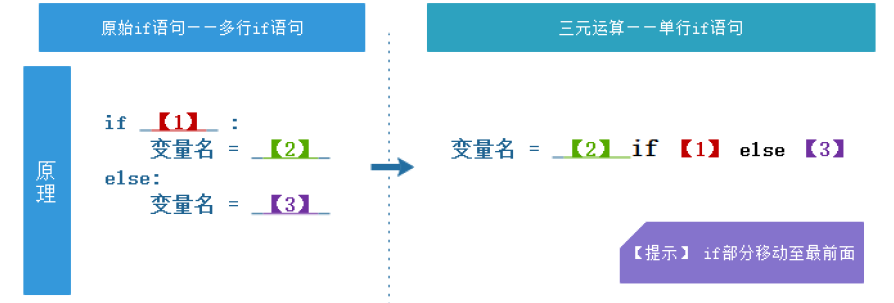
1. if 语句单行写法 （简洁写法，三元运算）

<表达式1> if <条件> else <表达式2>

说明如下:

当<条件>为True时返回 <表达式1>，否则返回 <表达式2>



例：

num = 100

info = ("偶数" if num%2==0 else "奇数")

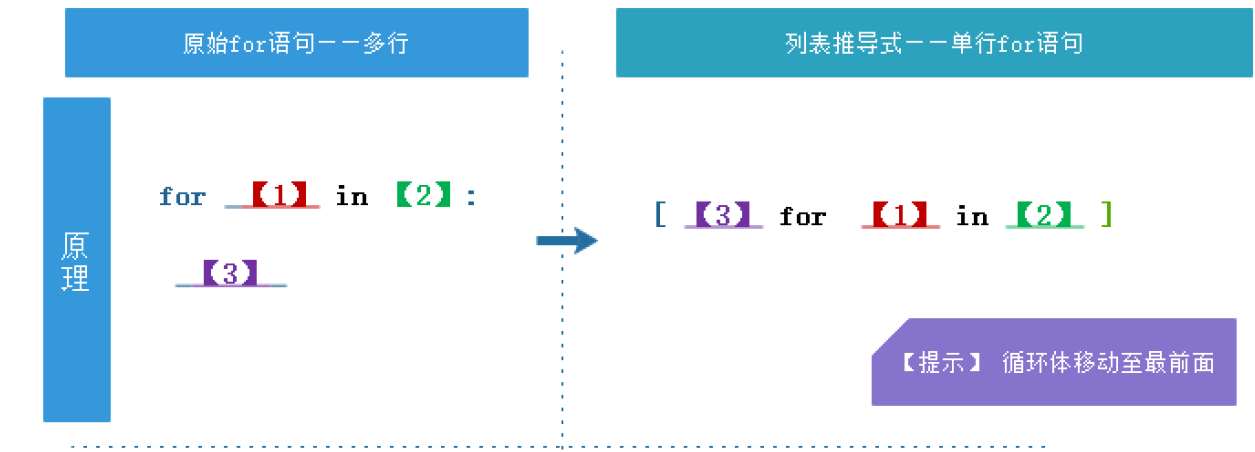
1. for 语句 单行写法

for iterating\_var in sequence:

statements(s)

for 迭代变量 in 字符串|列表|元组|字典|集合：  
代码块

列表（元组、字典、集合等）推导式



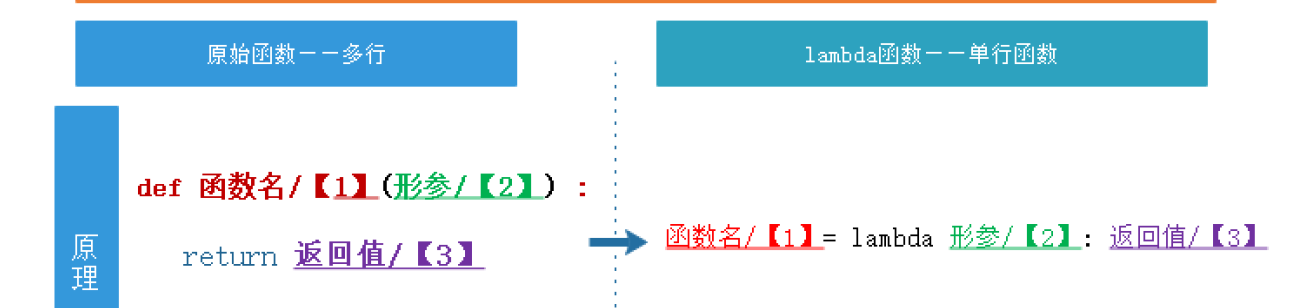
例子：

|  |
| --- |
| for i in [True, 124,**'abc'**,1.0]:  print(type(i)) |

|  |
| --- |
| def dataTpye(x):  *#a=[True, 124,'abc',1.0]*  for i in x :  print(i,type(i))  a=[True, 124,**'abc'**,1.0] dataTpye(a) |

|  |
| --- |
| s=[pow(i,2)+5 for i in range(10)] s=[ type(i) for i in [True, 124,”abc”,1.0] ] |

1. 函数单行写法 -----匿名函数，一般作为另外一个函数的参数，即函数本身就是参数



例子：

|  |
| --- |
| list(map(lambda x: x \* x, [1, 2, 3, 4, 5, 6, 7, 8, 9]))  [1, 4, 9, 16, 25, 36, 49, 64, 81] |

**——————————————————————**

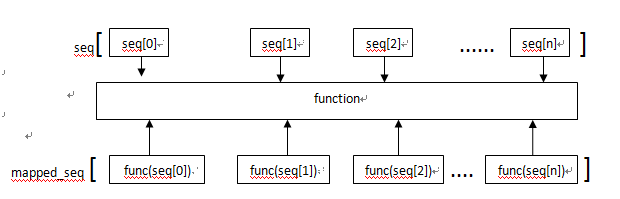
|  |
| --- |
| lambda x: x \* x  **def** **f**(x):  **return** x \* x |

|  |
| --- |
| def adds(x,y):  return x+y *#l=map(adds,[1,-1,-22,-3,5],[1,-1,-22,-3,5])* l=map(lambda x,y:x+y,[1,-1,-22,-3,5],[1,-1,-22,-3,5]) print(list(l)) |

一、函数不带括号时，调用的是这个函数本身 ，是整个函数体，是一个函数对象，不须等该函数执行完成  
二、带括号（参数或者无参），调用的是函数的执行结果，须等该函数执行完成的结果

|  |
| --- |
| # -\*- coding:utf-8 -\*-    def bracket(data):  return data  if \_\_name\_\_ == '\_\_main\_\_':  # 不带括号调用的结果：<function bracket at 0x0000000004DD0B38>, a是整个函数体，是一个函数对象，不须等该函数执行完成  a = bracket  print a  # 带括号调用的结果：6, b是函数执行后返回的值6, 须等该函数执行完成的结果  b = bracket(6)  print b |

1. map() 返回一个将 function 应用于 iterable 中每一项并输出其结果的迭代器(可以放在for循环 in 后)。 如果传入了额外的 iterable 参数，function 必须接受相同个数的实参并被应用于从所有可迭代对象中并行获取的项。 当有多个可迭代对象时，最短的可迭代对象耗尽则整个迭代就将结束。



1. **filter**(function, iterable)[¶](file:///E:\%E4%B8%8A%E8%AF%BE\python\2020%E4%B8%8B%E6%95%99%E5%AD%A6%E7%9B%B8%E5%85%B3\python-3.8.5-docs-html\library\functions.html#filter)**函数:** 用 iterable 中函数 function 返回真的那些元素，构建一个新的迭代器。iterable 可以是一个序列，一个支持迭代的容器，或一个迭代器。如果 function 是 None ，则会假设它是一个身份函数，即 iterable 中所有返回假的元素会被移除。

|  |
| --- |
| foo = [2, 18, 9, 22, 17, 24, 8, 12, 27] s=filter(lambda x: x % 3 == 0, foo) print (list(s)) |

[18, 9, 24, 12, 27]

>>>>>> print map(lambda x: x \* 2 + 10, foo)

[14, 46, 28, 54, 44, 58, 26, 34, 64]

1. **reduce()** 函数会对参数序列中元素进行累积。

函数将一个数据集合（链表，元组等）中的所有数据进行下列操作：用传给 reduce 中的函数 function（有两个参数）先对集合中的第 1、2 个元素进行操作，得到的结果再与第三个数据用 function 函数运算，最后得到一个结果。

***注意：****Python3.x reduce() 已经被移到 functools 模块里，如果我们要使用，需要引入 functools 模块来调用 reduce() 函数：*

from functools import reduce

>>>>>> print reduce(lambda x, y: x + y, foo)

139

|  |
| --- |
| **from** functools **import** reduce  **def** add(x, y) :            # 两数相加     **return** x + y sum1 = reduce(add, [1,2,3,4,5])   # 计算列表和：1+2+3+4+5 sum2 = reduce(**lambda** x, y: x+y, [1,2,3,4,5])  # 使用 lambda 匿名函数 **print**(sum1) **print**(sum2) |

1. **enumerate**(*iterable*, *start=0*)

返回一个枚举对象。*iterable* 必须是一个序列，或 [iterator](file:///E:\%E5%8F%B8%E6%B5%B7%E5%B9%B3\%E6%95%B0%E6%8D%AE%E9%87%87%E9%9B%86%E4%B8%8E%E5%A4%84%E7%90%86\python-3.9.7-docs-html\python-3.9.7-docs-html\glossary.html#term-iterator)，或其他支持迭代的对象。 [enumerate()](file:///E:\%E5%8F%B8%E6%B5%B7%E5%B9%B3\%E6%95%B0%E6%8D%AE%E9%87%87%E9%9B%86%E4%B8%8E%E5%A4%84%E7%90%86\python-3.9.7-docs-html\python-3.9.7-docs-html\library\functions.html?highlight=enumerate#enumerate) 返回的迭代器的 [\_\_next\_\_()](file:///E:\%E5%8F%B8%E6%B5%B7%E5%B9%B3\%E6%95%B0%E6%8D%AE%E9%87%87%E9%9B%86%E4%B8%8E%E5%A4%84%E7%90%86\python-3.9.7-docs-html\python-3.9.7-docs-html\library\stdtypes.html#iterator.__next__) 方法返回一个元组，里面包含一个计数值（从 *start* 开始，默认为 0）和通过迭代 *iterable* 获得的值。

seasons = ['Spring', 'Summer', 'Fall', 'Winter']

list(enumerate(seasons))

list(enumerate(seasons, start=1))

s=["将军","却说","荆州","二人","不能","如此"]  
t=enumerate(s)  
print(s)  
print(list(t))

1. **zip()** 函数用于将可迭代的对象作为参数，将对象中对应的元素打包成一个个元组，然后返回由这些元组组成的列表。

如果各个迭代器的元素个数不一致，则返回列表长度与最短的对象相同，利用 \* 号操作符，可以将元组解压为列表。

|  |
| --- |
| x = [1, 2, 3]  y = [4, 5, 6]  zipped = zip(x, y)  list(zipped) |

[zip()](file:///E:\%E5%8F%B8%E6%B5%B7%E5%B9%B3\%E6%95%B0%E6%8D%AE%E9%87%87%E9%9B%86%E4%B8%8E%E5%A4%84%E7%90%86\python-3.9.7-docs-html\python-3.9.7-docs-html\library\functions.html#zip) 与 \* 运算符相结合可以用来拆解一个列表: # 与 zip 相反，\*zipped 可理解为解压，返回二维矩阵式

x2, y2 = zip(\*zip(x, y))

x == list(x2) **and** y == list(y2)

|  |
| --- |
| seq = ['one', 'two', 'three'] seq1=[1,2,3] seq3=[4,5,6,7] zz=zip(seq,seq1) *#print(list(zz))* print(list(zip(\*zz))) |