Python基本数据结构

列表

列表是Python6个序列的内置类型的一种。序列中的每个元素都分配一个数字——它的位置，或索引。第一个索引是0，第二个是1，依次类推。序列都可以进行的操作包括索引、切片、加、乘、成员检查。此外，Python已经内置确定序列的长度以及确定最大和最小的元素的方法。

列表是最常用的Python数据类型，其非常适合利用顺序和位置定位某一元素，而且其元素不需要类型相同。列表是可变的，可以直接对原始列表进行修改：添加、删除、覆盖元素。列表中具有相同值的元素允许出现多次。

创建列表

只要把逗号分隔的不同的数据项使用方括号（`[]`）括起来即可。或者使用内置的`list()`函数。

```
empty_list = []
companies = ['Google', 'Apple', 'Facebook', 'Microsoft']

alist = list()
blist = list(range(5))
print(blist)
```

与字符串的索引一样，列表索引从0开始。列表可以进行截取、组合等。

将其他数据类型转换成列表

```
list('abc')  # ['a', 'b', 'c']
a_tuple = ('a', 'b', 'c')
list(a_tuple)
```

索引切片

与字符串索引切片类似，参考之前内容。

列表嵌套

列表可以包括各种类型的元素，包括列表。

```
lower_alphas = list('abcdefg')
upper_alphas = list('abcdefg'.upper())
alphas = [lower_alphas, upper_alphas]
alphas[1][:3]
```

添加元素

使用 `append()` 函数在列表的末尾添加元素：

```
alist = [1, 2, 3]
alist.append(4)
alist
```

使用`insert()`函数在指定位置插入元素，其接受两个参数，第一个参数指定要插入的位置（偏移量），第二个参数指定要插入的元素。如果指定的偏移量超过了列表的尾部，则插入到列表最后。

```
alist.insert(0, 0)
alist.insert(10, 10)
alist
```

修改元素

通过索引（偏移量）访问某列表元素，并可以通过赋值操作对其进行修改：

```
alist[0] = 'modified'
alist[1:4] = 'a', 'b', 'c'
alist
```

删除元素

使用`del`删除指定位置的元素，当列表中的元素被删除后，位于其后面的元素自动迁移填补空位，且列表长度减去删除的元素个数。

```
alist
del alist[0:4]
alist
```

删除具有指定值的元素

使用`remove()`删除具有指定值的元素：

```
blist = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
blist.remove('b')
blist
```

获取并删除指定位置的元素

使用`pop()`可以获取列表中指定位置的元素，并将此元素从列表中删除。如果未指定参数，则默认返回列表末尾的元素，使用`pop(0)`将返回列表头部的元素。当列表为空或者索引超出范围时，将触发`IndexError`异常。

```
blist.pop()
blist.pop(10)
```

清空列表元素

使用`clear()`函数清空列表中的所有元素：

```
alist = [1, 2, 3]
alist.clear()
alist
```

合并列表

使用`extend()`函数或`+=`操作合并列表：

```
clist = ['a', 'b', 'c']
dlist = ['d']
elist = ['e', 'f']
clist.extend(dlist)
clist = clist + elist
clist
```

查询具有特定值的元素位置

使用`index()`函数查询具有特定值的元素在列表中的位置（偏移量）：

```
clist.index('e')
```

成员关系判断

使用`in`判断元素是否包含于某个列表中。

```
'e' in clist   # True
'h' not in clist  # True
```

记录特定值出现的次数

使用`count()`记录特定值在列表当中出现的次数。

```
clist.count('a')
```

列表元素排序

- 使用列表对象方法`sort()`对原列表进行排序，改变原列表内容。
- 使用通用函数`sorted()`返回排好序的列表副本，原列表内容不变。

如果列表中的元素都是数字，默认排序方式是从小到大的生序。如果元素都是字符串，则按照字母表顺序排列。默认的排序都是升序的，通过添加参数`reverse=True`可以改为降序排列。

```
numbers = [2, 1, 4.0, 3]
numbers.sort(reverse=True)
sorted_numbers = sorted(numbers)
print(numbers)
print(sorted_numbers)
```

复制列表

通过下面3种方法，将一个列表复制到另一个新列表中：

- 列表`copy()`函数
- `list()`转换函数
- 列表切片`[:]`

```
a = [1, 2, 3]
b = a.copy()
c = list(a)
d = a[:]
print('a: ', a, '\nb: ', b, '\nc: ', c, '\nd: ', d)
```

关于列表对象方法的介绍，可以使用`help(list)`来查看。

代码练习

实现一个函数，对于给定的一个字符串，比如`'aaabeeeezzzxxxxxw'`，将其中连续出现的字符转换成该字符和它重复出现的次数的组合并将结果返回，比如对于上面字符串，返回结果为`'a3b1e4z3x5w1'`。假定要编码的字符串只包含26个字母，没有数字，同时编写解码算法代码，假定要解码的字符串是合法编码过的。

元组

元组与列表有很多相似之处，但它是一种不可变的容器。元组一旦被定义以后，将无法再进行添加、删除或修改元素等操作。它能用来做一些列表不能做的事，如用作一个字典的键。鉴于其与列表很大程度上是相似的，所以这里着重描述元组和列表的不同之处。

创建元组

列表用`[]`者工厂函数`list()`，元组用`()`或工厂函数`tuple()`。

创建包含一个或多个元素的元组时，每一个元素后面都需要跟着一个逗号，即使只包含一个元素也不能省略。如果创建的元组所包含的元素数量超过1，最后一个元素后面的逗号可以省略。

```
empty_tuple = ()  # 使用 () 创建一个空元组
empty_tuple
```

```
type(empty_tuple)
```

```
atuple = 'mon', 'tue', 'wed'
atuple
```

Python的交互式解释器输出元组时会自动添加一对圆括号。定义元组真正靠的是每个元素的后缀逗号。建议使用圆括号将所有元素包裹起来，这样会使程序更加清晰。

```
btuple = ('mon', 'tue', 'wed')
btuple
```

元组解包

将元组赋值给多个变量，这个过程可被称作*元组解包*。

```
a, b, c = atuple
print(a, b, c)
```

注意，这里需要变量的数量必须跟元组元素的数量是一样的，如果个数不匹配，会产生`ValueError`异常。

```
a, b = atuple
```

实际上，这种解压赋值可以用在任何可迭代对象上面，包括元组、列表、字符串、文件对象、迭代器和生成器。

使用`tuple()`构造器可将其他可迭代类型对象转换成元组：

```
tuple()   # 空元组
tuple(range(5))   # (0, 1, 2, 3, 4)
tuple('abcdefg')  # ('a', 'b', 'c', 'd', 'e', 'f', 'g')
```

元组更新（不可以的！）

元组类型不能更新。

```
atuple = (1, 2, 4)
atuple[0] = 9
```

特殊情况：如果元组的某个成员是列表，那么这个成员的成员是可以变更的：

```
a = (1, 2, 3, 4, ['a', 'b'])
a[4][1] = 'modified'
a
```

注意：这里并没有违反元组不可更新的原则，我们更改的是一个列表对象，可以这样认为：这个元组里面的第4号元素是到列表对象的引用，我们并没有更新这个引用！

字典

字典（dictionary）是Python中唯一的“映射”类型，映射这个概念在高中就学过：一个函数f将键（key, 定义域）映射到值（value, 值域）。这样的函数在字典中可以称为哈希（HASH）函数。通过哈希函数可以对键通过计算快速得到值的位置，而避免了线性搜索，极大的提高了数据值的存取效率；此外，字典是容器类型，可更新模型。基于这些特性，字典通常被认为是Python中最强大的数据类型之一。 Python3.6版本之前，Python字典中的键是无序的，Python3.6重新实现了字典，可以记住元素的添加顺序。 字典中的键通常是字符串，但还可以是Python中其他任意的不可变对象：布尔型、整型、浮点型、元组、字符串。

创建和赋值

使用`{}`创建空字典。使用`dict()`构造器可从其他对象创建字典。

`dict()`构造器的具体用法：

- `dict()` -> new empty dictionary
- `dict(mapping)` -> new dictionary initialized from a mapping object's (key, value) pairs

  可以对任何包含**双值子序列**的序列使用`dict()`。
- `dict(iterable)` -> new dictionary initialized as if via:

  ```
  d = {}
  for k, v in iterable:
      d[k] = v
  ```
- `dict(**kwargs)` -> new dictionary initialized with the name=value pairs in the keyword argument list.

```
empty_dict = {}  # here create an empty dict
dict2 = {'name': 'earth', 'port': 80}  # a dict with two key-value pairs
dict3 = dict((['x', 1], ['y', 2]))  # factory function
dict4 = dict(['ab', 'cd', 'ef'])
dict5 = dict(one=1, two=2)
dict6 = dict(zip(['one', 'two'], [1, 2]))
dict7 = {}.fromkeys(('x', 'y'), -1)  # builtin fun fromkeys(), all valued -1
dict8 = {}.fromkeys(('x', 'y'))  # all valued default 'None'
print(empty_dict)
print(dict2)
print(dict3)
print(dict4)
print(dict5)
print(dict6)
print(dict7)
print(dict8)
```

`fromkeys()`可以使用一个可迭代的序列作为键集合创建一个默认字典，第二个参数是默认值，如果忽略的话值默认为`None`。

获取元素

使用键来获取字典中的元素。

```
dict1 = {'name': 'earth', 'port': 80}
dict1['name']
```

如果试图访问的键在字典中不存在，则会引发一个`KeyError`异常：

```
dict1['age']
```

所以，访问元素之前最好判断一下这个键是否存在：

```
if 'age' in dict1:
    print(dict1['age'])
else:
    print(dict1)
```

修改元素

```
dict1['name'] = 'moon'
dict1['port'] = 777
dict1['age'] = 20       # 添加新元素
dict2 = {'name': 'mars', 'port': 888}
dict1.update(dict2)  # update a dict with another
dict1
```

如果键已存在，则用新值更新旧值，否则，加入新键值对。

删除元素

```
del dict1['name']   # 删除"键"为"name"的条目
dict1.pop('age')    # 删除并返回"键"为"name"的条目
dict1.clear()       # 删除dict1中的所有条目
del dict1           # 删除dict1这个字典
dict1     # NameError: name 'dict1' is not defined
```

内建方法

- `dict.clear()` # 删除字典中的所有元素,并返回一个浅拷贝的副本
- `dict.fromkeys(seq, val=None)` # 以序列seq中的元素作为键，创建并返回一个新字典，val为键值对的默认value
- `dict.get(key, default=None)` # 对字典中的key，返回value，若key不存在，返回default
- `dict.items()` # 返回一个包含字典中键值对元组的列表
- `dict.keys()` # 返回字典中键的列表
- `dict.pop(key, [default])` # 和方法get相似，区别在于不仅返回，还要删除
- `dict.setdefault(key, default=None)`
- `dict.update(dict2)`
- `dict.values()` # 返回包含所有value的列表

对于不同的数据类型，讲到越往后就越粗糙了，因为很多内容有太多的相似性，你到现在应该可以熟练使用帮助文档了，所以不太清楚的时候多查查文档：

`dir(module_name)`, `dir(type)`, `help(module_name || type)`, `module.__doc__` ...

字典的键

字典中的值可以是任何Python对象，甚至是字典以及用户自定义类型，但对于键却是有一些限制的。

不允许一个键对应多个值

```
dict1 = {'name': 'green', 'age': 20, 'name': 'marry'}
```

上面的例子中出现了两个键`name`，对于这种冲突，Python直接取最后的赋值，所以，最后得到的是：

```
dict1
```

键必须是可哈希的

可哈希的对象才可以作为键，列表、字典这样的可变类型是不可哈希的，所以不能作为键。所有不可变类型都是可哈希的，都可以作为字典的键。要说明的是，对于数字类型来说，值相等的数字表示相同的键，**1**和**1.0**的哈希值是相同的，它们是相同的键。

对于可变对象，如果事先调用`__hash__()`方法并返回一个整形，那它的hash值也是不变的，因此这类对象可以作为键，这是一种特例。

数字和字符串毫无疑问可以作为键，那么元组（也是不可变类型）呢?

注意：元组并不一定是一成不变的，如果其中某个成员是列表、字典等可变类型，我们仍然可以"改变元组"。

```
tuple_demo = (12, ['a', 'b'])
tuple_demo[1][0] = 'name'
tuple_demo
```

元组中只包括数字、字符串这样的不可变参数，才可以作为字典中的有效键。

使用`hash(obj)`可以返回对象的hash值，如果返回异常，说明这个对象不能够被hash，你应该明白：它一定不能用作字典的键。

```
atuple = 1, [1, 2],
# atuple.__hash__()
hash(atuple)
```

集合

这里的“集合”概念和数学中是相同的，你可以对其进行交、并、差、补等一些列操作。

Python中的集合有两种类型，可变集合（set）和不可变集合（frozenset），对于可变集合，可以添加和删减元素，但不可哈希，因此不能用作字典的键，也不能作为其它集合的元素；而不可变集合可以哈希，可以被用作键或者集合成员。

创建集合

使用大括号将一系列以逗号隔开的值包裹起来就创建了一个集合，另外可以使用内建的`set()`和`frozenset()`构造器创建集合。

```
even_numbers = {0, 2, 4, 6, 8}
# 以下用三种方式创建了三个相同的可变集合
set1 = set('abcde')
set2 = set(('a', 'b', 'c', 'd', 'e'))
set3 = set(['a', 'b', 'c', 'd', 'e'])
# 以下用三种方式创建了三个相同的不可变集合
set4 = frozenset('abcde')
set5 = frozenset(('a', 'b', 'c', 'd', 'e'))
set6 = frozenset(['a', 'b', 'c', 'd', 'e'])
```

当字典作为参数传入`set()`构造器时，只有键会被使用：

```
set({'apple': 'red', 'orange': 'orange', 'cherry': 'red'})
```

访问集合成员

```
myset = set(['apple', 'pear', 'grape', 'banana'])
print('apple' in myset)
print('orange' in myset)
for fruit in myset:
    print(fruit)
```

更新集合

只有可变集合（set）支持更新。

```
myset.add('orange')    # 添加元素
myset
myset.remove('pear')   # 删除元素
newfruit = set(('coco', 'watermelon'))
myset.update(newfruit)   # 更新集合
myset

del newfruit  # 删除集合
```

集合类型操作符和内建方法

标准类型操作符

1. 成员关系判定：`in`、`not in`
2. 集合等价/不等价：`==`、`!`、`=`
3. 子集/超集判定：`<`、`<=`、`>`、`>=`

    ```
    set('book') <= set('bookshop')  # True
    set('supermarket') >= set('market')  # True
    ```

集合类型操作符（用于可变集合和不可变集合）

```
s = set('abcde')
t = set('defgh')
print(s | t)  # 联合操作，或称OR操作，也就是算并集
print(s & t)  # 求交集操作，或称AND操纵
print(s - t)  # 求差集/相对补集
print(s ^ t)  # 对称差分，类似C中异或操作XOR
```

集合类型操作符（用于可变集合）

```
s = set((1, 3, 5, 7))
t = set((5, 7, 9, 0))
s |= t
print(s)
s &= t
print(s)
s -= t
print(s)
s ^= t
print(s)
```

这里很好理解，因为是可变集合，所以可以将上一部分讲的集合类型操作符以算数自反赋值的方式应用到集合上，但是对`frozenset`来说是不行的。

集合类型的内建方法

```
# 用于所有集合的方法
s.issubset(t)             # 如果s是t的子集，返回True
s.issuperset(t)           # 如果s是t的超级，返回True
s.union(t)                # 返回一个新集合（s和t的并集）
s.intersection(t)         # 返回一个新集合（s和t的交集）
s.difference(t)           # 返回一个新集合（s - t）
s.symmetric_difference(t)
                          # 返回一个新集合（s ^ t）
s.copy()                  # 返回一个新集合，它是s的浅复制

# 仅用于可变集合的方法
s.update(t)               # 用t中的元素更新s
s.intersection_update(t) # 将t中的元素并入到s中
s.difference_update(t)    # s中现在是s和t的交集
s.symmetric_difference_update(t)
                          # s中现在是（s - t）
s.add(obj)                # 在s中添加对象obj
s.remove(obj)             # 从s中删除对象obj，如果不存在则引发KeyError异常
s.discard(obj)            # 如果obj是s的元素，就从s中删除
s.pop()                   # 删除s中任意一个对象，并返回
s.clear()                 # 删除集合s中的所有元素
```

至此，我们已经学完了Python所有的基本内建类型：数字，字符串，列表，元组，字典，集合。这些数据类型功能强大，使用简单，能够熟练使用它们是后续学习的关键。