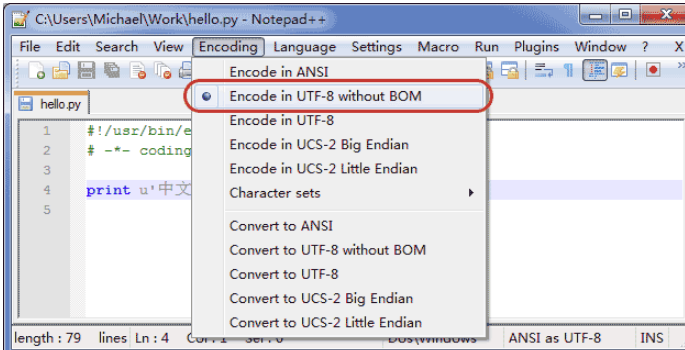
# 基础

## 中文编码

在文件开头加入**# -\*- coding: UTF-8 -\*-** 或者 **# coding=utf-8**就行了

注意：=两边不能有空格

这里指的是解释器读取文件时的编码。前提要python源文件用utf-8编码



## 标识符

以单下划线开头**\_foo** 的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用 from xxx import \* 而导入。

以双下划线开头的**\_\_foo** 代表类的私有成员，以双下划线开头和结尾的 **\_\_foo\_\_**代表 Python 里特殊方法专用的标识，如 \_\_init\_\_() 代表类的构造函数。

## **引号**



“”” text ”””其中的换行也会输出

## 布尔

运算符：and or not

## 空值

None不是0，是一个python特殊的值

## 写时复制

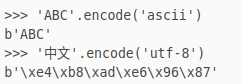


## 编码

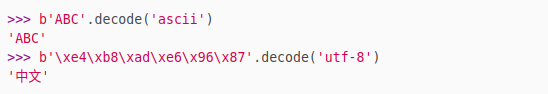
A.Python3内存中字符串以unicode编码的，所以其支持多语言

B.Bytes与str类型之间转换：

（1）Bytes - > str: .decode



（2）Str -> bytes: .encode



忽略错误



## len()对于str和bytes

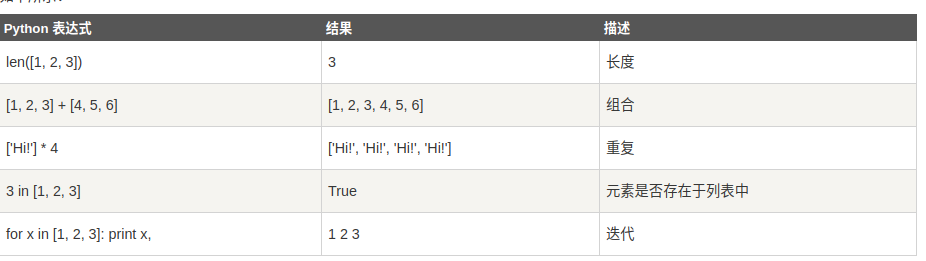
如果操作对象是str，则是字符数。

如果操作对象是bytes，则是字节数。

## 数据结构List--[]列表

1. 操作:索引，切片，加，乘，检查成员

### 脚本操作符



### 内置函数

6.del eg: del list1[3]

### 内置方法



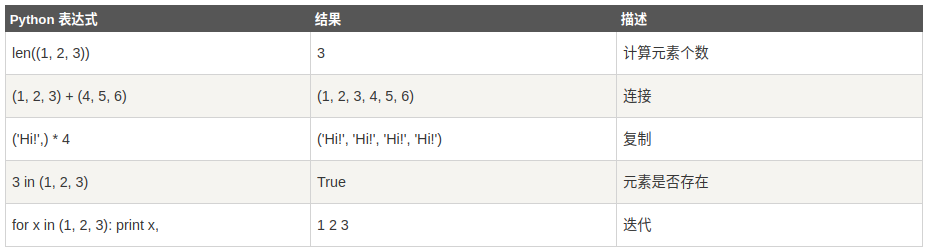
## 数据结构tuple--()元祖

和list类似，区别就是tuple中的元素不能变了，也没有append，insert方法，能用tuple就尽量用。

这里的不变指的是 **tuple的每个元素，指向永远不变，且元素数量也不能变。**

**无法往一个已经初始化好的tuple添加和删除元素**

### 脚本操作符



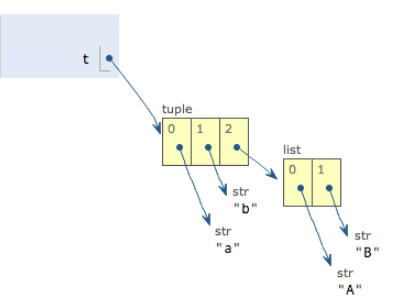
### 无关闭分隔符



### 内置函数

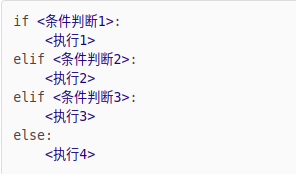


### “可变”tuple



不变指的是元素指向

## 条件判断



## 循环

for ... in range

for ... in list

## 数据结构Dict--{}字典

就是其他语言的map

### 内置函数



1. del 删除

### 内置方法



用in 和get可以避免索引不存在后出错

## 数据结构set--{}

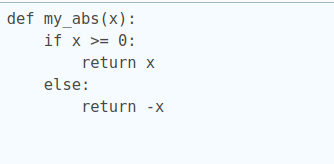
**和dict类似**也是一组key但是不存value（这是和dist的唯一区别），所以**key唯一**，也就是说其**无序**。

Set不能存放可变对象比如list。而list可以放list

### 内置方法



## 函数



函数执行完毕也没有return语句时，自动return None。

函数可以同时返回多个值，但其实就是一个tuple。

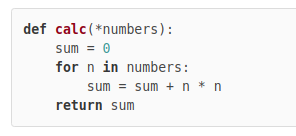
### 参数

<https://www.liaoxuefeng.com/wiki/1016959663602400/1017261630425888>

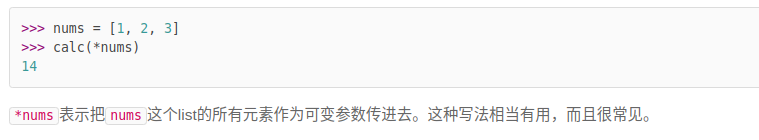
#### 位置参数

#### 默认参数

#### 可变参数(tuple)



调用时的\*



\*numbers 实质是一个tuple。调用时可以传入任意个参数。

#### 关键字参数（dict）

本质是dict



调用1：

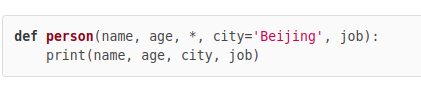


调用2：



#### 命名关键字参数

对于关键字参数，调用者可以传入任意不受限制的关键字参数。如果要限制，就可以用命名关键字参数。

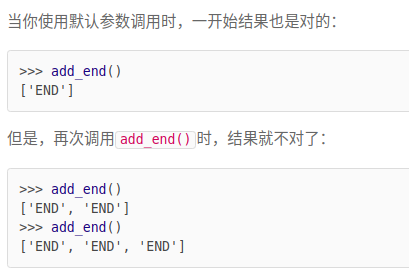


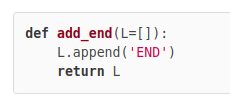
#### 参数组合

5种参数都可以组合使用。

参数定义的顺序必须是：必选参数、默认参数、可变参数、命名关键字参数和关键字参数

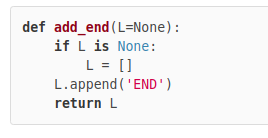
### 默认参数需不可变





Python函数在定义的时候，默认参数L的值就被计算出来了，即[]，因为默认参数L也是一个变量，它指向对象[]，每次调用该函数，如果改变了L的内容，则下次调用时，默认参数的内容就变了，不再是函数定义时的[]了。

解决改善：



## 高级特性

### 切片

比go多了第三个项，可以调间距。

### 迭代

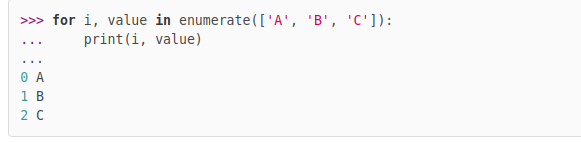
1. str,List,tuple,dict都可迭代
2. Dict的迭代比较特殊：

默认情况下，dict迭代的是key。如果要迭代value，可以用**for value in d.values()**，如果要同时迭代key和value，可以用**for k, v in d.items()**。由于字符串也是可迭代对象，因此，也可以作用于for循环：

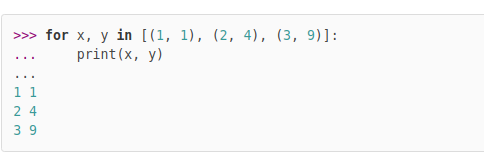
1. 判断是否可迭代(iterable)



1. 如何像go迭代slice一样有下标呢。

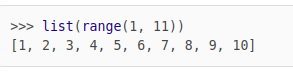


1. 独特迭代



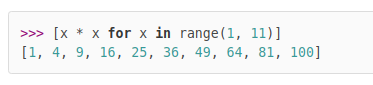
### 列表生成器

1. range：

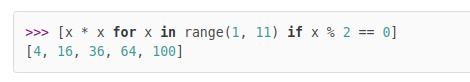


1. 如果要生成[1x1, 2x2, 3x3, ..., 10x10]呢？？？

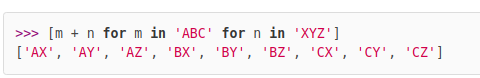
for...in range()

、

for...in range() if...



for in .. for ... in



### 生成器（generator）

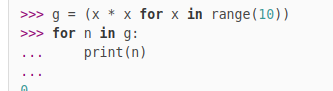
通过列表生成式，我们可以直接创建一个列表。但是，受到内存限制，列表容量肯定是有限的。而且，创建一个包含100万个元素的列表，不仅占用很大的存储空间，如果我们仅仅需要访问前面几个元素，那后面绝大多数元素占用的空间都白白浪费了

1. 将列表生成器[]换成()

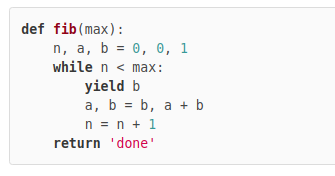
g=(x \* x for x in range(10))

使用：

1. next(g)#计算出g的下一个元素的值，直到计算到最后一个元素，没有更多的元素时，抛出StopIteration的错误。（基本不用，使用起来变态）
2. For



1. 通过函数方式把print换yield



### 迭代器(Interator)

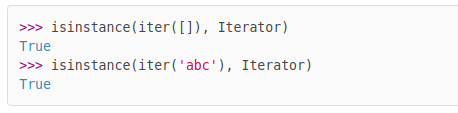
可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator。

可以直接作用于for循环的对象统称为可迭代对象：Iterable。

生成器都是Iterator对象，但list、dict、str虽然是Iterable，却不是Iterator。

#### 可迭代对象转换成迭代器

iter



1. 为什么list、dict、str等数据类型不是Iterator？

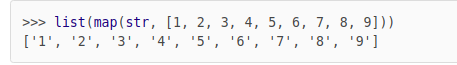
这是因为Python的Iterator对象表示的是一个数据流，Iterator对象可以被next()函数调用并不断返回下一个数据，直到没有数据时抛出StopIteration错误。可以把这个数据流看做是一个有序序列，但我们却不能提前知道序列的长度，只能不断通过next()函数实现按需计算下一个数据，所以Iterator的计算是惰性的，只有在需要返回下一个数据时它才会计算。

Iterator甚至可以表示一个无限大的数据流，例如全体自然数。而使用list是永远不可能存储全体自然数的。

## 高阶函数

### map

map()函数接收两个参数，一个是函数，一个是Iterable，map将传入的函数依次作用到序列的每个元素，并把结果作为新的Iterator返回



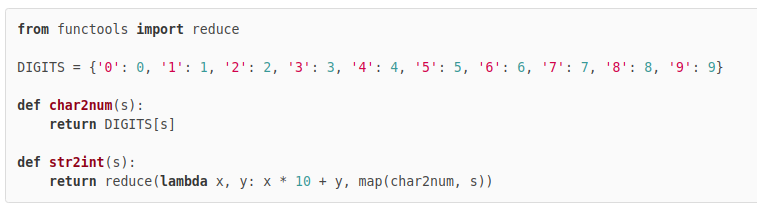
### reduce

reduce把一个函数作用在一个序列[x1, x2, x3, ...]上，这个函数必须接收两个参数，reduce把结果继续和序列的下一个元素做累积计算，其效果就是：





配合lambda



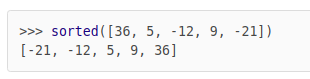
### filter

和map()类似，filter()也接收一个函数和一个序列。和map()不同的是，filter()把传入的函数依次作用于每个元素，然后根据返回值是True还是False决定保留还是丢弃该元素

注意到filter()函数返回的是一个Iterator，也就是一个惰性序列，所以要强迫filter()完成计算结果，**需要用list()函数获得所有结果并返回list。**

### sort

1. 普通用法



1. 加入key函数



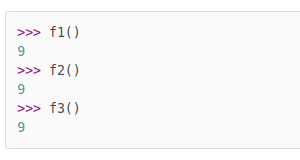
1. 传入reverse=True反向排序



## 闭包

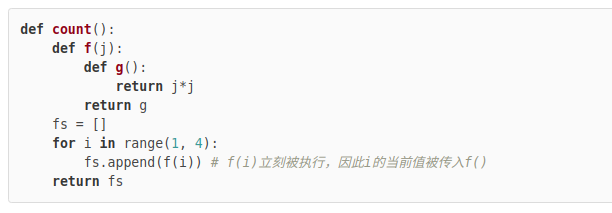
1. 注意点





返回闭包时牢记一点：返回函数不要引用任何循环变量，或者后续会发生变化的变量。

和go的用法差不多啦

**解决：**

## 匿名函数lambda

关键字lambda表示匿名函数，冒号前面的x表示函数参数。

匿名函数有个限制，就是只能有一个表达式，不用写return，返回值就是该表达式的结果。

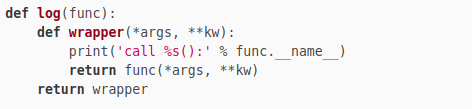
## 装饰器

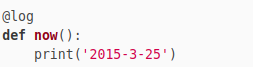
在代码运行期间动态增加功能的方式称之为装饰器。

1. 两层。

<-原始

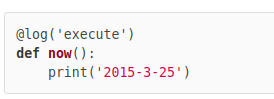
----



<-本质：now = log(now)

1. 三层，装饰器本身需要传入参数



<-本质：now=log(‘execute’)(now)

1. 最后注意点

因为返回的那个wrapper()函数名字就是'wrapper'，所以，需要把原始函数的\_\_name\_\_等属性复制到wrapper()函数中，否则，有些依赖函数签名的代码执行就会出错。

不需要编写wrapper.\_\_name\_\_ = func.\_\_name\_\_这样的代码，Python内置的functools.wraps就是干这个事的，所以，一个完整的decorator的写法如下：

