

Python3 数据结构

本章节我们主要结合前面所学的知识点来介绍Python数据结构。

列表

Python中列表是可变的，这是它区别于字符串和元组的最重要的特点，一句话概括即：列表可以修改，而字符串和元组不能。

以下是 Python 中列表的方法：

方法	描述
<code>list.append(x)</code>	把一个元素添加到列表的结尾，相当于 <code>a[len(a):] = [x]</code> 。
<code>list.extend(L)</code>	通过添加指定列表的所有元素来扩充列表，相当于 <code>a[len(a):] = L</code> 。
<code>list.insert(i, x)</code>	在指定位置插入一个元素。第一个参数是准备插入到其前面的那个元素的索引，例如 <code>a.insert(0, x)</code> 会插入到整个列表之前，而 <code>a.insert(len(a), x)</code> 相当于 <code>a.append(x)</code> 。
<code>list.remove(x)</code>	删除列表中值为 <code>x</code> 的第一个元素。如果没有这样的元素，就会返回一个错误。
<code>list.pop([i])</code>	从列表的指定位置移除元素，并将其返回。如果没有指定索引， <code>a.pop()</code> 返回最后一个元素。元素随即从列表中被移除。（方法中 <code>i</code> 两边的方括号表示这个参数是可选的，而不是要求你输入一对方括号，你会经常在 Python 库参考手册中遇到这样的标记。）
<code>list.clear()</code>	移除列表中的所有项，等于 <code>del a[:]</code> 。
<code>list.index(x)</code>	返回列表中第一个值为 <code>x</code> 的元素的索引。如果没有匹配的元素就会返回一个错误。
<code>list.count(x)</code>	返回 <code>x</code> 在列表中出现的次数。
<code>list.sort()</code>	对列表中的元素进行排序。
<code>list.reverse()</code>	倒排列表中的元素。
<code>list.copy()</code>	返回列表的浅复制，等于 <code>a[:]</code> 。

下面示例演示了列表的大部分方法：

```
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print(a.count(333), a.count(66.25), a.count('x'))
2 1 0
>>> a.insert(2, -1)
>>> a.append(333)
>>> a
```

```
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index(333)
1
>>> a.remove(333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

注意：类似 insert, remove 或 sort 等修改列表的方法没有返回值。

将列表当做堆栈使用

列表方法使得列表可以很方便的作为一个堆栈来使用，堆栈作为特定的数据结构，最先进入的元素最后一个被释放（后进先出）。用 append() 方法可以把一个元素添加到堆栈顶。用不指定索引的 pop() 方法可以把一个元素从堆栈顶释放出来。例如：

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack
[3, 4]
```

将列表当作队列使用

也可以把列表当做队列用，只是在队列里第一加入的元素，第一个取出来；但是拿列表用作这样的目的效率不高。在列表的最后添加或者弹出元素速度快，然而在列表里插入或者从头部弹出速度却不快（因为所有其他的元素都得一个一个地移动）。

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")         # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
```

```
'Eric'  
>>> queue.popleft()           # The second to arrive now leaves  
'John'  
>>> queue                     # Remaining queue in order of arrival  
deque(['Michael', 'Terry', 'Graham'])
```

列表推导式

列表推导式提供了从序列创建列表的简单途径。通常应用程序将一些操作应用于某个序列的每个元素，用其获得的结果作为生成新列表的元素，或者根据确定的判定条件创建子序列。

每个列表推导式都在 for 之后跟一个表达式，然后有零到多个 for 或 if 子句。返回结果是一个根据表达从其后的 for 和 if 上下文中生成出来的列表。如果希望表达式推导出一个元组，就必须使用括号。

这里我们将列表中每个数值乘三，获得一个新的列表：

```
>>> vec = [2, 4, 6]  
>>> [3*x for x in vec]  
[6, 12, 18]
```

现在我们玩一点小花样：

```
>>> [[x, x**2] for x in vec]  
[[2, 4], [4, 16], [6, 36]]
```

这里我们对序列里每一个元素逐个调用某方法：

```
>>> freshfruit = [' banana', ' loganberry ', 'passion fruit ']  
>>> [weapon.strip() for weapon in freshfruit]  
['banana', 'loganberry', 'passion fruit']
```

我们可以用 if 子句作为过滤器：

```
>>> [3*x for x in vec if x > 3]  
[12, 18]  
>>> [3*x for x in vec if x < 2]  
[]
```

以下是一些关于循环和其它技巧的演示：

```
>>> vec1 = [2, 4, 6]  
>>> vec2 = [4, 3, -9]  
>>> [x*y for x in vec1 for y in vec2]  
[8, 6, -18, 16, 12, -36, 24, 18, -54]  
>>> [x+y for x in vec1 for y in vec2]
```

```
[6, 5, -7, 8, 7, -5, 10, 9, -3]
>>> [vec1[i]*vec2[i] for i in range(len(vec1))]
[8, 12, -54]
```

列表推导式可以使用复杂表达式或嵌套函数：

```
>>> [str(round(355/113, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

嵌套列表解析

Python的列表还可以嵌套。

以下实例展示了3X4的矩阵列表：

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

以下实例将3X4的矩阵列表转换为4X3列表：

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

以下实例也可以使用以下方法来实现：

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

另外一种实现方法：

```
>>> transposed = []
>>> for i in range(4):
...     # the following 3 lines implement the nested listcomp
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
```

```
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

del 语句

使用 del 语句可以从一个列表中依索引而不是值来删除一个元素。这与使用 pop() 返回一个值不同。可以用 del 语句从列表中删除一个切割，或清空整个列表（我们以前介绍的方法是给该切割赋一个空列表）。例如：

```
>>> a = [-1, 1, 66.25, 333, 333, 1234.5]
>>> del a[0]
>>> a
[1, 66.25, 333, 333, 1234.5]
>>> del a[2:4]
>>> a
[1, 66.25, 1234.5]
>>> del a[:]
>>> a
[]
```

也可以用 del 删除实体变量：

```
>>> del a
```

元组和序列

元组由若干逗号分隔的值组成，例如：

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

如你所见，元组在输出时总是有括号的，以便于正确表达嵌套结构。在输入时可能有或没有括号，不过括号通常是必须的（如果元组是更大的表达式的一部分）。

集合

集合是一个无序不重复元素的集。基本功能包括关系测试和消除重复元素。

可以用大括号({})创建集合。注意：如果要创建一个空集合，你必须用 set() 而不是 {} ；后者创建一个空的字典，下一节我们会介绍这个数据结构。

以下是一个简单的演示：

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                # 删除重复的
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket           # 检测成员
True
>>> 'crabgrass' in basket
False

>>> # 以下演示了两个集合的操作
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a                            # a 中唯一的字母
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                        # 在 a 中的字母，但不在 b 中
{'r', 'd', 'b'}
>>> a | b                        # 在 a 或 b 中的字母
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                        # 在 a 和 b 中都有的字母
{'a', 'c'}
>>> a ^ b                        # 在 a 或 b 中的字母，但不同时在 a 和 b 中
{'r', 'd', 'b', 'm', 'z', 'l'}
```

集合也支持推导式：

```
>>> a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

字典

另一个非常有用的 Python 内建数据类型是字典。

序列是以连续的整数为索引，与此不同的是，字典以关键字为索引，关键字可以是任意不可变类型，通常用字符串或数值。

理解字典的最佳方式是把它看做无序的键=>值对集合。在同一个字典之内，关键字必须是互不相同。

一对大括号创建一个空的字典：{}

这是一个字典运用的简单例子：

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
```

```
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> list(tel.keys())
['irv', 'guido', 'jack']
>>> sorted(tel.keys())
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
>>> 'jack' not in tel
False
```

构造函数 dict() 直接从键值对元组列表中构建字典。如果有固定的模式，列表推导式指定特定的键值对：

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

此外，字典推导可以用来创建任意键和值的表达式词典：

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}
```

如果关键字只是简单的字符串，使用关键字参数指定键值对有时候更方便：

```
>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

遍历技巧

在字典中遍历时，关键字和对应的值可以使用 items() 方法同时解读出来：

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

在序列中遍历时，索引位置和对应该值可以使用 enumerate() 函数同时得到：

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):  
...     print(i, v)  
...  
0 tic  
1 tac  
2 toe
```

同时遍历两个或更多的序列，可以使用 `zip()` 组合：

```
>>> questions = ['name', 'quest', 'favorite color']  
>>> answers = ['lancelot', 'the holy grail', 'blue']  
>>> for q, a in zip(questions, answers):  
...     print('What is your {0}? It is {1}.'.format(q, a))  
...  
What is your name? It is lancelot.  
What is your quest? It is the holy grail.  
What is your favorite color? It is blue.
```

要反向遍历一个序列，首先指定这个序列，然后调用 `reversed()` 函数：

```
>>> for i in reversed(range(1, 10, 2)):  
...     print(i)  
...  
9  
7  
5  
3  
1
```

要按顺序遍历一个序列，使用 `sorted()` 函数返回一个已排序的序列，并不修改原值：

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']  
>>> for f in sorted(set(basket)):  
...     print(f)  
...  
apple  
banana  
orange  
pear
```

参阅文档

- [Python3 列表](#)
- [Python3 元组](#)

● Python3 字典

← Python3 函数

Python3 模块 →



3 篇笔记



写笔记