1.Python的数据类型和基础函数

1.1 元组

元组（turple）是一个固定长度，不可改变的Python序列对象。创建元组只需要用()包括值即可，或者直接使用逗号分隔。与list最大的不同在于，元组属于不可变数据类型，我们不能修改元组

In [6]:
```
tu=(1,2)
tu2=(1,)
tu3=1,2,3,4
type(tu3)
```

Out[6]:
```
tuple
```

In [8]:
```
#选择元组的元素
tu3[0]
```

Out[8]:
```
1
```

In [79]:
```
#元组是不可变的对象，但如果元组中的某个对象是可变的，比如列表，可以进行修改
tup = tuple(['foo', [1, 2], True])
tup[1].append(3)
tup
```

Out[79]:
```
('foo', [1, 2, 3], True)
```

In [82]:
```
#可以用加法和乘法串联
tup1 = (4, None, 'foo') + (6, 0) + ('bar',)
tup2 = ('foo', 'bar') * 5
tup1,tup2
```

Out[82]:
```
((4, None, 'foo', 6, 0, 'bar'),
 ('foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'bar'))
```

In [78]:
```
#用tuple可以将任意序列或迭代器转换成元组
tup = tuple('string')
tup
```

Out[78]:
```
('s', 't', 'r', 'i', 'n', 'g')
```

In [90]:
```
#count（也适用于列表），它可以统计某个值得出现频率
a = (1, 2, 2, 2, 3, 4, 2)
a.count(2)
```

Out[90]:
```
4
```

1.2 列表

与元组对比，列表的长度可变、内容可以被修改。列表属于可变的数据类型，因此可以添执行加、删除，或者是搜索列表中的项目。列表可以嵌套

In [108…

```
li = [1,2,3,4]
li2 = ['a','b','c','d']
li3 = [1,2,'a','b']
li,li2,li3
```

Out[108…

```
([1, 2, 3, 4], ['a', 'b', 'c', 'd'], [1, 2, 'a', 'b'])
```

In [15]:

```
#列表的常用操作1（turple也有同样的操作）
li = [1,2,3,4]
li2=[5,6,7,8]
1 in li ##返回布尔值，1是否在li中
1 not in li ##返回布尔值，1是否不在li中
li+li2 ##li和li2的拼接，可以+=
li[2] ##返回li索引为2的元素，从0开始
li[0:2] ##返回0到（2-1）的元素
li[:2]#同上
len(li) ##li元素个数
min(li)##最小元素
max(li)##最大元素
li.count(1)##统计li中为1的个数
```

Out[15]:

```
1
```

In [22]:

```
#列表的常用操作2（turple没有）
##list的操作
li = [1,2,3,4]
li[3]=8 ##赋值
li.append(5) ##尾部赋值,等价于li.insert(len(li),5)
li.insert(1,10)##在索引为2的位置插入10，不是赋值
#li.clear()##清空
li.extend(li2)##插入一组数值，等于+=
li.remove(3)##删除第一个等于3的元素，如果要删除所有，则需要使用循环
li.reverse()##列表反向
li.sort()#排序，字母顺序，大写在小写前面，sort的排序是永久的，即sort后，原序列就改变了
li.sort(reverse=True)#排序，反过来进行
li2 = sorted(li)#和sort相反，sorted排序是临时的，不会影响原序列
li.sort(reverse=True)#临时排序，反过来进行
```

In [111…

```
a_list.pop(2)#pop移除并返回指定位置的元素
```

Out[111…

```
3
```

In [24]:

```
## list的操作
li = [1,2,3]
print(li*3)#列表的复制
del li[2] #删除索引为2的元素（3）
print(li)
li = [1,2,3,1]
print(li.index(2))#查找元素的索引
print(li.index(1))#如果元素出现多次，则显示第一个
```

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

```
[1, 2]
1
0
```

列表的赋值，引用和复制

对于变量来说，赋值是复制值。但是对于列表来说，赋值其实只是赋值引用

In [1]:
```
spam=42
cheese = spam
spam=100
print(spam,cheese)
```

```
100 42
```

In [2]:
```
#对于列表来说，赋值是两个一起改变，不管是修改源列表还是目标列表
spam = [0,1,2,3,4,5]
cheese=spam
cheese[0]='hello'
print(spam,cheese)
```

```
['hello', 1, 2, 3, 4, 5] ['hello', 1, 2, 3, 4, 5]
```

In [3]:
```
#如果不想单纯引用，可以使用copy()
import copy
spam = ['A','B','C','D']
cheese = copy.copy(spam)
cheese[1]=42
print(spam,cheese)
```

```
['A', 'B', 'C', 'D'] ['A', 42, 'C', 'D']
```

In [9]:
```
#或者把元素复制
spam = ['A','B','C','D']
cheese = spam[:]
cheese[1]=42
print(spam,cheese)
```

```
['A', 'B', 'C', 'D'] ['A', 42, 'C', 'D']
```

1.3 字典

字典是一种"键-值"(key-value)映射结构，字典使用花括号{}包括，键和值之间用冒号：分割，每对键-值"用逗号，分割，键必须唯一。

1.3.1 字典的基本操作

In [6]:
```
#创建字典的方法之一是使用尖括号，用冒号分隔键和值：
#创建空字典
empty_dict = {}
#创建一个字典
d1 = {'a' : 'some value', 'b' : [1, 2, 3, 4]}
d1
```

Out[6]:
```
{'a': 'some value', 'b': [1, 2, 3, 4]}
```

In [9]:
```
#字典元素的访问，但是如果访问字典中不存在的键则会报错
d1['a']
```

Out[9]:
```
'some value'
```

In [11]:
```
#为了处理报错，可以使用get()提取元素，键不存在时，返回默认
#get(键，默认值)它有两个参数：要取得其值的键，以及如果该键不存在时，返回的备用值。
spam =  {'color':'red','age':42}
spam.get('name','None')
```

Out[11]:
```
'None'
```

In [30]:
```
#插入元素
d1[7] = 'an integer'#插入字典中的元素，7为key，'an integer'为value
d1
```

Out[30]:
```
{'a': 'some value', 'b': 'foo', 'c': 12, 7: 'an integer'}
```

In [31]:
```
#update方法可以插入多个元素
d1.update({'b' : 'foo', 'c' : 12})
d1
```

Out[31]:
```
{'a': 'some value', 'b': 'foo', 'c': 12, 7: 'an integer'}
```

In [20]:
```
#删除元素
#del关键字或pop方法（返回值的同时删除键）删除值
del d1[7]
print(d1)
```

```
{'a': 'some value', 'b': [1, 2, 3, 4], 'dummy': 'another value'}
```

In [21]:
```
#pop方法
d1['dummy'] = 'another value'
ret = d1.pop('dummy')
ret,d1
```

Out[21]:
```
('another value', {'a': 'some value', 'b': [1, 2, 3, 4]})
```

1.3.2 字典的遍历

In [25]:
```
#keys()返回键,values()返回值，items()方法返回键值对
spam =  {'color':'red','age':42}
spam.values()
for i in spam.items():
    print(i)
```

```
('color', 'red')
('age', 42)
```

In [26]:
```
#keys和values是字典的键和值的迭代器方法。虽然键值对没有顺序，这两个方法可以用相同的顺
list(d1.keys()),list(d1.values())
```

Out[26]:
```
(['a', 'b'], ['some value', [1, 2, 3, 4]])
```

In [28]:
```
#遍历key值 在使用上，for key in a和 for key in a.keys():完全等价
#字典记录键和值之间的关联关系，但获取字典的元素时，获取顺序是不可预测的
#所以除了使用items还可以使用这个方法保证关联
a={'a': '1', 'b': '2', 'c': '3'}
print("遍历key值:")
for key in a:
    print(key+':'+a[key])
```

```
遍历key值:
a:1
b:2
c:3
```

1.4 集合

集合是无序的不可重复的元素的集合。你可以把它当做字典，但是只有键没有值。所以可以利用集合方法去重

In [16]:
```
#两种方式创建集合：通过set函数或使用尖括号set语句
set([2, 2, 2, 1, 3, 3]),{2, 2, 2, 1, 3, 3}
```

Out[16]:
```
({1, 2, 3}, {1, 2, 3})
```

函数	替代语法	说明
a.union(b)	a|b	并集，不改变a和b，将并集返回
a.update(b)	a|=b	并集，改变a，不改变b
a.intersection(b)	a&b	a和b中交叉的元素
a.intersection_update(b)	a&=b	a和b中交叉的元素,改变a
a.difference(b)	a-b	存在于a但不存在于b的元素
a.difference_update(b)	a-=b	存在于a但不存在于b的元素，改变a
a.symmetric_difference(b)	a^b	只在a或只在b的元素
a.symmetric_difference_update(b)	a^=b	只在a或只在b的元素，改变a

In [37]:
```
a = {1, 2, 3, 4, 5}
b = {3, 4, 5, 6, 7, 8}
a.union(b)
a
```

Out[37]:
```
{1, 2, 3, 4, 5}
```

1.5常用序列函数

1.5.1 enumerate函数

enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

enumerate(sequence, [start=0])

- sequence 一个序列、迭代器或其他支持迭代对象。
- start 下标起始位置。

In [124…

```
seasons = ['Spring','Summer','Fall','Winter']
for i,ele in enumerate(seasons):
    print(i,ele)
```

```
0 Spring
1 Summer
2 Fall
3 Winter
```

1.5.2 zip函数

zip可以将多个列表、元组或其它序列成对组合成一个元组列表.zip可以处理任意多的序列，元素的个数取决于最短的序列

In [30]:

```
zipped = zip(seq1, seq2)
#zip后的zipped是一个迭代器，迭代器可以通过for输出，也可以使用list
list(zipped)
```

Out[30]:

```
[('foo', 'one'), ('bar', 'two'), ('baz', 'three')]
```

In [32]:

```
#zip(*)可以理解为解压
zipped = zip(seq1, seq2)
unlist = zip(*zipped)
list(unlist)
```

Out[32]:

```
[('foo', 'bar', 'baz'), ('one', 'two', 'three')]
```

In [20]:

```
#.zip可以处理任意多的序列，元素的个数取决于最短的序列
seq1 = ['foo', 'bar', 'baz']
seq2 = ['one', 'two', 'three']
seq3 = [False, True]
list(zip(seq1, seq2, seq3))
```

Out[20]:

```
[('foo', 'one', False), ('bar', 'two', True)]
```

In [18]:

```
#注意迭代器有一个特别的性质，一次迭代后则耗尽内容
#可以看到这个对象第一次迭代有结果，但是没有第二次的迭代结果
l1 = [1,2,3,4]
l2 = [2,3,4,5]
l3 = zip(l1,l2)

for x in l3:
    print('for循环{}'.format(x))
print('----')
for i in l3:
    print('for循环{}'.format(i))
```

```
for循环(1, 2)
for循环(2, 3)
for循环(3, 4)
for循环(4, 5)
----
```

In [3]:

```
#zip的常见用法之一是同时迭代多个序列，可能结合enumerate使用
for i, (a, b) in enumerate(zip(seq1, seq2)):
    print('{0}: {1}, {2}'.format(i, a, b))
```

```
0: foo, one
1: bar, two
2: baz, three
```

1.5.3 sorted与reversed函数

sorted函数可以从任意序列的元素返回一个新的排好序的列表

In [126…
```
sorted([7, 1, 2, 6, 0, 3, 2]),sorted('horse race')
```

Out[126…
```
([0, 1, 2, 2, 3, 6, 7], [' ', 'a', 'c', 'e', 'e', 'h', 'o', 'r', 'r', 's'])
```

reversed函数

In [4]:
```
list(reversed(range(10)))#reversed可以从后向前迭代一个序列
```

Out[4]:
```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

1.6 变量

1.6.1 动态变量

In [2]:
```
#创建动态变量
createVar = locals()
for i in ['1','2','3']:
    createVar['li'+i]= int(i)
    print(createVar['li'+i])
```

```
1
2
3
```