练习答案

这里提供了部分练习的答案。请不要着急看答案，设法搞明白新学的概念再看。另外，很多练习的解决方案都有多种，这里列出的只是其中一种。

这里没有提供第 12~14 章及第 18~20 章的练习答案，因为这些练习实际上都是项目。如果你在完成这些练习的过程中遇到困难，可前往 Stack Overflow 或 r/learnpython 提问，也可与作者联系。

第 2 章

练习 2-2　多条简单消息

将一条消息赋给变量，并将其打印出来；再将变量的值修改为一条新消息，并将其打印出来。

```
msg = "I love learning to use Python."
print(msg)

msg = "It's really satisfying!"
print(msg)
```

输出：

```
I love learning to use Python.
It's really satisfying!
```

练习 2-5　名言

找一句你钦佩的名人说的名言，将其姓名和名言打印出来。输出应类似于下面这样（包括引号）：Albert Einstein once said, “A person who never made a mistake never tried anything new.”

```
print('Albert Einstein once said, "A person who never made a mistake')
print('never tried anything new."')
```

输出：

```
Albert Einstein once said, "A person who never made a mistake
never tried anything new."
```

练习 2-7　剔除人名中的空白

用变量表示一个人的名字，并在开头和末尾都包含一些空白字符。务必至少使用字符组合"\t"和"\n"各一次。

打印这个人名，显示开头和末尾的空白。然后，分别使用剔除函数 `lstrip()`、`rstrip()` 和 `strip()` 对人名进行处理，并将结果打印出来。

```
name = "\tEric Matthes\n"

print("Unmodified:")
print(name)

print("\nUsing lstrip():")
print(name.lstrip())

print("\nUsing rstrip():")
print(name.rstrip())

print("\nUsing strip():")
print(name.strip())
```

输出：

```
Unmodified:
    Eric Matthes


Using lstrip():
Eric Matthes


Using rstrip():
    Eric Matthes

Using strip():
Eric Matthes
```

练习 2-9　最喜欢的数

用一个变量来表示你最喜欢的数，再使用这个变量创建一条消息，指出你最喜欢的数是什么，然后将这条消息打印出来。

```
fav_num = 42
msg = f"My favorite number is {fav_num}."

print(msg)
```

输出：

```
My favorite number is 42.
```

第 3 章

练习 3-1　姓名

将一些朋友的姓名存储在一个列表中，并将其命名为 `names`。依次访问该列表中的每个元素，从而将每个朋友的姓名都打印出来。

```
names = ['ron', 'tyler', 'dani']

print(names[0])
print(names[1])
print(names[2])
```

输出：

```
ron
tyler
dani
```

练习 3-2　问候语

继续使用练习 3-1 中的列表，但不打印朋友的姓名，而为每人打印一条消息。每条消息都包含相同的问候语，但抬头为相应朋友的姓名。

```
names = ['ron', 'tyler', 'dani']

msg = f"Hello, {names[0].title()}!"
```

```
print(msg)

msg = f"Hello, {names[1].title()}!"
print(msg)

msg = f"Hello, {names[2].title()}!"
print(msg)
```

输出：

```
Hello, Ron!
Hello, Tyler!
Hello, Dani!
```

练习 3-4　嘉宾名单

如果你可以邀请任何人共进晚餐（无论是故去的还是在世的），你会邀请哪些人？请创建一个列表，其中包含至少三个你想邀请的人；然后，使用这个列表打印消息，邀请这些人来与你共进晚餐。

```
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")
```

输出：

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.
```

练习 3-5　修改嘉宾名单

你刚得知有位嘉宾无法赴约，因此需要另外邀请一位嘉宾。

- 以完成练习 3-4 时编写的程序为基础，在程序末尾添加一条 `print` 语句，指出哪位嘉宾无法赴约。
- 修改嘉宾名单，将无法赴约的嘉宾的姓名替换为新邀请的嘉宾的姓名。
- 再次打印一系列消息，向名单中的每位嘉宾发出邀请。

```
# 邀请一些嘉宾与你共进晚餐。
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"\nSorry, {name} can't make it to dinner.")

# Jack 无法赴约，转而邀请 Gary。
del(guests[1])
guests.insert(1, 'gary snyder')

# 重新打印邀请函。
name = guests[0].title()
print(f"\n{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")
```

输出：

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.

Sorry, Jack Turner can't make it to dinner.

Guido Van Rossum, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.
```

练习 3-6　添加嘉宾

你刚找到了一个更大的餐桌，可容纳更多的嘉宾。请想想你还想邀请哪三位嘉宾。

- 以你完成练习 3-4 或 3-5 时编写的程序为基础，在程序末尾添加一条 `print` 语句，指出你找到了一个更大的餐桌。
- 使用 `insert()` 将一位新嘉宾添加到名单开头。
- 使用 `insert()` 将另一位新嘉宾添加到名单中间。
- 使用 `append()` 将最后一位新嘉宾添加到名单末尾。
- 打印一系列消息，向名单中的每位嘉宾发出邀请。

```
# 邀请一些嘉宾与你共进晚餐。
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"\nSorry, {name} can't make it to dinner.")
```

```
# Jack 无法赴约，转而邀请 Gary。
del(guests[1])
guests.insert(1, 'gary snyder')

# 重新打印邀请函。
name = guests[0].title()
print(f"\n{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")

# 找到了更大的餐桌，再邀请一些嘉宾。
guests.insert(0, 'frida kahlo')
guests.insert(2, 'reinhold messner')
guests.append('elizabeth peratrovich')

name = guests[0].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")

name = guests[3].title()
print(f"{name}, please come to dinner.")

name = guests[4].title()
print(f"{name}, please come to dinner.")

name = guests[5].title()
print(f"{name}, please come to dinner.")
```

输出：

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.

Sorry, Jack Turner can't make it to dinner.

Guido Van Rossum, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.

We got a bigger table!
Frida Kahlo, please come to dinner.
Guido Van Rossum, please come to dinner.
Reinhold Messner, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.
Elizabeth Peratrovich, please come to dinner.
```

练习 3-7　缩减名单

你刚得知新购买的餐桌无法及时送达，因此只能邀请两位嘉宾。

- 以完成练习 3-6 时编写的程序为基础，在程序末尾添加一行代码，打印一条消息，指出你只能邀请两位嘉宾共进晚餐。
- 使用 `pop()` 不断地删除名单中的嘉宾，直到只有两位嘉宾为止。每次从名单中弹出一位嘉宾时，都打印一条消息，让该嘉宾知悉你很抱歉，无法邀请他来共进晚餐。
- 对于余下的两位嘉宾中的每一位，都打印一条消息，指出他依然在邀请之列。
- 使用 `del` 将最后两位嘉宾从名单中删除，让名单变成空的。打印该名单，核实程序结束时名单确实是空的。

```
# 邀请一些嘉宾与你共进晚餐。
guests = ['guido van rossum', 'jack turner', 'lynn hill']

name = guests[0].title()
print(f"{name}, please come to dinner.")
```

```
name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"\nSorry, {name} can't make it to dinner.")

# Jack 无法赴约，转而邀请 Gary。
del(guests[1])
guests.insert(1, 'gary snyder')

# 重新打印邀请函。
name = guests[0].title()
print(f"\n{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")

# 找到了更大的餐桌，再邀请一些嘉宾。
print("\nWe got a bigger table!")
guests.insert(0, 'frida kahlo')
guests.insert(2, 'reinhold messner')
guests.append('elizabeth peratrovich')

name = guests[0].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

name = guests[2].title()
print(f"{name}, please come to dinner.")
```

```
name = guests[3].title()
print(f"{name}, please come to dinner.")

name = guests[4].title()
print(f"{name}, please come to dinner.")

name = guests[5].title()
print(f"{name}, please come to dinner.")

# 糟糕，餐桌无法及时送达！
print("\nSorry, we can only invite two people to dinner.")

name = guests.pop()
print(f"Sorry, {name.title()} there's no room at the table.")

name = guests.pop()
print(f"Sorry, {name.title()} there's no room at the table.")

name = guests.pop()
print(f"Sorry, {name.title()} there's no room at the table.")

name = guests.pop()
print(f"Sorry, {name.title()} there's no room at the table.")

# 应该只剩下两位嘉宾了，向他们发出邀请。
name = guests[0].title()
print(f"{name}, please come to dinner.")

name = guests[1].title()
print(f"{name}, please come to dinner.")

# 清空名单。
del(guests[0])
del(guests[0])

# 核实名单是空的。
print(guests)
```

输出：

```
Guido Van Rossum, please come to dinner.
Jack Turner, please come to dinner.
Lynn Hill, please come to dinner.

Sorry, Jack Turner can't make it to dinner.

Guido Van Rossum, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.

We got a bigger table!
Frida Kahlo, please come to dinner.
Guido Van Rossum, please come to dinner.
Reinhold Messner, please come to dinner.
Gary Snyder, please come to dinner.
Lynn Hill, please come to dinner.
Elizabeth Peratrovich, please come to dinner.

Sorry, we can only invite two people to dinner.
Sorry, Elizabeth Peratrovich there's no room at the table.
Sorry, Lynn Hill there's no room at the table.
Sorry, Gary Snyder there's no room at the table.
Sorry, Reinhold Messner there's no room at the table.
Frida Kahlo, please come to dinner.
Guido Van Rossum, please come to dinner.
[]
```

练习 3-8. 放眼世界

想出至少 5 个你渴望去旅游的地方。

- 将这些地方存储在一个列表中，并确保其中的元素不是按字母顺序排列的。
- 按原始排列顺序打印该列表。不要考虑输出是否整洁的问题，只管打印原始 Python 列表。
- 使用 `sorted()` 按字母顺序打印这个列表，但不修改它。

- 再次打印该列表，核实排列顺序未变。
- 使用 `sorted()` 按与字母顺序相反的顺序打印这个列表，但不修改它。
- 再次打印该列表，核实排列顺序未变。
- 使用 `reverse()` 修改列表元素的排列顺序。打印该列表，核实排列顺序确实变了。
- 使用 `reverse()` 再次修改列表元素的排列顺序。打印该列表，核实已恢复到原来的排列顺序。
- 使用 `sort()` 修改该列表，使其元素按字母顺序排列。打印该列表，核实排列顺序确实变了。
- 使用 `sort()` 修改该列表，使其元素按与字母顺序相反的顺序排列。打印该列表，核实排列顺序确实变了。

```
locations = ['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

print("Original order:")
print(locations)

print("\nAlphabetical:")
print(sorted(locations))

print("\nOriginal order:")
print(locations)

print("\nReverse alphabetical:")
print(sorted(locations, reverse=True))

print("\nOriginal order:")
print(locations)

print("\nReversed:")
locations.reverse()
print(locations)

print("\nOriginal order:")
locations.reverse()
print(locations)
```

```
print("\nAlphabetical")
locations.sort()
print(locations)

print("\nReverse alphabetical")
locations.sort(reverse=True)
print(locations)
```

输出：

```
Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Alphabetical:
['andes', 'guam', 'himalaya', 'labrador', 'tierra del fuego']

Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Reverse alphabetical:
['tierra del fuego', 'labrador', 'himalaya', 'guam', 'andes']

Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Reversed:
['guam', 'labrador', 'tierra del fuego', 'andes', 'himalaya']

Original order:
['himalaya', 'andes', 'tierra del fuego', 'labrador', 'guam']

Alphabetical
['andes', 'guam', 'himalaya', 'labrador', 'tierra del fuego']

Reverse alphabetical
['tierra del fuego', 'labrador', 'himalaya', 'guam', 'andes']
```

第 4 章

练习 4-1　比萨

想出至少三种你喜欢的比萨，将其名称存储在一个列表中，再使用 `for` 循环将每种比萨的名称都打印出来。

- 修改这个 `for` 循环，使其打印包含比萨名称的句子，而不仅仅是比萨的名称。对于每种比萨，都显示一行输出，如 I like pepperoni pizza。
- 在程序末尾添加一行代码，它不在 `for` 循环中，指出你有多喜欢比萨。输出应包含针对每种比萨的消息，还有一个总结性句子，如 I really love pizza!。

```
favorite_pizzas = ['pepperoni', 'hawaiian', 'veggie']

# 打印所有比萨的名称。
for pizza in favorite_pizzas:
    print(pizza)

print("\n")

# 针对每种比萨打印一个句子。
for pizza in favorite_pizzas:
    print(f"I really love {pizza} pizza!")

print("\nI really love pizza!")
```

输出：

```
pepperoni
hawaiian
veggie



I really love pepperoni pizza!
I really love hawaiian pizza!
I really love veggie pizza!

I really love pizza!
```

练习 4-3　数到 20

使用一个 `for` 循环打印数 1~20（含）。

```
numbers = list(range(1, 21))

for number in numbers:
    print(number)
```

输出：

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
```

练习 4-5　一百万求和

创建一个包含数 1～1 000 000 的列表，再使用 `min()` 和 `max()` 核实该列表确实是从 1 开始、到 100000 结束的。另外，对这个列表调用函数 `sum()`，看看 Python 将一百万个数相加需要多长时间。

```
numbers = list(range(1, 1_000_001))
```

```
print(min(numbers))
print(max(numbers))
print(sum(numbers))
```

输出：

```
1
1000000
500000500000
```

练习 4-7　3 的倍数

创建一个列表，其中包含 3~30 能被 3 整除的数；再使用一个 `for` 循环将这个列表中的数都打印出来。

```
threes = list(range(3, 31, 3))

for number in threes:
    print(number)
```

输出：

```
3
6
9
12
15
18
21
24
27
30
```

练习 4-8　立方

将同一个数乘三次称为立方。例如，在 Python 中，2 的立方用 `2**3` 表示。请创建一个列表，其中包含前 10 个整数（即 1～10）的立方，再使用一个 `for` 循环将这些立方数都打印出来。

```
cubes = []
for number in range(1, 11):
    cube = number**3
    cubes.append(cube)

for cube in cubes:
    print(cube)
```

输出：

```
1
8
27
64
125
216
343
512
729
1000
```

练习 4-9　立方解析

使用列表解析生成一个列表，其中包含前 10 个整数的立方。

```
cubes = [number**3 for number in range(1,11)]

for cube in cubes:
    print(cube)
```

输出：

```
1
8
27
64
125
216
343
512
```

```
729
1000
```

练习 4-11　你的比萨，我的比萨

在你为完成练习 4-1 而编写的程序中，创建比萨列表的副本，并将其存储到变量 `friend_pizzas` 中，再完成如下任务：

- 在原来的比萨列表中添加一种比萨。
- 在列表 `friend_pizzas` 中添加另一种比萨。
- 核实你有两个不同的列表。为此，打印消息“My favorite pizzas are:”，再使用一个 `for` 循环打印第一个列表；打印消息“My friend’s favorite pizzas are:”，再使用一个 `for` 循环打印第二个列表。核实新增的比萨被添加到正确的列表中。

```
favorite_pizzas = ['pepperoni', 'hawaiian', 'veggie']
friend_pizzas = favorite_pizzas[:]

favorite_pizzas.append("meat lover's")
friend_pizzas.append('pesto')

print("My favorite pizzas are:")
for pizza in favorite_pizzas:
    print(f"- {pizza}")

print("\nMy friend's favorite pizzas are:")
for pizza in friend_pizzas:
    print(f"- {pizza}")
```

输出：

```
My favorite pizzas are:
- pepperoni
- hawaiian
- veggie
- meat lover's
```

```
My friend's favorite pizzas are:
- pepperoni
- hawaiian
- veggie
- pesto
```

练习 4-13　自助餐

有一家自助式餐馆，只提供五种简单的食品。请想出五种简单的食品，并将其存储在一个元组中。

- 使用一个 `for` 循环将该餐馆提供的五种食品都打印出来。
- 尝试修改其中的一个元素，核实 Python 确实会拒绝你这样做。
- 餐馆调整了菜单，替换了它原来提供的两种食品。请编写一个这样的代码块：给元组变量赋值，并使用一个 `for` 循环将新元组的每个元素都打印出来。

```
menu_items = (
    'rockfish sandwich', 'halibut nuggets', 'smoked salmon chowder',
    'salmon burger', 'crab cakes',
    )

print("You can choose from the following menu items:")
for item in menu_items:
    print(f"- {item}")

menu_items = (
    'rockfish sandwich', 'halibut nuggets', 'smoked salmon chowder',
    'black cod tips', 'king crab legs',
    )

print("\nOur menu has been updated.")
print("You can now choose from the following items:")
for item in menu_items:
    print(f"- {item}")
```

输出：

```
You can choose from the following menu items:
```

```
- rockfish sandwich
- halibut nuggets
- smoked salmon chowder
- salmon burger
- crab cakes

Our menu has been updated.
You can now choose from the following items:
- rockfish sandwich
- halibut nuggets
- smoked salmon chowder
- black cod tips
- king crab legs
```

第 5 章

练习 5-3　外星人颜色

假设在游戏中刚射杀了一个外星人，请创建一个名为 `alien_color` 的变量，并将其设置为 `'green'`、`'yellow'` 或 `'red'`。

- 编写一条 `if` 语句，检查外星人是否是绿色的；如果是，就打印一条消息，指出玩家获得了 5 分。
- 编写这个程序的两个版本，在一个版本中上述测试通过了，而在另一个版本中未通过（未通过测试时没有输出）。

能够通过测试的版本：

```
alien_color = 'green'

if alien_color == 'green':
    print("You just earned 5 points!")
```

输出：

```
You just earned 5 points!
```

不能通过测试的版本：

```
alien_color = 'red'

if alien_color == 'green':
    print("You just earned 5 points!")
```

（没有输出）

练习 5-4　外星人颜色 2

像练习 5-3 那样设置外星人的颜色，并编写一个 `if-else` 结构。

- 如果外星人是绿色的，就打印一条消息，指出玩家因射杀该外星人获得了 5 分。
- 如果外星人不是绿色的，就打印一条消息，指出玩家获得了 10 分。
- 编写这个程序的两个版本，在一个版本中执行 `if` 代码块，而在另一个版本中执行 `else` 代码块。

执行 `if` 代码块的版本：

```
alien_color = 'green'

if alien_color == 'green':
    print("You just earned 5 points!")
else:
    print("You just earned 10 points!")
```

输出：

```
You just earned 5 points!
```

执行 `else` 代码块的版本：

```
alien_color = 'yellow'

if alien_color == 'green':
    print("You just earned 5 points!")
else:
```

```
    print("You just earned 10 points!")
```

输出：

```
You just earned 10 points!
```

练习 5-5　外星人颜色 3

将练习 5-4 中的 `if-else` 结构改为 `if-elif-else` 结构。

- 如果外星人是绿色的，就打印一条消息，指出玩家获得了 5 分。
- 如果外星人是黄色的，就打印一条消息，指出玩家获得了 10 分。
- 如果外星人是红色的，就打印一条消息，指出玩家获得了 15 分。
- 编写这个程序的三个版本，它们分别在外星人为绿色、黄色和红色时打印一条消息。

外星人为红色的版本：

```
alien_color = 'red'

if alien_color == 'green':
    print("You just earned 5 points!")
elif alien_color == 'yellow':
    print("You just earned 10 points!")
else:
    print("You just earned 15 points!")
```

输出：

```
You just earned 15 points!
```

练习 5-6　人生的不同阶段

设置变量 `age` 的值，再编写一个 `if-elif-else` 结构，根据 `age` 的值判断处于人生的哪个阶段：

- 如果一个人的年龄小于 2 岁，就打印一条消息，指出他是婴儿。
- 如果一个人的年龄为 2（含）~4 岁，就打印一条消息，指出他是幼儿。

- 如果一个人的年龄为 4（含）~13 岁，就打印一条消息，指出他是儿童。
- 如果一个人的年龄为 13（含）~20 岁，就打印一条消息，指出他是少年。
- 如果一个人的年龄为 20（含）~65 岁，就打印一条消息，指出他是成年人。
- 如果一个人的年龄超过 65（含）岁，就打印一条消息，指出他是老年人。

```
age = 17

if age < 2:
    print("You're a baby!")
elif age < 4:
    print("You're a toddler!")
elif age < 13:
    print("You're a kid!")
elif age < 20:
    print("You're a teenager!")
elif age < 65:
    print("You're an adult!")
else:
    print("You're an elder!")
```

输出：

```
You're a teenager!
```

练习 5-7　喜欢的水果

创建一个列表，其中包含你喜欢的水果，再编写一系列独立的 `if` 语句，检查列表中是否包含特定的水果。

- 将该列表命名为 `favorite_fruits`，并在其中包含三种水果。
- 编写 5 条 `if` 语句，每条都检查某种水果是否包含在列表中，如果包含在列表中，就打印一条消息，如 You really like bananas!。

```
favorite_fruits = ['blueberries', 'salmonberries', 'peaches']

if 'bananas' in favorite_fruits:
    print("You really like bananas!")
```

```
if 'apples' in favorite_fruits:
    print("You really like apples!")
if 'blueberries' in favorite_fruits:
    print("You really like blueberries!")
if 'kiwis' in favorite_fruits:
    print("You really like kiwis!")
if 'peaches' in favorite_fruits:
    print("You really like peaches!")
```

输出：

```
You really like blueberries!
You really like peaches!
```

练习 5-8　以特殊方式跟管理员打招呼

创建一个至少包含 5 个用户名的列表，且其中一个用户名为`'admin'`。想象你要编写代码，在每位用户登录网站后都打印一条问候消息。遍历用户名列表，并向每位用户打印一条问候消息：

- 如果用户名为`'admin'`，就打印一条特殊的问候消息，如 Hello admin, would you like to see a status report?。
- 否则，打印一条普通的问候消息，如 Hello Eric, thank you for logging in again。

```
usernames = ['eric', 'willie', 'admin', 'erin', 'ever']

for username in usernames:
    if username == 'admin':
        print("Hello admin, would you like to see a status report?")
    else:
        print(f"Hello {username}, thank you for loggin in again!")
```

输出：

```
Hello eric, thank you for logging in again!
Hello willie, thank you for logging in again!
Hello admin, would you like to see a status report?
Hello erin, thank you for logging in again!
```

```
Hello ever, thank you for logging in again!
```

练习 5-9　处理没有用户的情形

在为完成练习 5-8 编写的程序中，添加一条 `if` 语句，检查用户名列表是否为空。

- 如果为空，就打印消息 We need to find some users!。
- 删除列表中的所有用户名，确定将打印正确的消息。

```
usernames = []

if usernames:
    for username in usernames:
        if username == 'admin':
            print("Hello admin, would you like to see a status report?")
        else:
            print(f"Hello {username}, thank you for loggin in again!")
else:
    print("We need to find some users!")
```

输出：

```
We need to find some users!
```

练习 5-10　检查用户名

按下面说的编写一个程序，模拟网站确保每位用户的用户名都独一无二的方式。

- 创建一个至少包含 5 个用户名的列表，并将其命名为 `current_users`。
- 再创建一个包含 5 个用户名的列表，将其命名为 `new_users`，并确保其中有一两个用户名也包含在列表 `current_users` 中。
- 遍历列表 `new_users`，对于其中的每个用户名，都检查它是否已被使用。如果是这样，就打印一条消息，指出需要输入别的用户名；否则，打印一条消息，指出这个用户名未被使用。

- 确保比较是不区分大消息；换句话说，如果用户名`'John'`已被使用，应拒绝用户名`'JOHN'`。

```
current_users = ['eric', 'willie', 'admin', 'erin', 'Ever']
new_users = ['sarah', 'Willie', 'PHIL', 'ever', 'Iona']

current_users_lower = [user.lower() for user in current_users]

for new_user in new_users:
    if new_user.lower() in current_users_lower:
        print(f"Sorry {new_user}, that name is taken.")
    else:
        print(f"Great, {new_user} is still available.")
```

输出：

```
Great, sarah is still available.
Sorry Willie, that name is taken.
Great, PHIL is still available.
Sorry ever, that name is taken.
Great, Iona is still available.
```

注意：如果你还不熟悉列表解析，可像下面这样使用循环来生成列表`current_users_lower`：

```
current_users_lower = []
for user in current_users:
    current_users_lower.append(user.lower())
```

练习 5-11　序数

序数表示位置，如 1st 和 2nd。大多数序数都以 th 结尾，只有 1、2 和 3 例外。

- 在一个列表中存储数字 1~9。
- 遍历这个列表。
- 在循环中使用一个`if-elif-else`结构，以打印每个数字对应的序数。输出内容

应为 `1st 2nd 3rd 4th 5th 6th 7th 8th 9th`，但每个序数都独占一行。

```
numbers = list(range(1,10))

for number in numbers:
    if number == 1:
        print("1st")
    elif number == 2:
        print("2nd")
    elif number == 3:
        print("3rd")
    else:
        print(f"{number}th")
```

输出：

```
1st
2nd
3rd
4th
5th
6th
7th
8th
9th
```

第 6 章

练习 6-1　人

使用一个字典来存储一个熟人的信息，包括名、姓、年龄和居住的城市。该字典应包含键 `first_name`、`last_name`、`age` 和 `city`。将存储在该字典中的每项信息都打印出来。

```
person = {
    'first_name': 'eric',
    'last_name': 'matthes',
    'age': 43,
    'city': 'sitka',
    }
```

```
print(person['first_name'])
print(person['last_name'])
print(person['age'])
print(person['city'])
```

输出：

```
eric
matthes
43
sitka
```

练习 6-2　喜欢的数

使用一个字典来存储一些人喜欢的数。请想出 5 个人的名字，并将这些名字用作字典中的键；想出每个人喜欢的一个数，并将这些数作为值存储在字典中。打印每个人的名字和喜欢的数。为让这个程序更有趣，通过询问朋友确保数据是真实的。

```
favorite_numbers = {
    'mandy': 42,
    'micah': 23,
    'gus': 7,
    'hank': 1000_000,
    'maggie': 0,
    }

num = favorite_numbers['mandy']
print(f"Mandy's favorite number is {num}.")

num = favorite_numbers['micah']
print(f"Micah's favorite number is {num}.")

num = favorite_numbers['gus']
print(f"Gus's favorite number is {num}.")

num = favorite_numbers['hank']
print(f"Hank's favorite number is {num}.")
```

```
num = favorite_numbers['maggie']
print(f"Maggie's favorite number is {num}.")
```

输出：

```
Mandy's favorite number is 42.
Micah's favorite number is 23.
Gus's favorite number is 7.
Hank's favorite number is 1000000.
Maggie's favorite number is 0.
```

练习 6-3　词汇表

Python 字典可用于模拟现实生活中的字典，但为避免混淆，我们将后者称为词汇表。

- 想出你在前面学过的 5 个编程词汇，将它们用作词汇表中的键，并将它们的含义作为值存储在词汇表中。
- 以整洁的方式打印每个词汇及其含义。为此，可以先打印词汇，在它后面加上一个冒号，再打印词汇的含义；也可在一行打印词汇，再使用换行符（`\n`）插入一个空行，然后在下一行以缩进的方式打印词汇的含义。

```
glossary = {
    'string': 'A series of characters.',
    'comment': 'A note in a program that the Python interpreter ignores.',
    'list': 'A collection of items in a particular order.',
    'loop': 'Work through a collection of items, one at a time.',
    'dictionary': "A collection of key-value pairs.",
    }

word = 'string'
print(f"\n{word.title()}: {glossary[word]}")

word = 'comment'
print(f"\n{word.title()}: {glossary[word]}")

word = 'list'
print(f"\n{word.title()}: {glossary[word]}")
```

```
word = 'loop'
print(f"\n{word.title()}: {glossary[word]}")

word = 'dictionary'
print(f"\n{word.title()}: {glossary[word]}")
```

输出：

```
String: A series of characters.

Comment: A note in a program that the Python interpreter ignores.

List: A collection of items in a particular order.

Loop: Work through a collection of items, one at a time.

Dictionary: A collection of key-value pairs.
```

练习 6-4　词汇表 2

现在你知道了如何遍历字典，请整理你为完成练习 6-3 而编写的代码，将其中的一系列函数调用 `print()` 替换为一个遍历字典中键和值的循环。确定该循环正确无误后，再在词汇表中添加 5 个 Python 术语。当你再次运行这个程序时，这些新术语及其含义将自动包含在输出中。

```
glossary = {
    'string': 'A series of characters.',
    'comment': 'A note in a program that the Python interpreter ignores.',
    'list': 'A collection of items in a particular order.',
    'loop': 'Work through a collection of items, one at a time.',
    'dictionary': "A collection of key-value pairs.",
    'key': 'The first item in a key-value pair in a dictionary.',
    'value': 'An item associated with a key in a dictionary.',
    'conditional test': 'A comparison between two values.',
    'float': 'A numerical value with a decimal component.',
    'boolean expression': 'An expression that evaluates to True or False.',
    }

for word, definition in glossary.items():
```

```
    print(f"\n{word.title()}: {definition}")
```

输出：

```
Dictionary: A collection of key-value pairs.

String: A series of characters.

Boolean Expression: An expression that evaluates to True or False.

Comment: A note in a program that the Python interpreter ignores.

Value: An item associated with a key in a dictionary.

Loop: Work through a collection of items, one at a time.

List: A collection of items in a particular order.

Conditional Test: A comparison between two values.

Key: The first item in a key-value pair in a dictionary.

Float: A numerical value with a decimal component.
```

练习 6-5　河流

创建一个字典，在其中存储三条大河流及其流经的国家。其中一个键值对可能是`'nile': 'egypt'`。

- 使用循环为每条河流打印一条消息，如 The Nile runs through Egypt。
- 使用循环将该字典中每条河流的名字都打印出来。
- 使用循环将该字典包含的每个国家的名字都打印出来。

```
rivers = {
    'nile': 'egypt',
    'mississippi': 'united states',
    'fraser': 'canada',
```

```
    'kuskokwim': 'alaska',
    'yangtze': 'china',
    }

for river, country in rivers.items():
    print(f"The {river.title()} flows through {country.title()}.")

print("\nThe following rivers are included in this data set:")
for river in rivers.keys():
    print(f"- {river.title()}")

print("\nThe following countries are included in this data set:")
for country in rivers.values():
    print(f"- {country.title()}")
```

输出：

```
The Mississippi flows through United States.
The Yangtze flows through China.
The Fraser flows through Canada.
The Nile flows through Egypt.
The Kuskokwim flows through Alaska.

The following rivers are included in this data set:
- Mississippi
- Yangtze
- Fraser
- Nile
- Kuskokwim

The following countries are included in this data set:
- United States
- China
- Canada
- Egypt
- Alaska
```

注意：有些人将阿拉斯加视为美国的一个附属国。

练习 6-6　调查

在 6.3.1 节编写的程序 favorite_languages.py 中执行以下操作。

- 创建一个应该会接受调查的人员名单，其中有些人已包含在字典中，而其他人未包含在字典中。
- 遍历这个人员名单，对于已参与调查的人，打印一条消息表示感谢。对于还未参与调查的人，打印一条消息邀请他参与调查。

```
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()}.")

print("\n")

coders = ['phil', 'josh', 'david', 'becca', 'sarah', 'matt', 'danielle']
for coder in coders:
    if coder in favorite_languages.keys():
        print(f"Thank you for taking the poll, {coder.title()}!")
    else:
        print(f"{coder.title()}, what's your favorite programming language?")
```

输出：

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


Thank you for taking the poll, Phil!
Josh, what's your favorite programming language?
David, what's your favorite programming language?
Becca, what's your favorite programming language?
```

```
Thank you for taking the poll, Sarah!
Matt, what's your favorite programming language?
Danielle, what's your favorite programming language?
```

练习 6-7　人们

在为完成练习 6-1 而编写的程序中，再创建两个表示人的字典，然后将这三个字典都存储在一个名为 `people` 的列表中。遍历这个列表，将其中每个人的所有信息都打印出来。

```
# 创建一个用于存储人的空列表。
people = []

# 定义一些人并将他们添加到前述列表中。
person = {
    'first_name': 'eric',
    'last_name': 'matthes',
    'age': 46,
    'city': 'sitka',
    }
people.append(person)

person = {
    'first_name': 'lemmy',
    'last_name': 'matthes',
    'age': 2,
    'city': 'sitka',
    }
people.append(person)

person = {
    'first_name': 'willie',
    'last_name': 'matthes',
    'age': 11,
    'city': 'sitka',
    }
people.append(person)
```

```
# 显示列表包含的每个字典中的信息。
for person in people:
    name = f"{person['first_name'].title()} {person['last_name'].title()}"
    age = person['age']
    city = person['city'].title()

    print(f"{name}, of {city}, is {age} years old.")
```

输出：

```
Eric Matthes, of Sitka, is 46 years old.
Lemmy Matthes, of Sitka, is 2 years old.
Willie Matthes, of Sitka, is 11 years old.
```

练习 6-8　宠物

创建多个字典，对于每个字典，都使用一个宠物的名称来给它命名；在每个字典中，包含宠物的类型及其主人的名字。将这些字典存储在一个名为 `pets` 的列表中，再遍历该列表，并将有关每个宠物的所有信息都打印出来。

注意：提供练习答案时，我才发现这个问题表述得不太完美。实际上，使用宠物的名称给描述它的字典命名不合理，而应将宠物的名称包含在字典。下面的练习答案就是按这里说的做的。

```
# 创建一个用于存储宠物的空列表。
pets = []

# 定义各个宠物并将其存储到列表中。
pet = {
    'animal type': 'python',
    'name': 'john',
    'owner': 'guido',
    'weight': 43,
    'eats': 'bugs',
}
pets.append(pet)

pet = {
```

```
    'animal type': 'chicken',
    'name': 'clarence',
    'owner': 'tiffany',
    'weight': 2,
    'eats': 'seeds',
}
pets.append(pet)

pet = {
    'animal type': 'dog',
    'name': 'peso',
    'owner': 'eric',
    'weight': 37,
    'eats': 'shoes',
}
pets.append(pet)

# 显示每个宠物的信息。
for pet in pets:
    print(f"\nHere's what I know about {pet['name'].title()}:")
    for key, value in pet.items():
        print(f"\t{key}: {value}")
```

输出：

```
Here's what I know about John:
    animal type: python
    name: john
    owner: guido
    weight: 43
    eats: rats

Here's what I know about Clarence:
    animal type: chicken
    name: clarence
    owner: tiffany
    weight: 2
    eats: seeds
```

```
Here's what I know about Peso:
    animal type: dog
    name: peso
    owner: eric
    weight: 37
    eats: shoes
```

练习 6-9　喜欢的地方

创建一个名为 `favorite_places` 的字典。在这个字典中，将三个人的名字用作键；对于其中的每个人，都存储他喜欢的 1~3 个地方。为让这个练习更有趣些，让一些朋友指出他们喜欢的几个地方。遍历这个字典，并将其中每个人的名字及其喜欢的地方打印出来。

```
favorite_places = {
    'eric': ['bear mountain', 'death valley', 'tierra del fuego'],
    'erin': ['hawaii', 'iceland'],
    'willie': ['mt. verstovia', 'the playground', 'new hampshire']
    }

for name, places in favorite_places.items():
    print(f"\n{name.title()} likes the following places:")
    for place in places:
        print(f"- {place.title()}")
```

输出：

```
Eric likes the following places:
- Bear Mountain
- Death Valley
- Tierra Del Fuego

Erin likes the following places:
- Hawaii
- Iceland

Willie likes the following places:
- Mt. Verstovia
- The Playground
```

```
    - New Hampshire
```

练习 6-10　喜欢的数 2

修改为完成练习 6-2 而编写的程序，让每个人都可以有多个喜欢的数，然后将每个人的名字及其喜欢的数打印出来。

```
favorite_numbers = {
    'mandy': [42, 17],
    'micah': [42, 39, 56],
    'gus': [7, 12],
    }

for name, numbers in favorite_numbers.items():
    print(f"\n{name.title()} likes the following numbers:")
    for number in numbers:
        print(f"  {number}")
```

输出：

```
Micah likes the following numbers:
  42
  39
  56

Mandy likes the following numbers:
  42
  17

Gus likes the following numbers:
  7
  12
```

练习 6-11　城市

创建一个名为 `cities` 的字典，在其中将三个城市名用作键；对于每座城市，都创建一个字典，并在其中包含该城市所属的国家、人口约数以及一个有关该城市的事实。在表

示每座城市的字典中，应包含 `country`、`population` 和 `fact` 等键。将每座城市的名字以及有关它们的信息都打印出来。

```
cities = {
    'santiago': {
        'country': 'chile',
        'population': 6_310_000,
        'nearby mountains': 'andes',
        },
    'talkeetna': {
        'country': 'united states',
        'population': 876,
        'nearby mountains': 'alaska range',
        },
    'kathmandu': {
        'country': 'nepal',
        'population': 975_453,
        'nearby mountains': 'himilaya',
        }
    }

for city, city_info in cities.items():
    country = city_info['country'].title()
    population = city_info['population']
    mountains = city_info['nearby mountains'].title()

    print(f"\n{city.title()} is in {country}.")
    print(f"  It has a population of about {population}.")
    print(f"  The {mountains} mounats are nearby.")
```

输出：

```
Santiago is in Chile.
  It has a population of about 6310000.
  The Andes mounats are nearby.

Talkeetna is in United States.
  It has a population of about 876.
```

```
  The Alaska Range mounats are nearby.

Kathmandu is in Nepal.
  It has a population of about 975453.
  The Himilaya mounats are nearby.
```

第 7 章

注意：Sublime Text不能运行提示用户输入的程序。你可使用Sublime Text来编写提示用户输入的程序，但必须从终端运行它们。详情请参阅1.4节。

练习 7-1　汽车租赁

编写一个程序，询问用户要租赁什么样的汽车，并打印一条消息，如“Let me see if I can find you a Subaru.”。

```
car = input("What kind of car would you like? ")

print(f"Let me see if I can find you a {car.title()}.")
```

输出：

```
What kind of car would you like? Toyota Tacoma
Let me see if I can find you a Toyota Tacoma.
```

练习 7-2　餐馆订位

编写一个程序，询问用户有多少人用餐。如果超过 8 位，就打印一条消息，指出没有空桌；否则就指出有空桌。

```
party_size = input("How many people are in your dinner party tonight? ")
party_size = int(party_size)

if party_size > 8:
    print("I'm sorry, you'll have to wait for a table.")
else:
    print("Your table is ready.")
```

输出：

```
How many people are in your dinner party tonight? 12
I'm sorry, you'll have to wait for a table.
```

或：

```
How many people are in your dinner party tonight? 6
Your table is ready.
```

练习 7-3　10 的整数倍

让用户输入一个数，并指出这个数是否是 10 的整数倍。

```
number = input("Give me a number, please: ")
number = int(number)

if number % 10 == 0:
    print(f"{number} is a multiple of 10.")
else:
    print(f"{number} is not a multiple of 10.")
```

输出：

```
Give me a number, please: 23
23 is not a multiple of 10.
```

或：

```
Give me a number, please: 90
90 is a multiple of 10.
```

练习 7-4　比萨配料

编写一个循环，提示用户输入一系列的比萨配料，并在用户输入'quit'时结束循环。每当用户输入一种配料后，都打印一条消息，说我们会在比萨中添加这种配料。

```
prompt = "\nWhat topping would you like on your pizza?"
prompt += "\nEnter 'quit' when you are finished: "

while True:
    topping = input(prompt)
    if topping != 'quit':
        print(f"  I'll add {topping} to your pizza.")
    else:
        break
```

输出：

```
What topping would you like on your pizza?
Enter 'quit' when you are finished: pepperoni
  I'll add pepperoni to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: sausage
  I'll add sausage to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: bacon
  I'll add bacon to your pizza.

What topping would you like on your pizza?
Enter 'quit' when you are finished: quit
```

练习 7-5　电影票

有家电影院根据观众的年龄收取不同的票价：不到 3 岁的观众免费；3~12 岁的观众为 10 美元；超过 12 岁的观众为 15 美元。请编写一个循环，在其中询问用户的年龄，并指出其票价。

```
prompt = "How old are you?"
prompt += "\nEnter 'quit' when you are finished. "

while True:
    age = input(prompt)
    if age == 'quit':
        break
    age = int(age)

    if age < 3:
        print("  You get in free!")
    elif age < 13:
        print("  Your ticket is $10.")
    else:
        print("  Your ticket is $15.")
```

输出：

```
How old are you?
Enter 'quit' when you are finished. 2
  You get in free!
How old are you?
Enter 'quit' when you are finished. 3
  Your ticket is $10.
How old are you?
Enter 'quit' when you are finished. 12
  Your ticket is $10.
How old are you?
Enter 'quit' when you are finished. 18
  Your ticket is $15.
How old are you?
Enter 'quit' when you are finished. quit
```

练习 7-8　熟食店

创建一个名为 `sandwich_orders` 的列表，在其中包含各种三明治的名字；再创建一个名为 `finished_sandwiches` 的空列表。遍历列表 `sandwich_orders`，对于其中的每种三明治，都打印一条消息，如 `I made your tuna sandwich`，并将其移到列表 `finished_sandwiches`。所有三明治都制作好后，打印一条消息，将这些三明治列出来。

```
sandwich_orders = ['veggie', 'grilled cheese', 'turkey', 'roast beef']
finished_sandwiches = []

while sandwich_orders:
    current_sandwich = sandwich_orders.pop()
    print(f"I'm working on your {current_sandwich} sandwich.")
    finished_sandwiches.append(current_sandwich)

print("\n")
for sandwich in finished_sandwiches:
    print(f"I made a {sandwich} sandwich.")
```

输出：

```
I'm working on your roast beef sandwich.
I'm working on your turkey sandwich.
I'm working on your grilled cheese sandwich.
I'm working on your veggie sandwich.

I made a roast beef sandwich.
I made a turkey sandwich.
I made a grilled cheese sandwich.
I made a veggie sandwich.
```

练习 7-9　五香烟熏牛肉卖完了

使用为完成练习 7-8 而创建的列表 `sandwich_orders`，并确保 `'pastrami'` 在其中至少出现了三次。在程序开头附近添加这样的代码：打印一条消息，指出熟食店的五香烟熏牛肉（pastrami）卖完了；再使用一个 `while` 循环将列表 `sandwich_orders` 中的 `'pastrami'` 都删除。确认最终的列表 `finished_sandiches` 中未包含 `'pastrami'`。

```
sandwich_orders = [
    'pastrami', 'veggie', 'grilled cheese', 'pastrami',
    'turkey', 'roast beef', 'pastrami']
finished_sandwiches = []

print("I'm sorry, we're all out of pastrami today.")
while 'pastrami' in sandwich_orders:
    sandwich_orders.remove('pastrami')

print("\n")
while sandwich_orders:
    current_sandwich = sandwich_orders.pop()
    print(f"I'm working on your {current_sandwich} sandwich.")
    finished_sandwiches.append(current_sandwich)

print("\n")
```

```
for sandwich in finished_sandwiches:
    print(f"I made a {sandwich} sandwich.")
```

输出：

```
I'm sorry, we're all out of pastrami today.

I'm working on your roast beef sandwich.
I'm working on your turkey sandwich.
I'm working on your grilled cheese sandwich.
I'm working on your veggie sandwich.

I made a roast beef sandwich.
I made a turkey sandwich.
I made a grilled cheese sandwich.
I made a veggie sandwich.
```

练习 7-10　梦想的度假胜地

编写一个程序，调查用户梦想的度假胜地。使用类似于“If you could visit one place in the world, where would you go?”的提示，并编写一个打印调查结果的代码块。

```
name_prompt = "\nWhat's your name? "
place_prompt = "If you could visit one place in the world, where would it be? "
continue_prompt = "\nWould you like to let someone else respond? (yes/no) "

# 调查结果将存储在形如{name: place}的字典中。
responses = {}

while True:
    # 询问用户想去哪里度假。
    name = input(name_prompt)
    place = input(place_prompt)

    # 存储调查结果。
    responses[name] = place

    # 询问是否还有其他人要参与调查。
    repeat = input(continue_prompt)
```

```
        if repeat != 'yes':
            break

    # 显示调查结果。
    print("\n--- Results ---")
    for name, place in responses.items():
        print(f"{name.title()} would like to visit {place.title()}.")
```

输出：

```
What's your name? eric
If you could visit one place in the world, where would it be? china

Would you like to let someone else respond? (yes/no) yes

What's your name? erin
If you could visit one place in the world, where would it be? iceland

Would you like to let someone else respond? (yes/no) yes

What's your name? ever
If you could visit one place in the world, where would it be? japan

Would you like to let someone else respond? (yes/no)

--- Results ---
Eric would like to visit China.
Erin would like to visit Iceland.
Ever would like to visit Japan.
```

第 8 章

练习 8-1　消息

编写一个名为 `display_message()` 的函数，它打印一个句子，指出你在本章学的是什么。调用这个函数，确认显示的消息正确无误。

```
def display_message():
    """显示一条消息，指出你在本章学的是什么。"""
```

```
    msg = "I'm learning to store code in functions."
    print(msg)

display_message()
```

输出：

```
I'm learning to store code in functions.
```

练习 8-2　喜欢的图书

编写一个名为 `favorite_book()` 的函数，其中包含一个名为 `title` 的形参。这个函数打印一条消息，如 One of my favorite books is Alice in Wonderland。调用这个函数，并将一本图书的名称作为实参传递给它。

```
def favorite_book(title):
    """显示一条消息，指出喜欢的一本图书。"""
    print(f"{title} is one of my favorite books.")

favorite_book('The Abstract Wild')
```

输出：

```
The Abstract Wild is one of my favorite books.
```

练习 8-3　T 恤

编写一个名为 `make_shirt()` 的函数，它接受尺码以及要印到 T 恤上的字样。这个函数应打印一个句子，概要地说明 T 恤衫的尺码和字样。

使用位置实参调用这个函数来制作一件 T 恤，再使用关键字实参来调用这个函数。

```
def make_shirt(size, message):
    """概述要制作的 T 恤什么样。"""
    print(f"\nI'm going to make a {size} t-shirt.")
    print(f'It will say, "{message}"')

make_shirt('large', 'I love Python!')
```

```
make_shirt(message="Readability counts.", size='medium')
```

输出：

```
I'm going to make a large t-shirt.
It will say, "I love Python!"

I'm going to make a medium t-shirt.
It will say, "Readability counts."
```

练习 8-4　大号 T 恤

修改函数 `make_shirt()`，使其在默认情况下制作一件印有 “I love Python” 字样的大号 T 恤。调用这个函数来制作如下 T 恤：一件印有默认字样的大号 T 恤、一件印有默认字样的中号 T 恤、一件印有其他字样的 T 恤（尺码无关紧要）。

```
def make_shirt(size='large', message='I love Python!'):
    """概述要制作的 T 恤什么样。"""
    print(f"\nI'm going to make a {size} t-shirt.")
    print(f'It will say, "{message}"')

make_shirt()
make_shirt(size='medium')
make_shirt('small', 'Programmers are loopy.')
```

输出：

```
I'm going to make a large t-shirt.
It will say, "I love Python!"

I'm going to make a medium t-shirt.
It will say, "I love Python!"

I'm going to make a small t-shirt.
It will say, "Programmers are loopy."
```

练习 8-5　城市

编写一个名为 `describe_city()` 的函数，它接受一座城市的名称和所属的国家。这个函数应打印一个简单的句子，如 Reykjavik is in Iceland。给用于存储国家的形参指定默认值。为三座不同的城市调用这个函数，且其中至少有一座城市不属于默认国家。

```
def describe_city(city, country='chile'):
    """描述城市。"""
    msg = f"{city.title()} is in {country.title()}."
    print(msg)

describe_city('santiago')
describe_city('reykjavik', 'iceland')
describe_city('punta arenas')
```

输出：

```
Santiago is in Chile.
Reykjavik is in Iceland.
Punta Arenas is in Chile.
```

练习 8-6　城市名

编写一个名为 `city_country()` 的函数，它接受城市的名称和所属的国家。这个函数应返回一个格式类似于下面的字符串：

```
"Santiago, Chile"
```

至少使用三个城市-国家对调用这个函数，并打印返回的值。

```
def city_country(city, country):
    """返回一个类似于'Santiago, Chile'的字符串。"""
    return f"{city.title()}, {country.title()}"

city = city_country('santiago', 'chile')
print(city)

city = city_country('ushuaia', 'argentina')
```

```
    print(city)

    city = city_country('longyearbyen', 'svalbard')
    print(city)
```

输出：

```
Santiago, Chile
Ushuaia, Argentina
Longyearbyen, Svalbard
```

练习 8-7　专辑

编写一个名为 `make_album()` 的函数，它创建一个描述音乐专辑的字典。这个函数应接受歌手的名字和专辑名，并返回一个包含这两项信息的字典。使用这个函数创建三个表示不同专辑的字典，并打印每个返回的值，以核实字典正确地存储了专辑的信息。

给函数 `make_album()` 添加一个可选形参，以便能够存储专辑包含的歌曲数。如果调用这个函数时指定了歌曲数，就将这个值添加到表示专辑的字典中。调用这个函数，并至少在一次调用中指定专辑包含的歌曲数。

简单版：

```
def make_album(artist, title):
    """创建一个包含专辑信息的字典。"""
    album_dict = {
        'artist': artist.title(),
        'title': title.title(),
        }
    return album_dict

album = make_album('metallica', 'ride the lightning')
print(album)

album = make_album('beethoven', 'ninth symphony')
print(album)

album = make_album('willie nelson', 'red-headed stranger')
```

```
print(album)
```

输出：

```
{'title': 'Ride The Lightning', 'artist': 'Metallica'}
{'title': 'Ninth Symphony', 'artist': 'Beethoven'}
{'title': 'Red-Headed Stranger', 'artist': 'Willie Nelson'}
```

包含歌曲数的版本：

```
def make_album(artist, title, tracks=0):
    """创建一个包含专辑信息的字典。"""
    album_dict = {
        'artist': artist.title(),
        'title': title.title(),
        }
    if tracks:
        album_dict['tracks'] = tracks
    return album_dict

album = make_album('metallica', 'ride the lightning')
print(album)

album = make_album('beethoven', 'ninth symphony')
print(album)

album = make_album('willie nelson', 'red-headed stranger')
print(album)

album = make_album('iron maiden', 'piece of mind', tracks=8)
print(album)
```

输出：

```
{'artist': 'Metallica', 'title': 'Ride The Lightning'}
{'artist': 'Beethoven', 'title': 'Ninth Symphony'}
{'artist': 'Willie Nelson', 'title': 'Red-Headed Stranger'}
{'tracks': 8, 'artist': 'Iron Maiden', 'title': 'Piece Of Mind'}
```

练习 8-8　用户的专辑

在为完成练习 8-7 编写的程序中，编写一个 `while` 循环，让用户输入一个专辑的歌手和名称。获取这些信息后，使用它们来调用函数 `make_album()` 并将创建的字典打印出来。在这个 `while` 循环中，务必提供退出途径。

```
def make_album(artist, title, tracks=0):
    """创建一个包含专辑信息的字典。"""
    album_dict = {
        'artist': artist.title(),
        'title': title.title(),
        }
    if tracks:
        album_dict['tracks'] = tracks
    return album_dict

# 生成提示语。
title_prompt = "\nWhat album are you thinking of? "
artist_prompt = "Who's the artist? "

# 让用户知道如何退出。
print("Enter 'quit' at any time to stop.")

while True:
    title = input(title_prompt)
    if title == 'quit':
        break

    artist = input(artist_prompt)
    if artist == 'quit':
        break

    album = make_album(artist, title)
    print(album)

print("\nThanks for responding!")
```

输出：

```
Enter 'quit' at any time to stop.

What album are you thinking of? number of the beast
Who's the artist? iron maiden
{'artist': 'Iron Maiden', 'title': 'Number Of The Beast'}

What album are you thinking of? touch of class
Who's the artist? angel romero
{'artist': 'Angel Romero', 'title': 'Touch Of Class'}

What album are you thinking of? rust in peace
Who's the artist? megadeth
{'artist': 'Megadeth', 'title': 'Rust In Peace'}

What album are you thinking of? quit

Thanks for responding!
```

练习 8-9　消息

创建一个列表，其中包含一系列简短的文本消息；将这个列表传递给一个名为`show_messages()`的函数，它打印列表中的每条文本消息。

```
def show_messages(messages):
    """打印列表中的所有消息"""
    for message in messages:
        print(message)

messages = ["hello there", "how are u?", ":)"]
show_messages(messages)
```

输出：

```
hello there
how are u?
:)
```

练习 8-10　发送消息

在你为完成练习 8-9 而编写的程序中，编写一个名为 `send_messages()` 的函数，将每条消息都打印出来并移到一个名为 `sent_messages` 的列表中。调用函数 `send_messages()`，再将两个列表都打印出来，确认正确地移动了消息。

```
def show_messages(messages):
    """打印列表中的所有消息。"""
    print("Showing all messages:")
    for message in messages:
        print(message)

def send_messages(messages, sent_messages):
    """打印每条消息，再将其移到列表 sent_messages 中。"""
    print("\nSending all messages:")
    while messages:
        current_message = messages.pop()
        print(current_message)
        sent_messages.append(current_message)

messages = ["hello there", "how are u?", ":)"]
show_messages(messages)

sent_messages = []
send_messages(messages, sent_messages)

print("\nFinal lists:")
print(messages)
print(sent_messages)
```

输出：

```
Showing all messages:
hello there
how are u?
:)

Sending all messages:
```

```
:)
how are u?
hello there

Final lists:
[]
[':)', 'how are u?', 'hello there']
```

练习 8-11　消息归档

修改你为完成练习 8-10 而编写的程序，在调用函数 `send_messages()` 时，向它传递消息列表的副本。调用函数 `send_messages()` 后，将两个列表都打印出来，确认保留了原始列表中的消息。

```
def show_messages(messages):
    """打印列表中的所有消息。"""
    print("Showing all messages:")
    for message in messages:
        print(message)

def send_messages(messages, sent_messages):
    """打印每条消息，再将其移到列表 sent_messages 中。"""
    print("\nSending all messages:")
    while messages:
        current_message = messages.pop()
        print(current_message)
        sent_messages.append(current_message)

messages = ["hello there", "how are u?", ":)"]
show_messages(messages)

sent_messages = []
send_messages(messages[:], sent_messages)

print("\nFinal lists:")
print(messages)
```

```
print(sent_messages)
```

输出：

```
Showing all messages:
hello there
how are u?
:)

Sending all messages:
:)
how are u?
hello there

Final lists:
['hello there', 'how are u?', ':)']
[':)', 'how are u?', 'hello there']
```

练习 8-12　三明治

编写一个函数，它接受顾客要在三明治中添加的一系列食材。这个函数只有一个形参（它收集函数调用中提供的所有食材），并打印一条消息，对顾客点的三明治进行概述。调用这个函数三次，每次都提供不同数量的实参。

```
def make_sandwich(*items):
    """使用指定的食材制作三明治。"""
    print("\nI'll make you a great sandwich:")
    for item in items:
        print(f"  ...adding {item} to your sandwich.")
    print("Your sandwich is ready!")

make_sandwich('roast beef', 'cheddar cheese', 'lettuce', 'honey dijon')
make_sandwich('turkey', 'apple slices', 'honey mustard')
make_sandwich('peanut butter', 'strawberry jam')
```

输出：

```
I'll make you a great sandwich:
  ...adding roast beef to your sandwich.
  ...adding cheddar cheese to your sandwich.
```

```
  ...adding lettuce to your sandwich.
  ...adding honey dijon to your sandwich.
Your sandwich is ready!

I'll make you a great sandwich:
  ...adding turkey to your sandwich.
  ...adding apple slices to your sandwich.
  ...adding honey mustard to your sandwich.
Your sandwich is ready!

I'll make you a great sandwich:
  ...adding peanut butter to your sandwich.
  ...adding strawberry jam to your sandwich.
Your sandwich is ready!
```

练习 8-14　汽车

编写一个函数，将一辆汽车的信息存储在字典中。这个函数总是接受制造商和型号，还接受任意数量的关键字实参。这样调用这个函数：提供必不可少的信息，还有两个名称-值对，如颜色和选装配件。这个函数必须能够像下面这样进行调用：

```
car = make_car('subaru', 'outback', color='blue', tow_package=True)
```

打印返回的字典，确认正确地处理了所有的信息。

```
def make_car(manufacturer, model, **options):
    """创建一个表示汽车的字典。"""
    car_dict = {
        'manufacturer': manufacturer.title(),
        'model': model.title(),
        }
    for option, value in options.items():
        car_dict[option] = value

    return car_dict

my_outback = make_car('subaru', 'outback', color='blue', tow_package=True)
print(my_outback)
```

```
    my_old_accord = make_car('honda', 'accord', year=1991, color='white',
            headlights='popup')
    print(my_old_accord)
```

输出：

```
    {'manufacturer': 'Subaru', 'model': 'Outback', 'color': 'blue', 'tow_package':
True}
    {'manufacturer': 'Honda', 'model': 'Accord', 'year': 1991, 'color': 'white',
'headlights': 'popup'}
```

练习 8-15　打印模型

将示例 printing_models.py 中的函数放在另一个名为 printing_functions.py 的文件中。在 printing_models.py 的开头编写一条 `import` 语句，并修改这个文件以使用导入的函数。

```
    """与打印 3D 模型相关的函数。"""

    def print_models(unprinted_designs, completed_models):
        """
        模拟打印每个设计，直到没有未打印的设计为止。
       打印每个设计后，都将其移到列表 completed_models 中。
        """
        while unprinted_designs:
            current_design = unprinted_designs.pop()

            # 模拟根据设计制作 3D 打印模型的过程。
            print(f"Printing model: {current_design}")
            completed_models.append(current_design)

    def show_completed_models(completed_models):
        """显示打印好的所有模型。"""
        print("\nThe following models have been printed:")
        for completed_model in completed_models:
            print(completed_model)
```

printing_models.py:

```
import printing_functions as pf

unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']
completed_models = []

pf.print_models(unprinted_designs, completed_models)
pf.show_completed_models(completed_models)
```

输出：

```
Printing model: dodecahedron
Printing model: robot pendant
Printing model: iphone case

The following models have been printed:
dodecahedron
robot pendant
iphone case
```

第 9 章

练习 9-1　餐馆

创建一个名为 `Restaurant` 的类，其方法 `__init__()` 设置两个属性：`restaurant_name` 和 `cuisine_type`。创建一个名为 `describe_restaurant()` 的方法和一个名为 `open_restaurant()` 的方法，其中前者打印前述两项信息，而后者打印一条消息，指出餐馆正在营业。

根据这个类创建一个名为 `restaurant` 的实例，分别打印其两个属性，再调用前述两个方法。

```
class Restaurant():
    """一个表示餐馆的类。"""
```

```
    def __init__(self, name, cuisine_type):
        """初始化餐馆。"""
        self.name = name.title()
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        """显示餐馆信息摘要。"""
        msg = f"{self.name} serves wonderful {self.cuisine_type}."
        print(f"\n{msg}")

    def open_restaurant(self):
        """显示一条消息，指出餐馆正在营业。"""
        msg = f"{self.name} is open. Come on in!"
        print(f"\n{msg}")

restaurant = Restaurant('the mean queen', 'pizza')
print(restaurant.name)
print(restaurant.cuisine_type)

restaurant.describe_restaurant()
restaurant.open_restaurant()
```

输出：

```
The Mean Queen
pizza

The Mean Queen serves wonderful pizza.

The Mean Queen is open. Come on in!
```

练习 9-2　三家餐馆

根据你为完成练习 9-1 而编写类创建三个实例，并对每个实例调用方法 `describe_restaurant()`。

```
class Restaurant():
    """一个表示餐馆的类。"""
```

```
    def __init__(self, name, cuisine_type):
        """初始化餐馆。"""
        self.name = name.title()
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        """显示餐馆信息摘要概述。"""
        msg = f"{self.name} serves wonderful {self.cuisine_type}."
        print(f"\n{msg}")

    def open_restaurant(self):
        """显示一条消息，指出餐馆正在营业。"""
        msg = f"{self.name} is open. Come on in!"
        print(f"\n{msg}")

mean_queen = Restaurant('the mean queen', 'pizza')
mean_queen.describe_restaurant()

ludvigs = Restaurant("ludvig's bistro", 'seafood')
ludvigs.describe_restaurant()

mango_thai = Restaurant('mango thai', 'thai food')
mango_thai.describe_restaurant()
```

输出：

```
The Mean Queen serves wonderful pizza.

Ludvig'S Bistro serves wonderful seafood.

Mango Thai serves wonderful thai food.
```

练习 9-3　用户

创建一个名为 `User` 的类，其中包含属性 `first_name` 和 `last_name`，还有用户简介通常会存储的其他几个属性。定义一个名为 `describe_user()` 的方法，它打印用户信息摘

要：再定义一个名为 `greet_user()` 的方法，它向用户发出个性化的问候。

创建多个表示不同用户的实例，并对每个实例都调用上述两个方法。

```
class User():
    """一个表示用户的简单类。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化用户。"""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()

    def describe_user(self):
        """显示用户信息摘要。"""
        print(f"\n{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """向用户发出个性化的问候。"""
        print(f"\nWelcome back, {self.username}!")

eric = User('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()
eric.greet_user()

willie = User('willie', 'burger', 'willieburger', 'wb@example.com', 'alaska')
willie.describe_user()
willie.greet_user()
```

输出：

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
```

```
  Location: Alaska

Welcome back, e_matthes!

Willie Burger
  Username: willieburger
  Email: wb@example.com
  Location: Alaska

Welcome back, willieburger!
```

练习 9-4　就餐人数

在为完成练习 9-1 而编写的程序中，添加一个名为 `number_served` 的属性，并将其默认值设置为 0。根据这个类创建一个名为 `restaurant` 的实例；打印有多少人在这家餐馆就餐过，然后修改这个值并再次打印它。

添加一个名为 `set_number_served()` 的方法，它让你能够设置就餐人数。调用这个方法并向它传递一个值，然后再次打印这个值。

添加一个名为 `increment_number_served()` 的方法，它让你能够增加就餐人数。调用这个方法并向它传递一个这样的值：你认为这家餐馆每天可能接待的就餐人数。

```
class Restaurant():
    """一个表示餐馆的类。"""

    def __init__(self, name, cuisine_type):
        """初始化餐馆。"""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """显示餐馆信息摘要。"""
        msg = f"{self.name} serves wonderful {self.cuisine_type}."
```

```
            print(f"\n{msg}")

        def open_restaurant(self):
            """显示一条消息，指出餐馆正在营业。"""
            msg = f"{self.name} is open. Come on in!"
            print(f"\n{msg}")

        def set_number_served(self, number_served):
            """让用户能够设置就餐人数。"""
            self.number_served = number_served

        def increment_number_served(self, additional_served):
            """让用户能够增加就餐人数。"""
            self.number_served += additional_served


    restaurant = Restaurant('the mean queen', 'pizza')
    restaurant.describe_restaurant()

    print(f"\nNumber served: {restaurant.number_served}")
    restaurant.number_served = 430
    print(f"Number served: {restaurant.number_served}")

    restaurant.set_number_served(1257)
    print(f"Number served: {restaurant.number_served}")

    restaurant.increment_number_served(239)
    print(f"Number served: {restaurant.number_served}")
```

输出：

```
The Mean Queen serves wonderful pizza.

Number served: 0
Number served: 430
Number served: 1257
Number served: 1496
```

练习 9-5　尝试登录次数

在为完成练习 9-3 而编写的 `User` 类中，添加一个名为 `login_attempts` 的属性。编写一个名为 `increment_login_attempts()` 的方法，它将属性 `login_attempts` 的值加 1。再编写一个名为 `reset_login_attempts()` 的方法，它将属性 `login_attempts` 的值重置为 0。

根据 `User` 类创建一个实例，再调用方法 `increment_login_attempts()` 多次。打印属性 `login_attempts` 的值，确认它被正确地递增；然后，调用方法 `reset_login_attempts()`，并再次打印属性 `login_attempts` 的值，确认它被重置为 0。

```
class User():
    """一个表示用户的简单类。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化用户."""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """显示用户信息摘要。"""
        print(f"\n{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """向用户发出个性化问候。"""
        print(f"\nWelcome back, {self.username}!")
```

```
    def increment_login_attempts(self):
        """将属性 login_attempts 的值加 1。"""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """将 login_attempts 重置为 0。"""
        self.login_attempts = 0

eric = User('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()
eric.greet_user()

print("\nMaking 3 login attempts...")
eric.increment_login_attempts()
eric.increment_login_attempts()
eric.increment_login_attempts()
print(f"  Login attempts: {eric.login_attempts}")

print("Resetting login attempts...")
eric.reset_login_attempts()
print(f"  Login attempts: {eric.login_attempts}")
```

输出：

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Welcome back, e_matthes!

Making 3 login attempts...
  Login attempts: 3
Resetting login attempts...
  Login attempts: 0
```

练习 9-6　冰激凌小店

冰激凌小店是一种特殊的餐馆。编写一个名为 `IceCreamStand` 的类，让它继承你为完成练习 9-1 或 9-4 而编写的 `Restaurant` 类。这两个版本的 `Restaurant` 类都可以，挑选你更喜欢的那个即可。添加一个名为 `flavors` 的属性，用于存储一个由各种口味的冰激凌组成的列表。编写一个显示这些冰激凌的方法。创建一个 `IceCreamStand` 实例，并调用这个方法。

```
class Restaurant():
    """一个表示餐馆的类。"""

    def __init__(self, name, cuisine_type):
        """初始化餐馆。"""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0

    def describe_restaurant(self):
        """显示餐馆信息摘要。"""
        msg = f"{self.name} serves wonderful {self.cuisine_type}."
        print(f"\n{msg}")

    def open_restaurant(self):
        """显示一条消息，指出餐馆正在营业。"""
        msg = f"{self.name} is open. Come on in!"
        print(f"\n{msg}")

    def set_number_served(self, number_served):
        """让用户能够设置就餐人数。"""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
        """让用户能够增加就餐人数。"""
        self.number_served += additional_served
```

```
class IceCreamStand(Restaurant):
    """一个表示冰激凌小店的类。"""

    def __init__(self, name, cuisine_type='ice_cream'):
        """初始化冰激凌小店。"""
        super().__init__(name, cuisine_type)
        self.flavors = []

    def show_flavors(self):
        """显示出售的冰激凌品种。"""
        print("\nWe have the following flavors available:")
        for flavor in self.flavors:
            print(f"- {flavor.title()}")


big_one = IceCreamStand('The Big One')
big_one.flavors = ['vanilla', 'chocolate', 'black cherry']

big_one.describe_restaurant()
big_one.show_flavors()
```

输出：

```
The Big One serves wonderful ice_cream.

We have the following flavors available:
- Vanilla
- Chocolate
- Black Cherry
```

练习 9-7　管理员

管理员是一种特殊的用户。编写一个名为 `Admin` 的类，让它继承你为完成练习 9-3 或 9-5 而编写的 `User` 类。添加一个名为 `privileges` 的属性，用于存储一个由字符串（如 `"can add post"`、`"can delete post"` 和 `"can ban user"` 等）组成的列表。编写一个名为

`show_privileges()`的方法，它显示管理员的权限。创建一个 `Admin` 实例，并调用这个方法。

```
class User():
    """一个表示用户的简单类。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化用户。"""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """显示用户信息摘要。"""
        print(f"\n{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """向用户发出个性化问候。"""
        print(f"\nWelcome back, {self.username}!")

    def increment_login_attempts(self):
        """将属性 login_attempts 的值加 1。"""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """将 login_attempts 重置为 0。"""
        self.login_attempts = 0


class Admin(User):
    """有管理权限的用户。"""
```

```
    def __init__(self, first_name, last_name, username, email, location):
        """初始化管理员。"""
        super().__init__(first_name, last_name, username, email, location)
        self.privileges = []

    def show_privileges(self):
        """显示当前管理员的权限。"""
        print("\nPrivileges:")
        for privilege in self.privileges:
            print(f"- {privilege}")


eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric.privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]

eric.show_privileges()
```

输出：

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

练习 9-8　权限

编写一个名为 `Privileges` 的类，它只有一个属性 `privileges`，其中存储了练习 9-7 所说的字符串列表。将方法 `show_privileges()` 移到这个类中。在 `Admin` 类中，将一个 `Privileges` 实例用作其属性。创建一个 `Admin` 实例，并使用方法 `show_privileges()` 来显示其权限。

```
class User():
    """一个表示用户的简单类。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化用户。"""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """显示用户信息摘要。"""
        print(f"\n{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """向用户发出个性化问候。"""
        print(f"\nWelcome back, {self.username}!")

    def increment_login_attempts(self):
        """将属性 login_attempts 的值加 1。"""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """将 login_attempts 重置为 0。"""
```

```
        self.login_attempts = 0


class Admin(User):
    """有管理权限的用户。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化管理员。"""
        super().__init__(first_name, last_name, username, email, location)

        # 将权限集初始化为空。
        self.privileges = Privileges()


class Privileges():
    """一个存储管理员权限的类。"""

    def __init__(self, privileges=[]):
        self.privileges = privileges

    def show_privileges(self):
        print("\nPrivileges:")
        if self.privileges:
            for privilege in self.privileges:
                print(f"- {privilege}")
        else:
            print("- This user has no privileges.")


eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric.privileges.show_privileges()

print("\nAdding privileges...")
eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
```

```
        'can suspend accounts',
        ]
    eric.privileges.privileges = eric_privileges
    eric.privileges.show_privileges()
```

输出：

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

Privileges:
- This user has no privileges.

Adding privileges...

Privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

练习 9-9　电瓶升级

在本节最后一个 electric_car.py 版本中，给 `Battery` 类添加一个名为 `upgrade_battery()` 的方法。这个方法检查电瓶容量，如果它不是 85，就将它设置为 85。创建一辆电瓶容量为默认值的电动汽车，调用方法 `get_range()`，然后对电瓶进行升级，并再次调用 `get_range()`。你将看到这辆汽车的续航里程增加了。

```
class Car():
    """一次模拟汽车的简单尝试。"""

    def __init__(self, manufacturer, model, year):
        """初始化描述汽车的属性。"""
        self.manufacturer = manufacturer
        self.model = model
```

```
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """返回整洁的描述性信息。"""
        long_name = f"{self.year} {self.manufacturer} {self.model}"
        return long_name.title()

    def read_odometer(self):
        """打印一条指出汽车里程的消息。"""
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        """
        将里程表读数设置为指定的值。
        禁止将里程表读数往回调。
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """将里程表读数增加指定的量。"""
        self.odometer_reading += miles

class Battery():
    """一次模拟电动汽车电瓶的简单尝试。"""

    def __init__(self, battery_size=75):
        """初始化电瓶的属性。"""
        self.battery_size = battery_size

    def describe_battery(self):
        """打印一条描述电瓶容量的消息。"""
        print(f"This car has a {self.battery_size}-kWh battery.")
```

```
    def get_range(self):
        """打印一条消息，指出电瓶的续航里程。"""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        message = f"This car can go approximately {range}"
        message += " miles on a full charge."
        print(message)

    def upgrade_battery(self):
        """在可能的情况下将电瓶升级。"""
        if self.battery_size == 75:
            self.battery_size = 100
            print("Upgraded the battery to 100 kWh.")
        else:
            print("The battery is already upgraded.")


class ElectricCar(Car):
    """电动汽车的独特之处。"""

    def __init__(self, manufacturer, model, year):
        """
        初始化父类的属性。
        再初始化电动汽车特有的属性。
        """
        super().__init__(manufacturer, model, year)
        self.battery = Battery()


print("Make an electric car, and check the battery:")
my_tesla = ElectricCar('tesla', 'roadster', 2019)
my_tesla.battery.describe_battery()

print("\nUpgrade the battery, and check it again:")
my_tesla.battery.upgrade_battery()
```

```
my_tesla.battery.describe_battery()

print("\nTry upgrading the battery a second time.")
my_tesla.battery.upgrade_battery()
my_tesla.battery.describe_battery()
```

输出：

```
Make an electric car, and check the battery:
This car has a 75-kWh battery.

Upgrade the battery, and check it again:
Upgraded the battery to 100 kWh.
This car has a 100-kWh battery.

Try upgrading the battery a second time.
The battery is already upgraded.
This car has a 100-kWh battery.
```

练习 9-10　导入 Restaurant 类

将最新的 `Restaurant` 类存储在一个模块中。在另一个文件中，导入 `Restaurant` 类、创建一个 `Restaurant` 实例并调用 `Restaurant` 的一个方法，以确认 `import` 语句正确无误。

restaurant.py:

```
"""一个表示餐馆的类。"""

class Restaurant():
    """一个表示餐馆的类。"""

    def __init__(self, name, cuisine_type):
        """初始化餐馆。"""
        self.name = name.title()
        self.cuisine_type = cuisine_type
        self.number_served = 0
```

```
    def describe_restaurant(self):
        """显示餐馆信息摘要。"""
        msg = f"{self.name} serves wonderful {self.cuisine_type}."
        print(f"\n{msg}")

    def open_restaurant(self):
        """显示一条消息，指出餐馆正在营业。"""
        msg = f"{self.name} is open. Come on in!"
        print(f"\n{msg}")

    def set_number_served(self, number_served):
        """让用户能够设置就餐人数。"""
        self.number_served = number_served

    def increment_number_served(self, additional_served):
        """让用户能够增加就餐人数。"""
        self.number_served += additional_served
```

my_restaurant.py:

```
from restaurant import Restaurant

channel_club = Restaurant('the channel club', 'steak and seafood')
channel_club.describe_restaurant()
channel_club.open_restaurant()
```

输出：

```
The Channel Club serves wonderful steak and seafood.

The Channel Club is open. Come on in!
```

练习 9-11　导入 Admin 类

以为完成练习 9-8 而做的工作为基础。将 `User`、`Privileges` 和 `Admin` 类存储在一个模块中，再创建一个文件，在其中创建一个 `Admin` 实例并对其调用方法 `show_priveleges()`，以确认一切都能正确地运行。

user.py:

```
"""一系列模拟用户的类。"""

class User():
    """一个表示用户的简单类。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化用户。"""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """显示用户信息摘要。"""
        print(f"\n{self.first_name} {self.last_name}")
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """向用户发出个性化问候。"""
        print(f"\nWelcome back, {self.username}!")

    def increment_login_attempts(self):
        """将属性 login_attempts 的值加 1。"""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """将 login_attempts 重置为 0。"""
        self.login_attempts = 0


class Admin(User):
    """有管理权限的用户。"""
```

```
    def __init__(self, first_name, last_name, username, email, location):
        """初始化管理员。"""
        super().__init__(first_name, last_name, username, email, location)

        # 将权限集初始化为空。
        self.privileges = Privileges()


class Privileges():
    """存储管理员权限的类。"""

    def __init__(self, privileges=[]):
        self.privileges = privileges

    def show_privileges(self):
        print("\nPrivileges:")
        if self.privileges:
            for privilege in self.privileges:
                print(f"- {privilege}")
        else:
            print("- This user has no privileges.")
```

my_user.py:

```
from user import Admin

eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges

print(f"\nThe admin {eric.username} has these privileges: ")
eric.privileges.show_privileges()
```

输出：

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

The admin e_matthes has these privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

练习 9-12　多个模块

将 `User` 类存储在一个模块中，并将 `Privileges` 和 `Admin` 类存储在另一个模块中。再创建一个文件，在其中创建一个 `Admin` 实例并对其调用方法 `show_privileges()`，以确认一切都依然能够正确地运行。

user.py:

```
"""一个模拟用户的类。"""

class User():
    """一个表示用户的简单类。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化用户。"""
        self.first_name = first_name.title()
        self.last_name = last_name.title()
        self.username = username
        self.email = email
        self.location = location.title()
        self.login_attempts = 0

    def describe_user(self):
        """显示用户信息摘要。"""
        print(f"\n{self.first_name} {self.last_name}")
```

```
        print(f"  Username: {self.username}")
        print(f"  Email: {self.email}")
        print(f"  Location: {self.location}")

    def greet_user(self):
        """向用户发出个性化问候。"""
        print(f"\nWelcome back, {self.username}!")

    def increment_login_attempts(self):
        """将属性 login_attempts 的值加 1。"""
        self.login_attempts += 1

    def reset_login_attempts(self):
        """将 login_attempts 重置为 0。"""
        self.login_attempts = 0
```

admin.py:

```
"""一系列模拟管理员的类。"""

from user import User


class Admin(User):
    """有管理权限的用户。"""

    def __init__(self, first_name, last_name, username, email, location):
        """初始化管理员。"""
        super().__init__(first_name, last_name, username, email, location)

        # 将权限集初始化为空。
        self.privileges = Privileges()


class Privileges():
    """存储管理员权限的类。"""

    def __init__(self, privileges=[]):
        self.privileges = privileges
```

```
    def show_privileges(self):
        print("\nPrivileges:")
        if self.privileges:
            for privilege in self.privileges:
                print(f"- {privilege}")
        else:
            print("- This user has no privileges.")
```

my_admin.py

```
from admin import Admin

eric = Admin('eric', 'matthes', 'e_matthes', 'e_matthes@example.com', 'alaska')
eric.describe_user()

eric_privileges = [
    'can reset passwords',
    'can moderate discussions',
    'can suspend accounts',
    ]
eric.privileges.privileges = eric_privileges

print(f"\nThe admin {eric.username} has these privileges: ")
eric.privileges.show_privileges()
```

输出：

```
Eric Matthes
  Username: e_matthes
  Email: e_matthes@example.com
  Location: Alaska

The admin e_matthes has these privileges:
- can reset passwords
- can moderate discussions
- can suspend accounts
```

练习 9-13　骰子

模块 `random` 包含以各种方式生成随机数的函数，其中的 `randint()` 随机返回一个位于指定范围内的整数，例如，下面的代码随机地返回一个 1~6 的整数：

```
from random import randint
x = randint(1, 6)
```

请创建一个 `Die` 类，它包含一个名为 `sides` 的属性，该属性的默认值为 6。编写一个名为 `roll_die()` 的方法，它打印位于 1 和骰子面数之间的随机数。创建一个 6 面的骰子再掷 10 次。

创建一个 10 面的骰子和一个 20 面的骰子，再分别掷 10 次。

```
from random import randint

class Die():
    """一个表示骰子的类。"""

    def __init__(self, sides=6):
        """初始化骰子。"""
        self.sides = sides

    def roll_die(self):
        """返回一个位于 1 和骰子面数之间的随机数。"""
        return randint(1, self.sides)

# 创建一个 6 面的骰子，再掷 10 次并显示结果。
d6 = Die()

results = []
for roll_num in range(10):
    result = d6.roll_die()
    results.append(result)
print("10 rolls of a 6-sided die:")
print(results)
```

```
# 创建一个 10 面的骰子，再掷 10 次并显示结果。
d10 = Die(sides=10)

results = []
for roll_num in range(10):
    result = d10.roll_die()
    results.append(result)
print("\n10 rolls of a 10-sided die:")
print(results)

# 创建一个 20 面的骰子，再掷 10 次并显示结果。
d20 = Die(sides=20)

results = []
for roll_num in range(10):
    result = d20.roll_die()
    results.append(result)
print("\n10 rolls of a 20-sided die:")
print(results)
```

输出：

```
10 rolls of a 6-sided die:
[5, 5, 6, 3, 6, 4, 2, 2, 1, 1]

10 rolls of a 10-sided die:
[8, 9, 8, 10, 7, 1, 3, 5, 3, 4]

10 rolls of a 20-sided die:
[4, 3, 18, 17, 3, 1, 13, 12, 5, 14]
```

练习 9-14　彩票

创建一个列表或元组，其中包含 10 个数和 5 个字母。从这个列表或元组中随机地选择 4 个数或字母，并打印一条消息，指出只要彩票上是这 4 个数或字母，就中大奖了。

```
from random import choice

possibilities = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'b', 'c', 'd', 'e']
```

```
winning_ticket = []
print("Let's see what the winning ticket is...")

# 中奖组合中不能包含重复的数或字母，因此使用了 while 循环。
while len(winning_ticket) < 4:
    pulled_item = choice(possibilities)

    # 仅当摇出的数字或字母不在组合中时，才将其添加到组合中。
    if pulled_item not in winning_ticket:
        print(f"  We pulled a {pulled_item}!")
        winning_ticket.append(pulled_item)
```

输出：

```
Let's see what the winning ticket is...
  We pulled a 10!
  We pulled a a!
  We pulled a 2!
  We pulled a 4!

The final winning ticket is: [10, 'a', 2, 4]
```

练习 9-15　彩票分析

可使用循环来搞明白中彩票大奖有多难。为此，创建一个名为 `my_ticket` 的列表或元组，再编写一个循环，不断地随机选择数或字母，直到中大奖为止。请打印一条消息，指出执行循环多少次才中了大奖。

```
from random import choice

def get_winning_ticket(possibilities):
    """摇出中奖组合。"""
    winning_ticket = []

    # 中奖组合中不能包含重复的数字或字母，因此使用了 while 循环。
    while len(winning_ticket) < 4:
```

```
            pulled_item = choice(possibilities)

            # 仅当摇出的数字或字母不在组合中时，才将其添加到组合中。
            if pulled_item not in winning_ticket:
                winning_ticket.append(pulled_item)

        return winning_ticket

    def check_ticket(played_ticket, winning_ticket):
        # 检查彩票的每个数字或字母，只要有一个不在中奖组合中，就返回 False。
        for element in played_ticket:
            if element not in winning_ticket:
                return False

        # 如果代码执行到这里，就说明中奖了！
        return True

    def make_random_ticket(possibilities):
        """随机地生成彩票。"""
        ticket = []
        # 彩票不能包含重复的数字或字母，因此使用了 while 循环。
        while len(ticket) < 4:
            pulled_item = choice(possibilities)

            # 仅当随机生成的数字或字母不在彩票中时，才将其添加到彩票中。
            if pulled_item not in ticket:
                ticket.append(pulled_item)

        return ticket


    possibilities = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 'a', 'b', 'c', 'd', 'e']
    winning_ticket = get_winning_ticket(possibilities)

    plays = 0
    won = False

    # 为避免程序执行时间太长，设置最多随机生成多少张彩票。
```

```
max_tries = 1_000_000

while not won:
    new_ticket = make_random_ticket(possibilities)
    won = check_ticket(new_ticket, winning_ticket)
    plays += 1
    if plays >= max_tries:
        break

if won:
    print("We have a winning ticket!")
    print(f"Your ticket: {new_ticket}")
    print(f"Winning ticket: {winning_ticket}")
    print(f"It only took {plays} tries to win!")
else:
    print(f"Tried {plays} times, without pulling a winner. :(")
    print(f"Your ticket: {new_ticket}")
    print(f"Winning ticket: {winning_ticket}")
```

输出：

```
We have a winning ticket!
Your ticket: ['a', 8, 'b', 7]
Winning ticket: [7, 'b', 8, 'a']
It only took 408 tries to win!
```

第 10 章

练习 10-1　Python 学习笔记

在文本编辑器中新建一个文件，写几句话来总结一下你至此学到的 Python 知识，其中每一行都以“In Python you can”打头。将这个文件命名为 learning_python.txt，并将其存储到为完成本章练习而编写的程序所在的目录中。编写一个程序，它读取这个文件，并将你所写的内容打印三次：第一次打印时读取整个文件；第二次打印时遍历文件对象；第三次打印时将各行存储在一个列表中，再在 `with` 代码块外打印它们。

learning_python.txt:

```
In Python you can store as much information as you want.
```

```
In Python you can connect pieces of information.
In Python you can model real-world situations.
```

learning_python.py:

```
filename = 'learning_python.txt'

print("--- Reading in the entire file:")
with open(filename) as f:
    contents = f.read()
print(contents)

print("\n--- Looping over the lines:")
with open(filename) as f:
    for line in f:
        print(line.rstrip())

print("\n--- Storing the lines in a list:")
with open(filename) as f:
    lines = f.readlines()

for line in lines:
    print(line.rstrip())
```

输出：

```
--- Reading in the entire file:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.

--- Looping over the lines:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.

--- Storing the lines in a list:
In Python you can store as much information as you want.
In Python you can connect pieces of information.
In Python you can model real-world situations.
```

练习 10-2　C 语言学习笔记

你可使用方法 `replace()` 将字符串中的特定单词都替换为另一个单词。下面是一个简单的示例，演示了如何将句子中的 `'dog'` 替换为 `'cat'`：

```
>>> message = "I really like dogs."
>>> message.replace('dog', 'cat')
'I really like cats.'
```

读取你刚创建的文件 learning_python.txt 中的每一行，将其中的 Python 都替换为另一门语言的名称，如 C。将修改后的各行都打印到屏幕上。

```
filename = 'learning_python.txt'

with open(filename) as f:
    lines = f.readlines()

for line in lines:
    # 删除行尾的换行符，再将 Python 替换为 C。
    line = line.rstrip()
    print(line.replace('Python', 'C'))
```

输出：

```
In C you can store as much information as you want.
In C you can connect pieces of information.
In C you can model real-world situations.
```

可在一行代码中依次调用 `rstrip()` 和 `replace()`，这被称为方法串接。在下面的代码中，删除行尾的换行符再将 Python 替换为 C，输出与前面的代码相同。

```
filename = 'learning_python.txt'

with open(filename) as f:
    lines = f.readlines()

for line in lines:
    # 删除行尾的换行符，再将 Python 替换为 C。
    print(line.rstrip().replace('Python', 'C'))
```

练习 10-3 访客

编写一个程序，提示用户输入其名字；用户做出响应后，将其名字写入到文件 guest.txt 中。

```
name = input("What's your name? ")

filename = 'guest.txt'

with open(filename, 'w') as f:
    f.write(name)
```

输出：

```
What's your name? eric
```

guest.txt:

```
eric
```

练习 10-4　访客名单

编写一个 `while` 循环，提示用户输入其名字。用户输入其名字后，在屏幕上打印问候语，并将一条到访记录添加到文件 guest_book.txt 中。确保这个文件中的每条记录都独占一行。

```
filename = 'guest_book.txt'

print("Enter 'quit' when you are finished.")
while True:
    name = input("\nWhat's your name? ")
    if name == 'quit':
        break
    else:
        with open(filename, 'a') as f:
            f.write(f"{name}\n")
        print(f"Hi {name}, you've been added to the guest book.")
```

输出：

```
Enter 'quit' when you are finished.
```

```
What's your name? eric
Hi eric, you've been added to the guest book.

What's your name? willie
Hi willie, you've been added to the guest book.

What's your name? ever
Hi ever, you've been added to the guest book.

What's your name? erin
Hi erin, you've been added to the guest book.

What's your name? quit
```

guest_book.txt:

```
eric
willie
ever
erin
```

练习 10-5　调查

编写一个 `while` 循环，询问用户为何喜欢编程。每当用户输入一个原因后，都将其添加到一个存储所有原因的文件中。

```
filename = 'programming_poll.txt'

responses = []
while True:
    response = input("\nWhy do you like programming? ")
    responses.append(response)

    continue_poll = input("Would you like to let someone else respond? (y/n) ")
    if continue_poll != 'y':
        break
```

```
    with open(filename, 'a') as f:
        for response in responses:
            f.write(f"{response}\n")
```

输出：

```
Why do you like programming? Programmers can build almost anything they can imagine.
Would you like to let someone else respond? (y/n) y

Why do you like programming? It's really fun, and really satisfying.
Would you like to let someone else respond? (y/n) y

Why do you like programming? It just never gets old.
Would you like to let someone else respond? (y/n) n
```

programming_poll.txt:

```
Programmers can build almost anything they can imagine.
It's really fun, and really satisfying.
It just never gets old.
```

练习 10-6　加法运算

提示用户提供数值输入时，常出现的一个问题是，用户提供的是文本而不是数。在这种情况下，当你尝试将输入转换为整数时，将引发 `ValueError` 异常。编写一个程序，提示用户输入两个数，再将它们相加并打印结果。在用户输入的任何一个值不是数字时都捕获 `ValueError` 异常，并打印一条友好的错误消息。对你编写的程序进行测试：先输入两个数，再输入一些文本而不是数。

```
try:
    x = input("Give me a number: ")
    x = int(x)

    y = input("Give me another number: ")
    y = int(y)
except ValueError:
    print("Sorry, I really needed a number.")
else:
```

```
    sum = x + y
    print(f"The sum of {x} and {y} is {sum}.")
```

用户输入两个整数时的输出：

```
Give me a number: 23
Give me another number: 47
The sum of 23 and 47 is 70.
```

用户输入的不是数时的输出：

```
Give me a number: 23
Give me another number: fred
Sorry, I really needed a number.
```

练习 10-7　加法计算器

将你为完成练习 10-6 而编写的代码放在一个 `while` 循环中，让用户犯错（输入的是文本而不是数）后能够继续输入数。

```
print("Enter 'q' at any time to quit.\n")

while True:
    try:
        x = input("\nGive me a number: ")
        if x == 'q':
            break

        x = int(x)

        y = input("Give me another number: ")
        if y == 'q':
            break

        y = int(y)

    except ValueError:
        print("Sorry, I really needed a number.")
```

```
        else:
            sum = x + y
            print(f"The sum of {x} and {y} is {sum}.")
```

输出：

```
Enter 'q' at any time to quit.

Give me a number: 23
Give me another number: 47
The sum of 23 and 47 is 70.

Give me a number: three
Sorry, I really needed a number.

Give me a number: 3
Give me another number: five
Sorry, I really needed a number.

Give me a number: -12
Give me another number: 20
The sum of -12 and 20 is 8.

Give me a number: q
```

练习 10-8　猫和狗

创建两个文件 cats.txt 和 dogs.txt，在第一个文件中至少存储三只猫的名字，在第二个文件中至少存储三条狗的名字。编写一个程序，尝试读取这些文件，并将其内容打印到屏幕上。将这些代码放在一个 `try-except` 代码块中，以便在文件不存在时捕获 `FileNotFound` 错误，并打印一条友好的消息。将文件之一移到另一个地方，并确认 `except` 代码块中的代码将正确地执行。

cats.txt:

```
henry
clarence
```

```
mildred
```

dogs.txt:

```
willie
annahootz
summit
```

cats_and_dogs.py:

```
filenames = ['cats.txt', 'dogs.txt']

for filename in filenames:
    print(f"\nReading file: {filename}")
    try:
        with open(filename) as f:
            contents = f.read()
            print(contents)
    except FileNotFoundError:
        print("  Sorry, I can't find that file.")
```

两个文件都存在时的输出：

```
Reading file: cats.txt
henry
clarence
mildred

Reading file: dogs.txt
willie
annahootz
summit
```

移走文件 cats.txt 后的输出：

```
Reading file: cats.txt
  Sorry, I can't find that file.

Reading file: dogs.txt
willie
annahootz
summit
```

练习 10-9　静默的猫和狗

修改你在练习 10-8 中编写的 `except` 代码块，让程序在文件不存在时静默失败。

```
filenames = ['cats.txt', 'dogs.txt']

for filename in filenames:

    try:
        with open(filename) as f:
            contents = f.read()

    except FileNotFoundError:
        pass

    else:
        print(f"\nReading file: {filename}")
        print(contents)
```

两个文件都存在时的输出：

```
Reading file: cats.txt
henry
clarence
mildred

Reading file: dogs.txt
willie
annahootz
summit
```

移走文件 cats.txt 后的输出：

```
Reading file: dogs.txt
willie
annahootz
summit
```

练习 10-10　常见单词

访问古登堡计划（http://gutenberg.org/），并找一些你想分析的图书。下载这些作品的文本文件或将浏览器中的原始文本复制到文本文件中。

你可使用方法 `count()` 来确定特定的单词或短语在字符串中出现了多少次。例如，下面的代码计算 `'row'` 在一个字符串中出现了多少次：

```
>>> line = "Row, row, row your boat"
>>> line.count('row')
2
>>> line.lower().count('row')
3
```

请注意，通过使用 `lower()` 将字符串转换为小写，可捕捉要查找的单词的各种外观，而不管其大小写格式如何。

编写一个程序，它读取你在古登堡计划中获取的文件，并计算单词`'the'`在每个文件中分别出现了多少次。这里计算得到的结果并不准确，因为将诸如`'then'`和`'there'`等单词也计算在内了。请尝试计算`'the '`（包含空格）出现的次数，看看结果相差多少。

common_words.py

```
def count_common_words(filename, word):
    """计算指定的单词在图书中出现了多少次。"""
    # 请注意，这里计算得到的结果并不准确，比实际出现的次数要多。
    try:
        with open(filename, encoding='utf-8') as f:
            contents = f.read()
    except FileNotFoundError:
        pass
    else:
        word_count = contents.lower().count(word)

        msg = f"'{word}' appears in {filename} about {word_count} times."
        print(msg)

filename = 'alice.txt'
```

```
count_common_words(filename, 'the')
```

输出：

```
'the' appears in alice.txt about 2505 times.
```

这里只计算了'the '在一部图书中出现的次数，但可使用这个函数来计算特定单词在任意图书中出现的次数。

练习 10-11　喜欢的数

编写一个程序，提示用户输入他喜欢的数，并使用 `json.dump()` 将这个数字存储到文件中。再编写一个程序，从文件中读取这个值，并打印消息“I know your favorite number! It’s _____.”。

favorite_number_write.py:

```
import json

number = input("What's your favorite number? ")

with open('favorite_number.json', 'w') as f:
    json.dump(number, f)
    print("Thanks! I'll remember that.")
```

输出：

```
What's your favorite number? 42
Thanks! I'll remember that.
```

favorite_number_read.py:

```
import json

with open('favorite_number.json') as f:
    number = json.load(f)

print(f"I know your favorite number! It's {number}.")
```

输出：

```
I know your favorite number! It's 42.
```

练习 10-12　记住喜欢的数

将练习 10-11 中的两个程序合而为一。如果存储了用户喜欢的数，就向用户显示它；否则提示用户输入他喜欢的数并将其存储到文件中。运行这个程序两次，看看它是否像预期的那样工作。

```
import json

try:
    with open('favorite_number.json') as f:
        number = json.load(f)
except FileNotFoundError:
    number = input("What's your favorite number? ")
    with open('favorite_number.json', 'w') as f:
        json.dump(number, f)
    print("Thanks, I'll remember that.")
else:
    print(f"I know your favorite number! It's {number}.")
```

第一次运行时的输出：

```
What's your favorite number? 42
Thanks, I'll remember that.
```

第二次运行的输出：

```
I know your favorite number! It's 42.
```

练习 10-13　验证用户

最后一个 remember_me.py 版本假设用户要么已输入其用户名，要么是首次运行该程序。应修改这个程序，以应对这样的情形：当前和最后一次运行该程序的用户并非同一个人。

为此，在 `greet_user()` 中打印欢迎用户回来的消息前，询问他用户名是否是对的。如果不对，就调用 `get_new_username()` 让用户输入正确的用户名。

```
import json

def get_stored_username():
```

```
    """获取存储的用户名——如果存储了。"""
    filename = 'username.json'
    try:
        with open(filename) as f_obj:
            username = json.load(f_obj)
    except FileNotFoundError:
        return None
    else:
        return username

def get_new_username():
    """提示用户输入用户名。"""
    username = input("What is your name? ")
    filename = 'username.json'
    with open(filename, 'w') as f_obj:
        json.dump(username, f_obj)
    return username

def greet_user():
    """基于用户名问候用户。"""
    username = get_stored_username()
    if username:
        correct = input(f"Are you {username}? (y/n) ")
        if correct == 'y':
            print(f"Welcome back, {username}!")
        else:
            username = get_new_username()
            print(f"We'll remember you when you come back, {username}!")
    else:
        username = get_new_username()
        print(f"We'll remember you when you come back, {username}!")

greet_user()
```

输出：

```
> python verify_user.py
What is your name? eric
We'll remember you when you come back, eric!
```

```
> python verify_user.py
Are you eric? (y/n) y
Welcome back, eric!

> python verify_user.py
Are you eric? (y/n) n
What is your name? ever
We'll remember you when you come back, ever!

> python verify_user.py
Are you ever? (y/n) y
Welcome back, ever!
```

你可能注意到了，在这个版本的 `greet_user()` 中，有两个相同的 `else` 代码块。要整理这个函数，一种方法是使用空的 `return` 语句。空的 `return` 语句让 Python 离开当前函数，不执行后面的代码。

下面的 `greet_user()` 版本更清晰的：

```
def greet_user():
    """基于用户名问候用户。"""
    username = get_stored_username()
    if username:
        correct = input(f"Are you {username}? (y/n) ")
        if correct == 'y':
            print(f"Welcome back, {username}!")
            return

    # 获得了用户名，但不对。
    # 因此提示用户输入其用户名。
    username = get_new_username()
    print(f"We'll remember you when you come back, {username}!")
```

这条 `return` 语句意味着打印欢迎回来的消息后，不再执行后面的代码。如果用户名

不存在或不对，就根本不会执行这条 `return` 语句。仅当 `if` 语句中的条件不满足时，才会执行这个函数的第二部分，因此不需要将它们放在 `else` 代码块中。现在，只要有一条 `if` 语句中的条件不满足，这个函数就会提示用户输入其用户名。

现在唯一的问题是嵌套了 `if` 语句。为解决这个问题，可将检查用户名是否正确的代码移到另一个函数中。如果你觉得这个练习很有意思，可再尝试编写一个名为 `check_username()` 的函数，以免在 `greet_user()` 中嵌套 `if` 语句。

第 11 章

练习 11-1　城市和国家

编写一个函数，它接受两个形参：一个城市名和一个国家名。这个函数返回一个格式为 `City, Country` 的字符串，如 `Santiago, Chile`。将这个函数存储在一个名为 city_functions.py 的模块中。

创建一个名为 test_cities.py 的程序，对刚编写的函数进行测试（别忘了，你需要导入模块 `unittest` 以及要测试的函数）。编写一个名为 `test_city_country()` 的方法，核实使用类似于 `'santiago'` 和 `'chile'` 这样的值来调用前述函数时，得到的字符串是正确的。运行 test_cities.py，确认测试 `test_city_country()` 通过了。

city_functions.py:

```
"""一系列处理城市的函数。"""

def city_country(city, country):
    """返回一个形如'Santiago, Chile'的字符串。"""
    return f"{city.title()}, {country.title()}"
```

注意：这个函数是在练习 8-6 中编写的。

test_cities.py:

```
import unittest

from city_functions import city_country

class CitiesTestCase(unittest.TestCase):
    """测试 city_functions.py。"""

    def test_city_country(self):
        """传入简单的城市和国家可行吗？"""
        santiago_chile = city_country('santiago', 'chile')
        self.assertEqual(santiago_chile, 'Santiago, Chile')

if __name__ == '__main__':
    unittest.main()
```

输出：

```
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

练习 11-2　人口数量

修改前面的函数，使其包含第三个必不可少的形参 `population`，并返回一个格式为 `City, Country - population xxx` 的字符串，如 `Santiago, Chile - population 5000000`。运行 test_cities.py，确认测试 `test_city_country()` 未通过。

修改上述函数，将形参 `population` 设置为可选的。再次运行 test_cities.py，确认测试 `test_city_country()` 又通过了。

再编写一个名为 `test_city_country_population()` 的测试，核实可以使用类似于 `'santiago'`、`'chile'` 和 `'population=5000000'` 这样的值来调用这个函数。再次运行 test_cities.py，确认测试 `test_city_country_population()` 通过了。

修改后的 city_functions.py，包含必不可少的形参 `population`：

```
"""一系列处理城市的函数。"""

def city_country(city, country, population):
    """返回一个形如'Santiago, Chile - population 5000000'的字符串。"""
    output_string = f"{city.title()}, {country.title()}"
    output_string += f" -population {population}"
    return output_string
```

运行 test_cities.py 得到的输出：

```
E
======================================================================
ERROR: test_city_country (__main__.CitiesTestCase)
Does a simple city and country pair work?
----------------------------------------------------------------------
Traceback (most recent call last):
  File "pcc\solutions\test_cities.py", line 10, in test_city_country
    santiago_chile = city_country('santiago', 'chile')
TypeError: city_country() missing 1 required positional argument: 'population'

----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
```

修改后的 city_functions.py，包含可选形参 `population`：

```
"""一系列处理城市的函数。"""

def city_country(city, country, population=0):
    """返回一个表示城市和国家的字符串。"""
```

```
    output_string = f"{city.title()}, {country.title()}"
    if population:
        output_string += f" - population {population}"
    return output_string
```

运行 test_cities.py 得到的输出：

```
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

修改后的 test_cities.py：

```
import unittest

from city_functions import city_country

class CitiesTestCase(unittest.TestCase):
    """测试'city_functions.py'。"""

    def test_city_country(self):
        """传入简单的城市和国家可行吗？"""
        santiago_chile = city_country('santiago', 'chile')
        self.assertEqual(santiago_chile, 'Santiago, Chile')

    def test_city_country_population(self):
        """可向形参 population 传递值吗？"""
        santiago_chile = city_country('santiago', 'chile', population=5_000_000)
        self.assertEqual(santiago_chile, 'Santiago, Chile - population 5000000')

if __name__ == '__main__':
    unittest.main()
```

输出：

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s
```

```
OK
```

练习 11-3　雇员

编写一个名为 `Employee` 的类，其方法 `__init__()` 接受名、姓和年薪，并将它们都存储在属性中。编写一个名为 `give_raise()` 的方法，它默认将年薪增加 5000 美元，但也能够接受其他的年薪增加量。

为 `Employee` 编写一个测试用例，其中包含两个测试方法：`test_give_default_raise()` 和 `test_give_custom_raise()`。使用方法 `setUp()`，以免在每个测试方法中都创建新的雇员实例。运行这个测试用例，确认两个测试都通过了。

employee.py:

```
class Employee():
    """一个表示雇员的类。"""

    def __init__(self, f_name, l_name, salary):
        """初始化雇员。"""
        self.first = f_name.title()
        self.last = l_name.title()
        self.salary = salary

    def give_raise(self, amount=5000):
        """给雇员加薪。"""
        self.salary += amount
```

test_employee.py:

```
import unittest

from employee import Employee

class TestEmployee(unittest.TestCase):
    """测试模块 employee。"""
```

```
    def setUp(self):
        """创建一个 Employee 实例，以便在测试中使用。"""
        self.eric = Employee('eric', 'matthes', 65_000)

    def test_give_default_raise(self):
        """测试使用默认的年薪增加量是否可行。"""
        self.eric.give_raise()
        self.assertEqual(self.eric.salary, 70000)

    def test_give_custom_raise(self):
        """测试自定义年薪增加量是否可行。"""
        self.eric.give_raise(10000)
        self.assertEqual(self.eric.salary, 75000)

if __name__ == '__main__':
    unittest.main()
```

输出：

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

第 15 章

练习 15-1　立方

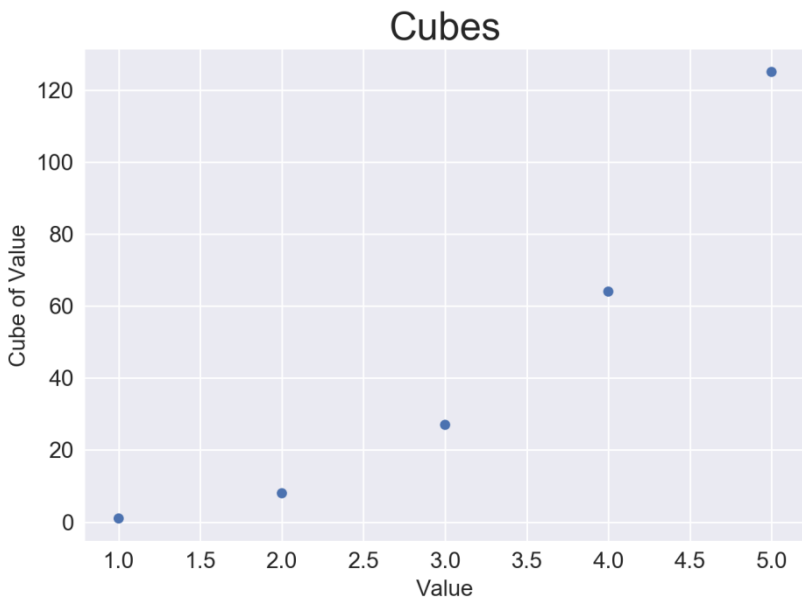
数字的三次方称为立方。请绘制一个图形，显示前 5 个整数的立方值；再绘制一个图形，显示前 5000 个整数的立方值。

绘制一个图形，显示前 5 个整数的立方值：

```
from matplotlib import pyplot as plt

# 定义数据。
x_values = [1, 2, 3, 4, 5]
cubes = [1, 8, 27, 64, 125]
```

```
# 创建图形。
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(x_values, cubes, edgecolor='none', s=40)

# 设置图表名称和坐标轴名称。
ax.set_title("Cubes", fontsize=24)
ax.set_xlabel('Value', fontsize=14)
ax.set_ylabel('Cube of Value', fontsize=14)

# 设置刻度标签的大小。
ax.tick_params(axis='both', labelsize=14)

# 显示图形。
plt.show()
```

输出：

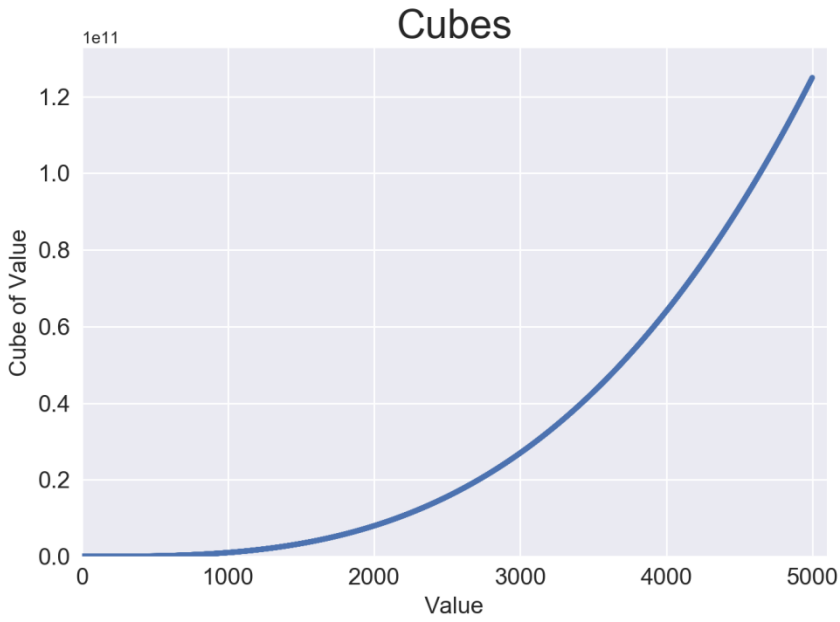


绘制一个图形，显示前 5000 个整数的立方值：

```
from matplotlib import pyplot as plt
```

```
# 定义数据。
x_values = list(range(5001))
cubes = [x**3 for x in x_values]

# 创建图形。
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(x_values, cubes, s=10)

# 设置图表名称和坐标轴名称。
ax.set_title("Cubes", fontsize=24)
ax.set_xlabel('Value', fontsize=14)
ax.set_ylabel('Cube of Value', fontsize=14)

# 设置刻度标签的大小以及每个坐标轴的取值范围。
ax.tick_params(axis='both', labelsize=14)
ax.axis([0, 5100, 0, 5100**3])

# 显示图形。
plt.show()
```

输出：

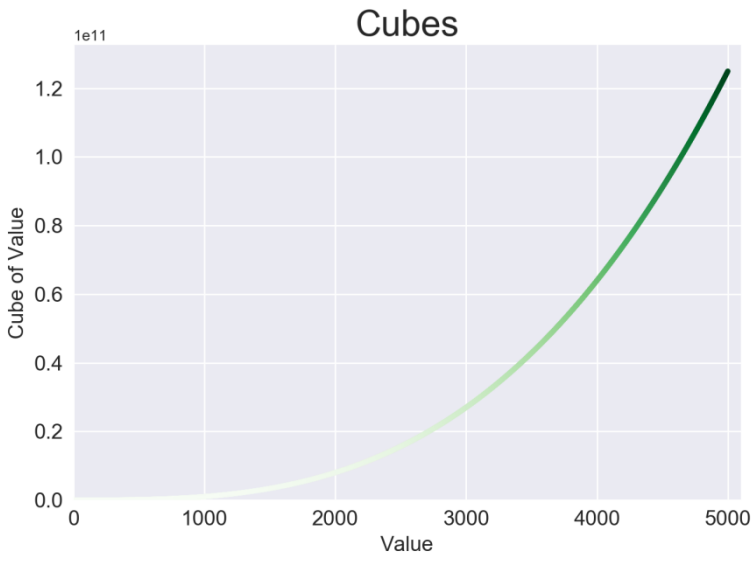


练习 15-2　彩色立方

给前面绘制的立方图指定颜色映射。

```
from matplotlib import pyplot as plt

# 定义数据。
x_values = list(range(5001))
cubes = [x**3 for x in x_values]

# 创建图形。
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.scatter(x_values, cubes, c=cubes, cmap=plt.cm.Greens, s=10)

# 设置图表名称和坐标轴名称。
ax.set_title("Cubes", fontsize=24)
ax.set_xlabel('Value', fontsize=14)
ax.set_ylabel('Cube of Value', fontsize=14)

# 设置刻度标签的大小以及每个坐标轴的取值范围。
ax.tick_params(axis='both', labelsize=14)
ax.axis([0, 5100, 0, 5100**3])

# 显示图形。
plt.show()
```

输出：



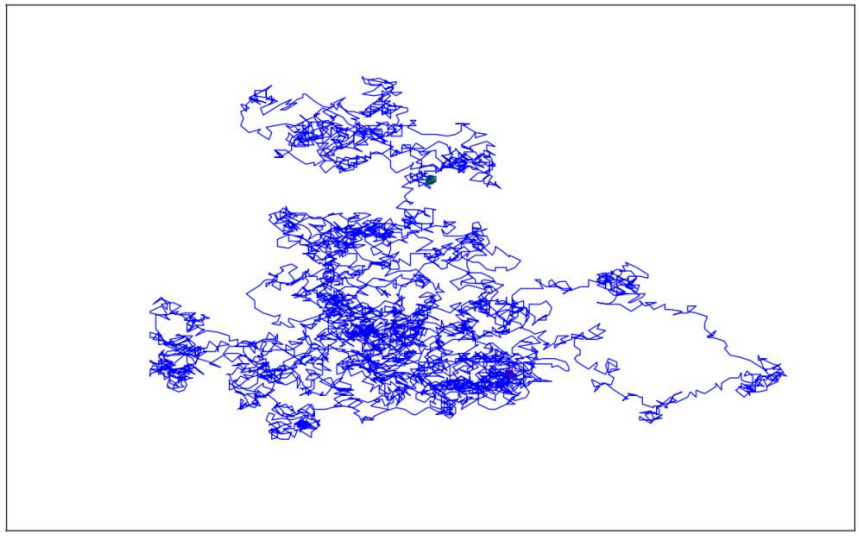
练习 15-3　分子运动

修改 rw_visual.py，将其中的 `ax.scatter()` 替换为 `ax.plot()`。为模拟花粉在水滴表面的运动路径，向 `ax.plot()` 传递 `rw.x_values` 和 `rw.y_values`，并指定实参值 `linewidth`。使用 5000 个点而不是 50 000 个点。

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 只要程序处于活动状态，就不断地模拟随机漫步。
while True:
    # 创建一个 RandomWalk 实例。
    rw = RandomWalk(5_000)
    rw.fill_walk()

    # 将 RandomWalk 实例包含的点都绘制出来。
    plt.style.use('classic')
    fig, ax = plt.subplots(figsize=(15, 9))
    point_numbers = range(rw.num_points)
```

```
    ax.plot(rw.x_values, rw.y_values, linewidth=1)

    # 突出起点和终点。
    ax.scatter(0, 0, c='green', edgecolors='none', s=100)
    ax.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
        s=100)

    # 隐藏坐标轴。
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    plt.show()

    keep_running = input("Make another walk? (y/n): ")
    if keep_running == 'n':
        break
```

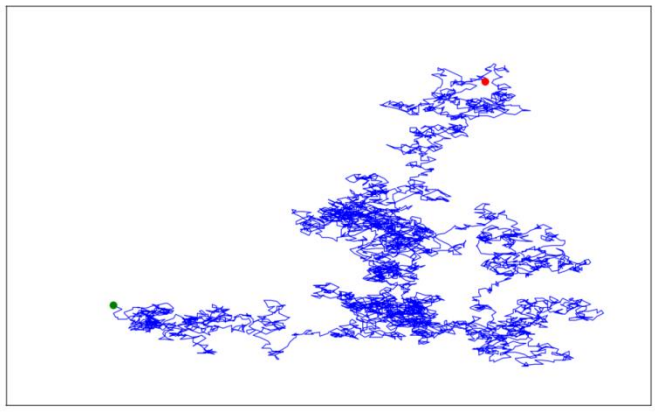
输出：



起点和终点看起来位于折线的后面。要将它们放在折线的前面，可使用参数 `zorder`。`zorder` 值大的图表元素位于 `zorder` 值小的图表元素前面。

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 只要程序处于活动状态，就不断地模拟随机漫步。
while True:
    # 创建一个 RandomWalk 实例。
    rw = RandomWalk(5_000)
    rw.fill_walk()

    # 将 RandomWalk 实例包含的点都绘制出来。
    plt.style.use('classic')
    fig, ax = plt.subplots(figsize=(15, 9))
    point_numbers = range(rw.num_points)
    ax.plot(rw.x_values, rw.y_values, linewidth=1, zorder=1)

    # 突出起点和终点。
    ax.scatter(0, 0, c='green', edgecolors='none', s=100, zorder=2)
    ax.scatter(rw.x_values[-1], rw.y_values[-1], c='red', edgecolors='none',
        s=100, zorder=2)

    # 隐藏坐标轴。
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    plt.show()

    keep_running = input("Make another walk? (y/n): ")
    if keep_running == 'n':
        break
```

输出：



练习 15-5　重构

方法 `fill_walk()` 很长。请新建一个名为 `get_step()` 的方法，用于确定每次漫步的距离和方向，并计算这次漫步将如何移动。然后，在 `fill_walk()` 中调用 `get_step()` 两次：

```
x_step = self.get_step()
y_step = self.get_step()
```

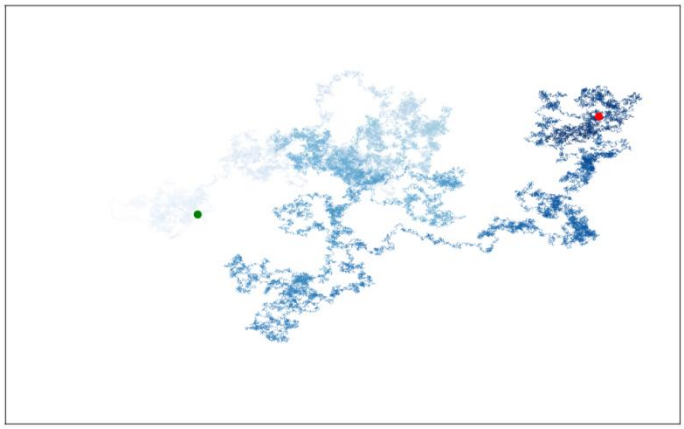
通过这样的重构，可缩小 `fill_walk()` 的规模，让这个方法阅读和理解起来更容易。

random_walk.py:

```
from random import choice

class RandomWalk:
    """一个生成随机漫步数据的类。"""

    def __init__(self, num_points=5000):
        """初始化随机漫步的属性。"""
        self.num_points = num_points

        # 所有随机漫步都始于(0, 0)。
        self.x_values = [0]
        self.y_values = [0]
```

```
def get_step(self):
    """决定漫步的方向和距离。"""
    direction = choice([1, -1])
    distance = choice([0, 1, 2, 3, 4])
    step = direction * distance
    return step

def fill_walk(self):
    """计算随机漫步包含的所有点。"""

    # 不断漫步，直到列表达到指定的长度。
    while len(self.x_values) < self.num_points:

        # 决定前进方向以及沿这个方向前进的距离。
        x_step = self.get_step()
        y_step = self.get_step()

        # 拒绝原地踏步。
        if x_step == 0 and y_step == 0:
            continue

        # 计算下一个点的 x 和 y 值。
        x = self.x_values[-1] + x_step
        y = self.y_values[-1] + y_step

        self.x_values.append(x)
        self.y_values.append(y)
```

输出：



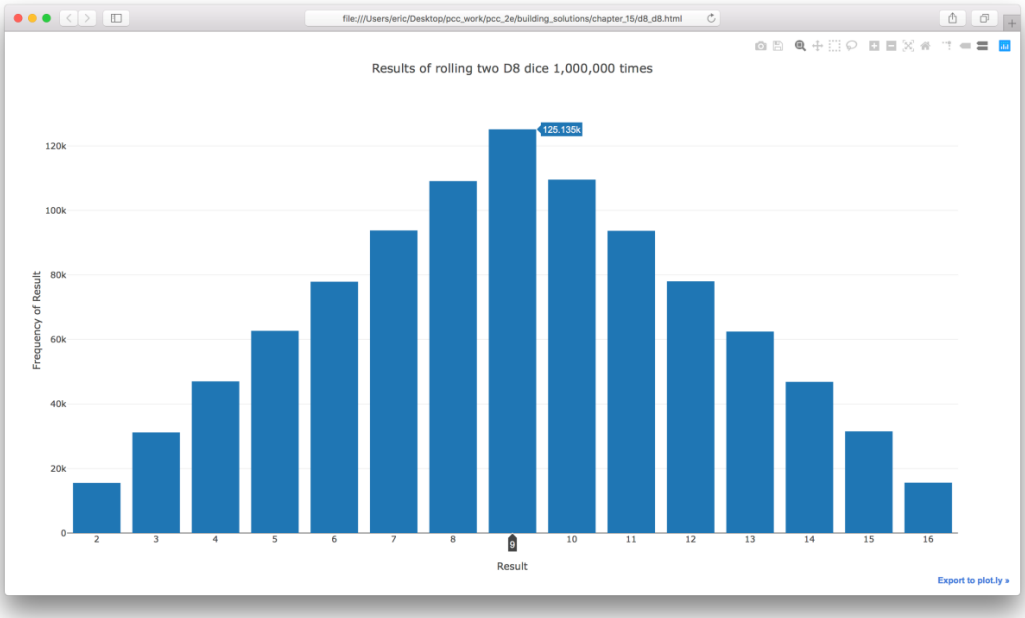
练习 15-6　两个 D8

编写一个程序，模拟同时掷两个 8 面骰子 1000 次的结果。先想象一下结果会是什么样的，再运行这个程序，看看你的直觉准不准。逐渐增加掷骰子的次数，直到系统不堪重负为止。

```
from plotly.graph_objs import Bar, Layout
from plotly import offline

from die import Die

# 创建两个 D8 骰子。
die_1 = Die(num_sides=8)
die_2 = Die(num_sides=8)

# 掷骰子多次，并将结果存储在一个列表中。
results = []
for roll_num in range(1_000_000):
    result = die_1.roll() + die_2.roll()
    results.append(result)

# 分析结果。
frequencies = []
max_result = die_1.num_sides + die_2.num_sides
for value in range(2, max_result+1):
```

```
    frequency = results.count(value)
    frequencies.append(frequency)

# 可视化结果。
x_values = list(range(2, max_result+1))
data = [Bar(x=x_values, y=frequencies)]

x_axis_config = {'title': 'Result', 'dtick': 1}
y_axis_config = {'title': 'Frequency of Result'}
my_layout = Layout(title='Results of rolling two D8 dice 1,000,000 times',
       xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d8_d8.html')
```

输出：



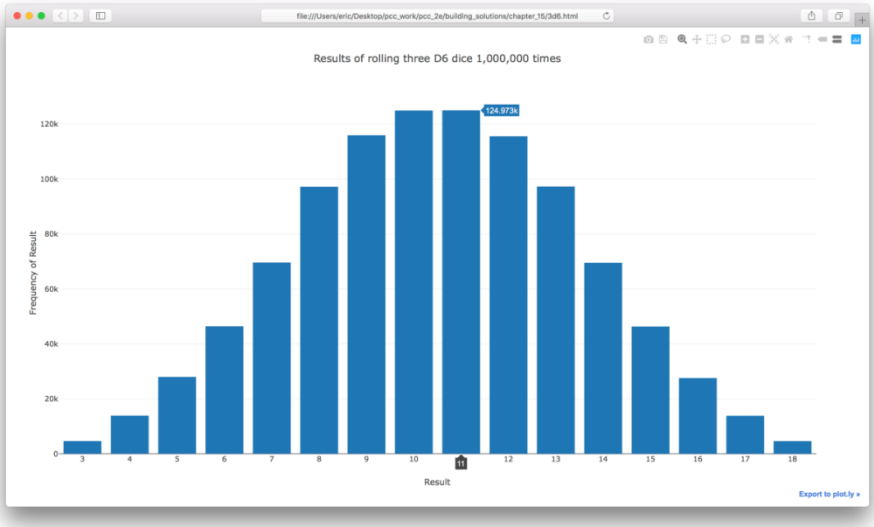
练习 15-7　同时掷三个骰子

如果你同时掷三个 D6 骰子，可能得到的最小点数为 3，而最大点数为 18。请通过可视化展示同时掷三个 D6 骰子的结果。

```
from plotly.graph_objs import Bar, Layout
```

```
from plotly import offline

from die import Die

# 创建三个 D6 骰子。
die_1 = Die()
die_2 = Die()
die_3 = Die()

# 掷骰子多次，并将结果存储在一个列表中。
results = []
for roll_num in range(1_000_000):
    result = die_1.roll() + die_2.roll() + die_3.roll()
    results.append(result)

# 分析结果。
frequencies = []
max_result = die_1.num_sides + die_2.num_sides + die_3.num_sides
for value in range(3, max_result+1):
    frequency = results.count(value)
    frequencies.append(frequency)

# 可视化结果。
x_values = list(range(3, max_result+1))
data = [Bar(x=x_values, y=frequencies)]

x_axis_config = {'title': 'Result', 'dtick': 1}
y_axis_config = {'title': 'Frequency of Result'}
my_layout = Layout(title='Results of rolling three D6 dice 1,000,000 times',
       xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='3d6.html')
```

输出：



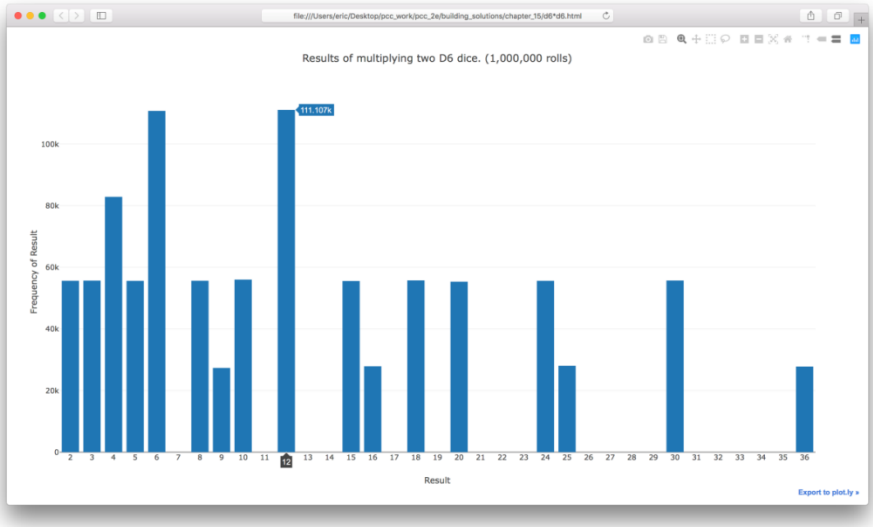
练习 15-8　将点数相乘

同时掷两个骰子时，通常将它们的点数相加。请通过可视化展示将两个骰子的点数相乘的结果。

```
from plotly.graph_objs import Bar, Layout
from plotly import offline

from die import Die

# 创建两个 D6 骰子。
die_1 = Die()
die_2 = Die()

# 掷骰子多次，并将结果存储在一个列表中。
results = []
for roll_num in range(1_000_000):
    result = die_1.roll() * die_2.roll()
    results.append(result)

# 分析结果。
frequencies = []
max_result = die_1.num_sides * die_2.num_sides
for value in range(1, max_result+1):
```

```
    frequency = results.count(value)
    frequencies.append(frequency)

# 可视化结果。
x_values = list(range(1, max_result+1))
data = [Bar(x=x_values, y=frequencies)]

x_axis_config = {'title': 'Result', 'dtick': 1}
y_axis_config = {'title': 'Frequency of Result'}
my_layout = Layout(
        title='Results of multiplying two D6 dice. (1,000,000 rolls)',
        xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d6*d6.html')
```

输出：



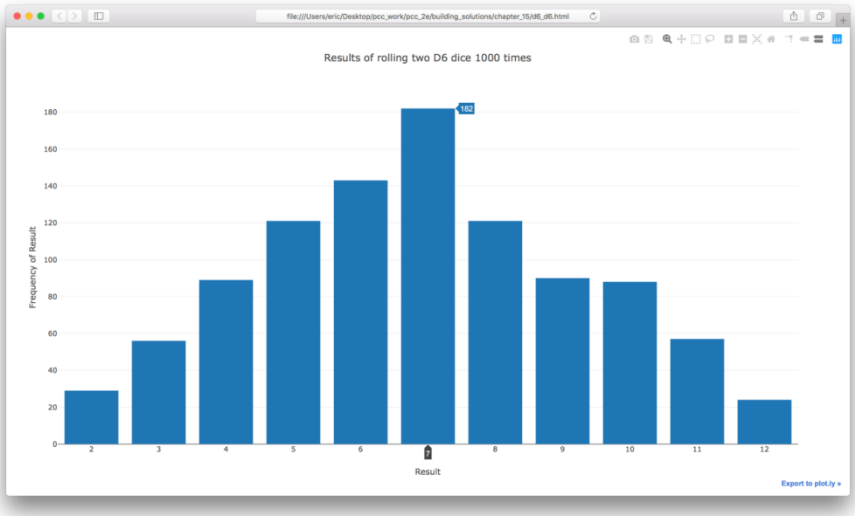
练习 15-9 改用列表解析

为清晰起见，本节模拟掷骰子的结果时，使用的是较长的 `for` 循环。如果你熟悉列表解析，尝试将这些程序中的一个或两个 `for` 循环改为列表解析。

```
from plotly.graph_objs import Bar, Layout
```

```
from plotly import offline

from die import Die

# 创建两个 D6 骰子。
die_1, die_2 = Die(), Die()

# 掷骰子多次，并将结果存储在一个列表中。
results = [die_1.roll() + die_2.roll() for roll_num in range(1000)]

# 分析结果。
max_result = die_1.num_sides + die_2.num_sides
frequencies = [results.count(value) for value in range(2, max_result+1)]

# 可视化结果。
x_values = list(range(2, max_result+1))
data = [Bar(x=x_values, y=frequencies)]

x_axis_config = {'title': 'Result', 'dtick': 1}
y_axis_config = {'title': 'Frequency of Result'}
my_layout = Layout(title='Results of rolling two D6 dice 1000 times',
       xaxis=x_axis_config, yaxis=y_axis_config)
offline.plot({'data': data, 'layout': my_layout}, filename='d6_d6.html')
```

输出：



第 16 章

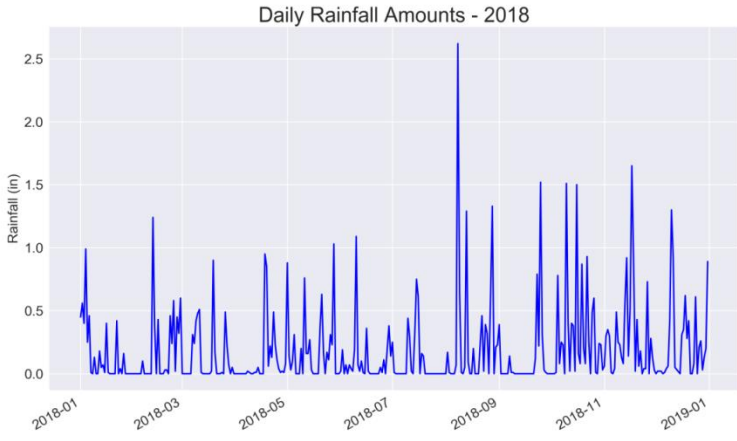
练习 16-1　锡特卡的降雨量

锡特卡属于温带雨林，降水量非常丰富。在数据文件 sitka_weather_2018_simple.csv 中，文件头 PRCP 表示的是每日降水量，请对这列数据进行可视化。如果你想知道沙漠的降水量有多低，可针对死亡谷完成这个练习。

```
import csv
from datetime import datetime

from matplotlib import pyplot as plt

filename = 'data/sitka_weather_2018_simple.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)

    # 从文件中获取日期和降雨量。
    dates, precips = [], []
    for row in reader:
        current_date = datetime.strptime(row[2], '%Y-%m-%d')
        dates.append(current_date)
        precip = float(row[3])
```

```
        precips.append(precip)

# 绘制降雨量图形。
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(dates, precips, c='blue')

# 设置图形的格式。
plt.title("Daily Rainfall Amounts - 2018", fontsize=24)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Rainfall (in)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)

plt.show()
```

输出：



练习 16-2　比较锡特卡和死亡谷的温度

在有关锡特卡和死亡谷的图表中，气温刻度反映了数据范围的不同。为准确地比较锡特卡和死亡谷的气温范围，需要在 *y* 轴上使用相同的刻度。为此，请修改图 16-5 和图 16-6 所示图表的 *y* 轴设置，对锡特卡和死亡谷的气温范围进行直接比较（你也可以对任何两个地方的气温范围进行比较）。

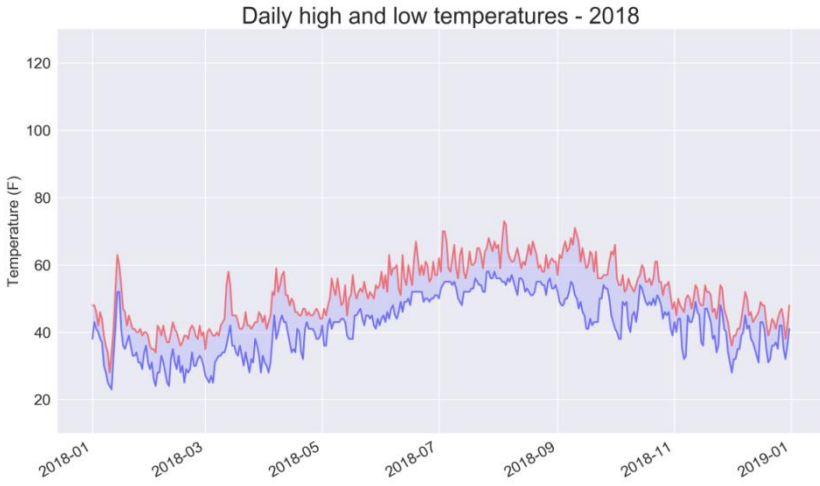
要设置 *y* 轴的取值范围，可使用 `pyplot` 函数 `ylim()`；要设置 *x* 轴的取值范围，可使

用函数 `xlim()`。

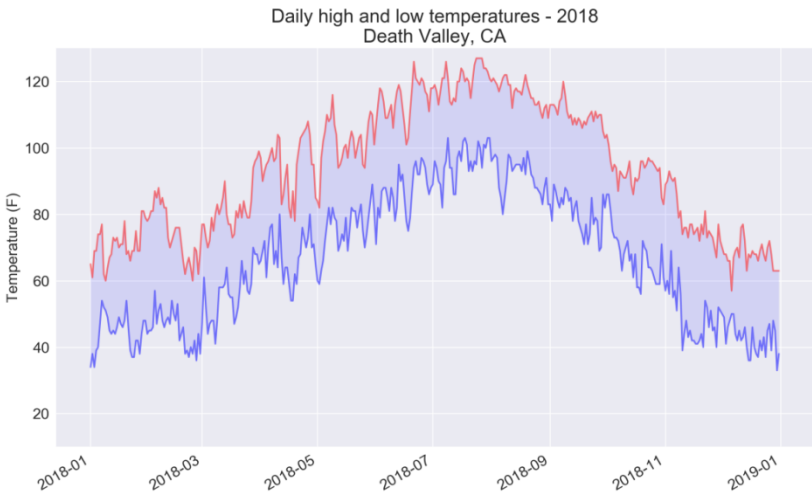
```
import csv
from datetime import datetime

from matplotlib import pyplot as plt

filename = 'data/sitka_weather_2018_simple.csv'
with open(filename) as f:
    reader = csv.reader(f)
    header_row = next(reader)

    # 从文件中获取日期、最高气温和最低气温。
    dates, highs, lows = [], [], []
    for row in reader:
        current_date = datetime.strptime(row[2], '%Y-%m-%d')
        high = int(row[5])
        low = int(row[6])
        dates.append(current_date)
        highs.append(high)
        lows.append(low)

# 绘制显示最高气温和最低气温的图形。
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(dates, highs, c='red', alpha=0.5)
ax.plot(dates, lows, c='blue', alpha=0.5)
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)

# 设置图形的格式。
plt.title("Daily high and low temperatures - 2018", fontsize=24)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)
plt.ylim(10, 130)
```

```
plt.show()
```

输出：



根据死亡谷的气温数据绘制图形时，通过使用函数`ylim()`指定同样的取值范围，可让两个图形的 y 轴刻度相同：



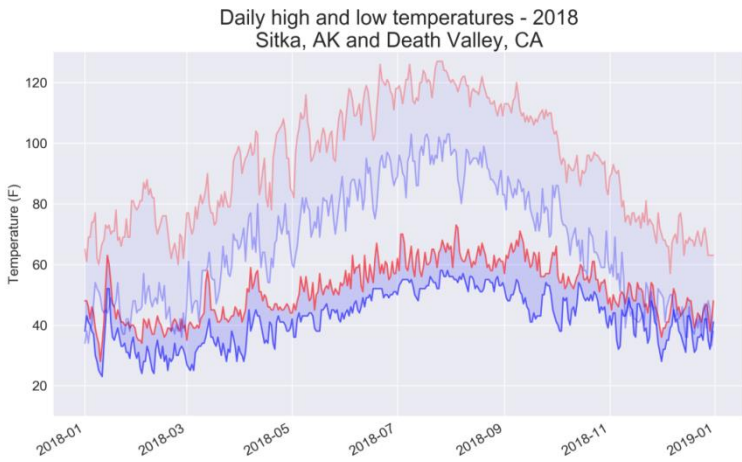
在同一个图表中呈现两个数据集的方法很多。在下面的解决方案中，我将读取 csv 文

件的代码放在一个函数中。然后调用这个函数来获取锡特卡的最高气温和最低气温，并绘制图表；接下来，我再次调用这个函数获取死亡谷的数据，并将它们添加到既有的图表中。我稍微调整了颜色，以便将这两个地方的数据区分开来。

```
import csv
from datetime import datetime

from matplotlib import pyplot as plt

def get_weather_data(filename, dates, highs, lows, date_index, high_index,
        low_index):
    """从数据文件中获取最高气温和最低气温。"""
    with open(filename) as f:
        reader = csv.reader(f)
        header_row = next(reader)

        # 从文件中获取日期、最高气温和最低气温。
        for row in reader:
            current_date = datetime.strptime(row[date_index], '%Y-%m-%d')
            try:
                high = int(row[high_index])
                low = int(row[low_index])
            except ValueError:
                print(f"Missing data for {current_date}")
            else:
                dates.append(current_date)
                highs.append(high)
                lows.append(low)

# 获取锡特卡的气温数据。
filename = 'data/sitka_weather_2018_simple.csv'
dates, highs, lows = [], [], []
get_weather_data(filename, dates, highs, lows, date_index=2, high_index=5,
        low_index=6)

# 根据锡特卡的数据绘制图表。
plt.style.use('seaborn')
fig, ax = plt.subplots()
```

```
ax.plot(dates, highs, c='red', alpha=0.6)
ax.plot(dates, lows, c='blue', alpha=0.6)
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.15)

# 获取死亡谷的数据。
filename = 'data/death_valley_2018_simple.csv'
dates, highs, lows = [], [], []
get_weather_data(filename, dates, highs, lows, date_index=2, high_index=4,
        low_index=5)

# 将死亡谷的数据添加到当前图表中。
ax.plot(dates, highs, c='red', alpha=0.3)
ax.plot(dates, lows, c='blue', alpha=0.3)
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.05)

# 设置图表的格式。
title = "Daily high and low temperatures - 2018"
title += "\nSitka, AK and Death Valley, CA"
plt.title(title, fontsize=24)
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)
plt.ylim(10, 130)

plt.show()
```

输出：



练习 16-4　自动索引

本节以硬编码的方式指定了 `TMIN` 和 `TMAX` 列的索引。请根据文件头行确定这些列的索引，让程序能够同时适用于锡特卡和死亡谷。另外，根据气象站的名称自动生成图表的标题。

方法 `index()` 返回列表中特定元素的索引，如下面的代码所示：

```
>>> animals = ['cat', 'dog', 'mouse', 'elephant']
>>> animals.index('dog')
1
```

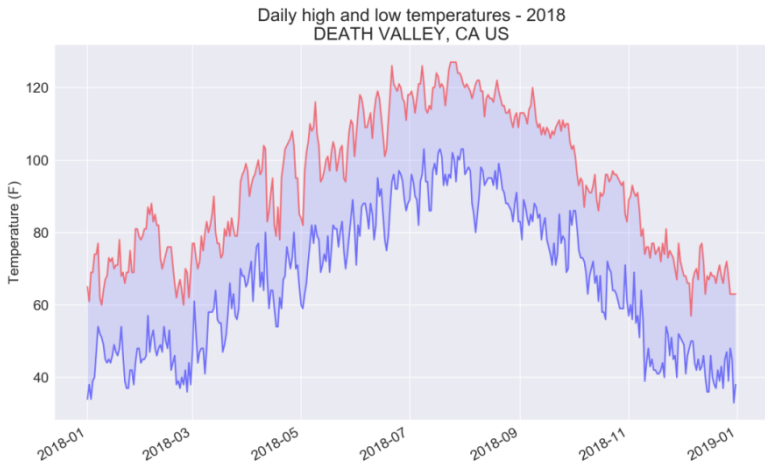
使用这个方法可获取文件头行中特定文件头的索引。

```
import csv
from datetime import datetime

from matplotlib import pyplot as plt

filename = 'data/death_valley_2018_simple.csv'
filename = 'data/sitka_weather_2018_simple.csv'
place_name = ''
with open(filename) as f:
```

```
    reader = csv.reader(f)
    header_row = next(reader)

    print(header_row)
    date_index = header_row.index('DATE')
    high_index = header_row.index('TMAX')
    low_index = header_row.index('TMIN')
    name_index = header_row.index('NAME')

    # 从文件中获取日期、最高气温和最低气温。
    dates, highs, lows = [], [], []
    for row in reader:
        # 如果没有设置气象站的名称，就获取它。
        if not place_name:
            place_name = row[name_index]
            print(place_name)

        current_date = datetime.strptime(row[date_index], '%Y-%m-%d')
        try:
            high = int(row[high_index])
            low = int(row[low_index])
        except ValueError:
            print(f"Missing data for {current_date}")
        else:
            dates.append(current_date)
            highs.append(high)
            lows.append(low)

# 绘制显示最高气温和最低气温的图形。
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(dates, highs, c='red', alpha=0.5)
ax.plot(dates, lows, c='blue', alpha=0.5)
plt.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)

# 设置图形的格式。
title = f"Daily high and low temperatures - 2018\n{place_name}"
plt.title(title, fontsize=20)
```

```
plt.xlabel('', fontsize=16)
fig.autofmt_xdate()
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major', labelsize=16)

plt.show()
```

输出：



第 17 章

练习 17-1　其他语言

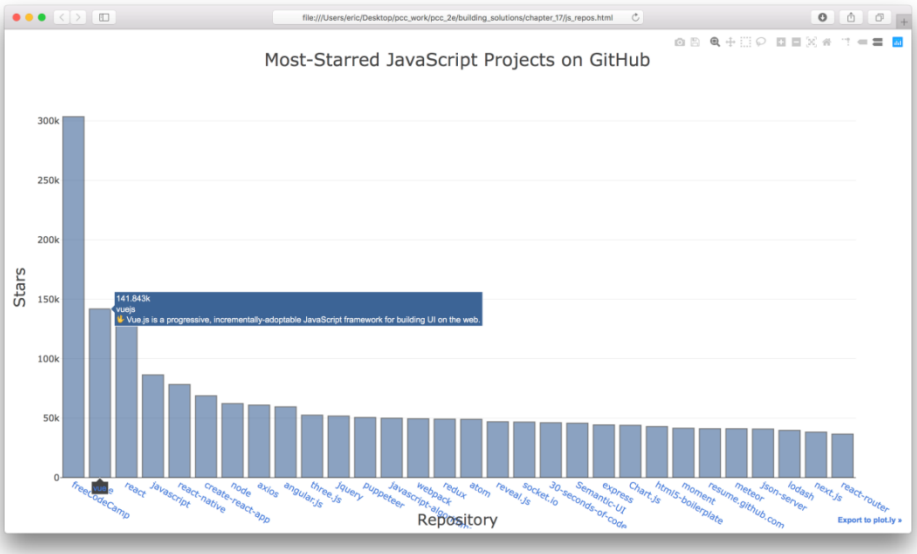
修改 python_repos.py 中的 API 调用，使其在生成的图表中显示使用其他语言编写的最受欢迎的项目。请尝试语言 JavaScript、Ruby、C、Java、Perl、Haskell 和 Go。

```
import requests

from plotly.graph_objs import Bar
from plotly import offline

# 执行 API 调用并存储响应。
url = 'https://api.github.com/search/repositories?q=language:javascript&sort=stars'
headers = {'Accept': 'application/vnd.github.v3+json'}
r = requests.get(url, headers=headers)
print(f"Status code: {r.status_code}")
```

```
# 处理结果。
response_dict = r.json()
repo_dicts = response_dict['items']
repo_links, stars, labels = [], [], []
for repo_dict in repo_dicts:
    repo_name = repo_dict['name']
    repo_url = repo_dict['html_url']
    repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
    repo_links.append(repo_link)

    stars.append(repo_dict['stargazers_count'])

    owner = repo_dict['owner']['login']
    description = repo_dict['description']
    label = f"{owner}<br />{description}"
    labels.append(label)


# 可视化。
data = [{
    'type': 'bar',
    'x': repo_links,
    'y': stars,
    'hovertext': labels,
    'marker': {
        'color': 'rgb(60, 100, 150)',
        'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)'}
    },
    'opacity': 0.6,
}]

my_layout = {
    'title': 'Most-Starred JavaScript Projects on GitHub',
    'titlefont': {'size': 28},
    'xaxis': {
        'title': 'Repository',
        'titlefont': {'size': 24},
```

```
        'tickfont': {'size': 14},
    },
    'yaxis': {
        'title': 'Stars',
        'titlefont': {'size': 24},
        'tickfont': {'size': 14},
    },

}

fig = {'data': data, 'layout': my_layout}
offline.plot(fig, filename='js_repos.html')
```

输出：



练习 17-3　测试 python_repos.py

在 python_repos.py 中，我们打印 `status_code` 的值，以核实 API 调用是否成功了。请编写一个名为 test_python_repos.py 的程序，它使用 `unittest` 来断言 `status_code` 的值为 `200`。想想你还可做出哪些断言，如返回的条目（item）数符合预期、仓库总数超过特定

的值。

注意：编写测试可促使你以合理的方式组织代码，确保可对其进行测试。下面是修订后的python_repos.py，其执行的所有操作都是在4个函数中完成的：

```
import requests

from plotly.graph_objs import Bar
from plotly import offline

def get_response():
    """执行 API 调用，并返回响应。"""
    url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'
    headers = {'Accept': 'application/vnd.github.v3+json'}
    r = requests.get(url, headers=headers)
    return r

def get_repo_dicts(r):
    """返回一系列表示最受欢迎仓库的字典。"""
    response_dict = r.json()
    repo_dicts = response_dict['items']
    return repo_dicts

def get_project_data(repo_dicts):
    """提取有关项目的数据，以便用于可视化。"""
    repo_links, stars, labels = [], [], []
    for repo_dict in repo_dicts:
        repo_name = repo_dict['name']
        repo_url = repo_dict['html_url']
        repo_link = f"<a href='{repo_url}'>{repo_name}</a>"
        repo_links.append(repo_link)

        stars.append(repo_dict['stargazers_count'])

        owner = repo_dict['owner']['login']
        description = repo_dict['description']
        label = f"{owner}<br />{description}"
        labels.append(label)
```

```
    return repo_links, stars, labels

def make_visualization(repo_links, stars, labels):
    """可视化最受欢迎的项目。"""
    data = [{
        'type': 'bar',
        'x': repo_links,
        'y': stars,
        'hovertext': labels,
        'marker': {
            'color': 'rgb(60, 100, 150)',
            'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)'}
        },
        'opacity': 0.6,
    }]

    my_layout = {
        'title': 'Most-Starred Python Projects on GitHub',
        'titlefont': {'size': 28},
        'xaxis': {
            'title': 'Repository',
            'titlefont': {'size': 24},
            'tickfont': {'size': 14},
        },
        'yaxis': {
            'title': 'Stars',
            'titlefont': {'size': 24},
            'tickfont': {'size': 14},
        },

    }

    fig = {'data': data, 'layout': my_layout}
    offline.plot(fig, filename='python_repos.html')

if __name__ == '__main__':
    r = get_response()
    repo_dicts = get_repo_dicts(r)
```

```
    repo_links, stars, labels = get_project_data(repo_dicts)
    make_visualization(repo_links, stars, labels)
```

将函数调用放在一个`if`代码块中，确保当前文件被直接执行时才调用这些函数，而在这个文件被导入时不调用它们。

现在可以给这些函数编写测试了。在这里，我们做如下测试：获得的响应的状态码为200；在表示第一个仓库的字典中，包含我们期望在每个表示仓库的字典中都有的键。

```
import unittest

import python_repos as pr

class PythonReposTestCase(unittest.TestCase):
    """测试 python_repos.py。"""

    def setUp(self):
        """调用所有的函数，并分别测试各个方面。"""
        self.r = pr.get_response()
        self.repo_dicts = pr.get_repo_dicts(self.r)
        self.repo_dict = self.repo_dicts[0]
        self.repo_links, self.stars, self.labels = pr.get_project_data(
                self.repo_dicts)

    def test_get_response(self):
        """测试获得了有效的响应。"""
        self.assertEqual(self.r.status_code, 200)

    def test_repo_dicts(self):
        """测试获得了期望的数据。"""
        # 应获得 30 个描述仓库的字典。
        self.assertEqual(len(self.repo_dicts), 30)

        # 描述仓库的字典应包含必要的键。
        required_keys = ['name', 'owner', 'stargazers_count', 'html_url']
        for key in required_keys:
            self.assertTrue(key in self.repo_dict.keys())

if __name__ == '__main__':
```

```
    unittest.main()
```

输出：

```
..
----------------------------------------------------------------------
Ran 2 tests in 2.371s

OK
```