



Python基础教程

第3章 基础语法-列表、元组、字典





1) 定义：Python中的列表用`[]`标识，是最通用的复合数据类，它支持字符，数字，字符串甚至可以包含列表（所谓嵌套）。

```
>>> list1 = [1, 2, 'hell', [1, 2, 3]]
>>> type(list1)
<class 'list'>
>>> list1
[1, 2, 'hell', [1, 2, 3]]
```

2) 列表主要的操作包括：

索引：操作符（`[]`）。

切片：操作符（`[开始下标 : 结束下标]`）

相加：列表连接运算符（`+`）

乘法：重复操作符星号（`*`）。





1) **索引**操作：用来访问list中**每一个位置**的元素——索引从0开始递增，如：

```
>>> list1[0]
1
>>> list1[3]
[1, 2, 3]
>>> list1[-1]
[1, 2, 3]
```

2) **切片**（分片）操作：用来访问list中**一定范围内**的元素，如：

```
>>> tag = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> tag[3:8]
[4, 5, 6, 7, 8]
>>> tag[3:-1]
[4, 5, 6, 7, 8, 9]
>>> tag[2:-3]
[3, 4, 5, 6, 7]
```





2) **切片**（分片）操作：用来访问list中**一定范围内**的元素，如：

```
>>> list_1 = [ 'Mike', 'John', 'Bob', 1, 2, 3, 'A', 'B', 'C', {'A': 1}, [ 1.1, 2.2, 3.3 ] ]
>>> list_1[ 2 : 7 ]
['Bob', 1, 2, 3, 'A']
>>> list_1[ : 8 ]
['Mike', 'John', 'Bob', 1, 2, 3, 'A', 'B']
>>> list_1[ 5 : -1 ]
[3, 'A', 'B', 'C', {'A': 1}]
>>> list_1[ : -3 ]
['Mike', 'John', 'Bob', 1, 2, 3, 'A', 'B']
>>> list_1[ : ]
['Mike', 'John', 'Bob', 1, 2, 3, 'A', 'B', 'C', {'A': 1}, [1.1, 2.2, 3.3]]
```





3) **相加**操作：使用加运算符可以进行列表的**连接**操作。

```
>>> [1, 2, 3] + [4, 5, 6]  
[1, 2, 3, 4, 5, 6]
```

4) **乘法**操作：用数字N乘以一个列表会**生成新的列表**，新列表中，原列表被重复N次，原列表不变。

```
>>> list_2 = ['python']  
>>> list_2 * 5  
['python', 'python', 'python', 'python', 'python']  
>>> list_2  
['python']  
>>>
```





1.3 列表对象支持的方法

方法	描述
<code>append(x)</code>	在列表尾部追加单个对象 <code>x</code> ，使用多个参数会引起异常。
<code>count(x)</code>	返回对象 <code>x</code> 在列表中出现的次数。
<code>extend(L)</code>	将列表 <code>L</code> 中的表项添加到列表中，返回 <code>None</code> 。
<code>index(x)</code>	返回列表中匹配对象 <code>x</code> 的第一个列表项的索引，无匹配元素时产生异常。
<code>insert(i, x)</code>	在索引为 <code>i</code> 的元素前插入对象 <code>x</code> 。如： <code>list.insert(0,x)</code> 在第一项前插入对象。返回 <code>None</code> 。
<code>pop(x)</code>	删除列表中索引为 <code>x</code> 的表项，并返回该表项的值。若未指定索引， <code>pop</code> 返回列表最后一项。
<code>remove(x)</code>	删除列表中匹配对象 <code>x</code> 的第一个元素。匹配元素时产生异常。返回 <code>None</code> 。
<code>reverse()</code>	颠倒列表元素的顺序。
<code>sort()</code>	对列表排序，返回 <code>None</code> 。 <code>bisect</code> 模块可用于排序列表项的添加和删除。





1.3 列表对象支持的方法

1) **append(x)**: 用于在列表末尾追加新的对象，如：

```
>>> nums = [1, 2, 3]
>>> nums.append(4)
>>> nums
[1, 2, 3, 4]
>>> nums.append([5, 6, 7])
>>> nums
[1, 2, 3, 4, [5, 6, 7]]
```

2) **count(x)**: 用于统计某个元素在列表中出现的次数，如：

```
>>> ['to', 'be', 'or', 'not', 'to', 'be'].count('to')
2
>>> x = [[1, 2], 1, 1, [2, 1, [1, 2]]]
>>> x.count(1)
2
>>> x.count([1, 2])
1
```



1.3 列表对象支持的方法

3) **extend(L)**: 可在列表的末尾一次性追加另一个序列中的多个值，可扩展原有列表，如：

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

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> x.append(y)
>>> x
[1, 2, 3, [4, 5, 6]]
```

从结果可以看出**extend**方法与**append**方法的不同

4) **index(x)**: 用于从列表中找出某个值第一个匹配项的索引位置，如：

```
>>> greeting = ['Hello', 'Nice', 'to', 'meet', 'you']
>>> greeting.index('Nice')
1
>>> greeting.index('me')
```

```
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    greeting.index('me')
ValueError: 'me' is not in list
```





1.3 列表对象支持的方法

5) **insert(i,x)**: 用于将对象插入到列表中, 如:

```
>>> numbers = [1, 2, 3, 5, 6, 7]
>>> numbers.insert(3, 'four')
>>> numbers
[1, 2, 3, 'four', 5, 6, 7]
```

6) **pop(x)**: 移除列表中的一个元素 (默认是最后一个), 并且返回该元素的值, 如:

```
>>> numbers = [1, 2, 3, 4, 5]
>>> numbers.pop()
5
>>> numbers
[1, 2, 3, 4]
>>> numbers.pop(0)
1
>>> numbers
[2, 3, 4]
```



1.3 列表对象支持的方法

7) **remove(x)**: 用于移除列表中某个值的第一个匹配项, 如:

```
>>> words = ['to', 'be', 'or', 'not', 'to', 'be']
>>> words.remove('be')
>>> words
['to', 'or', 'not', 'to', 'be']
```

8) **reverse()**: 将列表中的元素反向存放, 如:

```
>>> x = [1, 2, 3]
>>> y = x.reverse()
>>> x
[3, 2, 1]
>>> y
>>> print(x, y)
[3, 2, 1] None
```





1.3 列表对象支持的方法

9) **sort()**: 用于在原位置对列表进行排序, 在“原位置排序”意味着改变原来的列表, 从而让其中的元素能按一定的顺序排列, 而不是简单地返回一个已排序的列表副本, 如:

```
>>> x = [4, 7, 1, 2, 9, 6]
>>> y = x.sort()
>>> x
[1, 2, 4, 6, 7, 9]
>>> y
>>> print( x )
[1, 2, 4, 6, 7, 9]
>>> print( y )
None
```

```
>>> mylist = [ 'alpha', 'Beta', 'gamma' ]
>>> mylist.sort()
>>> mylist
['Beta', 'alpha', 'gamma']
>>> mylist.sort(key = str.lower)
>>> mylist
['alpha', 'Beta', 'gamma']
```





2.1 元组：不可变序列

Python中的元组用()标识，内部元素用逗号隔开。但是元素不能二次赋值，相当于只读列表。

创建元组的语法很简单：如果用逗号分隔一些值，就会自动创建元组。

```
>>> 1, 2, 3
(1, 2, 3)
>>> (1, 2, 3)
(1, 2, 3)
>>> ()
()
>>> 42,
(42,)
```





2.1 元组：不可变序列

元组同列表一样，也是一种序列，元组也有索引、切片、相加、乘法等操作。

由于元组可以看做是只读列表，因而不可以修改元组中元素的值，而列表可以。

```
>>> a = [1, 2, 3]
>>> a[0] = 4
>>> a
[4, 2, 3]
>>>
>>> b = (1, 2, 3)
>>> b[0] = 4
```

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

```
File "<pyshell#118>", line 1, in <module>
```

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

```
TypeError: 'tuple' object does not support item assignment
```



索引：操作符（`[]`）。

切片：操作符（`[开始下标 : 结束下标]`）

相加：列表连接运算符（`+`）

乘法：重复操作符星号（`*`）。

```
>>> tuple1 = (1, 2, 3, 'A', 'B', 'C', {'A':1}, [4, 5, 6], (7, 8, 9))
>>> tuple1
(1, 2, 3, 'A', 'B', 'C', {'A': 1}, [4, 5, 6], (7, 8, 9))
>>> tuple1[0] #元组索引操作
1
>>> tuple1[3:5] #元组切片操作
('A', 'B')
>>> tuple1 + ('a', 'b', 'c') #元组相加操作
(1, 2, 3, 'A', 'B', 'C', {'A': 1}, [4, 5, 6], (7, 8, 9), 'a', 'b', 'c')
>>> tuple1 * 2 #元组乘法操作，即重复操作
(1, 2, 3, 'A', 'B', 'C', {'A': 1}, [4, 5, 6], (7, 8, 9), 1, 2, 3, 'A', 'B', 'C', {'A': 1}, [4, 5, 6], (7, 8, 9))
>>>
```





- 元组不可修改:

- 不能添加元素

- 不能删除元素

- 不能使用index去搜索

- 可以用in操作去查询

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

```
>>> 1 in x
```

```
True
```

```
>>> 9 in x
```

```
False
```

- 优点:

- 比list快

- 用于常量定义, 或字典的键值





Python中的字典用“{ }”标识，由索引(key)和它对应的值value组成。

字典(dictionary)是除列表外Python之中最灵活的内置数据结构类型。

```
>>> aDict = { 'host': 'earth' }
>>> aDict[ 'port' ] = 80
>>> aDict
{'host': 'earth', 'port': 80}
>>> aDict.keys()
dict_keys(['host', 'port'])
>>> aDict[ 'host' ]
'earth'
>>> for key in aDict:
    print( key, aDict[ key ] )
```

```
host earth
port 80
```

注意：

- 列表是有序的对象结合，字典是无序的对象集合。
- 字典当中的元素是通过键来存取的，而不是通过偏移存取。





```
>>> dict = {}
>>> dict['第一个'] = '这是第一个'
>>> dict[2] = '这是第二个'
>>> tinydict = { 'name': '张三', 'code': 6734, 'dept': 'sales' }
>>> print( dict['第一个'] )
这是第一个
>>> print( dict[2] )
这是第二个
>>> print( tinydict )
{'name': '张三', 'code': 6734, 'dept': 'sales'}
>>> print( tinydict.keys() )
dict_keys(['name', 'code', 'dept'])
>>> print( tinydict.values() )
dict_values(['张三', 6734, 'sales'])
>>> dict['第一个'] = '这是第三个'
>>> dict
{'第一个': '这是第三个', 2: '这是第二个'}
```





3.3 字典对象支持的方法

方法	描述
<code>keys()</code>	返回字典中键的列表。
<code>values()</code>	返回字典中值的列表。
<code>items()</code>	返回 tuples 的列表，每个 tuple 由字典的键和相应值组成。
<code>clear()</code>	删除字典的所有条目。
<code>copy()</code>	返回字典高层结构的一个拷贝，但不复制嵌入结构，而只复制对那些结构的引用。
<code>update(x)</code>	用字典 x 中的键值对更新字典内容。
<code>get(x, [y])</code>	返回键 x ，若未找到该键返回 None ，若提供 y ，则未找到 x 时返回 y 。





3.3 字典对象支持的方法

1) **keys()**: 将字典中的键以列表的形式返回, 如:

```
>>> d = {1: 'John', 3: 'Merkel'}  
>>> d.keys()  
dict_keys([1, 3])  
>>> list(d.keys())  
[1, 3]
```

2) **values()**: 以列表的形式返回字典中的值, 与返回的键的列表不同的是, 返回值的列表中可以包含重复元素, 如:

```
>>> d2 = {1:1, 2:2, 3:1}  
>>> d2.values()  
dict_values([1, 2, 1])  
>>> list(d2.values())  
[1, 2, 1]
```





3.3 字典对象支持的方法

3) **items()**: 将字典所有的项以列表的方式返回，列表中的每项都表示为（键，值）对的形式，如：

```
>>> d = {'title': 'Python Web Site', 'url': 'http://www.python.org', 'spam': 0}
>>> d.items()
[('url', 'http://www.python.org'), ('spam', 0), ('title', 'Python Web Site')]
```

4) **clear()**: 用于清除字典中的所有项，这是个原地操作（类似于`list.sort`），无返回值（或者说返回值为`None`），如：

```
>>> d3 = {}
>>> d3['name'] = 'Bob'
>>> d3['age'] = 23
>>> d
{1: 'John', 3: 'Merkel'}
>>> d.clear()
>>> d
{}
>>>
```





3.3 字典对象支持的方法

5) **copy()**: 返回一个具有相同键值的新字典（该方法实现的是浅复制），如：

```
>>> x = { 'username': 'admin', 'machines': [ 'foo', 'bar', 'baz' ] }
>>> x.copy()
{'username': 'admin', 'machines': ['foo', 'bar', 'baz']}
>>> y = x.copy()
>>> y
{'username': 'admin', 'machines': ['foo', 'bar', 'baz']}
>>> y[ 'username' ] = 'abc'
>>> y
{'username': 'abc', 'machines': ['foo', 'bar', 'baz']}
>>> x
{'username': 'admin', 'machines': ['foo', 'bar', 'baz']}
>>> y[ 'machines' ].remove( 'foo' )
>>> y
{'username': 'abc', 'machines': ['bar', 'baz']}
>>> x
{'username': 'admin', 'machines': ['bar', 'baz']}
```





3.3 字典对象支持的方法

5) **copy()**: 返回一个具有相同键值的新字典（该方法实现的是浅复制），如：

```
>>> x = { 'username' : 'admin', 'machines' : [ 'foo', 'bar', 'daz' ] }
>>> y = x.copy()
>>> y
{'username': 'admin', 'machines': ['foo', 'bar', 'daz']}
>>> y[ 'username' ] = 'abc'; y[ 'machines' ].remove( 'foo' )
>>> y
{'username': 'abc', 'machines': ['bar', 'daz']}
>>> x
{'username': 'admin', 'machines': ['bar', 'daz']}
>>> x[ 'username' ] = 123
>>> x
{'username': 123, 'machines': ['bar', 'daz']}
>>> y
{'username': 'abc', 'machines': ['bar', 'daz']}
```





- Python中对象的赋值都是进行对象引用（内存地址）传递；
- 使用copy(), 可以进行对象的浅拷贝，它复制了对象，但对于对象中的元素，依然使用原始的引用；
- 如果需要复制一个容器对象，以及它里面的所有元素（包含元素的子元素），可以使用deepcopy() 进行深拷贝。

```
>>> from copy import deepcopy

>>> d = {}

>>> d['names'] = ['Alfred', 'Bertrand']

>>> c = d.copy()

>>> dc = deepcopy(d)

>>> d['names'].append('clive')
```



```
>>> from copy import deepcopy

>>> d = {}

>>> d['names'] = ['Alfred', 'Bertrand']

>>> c = d.copy()

>>> dc = deepcopy(d)

>>> d['names'].append('clive')

{'names': ['Alfred', 'Bertrand', 'clive']}

>>> dc

{'names': ['Alfred', 'Bertrand']}

>>> d

{'names': ['Alfred', 'Bertrand', 'clive']}
```





copy模块:

1. copy.copy
(浅拷贝):只
拷贝父对象,
不会拷贝对象
的内部子对象。

2. copy.deepcopy
(深拷贝):
拷贝对象及其
子对象。

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

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

```
>>> c = a.copy()
```

```
>>> d = a.deepcopy()
```

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

```
File "<pyshell#3>", line 1, in <module>
```

```
d = a.deepcopy()
```

```
AttributeError: 'list' object has no attribute 'deepcopy'
```

```
>>> import copy
```

```
>>> d = copy.deepcopy(a)
```

```
>>> a[4].append('c')
```

```
>>> a
```

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

```
>>> b
```

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

```
>>> c
```

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

```
>>> d
```

```
[1, 2, 3, 4, ['a', 'b']]
```



3.3 字典对象支持的方法

6) **update(x)**: 可以利用一个字典项更新另一个字典, 如:

```
>>> d = {'title': 'Python Web Site', 'url': 'http://www.python.org', 'spam': 0}
>>> x = {'title': 'Python Language Website'}
>>> d.update(x)
>>> d
{'url': 'http://www.python.org', 'spam': 0, 'title': 'Python Language Website'}
```

7) **get(x, [y])**: 更宽松的访问字典的一种方法。一般来说, 当访问字典中不存在的项时会出错, 而get不会, 如:

```
>>> d = { }
>>> print( d[ 'name' ] )
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print( d[ 'name' ] )
KeyError: 'name'
>>> print(d.get( 'name' ))
None
```





集合（set）是一个无序的**不重复**元素序列。

可以使用大括号 `{ }` 或者 `set()` 函数创建集合，

【注意】：创建一个空集合必须用 `set()` 而不是 `{ }`，因为 `{ }` 是用来创建一个空字典。

【创建格式】：

```
parame = {value01,value02,...}
```

或者

```
set(value)
```





4.2 集合操作实例

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                # 这里演示的是去重功能
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket           # 快速判断元素是否在集合内
True
>>> 'crabgrass' in basket
False

>>> # 下面展示两个集合间的运算.
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                        # 集合a中包含而集合b中不包含的元素
{'r', 'd', 'b'}
>>> a | b                        # 集合a或b中包含的所有元素
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                        # 集合a和b中都包含了的元素
{'a', 'c'}
>>> a ^ b                        # 不同时包含于a和b的元素
{'r', 'd', 'b', 'm', 'z', 'l'}
```





4.3 集合对象支持的方法

集合内置方法完整列表

方法	描述
<code>add()</code>	为集合添加元素
<code>clear()</code>	移除集合中的所有元素
<code>copy()</code>	拷贝一个集合
<code>difference()</code>	返回多个集合的差集
<code>difference_update()</code>	移除集合中的元素，该元素在指定的集合也存在。
<code>discard()</code>	删除集合中指定的元素
<code>intersection()</code>	返回集合的交集
<code>intersection_update()</code>	返回集合的交集。
<code>isdisjoint()</code>	判断两个集合是否包含相同的元素，如果没有返回 True，否则返回 False。
<code>issubset()</code>	判断指定集合是否为该方法参数集合的子集。





4.3 集合对象支持的方法

<code>issuperset()</code>	判断该方法的参数集合是否为指定集合的子集
<code>pop()</code>	随机移除元素
<code>remove()</code>	移除指定元素
<code>symmetric_difference()</code>	返回两个集合中不重复的元素集合。
<code>symmetric_difference_update()</code>	移除当前集合中在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。
<code>union()</code>	返回两个集合的并集
<code>update()</code>	给集合添加元素





4.3 集合对象支持的方法

1) `s.add(x)`: 将元素 `x` 添加到集合 `s` 中, 如果元素已存在, 则不进行任何操作。

```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.add("Facebook")
>>> print(thisset)
{'Taobao', 'Facebook', 'Google', 'Runoob'}
```

1) `s.update(x)`: 也可以添加元素, 且参数可以是列表, 元组, 字典等。

```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.update({1,3})
>>> print(thisset)
{1, 3, 'Google', 'Taobao', 'Runoob'}
>>> thisset.update([1,4],[5,6])
>>> print(thisset)
{1, 3, 4, 5, 6, 'Google', 'Taobao', 'Runoob'}
>>>
```





4.3 集合对象支持的方法

3) `s.remove(x)`: 将元素 `x` 从集合 `s` 中移除, 如果元素不存在, 则会发生错误。

```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.remove("Taobao")
>>> print(thisset)
{'Google', 'Runoob'}
>>> thisset.remove("Facebook")    # 不存在会发生错误
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Facebook'
>>>
```





4.3 集合对象支持的方法

4) `s.discard(x)`: 也是移除集合中的元素，且如果元素不存在，不会发生错误。

```
>>> thisset = set(("Google", "Runoob", "Taobao"))
>>> thisset.discard("Facebook") # 不存在不会发生错误
>>> print(thisset)
{'Taobao', 'Google', 'Runoob'}
```





5. 布尔值

- 布尔值是特殊的整型，用来表示诸如：对与错，真与假，空与非空等概念。
 - (1) 由常量True和False来表示，True表示非空的量，所有非零数；False表示0，None, 空的量等。
 - (2) 如与一个数字相加，True会被当成整数型值“1”，而False则会被当成整数型值“0”。
- 作用：主要用于判断语句中，用来判断：
 - (1) 一个字符串是否是空的
 - (2) 一个运算结果是否为零
 - (3) 一个表达式是否可用
- 注意：True/False是Python语言内定的布尔值，使用true/false，或者TRUE/FALSE是无效的。





5. 布尔值

- 下列对象的布尔值是False:

- None;
- False(布尔类型);
- 所有的值为零的数;
- 0 (整型);
- 0.0 (浮点型);
- 0L (长整型);

- 布尔值应用举例:

```
>>> b = 100 < 200
>>> b
True
>>> bool([])
False
>>> bool('')
False
>>> [] == False
False
>>> bool([]) == False
True
```

```
>>> b = 100 < 200
>>> print(b)
True
>>> b = true
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    b = true
NameError: name 'true' is not defined
```





运算符：用于执行程序代码运算，会针对一个以上操作数来进行运算。例如：2+3,其操作数是2和3，而运算符则是“+”。

Python中有以下类型的操作符：

- [算术运算符](#)
- [关系运算符](#)
- [赋值运算符](#)
- [逻辑运算符](#)
- [位运算符](#)
- [成员运算符](#)
- [身份运算符](#)





6.2 算术运算符

运算符	描述
+	加 - 两个对象相加
-	减 - 得到负数或是一个数减去另一个数
*	乘 - 两个数相乘或是返回一个被重复若干次的字符串
/	除 - x除以y
%	取模 - 返回除法的余数
**	幂 - 返回x的y次幂
//	取整除 - 返回商的整数部分





6.2 算术运算符示例

```
>>> 1 / 3
0.3333333333333333
>>> 1//3
0
>>> 8 % 3
2
>>> 3**5
243
```

```
>>> 'a' + 'b'
'ab'
>>> 'a' * 10
'aaaaaaaaaa'
```



运算符	描述
<code>==</code>	等于 - 比较对象是否相等
<code>!=</code>	不等于 - 比较两个对象是否不相等
<code><></code>	不等于 - 比较两个对象是否不相等
<code>></code>	大于 - 返回x是否大于y
<code><</code>	小于 - 返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。注意，这些变量名的大写。
<code>>=</code>	大于等于 - 返回x是否大于等于y。
<code><=</code>	小于等于 - 返回x是否小于等于y。





6.3 关系运算符示例

>>> # == :等于, 比较对象是否相等

>>> x = 2

>>> y = 2

>>> x == y

True

>>> x = 'str'

>>> y = 'stR'

>>> x == y

False

>>>

>>> # != :不等于, 比较两个对象是否不相等

>>> x = 2

>>> y = 3

>>> x != y

True

>>>

>>> # < :小于, 返回x是否小于y, 所有比较运算符返回1表示真, 返回0表示假, 分别与True和False等价

>>> 5 < 3

False

>>> 3 < 5

True

>>>

>>> # > :大于, 返回x是否大于y

>>> 5 > 3

True





运算符	描述
=	简单的赋值运算符
+=	加法赋值运算符
-=	减法赋值运算符
*=	乘法赋值运算符
/=	除法赋值运算符
%=	取模赋值运算符
**=	幂赋值运算符
//=	取整除赋值运算符





运算符	描述
and	布尔"与" - 如果x为False， x and y返回False， 否则它返回y的计算值。
or	布尔"或" - 如果x是True， 它返回True， 否则它返回y的计算值。
not	布尔"非" - 如果x为True， 返回False。 如果x为False， 它返回True。





6.5 逻辑运算符示例

```
>>> # not :布尔“非”，如果x为True，not x返回False
>>> x = True
>>> not x
False
```

```
>>>
>>> # and :布尔“与”
>>> x = False
>>> y = True
>>> x and y
False
```

```
>>> #此例中，Python不会计算y，因为它知道这个表达式的值肯定是False(因为x是False)，这个现象称为短路计算
>>>
```

```
>>> # or :布尔“或”
>>> x = True
>>> y = False
>>> x or y
True
```





运算符	描述
in	如果在指定的序列中找到值返回 True ，否则返回 False 。
not in	如果在指定的序列中没有找到值返回 True ，否则返回 False 。

```
>>> x = list(range(0, 10))
>>> x
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> 1 in x
True
>>> 10 in x
False
>>> 10 not in x
True
>>> 0 in range(0, 10)
True
```





6.7 身份运算符

身份运算符用于比较两个对象的存储单元

运算符	描述
is	is是判断两个标识符是不是引用自一个对象
is not	is not是判断两个标识符是不是引用自不同对象

```
>>> a = 1
>>> b = 1
>>> a is b
True
>>> a == b
True
>>> a = 1
>>> b = 1.0
>>> a == b
True
>>> a is b
False
```





Python中的对象包含三个要素：`id`、`type`、`value`

- `id`用来唯一标识一个对象，`type`标识对象的类型，`value`是对象的值
- `is`判断的是a对象是否就是b对象，是通过id来判断的
- `==`判断的是a对象的值是否和b对象的值相等，是通过value来判断的

```
>>> a = 1;b = 1
>>> a is b
True
>>> id(a)
1377264816
>>> id(b)
1377264816
>>> c = 1.0
>>> a is c
False
>>> id(c)
61128624
```





位运算符是把数字看作二进制来进行计算。

运算符	描述
&	按位与运算符
	按位或运算符
^	按位异或运算符
~	按位取反运算符
<<	左移动运算符
>>	右移动运算符





```
>>> # & :按位“与”
>>> 5 & 3
1
>>> # | :按位“或”
>>> 5 | 3
7
>>> # ^ :按位“非”
>>> 5 ^ 3
6
>>> # ~ :按位“翻转”
>>> ~ 5
-6
>>> #x的按位翻转是 -(x + 1)
>>>
>>> # << :左移，把一个数的比特向左移一定数目
>>> 2 << 2
8
>>> # >> :右移
>>> 2 >> 2
0
```





6.9 运算符优先级

运算符	描述
**	指数 (最高优先级)
~ + -	按位翻转, 一元加号和减号 (最后两个的方法名为 +@ 和 -@)
* / % //	乘, 除, 取模和取整除
+ -	加法减法
>> <<	右移, 左移运算符
&	位 'AND'
^	位运算符
<= < > >=	比较运算符
<> == !=	等于运算符
= %= /= //= -= += *= **=	赋值运算符
is is not	身份运算符
in not in	成员运算符
not or and	逻辑运算符





6.9 运算符优先级

- 运算符优先级表决定了哪个运算符在别的运算符之前计算。当想要改变它们的计算顺序，需要使用圆括号()，写成类似 $(1 + 2) * 3$ 。
- 运算符通常由左向右结合，即具有相同优先级的运算符按照从左向右的顺序计算。如， $2 + 3 + 4$ 被计算成 $(2 + 3) + 4$ 。一些如赋值运算符那样的运算符是由右向左结合的，即 $a = b = c$ 被处理为 $a = (b = c)$ 。
- 建议使用圆括号来分组运算符和操作数，以便能够明确地指出运算的先后顺序，使程序尽可能地易读，如 $1 + (2 * 3)$ 会比 $1 + 2 * 3$ 更清晰。





6.9 运算符优先级

运算符优先级示例:

```
>>> a = 20
>>> b = 10
>>> c = 15
>>> d = 5
>>> e = 0
>>> e = (a + b) * c / d
>>> print(e)
90.0
>>> e = ((a + b) * c) / d
>>> e
90.0
>>> e = (a + b) * (c / d)
>>> e
90.0
>>> e = a + (b * c) / d
>>> e
50.0
```





思考

- (1) Python有哪些标准数据类型。
- (2) 参考教材和帮助文献，掌握列表、元组和字典的常见使用方法。