数据结构

1. 列表

`list`是处理一组有序项目的数据结构，即你可以在一个列表中存储一个 序列 的项目。假想你有一个购物列表，上面记载着你要买的东西，你就容易理解列表了。只不过在你的购物表上，可能每样东西都独自占有一行，而在 Python 中，你在每个项目之间用逗号分割。
列表中的项目应该包括在方括号中，这样 Python 就知道你是在指明一个列表。一旦你创建了一个列表，你可以添加、删除或是搜索列表中的项目。由于你可以增加或删除项目，我们说列表是 可变的 数据类型，即这种类型是可以被改变的。

尽管我一直推迟讨论对象和类，但是现在对它们做一点解释可以使你更好的理解列表。我们会在相应的章节详细探索这个主题。

列表是使用对象和类的一个例子。当你使用变量`i`并给它赋值的时候，比如赋整数`5`，你可以认为你创建了一个**类**（类型）`int`的**对象**（实例）`i`。事实上，你可以看一下`help(int)`以更好地理解这一点。
类也有**方法**，即仅仅为类而定义地函数。仅仅在你有一个该类的对象的时候，你才可以使用这些功能。例如，Python 为`list`类提供了`append`方法，这个方法让你在列表尾添加一个项目。例如`mylist.append('an item')`列表`mylist`中增加那个字符串。注意，使用点号来使用对象的方法。
一个类也有**域**，它是仅仅为类而定义的变量。仅仅在你有一个该类的对象的时候，你才可以使用这些变量/名称。类也通过点号使用，例如`mylist.field`。

```
#!/usr/bin/python
# Filename: using_list.py

# This is my shopping list
shoplist = ['apple', 'mango', 'carrot', 'banana']

print 'I have', len(shoplist),'items to purchase.'

print 'These items are:', # Notice the comma at end of the line
for item in shoplist:
    print item,

print '\nI also have to buy rice.'
shoplist.append('rice')
print 'My shopping list is now', shoplist

print 'I will sort my list now'
```

```
shoplist.sort()
print 'Sorted shopping list is', shoplist

print 'The first item I will buy is', shoplist[0]
olditem = shoplist[0]
del shoplist[0]
print 'I bought the', olditem
print 'My shopping list is now', shoplist
```

输出

```
$ python using_list.py
I have 4 items to purchase.
These items are: apple mango carrot banana
I also have to buy rice.
My shopping list is now ['apple', 'mango', 'carrot', 'banana', 'rice']
I will sort my list now
Sorted shopping list is ['apple', 'banana', 'carrot', 'mango', 'rice']
The first item I will buy is apple
I bought the apple
My shopping list is now ['banana', 'carrot', 'mango', 'rice']
```

它如何工作

变量 `shoplist` 是某人的购物列表。在 `shoplist` 中，我们只存储购买的东西的名字字符串，但是记住，你可以在列表中添加 任何种类的对象 包括数甚至其他列表。

我们也使用了 `for..in` 循环在列表中各项目间递归。从现在开始，你一定已经意识到列表也是一个序列。序列的特性会在后面的章节中讨论。

注意，我们在 `print` 语句的结尾使用了一个 逗号 来消除每个 `print` 语句自动打印的换行符。这样做有点难看，不过确实简单有效。

接下来，我们使用 `append` 方法在列表中添加了一个项目，就如前面已经讨论过的一样。然后我们通过打印列表的内容来检验这个项目是否确实被添加进列表了。打印列表只需简单地把列表传递给 `print` 语句，我们可以得到一个整洁的输出。

再接下来，我们使用列表的 `sort` 方法来对列表排序。需要理解的是，这个方法影响列表本身，而不是返回一个修改后的列表——这与字符串工作的方法不同。这就是我们所说的列表是 可变的 而字符串是 不可变的 。

最后，但我们完成了在市场购买一样东西的时候，我们想要把它从列表中删除。我们使用 `del` 语句来完成这个工作。这里，我们指出我们想要删除列表中的哪个项目，而 `del` 语句为我们从列表中删除它。我们指明我们想要删除列表中的第一个元素，因此我们使用 `del shoplist[0]`（记住，Python 从 0 开始计数）。

如果你想要知道列表对象定义的所有方法，可以通过 `help(list)` 获得完整的知识。

2. 元组

元组和列表十分类似，只不过元组和字符串一样是 不可变的 即你不能修改元组。元组通过圆括号中用逗号分割的项目定义。元组通常用在使语句或用户定义的函数能够安全地采用一组值的时候，即被使用的元组的值不会改变。

```
#!/usr/bin/python
# Filename: using_tuple.py

zoo = ('wolf', 'elephant', 'penguin')
print 'Number of animals in the zoo is', len(zoo)

new_zoo = ('monkey', 'dolphin', zoo)
print 'Number of animals in the new zoo is', len(new_zoo)
print 'All animals in new zoo are', new_zoo
print 'Animals brought from old zoo are', new_zoo[2]
print 'Last animal brought from old zoo is', new_zoo[2][2]
```

输出

```
$ python using_tuple.py
Number of animals in the zoo is 3
Number of animals in the new zoo is 3
All animals in new zoo are ('monkey', 'dolphin', ('wolf', 'elephant',
'penguin'))
Animals brought from old zoo are ('wolf', 'elephant', 'penguin')
Last animal brought from old zoo is penguin
```

它如何工作

变量 `zoo` 是一个元组，我们看到 `len` 函数可以用来获取元组的长度。这也表明元组也是一个序列。
由于老动物园关闭了，我们把动物转移到新动物园。因此，`new_zoo` 元组包含了一些已经在那里的动物和从老动物园带过来的动物。回到话题，注意元组之内的元组不会失去它的身份。
我们可以通过一对方括号来指明某个项目的位置从而来访问元组中的项目，就像我们对列表的用法一样。这被称作 索引 运算符。我们使用 `new_zoo[2]` 来访问 `new_zoo` 中的第三个项目。我们使用 `new_zoo[2][2]`来访问 `new_zoo` 元组的第三个项目的第三个项目。
含有 0 个或 1 个项目的元组。一个空的元组由一对空的圆括号组成，如 `myempty = ()`。然而，含有单个元素的元组就不那么简单了。你必须在第一个（唯一一个）项目后跟一个逗号，这样 Python 才能区分元组和表达式中一个带圆括号的对象。即如果你想要的是一个包含项目 `2` 的元组的时候，你应该指明 `singleton = (2 , )`。

给 Perl 程序员的注释 列表之中的列表不会失去它的身份，即列表不会像 Perl 中那样被打散。同样元组中的元组，或列表中的元组，或元组中的列表等等都是如此。只要是 Python，它们就只是使用另一个对象存储的对象。

元组最通常的用法是用在打印语句中，下面是一个例子：

```
#!/usr/bin/python
# Filename: print_tuple.py

age = 22
name = 'Swaroop'

print '%s is %d years old' % (name, age)
print 'Why is %s playing with that python?' % name
```

输出

```
$ python print_tuple.py
Swaroop is 22 years old
Why is Swaroop playing with that python?
```

它如何工作

`print` 语句可以使用跟着`%`符号的项目元组的字符串。这些字符串具备定制的功能。定制让输出满足某种特定的格式。定制可以是`%s` 表示字符串或`%d` 表示整数。元组必须按照相同的顺序来对应这些定制。
观察我们使用的第一个元组，我们首先使用`%s`，这对应变量 `name`，它是元组中的第一个项目。而第二个定制是`%d`，它对应元组的第二个项目 `age`。
Python 在这里所做的是把元组中的每个项目转换成字符串并且用字符串的值替换定制的位置。因此`%s` 被替换为变量 `name` 的值，依此类推。
`print` 的这个用法使得编写输出变得极其简单，它避免了许多字符串操作。它也避免了我们一直以来使用的逗号。
在大多数时候，你可以只使用`%s` 定制，而让 Python 来提你处理剩余的事情。这种方法对数同样奏效。然而，你可能希望使用正确的定制，从而可以避免多一层的检验程序是否正确。
在第二个 `print` 语句中，我们使用了一个定制，后面跟着`%`符号后的单个项目——没有圆括号。这只在字符串中只有一个定制的时候有效。

3. 字典

字典类似于你通过联系人名字查找地址和联系人详细情况的地址簿，即，我们把**键**（名字）和**值**（详细情况）联系在一起。注意，键必须是唯一的，就像如果有两个人恰巧同名的话，你无法找到正确的信息。

注意，你只能使用不可变的对象（比如字符串）来作为字典的键，但是你可以不可变或可变的对象作为字典的值。基本说来就是，你应该只使用简单的对象作为键。

键值对在字典中以这样的方式标记：`d = {key1 : value1, key2 : value2 }`。注意它们的键/值对用冒号分割，而各个对用逗号分割，所有这些都包括在花括号中。
记住字典中的键/值对是没有顺序的。如果你想要一个特定的顺序，那么你应该在使用前自己对它们排序。

字典是`dict`类的实例/对象。

```
#!/usr/bin/python
# Filename: using_dict.py

# 'ab' is short for 'a'ddress'b'ook

ab = {       'Swaroop'   : '[email protected]',
             'Larry'     : '[email protected]',
             'Matsumoto' : '[email protected]',
             'Spammer'   : '[email protected]'
     }

print "Swaroop's address is %s" % ab['Swaroop']

# Adding a key/value pair
ab['Guido'] = '[email protected]'

# Deleting a key/value pair
del ab['Spammer']

print '\nThere are %d contacts in the address-book\n' % len(ab)
for name, address in ab.items():
    print 'Contact %s at %s' % (name, address)

if 'Guido' in ab: # OR ab.has_key('Guido')
    print "\nGuido's address is %s" % ab['Guido']
```

输出

```
$ python using_dict.py
Swaroop's address is [email protected]

There are 4 contacts in the address-book

Contact Swaroop at [email protected]
Contact Matsumoto at [email protected]
```

```
    Contact Larry at [email protected]
    Contact Guido at [email protected]

    Guido's address is [email protected]
```

它如何工作

我们使用已经介绍过的标记创建了字典 `ab`。然后我们使用在列表和元组章节中已经讨论过的索引操作符来指定键，从而使用键/值对。我们可以看到字典的语法同样十分简单。
我们可以使用索引操作符来寻址一个键并为它赋值，这样就增加了一个新的键/值对，就像在上面的例子中我们对 Guido 所做的一样。

我们可以使用我们的老朋友——`del` 语句来删除键/值对。我们只需要指明字典和用索引操作符指明要删除的键，然后把它们传递给 `del` 语句就可以了。执行这个操作的时候，我们无需知道那个键所对应的值。
接下来，我们使用字典的 `items` 方法，来使用字典中的每个键/值对。这会返回一个元组的列表，其中每个元组都包含一对项目——键与对应的值。我们抓取这个对，然后分别赋给 `for..in` 循环中的变量 `name` 和 `address` 然后在 for 一块中打印这些值。
我们可以使用 `in` 操作符来检验一个键/值对是否存在，或者使用 `dict` 类的 `has_key` 方法。你可以使用 `help(dict)`来查看 `dict` 类的完整方法列表。
关键字参数与字典。如果换一个角度看待你在函数中使用的关键字参数的话，你已经使用了字典了！只需想一下——你在函数定义的参数列表中使用的键/值对。当你在函数中使用变量的时候，它只不过是使用一个字典的键（这在编译器设计的术语中被称作 符号表 ）。

4. 序列

列表、元组和字符串都是序列，但是序列是什么，它们为什么如此特别呢？序列的两个主要特点是**索引**操作符和**切片**操作符。索引操作符让我们可以从序列中抓取一个特定项目。切片操作符让我们能够获取序列的一个切片，即一部分序列。

```
#!/usr/bin/python
# Filename: seq.py

shoplist = ['apple', 'mango', 'carrot', 'banana']

# Indexing or 'Subscription' operation
print 'Item 0 is', shoplist[0]
print 'Item 1 is', shoplist[1]
print 'Item 2 is', shoplist[2]
print 'Item 3 is', shoplist[3]
```

```
print 'Item -1 is', shoplist[-1]
print 'Item -2 is', shoplist[-2]

# Slicing on a list
print 'Item 1 to 3 is', shoplist[1:3]
print 'Item 2 to end is', shoplist[2:]
print 'Item 1 to -1 is', shoplist[1:-1]
print 'Item start to end is', shoplist[:]

# Slicing on a string
name = 'swaroop'
print 'characters 1 to 3 is', name[1:3]
print 'characters 2 to end is', name[2:]
print 'characters 1 to -1 is', name[1:-1]
print 'characters start to end is', name[:]
```

输出

```
$ python seq.py
Item 0 is apple
Item 1 is mango
Item 2 is carrot
Item 3 is banana
Item -1 is banana
Item -2 is carrot
Item 1 to 3 is ['mango', 'carrot']
Item 2 to end is ['carrot', 'banana']
Item 1 to -1 is ['mango', 'carrot']
Item start to end is ['apple', 'mango', 'carrot', 'banana']
characters 1 to 3 is wa
characters 2 to end is aroop
characters 1 to -1 is waroo
characters start to end is swaroop
```

它如何工作

首先，我们来学习如何使用索引来取得序列中的单个项目。这也被称作是下标操作。每当你用方括号中的一个数来指定一个序列的时候，Python 会为你抓取序列中对应位置的项目。记住，Python 从 0 开始计数。因此，`shoplist[0]`抓取第一个项目，`shoplist[3]`抓取 `shoplist` 序列中的第四个元素。

索引同样可以是负数，在那样的情况下，位置是从序列尾开始计算的。因此，`shoplist[-1]`表示序列的最后一个元素而 `shoplist[-2]`抓取序列的倒数第二个项目。

切片操作符是序列名后跟一个方括号，方括号中有一对可选的数字，并用冒号分割。注意这与你使用的索引操作符十分相似。记住数是可选的，而冒号是必须的。

切片操作符中的第一个数（冒号之前）表示切片开始的位置，第二个数（冒号之后）表示切片到哪里结束。如果不指定第一个数，Python 就从序列首开始。如果没有指定第二个数，则 Python 会停止在序列尾。注意，返回的序列从开始位置 开始 ，刚好在 结束 位置之前结束。即开始位置是包含在序列切片中的，而结束位置被排斥在切片外。

这样，`shoplist[1:3]`返回从位置 1 开始，包括位置 2，但是停止在位置 3 的一个序列切片，因此返回一个含有两个项目的切片。类似地，`shoplist[:]`返回整个序列的拷贝。
你可以用负数做切片。负数用在从序列尾开始计算的位置。例如，`shoplist[:-1]`会返回除了最后一个项目外包含所有项目的序列切片。
使用 Python 解释器交互地尝试不同切片指定组合，即在提示符下你能够马上看到结果。序列的神奇之处在于你可以用相同的方法访问元组、列表和字符串。

5. 参考

当你创建一个对象并给它赋一个变量的时候，这个变量仅仅 参考 那个对象，而不是表示这个对象本身！也就是说，变量名指向你计算机中存储那个对象的内存。这被称作名称到对象的**绑定**。

一般说来，你不需要担心这个，只是在参考上有些细微的效果需要你注意。这会通过下面这个例子加以说明。

```
#!/usr/bin/python
# Filename: reference.py

print 'Simple Assignment'
shoplist = ['apple', 'mango', 'carrot', 'banana']
mylist = shoplist # mylist is just another name pointing to the same
object!

del shoplist[0]

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that both shoplist and mylist both print the same list without
# the 'apple' confirming that they point to the same object

print 'Copy by making a full slice'
```

```
mylist = shoplist[:] # make a copy by doing a full slice
del mylist[0] # remove first item

print 'shoplist is', shoplist
print 'mylist is', mylist
# notice that now the two lists are different
```

输出

```
$ python reference.py
Simple Assignment
shoplist is ['mango', 'carrot', 'banana']
mylist is ['mango', 'carrot', 'banana']
Copy by making a full slice
shoplist is ['mango', 'carrot', 'banana']
mylist is ['carrot', 'banana']
```

它如何工作

大多数解释已经在程序的注释中了。你需要记住的只是如果你想要复制一个列表或者类似的序列或者其他复杂的对象（不是如整数那样的简单 对象 ），那么你必须使用切片操作符来取得拷贝。如果你只是想要使用另一个变量名，两个名称都 参考 同一个对象，那么如果你不小心的话，可能会引来各种麻烦。

给 Perl 程序员的注释 记住列表的赋值语句不创建拷贝。你得使用切片操作符来建立序列的拷贝。

6. 更多字符串的内容

我们已经在前面详细讨论了字符串。我们还需要知道什么呢？那么，你是否知道字符串也是对象，同样具有方法。这些方法可以完成包括检验一部分字符串和去除空格在内的各种工作。

你在程序中使用的字符串都是 `str` 类的对象。这个类的一些有用的方法会在下面这个例子中说明。如果要了解这些方法的完整列表，请参见 `help(str)`。

```
#!/usr/bin/python
# Filename: str_methods.py

name = 'Swaroop' # This is a string object

if name.startswith('Swa'):
    print 'Yes, the string starts with "Swa"'

if 'a' in name:
```

```
    print 'Yes, it contains the string "a"'

if name.find('war') != -1:
    print 'Yes, it contains the string "war"'

delimiter = '_*_'
mylist = ['Brazil', 'Russia', 'India', 'China']
print delimiter.join(mylist)
```

输出

```
$ python str_methods.py
Yes, the string starts with "Swa"
Yes, it contains the string "a"
Yes, it contains the string "war"
Brazil_*_Russia_*_India_*_China
```

它如何工作

这里，我们看到使用了许多字符串方法。`startwith` 方法是用来测试字符串是否以给定字符串开始。`in` 操作符用来检验一个给定字符串是否为另一个字符串的一部分。

`find` 方法用来找出给定字符串在另一个字符串中的位置，或者返回**-1** 以表示找不到子字符串。`str` 类也有以一个作为分隔符的字符串 `join` 序列的项目的整洁的方法，它返回一个生成的大字符串。