第6章 字典

在本章中，将学习能够将相关信息关联起来的Python字典，以及如何访问和修改字典中的信息。 字典可存储的信息量几乎不受限制，因此我们会演示如何遍历字典中的数据。 另外，还将学习存储字典的列表、存储列表的字典和存储字典的字典。 理解字典后，就能够更准确地为各种真实物体建模。 你可以创建一个表示人的字典，然后想在其中存储多少信息就存储多少信息：姓名、年龄、地址、职业，以及能描述他的任何方面。 你还能够存储任意两种相关的信息，如一系列单词及其含义，一系列人名及其喜欢的数，以及一系列山脉及其海拔，等等。

6.1 一个简单的字典

In [7]:
```
location = {'Latitude': 20.04623, 'Longitude': 110.19565}
print(location['Latitude'])
print(location['Longitude'])
```

```
20.04623
110.19565
```

字典location存储了海口的GPS位置。

与大多数编程概念一样，要熟练使用字典，也需要一段时间的练习。 使用字典一段时间后，你就会明白为何它们能够高效地模拟现实世界中的情形。

6.2 使用字典

在Python中，字典是一系列键值对（key:value)。 每个键都与一个值相关联，你可使用键来访问相关联的值。 与键相关联的值可以是数、字符串、列表乃至字典。 事实上，可将任何Python对象用作字典中的值。 在Python中，字典用放在花括号（{}）中的一系列键值对表示，如前面的示例所示：

In [9]:
```
location = {'Latitude': 20.04623, 'Longitude': 110.19565}
marks = {'数学':79,'语文':80,'外语':92}

print(marks)
```

```
{'数学': 79, '语文': 80, '外语': 92}
```

6.2.1 访问字典中的值

要获取与键相关联的值，可依次指定字典名和放在方括号内的键，如下所示：

In [9]:
```
location = {'Latitude': 20.04623, 'Longitude': 110.19565}

print(location['Latitude'])
```

```
20.04623
```

6.2.2 添加键值对

字典是一种动态结构，可随时在其中添加键值对。 要添加键值对，可依次指定字典名、用方括号括起的键和相关联的值。 下面来在字典students_mark中添加一项信息：赵六的分数，如下所示：

In [21]:
```
students_mark = {'张三':99, '李四':96,'王五':60}
print(students_mark)

students_mark['赵六'] = 54
print(students_mark)
```

```
{'张三': 99, '李四': 96, '王五': 60}
{'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
```

注意在Python3.7中，字典中元素的排列顺序与定义时相同。 如果将字典打印出来或遍历其元素，将发现元素的排列顺序与添加顺序相同。

6.2.3 先创建一个空字典

在空字典中添加键值对有时候可提供便利，而有时候必须这样做。 为此，可先使用一对空花括号定义一个字典，再分行添加各个键。

In [ ]:
```
students_mark = {}

students_mark['赵六'] = 54
print(students_mark)
```

使用字典来存储用户提供的数据或在编写能自动生成大量键值对的代码时，通常需要先定义一个空字典。

6.2.4 修改字典中的值

要修改字典中的值，可依次指定字典名、用方括号括起的键，以及与该键相关联的新值。 例如，假设赵六补考合格，需要将分数改变：

In [1]:
```
students_mark = {}

students_mark['赵六'] = 54
print(students_mark)

students_mark['赵六'] = 80
print(students_mark)
```

```
{'赵六': 54}
{'赵六': 80}
```

6.2.5 删除键值对

对于字典中不再需要的信息，可使用del语句将相应的键值对彻底删除。使用del语句时，必须指定字典名和要删除的键。

In [1]:
```
students = {'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
print(students)

del students['张三']
print(students)
```

```
{'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
{'李四': 96, '王五': 60, '赵六': 54}
```

6.2.6 由类似对象组成的字典

可以使用字典来存储众多对象的同一种信息。 例如，假设你要调查很多人，询问他们最喜欢的编程语言，可使用一个字典来存储这种简单调查的结果，如下所示：

In [5]:
```
favorite_languages = {
    'Jen': 'python',
    'Sarah': 'c',
    'Edward': 'ruby',
    'Phil': 'python',
}
```

如你所见，我们将一个较大的字典放在了多行中。 每个键都是一个被调查者的名字，而每个值都是被调查者喜欢的语言。 确定需要使用多行来定义字典时，要在输入左花括号后按回车键。 在下一行缩进四个空格，指定第一个键值对，并在它后面加上一个逗号。 此后再按回车键时，文本编辑器将自动缩进后续键值对，且缩进量与第一个键值对相同。

In [6]:
```
favorite_languages['Sarah']
```

Out[6]:
```
'c'
```

6.2.7 使用get() 来访问值

使用放在方括号内的键从字典中获取感兴趣的值时，可能会引发问题：如果指定的键不存在就会出错。 如果你要求获取外星人的分数，而这个外星人没有分数，结果将如何呢？下面来看一看：

使用放在方括号内的键从字典中获取感兴趣的值时，可能会引发问题：如果指定的键不存在就会出错。 如果你要求获取学生的分数，而这个学生没有分数，结果将如何呢？下面来看一看：

In [7]:
```
students = {'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
print(students['孙七'])

# 这将导致Python显示traceback，指出存在键值错误（KeyError）
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [7], in <cell line: 2>()
      1 students = {'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
----> 2 print(students['孙七'])

KeyError: '孙七'
```

第10章将详细介绍如何处理类似的错误，但就字典而言，可使用方法get()在指定的键不存在时返回一个默认值，从而避免这样的错误。 方法get()的第一个参数用于指定键，是必不可少的； 第二个参数为指定的键不存在时要返回的值，是可选的：

In [9]:
```
students = {'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
print(students.get('孙七','No point value assigned.'))
```

```
No point value assigned.
```

如果字典中有键'孙七'，将获得与之相关联的值； 如果没有，将获得指定的默认值。 虽然这里没有键'孙七'，但将获得一条清晰的消息，不会引发错误 如果指定的键有可能不存在，应考虑使用方法get()，而不要使用方括号表示法。 注意 调用get()时，如果没有指定第二个参数且指定的键不存在，Python将返回值None。 这个特殊值表示没有相应的值。 None并非错误，而是一个表示所需值不存在的特殊值，第8章将介绍它的其他用途。

6.3 遍历字典

一个Python字典可能只包含几个键值对，也可能包含数百万个键值对。 鉴于字典可能包含大量数据，Python支持对字典进行遍历。 字典可用于以各种方式存储信息，因此有多种遍历方式：可遍历字典的所有键值对，也可仅遍历键或值。

In [12]:
```
students = {'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
for key, value in students.items():
    print(f"\nKey: {key}, Value:{value}")
```

```
Key: 张三, Value:99

Key: 李四, Value:96

Key: 王五, Value:60

Key: 赵六, Value:54
```

In [14]:
```
students = {'张三': 99, '李四': 96, '王五': 60, '赵六': 54}
for key in students.keys():
    print(f"\nKey: {key}, Value:{students[key]}")
```

```
Key: 张三, Value:99

Key: 李四, Value:96

Key: 王五, Value:60

Key: 赵六, Value:54
```

In [19]:
```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}

for name, language in favorite_languages.items():
    print(f"{name.title()}'s favorite language is {language.title()}")
```

```
Jen's favorite language is Python
Sarah's favorite language is C
Edward's favorite language is Ruby
Phil's favorite language is Python
```

In [20]:
```
friends = ['phil', 'sarah']

for name in favorite_languages.keys():
    print(f"Hi {name.title()}.")

    if name in friends:
        language = favorite_languages[name].title()
        print(f"\t{name.title()}, I see you love {language}!")
```

```
Hi Jen.
Hi Sarah.
        Sarah, I see you love C!
Hi Edward.
Hi Phil.
        Phil, I see you love Python!
```

还可使用方法keys() 确定某个人是否接受了调查。 下面的代码确定Erin是否接受了调查：

In [21]:
```
favorite_languages = {
```

```
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}

if 'erin' not in favorite_languages.keys():
    print("Erin, please take our poll!")
```

```
Erin, please take our poll!
```

6.3.3 按特定顺序遍历字典中的所有键

从Python 3.7起，遍历字典时将按插入的顺序返回其中的元素。 不过在有些情况下，你可能要按与此不同的顺序遍历字典。 要以特定顺序返回元素，一种办法是在for 循环中对返回的键进行排序。 为此，可使用函数sorted()来获得按特定顺序排列的键列表的副本：

In [23]:
```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}

for name in sorted(favorite_languages.keys()):
    print(f"{name.title()}, thank you for taking the poll.")
```

```
Edward, thank you for taking the poll.
Jen, thank you for taking the poll.
Phil, thank you for taking the poll.
Sarah, thank you for taking the poll.
```

这条for 语句类似于其他for 语句，不同之处是对方法dictionary.keys()的结果调用了函数sorted()。 这让Python列出字典中的所有键，并在遍历前对这A个列表进行排序。

6.3.4 遍历字典中的所有值

如果主要对字典包含的值感兴趣，可使用方法values()来返回一个值列表，不包含任何键。 例如，假设我们想获得一个列表，其中只包含被调查者选择的各种语言，而不包含被调查者的名字，可以这样做：

In [24]:
```
favorite_languages = {
'jen': 'python',
'sarah': 'c',
'edward': 'ruby',
'phil': 'python',
}

print("The following languages have been mentioned:")
for language in favorite_languages.values():
    print(language.title())
```

```
The following languages have been mentioned:
Python
C
Ruby
Python
```

这种做法提取字典中所有的值，而没有考虑是否重复。 涉及的值很少时，这也许不是问题，但如果被调查者很多，最终的列表可能包含大量重复项。 为剔除重复项，可使用集合（set）。集合中的每个元素都必须是独一无二的：

In [25]:
```
print("The following languages have been mentioned:")
for language in set(favorite_languages.values()):
    print(language.title())
```

```
The following languages have been mentioned:
Ruby
Python
C
```

随着你更深入地学习Python，经常会发现它内置的功能可帮助你以希望的方式处理数据。 注意 可使用一对花括号直接创建集合，并在其中用逗号分隔元素：

In [26]:
```
languages = {'python', 'ruby', 'python', 'c'}
print(languages)
```

```
{'ruby', 'python', 'c'}
```

集合和字典很容易混淆，因为它们都是用一对花括号定义的。 当花括号内没有键值对时，定义的很可能是集合。 不同于列表和字典，集合不会以特定的顺序存储元素。

6.4 嵌套

有时候，需要将一系列字典存储在列表中，或将列表作为值存储在字典中，这称为嵌套。 你可以在列表中嵌套字典、在字典中嵌套列表甚至在字典中嵌套字典。 正如下面的示例将演示的，嵌套是一项强大的功能。

6.4.1 字典列表

字典students包含一个班级的信息，但无法存储第二个班级的信息，更别说全院学生的信息了。 如何管理所有学生呢？一种办法是创建一个班级列表，其中每个班级都是一个字典，包含有关该班级学生的信息。 例如，下面的代码创建一个包含三个班级的列表：

In [30]:
```
students_0 = {'学生0': 100, '学生1': 50}
students_1 = {'学生0': 99, '学生1': 90}
students_2 = {'学生0': 88, '学生1': 87}
all_students = [students_0, students_1, students_2]
for students in all_students:
    print(students)
```

```
{'学生0': 100, '学生1': 50}
{'学生0': 99, '学生1': 90}
{'学生0': 88, '学生1': 87}
```

6.4.2 在字典中存储字典

可在字典中嵌套字典，但这样做时，代码可能很快复杂起来。

In [31]:
```
students_0 = {'学生0': 100, '学生1': 50}
students_1 = {'学生0': 99, '学生1': 90}
students_2 = {'学生0': 88, '学生1': 87}
all_students = {'班级0': students_0, '班级1': students_1, '班级2': students_2}

for cls in all_students.keys():
    print(students_0)
```

```
{'学生0': 100, '学生1': 50}
{'学生0': 100, '学生1': 50}
{'学生0': 100, '学生1': 50}
```

6.5 小结

在本章中，你学习了：

- 如何定义字典，以及如何使用存储在字典中的信息；
- 如何访问和修改字典中的元素，以及如何遍历字典中的所有信息；
- 如何遍历字典中所有的键值对、所有的键和所有的值；
- 如何在列表中嵌套字典、在字典中嵌套列表以及在字典中嵌套字典。

在下一章中，你将学习while循环以及如何从用户那里获取输入。 这是激动人心的一章，让你知道如何将程序变成交互性的： 能够对用户输入做出响应。

In [ ]:
```
# The end
```