



第二章 PYTHON数据结构

序列(列表, 元组)

Python常见内置数据类型

序列类型

- 列表：可变对象
- 元组：不可变
- 字符串：不可变

集合：无重复元素类型

字典：唯一映射类型

2.1 列表

列表：一个列表中可以包含任意个数据，每一个数据称为元素。

创建一个列表的最简单方法是将列表元素放在一对方括号（“[”和“]”）内并以逗号分隔，并用“=”将一个列表赋值给变量。

列表通过Python内置的list类定义。也可以使用list类的构造函数来创建列表。

Python允许同一个列表中元素的数据类型不同，可以是整数、字符串等基本类型，也可以是列表、集合及其他类型的对象。

2.2 列表的基本操作

1、下标访问元素(正向索引)

- 列表中的元素可以通过下标（索引）运算符来访问。
- 列表的下标从0开始的。如果一个列表的长度为 r ，则合法的下标在0到 $r-1$ 之间。
- `list[index]`可以像变量一样使用，进行读取或写入，所以它也被称为下标变量。

2、反向索引

- Python也允许使用负数作为下标记来引用相对于列表末端的位置。
- 反向索引的范围为 $-r$ 至 -1 。

2.2 列表的基本操作

```
>>> x = [1, 2, 3, 4, 5]
>>> x[0] #表示访问index为0的元素
1
>>> x[5] #访问index为5的元素, 因其不存在会出现错误
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    x[5] #访问index为5的元素, 因其不存在会出现错误
IndexError: list index out of range
>>> x[-1] #表示访问index为-1的元素, 即倒数第一个元素
5
>>> x[-5] #表示访问index为-5的元素, 即倒数第五个元素
1
>>> x[-6] #访问index为-6的元素, 因其不存在会出现错误
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    x[-6] #访问index为-6的元素, 因其不存在会出现错误
IndexError: list index out of range
```

正向索引

反向索引

2.2 列表的基本操作

3、in/not in运算符

- 使用in/not in运算符可以判断一个元素是否在列表中。

4、列表切片

- 列表的切片操作使用语法list[start:end]来返回列表list的一个片段。包含下标start到end-1的元素所构成的一个新列表。
- 在切片操作中，起始下标和结束下标是可以省略的。
 - 如果省略起始下标，则起始下标默认为0，即从列表的第一个元素开始截取。
 - 如果省略结束下标，则结束下标默认为列表长度，即截取到列表的最后一个元素。

2.2 列表的基本操作

```
>>> x = [1, 2, 3, 4, 5]
>>> 1 in x
True
>>> [2, 3] in x
False
>>> 6 not in x
True
```

成员关系判断

```
>>> x = [1, 2, 3, 4, 5, 6]
>>> x[1:4] # 切index为1->3的元素, 获得一个新列表
[2, 3, 4]
>>> x[1:7] # 切index为1->最大的元素, 可越过最大边界
[2, 3, 4, 5, 6]
>>> x[1:] # 切片至最后时, index最大的元素可省
[2, 3, 4, 5, 6]
>>> x[:4] # 切index为0->3的元素, 此时0可省
[1, 2, 3, 4]
```

注意切片的起始位置!

注意索引时不可越界!

2.2 列表的基本操作

```
>>> x[3]#索引, 获得一个元素
4
>>> x[3:4]#切片, 获得一个列表
[4]
>>> x[::2]#2为步长
[1, 3, 5]
>>> x[::0]#步长默认为1, 可以是任意正整数, 但不能是0
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    x[::0]#步长默认为1, 可以是任意正整数, 但不能是0
ValueError: slice step cannot be zero
>>> x[::100]
[1]
>>> x[::-1]#步长也可以是负整数, 表示列表翻转后的步长
[6, 5, 4, 3, 2, 1]
>>> x[::-3]
[6, 3]
```

注意索引和切片的差异!


负值步长实现列表的翻转。

2.2 列表的基本操作

```
>>> x
[1, 2, 3, 4, 5, 6]
>>> x[3]=100
>>> x
[1, 2, 3, 100, 5, 6]
>>> x[3:4]=5
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    x[3:4]=5
TypeError: can only assign an iterable
>>> x[3:4]=[5]
>>> x
[1, 2, 3, 5, 5, 6]
>>> x[3]=[50]
>>> x
[1, 2, 3, [50], 5, 6]
>>> x[1:4]=[20, 30, 40] #分片赋值实现对多个元素修改
>>> x
[1, 20, 30, 40, 5, 6]
```



修改单个元素：单独赋值



分片赋值：左侧切片，右侧为列表



修改一个或多个元素：分片赋值

2.2 列表的基本操作

```
>>> x
[1, 20, 30, 40, 5, 6]
>>> x[1:1]=[7,8,9]#分片赋值实现对列表扩展
>>> x
[1, 7, 8, 9, 20, 30, 40, 5, 6]
>>> x[1:4]=[]#分片赋值实现对列表删减
>>> x
[1, 20, 30, 40, 5, 6]
>>> y = x
>>> z = x[:]
>>> y is x
True
>>> z is x
False
```



扩展列表



删减列表



复制列表

列表的基本操作

思考题：已知 $x = [1, 20, 30, 40, 5, 6]$ ，则执行 $x[2]=[]$ 后， x 是什么？

```
>>> x
[1, 20, 30, 40, 5, 6]
>>> x[2]=[]
>>> x
[1, 20, [], 40, 5, 6]
```

2.2 列表的基本操作

列表的拼接和复制

- 在Python中，可使用运算符“+”来连接两个列表，并返回一个新列表。
- 使用运算符“*”可以将一个列表复制若干次后形成一个新的列表。

```
>>> x=[1, 2, 3]
>>> x*3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> x*3+[3, 5, 6]
[1, 2, 3, 1, 2, 3, 1, 2, 3, 3, 5, 6]
```


2.2 列表的基本操作

列表的比较

- 关系运算符（<、>、==、<=、>=、!=）也可以用来对列表进行比较。两个列表的比较规则如下：比较两个列表的第一个元素，如果两个元素相同，则继续比较下面两个元素；如果两个元素不同，则返回两个元素的比较结果；一直重复这个过程直到有不同的元素或比较完所有的元素为止。

列表推导式

- 列表推导式提供了一个生成列表的简洁方法。一个列表推导式由方括号括起来，方括号内包含跟着一个for子句的表达式，之后可以接0到多个for或if子句。列表推导式可以产生一个由表达式求值结果组成的列表。

2.2 列表的基本操作

```
>>> x=[1, 2, 3, 4, 5]
>>> y=[1, 2, 3, 100]
>>> x>y
False
>>> z=[1]
>>> x>z
True
>>> a=[1, 100]
>>> x>a
True
>>> x>a
False
```

列表解析是个有用的方法!

- 1: 按条件提取列表元素
- 2: 列表的生成



```
>>> x=[1, 2, 3, 4, 5, 6, 7, 8]
>>> [i%2 for i in x]
[1, 0, 1, 0, 1, 0, 1, 0]
>>> [i for i in x if i%2==0] #提取出偶数
[2, 4, 6, 8]
>>> ['男' if i%2 else '女' for i in x]
['男', '女', '男', '女', '男', '女', '男', '女']
```

2.3 列表相关的函数

列表相关的内置函数：

- `all(iterable)`
- `any(iterable)`
- `list(s)`
- `len(s)`
- `max(iterable)`
- `min(iterable)`
- `sorted(iterable[, cmp[, key[, reverse]]])`
- `sum(iterable[, start])`
- `reversed(iterable)`

2.3 列表相关的函数

```
>>> x=[1, 2, 3]
>>> any(x)
True
>>> all(x)
True
>>> y=[0, 1, 2]
>>> any(x)
True
>>> all(y)
False
>>> list('123')
['1', '2', '3']
>>> list((1, 2, 3))
[1, 2, 3]
```

```
>>> len(x)
3
>>> min(x)
1
>>> max(x)
3
>>> sorted(x)
[1, 2, 3]
>>> sum(x)
6
```


2.3 列表相关的函数

```
>>> x=[-100, 30, 2, -30, 0]
>>> sorted(x, key=abs, reverse=False)
[0, 2, 30, -30, -100]
>>> reversed(x)
<list_reverseiterator object at 0x0310A750>
>>> list(reversed(x))
[0, -30, 2, 30, -100]
```

注意：sorted()函数中key指排序方式，通常是一个函数，而reverse关键字指是否采用反序排列。

函数reversed()的参数是一个列表，实现该列表的翻转。

2.4 列表相关的方法

列表类的成员函数：

- **list.append(x)**
- **list.extend(L)**
- **list.insert(i, x)**
- **list.remove(x)**
- **list.pop(index)**
- **list.index(x)**
- **list.count(x)**
- **list.sort(key=None, reverse=False)**
- **list.reverse()**

2.4 列表相关的方法

```
>>> x=[1, 2]
>>> x.append(3)
>>> x
[1, 2, 3]
>>> x.extend([3])
>>> x
[1, 2, 3, 3]
```

注意： `extend` 的参数是列表。实现 `x` 与参数列表的拼接！

而 `append` 是把任意对象参数作为最后一个元素放到列表中！

```
>>> x=[1, 2, 3, 4, 5, 6, 3]
>>> x.remove(3)
>>> x
[1, 2, 4, 5, 6, 3]
>>> x.pop(3)
5
>>> x
[1, 2, 4, 6, 3]
>>> x.insert(2, 100)
>>> x
[1, 2, 100, 4, 6, 3]
```

注意： `remove` 将第一个值为 3 的元素删除！

而 `pop` 是将索引为 3 的元素删除，同时返回该元素的值！

`insert` 为在指定位置插入指定元素

2.4 列表相关的方法

sort、reverse、count和index方法

```
>>> x = [1, 2, 3, 100, 4, 100, 5, 78]
>>> x.index(100) #返回100的索引
3
>>> x.count(100) #返回值为100的元素数
2
>>> x.sort(key = lambda x:x%5, reverse = False)#依照除以5的余数大小正排序
>>> x
[100, 100, 5, 1, 2, 3, 78, 4]
>>> x.reverse() #对x就地翻转
>>> x
[4, 78, 3, 2, 1, 5, 100, 100]
```

注意：列表方法与列表函数的使用上的区别！

2.5 多维列表

多维列表：列表的元素可以是任何类型的对象，包括列表。

二维列表可以理解为一个由行组成的列表，每一行都可以使用下标访问，称为行下标，每一行中的值可以通过另一个下标来访问，称为列下标。

二维列表中的每个值都可以使用`myMat[i][j]`来访问，其中`i`和`j`分别代表行下标和列下标。

要遍历一个二维列表，一般需要使用两层嵌套的循环结构来实现。

		[0]	[1]	[2]	[3]	[4]
myMat = [[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15]]	[0]	1	2	3	4	5
	[1]	6	7	8	9	10
	[2]	11	12	13	14	15

2.5 多维列表

```
>>> x=[[1, 2, 3], [4, 5, 6]]
>>> x[1][1] #元素的元素
5
>>> x[1]
[4, 5, 6]
>>> len(x)
2
>>> len(x[1])
3
>>> x[1].append(7)
>>> x
[[1, 2, 3], [4, 5, 6, 7]]
>>> x.append(7)
>>> x
[[1, 2, 3], [4, 5, 6, 7], 7]
```

1: 多维列表本质上仍然是列表，只是元素也是列表罢了！

2: 列表的方法和函数对于多维列表仍然适用。

3: 注意多维列表中的成员数量统计口径。

2.6 元组

元组是不可变的，即元组一旦创建，其中的元素就不可以被修改。

元组由用逗号分隔的若干值组成。如果在使用中不会对列表的内容进行修改，那么可以使用元组来代替列表。

创建一个元组最简单的方法就是用一对圆括号括起来组成一个元组，元组内的元素使用逗号分隔。

元组通过Python内置的tuple类进行定义，因此也可以使用tuple函数创建一个列表。使用tuple函数也可以将列表、字符串等元素转换为元组。

元组也是序列，因此一些用于列表的基本操作也可以用在元组上。可以使用下标访问元组中的元素，使用in和not in运算符来判断元素是否在元组中，对元组进行切片，等等。

2.7 元组封装与序列拆封

元组封装：指的是将多个值自动封装到一个元组中。

- `t = 1, 1, 2, 3, 5`

序列拆封：元组封装的逆操作，用来将一个封装起来的序列自动拆分为若干个基本数据。

- `tuple1 = (1, 2, 3)`
- `x, y, z = tuple1`

同时赋值的语法是将元组封装和序列拆封操作相结合了。


2.8 元组与列表的比较

元组和列表都属于序列。元组属于不可变序列，元组中的元素一旦定义就不允许进行增加、删除和替换操作。因此，tuple类没有提供append()、insert()和remove()等函数。在使用下标访问或者切片操作时，也只允许读取元组中的值而不能对其进行修改或赋值。

元组的访问和处理速度比列表更快。因此，如果所需要定义的序列内容不会进行修改，那么最好使用元组而不是列表。另外，使用元组也可以使元素在实现上无法被修改，从而使代码更加安全。

2.9 关于元组

```
>>> y=tuple(x)
>>> y
([1, 2, 3], [4, 5, 6, 7], 7)
>>> y[2]=3 #元组的元素不能修改
Traceback (most recent call last):
  File "<pyshell#12>", line 1, in <module>
    y[2]=3 #元组的元素不能修改
TypeError: 'tuple' object does not support item assignment
>>> y[1].append(8) #元组的元素为可变对象时
>>> y
([1, 2, 3], [4, 5, 6, 7, 8], 7)
>>> len(y)
3
>>> 7 in y
True
>>> 7 not in y
False
>>> y[::-1]
(7, [4, 5, 6, 7, 8], [1, 2, 3])
```



元组的元素
为可变对象
时,这个元
素可以修改