第6章

Python语言中的列表与字典

6.0 引言

通过第5章的学习，我们已经了解了Python语言的基本知识。在这一章中，我们将学习两种关键的Python数据结构：列表和字典。

6.1 创建列表

面临问题

你想利用一个变量来存放一系列的值，而非单个值。

解决方案

你可以使用列表。在Python语言中，列表是按顺序存放的一组值的集合，所以，你可以通过位置来访问这些值。

你可以使用[和]来创建列表，两个方括号之间可以放入该列表的初始值。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>>
```

与类似于C之类的语言不同，它们对数组的定义非常严格，而Python在声明列表的时候，根本无需指定列表的长度。只要你喜欢，你可以随时改变列表中元素的数量。

进一步探讨

就像你在上面的例子中看到的那样，列表中的元素的类型不要求完全一致，当然，大部分情况下它们通常都是同类型的。

如果你想创建一个空列表，以便在将来需要时添加元素的话，你可以使用下列方式来新建空列表。

```
>>> a = []
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.2 访问列表元素

面临问题

你需要找到列表中的特定元素，或者修改它们。

解决方案

你可以使用[]表达式，通过指定元素在列表中的位置来访问它们，例如：

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a[1]
'Fred'
```

进一步探讨

列表中的位置（下标）是从 0 开始计数的，即第一个元素的下标为 0。

就像使用[]表达式从列表中读取元素值那样，你也可以使用它来修改指定位置的元素的值，例如：

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a[1] = 777
>>> a
[34, 777, 12, False, 72.3]
```

如果你试图利用一个很大的下标来修改（或就这里来说，读取）一个元素的话，就会收到一个“Index out of range”错误消息。

```
>>> a[50] = 777
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list assignment index out of range
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.3 确定列表长度

面临问题

你需要知道列表中含有多少个元素。

解决方案

你可以使用 Python 提供的 len 函数，具体如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> len(a)
5
```

进一步探讨

此外，len 命令还可用于字符串（见 5.13 节）。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.4 为列表添加元素

面临问题

你希望为列表添加元素。

解决方案

为此，你可以使用 Python 提供的 append、insert 或者 extend 函数。

要想在列表末尾添加单个元素的话，你可以使用 append 函数。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.append("new")
>>> a
[34, 'Fred', 12, False, 72.3, 'new']
```

进一步探讨

有时候，你想要做的并非在列表尾部添加新元素，而是要把新元素添加到指定的位置。为此，你可以使用 insert 命令。该命令的第一个参数是希望插入元素的位置的索引，第二个参数是希望插入的元素，具体如下所示。

```
>>> a.insert(2, "new2")
>>> a
[34, 'Fred', 'new2', 12, False, 72.3]
```

需要注意的是新插入元素之后，其后面元素的下标会依次加 1。

append 和 insert 函数每次只能向列表中添加一个元素。而对于 extend 函数来说，它可以将一个列表中的所有元素一次性添加到另一个列表的末尾。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> b = [74, 75]
>>> a.extend(b)
>>> a
[34, 'Fred', 12, False, 72.3, 74, 75]
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.5 删除列表元素

面临问题

你需要从列表中删除元素。

解决方案

你可以使用 Python 提供的 pop 函数。

当使用 pop 命令时，如果没有参数的话，它会删除列表中的最后一个元素。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.pop()
72.3
>>> a
[34, 'Fred', 12, False]
```

进一步探讨

请注意，pop 函数会返回从列表中删除的元素的值。

要想删除列表中指定位置的元素，而非最后一个元素的话，可以给 pop 命令指定待删除元素的位置，具体如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> a.pop(0)
34
```

如果你使用了一个列表尾部之外的位置下标的话，将会收到一个“Index out of range”异常消息。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.6 通过解析字符串创建列表

面临问题

你需要将一个包含由特定字符间隔开的单词的字符串转换成一个字符串数组，并且让该数组中的每个字符串都是一个单词。

解决方案

你可以使用 Python 提供的字符串函数 split。

在使用 split 命令的时候，如果不带参数的话，它会把字符串中的单词分割为数组的单个元素，具体如下所示。

```
>>> "abc def ghi".split()
['abc', 'def', 'ghi']
```

你可以使用含参数的 split 函数来拆分字符串，使用该参数作为一个分割器，具体如下所示：

```
>>> "abc--de--ghi".split('--')
['abc', 'de', 'ghi']
```

进一步探讨

当你需要从文件导入数据的时候，split 命令是非常有用的。同时，该命令还有一个可选参数，这个参数通常是一个字符串，用来指定分割字符串时所用的分隔符。所以，如果你想使用逗号作为分隔符的话，可以像下面这样来分割字符串。

```
>>> "abc,def,ghi".split(',')
['abc', 'def', 'ghi']
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.7 遍历列表

面临问题

你需要逐个对列表中的所有元素执行某些操作。

解决方案

你可以使用 Python 语言提供的 for 命令，具体如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for x in a:
...     print(x)
...
34
Fred
12
False
72.3
>>>
```

进一步探讨

在关键字 for 之后，需要紧跟一个变量名（本例为 x）。这个变量称为循环变量，它的取值将遍历关键字 in 之后的列表中的每一个元素。

For 语句下面的缩进行将会对列表中的每个元素都执行一次。

每循环一次，x 的值就会被设置为对应位置的元素的值。实际上，你可以使用 x 输出相应的值，具体见上例所示。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.8 枚举列表

面临问题

你需要对列表中的每个元素逐次执行某些代码，同时还需要知道各个元素的位置下标。

解决方案

你可以使用 Python 语言提供的 for 命令与 enumerate 命令。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for (i, x) in enumerate(a):
...     print(i, x)
...
(0, 34)
(1, 'Fred')
(2, 12)
(3, False)
```

```
(4, 72.3)
>>>
```

进一步探讨

在枚举每个值的时候，通常都需要知道它们在列表中的相应的位置。一个替代方法是简单使用一个下标变量进行计数，然后通过[]语法来访问相应的值，如下所示。

```
>>> a = [34, 'Fred', 12, False, 72.3]
>>> for i in range(len(a)):
...     print(i, a[i])
...
(0, 34)
(1, 'Fred')
(2, 12)
(3, False)
(4, 72.3)
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

要想在不知道每个元素的下标位置的情况下来遍历列表的话，请阅读 6.7 节。

6.9 列表排序

面临问题

你需要对列表中的元素进行排序。

解决方案

你可以使用 Python 语言提供的 sort 命令。

```
>>> a = ["it", "was", "the", "best", "of", "times"]
>>> a.sort()
>>> a
['best', 'it', 'of', 'the', 'times', 'was']
```

进一步探讨

当对一个列表进行排序的时候，你实际上是对列表本身进行了相应的修改，而非返回了原列表排序之后的一个副本。这就是说，如果你还需要原始列表的话，那么在对其进行排序之前，需要先用标准程序库中的 copy 命令生成原始列表的副本。

```
>>> import copy
```

```
>>> a = ["it", "was", "the", "best", "of", "times"]
>>> b = copy.copy(a)
>>> b.sort()
>>> a
['it', 'was', 'the', 'best', 'of', 'times']
>>> b
['best', 'it', 'of', 'the', 'times', 'was']
>>>
```

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.10 分割列表

面临问题

你需要利用原列表的部分内容来生成一个子列表。

解决方案

为此，你可以使用 Python 语言的[:]结构。下面的例子将返回一个列表，其内容由原来列表中下标位置 1 到下标位置 2（位于：之后的数字将会被排除在外）之间的元素所组成。

```
>>> l = ["a", "b", "c", "d"]
>>> l[1:3]
['b', 'c']
```

需要注意的是字符的位置是从 0 开始计数的，所以，位置 1 表示字符串中的第 2 个字符，位置 5 表示字符串中的第 6 个字符，不过，由于位置的范围并不包含最右边的数字，所以这里并不包括字母 d。

进一步探讨

实际上，[:]的作用是非常强大的，其中的两个参数，任何一个都可以忽略掉，这时候列表的开始和结束位置会视情况而定。举例来说：

```
>>> l = ["a", "b", "c", "d"]
>>> l[:3]
['a', 'b', 'c']
>>> l[3:]
['d']
>>>
```

此外，你还可以使用负数来表示从字符串的末尾开始反向计数。下面的例子将会返回列表中最后两个元素。

```
>>> l[-2:]
```

```
['c', 'd']
```

顺便提一句，上面的例子中，l[:-2]将返回['a','b']。

参考资料

在 6.1 节 ~ 6.11 节中的示例中，都会涉及列表的应用。

你还可以参考 5.15 节，其中的语法同样适用于字符串。

6.11 将函数应用于列表

面临问题

你需要对列表中的每个元素都应用某个函数，并收集相应的结果。

解决方案

你可以使用 Python 语言的一个特性，该特性名为推导式（comprehensions）。

下面的例子会将列表的每个字符串元素转换为大写形式，并返回长度与原来相同的一个新列表，只是所有字符串都已经变成大写了。

```
>>> l = ["abc", "def", "ghi", "ijk"]
>>> [x.upper() for x in l]
['ABC', 'DEF', 'GHI', 'IJK']
```

尽管这种方式有些乱，但是仍然没有理由来拒绝联合使用这种语句，将一个推导式嵌入到另一个语句中。

进一步探讨

这是使用推导式的一种非常简洁的方法。整个表达式都要求放入方括号（[]）之中。推导式的第一个元素是需要施加于列表各元素的代码。推导式的其余部分看上去更像是一个列表遍历命令（见 6.7 节）。循环变量紧跟在关键字 for 后面，关键字 in 之后是需要用到的列表。

参考资料

在 6.1 节 ~ 6.11 节中的示例都会涉及列表的应用。

6.12 创建字典

面临问题

你需要创建一个查找表，其中的值和键已经关联在一起。

解决方案

你可以使用 Python 的字典。

当你需要按照顺序访问一组元素，或者总是知道想要使用元素的下标时，使用数组是你的不二之选。字典可以代替列表来存放一组数据，不过两者之间的组织方式区别较大。

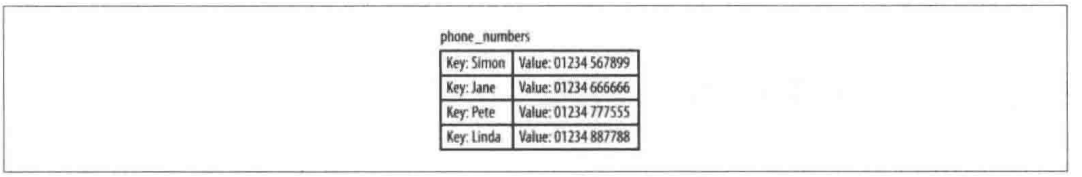

图 6-1　字典的组织方式

图 6-1 所示 Python 的字典使用的是键/值对的方式，也就是说你可以使用键来高效地访问值，而不必搜索整个字典。

为了创建一个字典，可以使用{}表达式。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
```

进一步探讨

在上面的例子中，字典的键都是些字符串，但是这不是硬性要求；这些键也可以是数字，事实上任何数据类型都是允许的，不过字符串是最常用的一种形式。

字典的值也可以是任何类型的数据，包括其他字典或者列表。下面的例子中，我们创建了一个字典（a），并且随后将其作为了第二个字典（b）的值。

```
>>> a = {'key1':'value1', 'key2':2}
>>> a
{'key2': 2, 'key1': 'value1'}
>>> b = {'b_key1':a}
>>> b
{'b_key1': {'key2': 2, 'key1': 'value1'}}
```

当你显示字典的内容时，你可能会发现，字典中的各个元素的顺序与当初创建并利用某些内容来初始化该字典时的顺序并不一定一致。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers
{'Jane': '01234 666666', 'Simon': '01234 567899'}
```

与列表不同的是字典并没有按顺序存放元素的要求。正是由于这种内在的表示方式，导致字典元素的顺序无论怎么看都是随机的。

导致顺序随机的原因在于这种基本数据结构实际上是一种哈希表。哈希表利用哈希函

数来决定将值存放到哪里，哈希函数能够为任意对象产生一个对应的数字。

要想了解哈希表的更多知识，请访问 Wikipedia(http://en.wikipedia.org/wiki/Hash_table)。

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及字典的应用。

6.13 访问字典

面临问题

你需要查找和修改字典元素。

解决方案

你可以使用 Python 的[]表达式。在括号中，你需要指定待访问的元素的键，具体如下所示。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers['Simon']
'01234 567899'
>>> phone_numbers['Jane']
'01234 666666'
```

进一步探讨

这个查找只能在单个方向上进行，即从键到值。

如果你使用的键并不存在于该字典中的话，那么就会收到一个 key error，具体如下例所示。

```
{'b_key1': {'key2': 2, 'key1': 'value1'}}
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers['Phil']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Phil'
>>>
```

就像使用[]表达式从字典中读取值一样，你还可以使用它来添加新的值，或者覆盖现有的值。

下面的例子展示了如何向字典中新加一个元素，该元素的键和值分别为 Pete 和 01234777555。

```
>>> phone_numbers['Pete'] = '01234 777555'
>>> phone_numbers['Pete']
'01234 777555'
```

如果该键尚未被这个字典所用的话，就会自动添加一个新元素。如果该键已经存在的话，那么无论之前该键对应的值是什么，都会被新值所覆盖掉。

这一点与从字典中读取值有所不同，那种情况下，使用未知的键会导致出错。

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及到字典的应用。

关于错误处理的详细情况，请参考 7.10 节。

6.14 删除字典元素

面临问题

你需要从字典中删除元素。

解决方案

你可以使用 pop 命令，同时指定要删除的元素的键。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> phone_numbers.pop('Jane')
'01234 666666'
>>> phone_numbers
{'Simon': '01234 567899'}
```

进一步探讨

命令 pop 将会返回从字典中删除的元素的值。

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及到字典的应用。

6.15 遍历字典

面临问题

你想对字典中的所有元素依次执行某种操作。

解决方案

你可以使用 for 命令来遍历字典的键。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> for name in phone_numbers:
...     print(name)
...
Jane
Simon
```

进一步探讨

还有其他几种方法，也可以用来遍历字典。

如果你需要同时访问键和值的话，下面的方法将非常有用。

```
>>> phone_numbers = {'Simon':'01234 567899', 'Jane':'01234 666666'}
>>> for name, num in phone_numbers.items():
...     print(name + " " + num)
...
Jane 01234 666666
Simon 01234 567899
```

参考资料

在 6.12 节 ~ 6.15 节之间的所有示例代码都涉及到字典的应用。

涉及 for 命令的用法的小节，包括 5.21 节、6.7 节和 6.11 节。