Python入门二高级数据类型

List，tupel，dict，set

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

1. 1、List
   * 1. list申明：

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

* + 1. list索引：

需要特别注意的是，索引从 0 开始，也就是说，第一个元素的索引是0，第二个元素的索引是1，以此类推。

因此，要打印第一名同学的名字，用 L[0]:

>>> print L[0]

Adam

要打印第四名同学的名字，用 L[3]:

>>> print L[3]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: list index out of range

报错了！IndexError意思就是索引超出了范围，因为上面的list只有3个元素，有效的索引是 0，1，2。

所以，使用索引时，千万注意不要越界。

* + 1. 倒序访问list：

Bart同学是最后一名，俗称倒数第一，所以，我们可以用 -1 这个索引来表示最后一个元素：

>>> print L[-1]

Bart

Bart同学表示躺枪。

类似的，倒数第二用 -2 表示，倒数第三用 -3 表示，倒数第四用 -4 表示：

>>> print L[-2]

Lisa

>>> print L[-3]

Adam

>>> print L[-4]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: list index out of range

L[-4] 报错了，因为倒数第四不存在，一共只有3个元素。

使用倒序索引时，也要注意不要越界。

对List切片

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

取前三个：

>>> L[0:3]

['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']

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

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

>>> L[-2:]

['Bart', 'Paul']

>>> L[:-2]

['Adam', 'Lisa']

>>> L[-3:-1]

['Lisa', 'Bart']

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

['Adam', 'Bart']

* + 1. 添加新元素：

用 list 的append()方法，向list中添加新元素

>>> L.append('Paul')

>>> print L

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

append()总是把新的元素添加到 list 的尾部。

用list的insert()方法，它接受两个参数，第一个参数是索引号，第二个参数是待添加的新元素：

>>> L.insert(0, 'Paul')

>>> print L

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

L.insert(0, 'Paul')的意思是，'Paul'将被添加到索引为 0 的位置上（也就是第一个），而原来索引为 0 的Adam同学，以及后面的所有同学，都自动向后移动一位。

* + 1. 删除元素：

list的pop()方法删除

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

>>> L.pop()

'Paul'

>>> print L

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

pop()方法总是**删掉list的最后一个元素**，并且它还返回这个元素，所以我们执行 L.pop() 后，会打印出 'Paul'。

如果Paul同学不是排在最后一个怎么办？比如Paul同学排在第三：

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

要把Paul踢出list，我们就必须先定位Paul的位置。由于Paul的索引是2，因此，用pop(2)把Paul删掉：

>>> L.pop(2)

'Paul'

>>> print L

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

任务：

注意右边编辑器代码中 list 如下：

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

Paul的索引是2，Bart的索引是3，如果我们要把Paul和Bart都删掉，请解释下面的代码为什么不能正确运行：

L.pop(2)  
L.pop(3)

怎样调整代码可以把Paul和Bart都正确删除掉？

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

L.pop(2)

L.pop(3)

print L

* + 1. 替换：

使用Paul替换Bart

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

>>> L[2] = 'Paul'

>>> print L

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

对list中的某一个索引赋值，就可以直接用新的元素替换掉原来的元素，list包含的元素个数保持不变。

由于Bart还可以用 -1 做索引，因此，下面的代码也可以完成同样的替换工作：

>>> L[-1] = 'Paul'

1. 2、Tupel

tuple是另一种有序的列表，中文翻译为“ 元组 ”。tuple 和 list 非常类似，但是，tuple一旦创建完毕，就不能修改了。

同样是表示班里同学的名称，用tuple表示如下：

>>> t = ('Adam', 'Lisa', 'Bart')

创建tuple和创建list唯一不同之处是用( )替代了[ ]。

现在，这个 t 就不能改变了，tuple没有 append()方法，也没有insert()和pop()方法。所以，新同学没法直接往 tuple 中添加，老同学想退出 tuple 也不行。

获取 tuple 元素的方式和 list 是一模一样的，我们可以正常使用 t[0]，t[-1]等索引方式访问元素，但是不能赋值成别的元素，不信可以试试：

>>> t[0] = 'Paul'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

创建单元素tupel ：

创建包含1个元素的 tuple 呢？来试试：

>>> t = (1)

>>> print t

1

好像哪里不对！t 不是 tuple ，而是整数1。为什么呢？

因为()既可以表示tuple，又可以作为括号表示运算时的优先级，结果 (1) 被Python解释器计算出结果 1，导致我们得到的不是tuple，而是整数 1。

正是因为用()定义单元素的tuple有歧义，所以 Python 规定，单元素 tuple 要多加一个逗号”,”，这样就避免了歧义：

>>> t = (1,)

>>> print t

(1,)

Python在打印单元素tuple时，也自动添加了一个“,”，为了更明确地告诉你这是一个tuple。

“可变”的tupel：

创建可变的tupel

>>> t = ('a', 'b', ['A', 'B'])

注意到 t 有 3 个元素：'a'，'b'和一个list：['A', 'B']。list作为一个整体是tuple的第3个元素。list对象可以通过 t[2] 拿到：

>>> L = t[2]

然后，我们把list的两个元素改一改：

>>> L[0] = 'X'

>>> L[1] = 'Y'

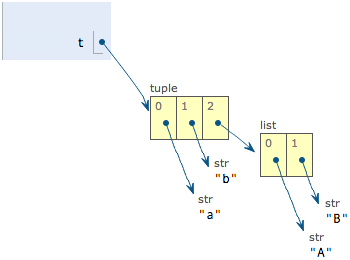
再看看tuple的内容：

>>> print t

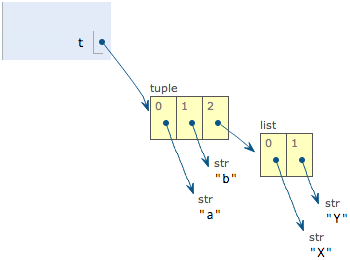
('a', 'b', ['X', 'Y'])

不是说tuple一旦定义后就不可变了吗？怎么现在又变了？（解释为何tupel可变了）

别急，我们先看看定义的时候tuple包含的3个元素：



当我们把list的元素'A'和'B'修改为'X'和'Y'后，tuple变为：



表面上看，tuple的元素确实变了，但其实变的不是 tuple 的元素，而是list的元素。

tuple一开始指向的list并没有改成别的list，所以，tuple所谓的“不变”是说，tuple的每个元素，指向永远不变。即指向'a'，就不能改成指向'b'，指向一个list，就不能改成指向其他对象，但指向的这个list本身是可变的！

理解了“指向不变”后，要创建一个内容也不变的tuple怎么做？那就必须保证tuple的每一个元素本身也不能变。

测试：

a='jie test 1'

t = (a, 'b', ['A', 'B'])

print t

a='jie test2'

t = (a, 'b', ['A', 'B'])

print t

结果：

('jie test 1', 'b', ['A', 'B'])

('jie test2', 'b', ['A', 'B'])

3、dict

创建dict

如果把名字和分数关联起来，组成类似的查找表：

'Adam' ==> 95

'Lisa' ==> 85

'Bart' ==> 59

给定一个名字，就可以直接查到分数。

Python的 dict 就是专门干这件事的。用 dict 表示“名字”-“成绩”的查找表如下：

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。

由于dict也是集合，len()函数可以计算任意集合的大小：

>>> len(d)

3

注意:一个 key-value 算一个，因此，dict大小为3

访问dict

可以简单地使用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：

>>> print d['Adam']

95

>>> print d['Paul']

Traceback (most recent call last):

File "index.py", line 11, in <module>

print d['Paul']

KeyError: 'Paul'

注意:通过 key 访问 dict 的value，只要 key 存在，dict就返回对应的value。如果key不存在，会直接报错：KeyError。

要避免 KeyError 发生，有两个办法：

一是先判断一下 key 是否存在，用 in 操作符：

if 'Paul' in d:

print d['Paul']

如果 'Paul' 不存在，if语句判断为False，自然不会执行 print d['Paul'] ，从而避免了错误。

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：

>>> print d.get('Bart')

59

>>> print d.get('Paul')

None

dict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。而list的查找速度随着元素增加而逐渐下降。

不过dict的查找速度快不是没有代价的，dict的缺点是占用内存大，还会浪费很多内容，list正好相反，占用内存小，但是查找速度慢。

由于dict是按 key 查找，所以，在一个dict中，key不能重复。

dict的第二个特点就是存储的key-value序对是没有顺序的！这和list不一样：

d = {

'Adam': 95,

'Lisa': 85,

'Bart': 59

}

当我们试图打印这个dict时：

>>> print d

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

打印的顺序不一定是我们创建时的顺序，而且，不同的机器打印的顺序都可能不同，这说明dict内部是无序的，不能用dict存储有序的集合。

dict的第三个特点是作为 key 的元素必须不可变，Python的基本类型如字符串、整数、浮点数都是不可变的，都可以作为 key。但是list是可变的，就不能作为 key。

更新dict

将新同学'Paul'的成绩 72 添加进去，用赋值语句：

>>> d['Paul'] = 72

如果 key 已经存在，则赋值会用新的 value 替换掉原来的 value：

>>> d['Bart'] = 60

>>> print d

{'Lisa': 85, 'Paul': 72, 'Adam': 95, 'Bart': 60}

遍历dict

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

for key in d:

print key

4、set

创建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。

创建 set 的方式是调用 set() 并传入一个 list，list的元素将作为set的元素：

>>> s = set(['A', 'B', 'C'])

可以查看 set 的内容：

>>> print s

set(['A', 'C', 'B'])

请注意，上述打印的形式类似 list， 但它不是 list，仔细看还可以发现，打印的顺序和原始 list 的顺序有可能是不同的，因为set内部存储的元素是无序的。

因为set不能包含重复的元素，所以，当我们传入包含重复元素的 list 会怎么样呢？

>>> s = set(['A', 'B', 'C', 'C'])

>>> print s

set(['A', 'C', 'B'])

>>> len(s)

3

访问set元素

Bart是该班的同学吗？

>>> 'Bart' in s

True

bart是该班的同学吗？

>>> 'bart' in s

False

看来大小写很重要，'Bart' 和 'bart'被认为是两个不同的元素。

set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。

set用处：

如果事先创建好一个set，包含'MON' ~ 'SUN'：

weekdays = set(['MON', 'TUE', 'WED', 'THU', 'FRI', 'SAT', 'SUN'])

再判断输入是否有效，只需要判断该字符串是否在set中：

x = '???' # 用户输入的字符串

if x in weekdays:

print 'input ok'

else:

print 'input error'

这样一来，代码就简单多了。

遍历set

示例：

s = set([('Adam', 95), ('Lisa', 85), ('Bart', 59)])

for x in s:

print x[0],':',x[1]

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：

一是把新的元素添加到set中，二是把已有元素从set中删除。

**添加**元素时，用set的add()方法：

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

>>> s.add(4)

>>> print s

set([1, 2, 3, 4])

如果添加的元素已经存在于set中，add()不会报错，但是不会加进去了：

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

>>> s.add(3)

>>> print s

set([1, 2, 3])

**删除**set中的元素时，用set的remove()方法：

>>> s = set([1, 2, 3, 4])

>>> s.remove(4)

>>> print s

set([1, 2, 3])

如果删除的元素不存在set中，remove()会报错：

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

>>> s.remove(4)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 4

所以用add()可以直接添加，而remove()前需要判断。