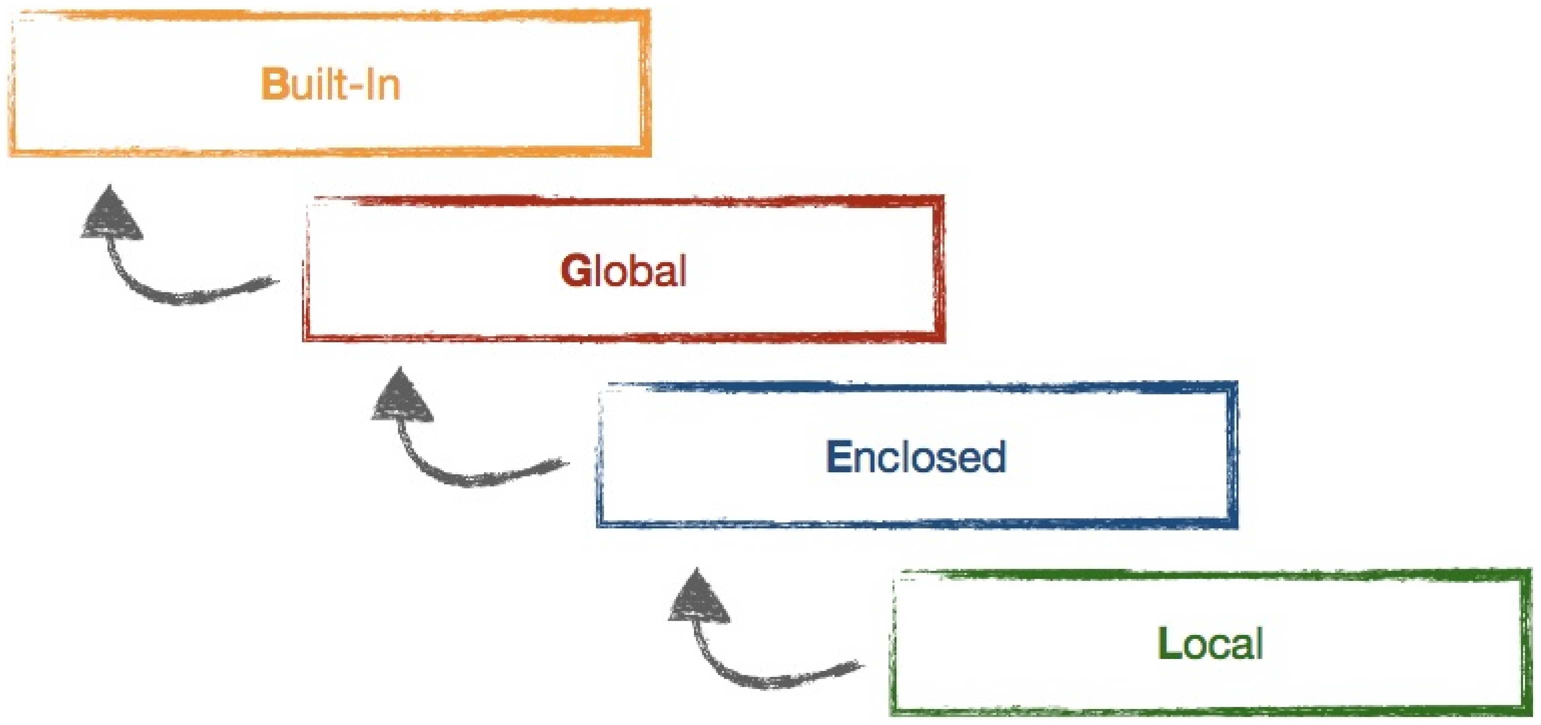
Посмотреть, что у объекта в namespace, можно через `dir()`

```
In [28]: dir(my_meal)
```

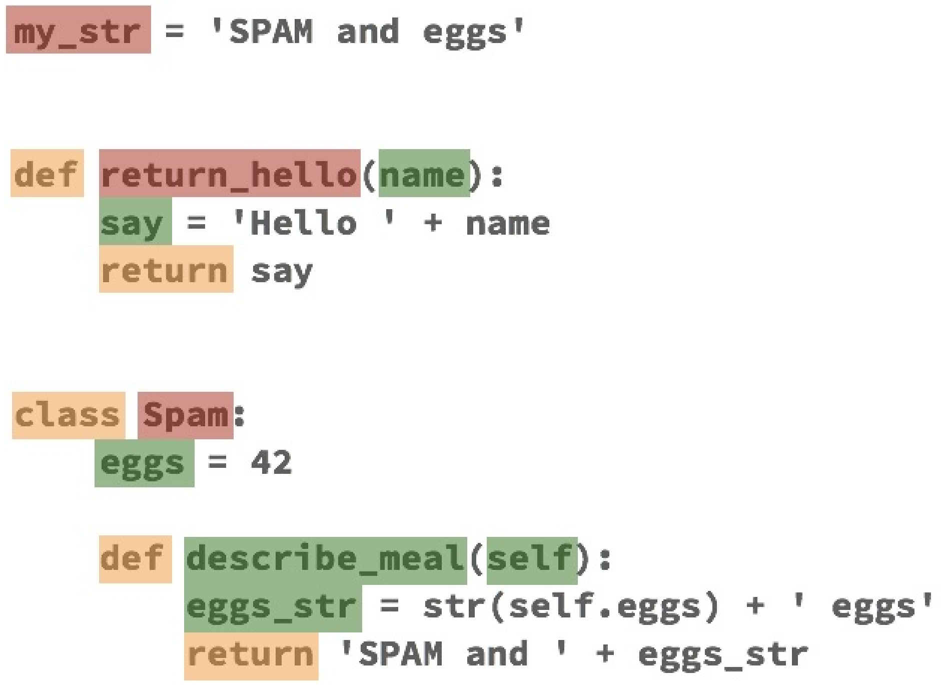
```
Out[28]: ['__class__',
          '__delattr__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__gt__',
          '__hash__',
          '__init__',
          '__init_subclass__',
          '__le__',
          '__lt__',
          '__module__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__setattr__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'eggs']
```

LEGB

Поиск имени ведётся не более, чем в четырёх областях видимости: локальной, затем в объемлющей функции (если такая имеется), затем в глобальной и, наконец, во встроенной.



Local– Names which are assigned within a function.
Enclosing – Names which are assigned in a closure (function in a function)
Global – Names which are assigned at the top-level of a module, for example on the top-level of your Python file
Built-in – Names which are standard Python built-ins, such as open, import, print, return, Exception



```
my_str = 'SPAM and eggs'


def return_hello(name):
    say = 'Hello ' + name
    return say


class Spam:
    eggs = 42

    def describe_meal(self):
        eggs_str = str(self.eggs) + ' eggs'
        return 'SPAM and ' + eggs_str
```

In [29]:
```python
glabal_var = 0

def func():
    var = 'variable'

    def print_vars():
        inner_var = 1
        print('inner_var', inner_var) # local
        print('var', var) # enclosing
        print('global_var') # global
        print('func', func)
    print_vars()

func()
```

```
inner_var 1
var variable
global_var
func <function func at 0x7fe4a4626f28>
```

In [30]:
```
from dis import dis
dis(func)
```

```
  4           0 LOAD_CONST               1 ('variable')
              2 STORE_DEREF              0 (var)

  6           4 LOAD_CLOSURE             0 (var)
              6 BUILD_TUPLE              1
              8 LOAD_CONST               2 (<code object print_vars at 0x7fe4a
4352420, file "<ipython-input-29-9aac0c951ca1>", line 6>)
             10 LOAD_CONST               3 ('func.<locals>.print_vars')
             12 MAKE_FUNCTION            8
             14 STORE_FAST               0 (print_vars)

 12          16 LOAD_FAST                0 (print_vars)
             18 CALL_FUNCTION            0
             20 POP_TOP
             22 LOAD_CONST               0 (None)
             24 RETURN_VALUE

Disassembly of <code object print_vars at 0x7fe4a4352420, file "<ipython-input
-29-9aac0c951ca1>", line 6>:
  7           0 LOAD_CONST               1 (1)
              2 STORE_FAST               0 (inner_var)

  8           4 LOAD_GLOBAL              0 (print)
              6 LOAD_CONST               2 ('inner_var')
              8 LOAD_FAST                0 (inner_var)
             10 CALL_FUNCTION            2
             12 POP_TOP

  9          14 LOAD_GLOBAL              0 (print)
             16 LOAD_CONST               3 ('var')
             18 LOAD_DEREF               0 (var)
             20 CALL_FUNCTION            2
             22 POP_TOP
 10          24 LOAD_GLOBAL              0 (print)
             26 LOAD_CONST               4 ('global_var')
             28 CALL_FUNCTION            1
             30 POP_TOP

 11          32 LOAD_GLOBAL              0 (print)
             34 LOAD_CONST               5 ('func')
             36 LOAD_GLOBAL              1 (func)
             38 CALL_FUNCTION            2
             40 POP_TOP
             42 LOAD_CONST               0 (None)
             44 RETURN_VALUE
```

LEXING / TOKENIZING.

In [0]:
```
b = 6
def f1(a):
    print(a)
    print(b)
```

In [0]:
```
from dis import dis
dis(f1)
```

```
  3           0 LOAD_GLOBAL              0 (print)
              2 LOAD_FAST                0 (a)
              4 CALL_FUNCTION            1
              6 POP_TOP

  4           8 LOAD_GLOBAL              0 (print)
             10 LOAD_GLOBAL              1 (b)
             12 CALL_FUNCTION            1
             14 POP_TOP
             16 LOAD_CONST               0 (None)
             18 RETURN_VALUE
```

- Load global name print.
- Load local name a.
- Call print function with 1 positional argument.
- Load global name b.
- Load constant, in which case there None.

Посмотреть, что в области видимости

In [35]:
```
global_var = 0

def func():
    var = 'variable'

    def print_vars(arg):
        inner_var = 1
        print(locals()) # {'arg': 'argument', 'inner_var': 1}
        print(globals()) # {'__name__': '__main__', '__doc__' ..., 'glabal_var'
 : 0}

    print_vars('argument')

func()
```

```
{'arg': 'argument', 'inner_var': 1}
{'__name__': '__main__', '__doc__': 'Automatically created module for IPython
interactive environment', '__package__': None, '__loader__': None, '__spec_
_': None, '__builtin__': <module 'builtins' (built-in)>, '__builtins__': <mod
ule 'builtins' (built-in)>, '_ih': ['', 'def avg(first, *args):\n    print(ty
pe(args))\n    numbers = (first,) + *args\n    return sum(args)/len(args)',
'def avg(first, *args):\n    print(type(args))\n    numbers = (first,) + args
\n    return sum(args)/len(args)', 'def avg(first, *args):\n    print(type(ar
gs))\n    numbers = (first,) + args\n    return sum(args)/len(args)\n\navg
()', 'def avg(first, *args):\n    print(type(args))\n    numbers = (first,) +
args\n    return sum(args)/len(args)\navg(1, 2, 4)\navg()', "def avg_with_kwa
rgs(first, *args, **kwargs):\n    numbers = (first,) + args\n    sum(number
s)/len(numbers)\n    if kwargs.get('print', False):\n        print('Some very
informative print telling us that return value is', \n              numbers)
\n    return numbers\n       ", "def avg_with_kwargs(first, *args, **kwarg
s):\n    numbers = (first,) + args\n    sum(numbers)/len(numbers)\n    if kwa
rgs.get('print', False):\n        print('Some very informative print telling
us that return value is', \n              numbers)\n    return numbers\n
\nprint(avg_with_kwargs(1, 10, 100))", "def avg_with_kwargs(first, *args, **k
wargs):\n    numbers = (first,) + args\n    sum(numbers)/len(numbers)\n    if
kwargs.get('do_print', False):\n        print('Some very informative print te
lling us that return value is', \n              numbers)\n    return numbers
```

```
\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs(1, 10,
100))\nsettings = {'do_print', True}\nprint(avg_with_kwargs(1, 10, 100, **set
tings))", "def avg_with_kwargs(first, *args, **kwargs):\n    numbers = (firs
t,) + args\n    sum(numbers)/len(numbers)\n    if kwargs.get('do_print', Fals
e):\n        print('Some very informative print telling us that return value
is', \n              numbers)\n    return numbers\n        \nprint(avg_with_k
wargs(1, 10, 100))\nprint(avg_with_kwargs(1, 10, 100))\nsettings = {'do_prin
t': True}\nprint(avg_with_kwargs(1, 10, 100, **settings))", "def avg_with_kwa
rgs(first, *args, **kwargs):\n    numbers = (first,) + args\n    res = sum(nu
mbers)/len(numbers)\n    if kwargs.get('do_print', False):\n        print('So
me very informative print telling us that return value is', res)\n    return
numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs
(1, 10, 100))\nsettings = {'do_print': True}\nprint(avg_with_kwargs(1, 10, 10
0, **settings))", "def avg_with_kwargs(first, *args, **kwargs):\n    numbers
= (first,) + args\n    res = sum(numbers)/len(numbers)\n    if kwargs.get('do
_print', False):\n        print('Some very informative print telling us that
return value is', res)\n    return numbers\n        \nprint(avg_with_kwargs
(1, 10, 100))\nprint(avg_with_kwargs(1, 10, 100), **{'do_print': True})\nsett
ings = {'do_print': True}\nprint(avg_with_kwargs(1, 10, 100, **settings))",
"def avg_with_kwargs(first, *args, **kwargs):\n    numbers = (first,) + args
\n    res = sum(numbers)/len(numbers)\n    if kwargs.get('do_print', Fals
e):\n        print('Some very informative print telling us that return value
is', res)\n    return numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\n
print(avg_with_kwargs(1, 10, 100, **{'do_print': True})\nsettings = {'do_prin
t': True}\nprint(avg_with_kwargs(1, 10, 100, **settings))", "def avg_with_kwa
rgs(first, *args, **kwargs):\n    numbers = (first,) + args\n    res = sum(nu
mbers)/len(numbers)\n    if kwargs.get('do_print', False):\n        print('So
me very informative print telling us that return value is', res)\n    return
numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs
(1, 10, 100, **{'do_print': True}))\nsettings = {'do_print': True}\nprint(avg
_with_kwargs(1, 10, 100, **settings))", "def foo():\n    return 'moo'\n\nfoo.
attr = 'foo'\nfoo.attr", 'def foo(): return foo.__name__\n\nfoo()', "def spam
():\n    eggs = 'spam and eggs'    \n    def cantine():\n        print(eggs)
\nspam()", "def spam():\n    eggs = 'spam and eggs'\n    print(eggs)\n \nspam
()       # spam and eggs\nprint(eggs)  # raises a NameError exception", "def
spam():\n    print(eggs)\n \neggs = 'spam and eggs'\nspam()  # spam and egg
s", "def spam():\n    eggs = 'spam and eggs'    \n    def cantine():\n
print(eggs)\nspam()", "def spam():\n    eggs = 'spam and eggs'    \n    def c
```

```
antine():\n        print(eggs)\n    cantine()\n    \nspam()", 'class Meal:\n
def __init__(self):\n        self.eggs = 2\n \n \nmy_meal = Meal()\nprint(my_
meal.eggs)    # 2\nprint(eggs)            # raises a NameError exception', 'd
el eggs', 'class Meal:\n    def __init__(self):\n        self.eggs = 2\n \n
\nmy_meal = Meal()\nprint(my_meal.eggs)    # 2\nprint(eggs)            # rais
es a NameError exception', "glabal_var = 0\n\ndef fucn(arg):\n    var = 'loca
l variable'\n    \n    def print_vars():\n        inner_var = ", "glabal_var
= 0\n\ndef fucn(arg):\n    var = 'variable'\n    \n    def print_vars():\n
inner_var = 1 \n        print('inner_var', inner_var) # local\n        print
('var', var) # enclosing\n        print('global_var') # global\n        print
('func', func)\n    print_vars()", "glabal_var = 0\n\ndef fucn(arg):\n    var
= 'variable'\n    \n    def print_vars():\n        inner_var = 1 \n        pr
int('inner_var', inner_var) # local\n        print('var', var) # enclosing\n
print('global_var') # global\n        print('func', func)\n    print_vars()\n
func()", "glabal_var = 0\n\ndef func(arg):\n    var = 'variable'\n    \n    d
ef print_vars():\n        inner_var = 1 \n        print('inner_var', inner_va
r) # local\n        print('var', var) # enclosing\n        print('global_va
r') # global\n        print('func', func)\n    print_vars()\n    \nfunc()",
"glabal_var = 0\n\ndef func():\n    var = 'variable'\n    \n    def print_var
s():\n        inner_var = 1 \n        print('inner_var', inner_var) # local\n
print('var', var) # enclosing\n        print('global_var') # global\n
print('func', func)\n    print_vars()\n    \nfunc()", 'dir(my_meal)', "glabal
_var = 0\n\ndef func():\n    var = 'variable'\n    \n    def print_vars():\n
inner_var = 1 \n        print('inner_var', inner_var) # local\n        print
('var', var) # enclosing\n        print('global_var') # global\n        print
('func', func)\n    print_vars()\n    \nfunc()", 'from dis import dis\ndis(fu
nc)', 'def f():\n    print(i)\n\nfor i in range(5):\n    f()', "var = 'SUPE
R'\ndef foo():\n    var += ' PUPER'\n    return var\nfoo()", "glabal_var = 0
\n\ndef func():\n    var = 'variable'\n    \n    def print_vars():\n        i
nner_var = 1 \n        print(locals())\n        print(globals())\n    print_v
ars()\n\nfunc()", "glabal_var = 0\n\ndef func():\n    var = 'variable'\n
\n    def print_vars(arg):\n        inner_var = 1 \n        print(locals()) #
\n        print(globals())\n    print_vars()\n\nfunc()", "glabal_var = 0\n\nd
ef func():\n    var = 'variable'\n    \n    def print_vars(arg):\n        inn
er_var = 1 \n        print(locals()) # \n        print(globals())\n        \n
print_vars('argument')\n\nfunc()"], '_oh': {13: 'foo', 14: 'foo', 28: ['__cla
ss__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format_
_', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_s
```

```
ubclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce_
_', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__', '__s
ubclasshook__', '__weakref__', 'eggs']}, '_dh': ['/home/andrey/python_cours
e'], 'In': ['', 'def avg(first, *args):\n    print(type(args))\n    numbers =
(first,) + *args\n    return sum(args)/len(args)', 'def avg(first, *args):\n
print(type(args))\n    numbers = (first,) + args\n    return sum(args)/len(ar
gs)', 'def avg(first, *args):\n    print(type(args))\n    numbers = (first,)
+ args\n    return sum(args)/len(args)\n\navg()', 'def avg(first, *args):\n
print(type(args))\n    numbers = (first,) + args\n    return sum(args)/len(ar
gs)\navg(1, 2, 4)\navg()', "def avg_with_kwargs(first, *args, **kwargs):\n
numbers = (first,) + args\n    sum(numbers)/len(numbers)\n    if kwargs.get
('print', False):\n        print('Some very informative print telling us that
return value is', \n              numbers)\n    return numbers\n        ", "d
ef avg_with_kwargs(first, *args, **kwargs):\n    numbers = (first,) + args\n
sum(numbers)/len(numbers)\n    if kwargs.get('print', False):\n        print
('Some very informative print telling us that return value is', \n
numbers)\n    return numbers\n        \nprint(avg_with_kwargs(1, 10, 100))",
"def avg_with_kwargs(first, *args, **kwargs):\n    numbers = (first,) + args
\n    sum(numbers)/len(numbers)\n    if kwargs.get('do_print', False):\n
print('Some very informative print telling us that return value is', \n
numbers)\n    return numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\np
rint(avg_with_kwargs(1, 10, 100))\nsettings = {'do_print', True}\nprint(avg_w
ith_kwargs(1, 10, 100, **settings))", "def avg_with_kwargs(first, *args, **kw
args):\n    numbers = (first,) + args\n    sum(numbers)/len(numbers)\n    if
kwargs.get('do_print', False):\n        print('Some very informative print te
lling us that return value is', \n              numbers)\n    return numbers
\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs(1, 10,
100))\nsettings = {'do_print': True}\nprint(avg_with_kwargs(1, 10, 100, **set
tings))", "def avg_with_kwargs(first, *args, **kwargs):\n    numbers = (firs
t,) + args\n    res = sum(numbers)/len(numbers)\n    if kwargs.get('do_prin
t', False):\n        print('Some very informative print telling us that retur
n value is', res)\n    return numbers\n        \nprint(avg_with_kwargs(1, 10,
100))\nprint(avg_with_kwargs(1, 10, 100))\nsettings = {'do_print': True}\npri
nt(avg_with_kwargs(1, 10, 100, **settings))", "def avg_with_kwargs(first, *ar
gs, **kwargs):\n    numbers = (first,) + args\n    res = sum(numbers)/len(num
bers)\n    if kwargs.get('do_print', False):\n        print('Some very inform
ative print telling us that return value is', res)\n    return numbers\n
\nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs(1, 10, 100), **
{'do_print': True})\nsettings = {'do_print': True}\nprint(avg_with_kwargs(1,
10, 100, **settings))", "def avg_with_kwargs(first, *args, **kwargs):\n    nu
mbers = (first,) + args\n    res = sum(numbers)/len(numbers)\n    if kwargs.g
et('do_print', False):\n        print('Some very informative print telling us
that return value is', res)\n    return numbers\n       \nprint(avg_with_kwa
rgs(1, 10, 100))\nprint(avg_with_kwargs(1, 10, 100, **{'do_print': True})\nse
ttings = {'do_print': True}\nprint(avg_with_kwargs(1, 10, 100, **settings))",
"def avg_with_kwargs(first, *args, **kwargs):\n    numbers = (first,) + args
\n    res = sum(numbers)/len(numbers)\n    if kwargs.get('do_print', Fals
e):\n        print('Some very informative print telling us that return value
is', res)\n    return numbers\n       \nprint(avg_with_kwargs(1, 10, 100))\n
print(avg_with_kwargs(1, 10, 100, **{'do_print': True}))\nsettings = {'do_pri
nt': True}\nprint(avg_with_kwargs(1, 10, 100, **settings))", "def foo():\n
return 'moo'\n\nfoo.attr = 'foo'\nfoo.attr", 'def foo(): return foo.__name__
\n\nfoo()', "def spam():\n    eggs = 'spam and eggs'    \n    def cantine
():\n        print(eggs)\nspam()", "def spam():\n    eggs = 'spam and eggs'\n
print(eggs)\n \nspam()       # spam and eggs\nprint(eggs)  # raises a NameErr
or exception", "def spam():\n    print(eggs)\n \neggs = 'spam and eggs'\nspam
()  # spam and eggs", "def spam():\n    eggs = 'spam and eggs'    \n    def c
antine():\n        print(eggs)\nspam()", "def spam():\n    eggs = 'spam and e
ggs'    \n    def cantine():\n        print(eggs)\n    cantine()\n    \nspam
()", 'class Meal:\n    def __init__(self):\n        self.eggs = 2\n \n \nmy_m
eal = Meal()\nprint(my_meal.eggs)    # 2\nprint(eggs)            # raises a N
ameError exception', 'del eggs', 'class Meal:\n    def __init__(self):\n
self.eggs = 2\n \n \nmy_meal = Meal()\nprint(my_meal.eggs)    # 2\nprint(egg
s)            # raises a NameError exception', "glabal_var = 0\n\ndef fucn(ar
g):\n    var = 'local variable'\n    \n    def print_vars():\n        inner_v
ar = ", "glabal_var = 0\n\ndef fucn(arg):\n    var = 'variable'\n    \n    de
f print_vars():\n        inner_var = 1 \n        print('inner_var', inner_va
r) # local\n        print('var', var) # enclosing\n        print('global_va
r') # global\n        print('func', func)\n    print_vars()", "glabal_var = 0
\n\ndef fucn(arg):\n    var = 'variable'\n    \n    def print_vars():\n
inner_var = 1 \n        print('inner_var', inner_var) # local\n        print
('var', var) # enclosing\n        print('global_var') # global\n        print
('func', func)\n    print_vars()\nfunc()", "glabal_var = 0\n\ndef func(ar
g):\n    var = 'variable'\n    \n    def print_vars():\n        inner_var = 1
\n        print('inner_var', inner_var) # local\n        print('var', var) #
enclosing\n        print('global_var') # global\n        print('func', func)
```

```
\n    print_vars()\n    \nfunc()", "glabal_var = 0\n\ndef func():\n    var = 
'variable'\n    \n    def print_vars():\n        inner_var = 1 \n        prin
t('inner_var', inner_var) # local\n        print('var', var) # enclosing\n    
print('global_var') # global\n        print('func', func)\n    print_vars()\n
\nfunc()", 'dir(my_meal)', "glabal_var = 0\n\ndef func():\n    var = 'variabl
e'\n    \n    def print_vars():\n        inner_var = 1 \n        print('inner
_var', inner_var) # local\n        print('var', var) # enclosing\n        pri
nt('global_var') # global\n        print('func', func)\n    print_vars()\n   
\nfunc()", 'from dis import dis\ndis(func)', 'def f():\n    print(i)\n\nfor i
in range(5):\n    f()', "var = 'SUPER'\ndef foo():\n    var += ' PUPER'\n    
return var\nfoo()", "glabal_var = 0\n\ndef func():\n    var = 'variable'\n   
\n    def print_vars():\n        inner_var = 1 \n        print(locals())\n   
print(globals())\n    print_vars()\n\nfunc()", "glabal_var = 0\n\ndef func
():\n    var = 'variable'\n    \n    def print_vars(arg):\n        inner_var 
= 1 \n        print(locals()) # \n        print(globals())\n    print_vars()
\n\nfunc()", "glabal_var = 0\n\ndef func():\n    var = 'variable'\n    \n    
def print_vars(arg):\n        inner_var = 1 \n        print(locals()) # \n   
print(globals())\n        \n    print_vars('argument')\n\nfunc()"], 'Out': {1
3: 'foo', 14: 'foo', 28: ['__class__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__gt__', '_
_hash__', '__init__', '__init_subclass__', '__le__', '__lt__', '__module__',
'__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr_
_', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'eggs']}, 'ge
t_ipython': <bound method InteractiveShell.get_ipython of <ipykernel.zmqshel
l.ZMQInteractiveShell object at 0x7fe4aa2cc9b0>>, 'exit': <IPython.core.autoc
all.ZMQExitAutocall object at 0x7fe4a709bf98>, 'quit': <IPython.core.autocal
l.ZMQExitAutocall object at 0x7fe4a709bf98>, '_': ['__class__', '__delattr_
_', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__ge
tattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le_
_', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex_
_', '__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__',
'__weakref__', 'eggs'], '__': 'foo', '___': 'foo', '_i': "glabal_var = 0\n\nd
ef func():\n    var = 'variable'\n    \n    def print_vars(arg):\n        inn
er_var = 1 \n        print(locals()) # \n        print(globals())\n    print_
vars()\n\nfunc()", '_ii': "glabal_var = 0\n\ndef func():\n    var = 'variabl
e'\n    \n    def print_vars():\n        inner_var = 1 \n        print(locals
())\n        print(globals())\n    print_vars()\n\nfunc()", '_iii': "var = 'S
UPER'\ndef foo():\n    var += ' PUPER'\n    return var\nfoo()", '_i1': 'def a
```

```
vg(first, *args):\n    print(type(args))\n    numbers = (first,) + *args\n    
return sum(args)/len(args)', '_i2': 'def avg(first, *args):\n    print(type(a
rgs))\n    numbers = (first,) + args\n    return sum(args)/len(args)', 'avg': 
<function avg at 0x7fe4a4718f28>, '_i3': 'def avg(first, *args):\n    print(t
ype(args))\n    numbers = (first,) + args\n    return sum(args)/len(args)\n\n
avg()', '_i4': 'def avg(first, *args):\n    print(type(args))\n    numbers = 
(first,) + args\n    return sum(args)/len(args)\navg(1, 2, 4)\navg()', '_i5': 
"def avg_with_kwargs(first, *args, **kwargs):\n    numbers = (first,) + args
\n    sum(numbers)/len(numbers)\n    if kwargs.get('print', False):\n        
print('Some very informative print telling us that return value is', \n        
numbers)\n    return numbers\n        ", 'avg_with_kwargs': <function avg_wit
h_kwargs at 0x7fe4a45d3a60>, '_i6': "def avg_with_kwargs(first, *args, **kwar
gs):\n    numbers = (first,) + args\n    sum(numbers)/len(numbers)\n    if kw
args.get('print', False):\n        print('Some very informative print telling 
us that return value is', \n              numbers)\n    return numbers\n      
\nprint(avg_with_kwargs(1, 10, 100))", '_i7': "def avg_with_kwargs(first, *ar
gs, **kwargs):\n    numbers = (first,) + args\n    sum(numbers)/len(numbers)
\n    if kwargs.get('do_print', False):\n        print('Some very informative 
print telling us that return value is', \n              numbers)\n    return 
numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs
(1, 10, 100))\nsettings = {'do_print', True}\nprint(avg_with_kwargs(1, 10, 10
0, **settings))", 'settings': {'do_print': True}, '_i8': "def avg_with_kwargs
(first, *args, **kwargs):\n    numbers = (first,) + args\n    sum(numbers)/le
n(numbers)\n    if kwargs.get('do_print', False):\n        print('Some very i
nformative print telling us that return value is', \n              numbers)\n
return numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_
kwargs(1, 10, 100))\nsettings = {'do_print': True}\nprint(avg_with_kwargs(1, 
10, 100, **settings))", '_i9': "def avg_with_kwargs(first, *args, **kwarg
s):\n    numbers = (first,) + args\n    res = sum(numbers)/len(numbers)\n    
if kwargs.get('do_print', False):\n        print('Some very informative print 
telling us that return value is', res)\n    return numbers\n        \nprint(a
vg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs(1, 10, 100))\nsettings = 
{'do_print': True}\nprint(avg_with_kwargs(1, 10, 100, **settings))", '_i10': 
"def avg_with_kwargs(first, *args, **kwargs):\n    numbers = (first,) + args
\n    res = sum(numbers)/len(numbers)\n    if kwargs.get('do_print', Fals
e):\n        print('Some very informative print telling us that return value 
is', res)\n    return numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\n
print(avg_with_kwargs(1, 10, 100), **{'do_print': True})\nsettings = {'do_pri
```

```
nt': True}\nprint(avg_with_kwargs(1, 10, 100, **settings))", '_i11': "def avg
_with_kwargs(first, *args, **kwargs):\n    numbers = (first,) + args\n    res
= sum(numbers)/len(numbers)\n    if kwargs.get('do_print', False):\n        p
rint('Some very informative print telling us that return value is', res)\n
return numbers\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_
kwargs(1, 10, 100, **{'do_print': True})\nsettings = {'do_print': True}\nprin
t(avg_with_kwargs(1, 10, 100, **settings))", '_i12': "def avg_with_kwargs(fir
st, *args, **kwargs):\n    numbers = (first,) + args\n    res = sum(numbers)/
len(numbers)\n    if kwargs.get('do_print', False):\n        print('Some very
informative print telling us that return value is', res)\n    return numbers
\n        \nprint(avg_with_kwargs(1, 10, 100))\nprint(avg_with_kwargs(1, 10,
100, **{'do_print': True}))\nsettings = {'do_print': True}\nprint(avg_with_kw
args(1, 10, 100, **settings))", '_i13': "def foo():\n    return 'moo'\n\nfoo.
attr = 'foo'\nfoo.attr", 'foo': <function foo at 0x7fe4a44b1730>, '_13': 'fo
o', '_i14': 'def foo(): return foo.__name__\n\nfoo()', '_14': 'foo', '_i15':
"def spam():\n    eggs = 'spam and eggs'    \n    def cantine():\n        pri
nt(eggs)\nspam()", 'spam': <function spam at 0x7fe4a4786c80>, '_i16': "def sp
am():\n    eggs = 'spam and eggs'\n    print(eggs)\n \nspam()       # spam an
d eggs\nprint(eggs)  # raises a NameError exception", '_i17': "def spam():\n
print(eggs)\n \neggs = 'spam and eggs'\nspam()  # spam and eggs", '_i18': "de
f spam():\n    eggs = 'spam and eggs'    \n    def cantine():\n        print
(eggs)\nspam()", '_i19': "def spam():\n    eggs = 'spam and eggs'    \n    de
f cantine():\n        print(eggs)\n    cantine()\n    \nspam()", '_i20': 'cla
ss Meal:\n    def __init__(self):\n        self.eggs = 2\n \n \nmy_meal = Mea
l()\nprint(my_meal.eggs)    # 2\nprint(eggs)           # raises a NameError
exception', 'Meal': <class '__main__.Meal'>, 'my_meal': <__main__.Meal object
at 0x7fe4a4336198>, '_i21': 'del eggs', '_i22': 'class Meal:\n    def __init_
_(self):\n        self.eggs = 2\n \n \nmy_meal = Meal()\nprint(my_meal.eggs)
# 2\nprint(eggs)            # raises a NameError exception', '_i23': "glabal_
var = 0\n\ndef fucn(arg):\n    var = 'local variable'\n    \n    def print_va
rs():\n        inner_var = ", '_i24': "glabal_var = 0\n\ndef fucn(arg):\n
var = 'variable'\n    \n    def print_vars():\n        inner_var = 1 \n
print('inner_var', inner_var) # local\n        print('var', var) # enclosing
\n        print('global_var') # global\n        print('func', func)\n    prin
t_vars()", 'glabal_var': 0, 'fucn': <function fucn at 0x7fe4a44448c8>, '_i2
5': "glabal_var = 0\n\ndef fucn(arg):\n    var = 'variable'\n    \n    def pr
int_vars():\n        inner_var = 1 \n        print('inner_var', inner_var) #
local\n        print('var', var) # enclosing\n        print('global_var') # g
```

Функции в Python могут использовать переменные, определенные во внешних областях видимости.

Важно помнить, что поиск переменных осуществляется во время исполнения функции, а не во время её объявления.

In [2]:
```
def f():
    print(i)

for f in range(5):
    f()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-2-f9e1cabb27b7> in <module>
      3 
      4 for f in range(5):
----> 5     f()

TypeError: 'int' object is not callable
```

```
In [3]: global_var = 0

        def func():
            global_var = 1

        print(global_var)
        func()
        print(global_var)
```

```
0
0
```

Для присваивания правило LEGB не работает

In [39]:
```
global_var = 0

def foo():
    global_var = global_var + 1

print(global_var)
foo()
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-39-2553de2cc291> in <module>
      5 
      6     print(global_var)
----> 7 foo()

<ipython-input-39-2553de2cc291> in foo()
      2 
      3 def foo():
----> 4     global_var = global_var + 1
      5 
      6     print(global_var)

UnboundLocalError: local variable 'global_var' referenced before assignment
```

Изменить стандартное поведение можно с помощью операторов nonlocal и global

global

Чтобы присвоить некоторое значение переменной, определённой на высшем уровне

программы, нужно воспользоваться оператором **global**.

```
In [44]:
global_var = 0

def foo():
    global global_var
    global_var = global_var + 1

print(global_var)
foo()
print(global_var)
```

```
0
1
```

nonlocal

Nonlocal namespace – объявление функции внутри другой функции.

Чтобы присвоить новое значение переменной объявленной в функции выше, используем оператор nonlocal

```
In [47]: def f1():
             a = 1
             b = 2
             def inner():
                 nonlocal a
                 a = a + b

             inner()
             print('local a is', a)
         f1()
```

```
local a is 3
```

Что нужно запомнить

1. В Python четыре области видимости: встроенная, глобальная, объемлющая и локальная.

2. Правило LEGB: поиск имени осуществляется от локальной к встроенной. При использовании операции присваивания имя считается локальным.

3. Это поведение можно изменить с помощью операторов global и nonlocal.

Function annotation

```
In [5]:
from typing import Union

def is_palindrome(s: Union[str, int], variant: int) -> bool:
    if variant == 1:
        return s == ''.join(reversed(s))
    if variant == 2:
        return s == s[::-1]
    return 'coose variant'

print(is_palindrome('madam', 1))
print(is_palindrome('madam', 2))
```

```
True
True
```

Naming и кое что еще

- Функция должна делать только одну вещь (логически)
- Функция-простыня на три экрана - БЕДА.
- Имя функции должно максимально коротко отражать то, что она делает
- Лучше длинно, но содержательно, чем коротко и туманно
- Лучше строить имя функции от глагола