学习Python Doc第四天:数据结构

张朝龙

目录

Python 提供了丰富的数据结构，极大提高编码效率。今天，我们讨论与数据结构有关的知识点。

1 深入了解 list

首先 list 数据类型提供了很多方法，我们把这些方法列举如下：

例 1.1

1. *list.append(x)* 添加一个元素到列表的尾部，等效于 a[len(a) :]=[x] 注意等式右边的写法是 *[x]* 而不是 *x*
2. *list.extend(iterable)* 用 *iterable* 中的元素扩展 *list* ，等效于 a[len(a) :] = iterable
3. *list.insert(i,x)* 在指定位置添加新元素。第一个输入 *i* 是插入新元素的位置，比如 *a.insert(0,x)* 是在 *list* 的第0个位置插入 *x* , *a.insert(len(a),x)* 等效于 *a.append(x)*
4. *list.remove(x)* 从 *list* 中移除第一个 *x* ,如果 *list* 中没有 *x* 则报错
5. *list.pop([i])* 从 *list* 的指定位置删除元素，并返回这个元素。如果没有给定具体位置，则弹出最后一个元素。注意 *[]* 表示这个函数参数是可选的。
6. *list.index(x,[start],[end])* 返回第一个 *x* 的位置。 *start* 和 *end* 表示查找 *x* 的范围。
7. *list.count(x)* 返回 *x* 在 *list* 中出现的次数
8. *list.sort(key=None,reverse=False)* 对 *list* 中的元素进行排序
9. *list.reverse()* 对 =*list*= 中的元素，逆序排序
10. *list.copy* 返回一个 *list* 的副本

1.1 把 list 当栈用

list 结构以及其附带的方法，使得可以方便的把 list 当做栈（后进先出）用。看代码：

```
In [179]: stack = [3,4,5]

In [194]: stack.append(6)

In [198]: stack.append(7)

In [202]: stack
Out[210]:
[3, 4, 5, 6, 7]

In [211]: stack.pop()
Out[215]:
7

In [216]: stack
Out[220]:
[3, 4, 5, 6]
```

1.2 把 list 当队列用

同样，也可以使用 list 实现队列。对立的特点是先进先出，然而 list 对于这个操作并不是很高效，因为从 list 的尾部插入元素和弹出元素比较快，但是从 list 的头部插入或者弹出元素比较慢(因为所有的其他元素都要移位一次)。

为了实现队列，我们使用 collections.deque 。 collections.deque 的设计使得从队列的头部和尾部插入或者弹出元素都很方便。

看代码：

```
In [221]: from collections import deque

In [234]: queue = deque(['Eric','John','Michael'])

In [265]: queue.append('Terry')

In [282]: queue.append('Grahm')

In [298]: queue
Out[298]:
deque(['Eric', 'John', 'Michael', 'Terry', 'Grahm'])

In [299]: queue.popleft()
Out[306]:
'Eric'

In [307]: queue.popleft()
Out[313]:
'John'
```

1.3 队列推导式

队列推导式提供了一种简单的创建列表的方法。当我们需要把一些运算的结果作为队列元素时，队列推导式显得非常的方便。

```
squares = []
for x in range(10):
    squares.append(x**2)
```

输出：

```
In [333]: squares
Out[337]:
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

当然，我们可以使用更方便的方法创建上面的这个 list

```
squares = list(map(lambda x:x**2,range(10)))
```

或者

```
squares = [x**2 for x in range(10)]
```

列表推导式由包含一个表达式的括号组成，表达式后面跟随一个 for 子句，之后可以有零或多个 for 或 if 子句。结果是一个列表，由表达式依据其后面的 for 和 if 子句上下文计算而来的结果构成。

比如

```
In [339]: [(x,y) for x in [1,2,3] for y in [3,1,4] if x!=y]
Out[397]:
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

上面的一行代码等效于：

```
1 comp = []
2 for x in [1,2,3]:
3     for y in [3,1,4]:
4         if x != y:
5             comp.append((x,y))
```

我们再给几个例子：

```
In [399]: vec = [-4,-2,0,2,4]

In [409]: [x*2 for x in vec]
Out[416]:
[-8, -4, 0, 4, 8]

In [417]: [x for x in vec if x>=0]
Out[444]:
[0, 2, 4]

In [445]: [abs(x) for x in vec]
Out[456]:
[4, 2, 0, 2, 4]

In [457]: freshfruit = [' banana',' loganberry','passion fruit ']

In [506]: [weapon.strip() for weapon in freshfruit]
Out[536]:
['banana', 'loganberry', 'passion fruit']

In [537]: [(x,x**2) for x in range(6)]
Out[562]:
[(0, 0), (1, 1), (2, 4), (3, 9), (4, 16), (5, 25)]

In [563]: [x,x**2 for x in range(6)]
  File "<ipython-input-563-8d6940458683>", line 1
    [x,x**2 for x in range(6)]
             ^
SyntaxError: invalid syntax


In [564]: vec = [[1,2,3],[4,5,6],[7,8,9]]
```

```
In [581]: [num for elem in vec for num in elem]
Out[596]:
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

从上面的例子，我们可以看到，如果生成的 list 中元素都是二元组的话，则必须用括号包起来。

list 生成器可以包括复杂的表达式或者嵌套函数

```
In [602]: [str(round(pi,i)) for i in range(1,6)]
Out[616]:
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

1.4 嵌套的 list 推导式

list 推导式的第一个表达式可以是任何表达式，包括 list 推导式。

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

输出为：

```
In [619]: [[row[i] for row in matrix] for i in range(4)]
Out[658]:
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

上面的代码实现了矩阵转置功能（交换了矩阵的行和列）。对于 matrix 这样的数据结构，

```
[x[0] for x in matrix]
```

输出的是 matrix 的第 0 列 [1,5,9] 。整个矩阵转换代码等效于：

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

鉴于 Python 强大的函数库，这个转置功能可以通过 zip 来实现。

```
list(zip(*matrix))
```

输出是：

```
[(1, 5, 9), (2, 6, 10), (3, 7, 11), (4, 8, 12)]
```

2 del 语句

list.remove(x) 删除 list 中第一个值为 x 的元素。在移除的过程中必须给定 x 。使用 del 语句可以不用给定 x 只用给定索引号就删除指定位置的元素。

```
In [736]: a = [1,2,3,4,5,6,7]
```

```
In [758]: del a[0]

In [770]: a
Out[770]:
[2, 3, 4, 5, 6, 7]

In [771]: del a[5]

In [787]: a
Out[787]:
[2, 3, 4, 5, 6]

In [788]: del a[2]

In [792]: a
Out[792]:
[2, 3, 5, 6]
```

del 也可以用来删除一个变量

```
del a
```

3 tuple 和 sequence

list 和 strings 是两个 sequence 类型的数据类型。由于 Python 是一个不断演进的语言， tuple 是新加入的 sequence 成员。一个 tuple 的成员用逗号隔开，看例子：

```
In [793]: t = 12345, 54321, 'hello!'

In [821]: t[0]
Out[828]:
12345

In [829]: t[2]
Out[832]:
'hello!'

In [833]: t
Out[837]:
(12345, 54321, 'hello!')

In [838]: u = t,(1,3,4)

In [854]: u
Out[854]:
((12345, 54321, 'hello!'), (1, 3, 4))

In [855]: t[0]
```

```
Out[863]:
12345

In [864]: t[0] = 8888
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-871-7f2f230bad03> in <module>()
----> 1 t[0] = 8888

TypeError: 'tuple' object does not support item assignment

In [872]: v = ([1,2,3],[4,5,6])

In [887]: v
Out[887]:
([1, 2, 3], [4, 5, 6])
```

从上面例子可以看出， tuple 的输出总是有括号包括。不可以给 tuple 中单个元素赋值。

尽管 tuple 和 list 有很多相似之处，但是他们经常在不同的场合使用。 tuple 是不可修改的。通常 tuple 包含不同种类的成员。 list 的成员则通常是相同类型的并可以通过迭代读写。

在创建另个或者一个元素的 tuple 时，有一些简单的技巧。

```
In [888]: empty = ()

In [903]: singleton = 'hello', # note the trailing comma

In [920]: len(empty)
Out[920]:
0

In [921]: len(singleton)
Out[933]:
1

In [934]: singleton
Out[942]:
('hello',)
```

4 set

Python 还有一个数据类型 set . 一个 set 是一组无重复元素的集合。 set 支持数学概念上的 *并 交 差 对称差* . 通常用花括号和 set() 来创建 set

```
In [943]: basket = {'apple','orange','apple','pear','orange','banana'}

In [989]: print(basket)
{'banana', 'orange', 'pear', 'apple'}
```

可以看出 set 自动删除集合中的重复元素。

```
In [997]: 'orange' in basket #fast membership testing
Out[1027]:
True

In [1028]: 'crabgrass' in basket
Out[1052]:
False
```

可以快速的进行成员验证。

```
In [1053]: a = set('abracadabra')

In [1066]: b = set('alacazam')

In [1078]: a
Out[1078]:
{'a', 'b', 'c', 'd', 'r'}

In [1079]: b
Out[1083]:
{'a', 'c', 'l', 'm', 'z'}

In [1084]: a-b # letters in a but not in b
Out[1091]:
{'b', 'd', 'r'}

In [1092]: a | b #letters in either a or b
Out[1099]:
{'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}

In [1100]: a + b #do not support +
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-1100-f96fb8f649b6> in <module>()
----> 1 a + b

TypeError: unsupported operand type(s) for +: 'set' and 'set'

In [1101]: a & b #letter in both a and b
Out[1108]:
{'a', 'c'}

In [1109]: a ^ b #letters in a or b nut not both
Out[1120]:
{'b', 'd', 'l', 'm', 'r', 'z'}
```

Python 除了支持列表推导式外，也支持集合推导式。

```
In [1121]: a = {x for x in 'abracadabra' if x not in 'abc'}
```

```
In [1146]: a
Out[1146]:
{'d', 'r'}
```

5 字典（dictionaries）

dictionary 是 Python 支持的又一数据类型，这个数据类型不是序列类型，而是映射类型(mapping types)。序列类型（ list tuple set ）通过数字来索引元素，映射类型的数据通过关键字 (key)来索引元素。字符创和数字可以当做 key 来使用。如果 tuple 的成员都是字符创，数字或者 tuple ，那么 tuple 也可以用来当做 key . 不能用 list 来当做 key ，因为 list 能够被修改。

通常，我们可以想象字典是一系列没有排序的 key:value 对，其中 key 在一个字典中是唯一的。 {} 创建空的字典。在字典中我们经常用的操作是按照某个 key 保存一个 value 或者，根据某个 key 读取 value 。同样，我们可以使用 del 来删除 key:value 。如果新存入的 key 和原来重复，那么原来的 key 对应的 value 就会被覆盖。如果试图从字典中读取某个不存在的 key 对应的 value 那么报错。

对一个字典执行 list(d.keys()) 操作，返回这个字典使用的所有 key ,返回的 list 是无序的，如果你需要返回结果有序，那么使用 sorted(d.keys()) . 使用 in 进行成员关系测试。看代码：

```
In [1147]: tel = {'jack':4098, 'sape':4139}

In [1186]: tel['guido'] = 4127

In [1198]: tel
Out[1198]:
{'guido': 4127, 'jack': 4098, 'sape': 4139}

In [1199]: tel['jack']
Out[1211]:
4098

In [1212]: del tel['sape']

In [1224]: tel['irv'] = 4127

In [1251]: tel
Out[1255]:
{'guido': 4127, 'irv': 4127, 'jack': 4098}

In [1256]: list(tel.keys())
Out[1276]:
['jack', 'irv', 'guido']

In [1277]: sorted(tel.keys())
Out[1284]:
['guido', 'irv', 'jack']

In [1285]: 'guido' in tel
```

```
Out[1293]:
True


In [1294]: 'jack' not in tel
Out[1300]:
False
```

可以使用 dict() 函数创建字典，看代码：

```
In [1301]: dict([('sape',4139),('guido',4127),('jack',4098)])
Out[1354]:
{'guido': 4127, 'jack': 4098, 'sape': 4139}
```

当然，字典也支持字典推导式，看代码：

```
In [1355]: {x:x**2 for x in (2,4,6)}
Out[1377]:
{2: 4, 4: 16, 6: 36}
```

当 key 是简单的字符创时，使用 dict() 来构建字典更容易，看代码：

```
In [1378]: dict(sape=4139,guido=4127,jack=4098)
Out[1433]:
{'guido': 4127, 'jack': 4098, 'sape': 4139}
```

6 使用循环

当在字典中使用循环的时候，可以使用 item() 来获取 key:value 。

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

在我的环境中输出是：

```
In [1435]: robin the brave
gallahad the pure
```

不知道为什么先输出了第二个 key:value ，难道输出是乱序的么？

当循环对象是序列类型的数据类型时，可以使用 enumerate() 来生成索引和该索引对应的值：

```
for i,v in enumerate(['tic','tac','toe']):
    print(i,v)
```

输出为：

```
In [1436]: 0 tic
1 tac
2 toe
```

在两个或者多个序列（ sequence ）中执行循环，可以使用 zip() 实现。

```
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))
```

输出为：

```
In [1437]: what is your name It is lancelot.
what is your quest It is the holy grail.
what is your favorite color It is blue.
```

其中 list(zip(questions,answers)) 的结果是：

```
In [1438]: list(zip(questions,answers))
Out[1456]:
[('name', 'lancelot'), ('quest', 'the holy grail'), ('favorite color', 'blue')]
```

如果要对一个 sequence 类型的数据进行逆向循环时，使用 reversed() 函数。

```
for i in reversed(range(1,10,2)):
    print(i)
```

输出是：

```
In [1457]: 9
7
5
3
1
```