# Python基本用法

## Dict和Set类型

## Python之什么是dict

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

我们把**名字称为key**，对应的**成绩称为value**，dict就是通过 **key**来查找 **value**。

花括号 {} 表示这是一个dict，然后按照**key: value**, 写出来即可。最后一个 key: value 的逗号可以省略。

Java中对应map哦。

len() 函数可以计算任意集合的大小：

>>> len(d)

3

## Python之访问dict

 d[key] 的形式来查找对应的 value，这和 list 很像，不同之处是，**list 必须使用索引返回对应的元素，而dict使用key**

如果key不存在，会直接报错：KeyError。

避免报错：

**一是先判断一下 key 是否存在，用 in 操作符**

**二是使用dict本身提供的一个 get 方法，在Key不存在的时候，返回None：d.get(‘Bart’)**

## Python中dict的特点

**dict的第一个特点是查找速度快，无论dict有10个元素还是10万个元素，查找速度都一样**。

## Python更新dict

>>> d['Paul'] = 72

## Python之 遍历dict

由于dict也是一个集合，所以，遍历dict和遍历list类似，都可以通过 for 循环实现。

## Python中什么是set

**dict的作用是建立一组 key 和一组 value 的映射关系，dict的key是不能重复的。**

有的时候，我们只想要 dict 的 key，不关心 key 对应的 value，目的就是保证这个集合的元素不会重复，这时，set就派上用场了。

**set 持有一系列元素，这一点和 list 很像，但是set的元素没有重复，而且是无序的，这点和 dict 的 key很像。**

## Python之 访问set

由于**set存储的是无序集合**，所以我们没法通过索引来访问。

访问 set中的某个元素实际上就是判断一个元素是否在set中。

## Python之 set的特点

**set的内部结构和dict很像，唯一区别是不存储value**，因此，判断一个元素是否在set中速度很快。

**set存储的元素和dict的key类似，必须是不变对象**，因此，任何可变对象是不能放入set中的。

最后，set存储的元素也是没有顺序的。

## Python之 遍历set

由于 set 也是一个集合，所以，遍历 set 和遍历 list 类似，都可以通过 for 循环实现。

直接使用 for 循环可以遍历 set 的元素：

## Python之 更新set

由于**set存储的是一组不重复的无序元素**，因此，更新set主要做两件事：

>>> s = set([1, 2, 3])

>>> s.add(3)

>>> s.remove(4)

## 函数

## ython之编写函数

在Python中，定义一个函数要使用**def**语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用 return语句返回。

我们以自定义一个求绝对值的 my\_abs 函数为例：

def my\_abs(x):

if x >= 0:

return x

else:

return -x

return None可以简写为return。

## Python函数之返回多值

函数可以返回多个值吗？答案是肯定的。

比如在游戏中经常需要从一个点移动到另一个点，给出坐标、位移和角度，就可以计算出新的坐标：

**# math**包提供了**sin()**和 **cos()**函数，我们先用import引用它：

import math

def move(x, y, step, angle):

nx = x + step \* math.cos(angle)

ny = y - step \* math.sin(angle)

return nx, ny

实际上返回的是一个tuple

## Python之定义默认参数

定义函数的时候，还可以有默认参

可见，**函数的默认参数的作用是简化调用**，你只需要把必须的参数传进去。但是在需要的时候，又可以传入额外的参数来覆盖默认参数值。

由于函数的参数按从左到右的顺序匹配，所以**默认参数只能定义在必需参数的后面：**

## Python之定义可变参数

如果想让一个函数能接受任意个参数，我们就可以定义一个可变参数：

def fn(\*args):

print args

可变参数的名字前面有个 **\***号，我们可以传入0个、1个或多个参数给可变参数：

>>> fn()

()

>>> fn('a')

('a',)

>>> fn('a', 'b')

('a', 'b')

>>> fn('a', 'b', 'c')

('a', 'b', 'c')

可变参数也不是很神秘，Python解释器会把传入的一组参数组装成一个tuple传递给可变参数，因此，在函数内部，直接把变量 args 看成一个 tuple 就好了。

## 切片

## 对list进行切片

对这种经常取指定索引范围的操作，用循环十分繁琐，因此，Python提供了切片（Slice）操作符，能大大简化这种操作。

对应上面的问题，取前3个元素，用一行代码就可以完成切片：

>>> L[0:3]

['Adam', 'Lisa', 'Bart']

L[0:3]表示，从索引0开始取，直到索引3为止，但不包括索引3。即索引0，1，2，正好是3个元素。

如果第一个索引是0，还可以省略：

>>> L[:3]

['Adam', 'Lisa', 'Bart']

也可以从索引1开始，取出2个元素出来：

>>> L[1:3]

['Lisa', 'Bart']

只用一个**:**，表示从头到尾：

>>> L[:]

['Adam', 'Lisa', 'Bart', 'Paul']

因此，L[:]实际上复制出了一个新list。

切片操作还可以指定第三个参数：

>>> L[::2]

['Adam', 'Bart']

第三个参数表示每N个取一个，上面的 L[::2] 会每两个元素取出一个来，也就是隔一个取一个。

把list换成tuple，切片操作完全相同，只是切片的结果也变成了tuple。

## 倒序切片

对于list，既然Python支持L[-1]取倒数第一个元素，那么它同样支持倒数切片，试试：

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

>>> L[-2:]

['Bart', 'Paul']

>>> L[:-2]

['Adam', 'Lisa']

>>> L[-3:-1]

['Lisa', 'Bart']

>>> L[-4:-1:2]

['Adam', 'Bart']

记住倒数第一个元素的索引是-1。倒序切片包含起始索引，不包含结束索引。

## 迭代

## 什么是迭代

在Python中，如果给定一个**list**或**tuple**，我们可以通过for循环来遍历这个list或tuple，这种遍历我们成为迭代（Iteration）。

**因为 Python 的 for循环不仅可以用在list或tuple上，还可以作用在其他任何可迭代对象上。**

因此，迭代操作就是对于一个集合，无论该集合是有序还是无序，我们用 for 循环总是可以依次取出集合的每一个元素。

**注意**: 集合是指包含一组元素的数据结构，我们已经介绍的包括：

1. **有序集合**：list，tuple，str和unicode；

2. **无序集合**：set

3. **无序集合并且具有 key-value 对**：dict

## 索引迭代

Python中，**迭代永远是取出元素本身，而非元素的索引。**

对于有序集合，元素确实是有索引的。有的时候，我们确实想在 for 循环中拿到索引，怎么办？

方法是使用 **enumerate() 函数**：

>>> L = ['Adam', 'Lisa', 'Bart', 'Paul']

>>> for index, name in enumerate(L):

... print index, '-', name

...

0 - Adam

1 - Lisa

2 - Bart

3 - Paul

使用 enumerate() 函数，我们可以在for循环中同时绑定索引index和元素name。但是，这不是 enumerate() 的特殊语法。实际上，enumerate() 函数把：

**['Adam', 'Lisa', 'Bart', 'Paul']**

变成了类似：

**[(0, 'Adam'), (1, 'Lisa'), (2, 'Bart'), (3, 'Paul')]**

因此，迭代的每一个元素实际上是一个tuple：

for t in enumerate(L):

index = t[0]

name = t[1]

print index, '-', name

如果我们知道每个tuple元素都包含两个元素，for循环又可以进一步简写为：

for index, name in enumerate(L):

print index, '-', name

这样不但代码更简单，而且还少了两条赋值语句。

可见，索引迭代也不是真的按索引访问，而是由 enumerate() 函数自动把每个元素变成 (index, element) 这样的tuple，再迭代，就同时获得了索引和元素本身。

## 迭代dict的value

我们已经了解了**dict对象**本身就是可**迭代对象**，用 for 循环直接迭代 dict，可以每次拿到dict的一个key。

如果我们希望迭代 dict 对象的value，应该怎么做？

dict 对象有一个 **values() 方法**，这个方法把dict转换成一个包含所有value的list，这样，我们迭代的就是 dict的每一个 value

如果仔细阅读Python的文档，还可以发现，dict除了**values()**方法外，还有一个**itervalues()**方法，用**itervalues()** 方法替代 **values()** 方法，迭代效果完全一样：

**那这两个方法有何不同之处呢？**

1. **values()** 方法实际上把一个 dict 转换成了包含 value 的list。

2. 但是 **itervalues()** 方法不会转换，它会在迭代过程中依次从 dict 中取出 value，所以 itervalues() 方法比 values() 方法节省了生成 list 所需的内存。

3. 打印 itervalues() 发现它返回一个 <dictionary-valueiterator> 对象，这说明在Python中，**for 循环可作用的迭代对象远不止 list，tuple，str，unicode，dict等**，任何可迭代对象都可以作用于for循环，而内部如何迭代我们通常并不用关心。

**如果一个对象说自己可迭代，那我们就直接用 for 循环去迭代它，可见，迭代是一种抽象的数据操作，它不对迭代对象内部的数据有任何要求。**

## 迭代dict的key和value

我们了解了如何**迭代 dict** 的**key**和**value**，那么，在一个 for 循环中，能否同时迭代 key和value？答案是肯定的。

首先，我们看看 dict 对象的 **items()** 方法返回的值：

## 生成列表

要生成list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]，我们可以用range(1, 11)：

range(1, 100, 2) 可以生成list [1, 3, 5, 7, 9,...]

## 复杂表达式

使用**for循环**的迭代不仅可以迭代普通的list，还可以迭代dict。

假设有如下的dict：

d = { 'Adam': 95, 'Lisa': 85, 'Bart': 59 }

完全可以通过一个复杂的列表生成式把它变成一个 HTML 表格：

tds = ['<tr><td>%s</td><td>%s</td></tr>' % (name, score) for name, score in d.iteritems()]

print '<table>'

print '<tr><th>Name</th><th>Score</th><tr>'

print '\n'.join(tds)

print '</table>'

**注：**字符串可以通过 % 进行格式化，用指定的参数替代%s。字符串的join()方法可以把一个 list 拼接成一个字符串。

把打印出来的结果保存为一个html文件，就可以在浏览器中看到效果了：

## 条件过滤

列表生成式的 **for 循环后面还可以加上 if 判断**。例如：

>>> [x \* x for x in range(1, 11)]

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

如果我们只想要偶数的平方，不改动 range()的情况下，可以加上 if 来筛选：

## 多层表达式

for循环可以嵌套，因此，在列表生成式中，也可以用多层 for 循环来生成列表。