

Chap07–List and Tuple

College of Computer Science
Nankai University

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

序列

- ▶ 序列是一系列连续值，它们通常是相关的，并且按一定顺序排列。
- ▶ 序列是程序设计中经常用到的数据存储方式，几乎每一种程序设计语言都提供了类似的数据结构，如C和Basic中的一维、多维数组等。

序列

- ▶ Python提供的序列类型在所有程序设计语言中是最丰富，最灵活，也是功能最强大的。
- ▶ Python中常用的序列结构有列表、元组、字典、字符串、集合以及range等等。
- ▶ 除字典和集合之外，列表、元组、字符串等序列均支持双向索引，第一个元素下标为0，第二个元素下标为1，以此类推；最后一个元素下标为-1，倒数第二个元素下标为-2，以此类推。

列表

- ▶ 列表是Python中内置可变序列，是若干元素的有序集合，列表中的每一个数据称为元素，列表的所有元素放在一对中括号 “[”和 “]”中，并使用逗号分隔开；
- ▶ 当列表元素增加或删除时，列表对象自动进行扩展或收缩内存，保证元素之间没有缝隙；
- ▶ 在Python中，一个列表中的数据类型可以各不相同，可以同时分别为整数、实数、字符串等基本类型，甚至是列表、元组、字典、集合以及其他自定义类型的对象。

列表

例如：

[10, 20, 30, 40]

['crunchy frog', 'ram bladder', 'lark vomit']

['spam', 2.0, 5, [10, 20]]

[['file1', 200,7], ['file2', 260,9]]

列表

方法	说明
<code>list.append(x)</code>	将元素x添加至列表尾部
<code>list.extend(L)</code>	将列表L中所有元素添加至列表尾部
<code>list.insert(index, x)</code>	在列表指定位置index处添加元素x
<code>list.remove(x)</code>	在列表中删除首次出现的指定元素
<code>list.pop([index])</code>	删除并返回列表对象指定位置的元素
<code>list.clear()</code>	删除列表中所有元素，但保留列表对象，该方法在Python2中没有
<code>list.index(x)</code>	返回值为x的首个元素的下标
<code>list.count(x)</code>	返回指定元素x在列表中的出现次数
<code>list.reverse()</code>	对列表元素进行原地逆序
<code>list.sort()</code>	对列表元素进行原地排序
<code>list.copy()</code>	返回列表对象的浅拷贝，该方法在Python2中没有

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的创建

- ▶ 使用 “=” 直接将一个列表赋值给变量即可创建列表对象，例如：

```
>>> a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
```

```
>>> a_list = [] #创建空列表
```

- ▶ 使用list()函数将元组、range对象、字符串或其他类型的可迭代对象类型的数据转换为列表。例如：

```
>>> a_list = list((3,5,7,9,11))
```

```
>>> a_list
```

```
[3, 5, 7, 9, 11]
```

列表的创建

```
>>> list(range(1,10,2))
```

```
[1, 3, 5, 7, 9]
```

```
>>> list('hello world')
```

```
['h', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd']
```

```
>>> x = list() #创建空列表
```



Outline

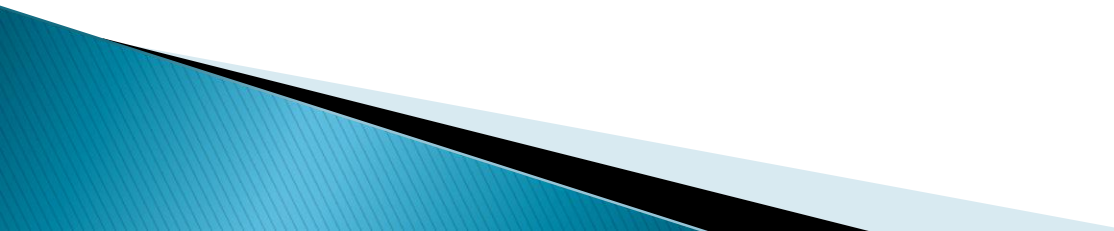
▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的添加

- ▶ 使用 “+” 运算符
 - ▶ 使用 “*” 运算符
 - ▶ 使用列表对象的append()方法
 - ▶ 使用列表对象的extend()方法
 - ▶ 使用列表对象的insert()方法
- 

列表的添加

- ▶ 使用 “+”运算符来实现将元素添加到列表中
- ▶ 严格意义上，这并不是真的为列表添加元素，而是创建一个新列表，并将原列表中的元素和新元素依次复制到新列表的内存空间
- ▶ 由于涉及大量元素的复制，该操作速度较慢，在涉及大量元素添加时不建议使用该方法。

```
>>> aList = [3,4,5]
```

```
>>> aList = aList + [7]
```

```
>>> aList
```

```
[3, 4, 5, 7]
```

列表的添加

- ▶ 使用列表对象的append()方法，原地修改列表，是真正意义上的在列表尾部添加元素，速度较快，也是推荐使用的方法。

```
>>> aList.append(9)
```

```
>>> aList
```

```
[3, 4, 5, 7, 9]
```

列表的添加

- ▶ 使用列表对象的extend()方法可以将另一个迭代对象的所有元素添加至该列表对象尾部。

```
>>> a = [1,2,3]
```

```
>>> id(a)
```

```
26973848
```

```
>>> a.extend([7,8,9])
```

```
>>> a
```

```
[1, 2, 3, 7, 8, 9]
```

```
>>> id(a)
```

```
26973848
```

列表的添加

- ▶ 使用列表对象的insert()方法将元素添加至列表的指定位置。

```
>>> a = [1,2,3]
```

```
>>> id(a)
```

```
26974248
```

```
>>> a.insert(3,4)
```

```
>>> a
```

```
[1, 2, 3, 4]
```

```
>>> id(a)
```

```
26974248
```


列表的添加

- ▶ 使用 “*” 运算符扩展列表对象，将列表与整数相乘，生成一个新列表。

```
>>> a = [1,2,3]
```

```
>>> b = a
```

```
>>> id(a)
```

```
36904552
```

```
>>> id(b)
```

```
36904552
```

```
>>> b = a * 3
```

```
>>> b
```

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

```
>>> id(b)
```

```
36904072
```

列表的添加-Note 1

- ▶ Python采用的是基于值的自动内存管理方式，当为对象修改值时，并不是真的直接修改变量的值，而是使变量指向新的值。
- ▶ 对于列表、集合、字典等可变序列类型而言，情况稍微复杂一些。
 - 如果是直接修改序列变量的值，则与Python普通变量的情况是一样的
 - 如果是通过下标来修改序列中元素的值或通过可变序列对象自身提供的方法来增加和删除元素时，序列对象在内存中的起始地址是不变的，仅仅是被改变值的元素地址发生变化。

列表的添加-Note 1

```
>>> a = [1,2,3]
```

```
>>> b = [1,2,4]
```

```
>>> id(a) == id(b)
```

False

```
>>> id(a[0]) == id(b[0])
```

True

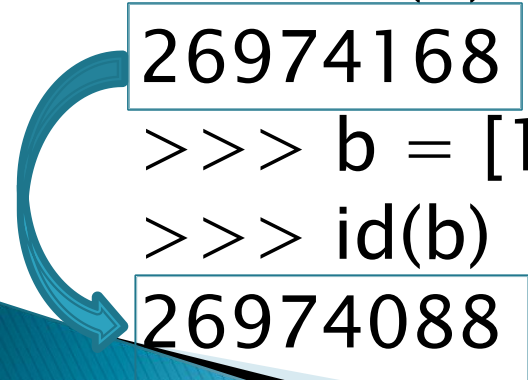
```
>>> id(b)
```

26974168

```
>>> b = [1,2,5]
```

```
>>> id(b)
```

26974088



列表的添加-Note 1

```
>>> a = [1,2,3]
```

```
>>> id(a)
```

```
36308992
```

```
>>> a.append(4)
```

```
>>> id(a)
```

```
36308992
```

```
>>> a.remove(3)
```

```
>>> id(a)
```

```
36308992
```

```
>>> a[0]= 5
```

```
>>> id(a)
```

```
36308992
```



列表的添加-Note 2

```
>>> a = [1,2,3]
```

```
>>> b = a
```

```
>>> b
```

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

```
>>> b[0] = 4
```

```
>>> a
```

```
?
```

列表的添加-Note 3

```
>>> a = [[0]*3]*4
```

```
>>> a
```

```
[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
```

```
>>> a[0][0] = 1
```

```
>>> a
```

```
?
```

列表的添加-summary

- ▶ 大量元素的添加不建议用 “+” 运算符
- ▶ 大量元素的添加建议用列表对象的append()方法
- ▶ 列表对象的extend()方法可以将另一个迭代对象的所有元素添加至该列表对象尾部
- ▶ 列表的insert()可以在列表的任意位置插入元素，但由于列表的自动内存管理功能，此方法会影响处理速度
- ▶ 列表对象赋值给变量后修改，或 “*” 运算符生成列表赋值给变量后修改，注意修改的影响

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的删除

- ▶ 基于索引的删除
 - 使用del命令删除
 - 使用列表的pop()方法删除
- ▶ 基于值的删除
 - 使用列表对象的remove()方法删除

列表的删除

- ▶ 使用del命令删除列表中的指定位置上的元素。del命令也可以直接删除整个列表。

```
>>> a = [1,2,3]
```

```
>>> del a[0]
```

```
>>> a
```

```
[2, 3]
```

```
>>> del a
```

```
>>> a
```

```
Traceback (most recent call last):
```

```
File "<pyshell#65>", line 1, in <module>
```

```
    a
```

```
NameError: name 'a' is not defined
```

列表的删除

- ▶ 列表的pop()方法删除并返回指定（默认为最后一个）位置上的元素，如果给定的索引超出了列表的范围则抛出异常。

```
>>> a = [1,2,3]
```

```
>>> a.pop()
```

```
3
```

```
>>> a
```

```
[1, 2]
```

列表的删除

- ▶ 列表对象的remove()方法删除首次出现的指定元素，如果列表中不存在要删除的元素，则抛出异常。

```
>>> a = [1,2,3]
```

```
>>> a.remove(2)
```

```
>>> a
```

```
[1, 3]
```

列表的删除-Note 1

```
>>> a = [1,2,3]
```

```
>>> for i in a:
```

```
    if i == 3:
```

```
        a.remove(i)
```

```
>>> a
```

```
?
```

```
>>> a = [1,2,3,3]
```

```
>>> for i in a:
```

```
    if i == 3:
```

```
        a.remove(i)
```

```
>>> a
```

```
?
```

列表的删除-Note 1

- ▶ 两个a的本质区别在于，第一a中没有连续的“3”，而第二a中存在连续的“3”。
- ▶ 原因是列表的自动内存管理功能
 - 在删除列表元素时，Python会自动对列表内存进行收缩并移动列表元素以保证所有元素之间没有空隙
 - 增加列表元素时也会自动扩展内存并对元素进行移动以保证元素之间没有空隙。
 - 每当插入或删除一个元素之后，该元素位置后面所有元素的索引就都改变了。

列表的删除-Note 2

```
>>> a = [1,2,3]
>>> for i,v in
enumerate(a):
    if i in (1,2):
        del a[i]
```

```
>>> a
?
```

```
>>> a = [1,2,3]
>>> for i,v in
enumerate(a):
    if i in (1,2):
        a.pop(i)
```

```
2
>>> a
```

列表的删除-Note 1 Solution

```
>>> a = [1,2,3,3]
```

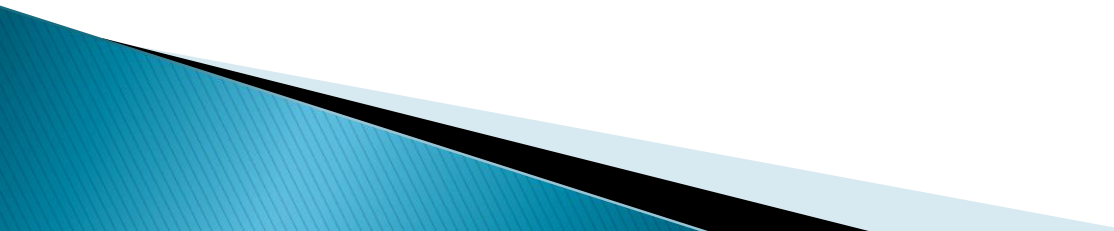
```
>>> for i in a[:]:
```

```
    if i == 3:
```

```
        a.remove(i)
```

```
>>> a
```

```
[1, 2]
```



列表的删除-Note 2 Solution

```
>>> a = [1,2,3]
```

```
>>> idx = [1,2]
```

```
>>> for i in sorted(idx,reverse=True):  
    del a[i]
```

```
>>> a
```

```
[1]
```



列表的删除

- ▶ `remove()`方法以及使用`pop()`函数弹出列表非尾部元素和使用`del`命令删除列表非尾部元素的情况，将会影响数据处理的速度

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的访问(index->value)

- ▶ 使用下标直接访问列表元素

```
>>> a = [1,2,3]
```

```
>>> a[1]
```

```
2
```

- ▶ 如果指定下标不存在，则抛出异常

```
>>> a[4]
```

Traceback (most recent call last):

```
File "<pyshell#2>", line 1, in <module>  
    a[4]
```

IndexError: list index out of range

列表的访问(index->value)

```
>>> i = 3
```

```
>>> if i < len(a):
```

```
    print(a[i])
```

```
else:
```

```
    print("index oor")
```



列表的访问(value->index)

- ▶ 使用列表对象的index方法获取指定元素首次出现的下标

```
>>> a = [1,2,3,1,2,3,3,2,1]
```

```
>>> a.index(3)
```

```
2
```

```
>>> a.index(10)
```

Traceback (most recent call last):

File "<pyshell#16>", line 1, in <module>

a.index(10)

ValueError: 10 is not in list

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的判断

```
>>> a = [1,2,3]
```

```
>>> 1 in a
```

```
True
```

```
>>> 4 in a
```

```
False
```

```
>>> a = [[1],[2],[3]]
```

```
>>> 1 in a
```

```
False
```

```
>>> [1] in a
```

```
True
```

```
>>> a = [1,2,3]
```

```
>>> b = [4,5,6]
```

```
>>> [1,4] in zip(a,b)
```

```
False
```

```
>>> (1,4) in zip(a,b)
```

```
True
```

```
>>> for x,y in
```

```
zip(a,b):
```

```
    print(x,y)
```


列表的判断

- ▶ 如果需要判断列表中是否存在指定的值，可以使用列表的count()方法。如果存在则返回大于0的数，如果返回0则表示不存在。

```
>>> a = [1,2,3,3]
```

```
>>> a.count(1)
```

```
1
```

```
>>> a.count(3)
```

```
2
```

```
>>> a.count(0)
```

```
0
```



Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作: 序列解包和生成器推导式

列表的操作:切片

- ▶ 切片是Python序列的操作之一，适用于列表、元组、字符串、range对象等类型。
- ▶ 切片可以用来截取列表中的任何部分，得到一个**新列表**，也可以通过切片来修改和删除列表中部分元素，甚至可以通过切片为列表增加元素。
- ▶ 与使用下标访问列表元素的方法不同，切片操作不会因为下标越界而抛出异常。

列表的操作:切片

```
>>> a = [1,2,3,4,5]
```

```
>>> b = a[::]
```

```
>>> id(a),id(b)
```

```
(36308392,  
15243464)
```

```
>>> a[1:]
```

```
[2, 3, 4, 5]
```

```
>>> a[1:5]
```

```
[2, 3, 4, 5]
```

```
>>> a[1:5:2]
```

```
[2, 4]
```

```
>>> a = [1,2,3,4,5]
```

```
>>> a[::-1]
```

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

```
>>> a[::-2]
```

```
[5, 3, 1]
```

```
>>> a[::2]
```

```
[1, 3, 5]
```

列表的操作:切片

```
>>> a = [1,2,3]
```

```
>>> a[3:]
```

```
[]
```

```
>>> a[3:] = [4]
```

```
>>> a
```

```
[1, 2, 3, 4]
```

```
>>> a[1:3] = []
```

```
>>> a
```

```
[1, 4]
```

```
>>> a = [1,2,3,4]
```

```
>>> a[4:] = [5]
```

```
>>> a
```

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

```
>>> a[::2] = [0]*3
```

```
>>> a
```

```
[0, 2, 0, 4, 0]
```

```
>>> a[::2] = [1,3,5]
```

```
>>> a
```

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

列表的操作:切片

- ▶ 使用del命令和切片删除列表元素

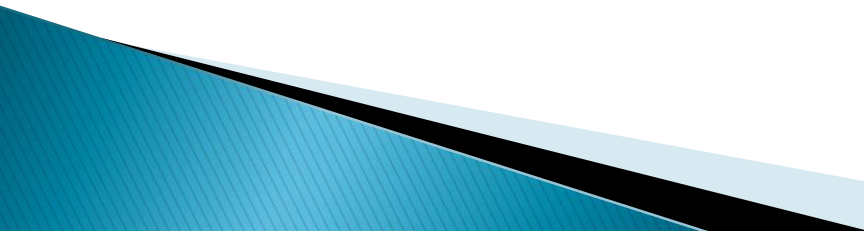
```
>>> a
```

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

```
>>> del a[:3]
```

```
>>> a
```

```
[4, 5]
```



列表的操作:切片

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a
>>> id(b),id(a)
(38744016, 38744016)
>>> a[2] = 6
>>> a
[1, 2, 6, 4, 5]
>>> b
[1, 2, 6, 4, 5]
```

```
>>> a = [1, 2, 3, 4, 5]
>>> b = a[:]
>>> id(b),id(a)
(34379736, 34379216)
>>> a == b
True
>>> a[2]=6
>>> a
[1, 2, 6, 4, 5]
>>> b
[1, 2, 3, 4, 5]
```

列表的操作:切片

```
>>> x = [[1,2],3,4,5]
```

```
>>> y = x[:] #Shallow Copy
```

```
>>> y
```

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

```
>>> y[0][0] = 0
```

```
>>> y
```

```
[[0, 2], 3, 4, 5]
```

```
>>> x
```

```
[[0, 2], 3, 4, 5]
```


Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、**排序**、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的操作:排序

- ▶ 列表对象的sort方法进行原地排序，支持多种不同的排序方法

```
>>> a = [1,2,3,4,11,22,33,44]
```

```
>>> random.shuffle(a)
```

```
>>> a
```

```
[4, 1, 44, 22, 11, 33, 2, 3]
```

```
>>> a.sort()
```

```
>>> a
```

```
[1, 2, 3, 4, 11, 22, 33, 44]
```

```
>>> a.sort(reverse=True)
```

```
>>> a
```

```
[44, 33, 22, 11, 4, 3, 2, 1]
```

```
>>> a.sort(key = lambda x:len(str(x)))
```

```
# a = [4, 3, 2, 1, 44, 33, 22, 11]
```

列表的操作:排序

- ▶ 使用列表对象reverse()方法将列表元素原地逆序

```
>>> a = [1,2,3,4,11,22,33,44]
```

```
>>> a.reverse()
```

```
>>> a
```

```
[44, 33, 22, 11, 4, 3, 2, 1]
```

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、**BIF**、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的操作:BIF

len()	返回列表中的元素个数，同样适用于元组、字典、字符串等等
max()	返回列表中的最大元素，同样适用于元组、range。
min()	返回列表中的最小元素，同样适用于元组、range。
sum()	数值型列表元素的求和运算，对非数值型列表运算则出错
sorted()	对列表进行排序并返回新列表
reversed()	对列表元素进行逆序排列并返回迭代对象
zip()	将多个列表对应位置元素组合为元组，并返回这些元组的列表
enumerate()	枚举列表元素，返回每个元素的下标和值的元组

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

列表的操作:列表推导式

- ▶ 列表推导式是Python程序开发时经常使用的技术之一。
- ▶ 列表推导式使用非常简洁的方式来快速生成满足特定需求的列表，代码具有非常强的可读性

```
>>>a = [i*i for i in range(10)]
```

```
>>> a = []
```

```
>>> for i in range(10):  
    a.append(i*i)
```

```
>>> a
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

列表的操作:列表推导式

- ▶ 使用列表推导式实现嵌套列表的平铺

```
>>> vec = [[1,2,3], [4,5,6], [7,8,9]]
```

```
>>> [num for ele in vec for num in ele]
```

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

- ▶ 过滤列表中不符合条件的元素

```
>>> a = [-1,-4,6,7.5,-2.3,9,-11]
```

```
>>> [i for i in a if i>0]
```

```
[6, 7.5, 9]
```


列表的操作:列表推导式

>>> [(x, y) for x in range(3) for y in range(3)]

[(0, 0), (0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (2, 0),
(2, 1), (2, 2)]

>>> [(x, y) for x in [1, 2, 3] for y in [3, 1, 4] if x
!= y]

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

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

元组

- ▶ 元组和列表类似，但属于不可变序列，元组一旦创建，用任何方法都不可以修改其元素。
- ▶ 元组的定义方式和列表相同，但定义时所有元素是放在一对圆括号“（”和“）”中，而不是方括号中。

元组的创建

```
>>> x = 3
```

```
>>> x
```

```
3
```

```
>>> x = 3,
```

```
>>> x
```

```
(3,)
```

```
>>> type(x)
```

```
<class 'tuple'>
```

```
>>> x=()
```

元组的创建

```
>>> x = [1,2,3]
```

```
>>> tuple(x)
```

```
(1, 2, 3)
```

```
>>> x = tuple()
```

```
>>> x
```

```
()
```

```
>>> tuple('abcd')
```

```
('a', 'b', 'c', 'd')
```

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

元组和列表的不同

- ▶ 元组中的数据一旦定义就不允许更改。
- ▶ 元组没有append()、extend()和insert()等方法，无法向元组中添加元素；
- ▶ 元组没有remove()或pop()方法，也无法对元组元素进行del操作，不能从元组中删除元素。
- ▶ tuple()函数接受一个列表参数，并返回一个包含同样元素的元组，而list()函数接受一个元组参数并返回一个列表。从效果上看，tuple()冻结列表，而list()融化元组。

元组和列表的不同

- ▶ **元组的I/O速度比列表更快。** 如果定义常量值序列，而所需的仅是对它进行遍历，那么一般使用元组而不用列表。
- ▶ **元组不允许数据的修变，效果相当于“写保护”，代码更加安全。**
- ▶ **是否用于字典键值。** 元组用作字典键，列表不作为字典键。

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：**序列解包**和生成器推导式

序列解包

- ▶ 对多个变量同时赋值

```
>>> a = (1,2,3)
```

```
>>> x,y,z = a
```

```
>>> keys=['a','b','c','d']
```

```
>>> values=[1,2,3,4]
```

```
>>> for k,v in zip(keys,values):  
    print(k,v)
```

Outline

▶ 7.1 列表

- 列表的创建
- 列表的添加和删除
- 列表的访问和判断
- 列表的操作:切片、排序、BIF、列表推导式

▶ 7.2 元组

- 元组的创建
- 元组和列表的不同
- 元组相关的操作：序列解包和生成器推导式

生成器推导式

- ▶ 生成器推导式与列表推导式非常接近，只是生成器推导式使用圆括号，而不是列表推导式所使用的方括号。
- ▶ 与列表推导式不同的是，生成器推导式的结果是一个生成器对象，而不是列表，也不是元组。
- ▶ 不管用哪种方法访问其元素，当所有元素访问结束以后，如果需要重新访问其中的元素，必须重新创建该生成器对象。

Thanks for listening

College of Computer Science
Nankai University