《Python程序设计》

Python函数

刘潇
机械科学与工程学院

2023年10月23日

2023秋季

本节要点

- 掌握函数的定义和使用
- 理解命名空间和作用域
- 理解参数传递规则和匹配模式

主要内容

1. 定义和使用

2. 命名空间和作用域

3. 参数传递与匹配

4. 函数应用实例

为什么要用函数?

如何增加新同学ma6的信息?

学生数据库

```
1  a = ["zhang3", "U202011054"]
2  b = ["li4", "U202011055"]
3  c = ["wang5","U202011056"]
4  student = [a, b, c]
5  print(student)
6
7  # 增加新同学ma6的信息
8  d = ["ma6", "U202011057"]
9  student.append(d)
10 print(student)
```


```
7  # 增加新同学的信息
8  def addnewstudent(name, ID):
9      newstudent = [name, ID]
10     student.append(newstudent)
11     return student
12
13 print(addnewstudent("ma6","U202011057"))
14 print(addnewstudent("xiaoming", "U202011058"))
```

→ **定义函数**

→ **重复使用代码**

打印出2到200的斐波拉契数列

打印出2到20内的斐波那契数列 F(1) = 1, F(2) = 1, F(n) = F(n-1) + F(n-2) (n>=2 , n∈N*)

```
list1 = [1, 1]
i = len(list1)
for number in range(2, 20):
    if number == list1[i-1] + list1[i-2]:
        list1.append(number)
        i += 1
print(list1)
```

```
[1, 1, 2, 3, 5, 8, 13]
```

```
def fib_list(n):
    list1 = [1, 1]
    i = len(list1)
    for number in range(2, n):
        if number == list1[i-1] + list1[i-2]:
            list1.append(number)
            i += 1
    return list1

print(fib_list(20))
print(fib_list(200))
```

定义函数

重复使用代码

```
[1, 1, 2, 3, 5, 8, 13]
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

Python函数

- 函数是一种**可重复使用的功能代码块**，通过定义和调用函数可以减少程序中的重复代码

$$f(x) = x^2$$

标准的函数结构（def语句）

```
def 函数名（参数1，参数2，...）：
    "注释（文档字符串）"
    功能代码块
    ...
    功能代码块
    return 返回值1，返回值2，...
```

- 函数名遵循标识符命名规则
- 参数是函数的输入，返回值是函数的输出
- 函数遇到return语句则停止，以元组的形式按顺序返回
- 没有return相当于return None
- 注意函数内代码块和代码块内的缩进规则

f(x)=x*x函数定义

```
1  def square(x):
2      "计算x的平方"
3      y = x*x
4      return y
5
6  print(square(2))
7
8  # 查看注释
9  help(square)
10 print(square.__doc__)
```

```
4
Help on function square in module __main__:

square(x)
    计算x的平方

计算x的平方
```

square：函数名

x：形参

y：局部变量，返回值

2：实参

```
1  def square(x):
2      "计算x的平方"
3      y = x*x
4      return y,x
5
6  square(2)
```

```
(4, 2)
```

以元组返回

函数参数的动态性

函数参数、返回值、局部变量的类型不需要声明，具有动态性

参数的动态性

```
1  def times(x,y):
2      return x*y
3
4  times(2,4)
```

参数类型可变

```
8
```

```
1  times("hello",4)
```

```
'hellohellohellohello'
```

```
1  def intersect(seq1, seq2):
2      list1 = []
3      for x in seq1:
4          if x in seq2:
5              list1.append(x)
6      return list1
7
8  intersect([1,2,3,4], (1,2))
```

列表和元组混合类型

```
[1, 2]
```

函数调用

函数调用需要在定义函数之后！！！

函数调用

```
1 times(2,4)
2
3 def times(x,y):
4     return x*y
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-4-6004928d8b94> in <module>
----> 1 times(2,4)
      2
      3 def times(x,y):
      4     return x*y

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

定义函数后，函数名会出现在全局命名空间中

```
1 print(dir())
```

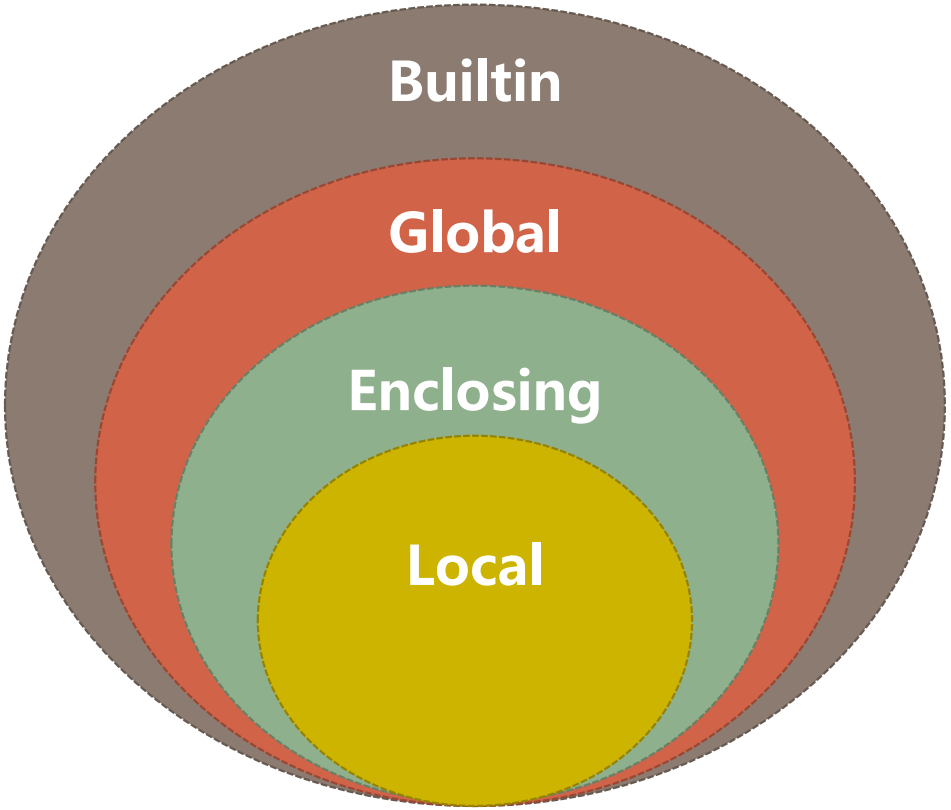
```
['In', 'Out', '_', '_5', '__', '___', '__builtin__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1', '_i2', '_i3', '_i4', '_i5', '_i6', '_ih', '_ii', '_iii', '_oh', 'exit', 'get_ipython', 'quit']
```

```
1 def times(x,y):
2     return x*y
3 print(dir())
```

```
['In', 'Out', '_', '_5', '__', '___', '__builtin__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', '_dh', '_i', '_i1', '_i2', '_i3', '_i4', '_i5', '_i6', '_i7', '_ih', '_ii', '_iii', '_oh', 'exit', 'get_ipython', 'quit', 'times']
```

命名空间

- 命名空间是名称到对象的映射，是有层次结构的标识符容器（字典），不同命名空间中的同名标识符不会冲突



Python中的4层命名空间

- Builtin：内置命名空间
- Global：函数定义所在的文件（模块）的全局命名空间
- Enclosing：外部嵌套函数的命名空间
- Local：函数内的命名空间

Builtin命名空间

内建命名空间在python解释器启动时创建，直到解释器退出时消失，__builtin__模块包含了内建命名空间中的成员（函数、变量、类等）

```
1 print(dir(__builtin__))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__IPYTHON__', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'filter', 'float', 'format', 'frozenset', 'get_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
```

dir()函数以列表的形式返回命名空间中名字（排序后）
vars()函数以字典的形式返回命名空间中成员的当前值

Global、Enclosing、Local命名空间

```
1  var1 = 1
2
3  def func():
4      var2 = 2
5      def innerfunc():
6          var3 = 3
7          print("local", var3)
8          print("enclosing function locals", var2)
9      innerfunc()
10     print("enclosing function locals", var2)
11
12 func()
13 print("global", var1)
14 print("local", var3)
```

Global命名空间

Enclosing命名空间

Local命名空间

函数执行完成后，Local命名空间消失

```
local 3
enclosing function locals 2
enclosing function locals 2
global 1

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-14-b5fb33048505> in <module>
     12 func()
     13 print("global", var1)
---> 14 print("local", var3)

NameError: name 'var3' is not defined
```

vars()查看当前命名空间

```
1  var1 = 1
2
3  def func():
4      var2 = 2
5      def innerfunc():
6          var3 = 3
7          print("local", var3)
8          # print("enclosing function locals", var2)
9          print(vars())
10     innerfunc()
11     print("enclosing function locals", var2)
12     print(vars())
13     # print("enclosing function locals", var3)
14
15 func()
16 print("global", var1)
```

只有var2

只有var3

```
local 3
{'var3': 3}
enclosing function locals 2
{'var2': 2, 'innerfunc': <function func.<locals>.innerfunc at 0x0000000005BF0C10>}
global 1
```

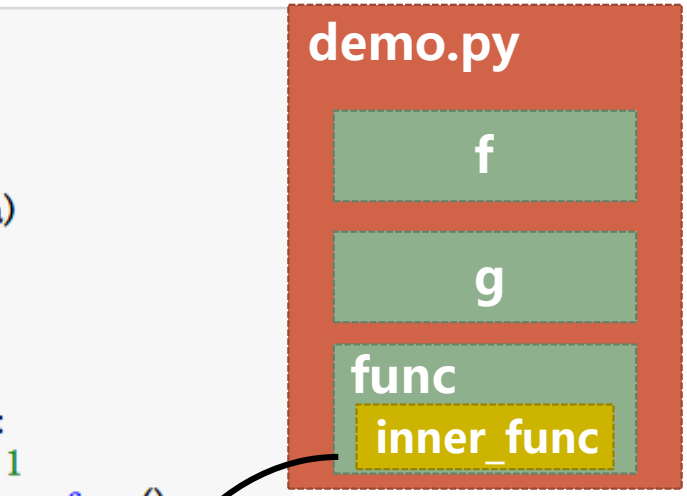
以字典的形式返回命名空间，表明变量与值的映射关系（引用）

LEGB规则

按照Local → Enclosing → Global → Builtin顺序寻找名字

```
1  a = 1
2
3  def f():
4      a = 2
5      print(a)
6
7  def g():
8      pass
9
10 def func():
11     name = 1
12     def inner_func():
13         print(a)
14     inner_func()
15
16 func()
```

```
1
```



sys	<module>
dir	<function>
...	...
len	<function>

B(builtin namespace)

a	1
f	<function>
f	<function>
func	<function>

G(global namespace)

name	1
inner_func	<function>

E(enclosing namespace)

L(local namespace)

LEGB规则

```python
x = 1

def func():
    x = 2
    def innerfunc():
        x = 3
        print("local", x)
    innerfunc()
    print("enclosing", x)

func()
print("global", x)
```

```
local 3
enclosing 2
global 1
```

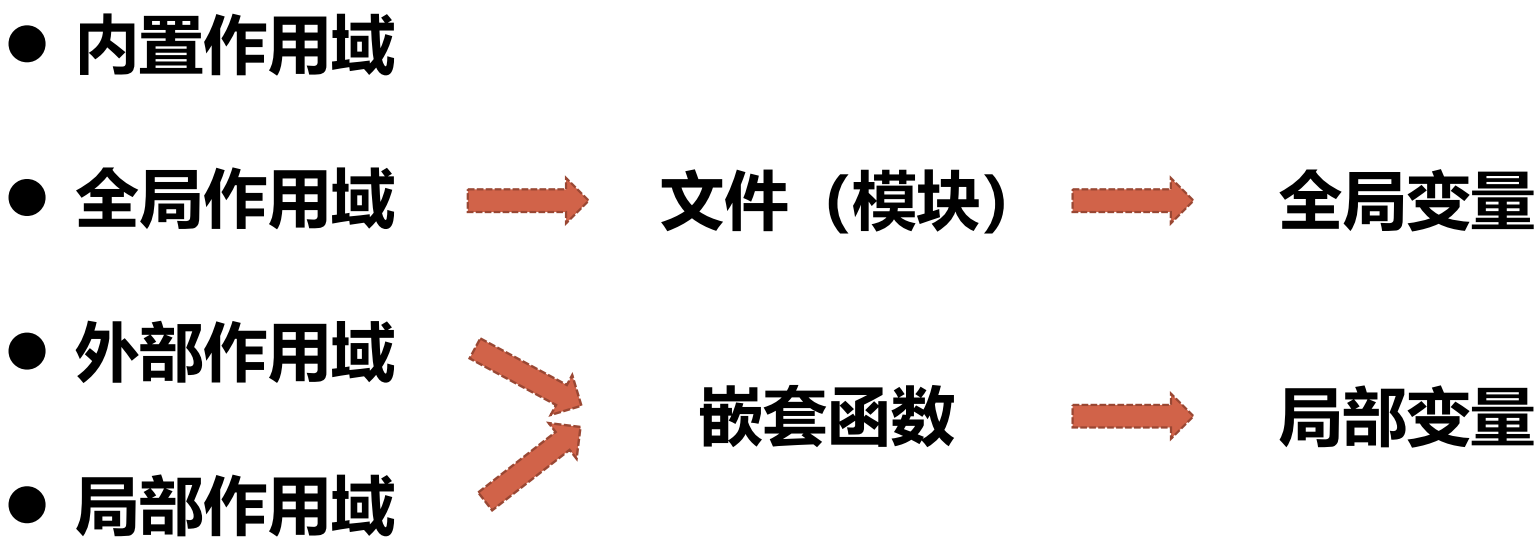
```python
x = 1

def func():
    x = 2
    def innerfunc():
        # x = 3
        print("local", x)
    innerfunc()
    print("enclosing", x)

func()
print("global", x)
```

```
local 2
enclosing 2
global 1
```

x = 3属于函数内部命名空间，当被注释后，函数innerfunc()通过print(“local”,x)使用x时，开始按照LEGB规则查找x变量

作用域

- 作用域是针对变量而言的，是程序中可以直接访问变量的代码区域，决定了在哪一部分程序可以访问哪个特定的变量



定义函数、模块和类会产生新的作用域，代码块如条件/循环语句不产生作用域

局部变量和全局变量

函数中赋值的名字默认是局部的，global语句把赋值的名字映射到全局作用域

```
x = 1 # 全局变量

def func():
    x = 2 # 局部变量
    def innerfunc():
        global x # 全局变量
        print(x)
    innerfunc()
    print("enclosing", x)

func()
print("global", x)
```

```
1
enclosing 2
global 1
```

```
x = 1 # 全局变量

def func():
    x = 2 # 局部变量
    def innerfunc():
        global x # 全局变量
        x = 3
        print(x)
    innerfunc()
    print("enclosing", x)

func()
print("global", x)
```

```
3
enclosing 2
global 3
```

globals()和locals()函数

globals()和locals()分别返回全局和局部命名空间字典

```
1  a = 1 # 全局变量
2
3  def func():
4      b = 2 # 局部变量
5      def innerfunc():
6          global c # 全局变量
7          c = 3
8          print("c" in globals(), "c" in locals())
9      innerfunc()
10     print("b" in locals())
11
12 func()
13 print("a" in globals(), "c" in globals())
14 print("b" in globals(), "b" in locals())
```

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

函数执行完后，局部命名空间消失

同名全局和局部变量引用

```
1  a = 1 # 全局变量
2
3  def func():
4      a = 2 # 相同名字的局部变量
5      print("a" in globals(), "a" in locals())
6      print(a, globals()["a"])
7
8  func()
```

```
True True
2 1
```

```
1  a = 1 # 全局变量
2
3  def func():
4      a = 2 # 相同名字的局部变量
5      print("a" in globals(), "a" in locals())
6      print(a, globals()["a"])
7      def innerfunc():
8          nonlocal a
9          a = 3 # 相同名字的局部变量
10     innerfunc()
11     print(a, globals()["a"])
12
13 func()
```

```
True True
2 1
3 1
```

globals()[“a”]

调用字典键为“a”的值

利用nonlocal语句调用enclosing作用域的同名变量a

局部变量静态检测

局部命名空间的检测是静态的

```
1  a = 1 # 全局变量
2
3  def func():
4      print(a)
5      a = 2
6
7  func()
```

如果函数内有赋值则a为局部变量，赋值必须在使用前！

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-26-e99371170e6c> in <module>
      5     a = 2
      6
----> 7 func()

<ipython-input-26-e99371170e6c> in func()
      2
      3 def func():
----> 4     print(a)
      5     a = 2
      6

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

局部变量静态检测

```
1  a = 1 # 全局变量
2
3  def func():
4      a = a + 1
5      print(a)
6
7  func()
```

如何修改?

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-27-0cbd633de724> in <module>
      5     print(a)
      6
----> 7 func()

<ipython-input-27-0cbd633de724> in func()
      2
      3 def func():
----> 4     a = a + 1
      5     print(a)
      6

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

函数返回函数（闭包）

在一个内部函数中，对外部作用域的变量进行引用，并且一般外部函数的返回值为内部函数，那么内部函数就被认为是闭包

```
1  def func():
2      list1 = []
3      def innerfunc(funcname):
4          list1.append(len(list1)+1)
5          print("{}\'s list1 = {}".format(funcname, list1))
6      return innerfunc
7
8  func1 = func()
9  func1("func1")
10 func1("func1")
11 func1("func1")
12 func2 = func()
13 func2("func2")
14 func2("func2")
15 func1("func1")
```

返回函数对象

闭包函数innerfunc引用的list1为自由变量，func函数执行完成后仍然存在

```
func1's list1 = [1]
func1's list1 = [1, 2]
func1's list1 = [1, 2, 3]
func2's list1 = [1]
func2's list1 = [1, 2]
func1's list1 = [1, 2, 3, 4]
```

自由变量的函数

闭包陷阱

```python
def func():
    func_list1 = []
    for i in range(3):
        def innerfunc(x):
            return x*i
        func_list1.append(innerfunc)
    return func_list1

func1 = func()
print(func1[0](2))
print(func1[1](2))
print(func1[2](2))
```

```
4
4
4
```

为什么都是4?

棋盘游戏

```python
# 棋盘游戏

def createpoint(startposition): # 落子
    pos = startposition
    def go(direction, step): # 走子
        new_x = pos[0] + direction[0]*step
        new_y = pos[1] + direction[1]*step
        pos[0] = new_x
        pos[1] = new_y
        return pos
    return go

point1 = createpoint([0,0]) # 落第一个棋子
print(point1([10,0], 1))
point2 = createpoint([10,10]) # 落第二个棋子
print(point2([5, 0], 1))
print(point1([0,10], 1))
print(point2([0, 5], 1))
print(point1([-1,0], 5))
```

```
[10, 0]
[15, 10]
[10, 10]
[15, 15]
[5, 10]
```

参数传递规则

- Python函数的参数传递遵循实参向形参的赋值规则（引用），形参是等待赋值的名字，无类型且是局部的

```python
def func(a, b):
    c = a
    a = b
    b = c
    print(a, b)

a = 1
b = 2
print(a, b)
func(a, b)
print(a, b)
```

```
1 2
2 1
1 2
```

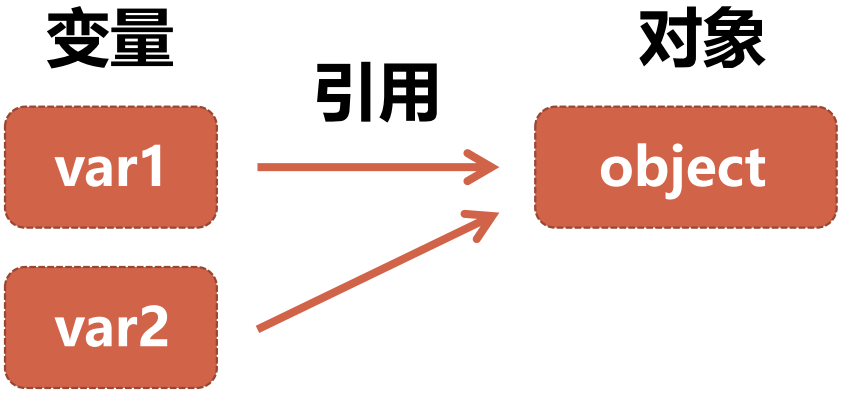
```python
def func(a, b):
    c = a[0]
    a[0] = b[0]
    b[0] = c
    print(a, b)

a = [1]
b = [2]
print(a, b)
func(a, b)
print(a, b)
```

```
[1] [2]
[2] [1]
[2] [1]
```

为什么结果会不一样?

赋值规则（第三节-变量与对象）

- Python解释器内存中有个对象池，所有对象都放在这个池子里。Python中对变量的赋值是**引用对象**的过程



变量指针指向具体对象的内存空间，取对象的值。

1、Python**缓存了整数和短字符串**，因此每个对象在内存中只存有一份，引用所指对象就是相同的，即使使用赋值语句，也只是创造新的引用，而不是对象本身；

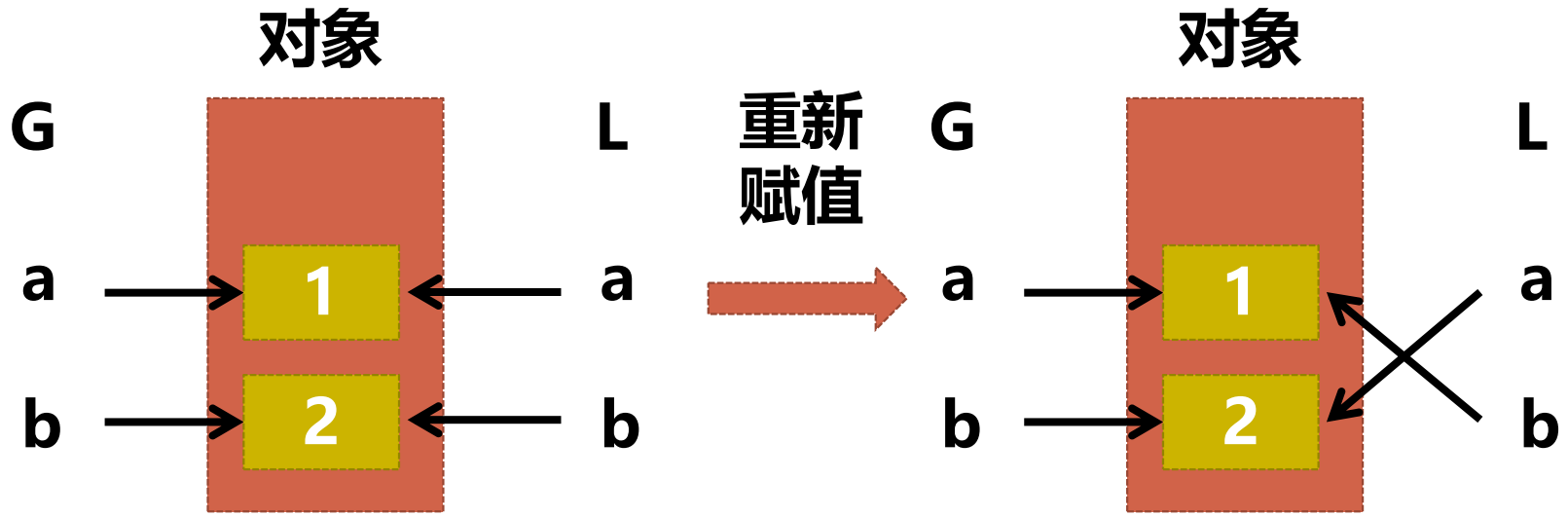
2、Python**没有缓存长字符串、列表及其他对象**，可以有多个相同的对象，可以使用赋值语句创建出新的对象。

赋值规则

参数赋值时创建了局部变量的引用，函数内对形参的重新赋值不影响实参

```
1  def func(a, b):
2      c = a
3      a = b
4      b = c
5      print(a, b, id(a), id(b))
6
7  a = 1
8  b = 2
9  print(a, b, id(a), id(b))
10 func(a, b)
11 print(a, b, id(a), id(b))
```

```
1 2 8791702648608 8791702648640
2 1 8791702648640 8791702648608
1 2 8791702648608 8791702648640
```

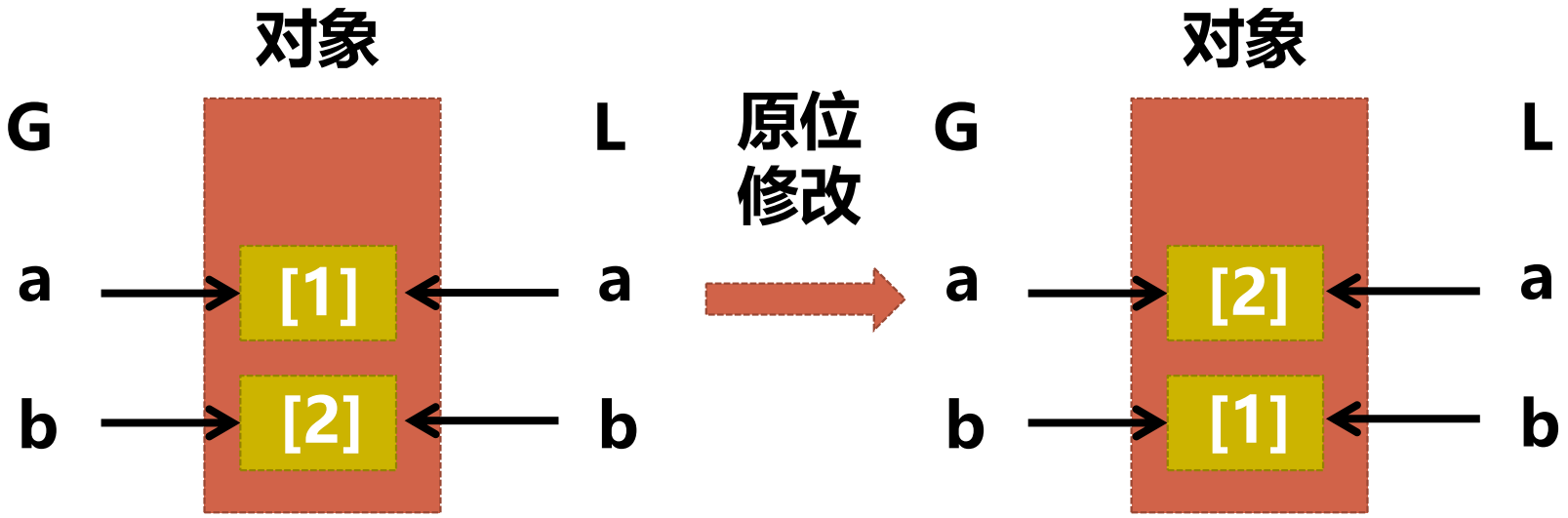


赋值规则

参数赋值时创建了局部变量的引用，函数内对形参的原位修改会影响实参

```
1  def func(a, b):
2      c = a[0]
3      a[0] = b[0]
4      b[0] = c
5      print(a, b, id(a), id(b))
6
7  a = [1]
8  b = [2]
9  print(a, b, id(a), id(b))
10 func(a, b)
11 print(a, b, id(a), id(b))
```

```
[1] [2] 95650304 95622720
[2] [1] 95650304 95622720
[2] [1] 95650304 95622720
```



传不可变对象

不可变对象（数字、字符串、元组）不能直接改变对象里面的内容，只能重新赋值，而且是在新的内存中创建新的对象后重新赋值

```
1  def func(a):
2      print(id(a))
3      a = 10
4      print(id(a))
5
6  a = 1
7  print(id(a))
8  func(a)
```

```
8791702648608
8791702648608
8791702648896
```

函数内对a重新赋值，会新创建对象10

传不可变对象

```
1  def func(a):
2      a[2] = 4
3
4  a = (1, 2, 3)
5  func(a)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-17-4af4d7f61b3f> in <module>
      3
      4 a = (1, 2, 3)
----> 5 func(a)

<ipython-input-17-4af4d7f61b3f> in func(a)
      1 def func(a):
----> 2     a[2] = 4
      3
      4 a = (1, 2, 3)
      5 func(a)

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

不可变对象无法原位修改

传可变对象

可变对象（列表和字典）可以直接改变对象里面的内容

```python
def func(a):
    print(a, id(a))
    a.append(4)
    print(a, id(a))

a = [1, 2, 3]
print(a, id(a))
func(a)
print(a, id(a))
```

```
[1, 2, 3] 95599872
[1, 2, 3] 95599872
[1, 2, 3, 4] 95599872
[1, 2, 3, 4] 95599872
```

列表原位修改

```python
def func(a):
    print(a, id(a))
    a["x"] = 2
    print(a, id(a))

a = {"x": 1, "y": 2}
print(a, id(a))
func(a)
print(a, id(a))
```

```
{'x': 1, 'y': 2} 96542592
{'x': 1, 'y': 2} 96542592
{'x': 2, 'y': 2} 96542592
{'x': 2, 'y': 2} 96542592
```

字典原位修改

```python
def func(a):
    print(a, id(a))
    a = [1, 2, 3, 4]
    print(a, id(a))

a = [1, 2, 3]
print(a, id(a))
func(a)
print(a, id(a))
```

```
[1, 2, 3] 95626048
[1, 2, 3] 95626048
[1, 2, 3, 4] 95580800
[1, 2, 3] 95626048
```

列表重新赋值

传可变对象

```python
def func(a):
    print(a, id(a))
    a.append(4)
    print(a, id(a))

a = [1, 2, 3]
print(a, id(a))
func(a)
print(a, id(a))
```

```
[1, 2, 3] 95599872
[1, 2, 3] 95599872
[1, 2, 3, 4] 95599872
[1, 2, 3, 4] 95599872
```

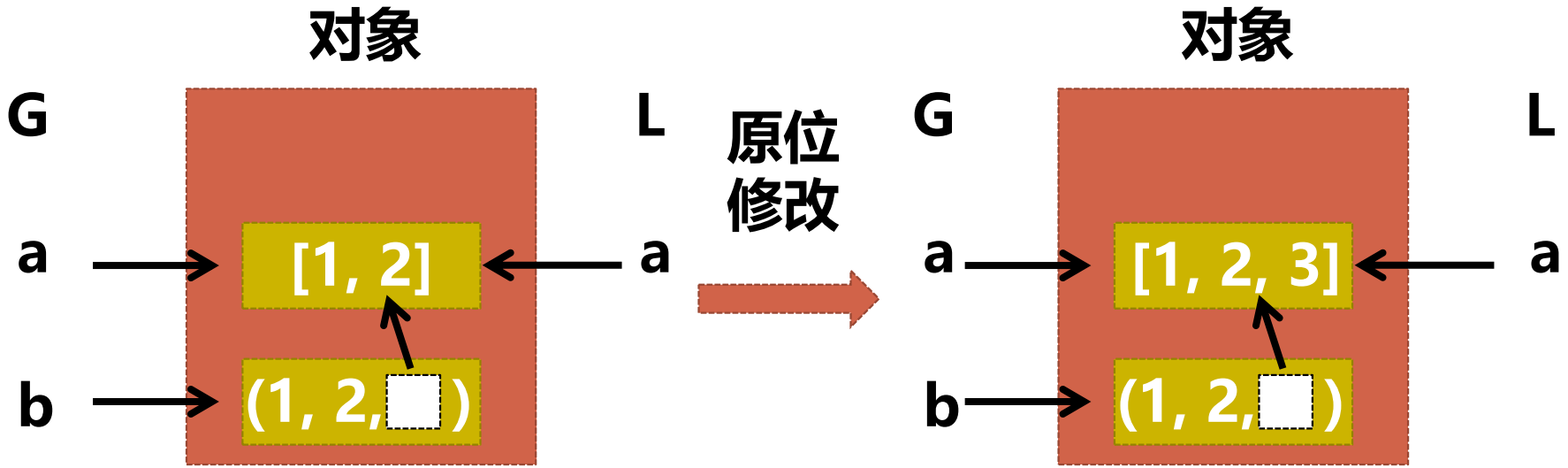
```python
def func(a):
    print(a, id(a))
    a.append(4)
    print(a, id(a))

a = [1, 2, 3]
print(a, id(a))
func(a[:])
print(a, id(a))
```

? ? ?

修改元组中的列表

```
1  def func(a):
2      print(a, b, id(a), id(b))
3      a.append(3)
4      print(a, b, id(a), id(b))
5
6  a = [1, 2]
7  b = (1, 2, a)
8  print(a, b, id(a), id(b))
9  func(a)
10 print(a, b, id(a), id(b))
```

```
[1, 2] (1, 2, [1, 2]) 95622144 95842368
[1, 2] (1, 2, [1, 2]) 95622144 95842368
[1, 2, 3] (1, 2, [1, 2, 3]) 95622144 95842368
[1, 2, 3] (1, 2, [1, 2, 3]) 95622144 95842368
```



参数匹配

四种参数形式

- 位置参数：从左到右顺序匹配
- 关键字参数：按参数名字乱序匹配
- 缺省参数：可以不提供实参的参数
- 可变参数：收集未匹配位置或关键字参数

```
1  def func(a, b, c):
2      return a + b + c
3
4  func(1, 2, 3) # 位置参数匹配
```

```
6
```

位置参数按顺序匹配

```
1  func(1, 2, c = 3) # 关键字参数匹配
```

```
6
```

c按照关键字参数匹配

位置匹配&关键字匹配

```
1 def func(a, b, c):
2     return a + b + c
3
4 func(1, c = 3, b = 2)
```

关键字参数顺序可以打乱

```
6
```

```
1 func(c = 3, b = 2, 1)
```

```
  File "<ipython-input-45-3410fa8c6f5f>", line 1
    func(c = 3, b = 2, 1)

SyntaxError: positional argument follows keyword argument
```

位置参数必须放在关键字参数前面

```
1 func(1, b = 2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-46-926cddb2ffb8> in <module>
----> 1 func(1, b = 2)

TypeError: func() missing 1 required positional argument: 'c'
```

非缺省参数不够

缺省参数

```
1  def func(a, b, c = 3):
2      return a + b + c
3
4  func(1, b = 2)
```

```
6
```

定义函数时给参数设定缺省值

```
1  func(1, b = 2, c = 4)
```

```
7
```

```
1  func(1, c = 4, b = 2)
```

```
7
```

```
1  def func(a, c = 3, b):
2      return a + b + c
3
4  func(1, b = 2)
```

缺省值必须放在位置参数后面

```
  File "<ipython-input-51-5920de32dfc2>", line 1
    def func(a, c = 3, b):

SyntaxError: non-default argument follows default argument
```

元组可变参数

```
1 def func(a, b, *c):
2     print(a, b, c)
3
4 func(1, 2, 3)
5 func(1, 2, 3, 4, 5, 6)
```

```
1 2 (3,)
1 2 (3, 4, 5, 6)
```

定义函数时在形参前面加*，表示元组可变参数，以元组的形式打包收集多余的位置参数

```
1 func(1, 2, *tuple(range(5)))
```

```
1 2 (0, 1, 2, 3, 4)
```

调用有元组可变参数的函数时，在元组实参前加星号，可以解包参数

```
1 func(1, 2, tuple(range(5)))
```

```
1 2 ((0, 1, 2, 3, 4),)
```

```
1 def func(a, b, c):
2     print(a, b, c)
3 func(1, 2, *tuple(range(5)))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-57-ae46cc8e9211> in <module>
      1 def func(a, b, c):
      2     print(a, b, c)
----> 3 func(1, 2, *tuple(range(5)))

TypeError: func() takes 3 positional arguments but 7 were given
```

字典可变参数

```
1  def func(a, b, **c):
2      print(a, b, c)
3
4  func(1, 2, x=3)
5  func(1, 2, x=3, y=4, z=5)
```

```
1 2 {'x': 3}
1 2 {'x': 3, 'y': 4, 'z': 5}
```

定义函数时在形参前面加，表示字典可变参数，以字典的形式打包收集多余的关键字参数**

```
1  func(1, 2, 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-63-0b0e6d51b54e> in <module>
----> 1 func(1, 2, 3)

TypeError: func() takes 2 positional arguments but 3 were given
```

不能收集多余的位置参数

```
1  func(1, 2, a = 3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-64-4c421753fde9> in <module>
----> 1 func(1, 2, a = 3)

TypeError: func() got multiple values for argument 'a'
```

已存在的形参不能作为关键字参数键值

字典可变参数

```
1  def func(a, b, **c):
2      print(a, b, c)
3
4  func(1, 2, **{"x":3, "y":4})
```

```
1 2 {'x': 3, 'y': 4}
```

调用有字典可变参数的函数时，在元组实参前加星号，可以解包参数

```
1  func(1, 2, {"x":3, "y":4})
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-66-13e3ff6d22a7> in <module>
----> 1 func(1, 2, {"x":3, "y":4})

TypeError: func() takes 2 positional arguments but 3 were given
```

不加**，实参无法传入形参

同时包括元组和字典可变参数

```
1  def func(a, b, *c, **d):
2      print(a, b, c, d)
3
4  func(1, 2, 3, 4, x=5, y=6)
```

```
1 2 (3, 4) {'x': 5, 'y': 6}
```

```
1  func(1, 2, 3, 4)
```

```
1 2 (3, 4) {}
```

```
1  func(1, 2)
```

```
1 2 () {}
```

元组或字典实参不够时，传入空的元组或字典给形参

```
1  func(1, 2, x=5, y=6)
```

```
1 2 () {'x': 5, 'y': 6}
```

```
1  func(1, 2, *tuple(range(3)), **{"x":5, "y":6})
```

```
1 2 (0, 1, 2) {'x': 5, 'y': 6}
```

同时包括元组和字典可变参数

```
1  func(1, 2, 3, x=5, y=6, 4)
```

```
  File "<ipython-input-81-24ae03297f43>", line 1
    func(1, 2, 3, x=5, y=6, 4)
                            ^
SyntaxError: positional argument follows keyword argument
```

元组可变参数→字典可变参数

```
1  def func(a, b, **c, *d):
2      print(a, b, c, d)
3
4  func(1, 2, 3, 4, x=5, y=6)
```

```
  File "<ipython-input-82-1c7fb3fe8746>", line 1
    def func(a, b, **c, *d):
                        ^
SyntaxError: invalid syntax
```

可变参数和缺省参数

```
1 def func(a, b, *c, **d, e = 7):
2     print(a, b, c, d, e)
3
4 func(1, 2, 3, 4, x=5, y=6)
```

```
  File "<ipython-input-94-275a057df3de>", line 1
    def func(a, b, *c, **d, e = 7):

SyntaxError: invalid syntax
```

位置参数→缺省参数→可变参数

```
1 def func(a, b, e = 7, *c, **d):
2     print(a, b, c, d, e)
3
4 func(1, 2, 3, 4, x=5, y=6)
```

```
1 2 (4,) {'x': 5, 'y': 6} 3
```

```
1 func(1, 2, 3, 4, x=5, y=6, e=9)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-91-73e4d4f88f2e> in <module>
----> 1 func(1, 2, 3, 4, x=5, y=6, e=9)

TypeError: func() got multiple values for argument 'e'
```

函数应用实例

求100~200里面所有的素数

```
# 定义判断是否为素数的函数

def isPrime(num):









for i in range(100, 200):
    if isPrime(i):
        print(i, end = ",")

```

101,103,107,109,113,127,131,137,139,149,151,157,163,167,173,179,181,191,193,197,199,

定义函数calculate，可以接收任意多个数，返回一个元组，元组第一个值为所有参数的平均值，第二个值是大于平均值的所有数

```python
def calculate(*num):
    li = []
    avg = sum(num)/len(num)
    for i in num:
        if i > avg:
            li.append(i)
    return avg, li

b = []
while True:
    a = float(input("请输入一个数："))
    if a == 0:
        break
    else:
        b.append(a)

count = calculate(*tuple(b))
print(count)
```

```
请输入一个数：10
请输入一个数：11
请输入一个数：12
请输入一个数：0
(11.0, [12.0])
```

编写函数，接收一个列表(包含10个整型数)和一个整型数k，返回一个新列表（将列表下标k之前对应(不包含k)的元素逆序，将下标k及之后的元素逆序)

```python
import random

def fun(alist, k):
    if k < 0 or k > len(alist):
        return "k值超出范围"
    newL1 = alist[:k]
    newL2 = newL1[::-1]
    newL3 = alist[k:]
    newL4 = newL3[::-1]
    return newL2 + newL4

list = []
for i in range(10):
    num = random.randint(1, 50)
    list.append(num)
print(list)
a = fun(list, 4)
print(a)
```

```
[38, 12, 15, 7, 12, 39, 46, 22, 16, 34]
[7, 15, 12, 38, 34, 16, 22, 46, 39, 12]
```

小结

- 函数定义：def和return语句， 参数的动态性
- 命名空间和作用域：LEGB规则，全局和局部变量
- 参数传递和匹配：可变和不可变参数，四种参数匹配形式

下一节：函数编程