Python

Тема 4. Функции

Функции. Определение

```
def function_name1(arg1, arg2, arg3, ...argN):
  statements

def function_name2(arg1, arg2, arg3, ... argN):
  statements
  return result
```

Функции. Определение

- def - исполняемый код
- Определяет именованный объект функции

```
x = 2

if x % 2 != 0:
    def func():
        print("odds")
else:
    def func():
        print("even")


func()

foo = func
foo()
```

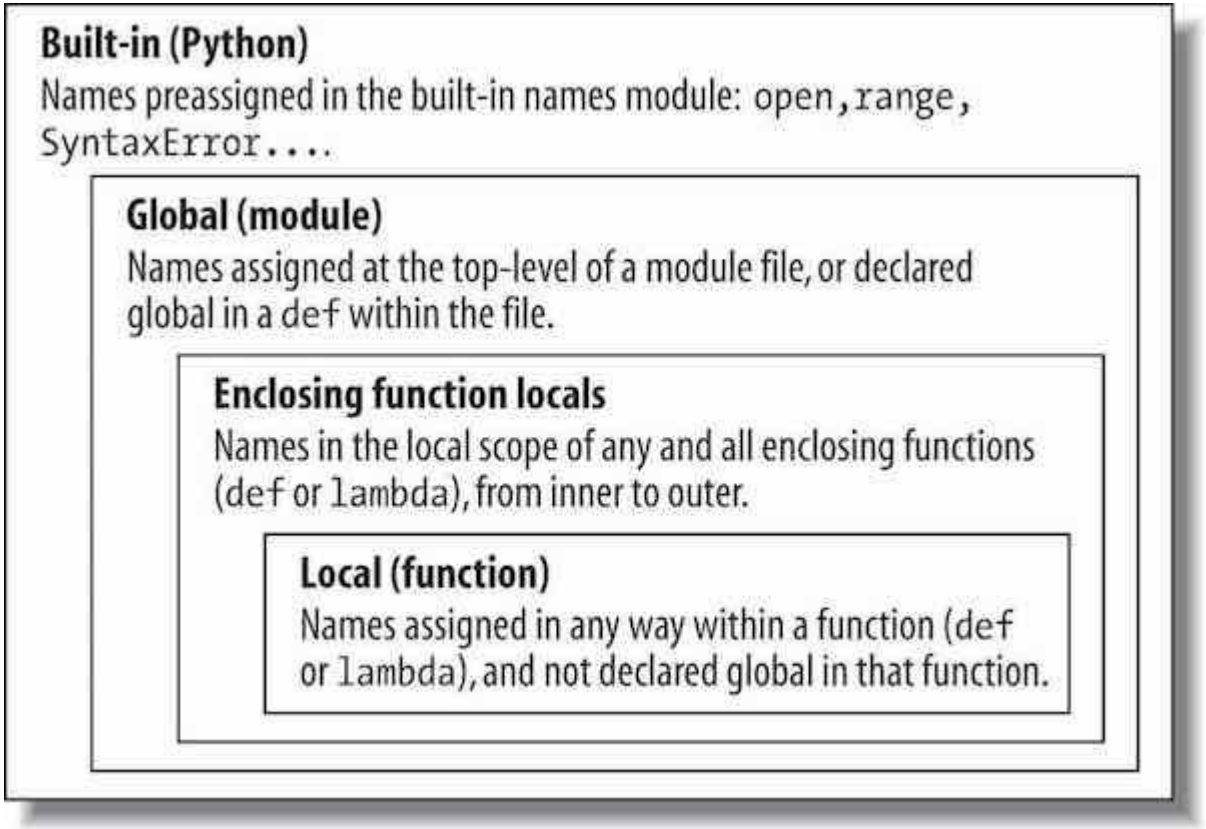
Функции. Полиморфизм

```
def twice(value):
    return value * 2


x = twice(3)
print(x)

x = twice(3.14)
print(x)

x = twice("ITMO")
print(x)

x = twice([1,2,3,4,5])
print(x)
```

Функции. Полиморфизм

```
def intersect(seq1, seq2):
    result = []
    for x in seq1:
        if x in seq2:
            result.append(x)
    return result


print(intersect([1, 2, 3, 4, 5], [3, 4, 5, 6]))

print(intersect((1, 2, 3), [3, 4, 5, 6]))

print(intersect({1: 'a', 2: 'b', 3: 'c', 4: 'd'}, [3, 4, 5, 6]))

print(intersect(5, [3, 4, 5, 6]))  # error
```

Функции. LEGB



Функции.LEGB

```
import builtins

print(dir(builtins))    # built-in

x = "Global variable"  # Global

def func():
    x = "Enclosing variable"  # Enclosing

    def subfunc():
        x = "Local variable"  # Local
        print(x)

    subfunc()


func()
```

Функции. LEGB

```
import time

print(time.localtime())
print(time)

time = "20:00"

print(time)


def fun1():
    print(time)
```

```
def fun2():
    time = 10
    print(time)


def fun3(time):
    print(time)


fun1()
fun2()
fun3()

print(time)
```

Функции. Лямбда-выражения (анонимные)

```
def increment(x):
    return x + 1


l = list(range(10))

print(list(map(increment, l)))
print(list(map(lambda x: x + 1, l)))

l = ['aaaa', 'aBb', "bB"]

l.sort()
print(l)
l.sort(key=lambda x: len(x))
print(l)
```

Функции. Global

```
x = 239

def fun1():
    x = "ITMO"
    print(x)

def fun2():
    global x
    x = [1, 2, 3, 4]
    print(x)

print(x)
fun1()
print(x)
fun2()
print(x)
```

Функции. Вложенные функции

```
def power_fabric(N):
    def execute(x):
        return x ** N
    return execute

pow2 = power_fabric( 2)
print(pow2(10))

pow10 = power_fabric( 10)
print(pow10(2))

def power_fabric_ex(N):
    return lambda x: x ** N

pow3 = power_fabric_ex( 3)
print(pow3(3))
```

Функции. **Nonlocal**

```
def pow_fabric(N):
    counter = 0

    def action(x):
        nonlocal counter
        counter += 1
        print(counter)
        return x ** N

    return action

pow2 = pow_fabric(2)
pow3 = pow_fabric(3)

pow2(2)
pow2(2)
pow3(2)
```

Функции. Аргументы

```
def func(x):
    x = 239
    print(239)


a = 10
s = "ITMO"

func(a)
print(a)

func(s)
print(s)
```

Функции. Аргументы

```
def func(L):
    L[0] = 1
    L[2] = 2
    L.append(10)
    print(L)


a = [7, 8, 9]
b = [7, 8, 9]

func(a)
print(a)
func(b[:])
print(b)
```

Функции. Значения по умолчанию

```
def fun1(x=100):
    print(x)

x = 99

def fun2(x=x):
    print(x)

def fun3():
    x = 101
    def internal_func(x=x):
        print(x)
    internal_func()

fun1()
fun2()
fun3()
```

Функции. Возвращаемое значение

```
def echo(text):
    print(text)


def increment(x):
    return x+1

def swap(x,y):
    return y,x

echo("Hello world")

print(increment(10))
print(swap(10,11))
```

Функции. Аргументы

```
def func(x, y, z):
    print(x, y, z)

def func2(*args):
    print(args)


def func3(**args):
    print(args)


func(1, 2, 3)
func(y=1, x=2, z=3)
func(1, z=2, y=3)

func2(1, 2, 3)
func3(a=1, b=2, c=3)
```

Функции. Аргументы

```
def min(first, *args):
    res = first
    for v in args:
        if v < res:
            res = v
    return res


print(min(7, 3, 4, 5, 1, 9))
print(min(1))
```

Функции. Аргументы

```
def min_ex(first, *args, **kargs):
    eq = lambda x, y: x < y
    if 'cmp' in kargs:
        eq = kargs['cmp']

    res = first
    for v in args:
        if eq(v, res):
            res = v
    return res


greater = lambda x, y: x > y
print(min_ex(7, 3, 4, 5, 1, 9, cmp=greater))
print(min_ex(7, 3, 4, 5, 1, 9))
```

Функции. Аргументы

```
def min_ex2(first, *args, cmp= lambda x, y: x < y):
    res = first
    for x in args:
        if cmp(x, res):
            res = x
    return res



greater = lambda x, y: x > y
print(min_ex2(7, 3, 4, 5, 1, 9, cmp=greater))
print(min_ex2(7, 3, 4, 5, 1, 9))
```

Функции. Рекурсия

```
def fib(n):
    if n <= 2:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)


def fib2(n):
    return 1 if n <= 2 else fib2(n - 1) + fib2(n - 2)



print(fib(20))
```

Функции. Рекурсия

```
import sys


def f(x):
    if x == 1:
        return 1
    else:
        return 1 + f(x - 1)


print(sys.getrecursionlimit())
sys.setrecursionlimit( 1002)

print(f(1000))
```

Функции.

```
def fib_ex(count):
    return [fib3(i+1) for i in range(count)]


def fib_ex2(count):
    F = [1,1]
    for i in range(2,count):
        F.append(F[i-1] + F[i-2])
    return F

print(fib_ex(20))
print(fib_ex2(20))
```

Функции. Рекурсия

```
def min_rec(first, *args):
    if not args:
        return first
    m = min_rec(args[ 0], *args[1:])
    return first if first < m else m


print(min_rec(5,9,5,7,3,8,9))
```

Функции.

```
def func(x, y, z):
    print(x, y, z)


print(dir(func))

print(func.__name__)
print(func.__code__)

print(dir(func.__code__))
print(func.__code__.co_varnames)
```

Функция. Генераторы

```
res1 = []

for i in range(10):
   res1.append(i** 2)

res2 = list(map(lambda x: x**2, range(10)))

res3 = [x** 2 for x in range(10)]

print(res1)
print(res2)
print(res3)
```

Функция. Генераторы

```
res1 = []

for i in range(10):
   if i%2 == 0:
       res1.append(i** 2)

res2 = list(map(lambda x: x**2, filter(lambda x: x%2==0, range(10))))

res3 = [x**2 for x in range(10) if x%2==0]

print(res1)
print(res2)
print(res3)
```

Функция. Генераторы

```
res = (x ** 2 for x in range(1000))

for i in res:
    print(i)


def generquares(N):
    for n in range(N):
        yield n ** 2

G = generquares(1000)
for i in G:
    print(i)
```

Функция. Генераторы

```
def pow2Generator(N):
    for i in range(N): yield i
    # for i in (x ** 2 for x in range(N)): yield i
    yield from (x ** 2 for x in range(N))


def powGenerator(N, M):
    # for p in range(1, M + 1):
    #     for i in range(N):
    #         yield i ** p

    for p in range(1, M + 1):
        yield from (x ** p for x in range(N))

print(list(pow2Generator(3)))
print(list(powGenerator(3, 3)))
```

Функции. Производительность

```
import time

def funcBenchmark(func, *args, repeat= 1000):
    start = time.clock()
    for i in range(repeat):
        func(*args)
    finish = time.clock()
    return finish - start

def simpleFunc(N):
    res = []
    for i in range(N):
        res.append(i** 2)
    return res

def listFunc(N):
    return [i**2 for i in range(N)]
```

Функции. Производительность

```
def mapFunc(N):
    return list(map(lambda x: x**2, range(N)))

def genFunc(N):
    return list((x**2 for x in range(N)))

def yieldFunc(N):
    def internal():
        for i in range(N):
            yield i**2
    return list(internal())


for f in (simpleFunc, listFunc, mapFunc, genFunc, yieldFunc):
    print(f.__name__, funcBenchmark(f, 10000))
```