

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不止一种，这里列出的只是其中之一。

如果你在完成练习的过程中遇到困难，可前往 [Stack Overflow](#)、[r/learnpython](#) 或在本书读者群中提问，也可与作者联系。

第 2 章

练习 2.1 简单消息

将一条消息赋给变量，并将其打印出来。

simple_message.py

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

输出：

```
I love learning to use Python.
```

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。

simple_messages.py

```
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 2.3 个性化消息

用变量表示一个人的名字，并向其显示一条消息。显示的消息应非常简单，如下所示。

Hello Eric, would you like to learn some Python today?

personal_message.py

```
name = "eric"
msg = f"Hello {name.title()}, would you like to learn some Python today?"

print(msg)
```

输出：

Hello Eric, would you like to learn some Python today?

练习 2.4 调整名字的大小写

用变量表示一个人的名字，再分别以全小写、全大写和首字母大写的方式显示这个人名。

name_cases.py

```
name = "eric"

print(name.lower())
print(name.upper())
print(name.title())
```

输出：

```
eric
ERIC
Eric
```

练习 2.5 名言 1

找到你钦佩的名人说的一句名言，将这个名人的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."

famous_quote.py

```
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 2.6 名言 2

重复练习2.5，但用变量`famous_person`表示名人的姓名，再创建要显示的消息并将其赋给变量`message`，然后打印这条消息。

famous_quote_2.py

```
famous_person = "Albert Einstein"  
  
message = f'{famous_person} once said, "A person who never made a mistake  
never tried anything new."'   
  
print(message)
```

输出：

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

注意：这里定义变量`message`的代码行超过了正常的长度限制。为了消除这个问题，可使用第7章将介绍的`+=`运算符给字符串添加内容。采用这种做法时，程序代码类似于下面这样，但输出没有任何变化：

famous_quote_3.py

```
famous_person = "Albert Einstein"  
  
message = f'{famous_person} once said, "A person who never made a mistake'  
message += ' never tried anything new.'"   
  
print(message)
```

练习 2.7 删除人名中的空白

用变量表示一个人的名字，并在其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。

stripping_names.py

```
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 2.8 文件扩展名

Python提供了`removesuffix()`方法，其工作原理与`removeprefix()`很像。请将值'`python_notes.txt`'赋给变量`filename`，再使用`removesuffix()`方法来显示不包含扩展名的文件名，就像文件浏览器所做的那样。

file_extensions.py

```
filename = 'python_notes.txt'
simple_filename = filename.removesuffix('.txt')

print(simple_filename)
```

输出：

```
python_notes
```

练习 2.10 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。

favorite_number.py

```
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`。依次访问该列表的每个元素，从而将每个朋友的姓名都打印出来。

names.py

```
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，但不打印每个朋友的姓名，而是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。

greetings.py

```
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一起共进晚餐（无论是在世的还是故去的），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，邀请这些人都来与你共进晚餐。

guest_list.py

```
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，在程序末尾添加函数调用`print()`，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。

changing_guest_list.py

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

你刚找到了一张更大的餐桌，可容纳更多的嘉宾就座。请想想你还想邀请哪三位嘉宾。

以完成练习3.4或练习3.5时编写的程序为基础，在程序末尾添加函数调用`print()`，指出你找到了一张更大的餐桌。

- ❑ 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。

more_guests.py

```
# 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.")
```

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 缩短名单

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，打印一条你只能邀请两位嘉宾共进晚餐的消息。

❑ 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，让该嘉宾知道你很抱歉，无法邀请他来共进晚餐。

❑ 对于余下两位嘉宾中的每一位，都打印一条消息，指出他依然在受邀之列。

❑ 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，核实名单在程序结束时确实是空的。

shrinking_guest_list.py

```
# 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个你想去旅游的地方。

- ☐ 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，只管打印原始Python列表就好。
- ☐ 使用`sorted()`按字母顺序打印这个列表，不要修改它。
- ☐ 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，不要修改它。
- ☐ 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。

seeing_the_world.py

```
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，如下所示。

I like pepperoni pizza.

□ 在程序末尾添加一行代码（不包含在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，还有一个总结性的句子，如下所示。

I really love pizza!

pizzas.py

```
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 4.2 动物

想出至少三种有共同特征的动物，将其名称存储在一个列表中，再使用 `for` 循环将每种动物的名称打印出来。

- ❑ 修改这个程序，使其针对每种动物都打印一个句子，如下所示。

A dog would make a great pet.

- ❑ 在程序末尾添加一行代码，指出这些动物的共同之处，如打印下面这样的句子。

Any of these animals would make a great pet!

animals.py

```
animals = ["spider monkey", "lemur", "giraffe"]
```

```
# 打印每种动物
```



```
for animal in animals:
    print(animal)

print("\n")

# 针对每种动物打印一个句子
for animal in animals:
    print(f"A {animal} has a long tail.")

print("\nAll of these animals have long tails.")
```

输出:

```
spider monkey
lemur
giraffe
```

A spider monkey has a long tail.

A lemur has a long tail.

A giraffe has a long tail.

All of these animals have long tails.

练习 4.3 数到 20

使用一个 for 循环打印数 1~20 (含)。

counting_to_twenty.py

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

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

输出:

```
1
2
3
...
18
19
```

20

练习 4.5 100 万求和

summing_a_million.py

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

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

输出：

```
1
1000000
500000500000
```

练习 4.6 奇数

通过给`range()`函数指定第三个参数来创建一个列表，其中包含1~20的奇数；再使用一个`for`循环将这些数打印出来。

odd_numbers.py

```
odd_numbers = list(range(1, 20, 2))

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

输出：

```
1
3
5
...
15
17
19
```

练习 4.7 3 的倍数

创建一个列表，其中包含3~30内能被3整除的数，再使用一个`for`循环将这个列表中的数打

印出来。

threes.py

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

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

输出:

```
3
6
9
...
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。创建一个列表，其中包含前10个整数（1~10）的立方，再使用一个for循环将这些立方数打印出来。

cubes.py

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

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

输出:

```
1
8
27
...
512
729
1000
```

练习 4.9 立方推导式

使用列表推导式生成一个列表，其中包含前10个整数的立方。

cube_comprehension.py

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

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

输出：

```
1
8
27
...
512
729
1000
```

练习 4.11 你的比萨，我的比萨

在你为练习4.1编写的程序中，创建比萨列表的副本，并将其赋给变量`friend_pizzas`，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`循环来打印第一个列表；打印消息“My friend's favorite pizzas are:”，再使用一个`for`循环来打印第二个列表。核实新增的比萨被添加到了正确的列表中。

my_pizzas_your_pizzas.py

```
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 4.12 使用多个循环

在本节中，为节省篇幅，程序foods.py的每个版本都没有使用for循环来打印列表。请选择一个版本的foods.py，在其中编写两个for循环，将各个食品列表都打印出来。

more_loops.py

```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods[:]
```

```
my_foods.append('cannoli')
friend_foods.append('ice cream')
```

```
print("My favorite foods are:")
for food in my_foods:
    print(f"- {food}")
```

```
print("\nMy friend's favorite foods are:")
for food in friend_foods:
    print(f"- {food}")
```

输出:

```
My favorite foods are:
```

```
- pizza
- falafel
- carrot cake
- cannoli
```

```
My friend's favorite foods are:
```

```
- pizza
- falafel
- carrot cake
- ice cream
```

练习 4.13 自助餐

有一家自助式餐馆，只提供5种简单的食品。请想出5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5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。
- ❑ 餐馆调整菜单，替换了两种食品。请编写一行给元组变量赋值的代码，并使用一个for循环将新元组的每个元素都打印出来。

buffet.py

```
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 1

假设玩家在游戏中消灭了一个外星人，请创建一个名为`alien_color`的变量，并将其赋值为`'green'`、`'yellow'`或`'red'`。

- 编写一条`if`语句，测试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，上述条件测试在其中的一个版本中通过，在另一个版本中未通过（未通过条件测试时没有输出）。

能够通过测试的版本:

alien_colors_1.py

```
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_colors_1_fail.py

```
alien_color = 'red'

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

这个版本没有输出。

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，指出玩家因消灭该外星人获得了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，在一个版本中将执行if代码块，在另一个版本中将执行else代码块。

执行 if 代码块的版本：

alien_colors_2_if_block.py

```
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_colors_2_else_block.py

```
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，分别在外星人为绿色、黄色和红色时打印一条消息。

alien_colors_3.py

```
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!")
```

外星人为红色时的输出:

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的值判断这个人处于人生的哪个阶段。

- ❑ 如果年龄小于2岁，就打印一条消息，指出这个人是婴儿。
- ❑ 如果年龄为2（含）~4岁，就打印一条消息，指出这个人是幼儿。
- ❑ 如果年龄为4（含）~13岁，就打印一条消息，指出这个人是儿童。
- ❑ 如果年龄为13（含）~18岁，就打印一条消息，指出这个人是少年。
- ❑ 如果年龄为18（含）~65岁，就打印一条消息，指出这个人是中年人。
- ❑ 如果年龄达到65岁，就打印一条消息，指出这个人是老年人。

stages_of_life.py

```
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 < 18:
    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`，并让其包含三种水果。
- ❑ 编写5条if语句，每条都检查某种水果是否在列表中。如果是，就打印一条像下面这样的消息。

You really like bananas!

favorite_fruits.py

```
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，并且其中一个用户名为 'admin'。想象你要编写代码，在每个用户登录网站后都打印一条问候消息。遍历用户名列表，向每个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，如下所示。

Hello admin, would you like to see a status report?

- ❑ 否则，打印一条普通的问候消息，如下所示。

Hello Jaden, thank you for logging in again.

hello_admin.py

```
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 login 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，就打印如下消息。

We need to find some users!

- ❑ 删除列表中的所有用户名，确认将打印正确的消息。

no_users.py

```
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

按照下面的说明编写一个程序，模拟网站如何确保每个用户的用户名都独一无二。

- ❑ 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在列表`current_users`中。
- ❑ 遍历列表`new_users`，检查其中的每个用户名是否已被使用。如果是，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。
- ❑ 确保比较时不区分大小写。换句话说，如果用户名'John'已被使用，应拒绝用户名'JOHN'。（为此，需要创建列表`current_users`的副本，其中包含当前所有用户名的全小写版本。）

checking_usernames.py

```
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

序数表示顺序位置，如1st和2nd。序数大多以th结尾，只有1st、2nd、3rd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，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"，每个序数都独占一行。

ordinal_numbers.py

```
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.py

```
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 1

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。再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。为了让这个程序更有趣，通过询问朋友确保数据是真实的。

favorite_numbers.py

```
favorite_numbers = {
    'mandy': 42,
    'micah': 23,
    'gus': 7,
```

```
'hank': 1_000_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 1

Python字典可用于模拟现实生活中的字典。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个编程术语，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术语及其含义。为此，可以先打印术语，在它后面加上一个冒号，再打印其含义；也可以先在一行里打印术语，再使用换行符（\n）插入一个空行，然后在下一行里以缩进的方式打印其含义。

glossary.py

```
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，请整理你为练习6.3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。确保该循环正确无误后，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_2.py

```
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:

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.

练习 6.5 河流

创建一个字典，在其中存储三条河流及其流经的国家。例如，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，如下所示。
The Nile runs through Egypt.
- ❑ 使用循环将该字典中每条河流的名字打印出来。
- ❑ 使用循环将该字典包含的每个国家的名字打印出来。

rivers.py

```
rivers = {  
    'nile': 'egypt',  
    'mississippi': 'united states',  
    'fraser': 'canada',  
    '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 Nile flows through Egypt.  
The Mississippi flows through United States.  
The Fraser flows through Canada.  
The Yangtze flows through China.
```

The following rivers are included in this data set:

```
- Nile  
- Mississippi  
- Fraser  
- Yangtze
```

The following countries are included in this data set:

- Egypt
- United States
- Canada
- China

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的名单，其中有些人已在字典中，而其他人不在字典中。
- ❑ 遍历这个名单。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，打印一条邀请参加调查的消息。

polling.py

```
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.
Edward's favorite language is Ruby.
Phil's favorite language is Python.
```

```
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

在为练习6.1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。

people.py

```
# 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

创建多个表示宠物的字典，每个字典都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打印出来。

pets.py

```
# 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: bugs

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，并存储每个人喜欢的1~3个地方。为让这个练习更有趣些，让一些朋友说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.py

```
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

修改为练习6.2编写的程序，让每个人都可以有多个喜欢的数字，然后将每个人的名字及其

喜欢的数打印出来。

favorite_numbers_6_10.py

```
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：

Mandy likes the following numbers:

42
17

Micah likes the following numbers:

42
39
56

Gus likes the following numbers:

7
12

练习 6.11 城市

创建一个名为 `cities` 的字典，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。表示每座城市的字典都应包含 `country`、`population` 和 `fact` 等键。将每座城市的名字以及相关信息都打印出来。

cities.py

```
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：VS Code并非在任何情况下都能运行提示用户输入的程序。在VS Code中运行提示用户输入的程序时，如果遇到麻烦，请参阅本书配套资源“安装说明”中的“运行使用input()的程序”一节。

练习 7.1 汽车租赁

编写一个程序，询问用户要租什么样的汽车，并打印一条消息，如下所示。

Let me see if I can find you a Subaru.

rental_car.py

```
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个人，就打印一条消息，指出没有空桌；否则指出有空桌。

restaurant_seating.py

```
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。

multiples_of_ten.py

```
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，指出要在比萨中添加这种配料。

pizza_toppings.py

```
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（含）~12岁的观众收费10美元；年满12岁的观众收费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。

movie_tickets.py

```
prompt = "\nHow 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 < 12:
        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. 11
    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，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`中。当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。

deli.py

```
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

使用为练习7.8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：先打印一条消息，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_sandwiches`中未包含'`pastrami`'。

no_pastrami.py

```
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 梦想中的度假胜地

编写一个程序，调查用户梦想中的度假胜地。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。

```
dream_vacation.py  
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，让它打印一个句子，指出本章的主题是什么。调用这个函数，确认显示的消息正确无误。

message.py

```
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 喜欢的书

favorite_book.py

```
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，它接受一个尺码以及要印到T恤上的字样。这个函数应该打印一个句子，简要地说明T恤的尺码和字样。

先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。

t_shirt.py

```
def make_shirt(size, message):
    """概述要制作的T恤是什么样的"""
    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`make_shirt()`函数，使其在默认情况下制作一件印有“I love Python”字样的大号T恤。调用这个函数分别制作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，以及一件印有其他字样的T恤（尺码无关紧要）。

large_shirts.py

```
def make_shirt(size='large', message='I love Python!'):
    """概述要制作的T恤是什么样的"""
    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字以及该城市所属的国家。这个函数应该打印一个像下面这样简单的句子。

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，其中至少有一座城市不属于默认的国家。

cities.py

```
def describe_city(city, country='chile'):
    """描绘城市"""
    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，它接受城市的名称及其所属的国家。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，并打印它返回的值。

city_names.py

```
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名和专辑名，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`make_album()`函数添加一个默认值为`None`的可选形参，以便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：

album.py

```
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：

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

包含歌曲数的版本：

album_num_songs.py

```
def make_album(artist, title, num_songs=0):
    """创建一个包含专辑信息的字典"""
    album_dict = {
        'artist': artist.title(),
        'title': title.title(),
```

```
    }
    if num_songs:
        album_dict['num_songs'] = num_songs
    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', num_songs=8)
print(album)
```

输出:

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

练习 8.8 用户的专辑

在为练习8.7编写的程序中，编写一个while循环，让用户输入歌手名和专辑名。获取这些信息后，使用它们来调用make_album()函数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。

user_albums.py

```
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，这个函数会打印列表中的每条文本消息。

messages.py

```
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

在为练习8.9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 `send_messages()` 函数，再将两个列表都打印出来，确认把消息移到了正确的列表中。

sending_messages.py

```
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

修改为练习8.10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`send_messages()`函数后，将两个列表都打印出来，确认原始列表保留了所有的消息。

archived_messages.py

```
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。

sandwiches.py

```
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。在调用这个函数时，提供必不可少的信息，以及两个名值对，如颜色和选装配件。这个函数必须能够像下面这样调用：

```
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。

cars.py

```
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。

printing_functions.py

```
"""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，为其`__init__()`方法设置两个属性：`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。

restaurant.py

```
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

根据为练习9.1编写的类创建三个实例，并对每个实例调用`describe_restaurant()`方法。

three_restaurants.py

```
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，还有用户简介中通常会有其他几个属性。在类User中定义一个名为describe_user()的方法，用于打印用户信息摘要。再定义一个名为greet_user()的方法，用于向用户发出个性化的问候。

创建多个表示不同用户的实例，并对每个实例调用上述两个方法。

users.py

```
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

在为练习9.1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。

添加一个名为`set_number_served()`的方法，用来设置就餐人数。调用这个方法并向它传递新的就餐人数，然后再次打印这个值。

添加一个名为`increment_number_served()`的方法，用来让就餐人数递增。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。

number_served.py

```
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 = 500
print(f"Number served: {restaurant.number_served}")

restaurant.set_number_served(1000)
print(f"Number served: {restaurant.number_served}")

restaurant.increment_number_served(250)
print(f"Number served: {restaurant.number_served}")
```

输出：

The Mean Queen serves wonderful pizza.

```
Number served: 0
Number served: 500
Number served: 1000
Number served: 1250
```

练习 9.5 尝试登录次数

在为练习9.3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，用来将属性login_attempts的值加1。再编写一个名为reset_login_attempts()的方法，用来将属性login_attempts的值重置为0。

根据User类创建一个实例，再调用increment_login_attempts()方法多次。打印属性login_attempts的值，确认它正确地递增了。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。

login_attempts.py

```
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，让它继承你为练习9.1或练习9.4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口味的方法。创建一个IceCreamStand实例，并调用这个方法。

ice_cream_stand.py

```
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):
        """显示出售的冰激凌口味"""
        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，让它继承你为练习9.3或练习9.5完成编写的`User`类。添加一个名为`privileges`的属性，用来存储一个由字符串（如`"can add post"`、`"can delete post"`、`"can ban user"`等）组成的列表。编写一个名为`show_privileges()`的方法，显示管理员的权限。创建一个`Admin`实例，并调用这个方法。

admin.py

```
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所述的字符串列表。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()`来显示权限。

privileges.py

```
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 电池升级

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。这个方法检查电池容量，如果电池容量不是65，就设置为65。创建一辆电池容量为默认值的电动汽车，调用方法get_range()，然后对电池进行升级，并再次调用get_range()。你将看到这辆汽车的续航里程增加了。

battery_upgrade.py

```
class Car:
    """一次模拟汽车的简单尝试"""

    def __init__(self, make, model, year):
        """初始化描述汽车的属性"""
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """返回整洁的描述性名称"""
        long_name = f"{self.year} {self.make} {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"""
        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池的简单尝试"""
```

```
def __init__(self, battery_size=40):
    """初始化电池的属性"""
    self.battery_size = battery_size

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

def get_range(self):
    """打印一条消息，指出电池的续航里程"""
    if self.battery_size == 40:
        range = 150
    elif self.battery_size == 65:
        range = 225

    print(f"This car can go about {range} miles on a full charge.")

def upgrade_battery(self):
    """在可能的情况下将电池升级"""
    if self.battery_size == 40:
        self.battery_size = 65
        print("Upgraded the battery to 65 kWh.")
    else:
        print("The battery is already upgraded.")

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

    def __init__(self, make, 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__(make, model, year)
        self.battery = Battery()
```

```
print("Make an electric car, and check the range:")
my_leaf = ElectricCar('nissan', 'leaf', 2024)
my_leaf.battery.get_range()

print("\nUpgrade the battery, and check the range again:")
my_leaf.battery.upgrade_battery()
my_leaf.battery.get_range()
```

输出：

```
Make an electric car, and check the range:
This car can go about 150 miles on a full charge.

Upgrade the battery, and check the range again:
Upgraded the battery to 65 kWh.
This car can go about 225 miles on a full charge.
```

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类、Privileges类和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show_privileges()方法，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类和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show_privileges()方法，以确认一切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

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面的骰子并掷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。

dice.py

```

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:

```
[6, 1, 2, 1, 6, 6, 2, 5, 3, 4]
```

```
10 rolls of a 10-sided die:
```

```
[5, 2, 6, 7, 6, 8, 10, 6, 7, 10]
```

```
10 rolls of a 20-sided die:
```

```
[5, 1, 14, 4, 10, 13, 3, 2, 18, 20]
```

练习 9.14 彩票

创建一个列表或元素，其中包含10个数和5个字母。从这个列表或元组中随机选择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。

lottery.py

```
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:
        print(f" We pulled a {pulled_item}!")
        winning_ticket.append(pulled_item)

print(f"\nThe final winning ticket is: {winning_ticket}")
```

输出：

```
Let's see what the winning ticket is...
```

```
We pulled a 2!
```

```
We pulled a 3!
```

```
We pulled a 5!
```

```
We pulled a c!
```

```
The final winning ticket is: [2, 3, 5, 'c']
```

练习 9.15 彩票分析

可以使用一个循环来理解中前述彩票大奖有多难。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，报告执行多少次循环才中了大奖。

```
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: [1, 4, 'a', 9]
Winning ticket: [1, 9, 'a', 4]
```

It only took 731 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，读取这个文件，并将你所写的内容打印两次：第一次打印时读取整个文件；第二次打印时先将所有行都存储在一个列表中，再遍历列表中的各行。

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

```
from pathlib import Path  
  
print("--- Reading in the entire file:")  
path = Path('learning_python.txt')  
contents = path.read_text()  
print(contents)  
  
print("\n--- Looping over the lines:")  
lines = contents.splitlines()  
for line in lines:  
    print(line)
```

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.
```

练习 10.2 C 语言学习笔记

可使用`replace()`方法将字符串中的特定单词替换为另一个单词。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。

learning_c.py

```
from pathlib import Path

path = Path('learning_python.txt')
contents = path.read_text()

lines = contents.splitlines()
for line in lines:
    line = line.replace('Python', 'C')
    print(line)
```

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.
```

练习 10.3 简化代码

本节前面的程序`file_reader.py`中使用了一个临时变量`lines`，来说明`splitlines()`的工作原理。可省略这个临时变量，直接遍历`splitlines()`返回的列表：

```
for line in contents.splitlines():
```

对于本节的每个程序，都删除其中的临时变量，让代码更简洁。

simpler_code_file_reader.py

```
from pathlib import Path

path = Path('pi_digits.txt')
contents = path.read_text()
```



```
for line in contents.splitlines():
    print(line)
```

输出:

```
3.1415926535
8979323846
2643383279
```

simpler_code_pi_string.py

```
from pathlib import Path

path = Path('pi_million_digits.txt')
contents = path.read_text()

pi_string = ''
for line in contents.splitlines():
    pi_string += line.lstrip()

print(f"{pi_string[:52]}...")
print(len(pi_string))
```

输出:

```
3.14159265358979323846264338327950288419716939937510...
1000002
```

simpler_code_pi_birthday.py

```
from pathlib import Path

path = Path('pi_million_digits.txt')
contents = path.read_text()

pi_string = ''
for line in contents.splitlines():
    pi_string += line.lstrip()

birthday = input("Enter your birthday, in the form mmddyy: ")
if birthday in pi_string:
```

```
print("Your birthday appears in the first million digits of pi!")
else:
    print("Your birthday does not appear in the first million digits of pi.")
```

输出：

```
Enter your birthday, in the form mmddyy: 040122
Your birthday appears in the first million digits of pi!
```

练习 10.4 访客

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

guest.py

```
from pathlib import Path

path = Path('guest.txt')

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

输出：

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

guest.txt

```
eric
```

练习 10.5 访客簿

编写一个while循环，提示用户输入其名字。收集用户输入的所有名字，将其写入guest_book.txt，并确保这个文件中的每条记录都独占一行。

guest_book.py

```
from pathlib import Path

path = Path('guest_book.txt')

prompt = "\nHi, what's your name? "
prompt += "\nEnter 'quit' if you're the last guest. "

guest_names = []
```

```
while True:
    name = input(prompt)
    if name == 'quit':
        break

    print(f"Thanks {name}, we'll add you to the guest book.")
    guest_names.append(name)

# 创建一个字符串，它包含所有的名字，且在每个名字后面都换行
file_string = ''
for name in guest_names:
    file_string += f"{name}\n"

path.write_text(file_string)
```

输出：

```
Hi, what's your name?
Enter 'quit' if you're the last guest. eric
Thanks eric, we'll add you to the guest book.
```

```
Hi, what's your name?
Enter 'quit' if you're the last guest. erin
Thanks erin, we'll add you to the guest book.
```

```
Hi, what's your name?
Enter 'quit' if you're the last guest. ever
Thanks ever, we'll add you to the guest book.
```

```
Hi, what's your name?
Enter 'quit' if you're the last guest. willie
Thanks willie, we'll add you to the guest book.
```

```
Hi, what's your name?
Enter 'quit' if you're the last guest. Quit
```

guest_book.txt

```
eric
erin
```

```
ever  
willie
```

练习 10.6 加法运算

在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。在用户输入的任意一个值不是数时都捕获`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。

addition.py

```
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

将为练习10.6编写的代码放在一个`while`循环中，让用户在犯错（输入的是文本而不是数）后能够继续输入数。

addition_calculator.py

```
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FileNotFoundError异常，并显示一条友好的消息。将任意一个文件移到另一个地方，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

```
from pathlib import Path

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

for filename in filenames:
    print(f"\nReading file: {filename}")

    path = Path(filename)
    try:
        contents = path.read_text()
    except FileNotFoundError:
        print(" Sorry, I can't find that file.")
    else:
        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: 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。

`silent_cats_and_dogs.py`

```
from pathlib import Path

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

for filename in filenames:
    path = Path(filename)
    try:
        contents = path.read_text()
    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，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。

可以使用方法 `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()` 将字符串转换为全小写的，可捕捉要查找的单词的各种格式，而不管其大小写如何。

编写一个程序，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，因为诸如 `'then'` 和 `'there'` 等单词也被计算在内了。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

```
from pathlib import Path

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
path = Path(filename)
try:
    contents = path.read_text()
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 2528 times.
```

这里只计算了'the'在一部图书中出现的次数，也可以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，提示用户输入自己喜欢的数，并使用`json.dumps()`将这个数存储在文件中。再编写一个程序，从文件中读取这个值，并打印如下消息。

I know your favorite number! It's ____.

favorite_number_writer.py

```
from pathlib import Path
import json

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

path = Path('favorite_number.json')
contents = json.dumps(number)
path.write_text(contents)

print("Thanks! I'll remember that number.")
```

输出：

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

favorite_number_reader.py

```
from pathlib import Path
import json

path = Path('favorite_number.json')
contents = path.read_text()
number = json.loads(contents)

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

将你在完成练习10.11时编写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，否则提示用户输入自己喜欢的数并将其存储在文件中。运行这个程序两次，看看它是否像预期的那样工作。

favorite_number_remembered.py

```
from pathlib import Path
import json

path = Path('favorite_number.json')
try:
    contents = path.read_text()
except FileNotFoundError:
    number = input("What's your favorite number? ")
    contents = json.dumps(number)
    path.write_text(contents)
    print("Thanks, I'll remember that.")
else:
    number = json.loads(contents)
    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 用户字典

示例remember_me.py只存储了一项信息——用户名。请扩展该示例，让用户同时提供另外两项信息，再将收集到的所有信息存储到一个字典中。使用`json.dumps()`将这个字典写入文件，并使用`json.loads()`从文件中读取它。打印一条摘要消息，指出程序记住了有关用户的哪些信息。

user_dictionary.py

```
from pathlib import Path
import json

def get_stored_user_info(path):
    """获取存储的用户信息（如果有的话）"""
    if path.exists():
        contents = path.read_text()
        user_dict = json.loads(contents)
        return user_dict
    else:
        return None

def get_new_user_info(path):
    """从新用户那里获取信息"""
    username = input("What is your name? ")
    game = input("What's your favorite game? ")
    animal = input("What's your favorite animal? ")

    user_dict = {
        'username': username,
        'game': game,
        'animal': animal,
    }
```

```
contents = json.dumps(user_dict)
path.write_text(contents)
return user_dict

def greet_user():
    """根据用户的新老情况发出不同的问候，并在用户为老用户时显示其信息"""
    path = Path('user_info.json')
    user_dict = get_stored_user_info(path)
    if user_dict:
        print(f"Welcome back, {user_dict['username']}!")
        print(f"Hope you've been playing some {user_dict['game']}. ")
        print(f"Have you seen a {user_dict['animal']} recently?")
    else:
        user_dict = get_new_user_info(path)
        msg = f"We'll remember you when you return, {user_dict['username']}!"
        print(msg)

greet_user()
```

第一次运行时的输出：

```
What is your name? eric
What's your favorite game? chess
What's your favorite animal? mountain goat
We'll remember you when you return, eric!
```

第二次运行的输出：

```
Welcome back, eric!
Hope you've been playing some chess.
Have you seen a mountain goat recently?
```

练习 10.14 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。我们应修改这个程序，以防当前用户并非上次运行该程序的用户。

为此，在greet_user()中打印欢迎用户回来的消息之前，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。

verify_user.py

```
from pathlib import Path
```

```
import json

def get_stored_username(path):
    """获取存储的用户名（如果存储了）"""
    if path.exists():
        contents = path.read_text()
        username = json.loads(contents)
        return username
    else:
        return None

def get_new_username(path):
    """提示用户输入用户名"""
    username = input("What is your name? ")
    contents = json.dumps(username)
    path.write_text(contents)
    return username

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

verify_user_clean.py

```
def greet_user():
```

```
    """基于用户名问候用户"""
```

```
    path = Path('username.json')
```

```
    username = get_stored_username(path)
```

```
    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(path)
```

```
    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，并将这个文件存储在一个新的文件夹中，以免 `pytest` 在运行时，尝试运行之前编写的测试。

创建一个名为 `test_cities.py` 的程序，对刚编写的函数进行测试。编写一个名为 `test_city_country()` 的函数，核实在使用类似于 `'santiago'` 和 `'chile'` 这样的值来调用该函数时，得到的字符串是正确的。运行测试，确认 `test_city_country()` 通过了。

city_country/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。

city_country/test_cities.py

```
from city_functions import city_country

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

输出：

```
city_country $ pytest
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
```

```

rootdir: ../../solution_files/chapter_11/city_country
collected 1 item

test_cities.py . [100%]
===== 1 passed in 0.01s =====

```

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`。运行测试，确认 `test_city_country()` 未通过。

修改上述函数，将形参 `population` 设置为可选的。再次运行测试，确认 `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。再次运行测试，确认 `test_city_country_population()` 通过了。

population/city_functions.py

```
"""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

```

输出：

```

population $ pytest
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
rootdir: ../../solution_files/chapter_11/population
collected 1 item

test_cities.py F [100%]

===== FAILURES =====
_____ test_city_country _____

    def test_city_country():

```



```
        """Does a simple city and country pair work?"""
>     santiago_chile = city_country('santiago', 'chile')
E     TypeError: city_country() missing 1 required positional argument: 'population'

test_cities.py:5: TypeError
===== short test summary info =====
FAILED test_cities.py::test_city_country - TypeError: city_country() missin...
===== 1 failed in 0.09s =====
```

population/city_functions_optional.py

```
"""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 and 国家的字符串"""

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

输出:

```
population $ pytest
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
rootdir: /.../solution_files/chapter_11/population
collected 1 item

test_cities.py . [100%]
===== 1 passed in 0.01s =====
```

population/test_cities.py

```
from city_functions_pop_optional import city_country
```

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

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

注意：我在同一个文件夹中存储了模块 `city_functions.py` 的两个版本，因此这里修改了 `import` 语句，以使用第二个版本中的函数 `city_country()`。

输出：

```
population $ pytest
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
rootdir: ../../solution_files/chapter_11/population
collected 2 items

test_cities.py ..                                [100%]
===== 2 passed in 0.01s =====
```

练习 11.3 雇员

编写一个名为 `Employee` 的类，其 `__init__()` 方法接受名、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，同时能够接受其他的年薪增加量。

为 `Employee` 类编写一个测试文件，其中包含两个测试函数：

`test_give_default_raise()` 和 `test_give_custom_raise()`。在不使用夹具的情况下编写这两个测试，并确保它们都通过了。然后，编写一个夹具，以免在每个测试函数中都创建一个 `Employee` 对象。重新运行测试，确认两个测试都通过了。

employee/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
```

employee/test_employee.py

```
from employee import Employee
```

```
def test_give_default_raise():
    """测试使用默认的年薪增加量是否可行"""
    employee = Employee('eric', 'matthes', 65_000)
    employee.give_raise()
    assert employee.salary == 70_000

def test_give_custom_raise():
    """测试自定义年薪增加量是否可行"""
    employee = Employee('eric', 'matthes', 65_000)
    employee.give_raise(10000)
    assert employee.salary == 75_000
```

输出:

```
employee $ pytest
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
rootdir: ../../solution_files/chapter_11/employee
collected 2 items

test_employee.py .. [100%]
===== 2 passed in 0.01s =====
```

employee_with_fixture/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
```

employee_with_fixture/test_employee.py

```
import pytest

from employee import Employee

@pytest.fixture
def employee():
    """创建一个可供所有测试函数使用的 Employee 对象"""
    employee = Employee('eric', 'matthes', 65_000)
    return employee

def test_give_default_raise(employee):
    """测试使用默认的年薪增加量是否可行"""
    employee.give_raise()
    assert employee.salary == 70_000

def test_give_custom_raise(employee):
    """测试自定义年薪增加量是否可行"""
    employee.give_raise(10000)
    assert employee.salary == 75_000
```

输出：

```
employee_with_fixture $ pytest
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
rootdir: ../../solution_files/chapter_11/employee_with_fixture
collected 2 items

test_employee.py .. [100%]
===== 2 passed in 0.01s =====
```

第 12 章

完成第 12~14 章的练习时，知道下面几点会有所帮助。

- 对于第 12~14 章的练习，答案放在文件夹 `solution_files` 中，因为每个练习都是一个微型项目。
- 在完成项目的过程中，如果犯错后不能回到之前的可行状态，必须从头再来，就太让人沮丧了。有一些资源可帮助你解决这个问题。
 - 在源代码文件中，第 12~14 章中的每节结束时，都提供了项目“外星人入侵”的完整代码。
 - 例如，当你编写代码让飞船移动时，如果项目不能正常运行，可进入与第 12 章相关的文件夹（源代码文件/`chapter_12`），再单击文件夹 `adding_ship_image`，从而获得 12.4 节开始时项目的完整源代码。
 - 如果要将你编写的代码与 12.6 节结束时的项目代码进行比较，可打开文件夹 `piloting_the_ship` 查看。
 - 如果你想知道如何拍摄项目快照，请阅读附录 D，其中介绍的知识非常值得你花时间去学习，而且在你的整个程序员生涯中都将用到。
- 配套资源提供了多个速查表，其中有一个是针对 Pygame 的（`beginners_python_cheat_sheet_pcc_pygame.pdf`），可能能够在你完成这些练习时提供帮助。
- 完成这些练习时，查看 Pygame 文档（<https://www.pygame.org/docs/>）可能会有所帮助。

练习 12.1 蓝色的天空

创建一个背景为蓝色的 Pygame 窗口。

答案见 `ex_12_1_blue_sky`。

练习 12.2 游戏角色

找一幅你喜欢的游戏角色的位图图像或将一幅图像转换为位图。创建一个类，将该角色绘制到屏幕中央，并将该图像的背景色设置为屏幕的背景色或将屏幕的背景色设置为该图像的背景色。

答案见 `ex_12_2_game_character`。

练习 12.4 火箭

编写一个游戏，它在屏幕中央显示一艘火箭，而玩家可使用上下左右四个方向键移动火箭。务必确保火箭不会移动到屏幕之外。

答案见 `ex_12_4_rocket`。

练习 12.5 按键

编写一个创建空屏幕的Pygame文件。在事件循环中，每当检测到`pygame.KEYDOWN`事件时都打印属性`event.key`。运行这个程序并按下不同的键，看看控制台窗口的输出，以便了解Pygame会如何响应。

注意：查看有关`pygame.key`的文档（<https://www.pygame.org/docs/ref/key.html>）对完成这个练习会有所帮助。另外，如果你运行这里提供的代码，得到的将是按键的整型编码。这是意料之中的，虽然使用的是`pygame.K_q`、`pygame.K_SPACE`等常量，但它们会被映射到相应的整型值。

答案见 `ex_12_5_keys`。

练习 12.6 《横向射击》

编写一个游戏，将一艘飞船放在屏幕左侧，并允许玩家上下移动飞船。在玩家按空格键时，让飞船发射一颗在屏幕中向右飞行的子弹，并在子弹从屏幕中消失后将其删除。

答案见 `ex_12_6_sideways_shooter`。

第 13 章

练习 13.1 星星

找一幅星星图像，并在屏幕上显示一系列排列整齐的星星。

答案见 `ex_13_1_stars`。

练习 13.2 更逼真的星星

为让星星的分布更逼真，可随机地放置星星。第9章说过，可像下面这样生成随机数：

```
from random import randint
random_number = randint(-10, 10)
```

上述代码返回一个-10和10之间的随机整数。在为练习13.1编写的程序中，使用该方法随机地调整每颗星星的位置吧。

下面是一种可行的基本答案。如果你愿意，可尝试使用不同的星星尺寸、间隔值和星星随机位置的取值范围。我所做的尝试表明，这种方案适合创建相对密集的小型星空。

答案见 `ex_13_2_better_stars`。

下面是一种完全不同的随机星空创建方法。这里没有使用偏移量来创建网格，并且随机地确定每颗星星在屏幕上的位置。

答案见 `ex_13_2_better_stars_alternate`。

练习 13.3 雨滴

寻找一幅雨滴图像，并创建一系列整齐排列的雨滴。让这些雨滴往下落，直到到达屏幕的下边缘后消失。

答案见 `ex_13_3_raindrops`。

练习 13.4 连绵细雨

修改为练习11.3编写的代码，使得当一行雨滴消失在屏幕的下边缘后，在屏幕上边缘附近又出现一行新雨滴，并开始往下落。

这里提供的也是一种简单可行的解决方案，未经优化。如果你喜欢这个练习，建议尝试不同的雨滴尺寸、确定雨滴初始位置的不同方法以及创建新雨滴行的不同方法。

答案见 `ex_13_4_steady_rain`。

练习 13.5 改进版《横向射击》

完成练习12.6后，我们给游戏《外星人入侵》添加了很多功能。在本练习中，请尝试给《横向射击》添加类似的功能。添加一个外星舰队（或让外星人的位置随机），并让其向飞船移动。另外，编写让被子弹击中的外星人消失的代码。

答案见 `ex_13_5_sideways_shooter_2`。

练习 13.6 游戏结束

在游戏《横向射击》中，记录飞船被撞到了多少次以及有多少个外星人被击落了。确定合适的游戏结束条件，并在满足该条件后结束游戏。

答案见 `ex_13_6_game_over`。

第 14 章

练习 14.1 按 P 键开始新游戏

鉴于游戏《外星人入侵》使用键盘来控制飞船，最好让玩家也能够通过按键来开始游戏。请添加在玩家按P键时开始游戏的代码。也许这样做会有所帮助：将 `_check_play_button()` 的一些代码提取出来，放到一个名为 `_start_game()` 的方法中，并在 `_check_play_button()` 和 `_check_keydown_events()` 中调用这个方法。

答案见 `ex_14_1_p_to_play`。

练习 14.2 射击练习

创建一个矩形，让它在屏幕右边缘以固定的速度上下移动。然后，在屏幕左边缘创建一艘飞船，玩家可上下移动飞船并射击前述矩形目标。添加一个用于开始游戏的Play按钮，在玩家三次未击中目标时结束游戏，并重新显示Play按钮，让玩家能够单击该按钮来重新开始游戏。

答案见 `ex_14_2_target_practice`。

练习 14.3 有一定难度的射击练习

以你为完成练习14.2而做的工作为基础，让标靶的移动速度随游戏进行而加快，并在玩家单击Play按钮时将其重置为初始值。

答案见 `ex_14_3_challenging_tp`。

练习 14.4 难度等级

在游戏《外星人入侵》中创建一组按钮，让玩家选择起始难度等级。每个按钮都给Settings中的属性指定合适的值，以实现相应的难度等级。

这里提供了两个答案。在第一个答案（`ex_14_4_difficulty_levels`）中，难度等级按钮比较简单，被单击时它们只改变游戏设置，而自身的颜色不变。第二个答案（`ex_14_4_difficulty_levels_toggle`）更复杂些，它突出被单击的难度等级按钮，以指出当前的难度等级设置。

练习 14.5 历史最高分

每当玩家关闭并重新开始游戏《外星人入侵》时，最高分都将被重置。请这样修复该问题：调用`sys.exit()`前将最高分写入文件，并在GameStats中初始化最高分时从文件中读取它。

答案见 `ex_14_5_high_score`。

练习 14.6 重构

找出执行多项任务的方法，对它们进行重构，让代码高效而有序。例如，对于`_check_bullet_alien_collisions()`，将在整个外星舰队被全部击落时开始新等级的代码移到一个名为`start_new_level()`的方法中。又例如，对于Scoreboard的`__init__()`方法，将调用四个不同方法的代码移到一个名为`prep_images()`的方法中，以缩短`__init__()`方法。如果你重构了`_check_play_button()`，`prep_images()`方法也可帮助简化`_check_play_button()`和`_start_game()`。

注意：重构项目前，请阅读附录D，了解如果重构时引入了bug，如何将项目恢复到可正确运行的状态。

答案见 `ex_14_6_refactoring`。

练习14.7 扩展游戏《外星人入侵》

想想如何扩展游戏《外星人入侵》。例如，让外星人也能够向飞船射击，或者为飞船添加盾牌，使得只有从两边射来的子弹才能摧毁飞船。另外，还可以使用像`pygame.mixer`这样的模块来添加声音效果，如爆炸声和射击声。

有关如何添加声音，请参阅这里（https://ehmatthes.github.io/pcc_2e/beyond_pcc/ai_player/）。还可将玩游戏的过程自动化（https://ehmatthes.github.io/pcc_2e/beyond_pcc/ai_player/#automating-game-play），进而看着计算机玩《外星人入侵》游戏。

第15章

练习15.1 立方

数的三次方称为**立方**。请先绘图显示前5个正整数的立方值，再绘图显示前5000个正整数的立方值。

cubes_5.py

```
import matplotlib.pyplot as plt

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

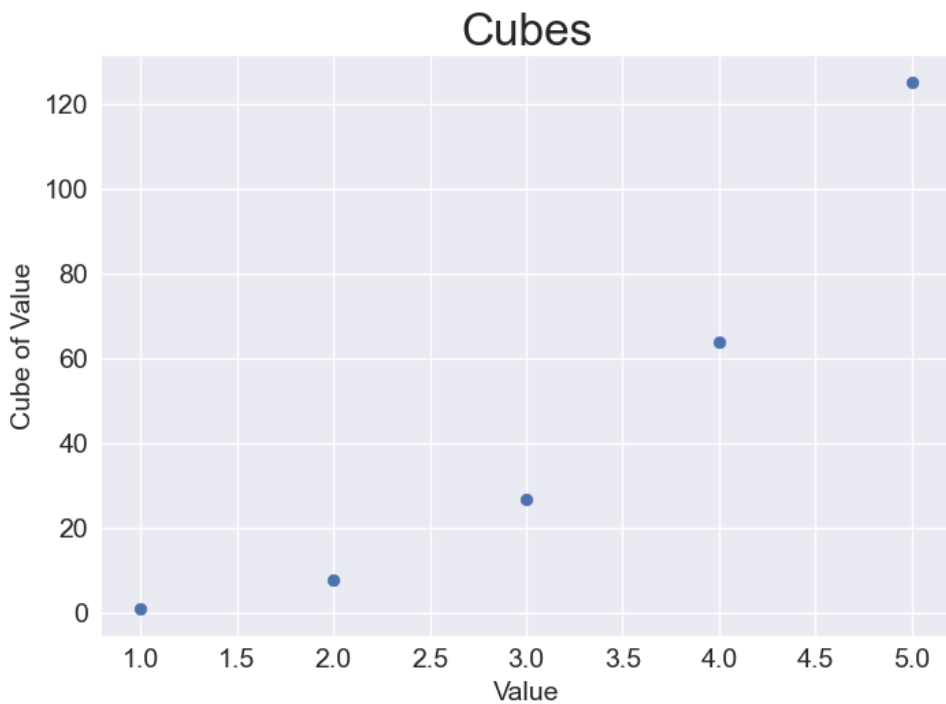
# 创建绘图
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.scatter(x_values, cubes, s=40)

# 设置图题和坐标轴标签
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)

# 显示绘图
plt.show()
```

输出：



绘制前 5000 个整数的立方值：

cubes_5000.py

```
import matplotlib.pyplot as plt

# 定义数据
x_values = range(1, 5001)
y_values = [x**3 for x in x_values]

# 创建绘图
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, s=10)

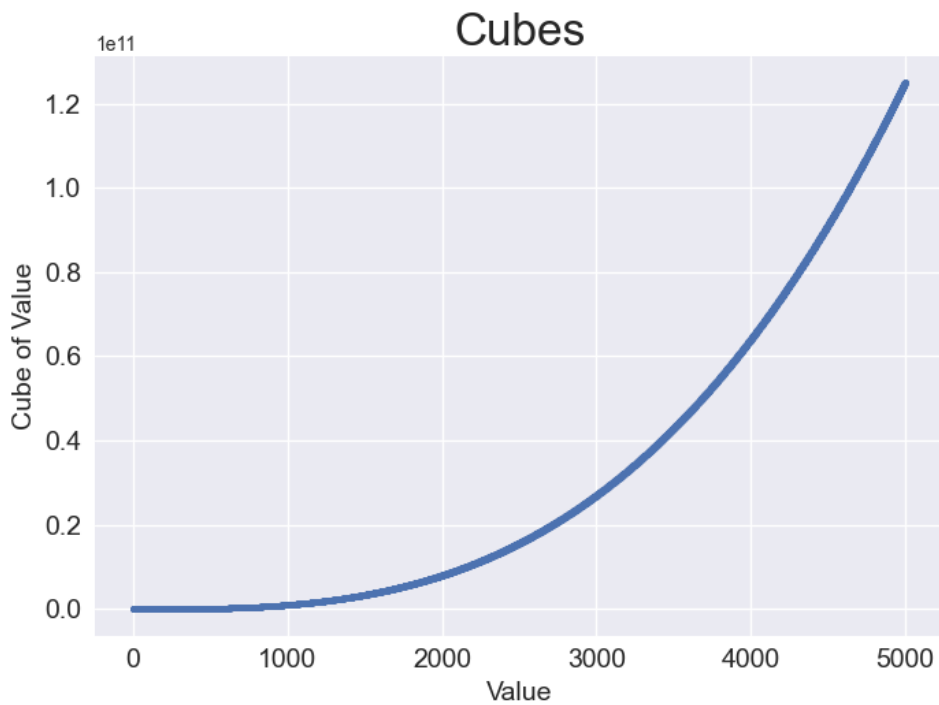
# 设置图题和坐标轴标签
ax.set_title("Cubes", fontsize=24)
```

```
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)

# 显示绘图
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。

colored_cubes.py

```
import matplotlib.pyplot as plt

# 定义数据
x_values = range(1, 5001)
```

```
y_values = [x**3 for x in x_values]

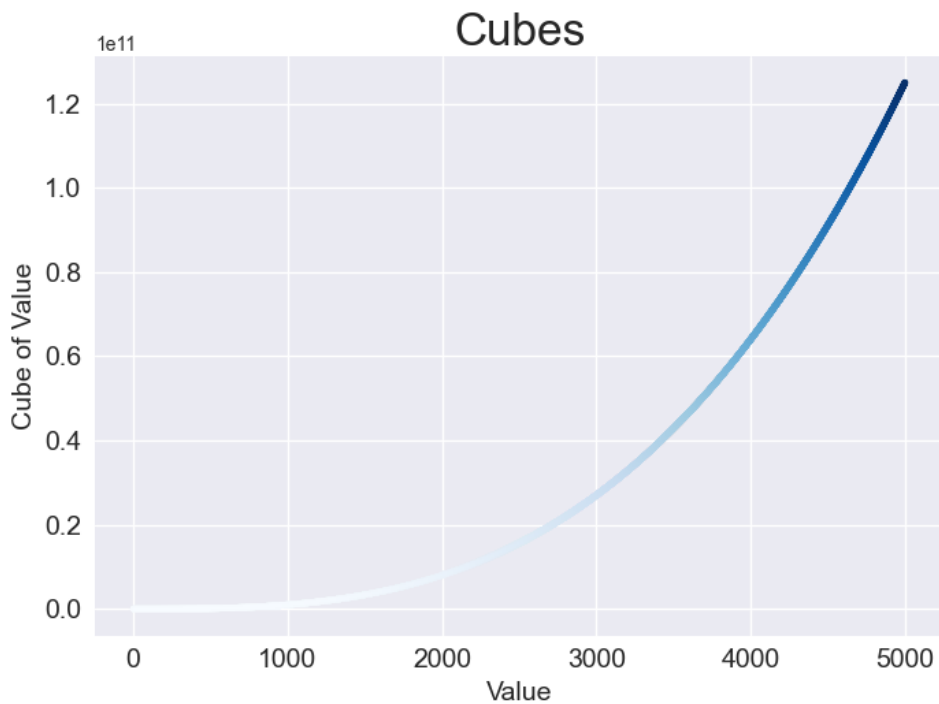
# 创建绘图
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.scatter(x_values, y_values, c=y_values, cmap=plt.cm.Blues, s=10)

# 设置图题和坐标轴标签
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)

# 显示绘图
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()`。为了模拟花粉在水滴表面的运动路径，向`plt.plot()`传递`rw.x_values`和`rw.y_values`，并指定实参`linewidth`。请使用5000个点而不是50 000个点，以免绘图中的点过于密集。

molecular_motion.py

```
import matplotlib.pyplot as plt

from random_walk import RandomWalk

# 创建一个 RandomWalk 实例
rw = RandomWalk(5_000)
rw.fill_walk()

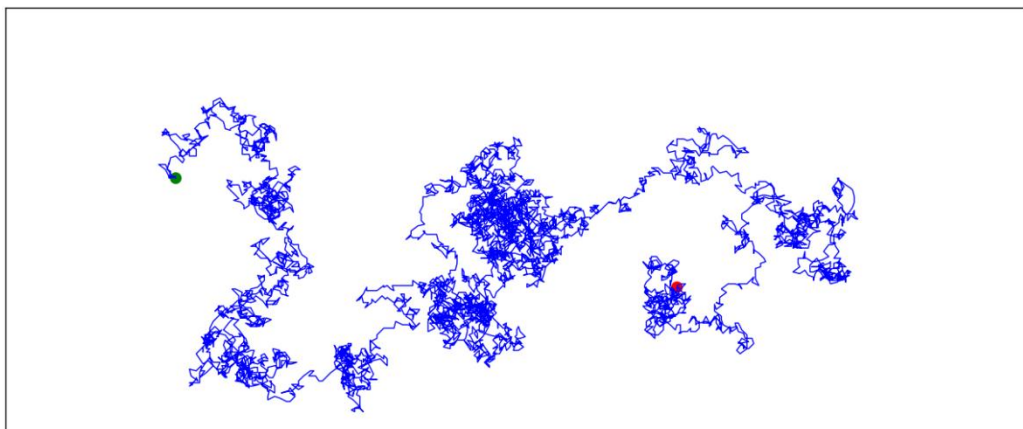
# 将 RandomWalk 实例包含的点都绘制出来
plt.style.use('classic')
fig, ax = plt.subplots()
point_numbers = range(rw.num_points)
ax.plot(rw.x_values, rw.y_values, linewidth=1)
ax.set_aspect('equal')

# 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()
```

输出：



练习 15.5 重构

`fill_walk()` 方法很长。请新建一个名为 `get_step()` 的方法，用于确定每次游走的距离和方向，并计算这次游走将如何移动。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()` 方法的规模，让它阅读和理解起来更容易。

random_walk_refactored.py

```
from random import choice
```

```
class RandomWalk:
```

```
    """一个生成随机游走数据的类"""
```

```
    def __init__(self, num_points=5000):
```

```
        """初始化随机游走的属性"""
```

```
        self.num_points = num_points
```

```
        # 所有随机游走都始于(0, 0)
```

```
        self.x_values = [0]
```

```
        self.y_values = [0]
```

```
    def fill_walk(self):
```

```
        """计算随机游走包含的所有点"""
```

```
        # 不断游走，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)

def get_step(self):
    """计算随机游走中的一步"""
    direction = choice([1, -1])
    distance = choice([0, 1, 2, 3, 4])
    step = direction * distance

    return step
```

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。

two_d8.py

```
import plotly.express as px

from die import Die

# 创建两个 D8
die_1 = Die(8)
die_2 = Die(8)
```

```
# 掷骰子多次，并将结果存储在一个列表中
results = []
for roll_num in range(50_000):
    result = die_1.roll() + die_2.roll()
    results.append(result)

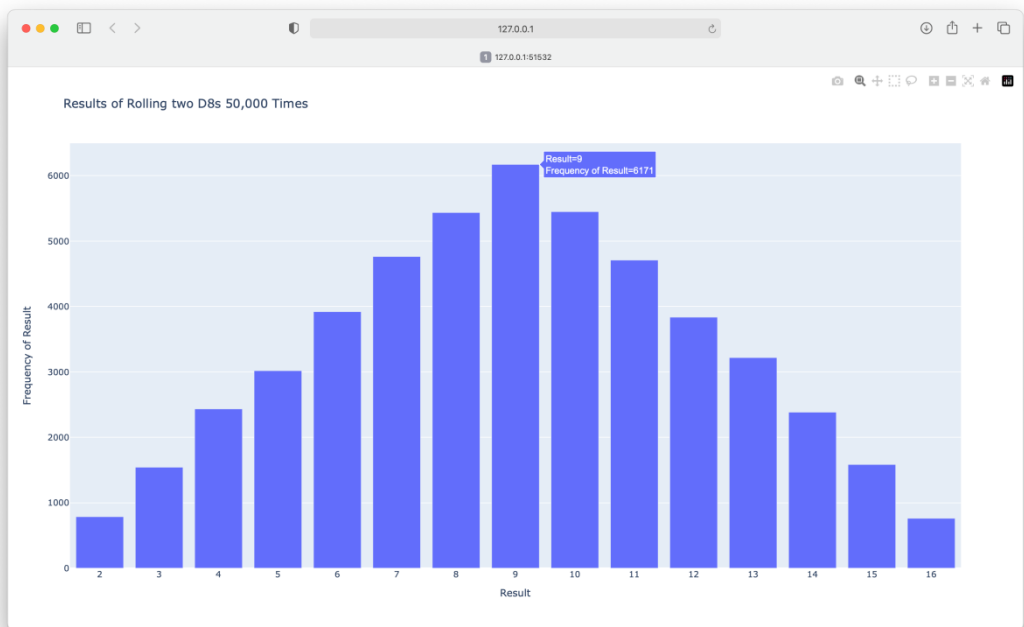
# 分析结果
frequencies = []
max_result = die_1.num_sides + die_2.num_sides
poss_results = range(2, max_result+1)
for value in poss_results:
    frequency = results.count(value)
    frequencies.append(frequency)

# 可视化结果
title = "Results of Rolling two D8s 50,000 Times"
labels = {'x': 'Result', 'y': 'Frequency of Result'}
fig = px.bar(x=poss_results, y=frequencies, title=title, labels=labels)

# 进一步定制图形
fig.update_layout(xaxis_dtick=1)

fig.show()
```

输出：



练习 15.7 同时掷三个骰子

在同时掷三个D6时，可能得到的最小点数为3，最大点数为18。请通过可视化展示同时掷三个D6的结果。

three_dice.py

```
import plotly.express as px
```

```
from die import Die
```

```
# 创建三个 D6
```

```
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(50_000):
```

```
    result = die_1.roll() + die_2.roll() + die_3.roll()
```

```
    results.append(result)
```

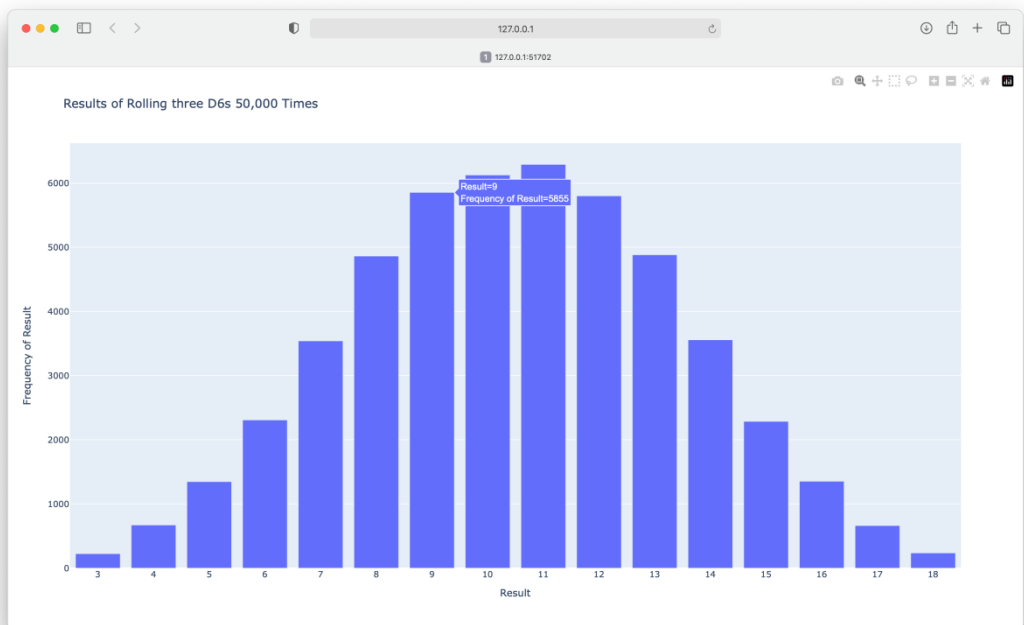
```
# 分析结果
frequencies = []
max_result = die_1.num_sides + die_2.num_sides + die_3.num_sides
poss_results = range(3, max_result+1)
for value in poss_results:
    frequency = results.count(value)
    frequencies.append(frequency)

# 可视化结果
title = "Results of Rolling three D6s 50,000 Times"
labels = {'x': 'Result', 'y': 'Frequency of Result'}
fig = px.bar(x=poss_results, y=frequencies, title=title, labels=labels)

# 进一步定制图形
fig.update_layout(xaxis_dtick=1)

fig.show()
```

输出：



练习 15.8 将点数相乘

在同时掷两个骰子时，通常将它们的点数相加，下面换个思路。请通过可视化展示将两个骰子的点数相乘的结果。

multiplication.py

```
import plotly.express as px

from die import Die

# 创建两个 D6
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(50_000):
    result = die_1.roll() * die_2.roll()
    results.append(result)

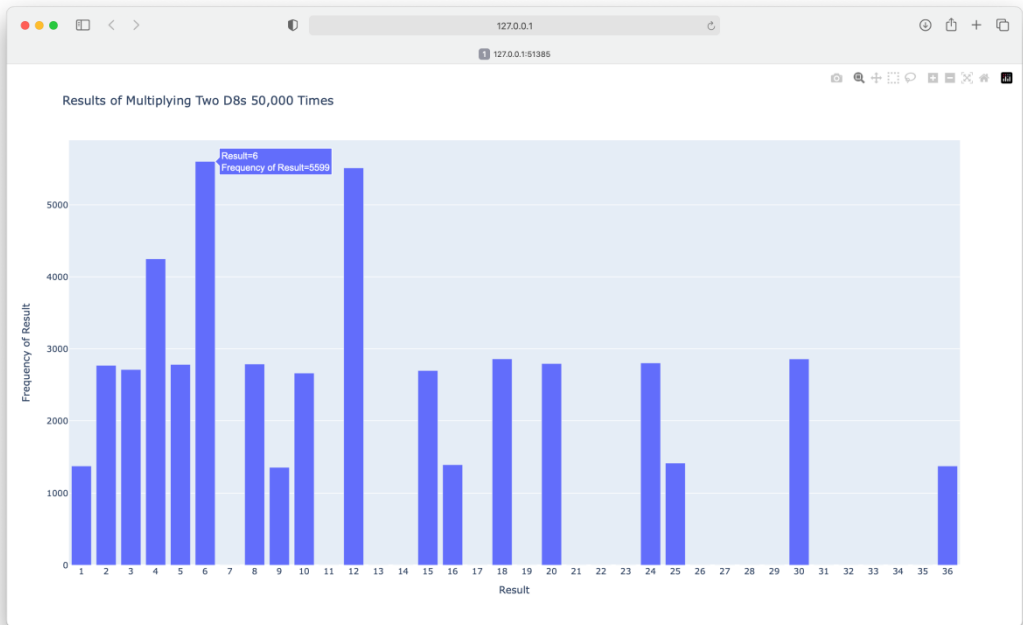
# 分析结果
frequencies = []
max_result = die_1.num_sides * die_2.num_sides
poss_results = range(1, max_result+1)
for value in poss_results:
    frequency = results.count(value)
    frequencies.append(frequency)

# 可视化结果
title = "Results of Multiplying Two D8s 50,000 Times"
labels = {'x': 'Result', 'y': 'Frequency of Result'}
fig = px.bar(x=poss_results, y=frequencies, title=title, labels=labels)

# 进一步定制图形
fig.update_layout(xaxis_dtick=1)

fig.show()
```

输出：



练习 15.9 改用列表推导式

为清晰起见，本节在模拟掷骰子的结果时，使用的是较长的 `for` 循环。如果你熟悉列表推导式，可以尝试将这些程序中的一个或两个 `for` 循环改为列表推导式。

die_comprehension.py

```
import plotly.express as px
```

```
from die import Die
```

```
# 创建一个 D6 和一个 D10
```

```
die_1 = Die()
```

```
die_2 = Die(10)
```

```
# 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(50_000)]
```

```
# 分析结果
```

```
max_result = die_1.num_sides + die_2.num_sides
poss_results = range(2, max_result+1)

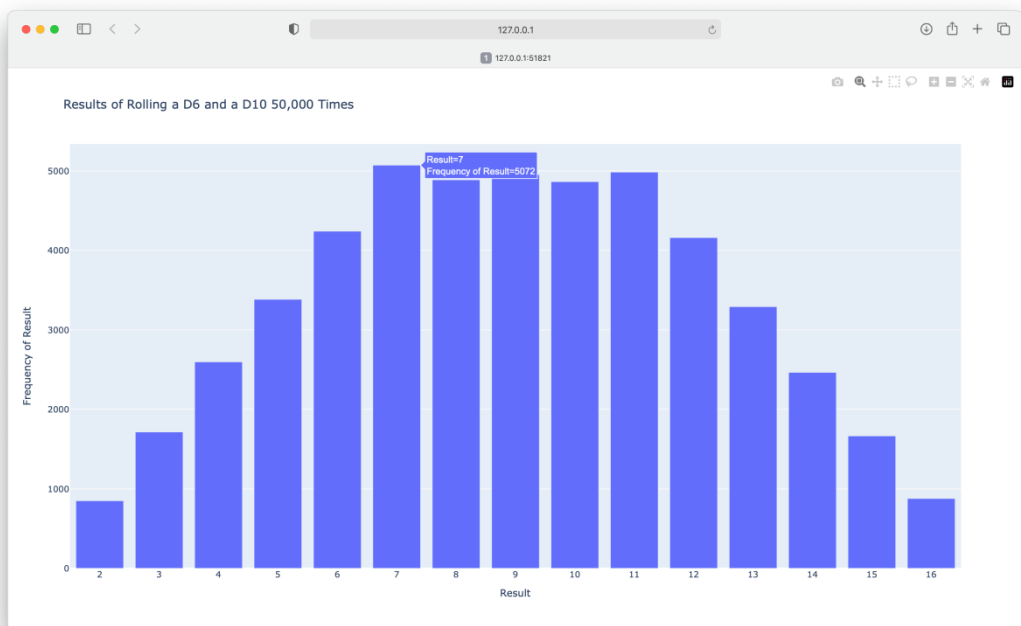
frequencies = [results.count(value) for value in poss_results]

# 可视化结果
title = "Results of Rolling a D6 and a D10 50,000 Times"
labels = {'x': 'Result', 'y': 'Frequency of Result'}
fig = px.bar(x=poss_results, y=frequencies, title=title, labels=labels)

# 进一步定制图形
fig.update_layout(xaxis_dtick=1)

fig.show()
```

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_2021_full.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少，可针对死亡谷完成这个练习。

sitka_rainfall.py

```
from pathlib import Path
import csv
from datetime import datetime

import matplotlib.pyplot as plt

path = Path('weather_data/sitka_weather_2021_full.csv')
lines = path.read_text().splitlines()

reader = csv.reader(lines)
header_row = next(reader)

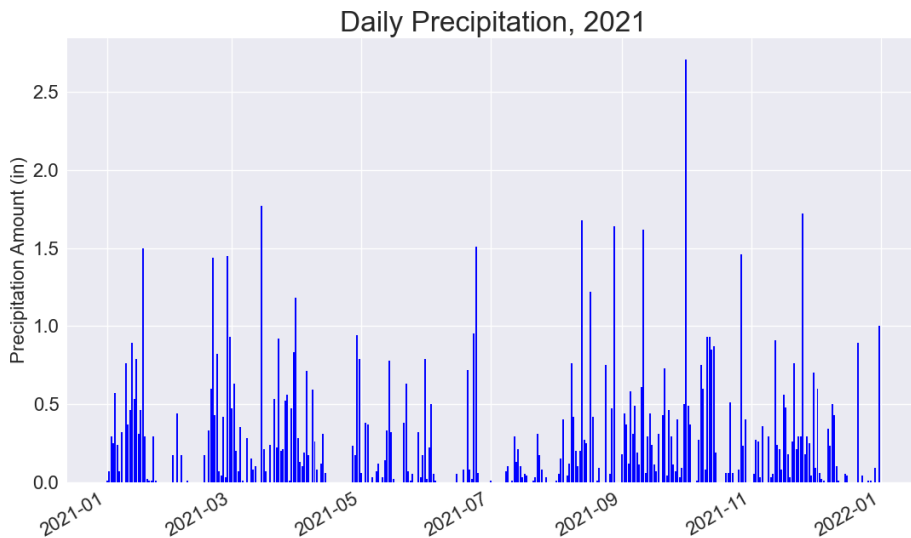
# 提取日期和降水量
dates, precip = [], []
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')
    precip = float(row[5])
    dates.append(current_date)
    precip.append(precip)

# 绘制降水量图形
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.bar(dates, precip, color='blue')

# 设置绘图的格式
ax.set_title("Daily Precipitation, 2021", fontsize=24)
ax.set_xlabel('', fontsize=16)
fig.autofmt_xdate()
ax.set_ylabel("Precipitation Amount (in)", fontsize=16)
ax.tick_params(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中，温度刻度表示的数据范围不同。为了准确地比较锡特卡和死亡谷的温度范围，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所示图形的y轴设置，对锡特卡和死亡谷的温度范围进行直接比较（也可对任意两个地方的温度范围进行比较）。

要设置y轴的取值范围,可使用函数 `set_ylim()`;要设置x轴的取值范围,可使用函数 `set_xlim()`。

sitka_highs_lows_comparison.py

```
from pathlib import Path
import csv
from datetime import datetime

import matplotlib.pyplot as plt

path = Path('weather_data/sitka_weather_2021_simple.csv')
lines = path.read_text().splitlines()

reader = csv.reader(lines)
header_row = next(reader)

# 提取日期、最高温度和最低温度
```

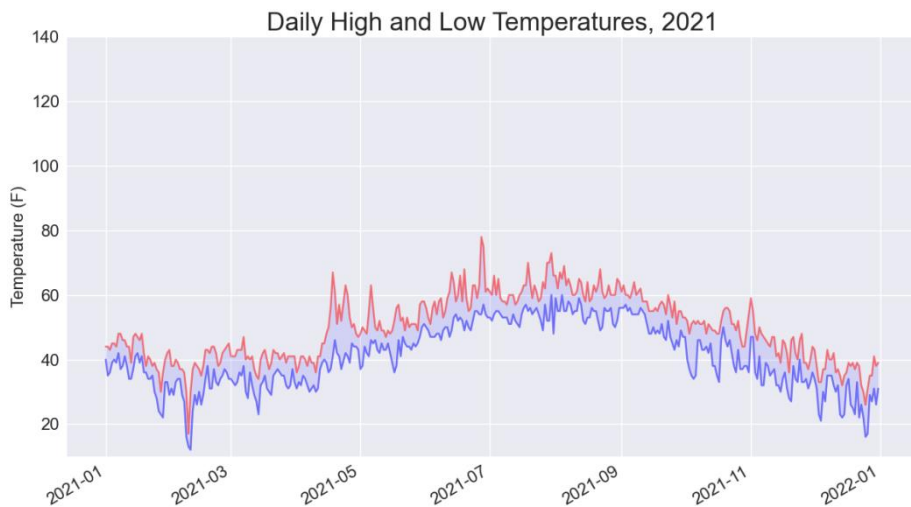
```
dates, highs, lows = [], [], []
for row in reader:
    current_date = datetime.strptime(row[2], '%Y-%m-%d')
    high = int(row[4])
    low = int(row[5])
    dates.append(current_date)
    highs.append(high)
    lows.append(low)

# 根据最高温度和最低温度绘图
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.plot(dates, highs, color='red', alpha=0.5)
ax.plot(dates, lows, color='blue', alpha=0.5)
ax.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)

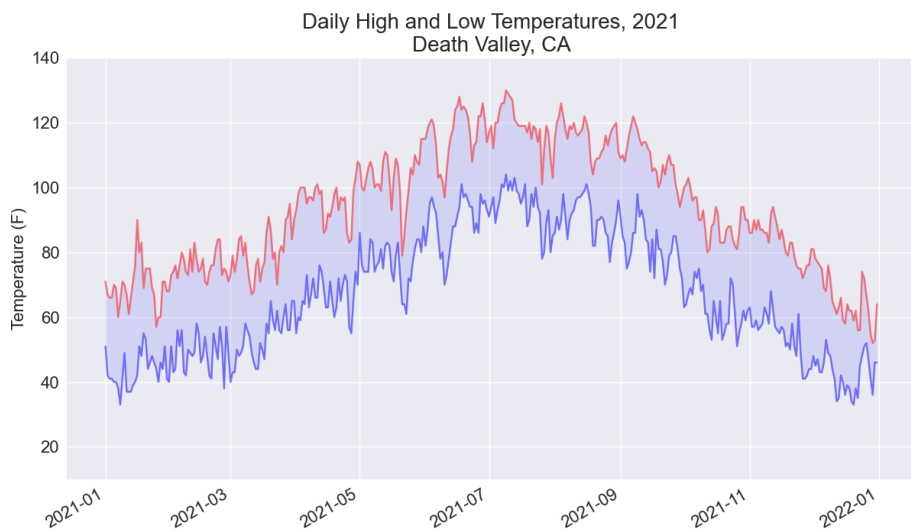
# 设置绘图的格式
ax.set_title("Daily High and Low Temperatures, 2021", fontsize=24)
ax.set_xlabel('', fontsize=16)
fig.autofmt_xdate()
ax.set_ylabel("Temperature (F)", fontsize=16)
ax.tick_params(labelsize=16)
ax.set_ylim(10, 140)

plt.show()
```

输出：



根据死亡谷的温度数据绘图时，使用函数 `set_ylim()` 指定同样的取值范围，可让两个绘图的 y 轴刻度相同：



在同一个图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温度和最低温度，并绘图。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中。我稍微调整了颜色，以便将这两个地方的数据区分开来。

sitka_death_valley_comparison.py

```
from pathlib import Path
```

```
import csv
from datetime import datetime

import matplotlib.pyplot as plt

def get_weather_data(path, dates, highs, lows, date_index, high_index,
                    low_index):
    """从数据文件中获取最高温度和最低温度"""
    lines = path.read_text().splitlines()
    reader = csv.reader(lines)
    header_row = next(reader)

    # 提取日期、最高温度和最低温度
    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)

    # 获取锡特卡的温度数据
    path = Path('weather_data/sitka_weather_2021_simple.csv')
    dates, highs, lows = [], [], []
    get_weather_data(path, dates, highs, lows, date_index=2, high_index=4,
                    low_index=5)

    # 根据锡特卡的数据绘图
    plt.style.use('seaborn-v0_8')
    fig, ax = plt.subplots()
    ax.plot(dates, highs, color='red', alpha=0.6)
    ax.plot(dates, lows, color='blue', alpha=0.6)
    ax.fill_between(dates, highs, lows, facecolor='blue', alpha=0.15)
```

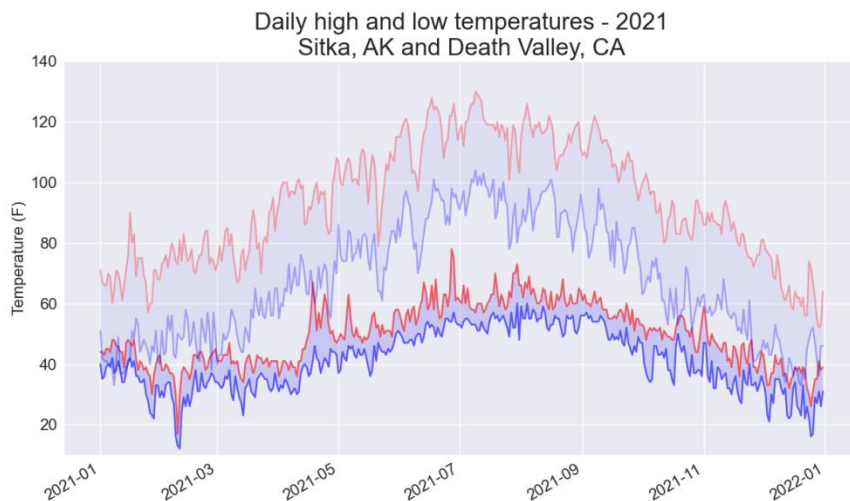
```
# 获取死亡谷的数据
path = Path('weather_data/death_valley_2021_simple.csv')
dates, highs, lows = [], [], []
get_weather_data(path, dates, highs, lows, date_index=2, high_index=3,
                  low_index=4)

# 将死亡谷的数据添加到当前绘图中
ax.plot(dates, highs, color='red', alpha=0.3)
ax.plot(dates, lows, color='blue', alpha=0.3)
ax.fill_between(dates, highs, lows, facecolor='blue', alpha=0.05)

# 设置绘图的格式
title = "Daily high and low temperatures - 2021"
title += "\nSitka, AK and Death Valley, CA"
ax.set_title(title, fontsize=24)
ax.set_xlabel('', fontsize=16)
fig.autofmt_xdate()
ax.set_ylabel("Temperature (F)", fontsize=16)
ax.tick_params(labelsize=16)
ax.set_ylim(10, 140)

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列和TMAX列的索引。请根据文件头行确定这些列的索引，让程序同时适用于锡特卡和死亡谷。另外，请根据气象站的名称自动生成图题。

`index()` 方法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。

automatic_indexes.py

```
from pathlib import Path
import csv
from datetime import datetime

import matplotlib.pyplot as plt

path = Path('weather_data/death_valley_2021_simple.csv')
lines = path.read_text().splitlines()

reader = csv.reader(lines)
header_row = next(reader)

date_index = header_row.index('DATE')
high_index = header_row.index('TMAX')
low_index = header_row.index('TMIN')
name_index = header_row.index('NAME')

# 提取日期、最高温度和最低温度
dates, highs, lows = [], [], []
place_name = ""
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]
```

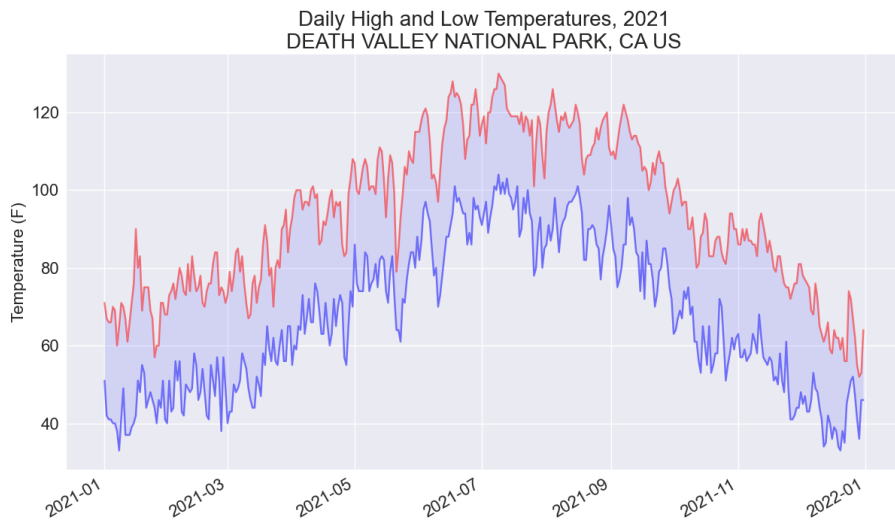
```
current_date = datetime.strptime(row[date_index], '%Y-%m-%d')
try:
    high = int(row[high_index])
    low = int(row[low_index])
except ValueError:
    print(f"Missing data for {current_date}")
else:
    dates.append(current_date)
    highs.append(high)
    lows.append(low)

# 根据最高温度和最低温度绘图
plt.style.use('seaborn-v0_8')
fig, ax = plt.subplots()
ax.plot(dates, highs, color='red', alpha=0.5)
ax.plot(dates, lows, color='blue', alpha=0.5)
ax.fill_between(dates, highs, lows, facecolor='blue', alpha=0.1)

# 设置绘图的格式
title = f"Daily High and Low Temperatures, 2021\n{place_name}"
ax.set_title(title, fontsize=20)
fig.autofmt_xdate()
ax.set_ylabel("Temperature (F)", fontsize=16)
ax.tick_params(labelsize=16)

plt.show()
```

输出:



练习 16.6 重构

在从`all_eq_dicts`中提取数据的循环中，使用了变量来存储震级、经度、纬度和标题，再将这些值分别追加到相应列表的末尾。这旨在清晰地演示如何从GeoJSON文件中提取数据，但并非必须这样做。你也可以不使用这些临时变量，而是直接从`eq_dict`中提取这些值，并将它们追加到相应的列表末尾。这样做将缩短这个循环的循环体，使其只包含4行代码。

eq_world_map_refactored.py

```
from pathlib import Path
import json

import plotly.express as px
import pandas as pd

# 将数据作为字符串读取并转换为 Python 对象
path = Path("eq_data/eq_data_30_day_m1.geojson")
try:
    contents = path.read_text()
except:
    contents = path.read_text(encoding="utf8")
all_eq_data = json.loads(contents)

# 获取数据集中的地震列表
all_eq_dicts = all_eq_data["features"]
```

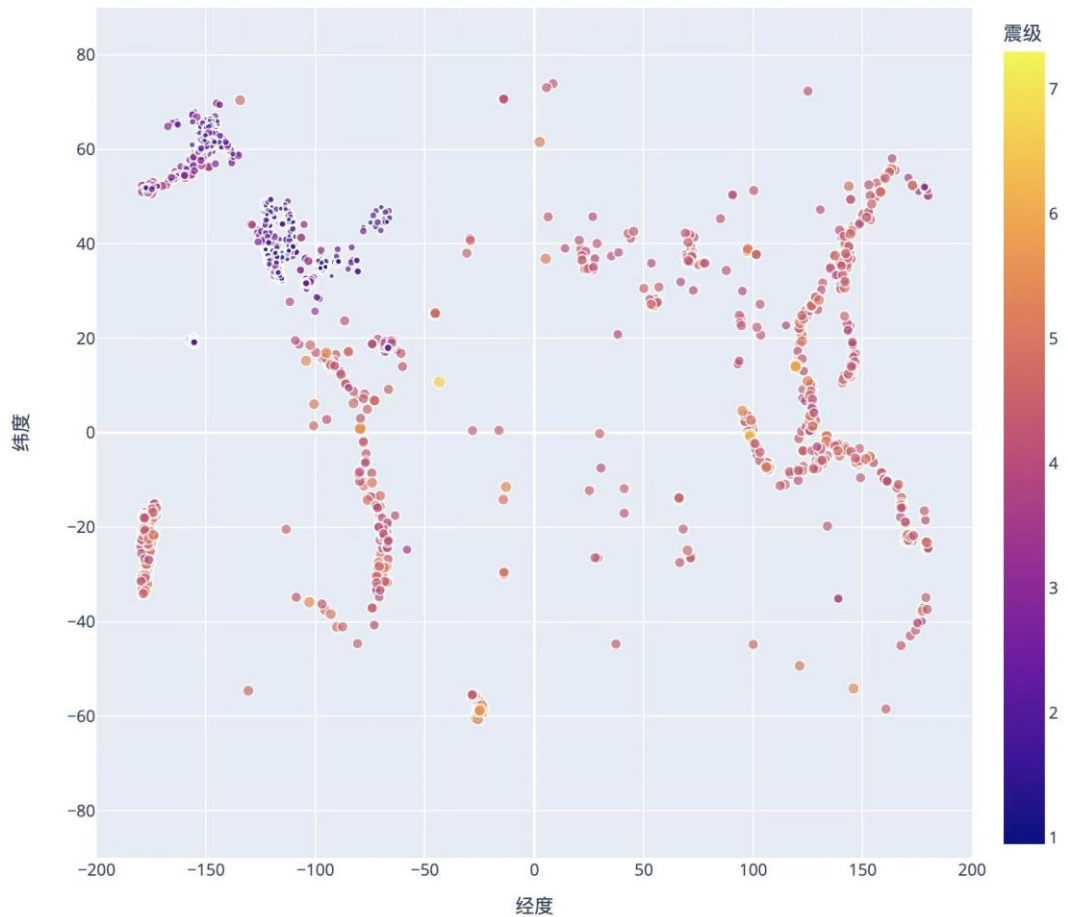
```
mags, titles, lons, lats = [], [], [], []
for eq_dict in all_eq_dicts:
    mags.append(eq_dict["properties"]["mag"])
    titles.append(eq_dict["properties"]["title"])
    lons.append(eq_dict["geometry"]["coordinates"][0])
    lats.append(eq_dict["geometry"]["coordinates"][1])

data = pd.DataFrame(
    data=zip(lons, lats, titles, mags), columns=["经度", "纬度", "位置", "
震级"]
)

fig = px.scatter(
    data,
    x="经度",
    y="纬度",
    range_x=[-200, 200],
    range_y=[-90, 90],
    width=800,
    height=800,
    title="全球地震散点图",
    size="震级",
    size_max=10,
    color="震级",
    hover_name="位置",
)
fig.write_html("eq_data/global_earthquakes_refactored.html")
fig.show()
```

输出:

全球地震散点图



练习 16.7 自动生成标题

本节中的图形使用的是通用标题“全球地震散点图”。你也可以不这样做，而是将数据集的名称（title，它位于GeoJSON文件的metadata部分）用作散点图的标题。为此，可提取这个值并将其赋给变量title。

eq_world_map_automated_title.py

```
from pathlib import Path
import json

import plotly.express as px
import pandas as pd
```



```
# 将数据作为字符串读取并转换为 Python 对象
path = Path("eq_data/eq_data_30_day_m1.geojson")
try:
    contents = path.read_text()
except:
    contents = path.read_text(encoding="utf8")
all_eq_data = json.loads(contents)

# 获取数据集中的地震列表
title = all_eq_data["metadata"]["title"]
all_eq_dicts = all_eq_data["features"]

mags, titles, lons, lats = [], [], [], []
for eq_dict in all_eq_dicts:
    mags.append(eq_dict["properties"]["mag"])
    titles.append(eq_dict["properties"]["title"])
    lons.append(eq_dict["geometry"]["coordinates"][0])
    lats.append(eq_dict["geometry"]["coordinates"][1])

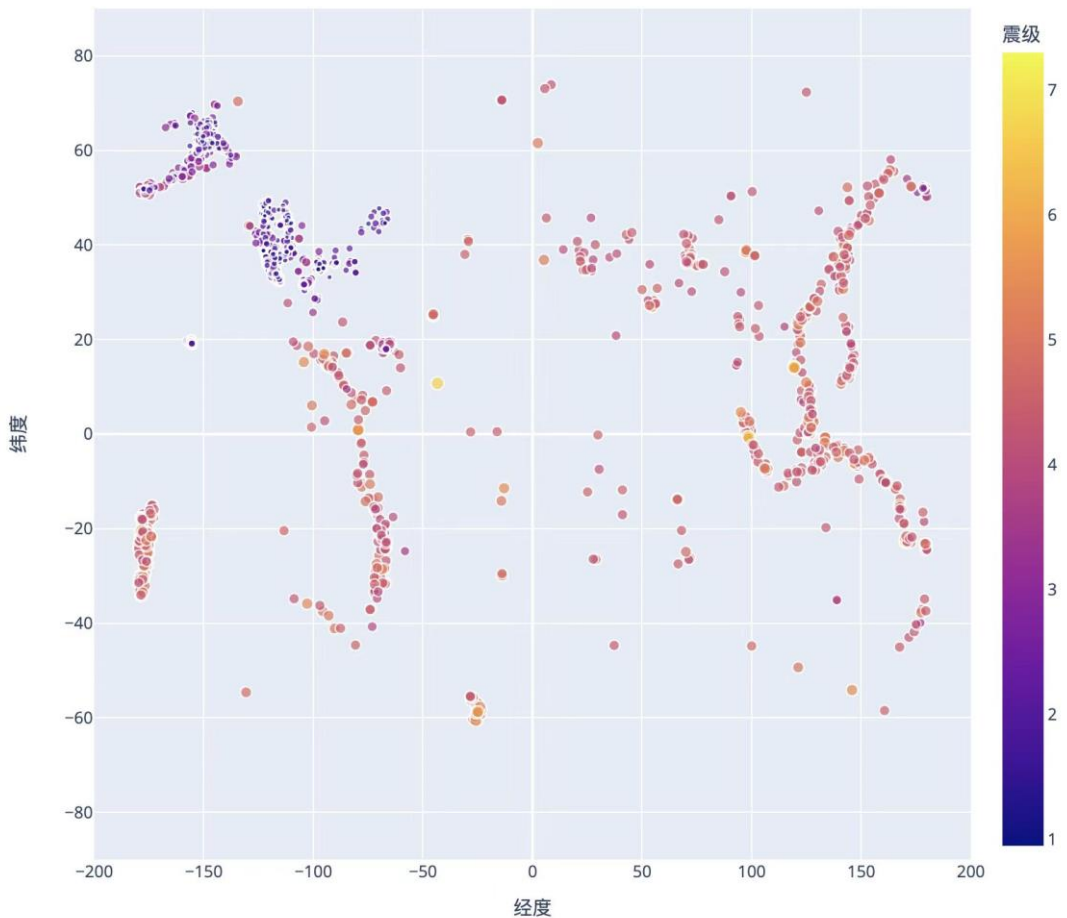
data = pd.DataFrame(
    data=zip(lons, lats, titles, mags), columns=["经度", "纬度", "位置", "
震级"]
)

fig = px.scatter(
    data,
    x="经度",
    y="纬度",
    range_x=[-200, 200],
    range_y=[-90, 90],
    width=800,
    height=800,
    title=title,
    size="震级",
    size_max=10,
    color="震级",
    hover_name="位置",
)
```

```
fig.write_html("eq_data/global_earthquakes_refactored.html")
fig.show()
```

输出：

USGS Magnitude 1.0+ Earthquakes, Past Month



练习 16.9 全球火灾

在本章的源代码文件中，有一个名为world_fires_1_day.csv的文件，其中包含全球各地的火灾信息，这些信息包括经度、纬度和火灾强度（brightness）。使用16.1节介绍的数据处理技术以及本节介绍的散点图绘制技术，绘制一幅散点图展示哪些地方发生了火灾。

注意：全球火灾情况是个波动性很大的变量，因此这个数据集的规模可能随下载时间的不同而差异巨大。当使用所有的数据时，处理时间可能很长。在这种情况下，可复制原始数据文件，但只保留开头的5000行（你可根据具体情况确定保留的行数，前提是你的系统能够在合理的时间内处理完这些数据）。

world_fires.py

```
from pathlib import Path
import csv

import plotly.express as px
import pandas as pd

path = Path("eq_data/world_fires_1_day.csv")
try:
    lines = path.read_text().splitlines()
except:
    lines = path.read_text(encoding="utf8").splitlines()
reader = csv.reader(lines)
header_row = next(reader)

# 提取经纬度和火灾强度
lats, lons, brights = [], [], []
for row in reader:
    try:
        lat = float(row[0])
        lon = float(row[1])
        bright = float(row[2])
    except ValueError:
        # 对于无效行，显示原始的日期信息
        print(f"Invalid data for {row[5]}")
    else:
        lats.append(lat)
        lons.append(lon)
        brights.append(bright)

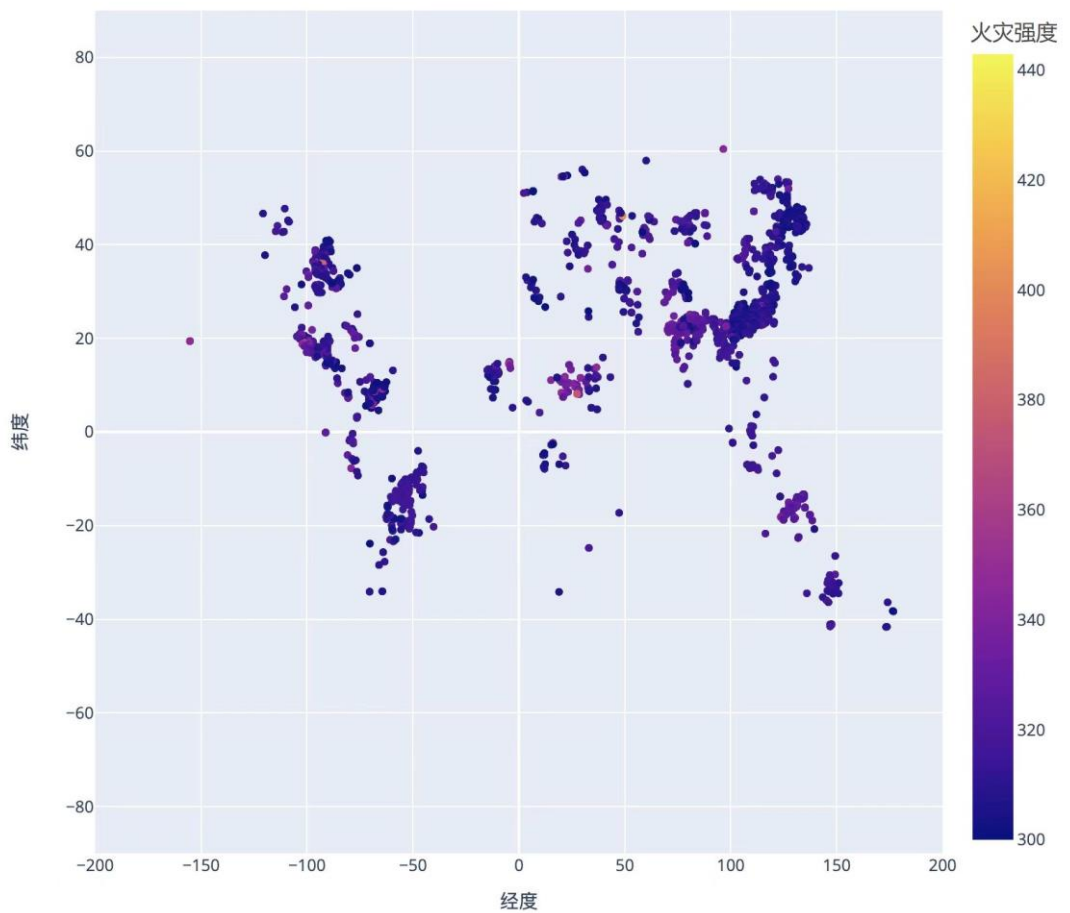
# 绘制林火活动
data = pd.DataFrame(data=zip(lons, lats, brights), columns=["经度", "纬度", "火灾强度"])

fig = px.scatter(
    data,
    x="经度",
```

```
y="纬度",
range_x=[-200, 200],
range_y=[-90, 90],
width=800,
height=800,
title="全球林火活动图",
color="火灾强度",
hover_name="火灾强度",
)
fig.write_html("eq_data/world_fires.html")
fig.show()
```

输出：

全球林火活动图



第 17 章

练习 17.1 其他语言

修改python_repos.py中的API调用，使其在生成的图形中显示其他语言最受欢迎的项目。请尝试语言JavaScript、Ruby、C、Java、Perl、Haskell和Go。

javascript_repos.py

```
import requests
import plotly.express as px

# 执行 API 调用并检查响应
url = "https://api.github.com/search/repositories"
url += "?q=language:javascript+sort:stars+stars:>10000"

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()
print(f"Complete results: {not response_dict['incomplete_results']}")

# 处理有关仓库信息
repo_dicts = response_dict['items']
repo_links, stars, hover_texts = [], [], []
for repo_dict in repo_dicts:
    # 将仓库名转换为链接
    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'])

# 创建悬停文本
owner = repo_dict['owner']['login']
```

```

description = repo_dict['description']
hover_text = f"{owner}<br />{description}"
hover_texts.append(hover_text)

# 可视化
title = "Most-Starred JavaScript Projects on GitHub"
labels = {'x': 'Repository', 'y': 'Stars'}
fig = px.bar(x=repo_links, y=stars, title=title, labels=labels,
             hover_name=hover_texts)

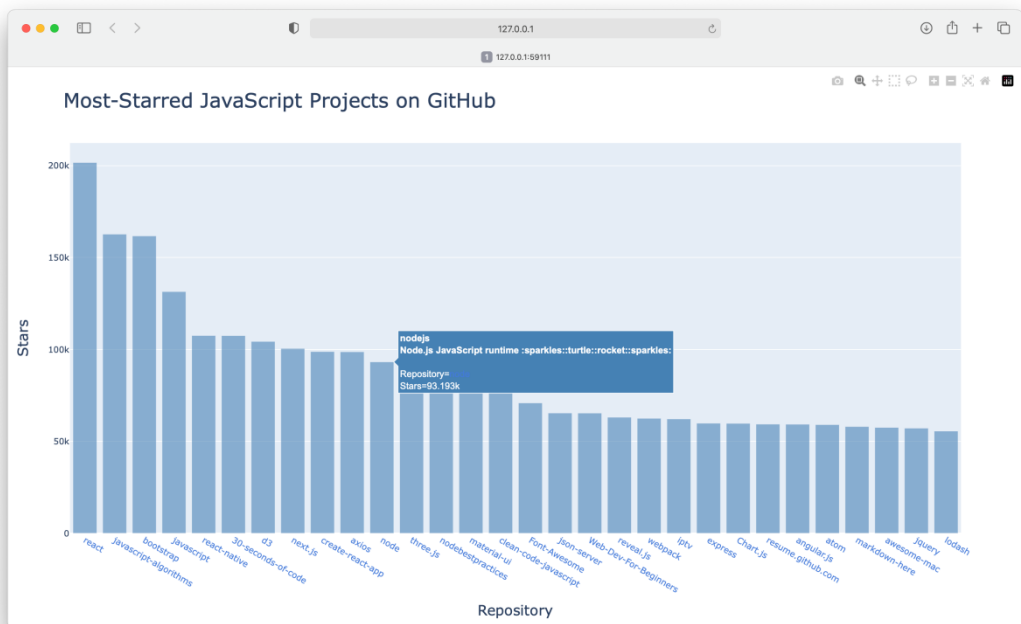
fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                  yaxis_title_font_size=20)

fig.update_traces(marker_color='SteelBlue', marker_opacity=0.6)

fig.show()

```

输出：



练习 17.2 最活跃的讨论

使用hn_submissions.py中的数据，创建一个条形图，显示Hacker News上当前哪些文章下

的讨论最活跃。条形的高度应对应于文章的评论数。条形的标签应包含文章的标题，并且充当文章的链接。如果创建图形时出现`KeyError`错误，请使用`try-except`代码块来忽略特殊的招聘帖子。

hn_discussions_visual.py

```
from operator import itemgetter

import requests
import plotly.express as px

# 执行 API 调用并检查响应
url = 'https://hacker-news.firebaseio.com/v0/topstories.json'
r = requests.get(url)
print(f"Status code: {r.status_code}")

# 针对每次提交处理信息
submission_ids = r.json()

submission_dicts = []
for submission_id in submission_ids[:20]:
    # 对于每篇文章，都执行一个 API 调用
    url = f"https://hacker-news.firebaseio.com/v0/item/{submission_id}.json"
    r = requests.get(url)
    print(f"id: {submission_id}\tstatus: {r.status_code}")
    response_dict = r.json()

    # 对于每篇文章，都创建一个字典
    try:
        submission_dict = {
            'title': response_dict['title'],
            'hn_link': f"https://news.ycombinator.com/item?id={submission_id}",
            'comments': response_dict['descendants'],
        }
    except KeyError:
        # 这是招聘帖子，关闭了评论
```

```
        continue
    else:
        submission_dicts.append(submission_dict)

submission_dicts = sorted(submission_dicts, key=itemgetter('comments'),
                           reverse=True)

# 处理数据以便绘图
article_links, comment_counts, hover_texts = [], [], []
for submission_dict in submission_dicts:
    # 将较长的文章标题缩短
    title = submission_dict['title'][:30]
    discussion_link = submission_dict['hn_link']
    article_link = f'<a href="{discussion_link}">{title}</a>'
    comment_count = submission_dict['comments']

    article_links.append(article_link)
    comment_counts.append(comment_count)
    # 悬停时显示完整的文章标题
    hover_texts.append(submission_dict['title'])

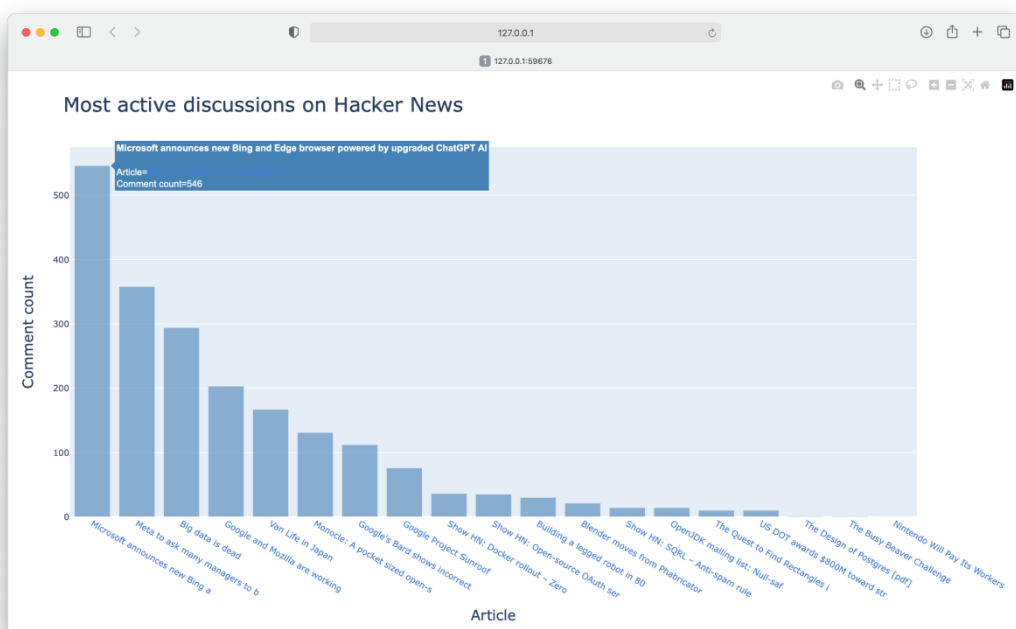
# 可视化
title = "Most active discussions on Hacker News"
labels = {'x': 'Article', 'y': 'Comment count'}
fig = px.bar(x=article_links, y=comment_counts, title=title,
labels=labels,
             hover_name=hover_texts)

fig.update_layout(title_font_size=28, xaxis_title_font_size=20,
                  yaxis_title_font_size=20)

fig.update_traces(marker_color='SteelBlue', marker_opacity=0.6)

fig.show()
```

输出：



练习 17.3 测试 python_repos.py

在python_repos.py中，我们打印了status_code的值，以核实API调用是否成功。请编写一个名为test_python_repos.py的程序，它使用pytest来断言status_code的值为200。想想还可做出哪些断言，如返回的条目（item）数符合预期，仓库总数超过特定的值，等等。

注意：通过将程序分解为函数或类，测试起来将容易得多。有鉴于此，这里的解决方案包含三部分：使用函数来完成工作的python_repos.py；一个较为简单的测试文件；一个使用夹具的测试文件。

python_repos_tested.py

```
import requests

def get_repos_info():
    """获取有关 GitHub 上 Python 仓库的信息"""
    # 执行 API 调用并检查响应
    url = "https://api.github.com/search/repositories"
    url += "?q=language:python+sort:stars+stars:>10000"

    headers = {"Accept": "application/vnd.github.v3+json"}
    r = requests.get(url, headers=headers)
```

```
print(f"Status code: {r.status_code}")

return r

def get_response_dict(response):
    """将响应对象转换为字典"""
    response_dict = response.json()
    return response_dict

def show_repos_info(response_dict):
    """显示有关返回的仓库的信息"""
    print(f"Total repositories: {response_dict['total_count']}")
    print(f"Complete results: {not response_dict['incomplete_results']}")

def get_repo_dicts(response_dict):
    """返回一系列字典（每个仓库一个）"""
    repo_dicts = response_dict['items']
    return repo_dicts

def show_repo_dicts_info(repo_dicts):
    """有关仓库的摘要信息"""
    print(f"Repositories returned: {len(repo_dicts)}")

    print("\nSelected information about each repository:")
    for repo_dict in repo_dicts:
        print("\nSelected information about first repository:")
        print(f"Name: {repo_dict['name']}")
        print(f"Owner: {repo_dict['owner']['login']}")
        print(f"Stars: {repo_dict['stargazers_count']}")
        print(f"Repository: {repo_dict['html_url']}")
        print(f"Created: {repo_dict['created_at']}")
        print(f"Updated: {repo_dict['updated_at']}")
        print(f>Description: {repo_dict['description']}")

response = get_repos_info()
response_dict = get_response_dict(response)
show_repos_info(response_dict)
repo_dicts = get_repo_dicts(response_dict)
```

```
show_repo_dicts_info(repo_dicts)
```

输出:

```
Status code: 200
```

```
Total repositories: 307
```

```
Complete results: True
```

```
Repositories returned: 30
```

Selected information about each repository:

Selected information about first repository:

```
Name: public-apis
```

```
Owner: public-apis
```

```
Stars: 227342
```

```
Repository: https://github.com/public-apis/public-apis
```

```
Created: 2016-03-20T23:49:42Z
```

```
Updated: 2023-02-08T01:53:56Z
```

```
Description: A collective list of free APIs
```

```
...
```

下面是没有使用夹具的测试文件 `test_python_repos.py`:

test_python_repos.py

```
import pytest

from python_repos_tested import get_repos_info, get_response_dict,
get_repo_dicts


def test_response_status_code():
    """测试响应的状态码为 200"""
    r = get_repos_info()
    assert r.status_code == 200


def test_response_dict():
    """核实返回的仓库数量是合适的，且结果是完整的"""
    r = get_repos_info()
    response_dict = get_response_dict(r)
```

```

total_count = response_dict['total_count']
complete_results = not response_dict['incomplete_results']

assert total_count > 240
assert complete_results

def test_repo_dicts():
    """核实 repo_dicts 中的结果是正确的"""
    r = get_repos_info()
    response_dict = get_response_dict(r)
    repo_dicts = get_repo_dicts(response_dict)

    assert len(repo_dicts) == 30

    # 检查返回的所有仓库都获得了超过 10 000 个星
    for repo_dict in repo_dicts:
        assert repo_dict['stargazers_count'] > 10_000

```

输出：

```

$ pytest test_python_repos.py
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
rootdir: /Users/eric/pcc_3e/solution_files/chapter_17
collected 3 items

test_python_repos.py ... [100%]

===== 3 passed in 6.42s =====

```

这是首次编写这些测试的结果，这样的结果合乎情理。但你可能注意到了，每个测试函数都需要调用 `get_repos_info()`，这意味着执行每个测试函数时，都将执行 API 调用，这很容易导致局势失控。

下面使用夹具重写了这三个测试函数，确保在整个测试过程中只调用 `get_repos_info()` 一次：

test_python_repos_with_fixture.py

```

import pytest

from python_repos_tested import get_repos_info, get_response_dict,

```

get_repo_dicts

```
@pytest.fixture
def response():
    """获取一个响应对象"""
    r = get_repos_info()
    return r

def test_response_status_code(response):
    """测试响应的状态码为 200"""
    assert response.status_code == 200

def test_response_dict(response):
    """核实返回的仓库数量是合适的，且结果是完整的"""
    response_dict = get_response_dict(response)

    total_count = response_dict['total_count']
    complete_results = not response_dict['incomplete_results']

    assert total_count > 240
    assert complete_results

def test_repo_dicts(response):
    """核实 repo_dicts 中的结果是正确的"""
    response_dict = get_response_dict(response)
    repo_dicts = get_repo_dicts(response_dict)

    assert len(repo_dicts) == 30

    # 检查返回的所有仓库都获得了超过 10 000 个星
    for repo_dict in repo_dicts:
        assert repo_dict['stargazers_count'] > 10_000
```

输出:

```
$ pytest test_python_repos_with_fixture.py
===== test session starts =====
platform darwin -- Python 3.10.0, pytest-7.1.2, pluggy-1.0.0
```

```

rootdir: /Users/eric/pcc_3e/solution_files/chapter_17
collected 3 items

test_python_repos_with_fixture.py ... [100%]

===== 3 passed in 7.46s =====

```

注意：pytest报告的时间并非运行测试套件所需的时间，因为它没有将等待网络调用的时间计算在内。如果你使用秒表或其他非pytest计时器测量这两次测试的时间，将发现当使用夹具时，测试文件的运行时间比未使用夹具时要短得多。当测试函数更多时，这种时间方面的差异将更大。

第18章

练习 18.1 新项目

为了更深入地了解Django的作用，可以创建两三个空项目来看看Django都创建了什么。新建一个文件夹，并给它指定简单的名称，如tik_gram或insta_tok（不要在目录ll_project中新建该文件夹）。在终端中切换到该文件夹，并创建一个虚拟环境。在这个虚拟环境中安装Django，并执行命令`django-admin.py startproject tg_project`。（千万不要忘了这个命令末尾的句点）。

看看这个命令创建了哪些文件和文件夹，并将其与项目“学习笔记”包含的文件和文件夹进行比较。这样多做几次，直到你对Django新建项目时创建的东西了如指掌。然后，将项目目录删除（如果你想这样做的话）。

输出应该类似于下面这样：

```

(tg_env) tik_gram$ django-admin startproject tg_project .
(tg_env) tik_gram$ ls
manage.py  tg_project
(tg_env) tik_gram$ ls tg_project
__init__.py asgi.py      settings.py urls.py      wsgi.py

```

练习 18.2 简短的条目

当前，当Django在管理网站或shell中显示Entry实例时，模型Entry的`__str__()`方法都在其末尾加上省略号。请在`__str__()`方法中添加一条if语句，仅在条目长度超过50个字符时才添加省略号。使用管理网站添加一个不超过50个字符的条目，并核实在显示它时不带省略号。

models.py

```

from django.db import models

class Topic(models.Model):
    ...

```

```
class Entry(models.Model):
    """学到的有关某个主题的具体知识"""
    topic = models.ForeignKey(Topic, on_delete=models.CASCADE)
    text = models.TextField()
    date_added = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name_plural = 'entries'

    def __str__(self):
        """返回一个表示条目的简单字符串"""
        if len(self.text) > 50:
            return f"{self.text[:50]}..."
        else:
            return self.text
```

练习 18.4 比萨店

新建一个名为pizzeria_project的项目，并在其中添加一个名为pizzas的应用程序。定义一个名为Pizza的模型，并让它包含字段name，用于存储比萨的名称，如Hawaiian和Meat Lovers。定义一个名为Topping的模型，并让它包含字段pizza和name其中pizza是一个关联到Pizza的外键，而name用于存储配料，如pineapple、Canadian bacon和sausage。

向管理网站注册这两个模型，并使用管理网站输入一些比萨名和配料。使用shell来查看输入的数据。

即便是很小的 Django 项目，包含的代码也很多，这里无法将其全部列出。完整的答案请见 ex_18_4_pizzeria。

练习 18.5 饮食规划程序

假设要创建一个应用程序，帮助用户规划一周的饮食。为此，新建一个文件夹并将其命名为meal_planner，再在这个文件夹中新建一个Django项目。然后，新建一个名为meal_plans的应用程序，并为这个项目创建一个简单的主页。

完整的答案见 ex_18_5_meal_planner。

练习 18.6 比萨店主页

在为练习18.4创建的项目pizzeria_project中，添加一个主页。

完整的答案见 `ex_18_6_pizzeria_home_page`。

练习 18.8 比萨店页面

在为练习18.6开发的项目 `pizzeria_project` 中添加一个页面，显示供应的比萨的名称。然后，将每个比萨名都设置成链接，可通过单击来显示一个列出相应配料的页面。请务必使用模板继承来高效地创建页面。

完整的答案见 `ex_18_8_pizzeria_pages`。

第 19 章

练习 19.1 博客

新建一个 Django 项目，将其命名为 `Blog`。创建一个名为 `blogs` 的应用程序，再创建两个分别表示博客和博文的模型，并让这些模型包含合适的字段。为这个项目创建一个超级用户，并使用管理网站创建一个博客和几篇简短的博文。创建一个主页，在其中按恰当的顺序显示所有的博文。

创建三个页面，分别用于创建博客、发布新博文和编辑现有的博文。尝试使用这些页面，确认它们能够正确地工作。

完整的答案见 `ex_19_1_blog`。

练习 19.2 博客账户

在为练习19.1开发的项目 `Blog` 中，添加用户身份验证和注册系统。向已登录的用户显示其用户名，向未注册的用户显示注册页面的链接。

完整的答案见 `ex_19_2_blog_accounts`。

练习 19.3 重构

在 `views.py` 中，我们在两个地方核实了主题关联到的用户为当前登录的用户。请将执行该检查的代码放在函数 `check_topic_owner()` 中，并在这两个地方调用这个函数。

在 `views.py` 中，受这种重构工作影响的地方有三个，如下面所示。完成重构后的完整项目见 `ex_19_3_refactoring`。

`views.py`

```
...
@login_required
def topic(request, topic_id):
    """显示单个主题及其所有的条目"""
```



```
        topic = Topic.objects.get(id=topic_id)
        check_topic_owner(topic, request.user)
        ...

...
@login_required
def edit_entry(request, entry_id):
    """编辑既有条目"""
    entry = Entry.objects.get(id=entry_id)
    topic = entry.topic
    check_topic_owner(topic, request.user)
    ...

def check_topic_owner(topic, user):
    """确认请求的主题属于当前用户

    如果不是这样的，就引发 Http404 错误
    """
    if topic.owner != user:
        raise Http404
```

练习 19.4 保护页面 new_entry

一个用户可以在另一个用户的学习笔记中添加条目，方法是在URL中指定属于另一个用户的主体的ID。为了防范这种攻击，请在保存新条目前，核实它所属的主题归属于当前用户。

注意：这里的答案以练习 19.3 要求的重构为基础，包含该答案的完整项目见 `ex_19_4_protecting_new_entry`。

```
views.py
...
@login_required
def new_entry(request, topic_id):
    """在特定主题中添加新条目"""
    topic = Topic.objects.get(id=topic_id)
    check_topic_owner(topic, request.user)
    ...

...
```

```
def check_topic_owner(topic, user):  
    """确认请求的主题属于当前用户  
  
    如果不是这样的，就引发 Http404 错误  
    """  
    if topic.owner != user:  
        raise Http404
```

练习 19.5 受保护的博客

在你创建的项目Blog中，确保每篇博文都与特定的用户相关联。确保任何用户都可访问所有的博文，但只有已登录的用户能够发表博文以及编辑既有博文。在让用户能够编辑其博文的视图中，在处理表单之前确认用户编辑的是其自己发表的博文。

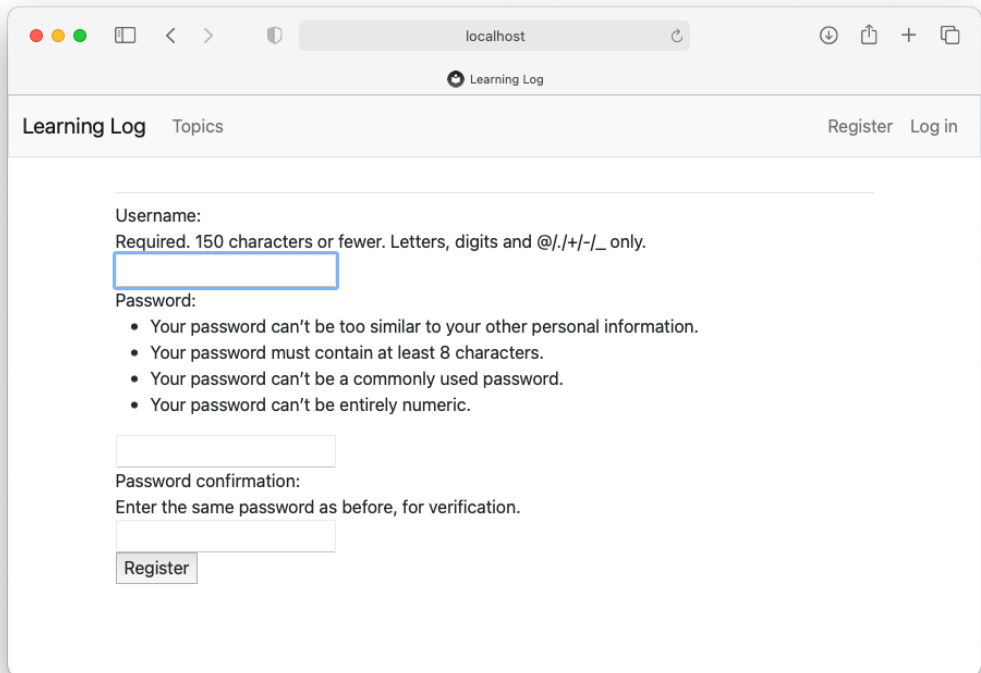
包含答案的完整项目见 `ex_19_5_protected_blog`。

第 20 章

练习 20.1 其他表单

本节对登录页面应用了Bootstrap样式。请对其他基于表单的页面做类似的修改，包括页面 `new_topic`、`new_entry`、`edit_entry` 和注册页面。

完整的答案见 `ex_20_1_other_forms`。下面显示了设置样式前后的注册页面。



The screenshot shows a web browser window with the address bar set to 'localhost'. The page title is 'Learning Log'. The navigation bar includes 'Learning Log' and 'Topics' on the left, and 'Register' and 'Log in' on the right. The main content area contains a registration form with the following elements:

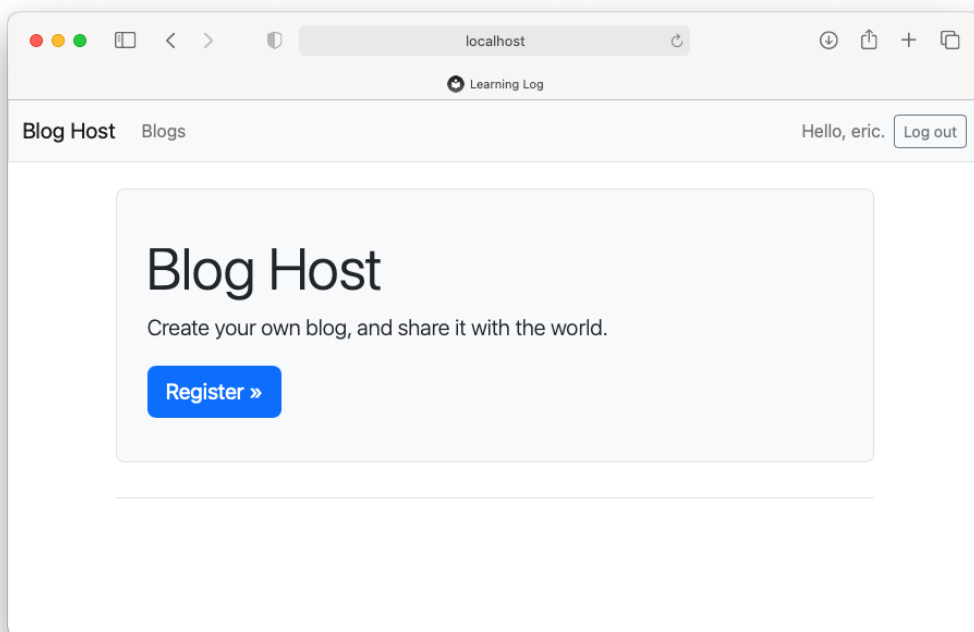
- Username:** A label followed by a text input field. Below the field is a requirement: 'Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.'
- Password:** A label followed by a text input field. Below the field is a list of requirements:
 - Your password can't be too similar to your other personal information.
 - Your password must contain at least 8 characters.
 - Your password can't be a commonly used password.
 - Your password can't be entirely numeric.
- Password confirmation:** A label followed by a text input field. Below the field is the instruction: 'Enter the same password as before, for verification.'
- Register:** A button located below the password confirmation field.

The screenshot shows a web browser window with the address bar set to 'localhost'. The page title is 'Learning Log' and the navigation bar includes 'Topics', 'Register', and 'Log in'. The main heading is 'Register a new account.' The form contains three input fields: 'Username', 'Password', and 'Password confirmation'. The 'Username' field has a blue border and a placeholder 'Username'. Below it, a requirement note states: 'Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.' The 'Password' field has a placeholder 'Password' and is followed by four bullet points: 'Your password can't be too similar to your other personal information.', 'Your password must contain at least 8 characters.', 'Your password can't be a commonly used password.', and 'Your password can't be entirely numeric.' The 'Password confirmation' field has a placeholder 'Password confirmation' and a note below it: 'Enter the same password as before, for verification.' A blue 'Register' button is at the bottom of the form.

练习 20.2 设置博客的样式

对于你在第19章创建的项目Blog，使用Bootstrap来设置样式。

完整的答案见 `ex_20_2_stylish_blog`。请参阅有关如何设置 `card` 样式（<https://getbootstrap.com/docs/5.2/components/card/>）和文本样式（<https://getbootstrap.com/docs/5.2/utilities/text/>）的 Bootstrap 文档，这对完成本练习会有所帮助。应用类似于“学习笔记”使用的样式后，主页是下面这样的。



练习 20.3 线上博客

将你一直在开发的项目Blog部署到Platform.sh上。确保将DEBUG设置为False，以免在出现错误时让用户看到完整的Django错误页面。

注意：使用命令`platform create`创建项目时，务必将项目名指定为`blog_project`。在`settings`文件中，也请使用这样的项目名。

```
(b_env)$ platform create
* Project title (--title)
Default: Untitled Project
> blog_project

* Region (--region)
The region where the project will be hosted
[au.platform.sh ] Sydney, Australia (AWS) [867 gC02eq/kWh]
[au-2.platform.sh] Sydney, Australia (AZURE) [867 gC02eq/kWh]
...
> us-3.platform.sh

* Plan (--plan)
```

```

The subscription plan
Default: development
Enter a number to choose:
  [0 ] development
  [1 ] standard
  ...
> 0

* Environments (--environments)
The number of environments
Default: 3
> 3

* Storage (--storage)
The amount of storage per environment, in GiB
Default: 5
> 5

```

完整的答案见 `ex_20_3_live_blog`。

练习 20.4 扩展“学习笔记”

在“学习笔记”中添加一项功能，并将修改推送给在线部署。先尝试做一个简单的修改，如主页中对项目做更详细的描述，再尝试添加一项高级功能，如让用户能够将主题设置为公开的。为此，需要在模型 `Topic` 中添加一个名为 `public` 的属性（其默认值为 `False`），并在页面 `new_topic` 中添加一个表单元素，让用户能够将私有主题改为公开的。然后，需要迁移项目，并修改 `views.py`，让未登录的用户也可以看到所有公开的主题。

如果你在完成项目“学习笔记”时才接触到 `Django`，让用户能够将某些主题设置为公开的将是一项非常艰巨的任务。在你尝试实现这项功能时，查看相关的 `Django` 文档可能会有所帮助。

- ❑ “模型字段参考”（<https://docs.djangoproject.com/zh-hans/4.1/ref/models/fields/>）：指出了可在 `models.py` 文件中使用的各种字段类型，以及可给这些字段指定的设置。
 - 给字段指定默认值（<https://docs.djangoproject.com/zh-hans/4.1/ref/models/fields/#default>）
 - 布尔字段（<https://docs.djangoproject.com/zh-hans/4.1/ref/models/fields/#booleanfield>）
- ❑ “执行查询”（<https://docs.djangoproject.com/zh-hans/4.1/topics/db/queries/>）：演示如何编写复杂的查询，如过滤出不与当前用户相关联的公开主题。
 - 过滤器（<https://docs.djangoproject.com/zh-hans/4.1/topics/db/queries/#retrieving-specific-objects-with-filters>）

- 链式过滤器 (<https://docs.djangoproject.com/zh-hans/4.1/topics/db/queries/#chaining-filters>)
- 当需要向匿名用户和已登录用户显示不同的信息时, 查看有关 `django.contrib.auth` 的文档 (<https://docs.djangoproject.com/zh-hans/4.1/ref/contrib/auth/>) 会有所帮助。
- 要确定用户是否已通过身份验证 (登录), 可使用属性 `is_authenticated` (https://docs.djangoproject.com/zh-hans/4.1/ref/contrib/auth/#django.contrib.auth.models.User.is_authenticated)。
- 另请参阅“限制对未登录用户的访问” (<https://docs.djangoproject.com/zh-hans/4.1/topics/auth/default/#limiting-access-to-logged-in-users>)。

下面具体说说需要对某些文件做的修改。首先, 在模型 `Topic` 中, 新增属性 `public`:

models.py

```
class Topic(models.Model):
    """用户学习的主题"""
    text = models.CharField(max_length=200)
    date_added = models.DateTimeField(auto_now_add=True)
    public = models.BooleanField(default=False)
    ...
```

在 `TopicForm` 中, 需要包含字段 `public`:

forms.py

```
class TopicForm(forms.ModelForm):
    class Meta:
        model = Topic
        fields = ['text', 'public']
    ...
```

不应再使用 `@login_required` 来保护视图 `topics` 和 `topic`。另外, 根据用户是否通过了身份验证, 执行不同的查询。对于通过身份验证的用户, 获取该用户的所有主题, 还有其他所有用户的公开主题; 对于未通过身份验证的用户, 只获取所有的公开主题。在视图 `topic` 中, 设置 `is_owner`, 以便确定是否显示链接 `new_entry` 和 `edit_entry`。另外, 还修改了返回 404 页面的条件。

views.py

```
def topics(request):
    """显示当前用户的所有主题及其他用户的所有公开主题"""
    # 获取所有合适的主题
```

```

# 如果用户已登录，就获取其所有主题，还有其他用户的所有公开主题
# 如果用户未登录，就获取所有的公开主题
if request.user.is_authenticated:
    topics = Topic.objects.filter(owner=request.user).order_by('date_added')
    # 获取不归当前用户所有的所有公开主题
    # 注意：通过将查询放在括号内，可将它分成多行
    public_topics = (Topic.objects
                      .filter(public=True)
                      .exclude(owner=request.user)
                      .order_by('date_added'))
else:
    # 用户未通过身份验证，因此返回所有的公开主题
    topics = None
    public_topics = Topic.objects.filter(public=True).order_by('date_added')

context = {'topics': topics, 'public_topics': public_topics}
return render(request, 'learning_logs/topics.html', context)

def topic(request, topic_id):
    """显示单个主题及其所有条目"""
    topic = Topic.objects.get(id=topic_id)

    # 仅当该主题归当前用户所有时，才显示链接 new_entry 和 edit_entry
    is_owner = False
    if request.user == topic.owner:
        is_owner = True

    # 如果该主题归他人所有，且不是公开的，就显示错误页面
    if (topic.owner != request.user) and (not topic.public):
        raise Http404

    entries = topic.entry_set.order_by('-date_added')
    context = {'topic': topic, 'entries': entries, 'is_owner': is_owner}
    return render(request, 'learning_logs/topic.html', context)

```

在页面 topics 中，显示两个无序列表，其中第一个仅向已通过身份验证的用户显示，而第二个向

所有用户显示。

topics.html

```
{% extends 'learning_logs/base.html' %}

{% block page_header %}
    <h1>Topics</h1>
{% endblock page_header %}

{% block content %}

    <!--仅当用户已登录时才显示 My topics-->
    {% if user.is_authenticated %}
        <ul class="list-group border-bottom pb-2 mb-4">
            <h5>My topics:</h5>
            {% for topic in topics %}
                <li class="list-group-item border-0">
                    <a href="{% url 'learning_logs:topic' topic.id %}">
                        {{ topic.text }}</a>
                </li>
            {% empty %}
                <li class="list-group-item border-0">No topics have been added
yet.</li>
            {% endfor %}
        </ul>
    {% endif %}

    <ul class="list-group border-bottom pb-2 mb-4">
        <h5>Public topics:</h5>
        {% for topic in public_topics %}
            <li class="list-group-item border-0">
                <a href="{% url 'learning_logs:topic' topic.id %}">
                    {{ topic.text }}</a>
            </li>
        {% empty %}
            <li class="list-group-item border-0">No users have posted any public
topics yet.</li>
        {% endfor %}
    </ul>
</div>
</pre>
```

```
</ul>
```

```
<a href="{% url 'learning_logs:new_topic' %}">Add a new topic</a>
```

```
{% endblock content %}
```

在页面 `topic` 中，将链接 `Add new entry` 和 `edit entry` 放在 `if` 代码块中，这些 `if` 代码块指定的条件是 `is_owner` 为 `True`。

topic.html

```
{% extends 'learning_logs/base.html' %}

{% block page_header %}
    <h1>{{ topic.text }}</h1>
{% endblock page_header %}

{% block content %}

    {% if is_owner %}
        <p>
            <a href="{% url 'learning_logs:new_entry' topic.id %}">Add new
entry</a>
        </p>
    {% endif %}

    {% for entry in entries %}
        <div class="card mb-3">
            <!--包含时间戳和编辑链接的标题-->
            <h4 class="card-header">
                {{ entry.date_added|date:'M d, Y H:i' }}
                {% if is_owner %}
                    <small><a href="{% url 'learning_logs:edit_entry' entry.id %}">
                        edit entry</a></small>
                {% endif %}
            </h4>
            <!--包含条目文本的正文-->
            <div class="card-body">{{ entry.text|linebreaks }}</div>
        </div>
```

```
{% empty %}
    <p>There are no entries for this topic yet.</p>
{% endfor %}

{% endblock content %}
```

让用户能够将主题设置为公开的完整解决方案见 `ex_20_4_public_topics`。