**Python编程：从入门到实践**

# 变量和简单数据类型

## hello\_world.py

#### hello\_world.py

print("hello Python world!")

## 变量

#### 变量

message="hello Python world!"

print(message)

message="hello Python Crash Course world!"

print(message)



Python将始终记录变量的最新值。

### 变量的命名和使用

只能是字母、数字和下划线，但不能以数字打头。

不能包含空格，但可使用下划线区间单词。

保留关键字。不使用函数名为变量名。

简短和有描述性。

慎用小写字母l和大写字母O。

尽量避免使用大写字母。

### 避免命名错误

当程序无法成功运行时，Python解析器会提供一个traceback，即包含错误信息的记录。

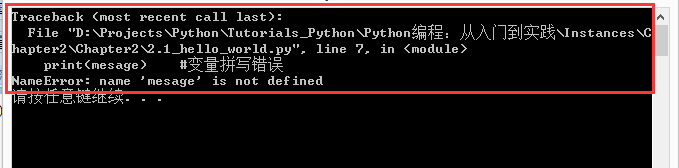
命名错误的原因：

1. 忘记为变量赋值
2. 变量名拼写不正确

#### 命名错误1

message="hello Python Crash Course world!"

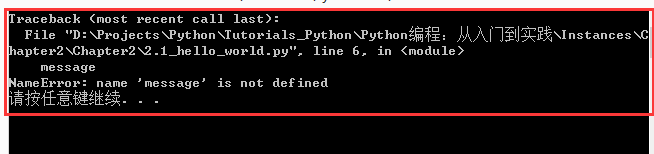
print(mesage) #变量拼写错误



#### 命名错误2

message

print(mesage) #变量没有赋值



## 字符串

使用引号括起。可以是单引号，也可以是双引号。

### 修改大小写

函数

.title() ：以首字母大写的方式显示每个单词。

.upper() ：全部大写

.lower() ：全部小写

#### 修改大小写

name="ada lovelace"

print(name.title()) #Ada Lovelace

print(name.upper()) #ADA LOVELACE

print(name.lower()) #ada lovelace



### 合并字符串

#### 合并字符串

first\_name="ada"

last\_name="lovelace"

full\_name=first\_name+" "+last\_name

print(full\_name) #ada lovelace

message="Hello, "+ full\_name.title()+"!"

print(message) #Hello, Ada Lovelace!

### 添加空白（制表符、换行符）

空白：包括空格、制表符、换行符

制表符：\t

换行符：\n

#### 添加空白1

print("Python")

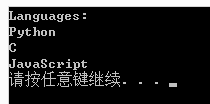
print("\tPython")

print("\nPython")



#### 添加空白2

print("Languages:\nPython\nC\nJavaScript")



#### 添加空白3

print("Languages:\n\tPython\n\tC\n\tJavaScript")



### 删除空白（函数strip()）

.strip() ：删除两侧的空白，字符串中的空白将保留

.lstrip() ：删除左侧的空白

.rstrip() ：删除右侧的空白

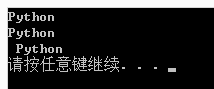
#### 删除空白（函数strip()）

message=" Python "

print(message.strip()) #"Python"

print(message.lstrip()) #"Python "

print(message.rstrip()) #" Python"



### 语法错误

正确使用单引号和双引号。

#### 引号引发的语法错误

message="One of Python's strength is its diverse community"

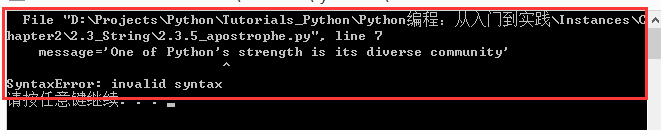
print(message) #正常解析

#引号引发的语法错误：

#错误使用单引号

message='One of Python's strength is its diverse community'

print(message)



## 数字

### 整数

#### 整数运算

print(2 + 3) #加法

print(2 - 3) #减法

print(2 \* 3) #乘法

print(2 / 3) #除法

print(2 \*\* 3) #乘方

#运算优先级

print(2 + 3 \* 4)

print((2 + 3) \* 4)



### 浮点数

#### 浮点运算

print(0.1 + 0.1)

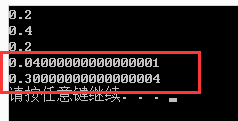
print(0.2 + 0.2)

print(2 \* 0.1)

#浮点之间的运算结果，其小数位数是不确定的

print(0.2 \* 0.2)

print(3 \* 0.1)



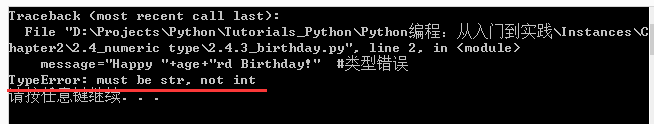
### 数字类型转化为字符串类型（函数.str()）

#### 类型转化（函数.str()）

age = 23

message="Happy "+age+"rd Birthday!" #类型错误

print(message)

。

age = 23

message="Happy "+str(age)+"rd Birthday!"

print(message)



## 注释

Python使用#号进行注释。

注释的目的是阐述代码要做什么，以及如何做。使用自然语言对解决方案进行概述，可以节省时间。

# 列表简介

## 列表List

可以将任何东西加入列表中。

使用方括号“[]”表示，使用逗号“,”分隔当中的元素。

#### 输出整个列表

bicycle=["trek" , "cannondale" , "redline" , "specialized"]

print(bicycle) #将输出中括号及整个列表

### 访问列表元素

指出列表名称和元素的索引。将索引放在方括号中。

起始索引值为0。

索引值-1表示列表最后一个元素。-2表示倒数第二个元素，如此类推。

#### 访问列表元素

bicycle=["trek" , "cannondale" , "redline" , "specialized"]

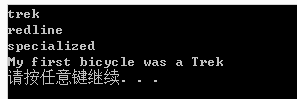
print(bicycle[0]) #trek

print(bicycle[2]) #redline

print(bicycle[-1]) #specialized

message="My first bicycle was a " + bicycle[0].title()

print(message)

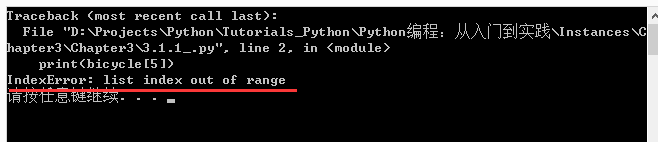


注意：索引超出列表长度时，将抛出异常。

#### 访问列表元素（异常）

bicycle=["trek" , "cannondale" , "redline" , "specialized"]

print(bicycle[5])



## 修改、添加和删除元素

### 修改列表元素

通过索引修改元素。

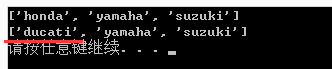
#### 修改列表元素

motorcycles=['honda','yamaha','suzuki']

print(motorcycles)

motorcycles[0]="ducati"

print(motorcycles)



### 添加元素

.append() ：将元素添加到列表末尾

.insert() ：在指定索引位置插入新元素

#### 动态创建列表

motorcycles=[] #创建空列表

motorcycles.append('honda') #逐一添加元素

motorcycles.append('yamaha')

motorcycles.append('suzuki')

print(motorcycles)



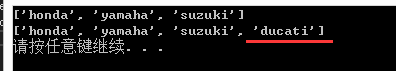
#### 添加元素（方法.append()）

motorcycles=['honda','yamaha','suzuki']

print(motorcycles)

motorcycles.append("ducati")

print(motorcycles)



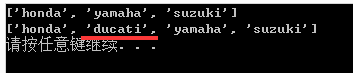
#### 添加元素（方法.insert()）

motorcycles=['honda','yamaha','suzuki']

print(motorcycles)

motorcycles.insert(1,'ducati')

print(motorcycles)



### 删除元素

del list(index) ：删除指定索引处的元素。如索引超出长度，则抛出异常。

.pop(index) ：删除列表指定索引处的元素，并获取该元素的值。如索引超出长度，则抛出异常。

.pop() ：删除列表末尾的元素，并获取末尾元素的值。如列表为空，则抛出异常。

.remove(value) ：删除指定元素值的元素。如索引超出长度，则抛出异常。

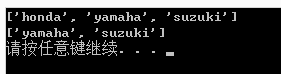
#### 删除元素（del语句）

motorcycles=['honda','yamaha','suzuki']

print(motorcycles)

del motorcycles[0]

print(motorcycles)



#### 删除元素（方法.pop(index)）

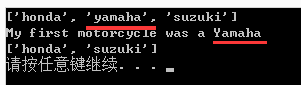
motorcycles=['honda','yamaha','suzuki']

print(motorcycles)

fist\_owned=motorcycles.pop(1)

print("My first motorcycle was a "+fist\_owned.title())

print(motorcycles)



#### 删除元素（方法.pop()）

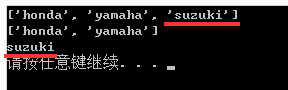
motorcycles=['honda','yamaha','suzuki']

print(motorcycles)

popped\_motorcycle=motorcycles.pop()

print(motorcycles)

print(popped\_motorcycle)



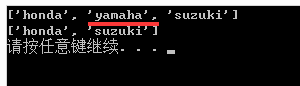
#### 删除元素（方法.remove(value)）

motorcycles=['honda','yamaha','suzuki']

print(motorcycles)

motorcycles.remove('yamaha')

print(motorcycles)



## 组织列表

### 永久性排序（方法.sort()）

.sort() ：永久性修改列表，按字母顺序排列

.sort(reverse=True) ：永久性修改列表，按字母倒序排列

#### 永久性排序（方法.sort()）

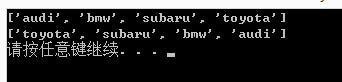
cars=['bmw','audi','toyota','subaru']

cars.sort() #按字母顺序永久排序

print(cars)

cars.sort(reverse=True) #按字母倒序永久排序

print(cars)



### 临时排序（函数sorted()）

sorted(List) ：临时性修改列表，按字母顺序排列

.sorted(reverse=True) ：临时性修改列表，按字母倒序排列

#### 临时性排序（函数sorted()）

cars=['bmw','audi','toyota','subaru']

print("Here is the original list:")

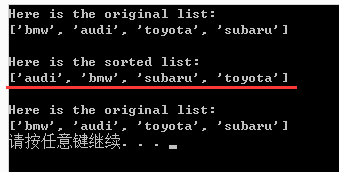
print(cars)

print("\nHere is the sorted list:")

print(sorted(cars))

print("\nHere is the original list:")

print(cars)



#### 临时性倒序（函数sorted()）

cars=['bmw','audi','toyota','subaru']

print("Here is the original list:")

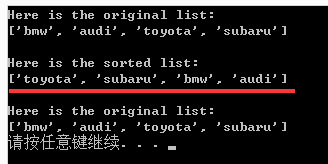
print(cars)

print("\nHere is the sorted list:")

print(sorted(cars,reverse=True))

print("\nHere is the original list:")

print(cars)



### 倒序排列（方法reverse()）

按照列表原来排序的倒序方式排列，不以字母顺序排列。

永久性修改，如要恢复，则可再次调用该方法。

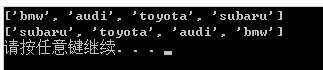
#### 倒序排列（方法reverse()）

cars=['bmw','audi','toyota','subaru']

print(cars)

cars.reverse()

print(cars)



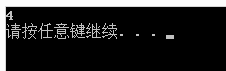
### 确定列表长度（函数len()）

len(list) ：返回列表长度，整数类型

#### 返回列表长度（函数len()）

cars=['bmw','audi','toyota','subaru']

print(len(cars))



# 操作列表

## 遍历列表（for语句）

**for** element **in** list**:**

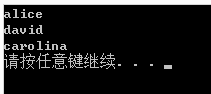
actions…

#### 遍历列表（for语句）

magicians=['alice','david','carolina']

for magician in magicians:

print(magician)



### 在for循环中执行更多操作

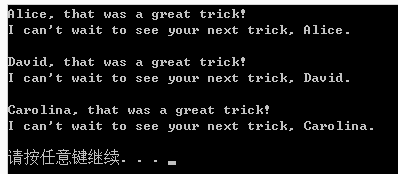
#### 遍历列表（多行代码）

magicians=['alice','david','carolina']

for magician in magicians:

print(magician.title() + ", that was a great trick!")

print("I can't wait to see your next trick, " + magician.title() + ".\n")



## 缩进错误

1. 必须的缩进

将抛出异常，如for语句后没有接缩进的语句。

1. 不需要的缩进

抛出异常。

1. 将语句遗漏缩进或加入缩进

不会异常，但发生逻辑错误。

1. for语句遗漏冒号

语法错误，抛出异常

## 创建数字列表

### 函数range()

range(stop)

range(start, stop[, step])

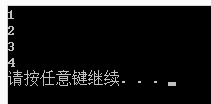
参数说明：

* start: 计数从 start 开始。默认是从 0 开始。例如range（5）等价于range（0， 5）;
* stop: 计数到 stop 结束，但不包括 stop。例如：range（0， 5） 是[0, 1, 2, 3, 4]没有5
* step：步长，默认为1。例如：range（0， 5） 等价于 range(0, 5, 1)

#### 函数range()

for value in range(1,5):

print(value)



### 函数list()转换为列表

list() 方法用于将元组或字符串转换为列表。

注：元组与列表是非常类似的，区别在于元组的元素值不能修改，元组是放在括号中，列表是放于方括号中。

#### 输出偶数

even\_numbers=list(range(2,11,2))

print(even\_numbers) #输出10以内的偶数



#### 平方运算

squares=[]

for value in range(1,11):

squares.append(value\*\*2)

print(squares)



### 简单统计计算

min() ：返回给定参数的最小值，参数可以为序列。

max() ：返回给定参数的最大值，参数可以为序列。

sum() ：对系列进行求和计算。

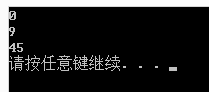
#### 最值和求和计算

digits=[1,2,3,4,5,6,7,8,9,0]

print(min(digits))

print(max(digits))

print(sum(digits))



### 列表解析

列表解析：根据已有列表，高效创建新列表的方式。

#### 列表解析

squares=[value\*\*2 for value in range(1,11)]

print(squares)

先指定表名squares，然后定义一个表达式1，用于生成列表中元素的值，然后定义一个表达式2，用以向表达式1提供值。

表达式中的for语句不需要添加冒号:。

## 切片

#### 切片

players=['chales','martina','michael','florence','eli']

print(players[1:3])



注意：如果方括号中的两个参数所指向的索引号在列表中不存在的，不会抛出异常。

players=['chales','martina','michael','florence','eli']

print(players[:3]) #从列表开头开始到第3个元素



players=['chales','martina','michael','florence','eli']

print(players[3:]) #从第4个元素开始到末尾的元素



players=['chales','martina','michael','florence','eli']

print(players[-3:]) #选取最后三个元素



### 复制列表

#### 复制列表

my\_foods=['pizza','falafel','carrot cake']

friend\_foods=my\_foods[:] #复制列表值（形参复制）

my\_foods.append('cannoli')

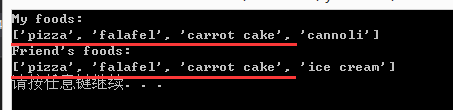
friend\_foods.append('ice cream')

print('My foods:')

print(my\_foods)

print("Friend's foods:")

print(friend\_foods)



## 元组

### 定义

元组，不可变的列表。

使用圆括号标识定义。

使用方括号读取。

修改元组内元素值，会抛出异常。

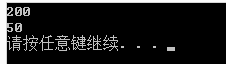
#### 定义和读取元组

dimensions=(200,50) #定义元组

#读取元组元素

print(dimensions[0])

print(dimensions[1])



### 修改元组变量

元组的元素不能修改，但储存元组的变量可以被赋值。

dimensions=(200,50)

print("Original dimensions:")

for dimension in dimensions:

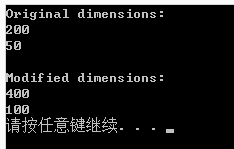
print(dimension)

dimensions=(400,100)

print("\nModified dimensions:")

for dimension in dimensions:

print(dimension)



## 代码格式

PEP8建议。

缩进使用4个空格。

每行不超过80个字符

注释不超过72个字符

# if语句

## if示例

#### if简单示例

cars=['audi','bmw','subaru','toyota']

for car in cars:

if car=='bmw':

print(car.upper())

else:

print(car.title())

## 条件测试

### 检查是否相等

一个等号是陈述；两个等号是发问。

### 大小写

检查时区分大小写。

如只想检查变量的值，可将变量的值转换为小写，再比较。

### 检查是否不相等

检查是否不等：！=

#### 检查是否不相等

requested\_topping = 'mushrooms'

if requested\_topping != 'anchovies': #检查是否相等

print("Hold the anchovies!")



### 比较数字

数字比较：

小于 <

小于等于 <=

大于 >

大于等于 >=

### 检查多个条件

1. 使用and检查多个条件
2. 使用or检查多个条件

注：为改善可读性，将每个条件放在一对括号内。如：

(age >= 21) and (age < =42)

### 检查特定值是否包含在列表中

使用关键字in。

#### 关键字in

requested\_toppings = ['mushrooms', 'onions','pineapple']

if 'mushrooms' in requested\_toppings:

print('True')

### 检查特定值是否不包含在列表中

使用关键字not in。

#### 关键字not in

banned\_users = ['andrew','carolina','david']

user = 'marie'

if user not in banned\_users:

print(user.title() + ', you can post a response if you wish.')

### 布尔表达式

## if语句

### 简单的if语句

#### 简单if语句

age = 19

if age >= 18:

print("You are old enough to vote!")

print("Have you registered to vote yet?")

### if-else语句

#### if-else语句

age = 17

if age>=18:

print('You are old enough to vote!')

print('Have you registered to vote yet?')

else:

print('Sorry, you are too young to vote.')

print('Please register to vote as soon as you turn 18!')

### if-elif-else结构

需要检查超过两个条件时使用。

#### if-elif-else结构

age = 12

if age < 4:

print('You admission cost is $0.')

elif age < 18:

print('Your admission cost is $5.')

else:

print('Your admission cost is $10.')



效率修改：

age = 12

if age < 4:

price = 0

elif age < 18:

price = 5

else:

price = 10

print('Your admission cost is $' + str(price) +'.')

### 使用多个elif代码块

判断任意数量的条件时，使用任意数量的elif代码块。

### 省略else代码块

else语句是兜底代码块，只要不满足任何if或elif条件测试，都会执行else代码块，可能会面临无效或恶意的数据。

应该仔细分析各个条件，尽量使用elif代码块代替else代码块。

#### 省略else代码块

age = 12

if age < 4:

price = 0

elif age < 18:

price = 5

elif age < 65:

price =10

elif age >=65:

price = 5

print('Your admission cost is $' + str(price) +'.')

### 测试多个条件

使用多个if代码块测试一个特定的条件。

#### 判断多个条件

requested\_toppings = ['mushrooms','extra cheese']

if 'mushrooms' in requested\_toppings:

print("Adding mushrooms.")

if 'pepperoni' in requested\_toppings:

print("Adding pepperoni.")

if 'extra cheese' in requested\_toppings:

print('Adding extra cheese.')

print('\nFinished making your pizza!')



如果想执行一个代码块，使用if-elif-else结构。

如果想执行多个代码块，使用一系列独立的if语句。

## 使用if语句处理列表

### 检查特殊元素

#### 检查特殊元素

requested\_toppings = ['mushrooms','green peppers','extra cheese']

for requested\_topping in requested\_toppings:

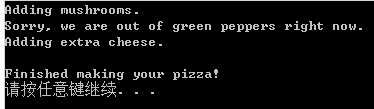
if requested\_topping == 'green peppers':

print('Sorry, we are out of green peppers right now.')

else:

print('Adding ' + requested\_topping + '.')

print('\nFinished making your pizza!')



### 确定列表不为空

#### 确定列表不为空

requested\_toppings = []

if requested\_toppings: #判断列表是否为空，若列表至少包含一个元素时返回True；若为空，则返回False

for requested\_topping in requested\_toppings:

print('Adding ' + requested\_topping + '.')

print('\nFinished making your pizza!')

else:

print("Are you sure you want a plain pizza?")



### 使用多个列表

#### 使用多个列表

available\_toppings = ['mushrooms', 'olives','green peppers','pepperoni','pineapple','extra cheese']

requested\_toppings = ['mushrooms','frech fries','extra cheese']

for requested\_topping in requested\_toppings:

if requested\_topping in available\_toppings:

print('Adding ' + requested\_topping + '.')

else:

print("Sorry, we don't have " + requested\_topping + ".")

print("\nFinished making your pizza!")

# 字典

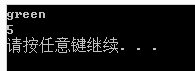
## 简单的字典

#### 字典

alien\_0 = {'color':'green','points':5}

print(alien\_0['color'])

print(alien\_0['points'])



## 使用字典

字典是一系列键-值对。

使用花括号“{}”标识。

键-值对之间使用逗号分隔。

alien\_0 = {'color':'green','points':5}

### 访问字典的值

使用键名。

使用方括号“[]”标识。

print(alien\_0['color'])

注意：如果使用的键名不存在，则抛出异常。

### 添加键-值对

#### 添加键-值对

alien\_0 = {'color':'green','points':5}

print(alien\_0)

alien\_0['x\_position'] = 0

alien\_0['y\_position'] = 25

print(alien\_0)



注意：Python不关心键-值对的添加顺序，只关心键和值之间的关联关系。

### 创建一个空字典

通行做法：先创建一个空字典，然后逐一添加各个键-值对。

#### 创建空字典

alien\_0 = {} #创建空字典

#逐一添加键-值对

alien\_0['color'] = 'green'

alien\_0['points'] = 5

print(alien\_0)



### 修改字典中的值

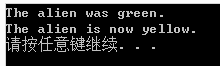
#### 修改字典中的值

alien\_0 = {'color':'green'}

print("The alien was " + alien\_0['color'] + ".")

alien\_0['color'] = 'yellow'

print('The alien is now ' + alien\_0['color'] + ".")



#### 修改字典中的值2

alien\_0 = {'x\_position':0, 'y\_position':25,'speed':'medium'}

print("Original x-position: " + str(alien\_0['x\_position']))

if alien\_0['speed'] == 'slow':

x\_increment = 1

elif alien\_0['speed'] == 'medium':

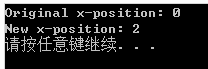
x\_increment = 2

else:

x\_increment = 3

alien\_0['x\_position'] = alien\_0['x\_position'] + x\_increment

print("New x-position: " + str(alien\_0['x\_position']))



### 删除键-值对

使用del语句。

#### 删除键-值对

alien\_0 = {'color':'green','points':5}

print(alien\_0)

del alien\_0['points']

print(alien\_0)



### 由类似对象组成的字典

建议使用多行来定义字典。

#### 使用多行定义字典

#使用多行来定义字典

favorite\_languages = {

'jen':'python',

'sarah':'c',

'edward':'ruby',

'phil':'python'

}

#使用多行来编写代码

print("Sarah's favorite language is " +

favorite\_languages['sarah'].title() +

"."

)

## 遍历字典

### 遍历所有的键-值对

使用方法items()

#### 遍历所有键-值对（方法.item()）

user\_0 = {

'username' : 'efermi',

'first':'enrico',

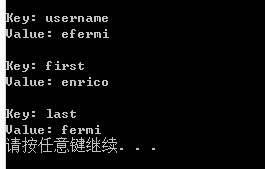
'last':'fermi'

}

for key,value in user\_0.items():

print("\nKey: " + key)

print("Value: " + value)



### 遍历字典中的所有键

方法keys()

#### 遍历字典中的所有键

favorite\_languages = {

'jen':'python',

'sarah':'c',

'edward':'ruby',

'phil':'python'

}

for name in favorite\_languages.keys():

print(name.title())

for name in favorite\_languages: #与使用方法keys()效果等价

print(name.title())

注意：方法keys()返回一个列表。

### 按顺序遍历字典中的所有键

使用函数sorted()

#### 按顺序遍历字典中的所有键（函数sorted()）

favorite\_languages = {

'jen':'python',

'sarah':'c',

'edward':'ruby',

'phil':'python'

}

for name in sorted(favorite\_languages.keys()):

print(name.title() + ",thank you for taking the poll.")

### 遍历字典中所有的值

使用方法values()。

#### 遍历字典所有的值（values()）

favorite\_languages = {

'jen':'python',

'sarah':'c',

'edward':'ruby',

'phil':'python'

}

print("The following languages have been mentioned:")

for language in favorite\_languages.values():

print(language.title())

使用函数set()，提取不重复的唯一值的集合。

#### 获取不重复值（函数set()）

favorite\_languages = {

'jen':'python',

'sarah':'c',

'edward':'ruby',

'phil':'python'

}

print("The following languages have been mentioned:")

for language in favorite\_languages.values():

print(language.title())

print("\nThe following languages have been mentioned:")

for language in set(favorite\_languages.values()):

print(language.title())

## 嵌套

### 字典列表

#### 字典列表

alien\_0 = {'color':'green','points':5}

alien\_1 = {'color':'yellow','points':10}

alien\_2 = {'color':'red','points':15}

aliens = [alien\_0,alien\_1,alien\_2]

for alien in aliens:

print(alien)

#### 字典列表2

#创建一个用于存储外星人的空列表

aliens = []

#创建30个绿色外星人

for alien\_number in range(30): #该处的函数range()仅仅是告诉Python循环的次数

new\_alien = {'color':'green','points':5,'speed':'slow'}

aliens.append(new\_alien)

#显示前5个外星人

for alien in aliens[:5]:

print(alien)

print('...')

#显示创建了多少个外星人

print('Total number of aliens: ' + str(len(aliens)))

#### 字典列表3

#创建一个用于存储外星人的空列表

aliens = []

#创建30个绿色外星人

for alien\_number in range(30): #该处的函数range()仅仅是告诉Python循环的次数

new\_alien = {'color':'green','points':5,'speed':'slow'}

aliens.append(new\_alien)

#修改前3个外星人

for alien in aliens[0:3]:

if alien['color'] == 'green':

alien['color'] = 'yellow'

alien['speed'] = 'medium'

alien['points'] = 10

#显示前5个外星人

for alien in aliens[:5]:

print(alien)

print('...')

#显示创建了多少个外星人

print('Total number of aliens: ' + str(len(aliens)))

### 在字典中存储列表

将列表作为字典的值，存储在字典中。

#### 在字典中存储列表

#存储所点pizza的信息

pizza = {'crust':'thick',

'toppings':['mushroom','extra cheese']

}

print("You ordered a " + pizza['crust'] + "-crust pizza" +

"with the following toppings:")

for topping in pizza['toppings']:

print("\t" + topping)

#### 在字典中存储列表2

favorite\_languages = {

'jen':['python','ruby'],

'sarah':['c'],

'edward':['ruby','go'],

'phil':['python','haskell']

}

for name,languages in favorite\_languages.items():

if len(languages) == 1:

print('\n' + name.title() + "'s favorite language is:" +

'\t' + languages[0].title()

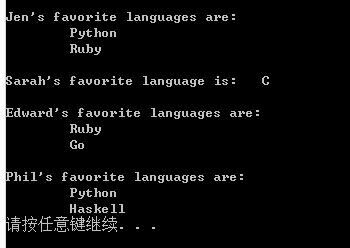
)

else:

print("\n" + name.title() + "'s favorite languages are:")

for language in languages:

print("\t" + language.title())



### 在字典中存储字典

#### 在字典中存储字典

users = {

'aeinstein':{

'first':'albert',

'last':'einstein',

'location':'princeton',

},

'mcurie':{

'first':'marie',

'last':'curie',

'location':'paris'

},

}

for username,user\_info in users.items():

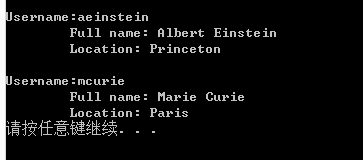
print("\nUsername:" + username)

full\_name = user\_info['first'] + " " + user\_info['last']

location = user\_info['location']

print("\tFull name: " + full\_name.title())

print("\tLocation: " + location.title())



# 用户输入和while循环

## 函数input()工作原理

函数input获取的输入信息类型为字符串。

#### 函数input()

message = input("Tell me something, and I will repeat it back to you:")

print(message)



#### 提示信息

name = input("please enter your name:")

print("Hello, " + name + '!')



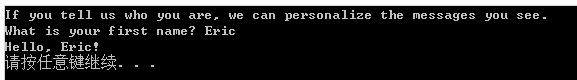
提供多行提示信息

prompt = 'If you tell us who you are, we can personalize the messages you see.'

prompt += '\nWhat is your first name? '

name = input(prompt)

print("Hello, " + name + '!')



### 获取输入的数值

使用函数int()，可以将输入的字符串转为数值类型。

#### 将字符串转为数值类型（函数int()）

age = input("How old are you?")

age = int(age)

print(age)

### 求模运算

使用运算符%

#### 判断奇偶数

number = input("Enter a number, and I'll tell you if it is even or odd:")

number = int(number)

if number % 2 == 0:

print("\nThe number " + str(number) + ' is evnen.')

else:

print("\nThe number " + str(number) + ' is odd.')

## while循环

for循环用于针对集合中每个元素。

while循环针对指定的条件判断。

### 使用while循环

#### while循环

current\_number = 1

while current\_number <= 5:

print(current\_number)

current\_number += 1

### while与input结合实例

#### while与input结合应用

prompt = "\nTell me something, and I will repeat it back to you:"

prompt += "\nEnter 'quit' to end the program."

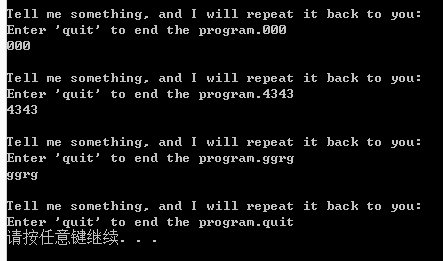
message = '' #初始化变量message，使之在第一次判断中能够被使用

while message != 'quit':

message = input(prompt)

if message != 'quit':

print(message)



### 使用标志

可以在代码中使用一个专门的变量作为判断while条件是否满足的标志。

使用独立的代码行进行判断，然后将判断结果赋值给标志，再在while代码中对该标志进行判断。

#### 使用标志

prompt = "\nTell me something, and I will repeat it back to you:"

prompt += "\nEnter 'quit' to end the program."

message = '' #初始化变量message，使之在第一次判断中能够被使用

active = True

while active == True:

message = input(prompt)

if message != 'quit':

print(message)

else:

active = False

### break语句

跳出循环。

#### 跳出循环break语句

prompt = "\nPlease enter the name of a city you have visited:"

prompt += "\n(Enter 'quit' when you are finished.)"

while True:

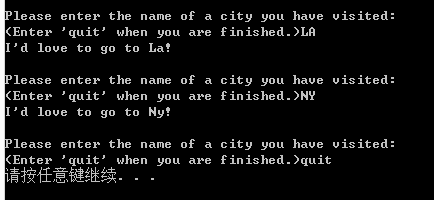
city = input(prompt)

if city == 'quit':

break

else:

print("I'd love to go to " + city.title() + "!")



### continue语句

跳过一次循环。

#### 跳出一次循环（continue语句）

current\_number = 0

while current\_number < 10:

current\_number +=1

if current\_number % 2 ==0: #当为偶数时，跳出一次循环

continue

print(current\_number) #当跳出循环时，跳过continue以后的代码



### 避免无限循环

确认程序至少有一行代码让循环条件为False或执行break语句。

## 使用while循环处理列表和字典

for循环可以遍历列表，但不方便修改列表。

while循环在遍历列表的同时对其进行修改。

### 在列表之间移动元素

#### 在列表之间移动元素

#一个待验证的用户列表

unconfirmed\_users = ['alice','brian','candace']

#一个用于存储已验证用户的空列表

confirmed\_users = []

#判断待验证用户列表是否为空，然后执行列表间元素的移动

while unconfirmed\_users:

current\_user = unconfirmed\_users.pop() #将待验证用户列表末尾删除用户，并赋值给中间变量

print("Verifing user: " + current\_user.title()) #此处可以替换成执行数据操作的代码

confirmed\_users.append(current\_user) #将中间变量添加到已验证用户的空列表。

print('\nThe following users have been confirmed:')

for confirmed\_user in confirmed\_users:

print(confirmed\_user.title())



### 删除包含特定值的所有列表元素

#### 删除包含特定值的所有列表元素

pets = ['dog','cat','dog','goldfish','cat','rabbit','cat']

print(pets)

#执行删除代码，直到列表中不再包含特定值

while 'cat' in pets:

pets.remove('cat')

print(pets)



### 使用用户输入值填充字典

#### 将输入值插入字典

responses = {}

#设置标识，指示循环是否继续

polling\_active = True

while polling\_active:

#将用户输入值分别赋值给的变量

name = input("\nWhat is your name?")

response = input("Which mountain would you like to climb someday?")

#将变量值分别作为键-值，赋给字典

responses[name] = response

#判断是否继续循环

repeat = input("Woulde you like to let another person respond? (yes/no)")

if repeat == 'no':

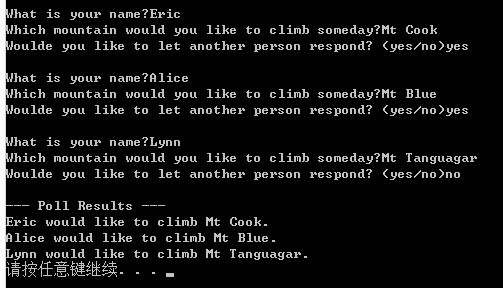
polling\_active = False

#输入结束时显示结果

print("\n--- Poll Results ---")

for name,response in responses.items():

print(name + " would like to climb " + response + ".")



# 函数

函数是带名字的代码块，用于完成具体的工作。

## 定义函数

使用关键字def。

定义以冒号结尾。

文档字符串（docstring），在函数体内，用于描述函数是做什么的。使用三引号括起。

#### 定义函数

def greet\_user(): #使用def关键字定义函数体；括号内可以带任意参数

"""显示简单的问候语""" #文档字符串

print("Hello!") #函数体内执行命令的代码块

#可以使用return关键字返回值

greet\_user() #调用函数



### 向函数传递信息

#### 向函数传递参数

def greet\_user(username): #使用def关键字定义函数体；括号内可以带任意参数

"""显示简单的问候语""" #文档字符串

print("Hello, " + username.title() + "!") #函数体内执行命令的代码块

#可以使用return关键字返回值

greet\_user('jessie') #调用函数



### 实参和形参

在定义函数时，向函数传递的变量为形参。

在调用函数时，向函数传递的变量为实参。

## 传递实参

### 位置实参

基于函数定义中的形参顺序。

#### 位置实参

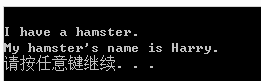
def describe\_pet(animal\_type,pet\_name):

""" 显示宠物的信息 """

print("\nI have a " + animal\_type + '.')

print("My " + animal\_type + "'s name is " + pet\_name.title() + ".")

describe\_pet('hamster','harry')



注意：如实参顺序不正确，将可能出现逻辑错误。

### 关键字实参

即向函数传递“名称-值”对。

无需考虑实参顺序。

#### 关键字实参

def describe\_pet(animal\_type,pet\_name):

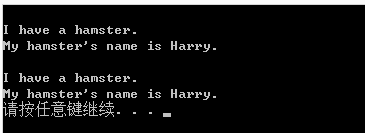
""" 显示宠物的信息 """

print("\nI have a " + animal\_type + '.')

print("My " + animal\_type + "'s name is " + pet\_name.title() + ".")

describe\_pet(animal\_type='hamster',pet\_name='harry')

describe\_pet(pet\_name='harry',animal\_type='hamster') #与实参顺序无关



注意：必须准确适用函数定义中的形参名。否则会抛出异常。

### 默认值

在定义函数时，可以为形参指定默认值。

在调用函数时，函数将使用实参值；如果没有赋实参值，将使用默认值。

#### 使用默认值为参数

def describe\_pet(pet\_name,animal\_type='dog'):

""" 显示宠物的信息 """

print("\nI have a " + animal\_type + '.')

print("My " + animal\_type + "'s name is " + pet\_name.title() + ".")

describe\_pet(pet\_name='willie')



注意：使用默认值时，在形参列表中必须先列出没有默认值的形参，再列出有默认值的形参。

### 等效的函数调用

### 避免实参错误

调用函数时，向函数传递的实参个数与函数需要的信息个数不一致时，可能多于或少于，都会引起异常。

## 返回值

使用return语句，将值返回到调用函数的代码行。

### 返回简单值

#### 返回简单值

def get\_formatted\_name(first\_name,last\_name):

"""返回全名"""

full\_name = first\_name + ' ' + last\_name

return full\_name.title() #使用return关键字，返回值

musician = get\_formatted\_name('jimi','hendrix')

print(musician)

### 可选的实参

使用默认值参数。

#### 可选的实参

def get\_formatted\_name(first\_name,last\_name,middle\_name=''):

"""返回全名"""

if middle\_name: #判断字符串参数是否为空，Python将非空字符串定义为True

full\_name = first\_name + ' ' + middle\_name + ' ' + last\_name

else:

full\_name = first\_name + ' ' + last\_name

return full\_name.title() #使用return关键字，返回值

musician = get\_formatted\_name('jimi','hendrix')

print(musician)

musician = get\_formatted\_name('john','hooker','lee')

print(musician)



### 返回字典

#### 返回字典

def build\_person(first\_name,last\_name,age=''):

"""返回一个包含有个人信息的字典"""

person = {'first':first\_name,'last':last\_name}

if age: #判断实参是否为空

person['age'] = age

return person

musician = build\_person('jimi','hendrix',27)

print(musician)

### 函数与while循环

#### 函数与while循环

def get\_formatted\_name(first\_name,last\_name):

"""返回全名"""

full\_name = first\_name + ' ' + last\_name

return full\_name.title() #使用return关键字，返回值

while True: #注意，该处为一个无限循环，以下使用break语句跳出循环

print('\nPlease tell me your name:')

print("(enter 'q' at any time to quit)")

f\_name = input("First name: ")

if f\_name == 'q':

break #使用break语句跳出循环

l\_name = input("Last name: ")

if l\_name == 'q':

break

formatted\_name = get\_formatted\_name(f\_name,l\_name)

print("\nHello, " + formatted\_name + "!")

## 传递列表

可以将列表作为参数，向函数传递值。

### 在函数中修改列表

#### 在函数中修改列表

#定义函数：操作列表

def print\_models(unprinted\_designs,completed\_models):

"""

根据输入的未完成列表执行相应的操作，

然后将完成操作的添加到已完成列表

"""

while unprinted\_designs: #当未完成列表还存在元素时

current\_design = unprinted\_designs.pop() #从列表末尾取出元素，并在列表中删除该元素

#对取出的元素进行操作

print("Printing model: " + current\_design)

#将完成操作的元素添加到完成的列表

completed\_models.append(current\_design)

#定义函数：显示完成的列表

def show\_completed\_model(completed\_models):

''' 显示完成的列表 '''

print("\nThe following models have been printed: ")

for completed\_model in completed\_models:

print(completed\_model)

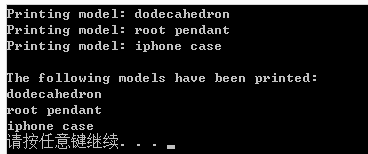
#程序主体

unprinted\_designs = ['iphone case','root pendant','dodecahedron'] #未完成列表

completed\_models = [] #完成列表

print\_models(unprinted\_designs,completed\_models)

show\_completed\_model(completed\_models)



注意：每个函数应只负责一项具体的工作。

### 禁止函数修改列表

如不想在函数中修改列表，可以使用列表副本传递给函数：

*function\_name(list\_name[ : ])*

但是，使用副本传递给函数，可能会要求额外的时间或开销进行复制，影响效率。尤其是处理大型列表时。

## 传递任意数量的实参

预先不确定函数需要的实参数量。

可以使用形参名\*parament。实际上此时parament为一个空元组，将接收到的所有值都封装到元组内。

#### 传递任意数量的实参

def make\_pizza(\*toppings):

"""打印所有配料"""

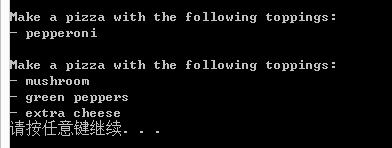
print("\nMake a pizza with the following toppings:")

for topping in toppings:

print("- " + topping)

make\_pizza('pepperoni')

make\_pizza('mushroom','green peppers','extra cheese')



### 位置实参与任意数量实参

如果函数需要接受不同类型的实参，则必须将接纳任意数量实参的形参放在最后。

#### 位置实参与任意数量实参

def make\_pizza(size,\*toppings):

"""打印所有配料"""

print("\nMake a " + str(size) +

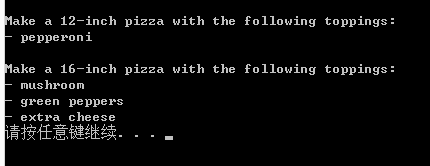
"-inch pizza with the following toppings:")

for topping in toppings:

print("- " + topping)

make\_pizza(12,'pepperoni')

make\_pizza(16,'mushroom','green peppers','extra cheese')



### 使用任意数量的关键字实参

可以将参数编写成键-值对；

数量可以是任意；

相当于传递一个字典。

使用\*\*标识形参。

#### 使用任意数量的关键字实参

def build\_profile(first, last,\*\*user\_info):

""" 创建一个字典存储用户信息 """

profile = {}

profile['first\_name'] = first

profile['last\_name'] = last

for key,value in user\_info.items():

profile[key] = value

return profile

user\_profile = build\_profile('albert','einstein',

location='princeton',

field='physics')

print(user\_profile)

## 将函数存储在模块中

将函数存储在模块（独立文件中），可以将代码块与主程序分离。

使用import语句将模块导入主程序。

### 导入整个模块

使用import语句：

**import** *module\_name*

调用函数：

*module\_name.function\_name()*

#### 导入整个模块

创建模块文件pizza

def make\_pizza(\*toppings):

"""打印所有配料"""

print("\nMake a pizza with the following toppings:")

for topping in toppings:

print("- " + topping)

创建主程序文件making\_pizza

import pizza

pizza.make\_pizza('pepperoni')

pizza.make\_pizza('mushroom','green peppers','extra cheese')

### 导入特定函数

导入单个函数

**from** *module\_name* **import** *function\_name*

导入多个函数

**from** *module\_name* **import** *function\_0, function\_1, function\_2, …*

调用函数：

直接在主程序中调用

*function\_name*

### 指定函数别名

导入函数：

**from** *module\_name* **import** *function\_name* **as** *fn*

调用函数：

*fn.function\_name*

### 指定模块别名

导入模块：

**import** *module\_name* **as** *mn*

调用函数：

*mn.function\_name*

### 导入模块中的所有函数

**from** *module\_name* **import** *\**

注意：要慎用该语句，可能会因为多个函数拥有相同的函数名而发生冲突。

# 类

类

对象

实例

## 创建和使用类

### 创建类

在python中，首字母大写的名称专指的是类。类的定义中的括号是空的。

构造函数\_\_init\_\_()，形参self必不可少，必须位于其他形参前面。

定义变量时都有前缀self，以self为前缀的变量都在类中所有方法使用。

#### 创建类

class Dog(): #类名首字母大写，括号内为空

"""创建类"""

def \_\_init\_\_(self,name,age): #定义构造函数，第一个形参必须为self

""" 初始化属性name和age """

self.name = name #定义属性

self.age = age

def sit(self): #定义方法，第一个形参为self

""" 定义sit方法 """

print(self.name.title() + " is now sitting.")

def roll\_over(self):

""" 定义roll over方法 """

print(self.name.title() + " rolled over!")

### 创建实例

访问实例属性和调用方法，使用句点标识法。

#### 定义类并创建实例

class Dog(): #类名首字母大写，括号内为空

"""创建类"""

def \_\_init\_\_(self,name,age): #定义构造函数，第一个形参必须为self

""" 初始化属性name和age """

self.name = name #定义属性

self.age = age

def sit(self): #定义方法，第一个形参为self

""" 定义sit方法 """

print(self.name.title() + " is now sitting.")

def roll\_over(self):

""" 定义roll over方法 """

print(self.name.title() + " rolled over!")

my\_dog = Dog('willie',6) #创建实例

print("My dog's name is " + my\_dog.name.title() + ".") #访问实例属性

print("My dog is " + str(my\_dog.age) + " years old.") #调用实例方法

## 使用类和实例

### 定义类

#### 定义类

class Car():

""" 定义Car类 """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

my\_new\_car = Car('audi','a4',2016)

print(my\_new\_car.get\_descriptive\_name())

### 指定属性默认值

在构造函数\_\_init\_\_()中定义。

#### 指定属性默认值

class Car():

""" 定义Car类 """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

my\_new\_car = Car('audi','a4',2016)

print(my\_new\_car.get\_descriptive\_name())

my\_new\_car.read\_odometer()

### 修改属性的值

1.直接修改属性值

#### 直接修改属性值

class Car():

""" 定义Car类 """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

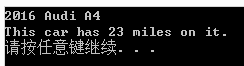
print("This car has " + str(self.odometer\_reading) + " miles on it.")

my\_new\_car = Car('audi','a4',2016)

print(my\_new\_car.get\_descriptive\_name())

my\_new\_car.odometer\_reading = 23

my\_new\_car.read\_odometer()



2.通过方法修改

#### 通过方法修改

class Car():

""" 定义Car类 """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

self.odometer\_reading = mileage

my\_new\_car = Car('audi','a4',2016)

print(my\_new\_car.get\_descriptive\_name())

#my\_new\_car.odometer\_reading = 23

my\_new\_car.update\_odemeter(23)

my\_new\_car.read\_odometer()

#### 限定属性修改范围

class Car():

""" 定义Car类 """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

my\_new\_car = Car('audi','a4',2016)

print(my\_new\_car.get\_descriptive\_name())

#my\_new\_car.odometer\_reading = 23

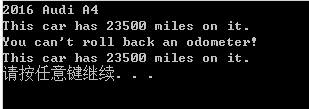
#my\_new\_car.update\_odemeter(23)

my\_new\_car.update\_odemeter(23500)

my\_new\_car.read\_odometer()

my\_new\_car.update\_odemeter(100) #将触发禁止回调的代码

my\_new\_car.read\_odometer()



#### 对属性值进行运算

class Car():

""" 定义Car类 """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odemeter(self,miles):

""" 增加里程 """

self.odometer\_reading += miles

my\_new\_car = Car('audi','a4',2016)

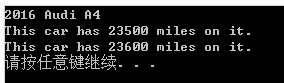
print(my\_new\_car.get\_descriptive\_name())

my\_new\_car.update\_odemeter(23500)

my\_new\_car.read\_odometer()

my\_new\_car.increment\_odemeter(100)

my\_new\_car.read\_odometer()



## 继承

一个类继承另一个类时，将自动获取另一个类的所有属性和方法；原有的类称为父类，新类称为子类。子类继承了父类所有的属性和方法，同时还可以定义自己的属性和方法。

创建子类时，父类必须包含在当前文件中，且位于子类前面。

在括号内指定父类的名称。

定义构造函数，指明接受父类所需的信息范围。

可以使用函数super()将父类和子类关联起来，使子类包含父类的属性。

### 子类的构造函数\_\_init\_\_()

#### 子类的构造函数\_\_init\_\_()

class Car():

""" 定义父类Car """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odemeter(self,miles):

""" 增加里程 """

self.odometer\_reading += miles

class ElectricCar(Car):

""" 定义子类 """

def \_\_init\_\_(self,make,model,year):

"""" 初始化父类的属性 """

super().\_\_init\_\_(make,model,year)

my\_tesla = ElectricCar('tesla','model s', 2016)

print(my\_tesla.get\_descriptive\_name())



### Python 2.7

### 定义子类的属性和方法

#### 定义子类的属性和方法

class Car():

""" 定义父类Car """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odemeter(self,miles):

""" 增加里程 """

self.odometer\_reading += miles

class ElectricCar(Car):

""" 定义子类 """

def \_\_init\_\_(self,make,model,year):

""""

初始化父类的属性

定义子类属性

"""

super().\_\_init\_\_(make,model,year)

self.battery\_size = 70

def describe\_battery(self):

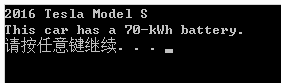
""" 定义子类方法 """

print("This car has a " + str(self.battery\_size) + "-kWh battery.")

my\_tesla = ElectricCar('tesla','model s', 2016)

print(my\_tesla.get\_descriptive\_name())

my\_tesla.describe\_battery()



### 重写父类的方法

可以在子类中重写父类方法，它要与父类中的方法同名。

### 将实例用做属性

#### 将实例用做属性

class Car():

""" 定义父类Car """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odemeter(self,miles):

""" 增加里程 """

self.odometer\_reading += miles

class Battery():

""" 定义电池类 """

def \_\_init\_\_(self,battery\_size=70):

""" 构造函数初始化 """

self.battery\_size = battery\_size

def describe\_battery(self):

""" 定义电池方法 """

print("This car has a " + str(self.battery\_size) + "-kWh battery.")

class ElectricCar(Car):

""" 定义子类 """

def \_\_init\_\_(self,make,model,year):

""""

初始化父类的属性

定义子类属性

"""

super().\_\_init\_\_(make,model,year)

self.battery = Battery() #使用实例作为类的属性

my\_tesla = ElectricCar('tesla','model s', 2016)

print(my\_tesla.get\_descriptive\_name())

my\_tesla.battery.describe\_battery() #当需要对电池进行描述时，没有在ElectricCar中定义方法，而是直接调用Battery中的方法。

注意：该处将Battery类的实例直接作为ElectricCar的battery属性，好处在于可以直接对Battery类进行修改，而无需对ElectricCar类进行修改，起到解耦的效果。

## 导入类

可以将类存储在模块中，然后在主程序中导入需要的模块。

### 导入单个类

**from** *文件名* **import** *类名*

在模块的开始处，包含一个模块级文档字符串，对该模块的内容进行简要描述。

#### 导入单个类

创建car.py文件：

class Car():

""" 定义父类Car """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odemeter(self,miles):

""" 增加里程 """

self.odometer\_reading += miles

创建主程序文件my\_car.py

from car import Car

my\_new\_car = Car('audi','a4',2016)

print(my\_new\_car.get\_descriptive\_name())

my\_new\_car.odometer\_reading = 23

my\_new\_car.read\_odometer()

### 在一个模块中存储多个类

#### 在一个模块中存储多个类

创建car.py文件：放入多个类

class Car():

""" 定义父类Car """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odemeter(self,miles):

""" 增加里程 """

self.odometer\_reading += miles

class Battery():

""" 定义电池类 """

def \_\_init\_\_(self,battery\_size=70):

""" 构造函数初始化 """

self.battery\_size = battery\_size

def describe\_battery(self):

""" 定义电池方法 """

print("This car has a " + str(self.battery\_size) + "-kWh battery.")

class ElectricCar(Car):

""" 定义子类 """

def \_\_init\_\_(self,make,model,year):

""""

初始化父类的属性

定义子类属性

"""

super().\_\_init\_\_(make,model,year)

self.battery = Battery() #使用实例作为类的属性

创建主程序文件my\_electric\_car.py：

from car import ElectricCar

my\_tesla = ElectricCar('tesla','model s', 2016)

print(my\_tesla.get\_descriptive\_name())

my\_tesla.battery.describe\_battery() #当需要对电池进行描述时，没有在ElectricCar中定义方法，而是直接调用Battery中的方法。

### 从一个模块中导入多个类

from car import ElectricCar, Car

### 导入整个模块

导入：

**import** *module\_name*

调用：

*实例名 = module\_name.class\_name*

### 导入模块所有类

**from** *module\_name* **import** *\**

不推荐，可能因为重名而引发冲突。

需要从一个模块导入多个类的，最好导入整个模块，并使用*module\_name.class\_name*语法访问类。

### 在一个模块中导入另一个模块

#### 在一个模块中导入另一个模块

创建模块文件car.py：

class Car():

""" 定义父类Car """

def \_\_init\_\_(self, make, model, year):

""" 定义Car类构造函数 """

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 #指定属性odometer\_reading的默认值为0

def get\_descriptive\_name(self):

""" 返回Car信息 """

long\_name = str(self.year) + ' ' + self.make + ' ' + self.model

return long\_name.title()

def read\_odometer(self):

""" 读取里程数 """

print("This car has " + str(self.odometer\_reading) + " miles on it.")

def update\_odemeter(self,mileage):

""" 修改里程数 """

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odemeter(self,miles):

""" 增加里程 """

self.odometer\_reading += miles

创建模块文件electric\_car.py：

from car import Car

class Battery():

""" 定义电池类 """

def \_\_init\_\_(self,battery\_size=70):

""" 构造函数初始化 """

self.battery\_size = battery\_size

def describe\_battery(self):

""" 定义电池方法 """

print("This car has a " + str(self.battery\_size) + "-kWh battery.")

class ElectricCar(Car):

""" 定义子类 """

def \_\_init\_\_(self,make,model,year):

""""

初始化父类的属性

定义子类属性

"""

super().\_\_init\_\_(make,model,year)

self.battery = Battery() #使用实例作为类的属性

创建主程序文件：

from car import Car

from electric\_car import ElectricCar

my\_new\_car = Car('audi','a4',2016)

print(my\_new\_car.get\_descriptive\_name())

my\_tesla = ElectricCar('tesla', 'roadster',2016)

print(my\_tesla.get\_descriptive\_name())



## Python标准库

Python标准库是一组模块。

#### 引用标准库

from collections import OrderedDict #导入标准库

favorite\_languages = OrderedDict()

favorite\_languages['jen'] = 'python'

favorite\_languages['sarah'] = 'c'

favorite\_languages['edward'] = 'ruby'

favorite\_languages['phil'] = 'python'

for name, language in favorite\_languages.items():

print(name.title() + "'s favorite language is " +

language.title() + '.')

# 文件和异常

## 读取文本文件

### 读取整个文件

#### 读取整个文本文件

#通常使用函数open()后需要调用函数close()关闭文件。

#使用关键字with后，Python将自动在合适的时候自动关闭。

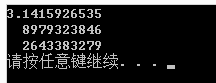
#使用函数open()打开文件，第一个参数为文件的路径和名称

with open('Txt\pi\_digits.txt') as file\_object:

#使用方法read()来读取文件全部内容。

contents = file\_object.read()

print(contents.rstrip())



### 文件路径

相对文件路径

绝对文件路径

注意，在Windows下路径使用的是反斜杠（\），反斜杠在Python中被视为转移符。所以应以原始字符串的方式指定路径，即在开头的单引号前加上r。

### 逐行读取

以每次一行的方式检查文件。结合使用for循环。

#### 逐行读取文本文件

filename = r"Txt\pi\_digits.txt"

with open(filename) as file\_object:

#逐行读取数据

for line in file\_object:

print(line.strip())

### 将文本数据读取至列表

#### 读取文本数据值列表

filename = r"Txt\pi\_digits.txt"

with open(filename) as file\_object:

#读取所有数据到列表lines中

lines = file\_object.readlines()

for line in lines:

print(line.rstrip())

### 操作读取的数据

#### 操作读取的数据

filename = r"Txt\pi\_digits.txt"

with open(filename) as file\_object:

#读取所有数据到列表lines中

lines = file\_object.readlines()

pi\_string = ''

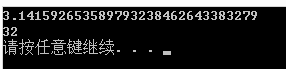
#将读取的数据合并为一行

for line in lines:

pi\_string += line.strip()

print(pi\_string)

print(len(pi\_string))



注意：Python读取文本文件时，默认读取为字符串。可以根据实际需要，使用函数int()转换为整数，或使用函数float()转换为浮点数。

### 读取大型文件

#### 读取大型文件

filename = r"Txt\pi\_million\_digits.txt" #包含有1百万位的圆周率文件

with open(filename) as file\_object:

#读取所有数据到列表lines中

lines = file\_object.readlines()

pi\_string = ''

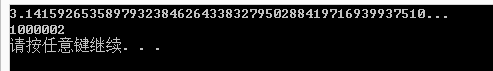
#将读取的数据合并为一行

for line in lines:

pi\_string += line.strip()

print(pi\_string[:52] + "...")

print(len(pi\_string))



Python没有限制处理文件的数据量，只要内存足够即可。

#### 在圆周率中查找生日

filename = r"Txt\pi\_million\_digits.txt" #包含有1百万位的圆周率文件

with open(filename) as file\_object:

#读取所有数据到列表lines中

lines = file\_object.readlines()

pi\_string = ''

#将读取的数据合并为一行

for line in lines:

pi\_string += line.strip()

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!")

## 写入文件

### 写入空文件

#### 写入空文件

#将要写入的目标文件路径赋值给变量

filename = r"txt\programming.txt"

#使用函数open()写入文件，函数第二个参数为枚举值。

#r:只读模式，默认值。

#w:写入模式，注意：该模式下将清空已存在的文件，重新写入新的数据

#a:附加模式

#r+:读取和写入模式

#如果open写入的文件不存在，将会自动创建

with open(filename,'w') as file\_object:

file\_object.write("I love programming.")

注意：Python只能将字符串写入文本文件，但需要写入数值时，需使用函数str()转换。

### 写入多行

使用转行符“\n”

#### 写入多行

filename = r"txt\programming.txt"

with open(filename,'w') as file\_object:

file\_object.write("I love programming.\n")

file\_object.write("I love creating new games.\n")

### 附加到文件

将写入的数据添加到文件末尾。

#### 附加到文件

filename = r"txt\programming.txt"

#在函数open()的第二个参数使用“a”，则方法write中的数据将添加到原有文件内容的末尾

with open(filename,'a') as file\_object:

file\_object.write("I love programming.\n")

file\_object.write("I love creating new games.\n")

## 异常

异常，是管理程序执行期间发生错误的特殊对象。当Python程序运行发生错误时，会创建一个异常对象。如果编写了应对相应异常的代码时，程序会继续执行；否则程序将停止，并显示一个traceback，报告异常情况。

使用try-except代码处理异常，当发生异常时，会执行该代码，程序将继续进行。

### 处理ZeroDivisionError异常

除数为0时会抛出该异常。

### 使用try-except代码块

#### 处理ZeroDivisionError异常

try:

print(5/0)

except ZeroDivisionError:

print("You can't divide by zero!")

### 使用异常避免崩溃

将可能发生异常的代码放入try代码块中。

### else代码块

try-except-else代码块：

1. Python将执行try代码块中的代码
2. 当发生异常时，执行except中的代码
3. 否则将执行else中的代码

else代码块中的代码执行与否，取决于try代码块是否成功执行。

#### 简单除法计算器

print("Give me two numbers, and I'll divide them.")

print("Enter 'q' to quit.")

while True:

first\_number = input("\nFirst number: ")

if first\_number == 'q':

break

second\_number = input("\nSecond number: ")

if second\_number == 'q':

break

try:

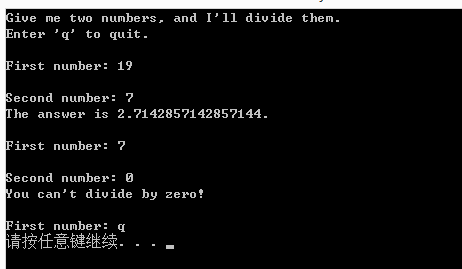
answer = int(first\_number) / int(second\_number)

except ZeroDivisionError:

print("You can't divide by zero!")

else:

print("The answer is " + str(answer) + '.')



### 处理FileNotFoundError异常

#### 处理FileNotFoundError异常

filename = 'alice.txt'

try:

with open(filename) as f\_obj:

contents = f\_obj.read()

except FileNotFoundError:

msg = "Sorry, the file " +filename + " does not exist."

print(msg)

### 分析文本

方法split()，以空格为分隔符将字符串分拆成多个部分，并储存到一个列表中。

#### 计算文本字数

filename = r'txt\alice.txt'

try:

with open(filename) as f\_obj:

contents = f\_obj.read()

except FileNotFoundError:

msg = "Sorry, the file " +filename + " does not exist."

print(msg)

else:

#计算文件包含的单词数

words = contents.split()

num\_words = len(words)

print("The file " + filename + " has about " + str(num\_words) + " words.")

### 模组化

将实现单一操作的代码块放入函数中。便于多次使用。

#### 模组化

def count\_words(filename):

"""计算文件包含的字符数"""

try:

with open(filename) as f\_obj:

contents = f\_obj.read()

except FileNotFoundError:

msg = "Sorry, the file " +filename + " does not exist."

print(msg)

else:

#计算文件包含的单词数

words = contents.split()

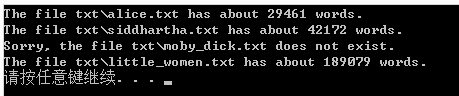
num\_words = len(words)

print("The file " + filename + " has about " + str(num\_words) + " words.")

filenames = ['alice.txt','siddhartha.txt','moby\_dick.txt','little\_women.txt']

for filename in filenames:

count\_words('txt\\' + filename)



### pass语句

如需要在发生异常时不进行任何操作，可以使用pass语句。

如：

try:

with open(filename) as f\_obj:

contents = f\_obj.read()

except FileNotFoundError:

pass

else:

#计算文件包含的单词数

words = contents.split()

num\_words = len(words)

print("The file " + filename + " has about " + str(num\_words) + " words.")

pass语句充当一个占位符，当前程序在该处没有任何操作，但以后可能需要在此处添加操作。

## 存储数据

使用模块json来存储数据。

### 存储和读取json数据

函数json.dump()：存储数据。有两个参数，要存储的数据和文件对象。

函数json.load()：读取数据到内存。参数为文件对象。

#### 存储和读取json数据

import json

numbers = [2,3,5,7,11,13]

filename = r'json\numbers.json'

#存储数据

with open(filename,'w') as f\_obj:

json.dump(numbers,f\_obj)

#读取数据

with open(filename) as f\_obj:

num\_list = json.load(f\_obj)

print(numbers)

### 保存和读取用户生成的数据

#### 存储和读取用户名

存储用户名程序：

import json

username = input("What is your name?")

filename = r'json\username.json'

with open(filename,'w') as f\_obj:

#将数据存储到文件中

json.dump(username,f\_obj)

print("We'll remember you when you come back, " + username + '!')

读取用户名程序：

import json

filename = r'json\username.json'

with open(filename) as f\_obj:

#读取json数据

username = json.load(f\_obj)

print("Welcome back, " + username + "!")



#### 存储和读取用户名（程序合并）

import json

def get\_stored\_username():

""" 获取用户名，如果存在则返回用户名，否则返回None """

filename = r'json\username.json'

try:

with open(filename) as f\_obj:

username = json.load(f\_obj)

except FileNotFoundError:

return None

else:

return username

def get\_new\_username():

""" 输入新用户名 """

username = input("What is your name?")

filename = r'json\username.json'

with open(filename,'w') as f\_obj:

json.dump(username,f\_obj)

return username

def greet\_user():

""" 问候用户 """

username = get\_stored\_username()

if username: #判断是否为空值

print("Welcome back, " + username + "!")

else:

username = get\_new\_username()

print("We'll remember you when you come back, " + username + '!')

greet\_user()

# 测试代码

## 测试函数

#### 测试函数

创建放置函数的文件name\_function.py：

def get\_formatted\_name(first,last):

""" 生成全名 """

full\_name = first + ' ' + last

return full\_name.title()

创建主程序names.py:

from name\_function import get\_formatted\_name

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

while True:

first = input("\nPlease give me a first name: ")

if first == 'q':

break

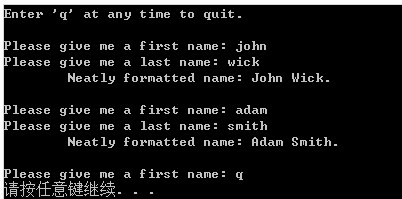
last = input("Please give me a last name: ")

if last == 'q':

break

formatted\_name = get\_formatted\_name(first,last)

print("\tNeatly formatted name: " + formatted\_name + '.')



### 单元测试和测试用例

使用标准库中的模块unittest测试代码。

测试用例，是一组单元测试，用于核实函数在各种情况下行为都符合要求。

### 测试

先导入模块unittest和要测试的函数。

再创建一个继承unittest.TestCase的类。

然后编写系列方法对函数行为不同方面进行测试。

#### 测试函数

import unittest #导入unittest模块

from name\_function import get\_formatted\_name #导入待测试的函数

class NamesTestCase(unittest.TestCase): #测试类必须继承自unittest.TestCase

""" 测试name\_function.py """

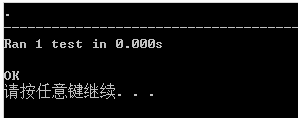
def test\_first\_last\_name(self): #定义测试方法，以test开头的方法都会自动运行

""" 测试是否正确Janis Joplin """

formatted\_name = get\_formatted\_name('janis','joplin')

self.assertEqual(formatted\_name,'Janis Joplin') #断言方法，用来核实得到的结果是否与期望的一致。

unittest.main()



### 不能通过的测试

#### 不能通过的测试

修改name\_function.py，待测试的函数为：

def get\_formatted\_name(first,middle,last):

""" 生成全名 """

full\_name = first + ' ' + middle + ' ' + last

return full\_name.title()

测试用例为test\_name\_function.py：

import unittest #导入unittest模块

from name\_function import get\_formatted\_name #导入待测试的函数

class NamesTestCase(unittest.TestCase): #测试类必须继承自unittest.TestCase

""" 测试name\_function.py """

def test\_first\_last\_name(self): #定义测试方法，以test开头的方法都会自动运行

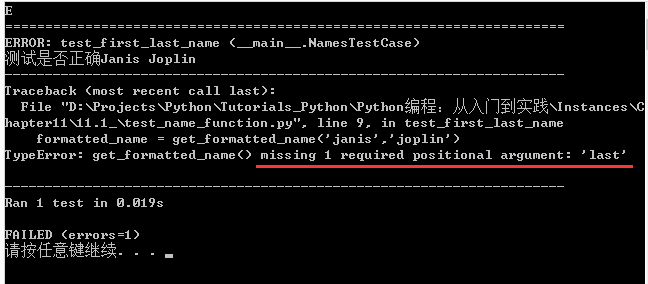
""" 测试是否正确Janis Joplin """

formatted\_name = get\_formatted\_name('janis','joplin')

self.assertEqual(formatted\_name,'Janis Joplin') #断言方法，用来核实得到的结果是否与期望的一致。

unittest.main()

测试结果：



处理：

不修改测试，应修复导致测试不通过的代码。

修改代码：

def get\_formatted\_name(first,last,middle=''):

""" 生成全名 """

if middle:

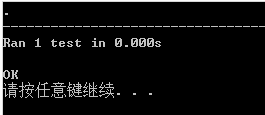
full\_name = first + ' ' + middle + ' ' + last

else:

full\_name = first + ' ' + last

return full\_name.title()

测试结果：



然后添加新测试，测试代码能否应对有中间名的情况：

import unittest #导入unittest模块

from name\_function import get\_formatted\_name #导入待测试的函数

class NamesTestCase(unittest.TestCase): #测试类必须继承自unittest.TestCase

""" 测试name\_function.py """

def test\_first\_last\_name(self): #定义测试方法，以test开头的方法都会自动运行

""" 测试是否正确Janis Joplin """

formatted\_name = get\_formatted\_name('janis','joplin')

self.assertEqual(formatted\_name,'Janis Joplin') #断言方法，用来核实得到的结果是否与期望的一致。

def test\_first\_last\_middle\_name(self): #添加新测试验证能应对有中间名的情况

""" 测试能否处理wolf amadeus mozart """

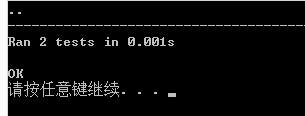
formatted\_name = get\_formatted\_name(

'wolf','mozart','amadeus')

self.assertEqual(formatted\_name,'Wolf Amadeus Mozart')

unittest.main()

测试结果：



## 测试类

### 断言方法

assertEqual(a,b) 核实a==b

assertNotEqual(a,b) 核实a!=b

assertTrue(x) 核x为True

assertFalse(x) 核实x为False

assertln(item,list) 核实item在list中

assertNotIn(item,list) 核实item不在Isit中

### 待测试的类

#### 待测试的类

待测试类survey.py：

class AnonymousSurvey():

""" 收集匿名调查问卷的答案 """

def \_\_init\_\_(self, question):

""" 定义属性question和response """

self.question = question

self.responses = []

def show\_question(self):

""" 显示调查问卷 """

print(self.question)

def store\_response(self,new\_response):

""" 存储调查答卷 """

self.responses.append(new\_response)

def show\_results(self):

""" 显示所有答卷 """

print("Survey results: ")

for response in self.responses:

print('- ' + response)

主程序：

from survey import AnonymousSurvey

#定义问题，创建实例

question = "What language did you first learn to speak?"

my\_survey = AnonymousSurvey(question)

#显示问题并存储答案

my\_survey.show\_question()

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

while True:

response = input("Language: ")

if response == 'q':

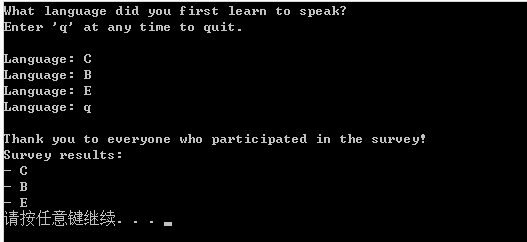
break

my\_survey.store\_response(response)

#显示调查结果

print("\nThank you to everyone who participated in the survey!")

my\_survey.show\_results()



### 测试AnonymousSurvey类

#### 测试类

测试代码：

import unittest

from survey import AnonymousSurvey

class TestAnonymousSurvey(unittest.TestCase):

""" 对Anonymous类测试 """

def test\_store\_single\_response(self):

""" 测试单个答案 """

question = "What language did you first learn to speak?"

my\_survey = AnonymousSurvey(question)

my\_survey.store\_response('English')

self.assertIn('English',my\_survey.responses)

def test\_store\_three\_responses(self):

""" 测试多个答案 """

question = "What language did you first learn to speak?"

my\_survey = AnonymousSurvey(question)

responses = ['English','Spanish','Madarin']

for response in responses:

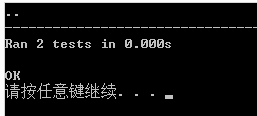
my\_survey.store\_response(response)

for response in responses:

self.assertIn(response,my\_survey.responses)

unittest.main()

结果：



### 方法setUp()

可以在方法setUp()中创建待测试类的实例，然后在每个测试方法中调用，而无需再每个测试方法中重复创建。

#### 方法setUp()

import unittest

from survey import AnonymousSurvey

class TestAnonymousSurvey(unittest.TestCase):

""" 对Anonymous类测试 """

def setUp(self):

"""

定义或创建通用变量或实例，供整个测试使用

"""

question = "What language did you first learn to speak?"

self.my\_survey = AnonymousSurvey(question)

self.responses = ['English','Spanish','Mandarin']

def test\_store\_single\_response(self):

""" 测试单个答案 """

self.my\_survey.store\_response(self.responses[0])

self.assertIn(self.responses[0],self.my\_survey.responses)

def test\_store\_three\_responses(self):

""" 测试多个答案 """

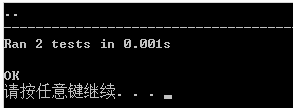
for response in self.responses:

self.my\_survey.store\_response(response)

for response in self.responses:

self.assertIn(response,self.my\_survey.responses)

unittest.main()



注意：每完成一个单元测试，Python都打印一个字符：通过测试的打印一个句点；测试引发错误的打印一个E；测试断言失败的打印一个F

# 绘制简单的折线图

### matplotlib

## 简单的折线图

#### 简单折线图

#导入模块并指定别名

import matplotlib.pyplot as plt

#定义列表

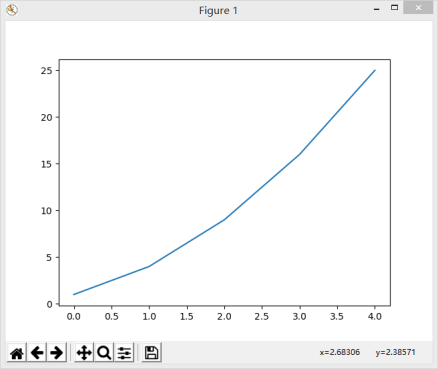
squares = [1,4,9,16,25]

#将列表传递给函数

plt.plot(squares)

#打开matplotlib查看器绘制图形

plt.show()



### 修改标签文字和线条粗细

#### 折线图（修改标签文字和线条粗细）

#导入模块并指定别名

import matplotlib.pyplot as plt

#定义列表

squares = [1,4,9,16,25]

#第二个参数指定线条粗细

plt.plot(squares,linewidth=5)

#设置图标标题，并给坐标轴加上标签

plt.title('Square Numbers',fontsize=24)

plt.xlabel('value',fontsize=14)

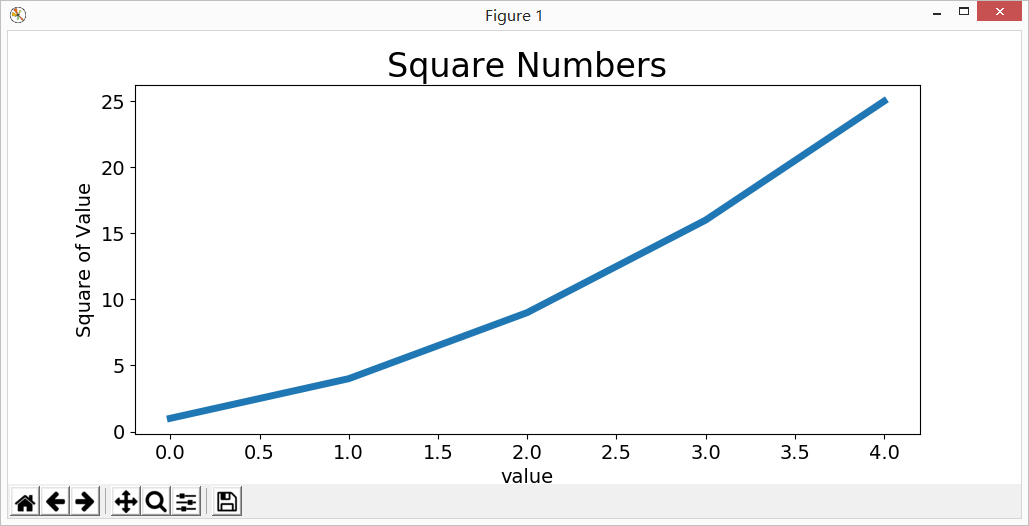
plt.ylabel('Square of Value',fontsize=14)

#设置刻度标记的大小

plt.tick\_params(axis='both',labelsize=14)

#打开matplotlib查看器绘制图形

plt.show()



### 校正图形

#### plot()函数的输入值和输出值

#导入模块并指定别名

import matplotlib.pyplot as plt

#定义列表

input\_value = [1,2,3,4,5]

squares = [1,4,9,16,25]

plt.plot(input\_value,squares,linewidth=5)

#设置图标标题，并给坐标轴加上标签

plt.title('Square Numbers',fontsize=24)

plt.xlabel('value',fontsize=14)

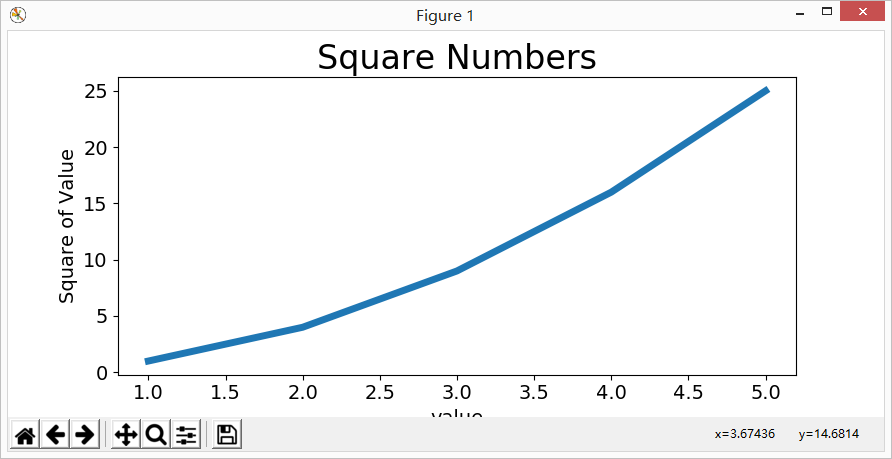
plt.ylabel('Square of Value',fontsize=14)

#设置刻度标记的大小

plt.tick\_params(axis='both',labelsize=14)

#打开matplotlib查看器绘制图形

plt.show()



### 绘制散点图（scatter()）

#### 绘制一个坐标点（函数scatter()）

#导入模块并指定别名

import matplotlib.pyplot as plt

plt.scatter(2,4,s=200)

#设置图表标题并给坐标轴加上标签

plt.title("Square Numbers",fontsize=24)

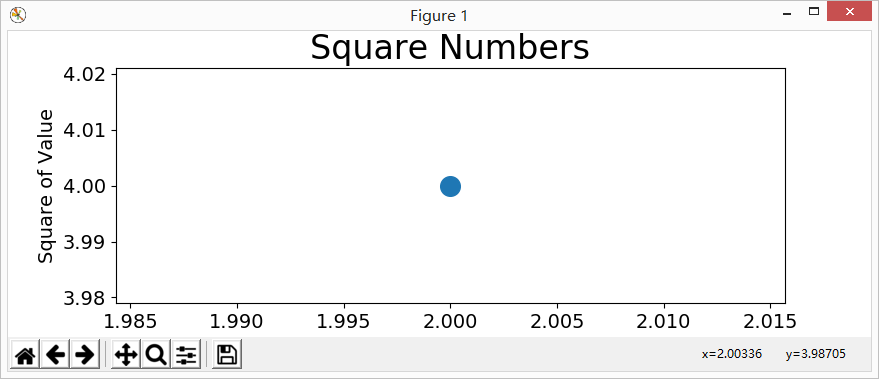
plt.xlabel("Value",fontsize=14)

plt.ylabel('Square of Value',fontsize=14)

#设置刻度标记的大小

plt.tick\_params(axis='both',which='major',labelsize=14)

plt.show()



### 绘制一些列的点

#### 绘制一些列的点（函数scatter()）

#导入模块并指定别名

import matplotlib.pyplot as plt

x\_values = [1,2,3,4,5]

y\_values = [1,4,9,16,25]

plt.scatter(x\_values,y\_values,s=50)

#设置图表标题并给坐标轴加上标签

plt.title("Square Numbers",fontsize=24)

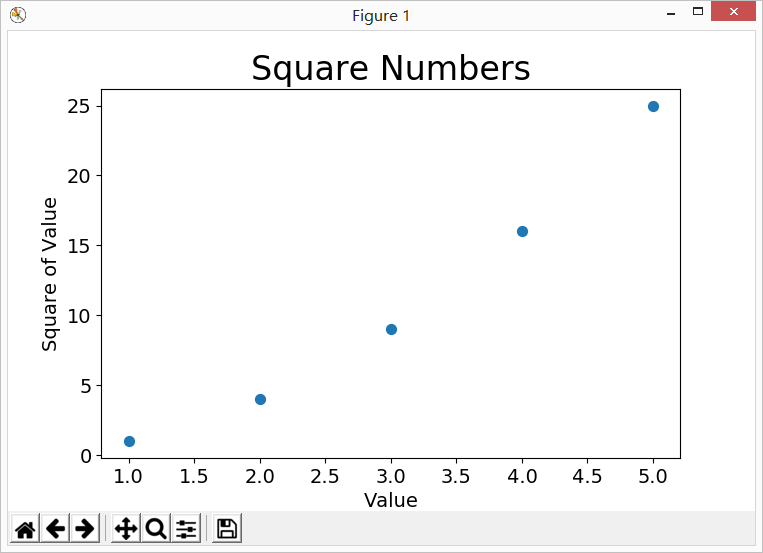
plt.xlabel("Value",fontsize=14)

plt.ylabel('Square of Value',fontsize=14)

#设置刻度标记的大小

plt.tick\_params(axis='both',which='major',labelsize=14)

plt.show()



### 自动计算数据

#### 自动计算数据

#导入模块并指定别名

import matplotlib.pyplot as plt

x\_values = list(range(1,1001)) #定义x轴坐标列表

y\_values = [x\*\*2 for x in x\_values] #遍历x值计算出y轴坐标列表

plt.scatter(x\_values,y\_values,s=10) #将x、y轴坐标列表导入函数

#设置图表标题并给坐标轴加上标签

plt.title("Square Numbers",fontsize=24)

plt.xlabel("Value",fontsize=14)

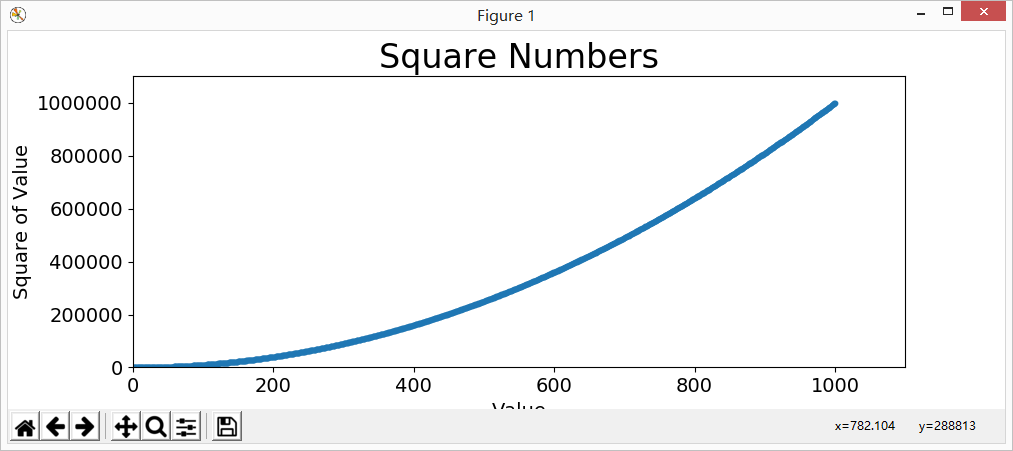
plt.ylabel('Square of Value',fontsize=14)

#设置刻度标记的大小

plt.tick\_params(axis='both',which='major',labelsize=14)

plt.axis([0,1100,0,1100000])

plt.show()



### 删除数据点的轮廓

使用函数scatter()的edgecolor参数设置为none

plt.scatter(x\_values,y\_values,edgecolors='none',s=10) #消除轮廓

### 自定义颜色

设置函数scatter()的参数c。

plt.scatter(x\_values,y\_values,edgecolors='none',c='red',s=10) #定义点颜色为红色

plt.scatter(x\_values,y\_values,edgecolors='none',c=(0,0,0.8),s=10) #定义点颜色为红色,使用RGB值，淡蓝色

### 使用颜色映射

颜色映射（colormap）是一些列颜色，从开始颜色渐变到结束颜色。

#### 颜色映射

#导入模块并指定别名

import matplotlib.pyplot as plt

x\_values = list(range(1,1001)) #定义x轴坐标列表

y\_values = [x\*\*2 for x in x\_values] #遍历x值计算出y轴坐标列表

plt.scatter(x\_values,y\_values,c=y\_values,cmap=plt.cm.Blues,s=10) #设置颜色映射

#设置图表标题并给坐标轴加上标签

plt.title("Square Numbers",fontsize=24)

plt.xlabel("Value",fontsize=14)

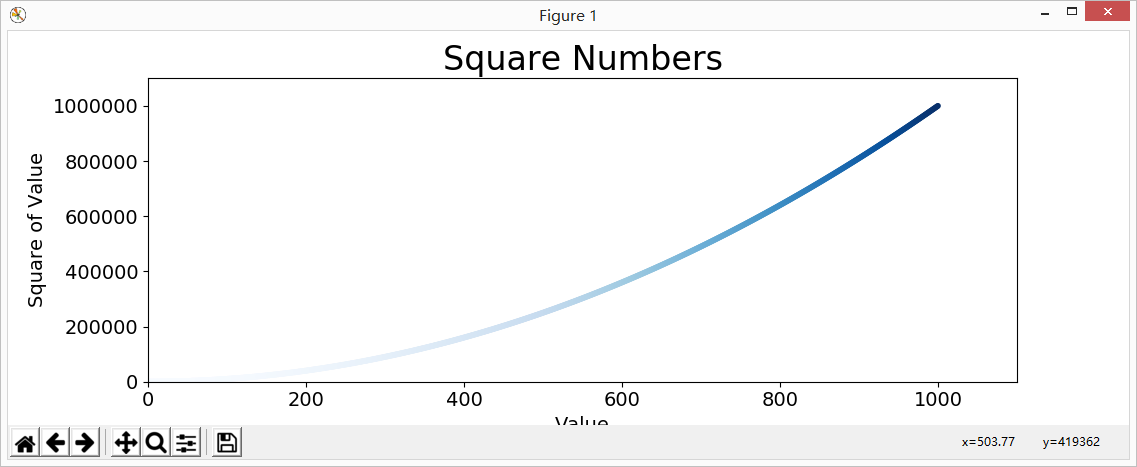
plt.ylabel('Square of Value',fontsize=14)

#设置刻度标记的大小

plt.tick\_params(axis='both',which='major',labelsize=14)

plt.axis([0,1100,0,1100000])

plt.show()



### 自动保存图表

方法savefig()。

#### 自动保存图表

#导入模块并指定别名

import matplotlib.pyplot as plt

x\_values = list(range(1,1001)) #定义x轴坐标列表

y\_values = [x\*\*2 for x in x\_values] #遍历x值计算出y轴坐标列表

plt.scatter(x\_values,y\_values,c=y\_values,cmap=plt.cm.Blues,s=10) #设置颜色映射

#设置图表标题并给坐标轴加上标签

plt.title("Square Numbers",fontsize=24)

plt.xlabel("Value",fontsize=14)

plt.ylabel('Square of Value',fontsize=14)

#设置刻度标记的大小

plt.tick\_params(axis='both',which='major',labelsize=14)

plt.axis([0,1100,0,1100000])

#保存图表，第一个实参为文件名，文件将储存在程序.py文件目录中

#第二实参为裁剪图表周围空白区域，如要保留则忽略该参数

plt.savefig('squares\_plot.png',bbox\_inches='tight')

注意，上例中如果目标文件夹已经存在一个同名的文件，将会覆盖以前的文件。

## 随机漫步

### 创建RandomWalk()类

#### 创建RandomWalk()类

from random import choice

class RandomWalk():

""" 生成随机漫步数据的类 """

def \_\_init\_\_(self,num\_points=5000):

""" 初始化随机漫步的属性 """

self.num\_points = num\_points

#所有随机漫步都始于(0,0)

self.x\_value = [0]

self.y\_value = [0]

def fill\_walk(self):

""" 计算随机漫步的所有点 """

while len(self.x\_value) < self.num\_points:

#决定前进方向以及沿这个方向前进的距离

x\_direction = choice([1,-1])

x\_distance = choice([0,1,2,3,4])

x\_step = x\_direction \* x\_distance

y\_direction = choice([1,-1])

y\_distance = choice([0,1,2,3,4])

y\_step = y\_direction \* y\_distance

#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

next\_x = self.x\_value[-1] + x\_step

next\_y = self.y\_value[-1] + y\_step

self.x\_value.append(next\_x)

self.y\_value.append(next\_y)

### 绘制随机漫步图

#### 绘制随机漫步图

import matplotlib.pyplot as plt

from random\_walk import RandomWalk

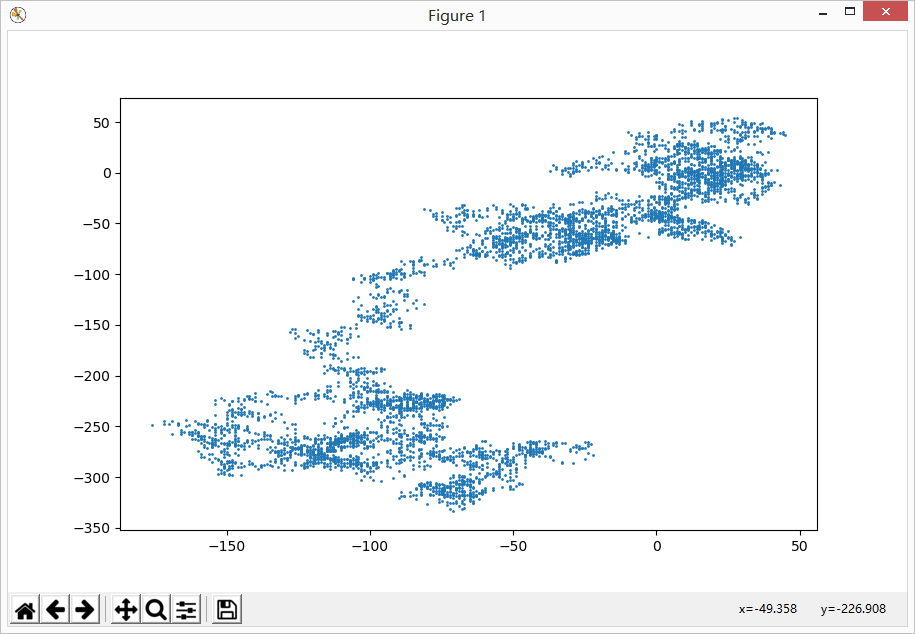
#创建实例并绘制所有点

rw = RandomWalk()

rw.fill\_walk()

plt.scatter(rw.x\_value,rw.y\_value,s=1)

plt.show()



### 多次随机漫步

#### 多次随机漫步

import matplotlib.pyplot as plt

from random\_walk import RandomWalk

#只要程序处于活动状态，就不断模拟随机漫步

while True:

#创建实例并绘制所有点

rw = RandomWalk()

rw.fill\_walk()

plt.scatter(rw.x\_value,rw.y\_value,s=1)

plt.show()

keep\_running = input("Make another walk?(y/n):")

if keep\_running == 'n':

break

### 设置随机漫步图的样式

### 给点着色

#### 给点着色

import matplotlib.pyplot as plt

from random\_walk import RandomWalk

#只要程序处于活动状态，就不断模拟随机漫步

while True:

#创建实例并绘制所有点

rw = RandomWalk()

rw.fill\_walk()

point\_numbers = list(range(rw.num\_points))

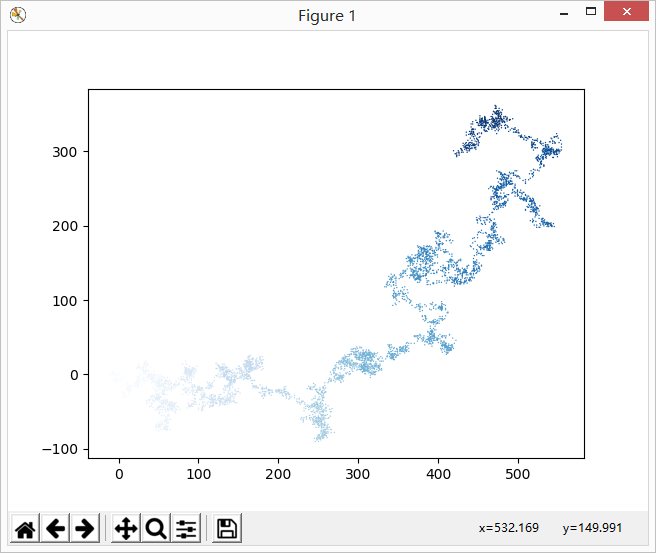
plt.scatter(rw.x\_value,rw.y\_value,c=point\_numbers,cmap=plt.cm.Blues,edgecolors="none",s=1)

plt.show()

keep\_running = input("Make another walk?(y/n):")

if keep\_running == 'n':

break



### 重新绘制起点和终点

#### 重新绘制起点和终点

import matplotlib.pyplot as plt

from random\_walk import RandomWalk

#只要程序处于活动状态，就不断模拟随机漫步

while True:

#创建实例并绘制所有点

rw = RandomWalk()

rw.fill\_walk()

point\_numbers = list(range(rw.num\_points))

plt.scatter(rw.x\_value,rw.y\_value,c=point\_numbers,cmap=plt.cm.Blues,edgecolors="none",s=5)

#突出起点和终点

plt.scatter(0,0,c='green',edgecolors='none',s=10)

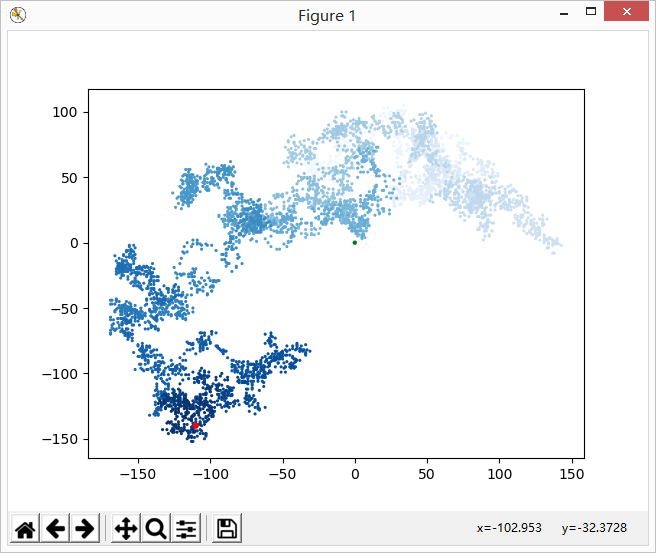
plt.scatter(rw.x\_value[-1],rw.y\_value[-1],c='red',edgecolors='none',s=20)

plt.show()

keep\_running = input("Make another walk?(y/n):")

if keep\_running == 'n':

break



### 隐藏坐标轴

#### 隐藏坐标轴

import matplotlib.pyplot as plt

from random\_walk import RandomWalk

#只要程序处于活动状态，就不断模拟随机漫步

while True:

#创建实例并绘制所有点

rw = RandomWalk()

rw.fill\_walk()

point\_numbers = list(range(rw.num\_points))

plt.scatter(rw.x\_value,rw.y\_value,c=point\_numbers,cmap=plt.cm.Blues,edgecolors="none",s=5)

#突出起点和终点

plt.scatter(0,0,c='green',edgecolors='none',s=10)

plt.scatter(rw.x\_value[-1],rw.y\_value[-1],c='red',edgecolors='none',s=20)

#隐藏坐标轴

plt.axes().get\_xaxis().set\_visible(False)

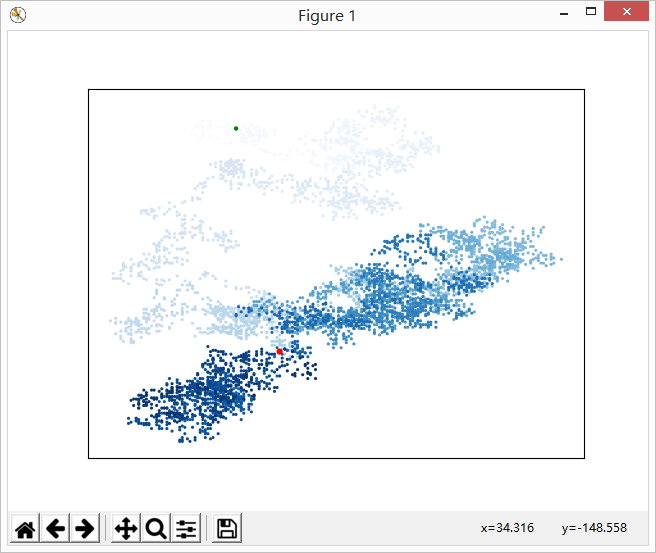
plt.axes().get\_yaxis().set\_visible(False)

plt.show()

keep\_running = input("Make another walk?(y/n):")

if keep\_running == 'n':

break



### 增加点数

#### 增加点数

import matplotlib.pyplot as plt

from random\_walk import RandomWalk

#只要程序处于活动状态，就不断模拟随机漫步

while True:

#创建实例并绘制所有点

rw = RandomWalk(50000)

rw.fill\_walk()

point\_numbers = list(range(rw.num\_points))

plt.scatter(rw.x\_value,rw.y\_value,c=point\_numbers,cmap=plt.cm.Blues,edgecolors="none",s=5)

#突出起点和终点

plt.scatter(0,0,c='green',edgecolors='none',s=10)

plt.scatter(rw.x\_value[-1],rw.y\_value[-1],c='red',edgecolors='none',s=20)

#隐藏坐标轴

plt.axes().get\_xaxis().set\_visible(False)

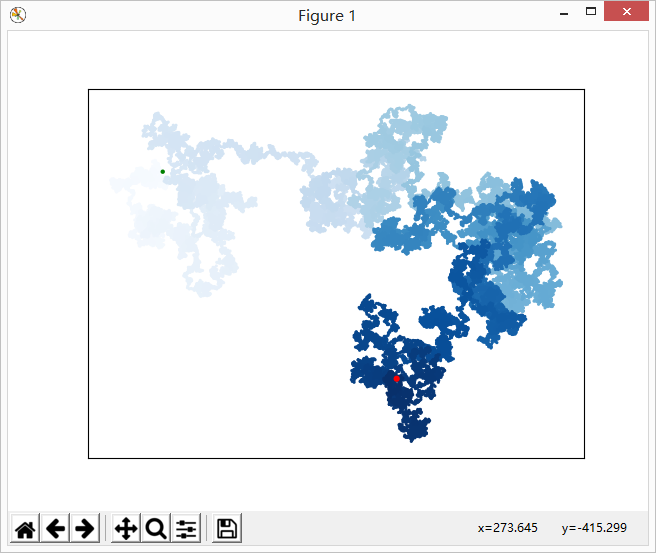
plt.axes().get\_yaxis().set\_visible(False)

plt.show()

keep\_running = input("Make another walk?(y/n):")

if keep\_running == 'n':

break



### 调整尺寸以适应屏幕

#### 调整尺寸以适应屏幕

import matplotlib.pyplot as plt

from random\_walk import RandomWalk

#只要程序处于活动状态，就不断模拟随机漫步

while True:

#创建实例并绘制所有点

rw = RandomWalk(50000)

rw.fill\_walk()

#设置绘图窗口的尺寸

plt.figure(figsize=(10,6))

point\_numbers = list(range(rw.num\_points))

plt.scatter(rw.x\_value,rw.y\_value,c=point\_numbers,cmap=plt.cm.Blues,edgecolors="none",s=5)

#突出起点和终点

plt.scatter(0,0,c='green',edgecolors='none',s=10)

plt.scatter(rw.x\_value[-1],rw.y\_value[-1],c='red',edgecolors='none',s=20)

#隐藏坐标轴

plt.axes().get\_xaxis().set\_visible(False)

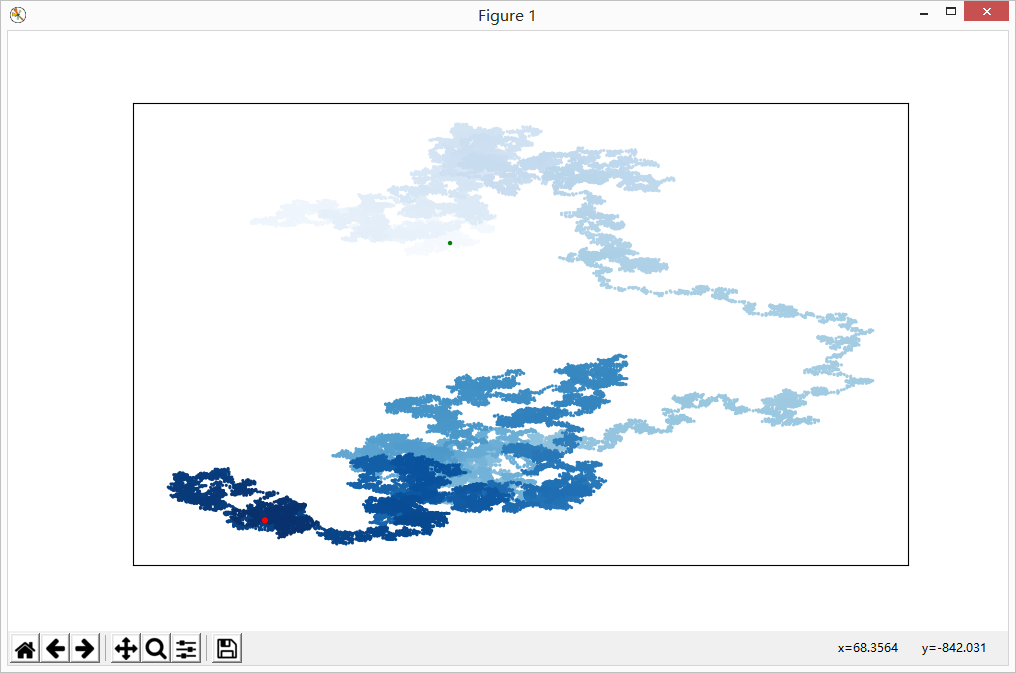
plt.axes().get\_yaxis().set\_visible(False)

plt.show()

keep\_running = input("Make another walk?(y/n):")

if keep\_running == 'n':

break



## 使用Pygal模拟掷骰子

#### 模拟掷骰子

骰子类：

from random import randint

class Die():

""" 表示一个骰子的类 """

def \_\_init\_\_(self,num\_sides=6):

""" 骰子默认为6面 """

self.num\_sides = num\_sides

def roll(self):

""" 返回一个位于1和骰子面数之间的随机值 """

return randint(1,self.num\_sides)

掷骰子：

from die import Die

die = Die()

#掷几次骰子，将结果存储在一个列表中

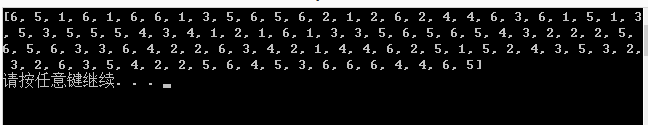
results = []

for roll\_num in range(100):

result = die.roll()

results.append(result)

print(results)



### 分析结果

#### 分析掷骰子结果

from die import Die

die = Die()

#掷几次骰子，将结果存储在一个列表中

results = []

for roll\_num in range(1000):

result = die.roll()

results.append(result)

#分析结果

frequencies = []

for value in range(1,die.num\_sides+1):

frequency = results.count(value) #计算出value值出现的次数

frequencies.append(frequency) #将结果记录到列表

print(frequencies)



### 绘制直方图

#### 绘制直方图

import pygal

from die import Die

die = Die()

#掷几次骰子，将结果存储在一个列表中

results = []

for roll\_num in range(1000):

result = die.roll()

results.append(result)

#分析结果

frequencies = []

for value in range(1,die.num\_sides+1):

frequency = results.count(value) #计算出value值出现的次数

frequencies.append(frequency) #将结果记录到列表

#对结果可视化

hist = pygal.Bar()

hist.title ="Results of rolling one D6 1000 times."

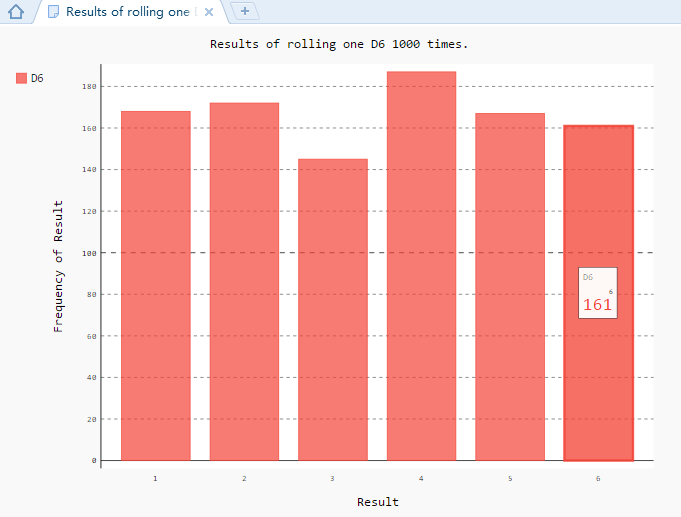
hist.x\_labels = ['1','2','3','4','5','6']

hist.x\_title = "Result"

hist.y\_title = "Frequency of Result"

hist.add('D6', frequencies)

hist.render\_to\_file('die\_visual.svg')



### 同时掷两个骰子

#### 同时掷两个骰子

import pygal

from die import Die

#创建两个骰子

die\_1 = Die()

die\_2 = Die()

#掷几次骰子，将结果存储在一个列表中

results = []

for roll\_num in range(1000):

result = die\_1.roll() + die\_2.roll()

results.append(result)

#分析结果

frequencies = []

max\_result = die\_1.num\_sides + die\_2.num\_sides

for value in range(1,max\_result+1):

frequency = results.count(value) #计算出value值出现的次数

frequencies.append(frequency) #将结果记录到列表

#对结果可视化

hist = pygal.Bar()

hist.title ="Results of rolling two D6 dice 1000 times."

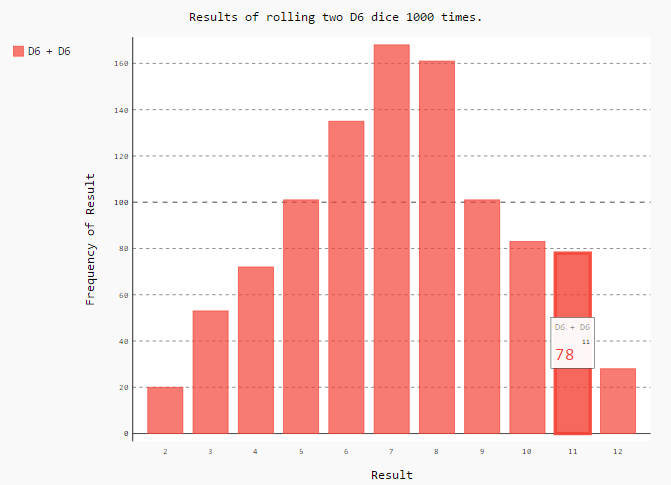
hist.x\_labels = ['2','3','4','5','6','7','8','9','10','11','12']

hist.x\_title = "Result"

hist.y\_title = "Frequency of Result"

hist.add('D6 + D6', frequencies)

hist.render\_to\_file('die\_visual.svg')



### 同时掷两个面数不同的骰子

#### 同时掷两个面数不同的骰子

import pygal

from die import Die

#创建1个D6的骰子

die\_1 = Die()

#创建1个D10的骰子

die\_2 = Die(10)

#掷几次骰子，将结果存储在一个列表中

results = []

for roll\_num in range(50000):

result = die\_1.roll() + die\_2.roll()

results.append(result)

#分析结果

frequencies = []

max\_result = die\_1.num\_sides + die\_2.num\_sides

for value in range(2,max\_result+1):

frequency = results.count(value) #计算出value值出现的次数

frequencies.append(frequency) #将结果记录到列表

#对结果可视化

hist = pygal.Bar()

hist.title ="Results of rolling two D6 dice 1000 times."

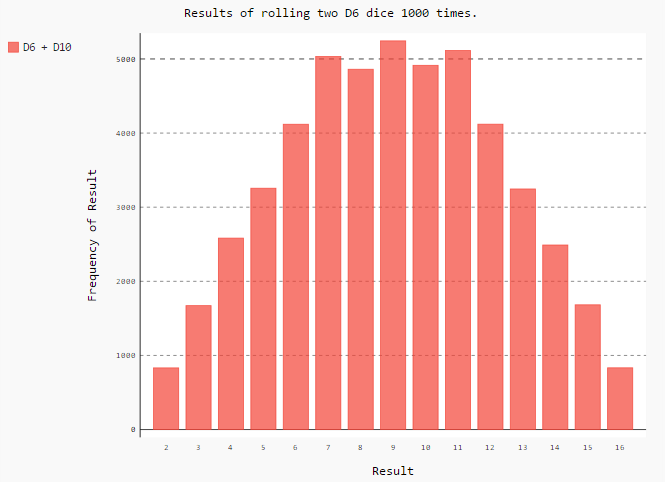
hist.x\_labels = list(range(2,17))

hist.x\_title = "Result"

hist.y\_title = "Frequency of Result"

hist.add('D6 + D10', frequencies)

hist.render\_to\_file('die\_visual.svg')



# 下载数据

## csv文件格式

以逗号分隔的一系列数据。

### 分析csv文件头

#### 分析csv文件头

import csv

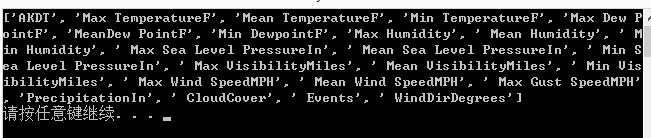
filename = r'Data\sitka\_weather\_07-2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()存储文件到变量

header\_row = next(reader) #调用函数next()读取文件第一行

print(header\_row)



### 打印文件头及其位置

#### 序列化列表（函数enumerate()）

import csv

filename = r'Data\sitka\_weather\_07-2014.csv' #文件路径

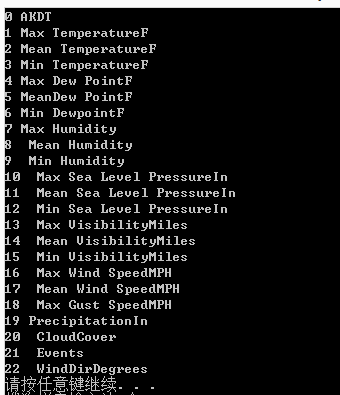
with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()存储文件到变量

header\_row = next(reader) #调用函数next()读取文件第一行

for index,column\_header in enumerate(header\_row): #函数enumerate()序列化列表

print(index,column\_header)



### 提取并读取数据

#### 提取并读取数据

import csv

filename = r'Data\sitka\_weather\_07-2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()读取文件为列表并存储到变量

header\_row = next(reader) #调用函数next()读取文件第一行

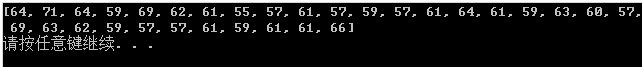
highs = []

for row in reader:

high = int(row[1]) #转换为整数类型

highs.append(high)

print(highs)



### 绘制气温图表

#### 绘制图形

import csv

from matplotlib import pyplot as plt

filename = r'Data\sitka\_weather\_07-2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()读取文件为列表并存储到变量

header\_row = next(reader) #调用函数next()读取文件第一行

highs = []

for row in reader:

high = int(row[1]) #转换为整数类型

highs.append(high)

#根据数据绘制图形

fig = plt.figure(dpi=128,figsize=(10,6))

plt.plot(highs,c='red') #将数据传递给plot函数

#设置图形的格式

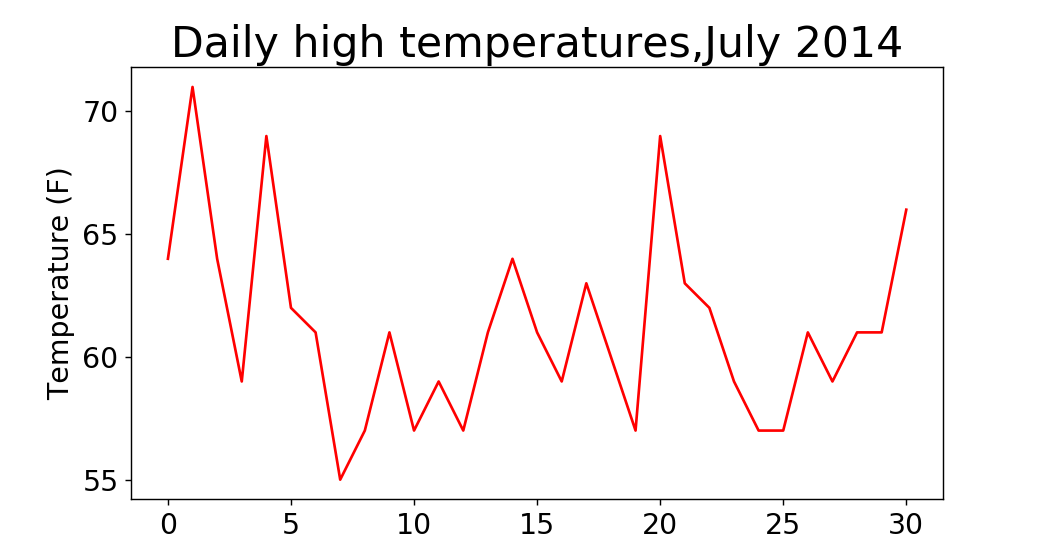
plt.title("Daily high temperatures,July 2014",fontsize=24)

plt.xlabel('',fontsize=16)

plt.ylabel("Temperature (F)",fontsize=16)

plt.tick\_params(axis='both',which='major',labelsize=16)

plt.show()



### 模块datetime

模块datetime的方法striptime()：可以将字符串转换为一个表示相应日期的对象。

第一个实参为表示日期的字符串。

第二个实参指示格式。

### 在图表中添加日期

#### 在图表中添加日期

import csv

from datetime import datetime

from matplotlib import pyplot as plt

filename = r'Data\sitka\_weather\_07-2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()读取文件为列表并存储到变量

header\_row = next(reader) #调用函数next()读取文件第一行

dates,highs = [],[] #创建空列表

for row in reader:

current\_date = datetime.strptime(row[0],"%Y-%m-%d") #使用方法striptime()转换为对象

dates.append(current\_date) #将数据填充列表

high = int(row[1]) #转换为整数类型

highs.append(high)

#根据数据绘制图形

fig = plt.figure(dpi=128,figsize=(10,6))

plt.plot(dates,highs,c='red') #将数据传递给plot函数

#设置图形的格式

plt.title("Daily high temperatures,July 2014",fontsize=24)

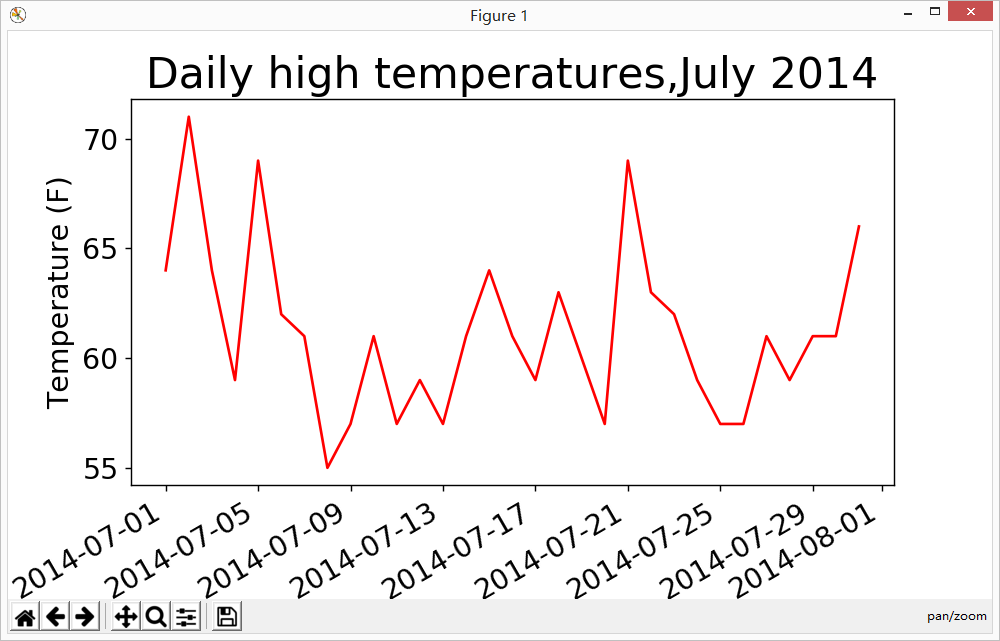
plt.xlabel('',fontsize=16)

fig.autofmt\_xdate() #绘制斜的日期标签，以免彼此重叠

plt.ylabel("Temperature (F)",fontsize=16)

plt.tick\_params(axis='both',which='major',labelsize=16)

plt.show()



### 涵盖更长的时间

#### 读取更长时间的数据

import csv

from datetime import datetime

from matplotlib import pyplot as plt

filename = r'Data\sitka\_weather\_2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()读取文件为列表并存储到变量

header\_row = next(reader) #调用函数next()读取文件第一行

dates,highs = [],[] #创建空列表

for row in reader:

current\_date = datetime.strptime(row[0],"%Y-%m-%d") #使用方法striptime()转换为对象

dates.append(current\_date) #将数据填充列表

high = int(row[1]) #转换为整数类型

highs.append(high)

#根据数据绘制图形

fig = plt.figure(dpi=128,figsize=(10,6))

plt.plot(dates,highs,c='red') #将数据传递给plot函数

#设置图形的格式

plt.title("Daily high temperatures - 2014",fontsize=18)

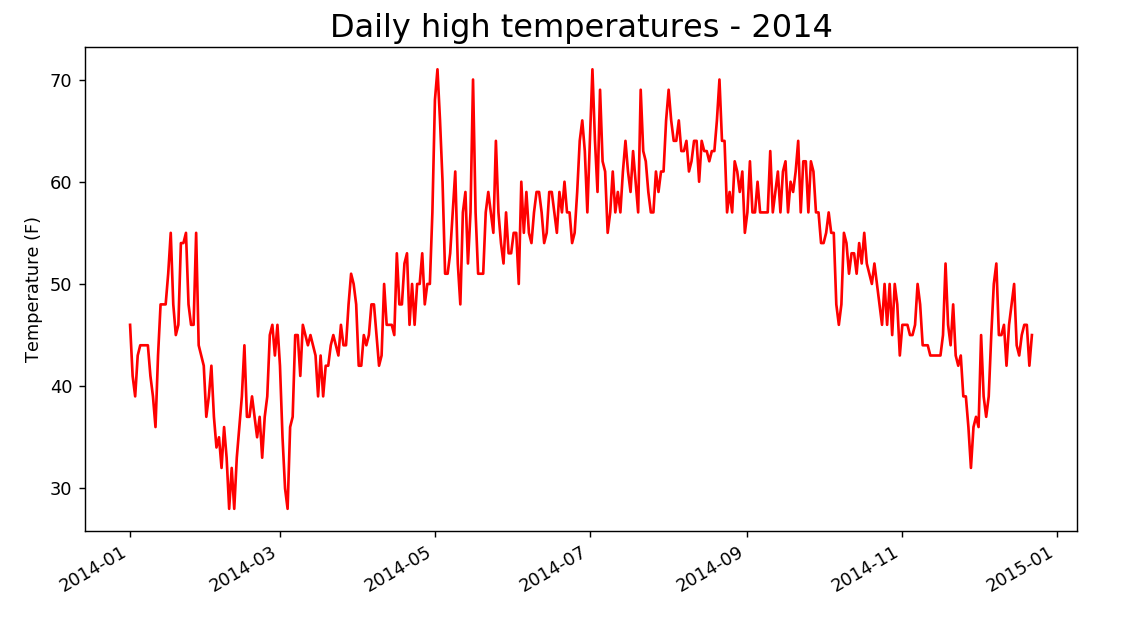
plt.xlabel('',fontsize=10)

fig.autofmt\_xdate() #绘制斜的日期标签，以免彼此重叠

plt.ylabel("Temperature (F)",fontsize=10)

plt.tick\_params(axis='both',which='major',labelsize=10)

plt.show()



### 绘制另一个数据系列

#### 绘制两个数据系列

import csv

from datetime import datetime

from matplotlib import pyplot as plt

filename = r'Data\sitka\_weather\_2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()读取文件为列表并存储到变量

header\_row = next(reader) #调用函数next()读取文件第一行

dates,highs,lows = [],[],[] #创建空列表

for row in reader:

current\_date = datetime.strptime(row[0],"%Y-%m-%d") #使用方法striptime()转换为对象

dates.append(current\_date) #将数据填充列表

high = int(row[1]) #转换为整数类型

highs.append(high)

low = int(row[3]) #转换为整数类型

lows.append(low)

#根据数据绘制图形

fig = plt.figure(dpi=128,figsize=(10,6))

plt.plot(dates,highs,c='red') #将数据传递给plot函数,绘制highs系列

plt.plot(dates,lows,c='blue') #将数据传递给plot函数,绘制lows系列

#设置图形的格式

plt.title("Daily high temperatures - 2014",fontsize=18)

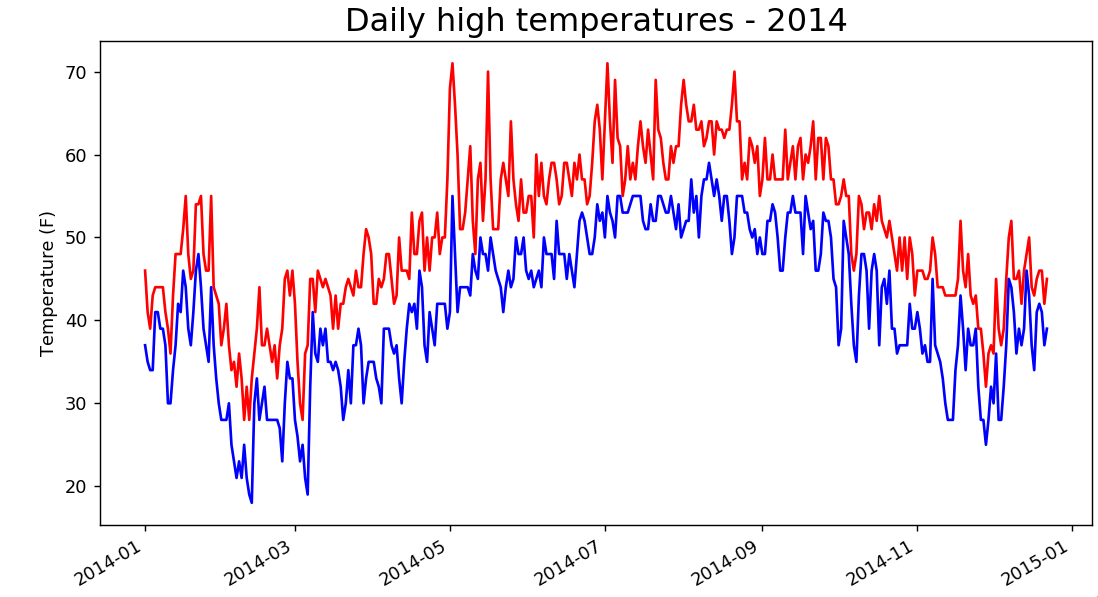
plt.xlabel('',fontsize=10)

fig.autofmt\_xdate() #绘制斜的日期标签，以免彼此重叠

plt.ylabel("Temperature (F)",fontsize=10)

plt.tick\_params(axis='both',which='major',labelsize=10)

plt.show()



### 给图表区域着色

使用方法fill\_between()，接受一个x值系列和两个y值系列，并填充两个y值系列之间的空间。

#### 给图表区域着色

import csv

from datetime import datetime

from matplotlib import pyplot as plt

filename = r'Data\sitka\_weather\_2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()读取文件为列表并存储到变量

header\_row = next(reader) #调用函数next()读取文件第一行

dates,highs,lows = [],[],[] #创建空列表

for row in reader:

current\_date = datetime.strptime(row[0],"%Y-%m-%d") #使用方法striptime()转换为对象

dates.append(current\_date) #将数据填充列表

high = int(row[1]) #转换为整数类型

highs.append(high)

low = int(row[3]) #转换为整数类型

lows.append(low)

#根据数据绘制图形

fig = plt.figure(dpi=128,figsize=(10,6))

plt.plot(dates,highs,c='red',alpha=0.5) #将数据传递给plot函数,绘制highs系列，设置透明值为0.5

plt.plot(dates,lows,c='blue',alpha=0.5) #将数据传递给plot函数,绘制lows系列，设置透明值为0.5

plt.fill\_between(dates,highs,lows,facecolor='blue',alpha=0.1) #填充区域

#设置图形的格式

plt.title("Daily high temperatures - 2014",fontsize=18)

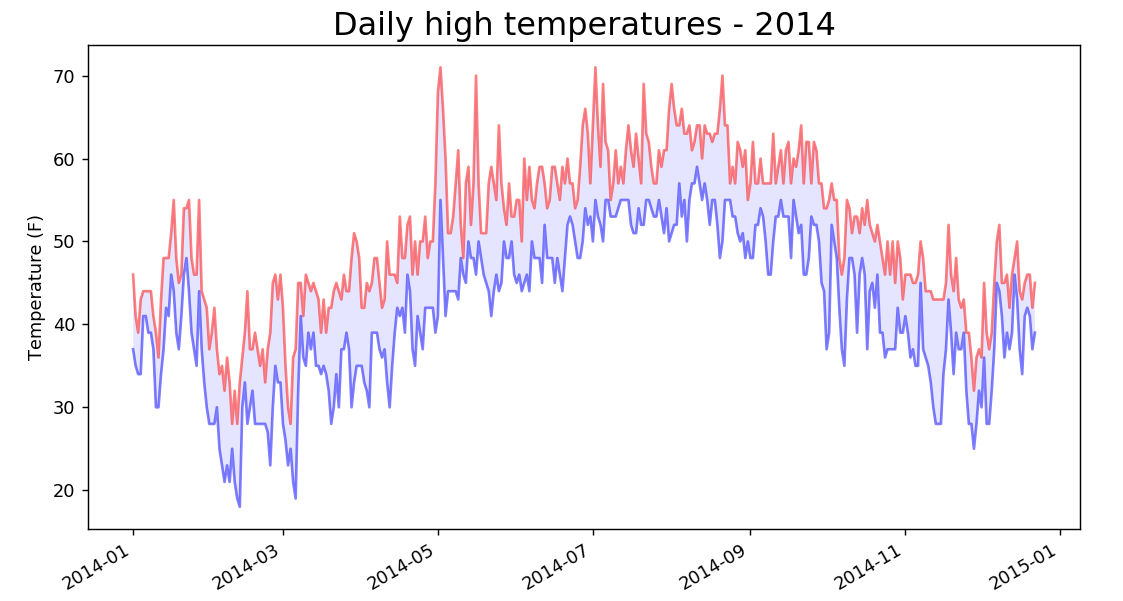
plt.xlabel('',fontsize=10)

fig.autofmt\_xdate() #绘制斜的日期标签，以免彼此重叠

plt.ylabel("Temperature (F)",fontsize=10)

plt.tick\_params(axis='both',which='major',labelsize=10)

plt.show()



### 错误检查

当数据集存在数据缺失时，可能会引起程序异常。

当读取CSV文件时，可以执行错误检查，对分析数据集可能出现的异常进行处理。

可以使用try-except-else代码块处理。视情况而使用continue来跳过一些数据，或使用remove()或del将已经提取的数据删除。

#### 错误检查

import csv

from datetime import datetime

from matplotlib import pyplot as plt

filename = r'Data\death\_valley\_2014.csv' #文件路径

with open(filename) as f: #打开文件

reader = csv.reader(f) #调用模块函数csv.reader()读取文件为列表并存储到变量

header\_row = next(reader) #调用函数next()读取文件第一行

dates,highs,lows = [],[],[] #创建空列表

for row in reader:

try:

current\_date = datetime.strptime(row[0],"%Y-%m-%d") #使用方法striptime()转换为对象

high = int(row[1]) #转换为整数类型

low = int(row[3]) #转换为整数类型

except ValueError: #异常处理

print(current\_date,'missing data')

else:

dates.append(current\_date) #将数据填充列表

highs.append(high)

lows.append(low)

#根据数据绘制图形

fig = plt.figure(dpi=128,figsize=(10,6))

plt.plot(dates,highs,c='red',alpha=0.5) #将数据传递给plot函数,绘制highs系列，设置透明值为0.5

plt.plot(dates,lows,c='blue',alpha=0.5) #将数据传递给plot函数,绘制lows系列，设置透明值为0.5

plt.fill\_between(dates,highs,lows,facecolor='blue',alpha=0.1) #填充区域

#设置图形的格式

plt.title("Daily high temperatures - 2014\nDeath Valley,CA",fontsize=18)

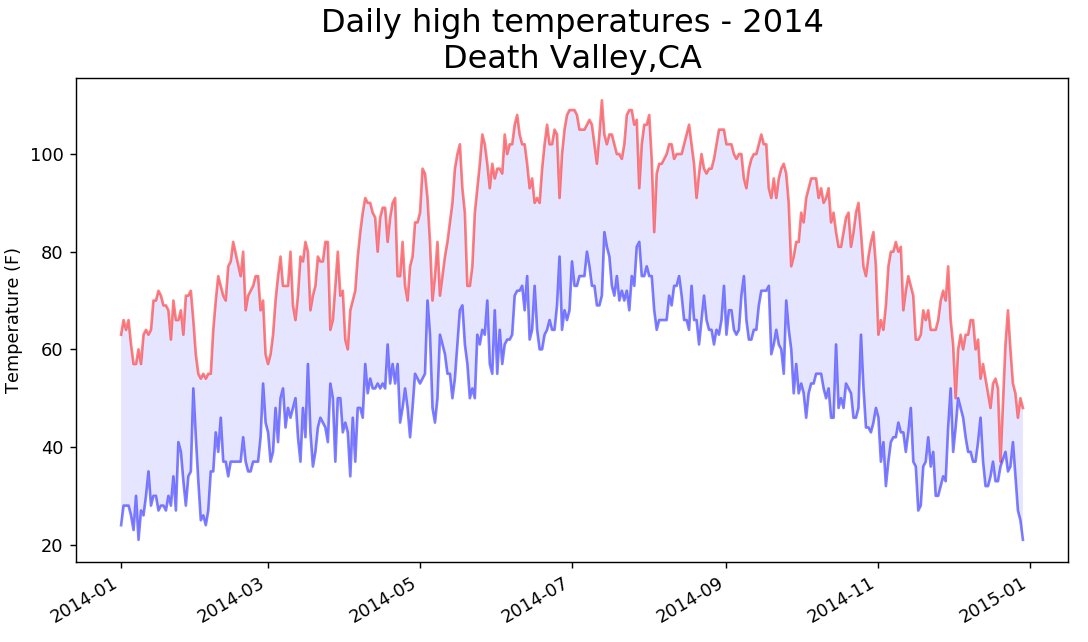
plt.xlabel('',fontsize=10)

fig.autofmt\_xdate() #绘制斜的日期标签，以免彼此重叠

plt.ylabel("Temperature (F)",fontsize=10)

plt.tick\_params(axis='both',which='major',labelsize=10)

plt.show()



## JSON格式

### 下载数据

#### 下载数据（函数urlopen()）

from \_\_future\_\_ import (absolute\_import, division, print\_function,

unicode\_literals)

from urllib.request import urlopen #导入下载模块

import json

json\_url = 'https://raw.githubusercontent.com/muxuezi/btc/master/btc\_close\_2017.json'

response = urlopen(json\_url) # 2

# 读取数据

req = response.read() #读取服务器响应

# 将数据写入文件

with open('btc\_close\_2017\_urllib.json', 'wb') as f: # 3

f.write(req) #将读取的数据写入文件

# 加载json格式

file\_urllib = json.loads(req.decode('utf8')) #方法load()将文件内容转换格式

print(file\_urllib)



#### 下载数据（模块requests）

import requests #需要另外安装该模块

json\_url = 'https://raw.githubusercontent.com/muxuezi/btc/master/btc\_close\_2017.json'

req = requests.get(json\_url) #方法get()向服务器发送请求并获取响应。

# 将数据写入文件

with open('btc\_close\_2017\_request.json', 'w') as f:

f.write(req.text) #将响应以字符串形式写入文件

file\_requests = req.json() #使用方法json将响应转换为Python列表

print(file\_requests)

### 提取相关数据

#### 提取json数据

import json

#将数据加载到列表中

filename = r'Data\btc\_close\_2017.json'

with open(filename) as f:

btc\_data = json.load(f) #将json文件存储到变量中

#打印每一天的信息

for btc\_dict in btc\_data:

date = btc\_dict['date']

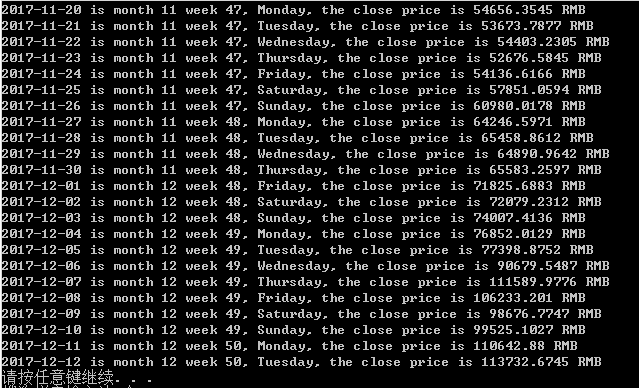
month = btc\_dict['month']

week = btc\_dict['week']

weekday = btc\_dict['weekday']

close = btc\_dict['close']

print("{} is month {} week {}, {}, the close price is {} RMB".format(date,month,week,weekday,close))



### 将字符串转换为数字值

含有小数点的字符串不能直接转换为整数值，需要先转换为浮点值，再转为整数值。

#### 将字符串转换为数字值

import json

#将数据加载到列表中

filename = r'Data\btc\_close\_2017.json'

with open(filename) as f:

btc\_data = json.load(f) #将json文件存储到变量中

#打印每一天的信息

for btc\_dict in btc\_data:

date = btc\_dict['date']

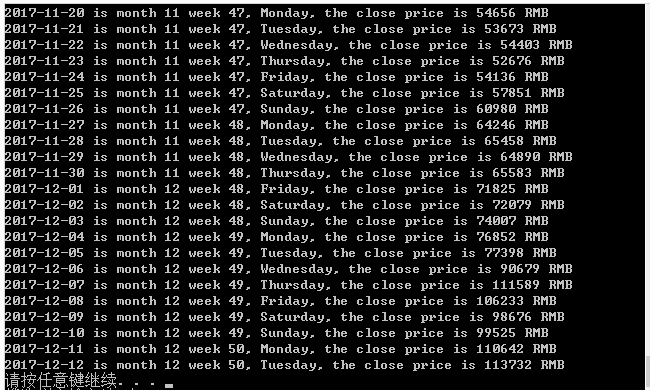
month = int(btc\_dict['month'])

week = int(btc\_dict['week'])

weekday = btc\_dict['weekday']

close =int(float(btc\_dict['close']))

print("{} is month {} week {}, {}, the close price is {} RMB".format(date,month,week,weekday,close))



### 绘制折线图

使用Pygal绘制。

#### 绘制折线图（Pygal）

import json

import pygal

#将数据加载到列表中

filename = r'Data\btc\_close\_2017.json'

with open(filename) as f:

btc\_data = json.load(f) #将json文件存储到变量中

#创建空列表

dates = []

months = []

weeks = []

weekdays = []

close = []

#每一天的信息

for btc\_dict in btc\_data:

dates.append(btc\_dict['date'])

months.append(int(btc\_dict['month']))

weeks.append(int(btc\_dict['week']))

weekdays.append(btc\_dict['weekday'])

close.append(int(float(btc\_dict['close'])))

#旋转日期标签；不显示所有x轴标签

line\_chart = pygal.Line(x\_label\_rotation=20,show\_minor\_x\_labels=False)

line\_chart.title = '收盘价'

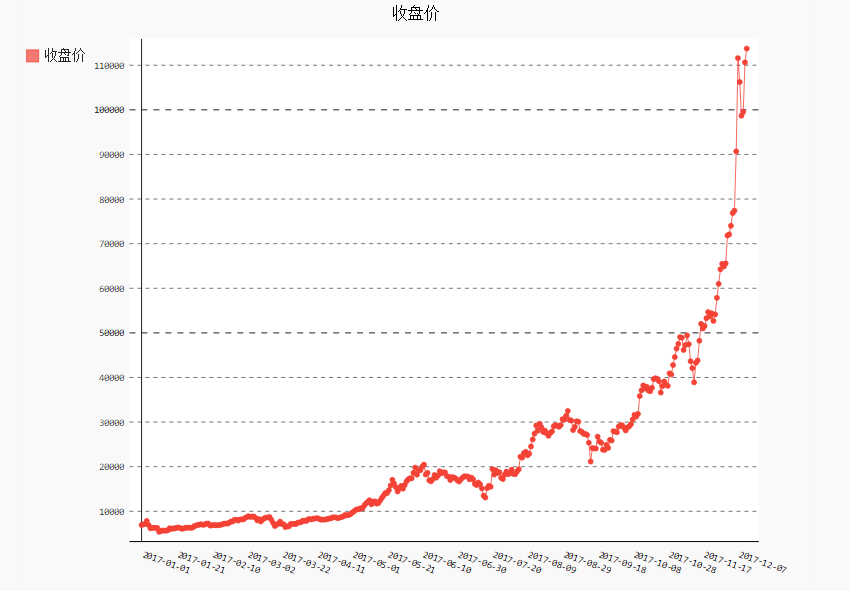
line\_chart.x\_labels = dates

N = 20 #x轴坐标每隔20天显示一次

line\_chart.x\_labels\_major = dates[::N]

line\_chart.add('收盘价',close)

line\_chart.render\_to\_file('收盘价折线图.svg') #保存图表



### 时间序列特征分析

时间序列分析：期望发现趋势、周期性和噪声，从而能描述事实、预测未来做出决策。

本例采用半对数变换。

#### 时间序列分析（半对数变换）

import json

import pygal

import math #导入数学模块

#将数据加载到列表中

filename = r'Data\btc\_close\_2017.json'

with open(filename) as f:

btc\_data = json.load(f) #将json文件存储到变量中

#创建空列表

dates = []

months = []

weeks = []

weekdays = []

close = []

#每一天的信息

for btc\_dict in btc\_data:

dates.append(btc\_dict['date'])

months.append(int(btc\_dict['month']))

weeks.append(int(btc\_dict['week']))

weekdays.append(btc\_dict['weekday'])

close.append(int(float(btc\_dict['close'])))

#旋转日期标签；不显示所有x轴标签

line\_chart = pygal.Line(x\_label\_rotation=20,show\_minor\_x\_labels=False)

line\_chart.title = '收盘价对数变换'

line\_chart.x\_labels = dates

N = 20 #x轴坐标每隔20天显示一次

line\_chart.x\_labels\_major = dates[::N]

close\_log = [math.log10(\_) for \_ in close] #使用以10为底的对数函数

line\_chart.add('log收盘价',close)

line\_chart.render\_to\_file('收盘价对数变换折线图.svg') #保存图表



### 均价

#### 收盘价均价

import json

import pygal

import math #导入数学模块

from itertools import groupby #导入模块itertools的函数groupby

def draw\_line(x\_data, y\_data, title, y\_legend):

xy\_map = []

for x, y in groupby(sorted(zip(x\_data, y\_data)), key=lambda \_: \_[0]): # 2

y\_list = [v for \_, v in y]

xy\_map.append([x, sum(y\_list) / len(y\_list)]) # 3

x\_unique, y\_mean = [\*zip(\*xy\_map)] # 4

line\_chart = pygal.Line()

line\_chart.title = title

line\_chart.x\_labels = x\_unique

line\_chart.add(y\_legend, y\_mean)

line\_chart.render\_to\_file(title + '.svg')

return line\_chart

#将数据加载到列表中

filename = r'Data\btc\_close\_2017.json'

with open(filename) as f:

btc\_data = json.load(f) #将json文件存储到变量中

#创建空列表

dates = []

months = []

weeks = []

weekdays = []

close = []

#每一天的信息

for btc\_dict in btc\_data:

dates.append(btc\_dict['date'])

months.append(int(btc\_dict['month']))

weeks.append(int(btc\_dict['week']))

weekdays.append(btc\_dict['weekday'])

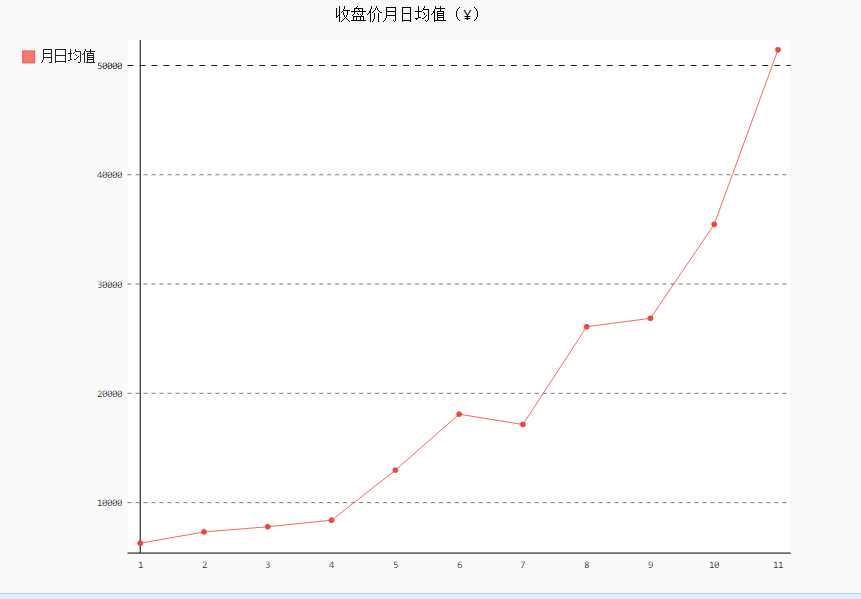
close.append(int(float(btc\_dict['close'])))

idx\_month = dates.index('2017-12-01')

line\_chart\_month = draw\_line(

months[:idx\_month], close[:idx\_month], '收盘价月日均值（¥）', '月日均值')

line\_chart\_month



#### 绘制周日均价

import json

import pygal

import math #导入数学模块

from itertools import groupby #导入模块itertools的函数groupby

def draw\_line(x\_data, y\_data, title, y\_legend):

xy\_map = []

for x, y in groupby(sorted(zip(x\_data, y\_data)), key=lambda \_: \_[0]): # 2

y\_list = [v for \_, v in y]

xy\_map.append([x, sum(y\_list) / len(y\_list)]) # 3

x\_unique, y\_mean = [\*zip(\*xy\_map)] # 4

line\_chart = pygal.Line()

line\_chart.title = title

line\_chart.x\_labels = x\_unique

line\_chart.add(y\_legend, y\_mean)

line\_chart.render\_to\_file(title + '.svg')

return line\_chart

#将数据加载到列表中

filename = r'Data\btc\_close\_2017.json'

with open(filename) as f:

btc\_data = json.load(f) #将json文件存储到变量中

#创建空列表

dates = []

months = []

weeks = []

weekdays = []

close = []

#每一天的信息

for btc\_dict in btc\_data:

dates.append(btc\_dict['date'])

months.append(int(btc\_dict['month']))

weeks.append(int(btc\_dict['week']))

weekdays.append(btc\_dict['weekday'])

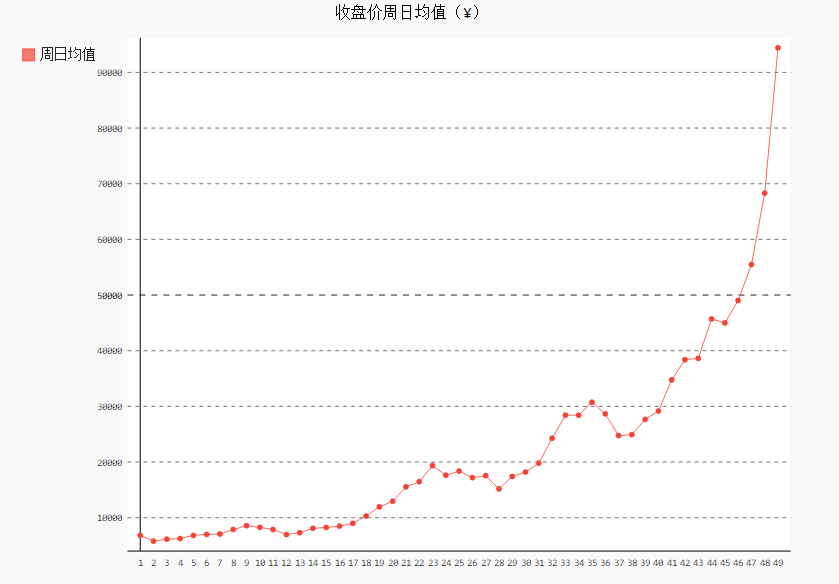
close.append(int(float(btc\_dict['close'])))

idx\_week = dates.index('2017-12-11')

line\_chart\_week = draw\_line(

weeks[1:idx\_week], close[1:idx\_week], '收盘价周日均值（¥）', '周日均值')

line\_chart\_week



#### 绘制星期均价

import json

import pygal

import math #导入数学模块

from itertools import groupby #导入模块itertools的函数groupby

def draw\_line(x\_data, y\_data, title, y\_legend):

xy\_map = []

for x, y in groupby(sorted(zip(x\_data, y\_data)), key=lambda \_: \_[0]): # 2

y\_list = [v for \_, v in y]

xy\_map.append([x, sum(y\_list) / len(y\_list)]) # 3

x\_unique, y\_mean = [\*zip(\*xy\_map)] # 4

line\_chart = pygal.Line()

line\_chart.title = title

line\_chart.x\_labels = x\_unique

line\_chart.add(y\_legend, y\_mean)

line\_chart.render\_to\_file(title + '.svg')

return line\_chart

#将数据加载到列表中

filename = r'Data\btc\_close\_2017.json'

with open(filename) as f:

btc\_data = json.load(f) #将json文件存储到变量中

#创建空列表

dates = []

months = []

weeks = []

weekdays = []

close = []

#每一天的信息

for btc\_dict in btc\_data:

dates.append(btc\_dict['date'])

months.append(int(btc\_dict['month']))

weeks.append(int(btc\_dict['week']))

weekdays.append(btc\_dict['weekday'])

close.append(int(float(btc\_dict['close'])))

idx\_week = dates.index('2017-12-11')

wd = ['Monday', 'Tuesday', 'Wednesday',

'Thursday', 'Friday', 'Saturday', 'Sunday']

weekdays\_int = [wd.index(w) + 1 for w in weekdays[1:idx\_week]]

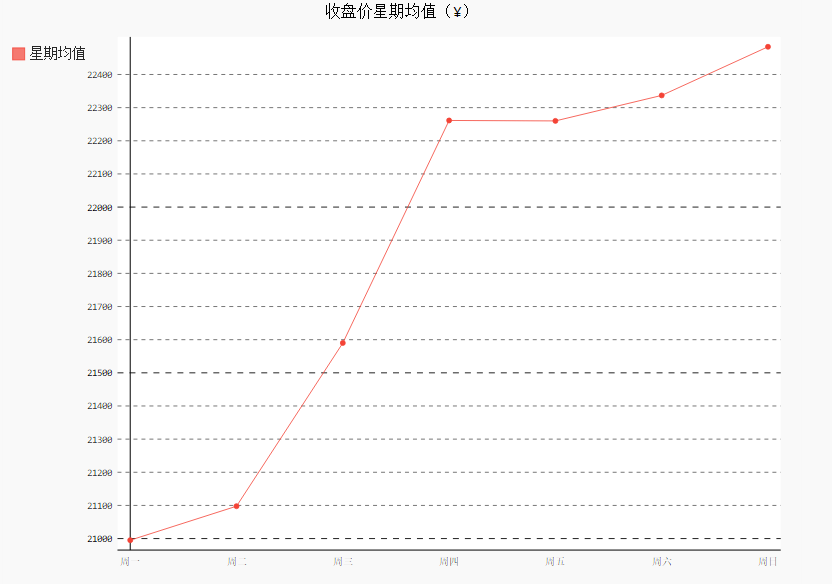
line\_chart\_weekday = draw\_line(

weekdays\_int, close[1:idx\_week], '收盘价星期均值（¥）', '星期均值')

line\_chart\_weekday.x\_labels = ['周一', '周二', '周三', '周四', '周五', '周六', '周日']

line\_chart\_weekday.render\_to\_file('收盘价星期均值（¥）.svg')

line\_chart\_weekday



### 数据仪表盘

#### 将图表写入html

with open('收盘价Dashboard.html', 'w', encoding='utf8') as html\_file:

html\_file.write(

'<html><head><title>收盘价Dashboard</title><meta charset="utf-8"></head><body>\n')

for svg in [

'收盘价折线图.svg', '收盘价对数变换折线图.svg', '收盘价月日均值（¥）.svg',

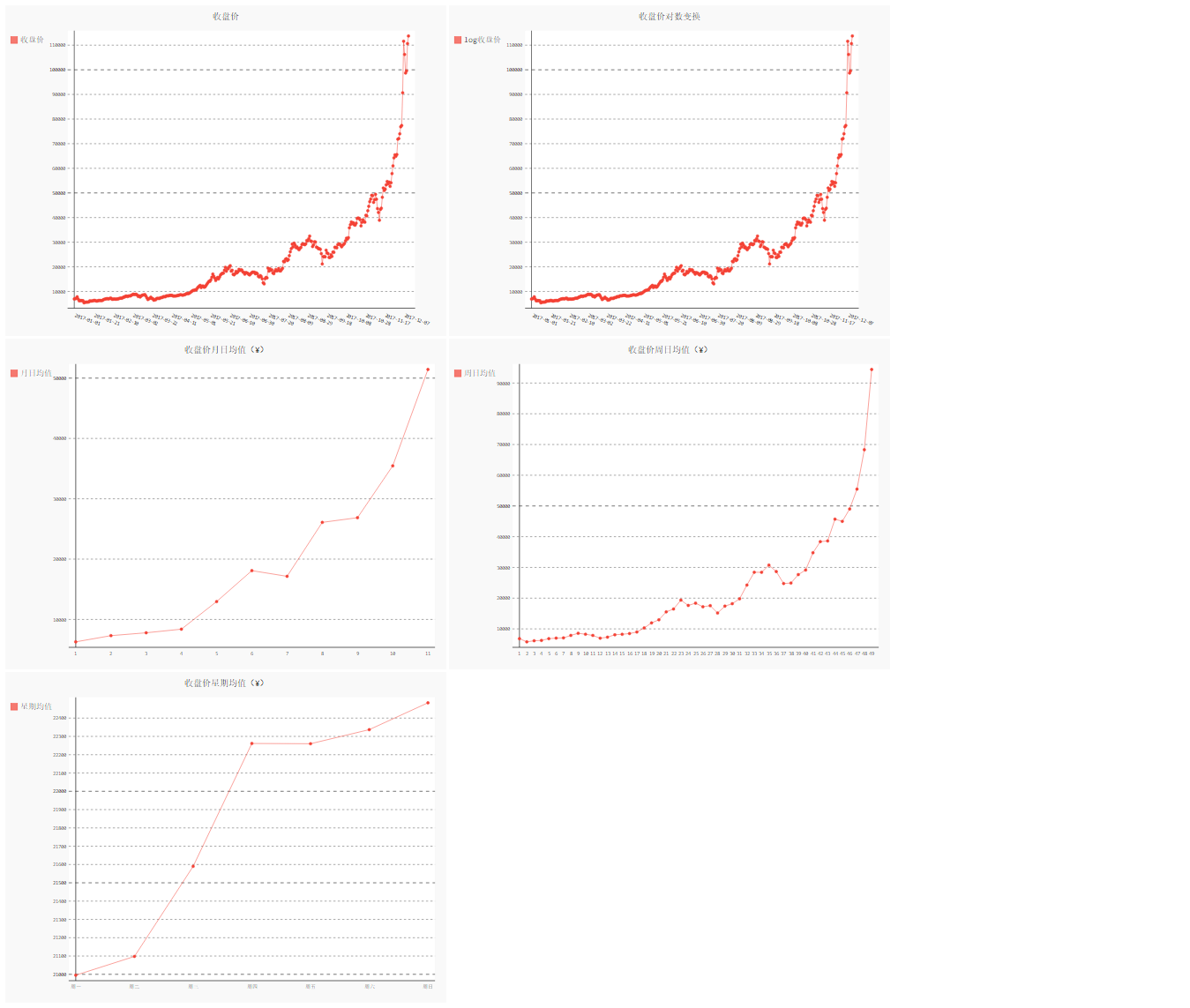
'收盘价周日均值（¥）.svg', '收盘价星期均值（¥）.svg'

]:

html\_file.write(

' <object type="image/svg+xml" data="{0}" height=500></object>\n'.format(svg)) # 1

html\_file.write('</body></html>')



# 使用API

## 使用Web API

### 处理API响应

#### 获取API数据（json）

import requests

#执行API调用并存储响应

url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'

r = requests.get(url) #将url传递给调用方法get()，然后将响应对象存储在变量r中

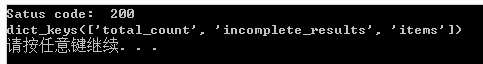
print("Satus code: ",r.status\_code) #status\_code属性，返回请求状态码。请求成功的代码为200

#将API响应存储到变量

response\_dict = r.json() #将API响应的json数据转换为Python字典

#处理结果

print(response\_dict.keys()) #打印所有键值.keys()



### 处理响应字典

#### 获取json的值

import requests

#执行API调用并存储响应

url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'

r = requests.get(url) #将url传递给调用方法get()，然后将响应对象存储在变量r中

print("Satus code: ",r.status\_code) #status\_code属性，返回请求状态码。请求成功的代码为200

#将API响应存储到变量

response\_dict = r.json() #将API响应的json数据转换为Python字典

print("Total respositories: ",response\_dict['total\_count'])

#items值是一个列表，当中包括多个字典。

repo\_dicts = response\_dict['items'] #将字典列表赋值给变量。

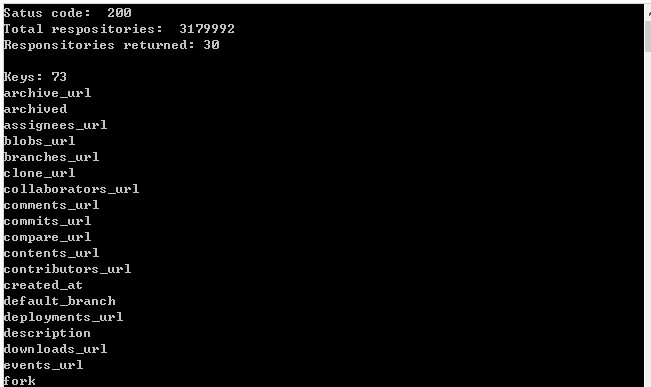
print("Responsitories returned:",len(repo\_dicts)) #返回列表长度

repo\_dict = repo\_dicts[0] #获取第一个字典并存储在变量中

print("\nKeys:",len(repo\_dict)) #返回字典的键数

for key in sorted(repo\_dict.keys()): #遍历所有键值

print(key)



#### 获取某一对象的重要数据

import requests

#执行API调用并存储响应

url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'

r = requests.get(url) #将url传递给调用方法get()，然后将响应对象存储在变量r中

print("Satus code: ",r.status\_code) #status\_code属性，返回请求状态码。请求成功的代码为200

#将API响应存储到变量

response\_dict = r.json() #将API响应的json数据转换为Python字典

print("Total respositories: ",response\_dict['total\_count'])

#items值是一个列表，当中包括多个字典。

repo\_dicts = response\_dict['items'] #将字典列表赋值给变量。

print("Responsitories returned:",len(repo\_dicts)) #返回列表长度

repo\_dict = repo\_dicts[0] #获取第一个字典并存储在变量中

print("\nSelected information about first repository:")

print('Name:',repo\_dict['name']) #使用键获取值数据

print('Owner:',repo\_dict['owner']['login']) #使用键owner获取字典，然后使用键key获取值数据

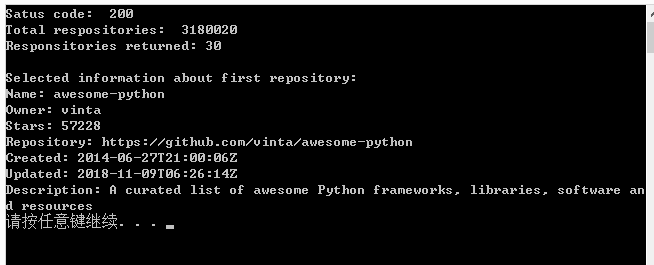
print('Stars:',repo\_dict['stargazers\_count'])

print('Repository:',repo\_dict['html\_url'])

print('Created:',repo\_dict['created\_at'])

print('Updated:',repo\_dict['updated\_at'])

print('Description:',repo\_dict['description'])



### 概述最受欢迎的仓库

#### 遍历所有字典

import requests

#执行API调用并存储响应

url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'

r = requests.get(url) #将url传递给调用方法get()，然后将响应对象存储在变量r中

print("Satus code: ",r.status\_code) #status\_code属性，返回请求状态码。请求成功的代码为200

#将API响应存储到变量

response\_dict = r.json() #将API响应的json数据转换为Python字典

print("Total respositories: ",response\_dict['total\_count'])

#items值是一个列表，当中包括多个字典。

repo\_dicts = response\_dict['items'] #将字典列表赋值给变量。

print("Responsitories returned:",len(repo\_dicts)) #返回列表长度

print("\nSelected information about first repository:")

for repo\_dict in repo\_dicts: #遍历所有字典

print('\nName:',repo\_dict['name']) #使用键获取值数据

print('Owner:',repo\_dict['owner']['login']) #使用键owner获取字典，然后使用键key获取值数据

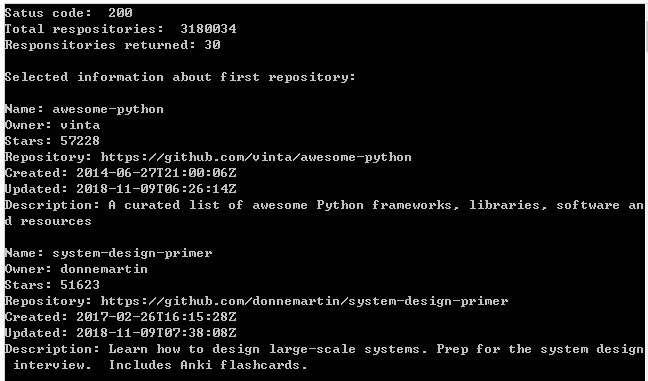
print('Stars:',repo\_dict['stargazers\_count'])

print('Repository:',repo\_dict['html\_url'])

print('Created:',repo\_dict['created\_at'])

print('Updated:',repo\_dict['updated\_at'])

print('Description:',repo\_dict['description'])



## 使用Pygal可视化仓库

#### 使用Pygal可视化仓库

import requests

import pygal

from pygal.style import LightColorizedStyle as LCS, LightenStyle as LS

#执行API调用并存储响应

url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'

r = requests.get(url) #将url传递给调用方法get()，然后将响应对象存储在变量r中

print("Satus code: ",r.status\_code) #status\_code属性，返回请求状态码。请求成功的代码为200

#将API响应存储到变量

response\_dict = r.json() #将API响应的json数据转换为Python字典

print("Total respositories: ",response\_dict['total\_count'])

#items值是一个列表，当中包括多个字典。

repo\_dicts = response\_dict['items'] #将字典列表赋值给变量。

names,stars = [], []

for repo\_dict in repo\_dicts: #遍历所有字典

names.append(repo\_dict['name'])

stars.append(repo\_dict['stargazers\_count'])

#可视化

my\_style = LS('#333366',base\_style=LCS)

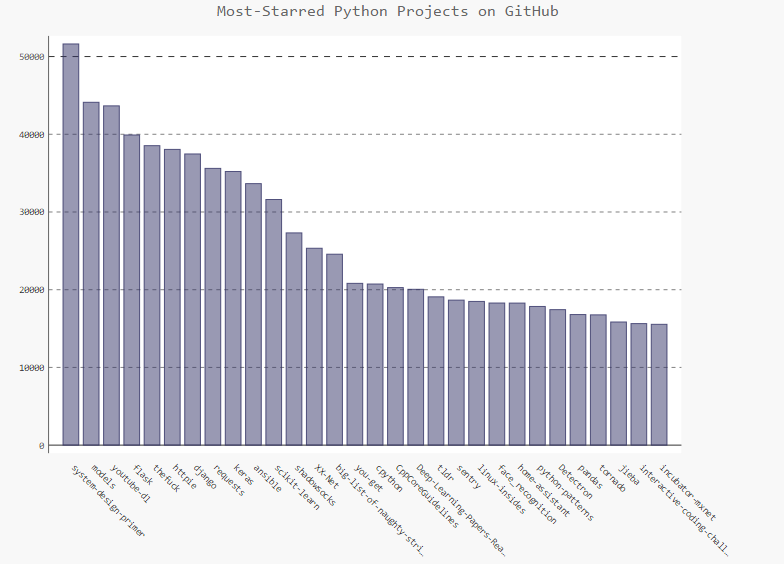
chart = pygal.Bar(style=my\_style,x\_label\_rotation=45,show\_legend=False)

chart.title = 'Most-Starred Python Projects on GitHub'

chart.x\_labels = names

chart.add('',stars)

chart.render\_to\_file('python\_repos.svg')



### 改进Pygal图表

#### 改进Pygal图表

import requests

import pygal

from pygal.style import LightColorizedStyle as LCS, LightenStyle as LS

#执行API调用并存储响应

url = 'https://api.github.com/search/repositories?q=language:python&sort=stars'

r = requests.get(url) #将url传递给调用方法get()，然后将响应对象存储在变量r中

print("Satus code: ",r.status\_code) #status\_code属性，返回请求状态码。请求成功的代码为200

#将API响应存储到变量

response\_dict = r.json() #将API响应的json数据转换为Python字典

print("Total respositories: ",response\_dict['total\_count'])

#items值是一个列表，当中包括多个字典。

repo\_dicts = response\_dict['items'] #将字典列表赋值给变量。

names,stars = [], []

for repo\_dict in repo\_dicts: #遍历所有字典

names.append(repo\_dict['name'])

stars.append(repo\_dict['stargazers\_count'])

#可视化

my\_style = LS('#333366',base\_style=LCS)

my\_config = pygal.Config() #定义Config的实例，通过修改该实例可以定制图表外观

my\_config.x\_label\_rotation = 45

my\_config.show\_legend = False

my\_config.title\_font\_size = 24

my\_config.label\_font\_size = 14

my\_config.major\_label\_font\_size = 18

my\_config.truncate\_label = 15 #将较长的项目名缩短为15个字符

my\_config.show\_y\_guides = False #隐藏水平线

my\_config.width = 1000 #自定义宽度

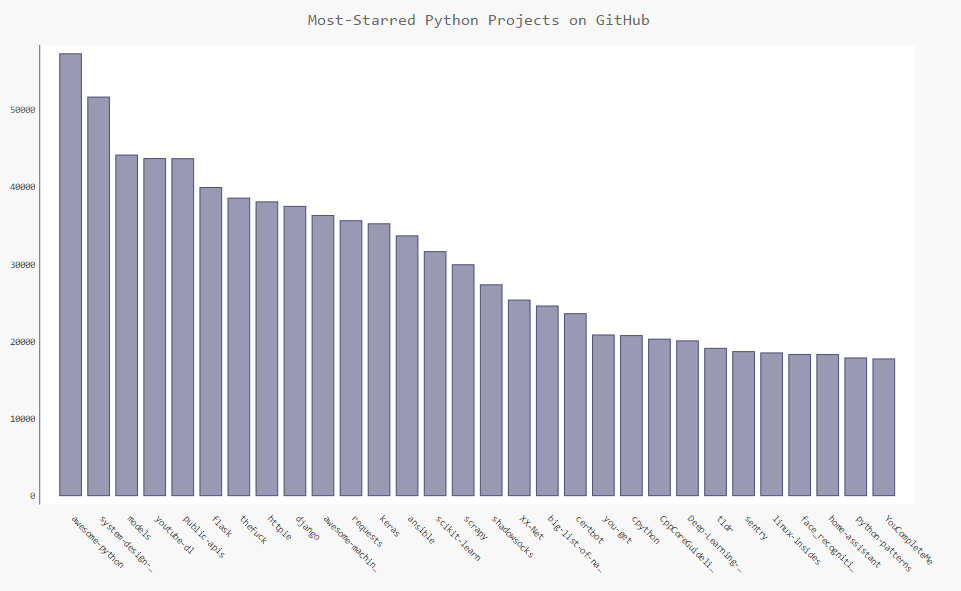
chart = pygal.Bar(my\_config,style=my\_style) #将Config实例传递给Bar实例

chart.title = 'Most-Starred Python Projects on GitHub'

chart.x\_labels = names

chart.add('',stars)

chart.render\_to\_file('python\_repos.svg')



### 添加自定义工具提示

#### 添加自定义工具提示

import pygal

from pygal.style import LightColorizedStyle as LCS, LightenStyle as LS

my\_style = LS('#333366', base\_style=LCS)

chart = pygal.Bar(style=my\_style, x\_label\_rotation=45, show\_legend=False)

chart.title = 'Python Projects'

chart.x\_labels = ['httpie', 'django', 'flask']

#定义一个列表，当中包含value和label键。

#Pygal根据键value的值确定图形高度，根据label的值显示工具提示

plot\_dicts = [

{'value': 16101, 'label': 'Description of httpie.'},

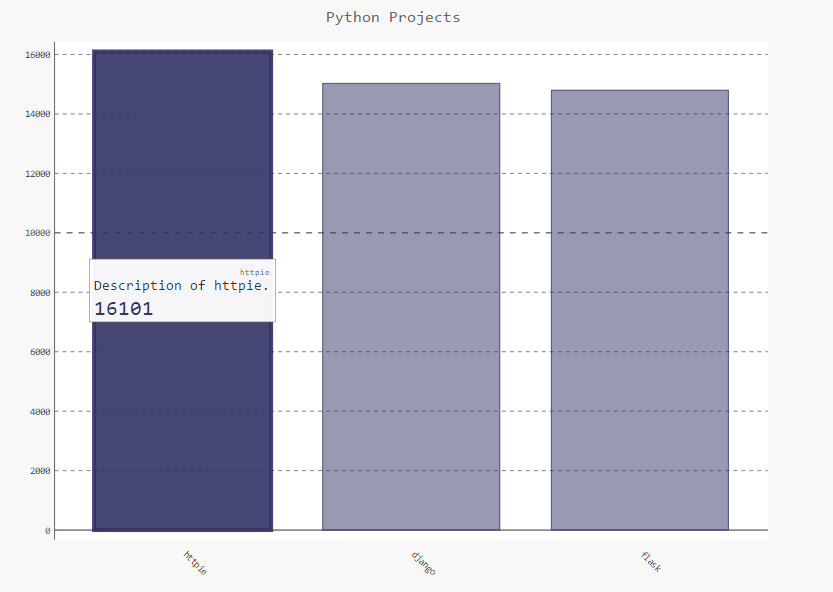
{'value': 15028, 'label': 'Description of django.'},

{'value': 14798, 'label': 'Description of flask.'},

]

chart.add('', plot\_dicts)

chart.render\_to\_file('bar\_descriptions.svg')



### 根据数据绘图

#### 根据数据绘图

发生异常待续

### 在图表中添加可单击的链接

#### 在图表中添加可单击的链接

发生异常待续

## Hacker News API

网络被墙，待续

#### Hacker News API

网络被墙，待续

# Django入门

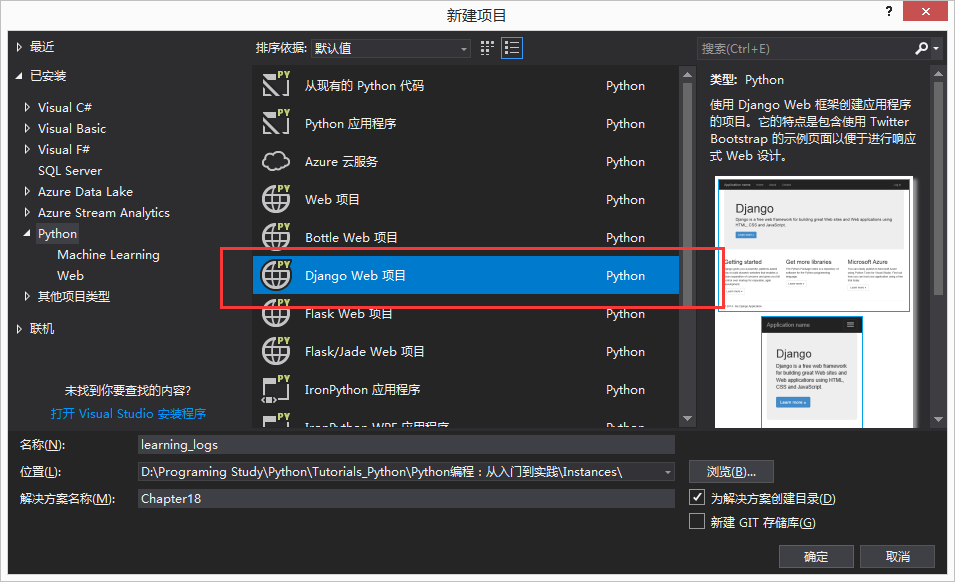
## 创建解决方案

书上使用的是Python3虚拟环境 + Django。

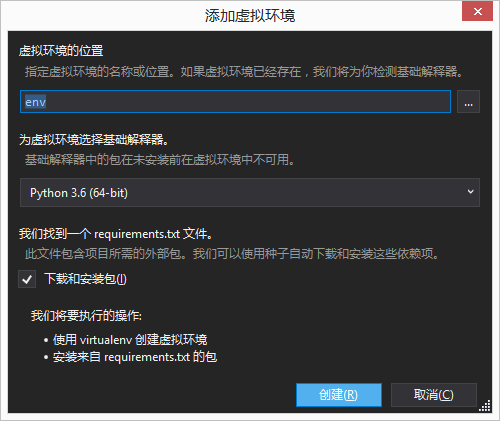
实际上使用的是Visual Studio 2017。

以下内容按照VS环境操作，不一定按照书中步骤进行操作，实例名称learning\_logs。

## 创建应用程序

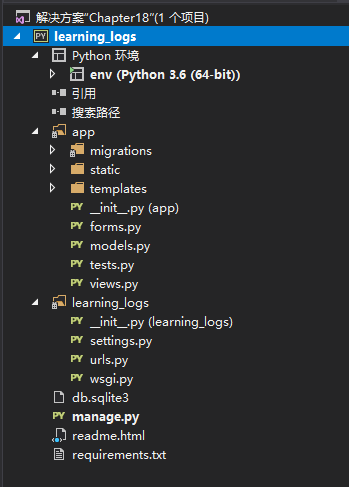


过程中会提示安装虚拟环境

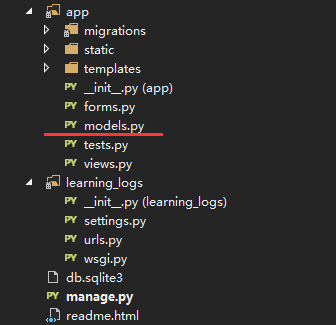


### 文件夹结构

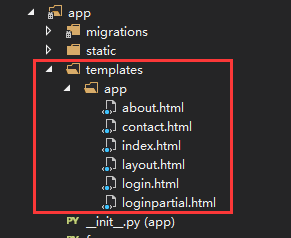
创建完毕后的文件夹结构：



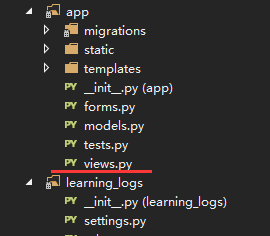
Django也是一个MVC框架。但是在Django中，控制器接受用户输入的部分由框架自行处理，所以 Django 里更关注的是模型（Model）、模板(Template)和视图（Views），称为 MTV模式：  
  
    M 代表模型（Model），即数据存取层。 该层处理与数据相关的所有事务： 如何存取、如何验证有效性、包含哪些行为以及数据之间的关系等。



    T 代表模板(Template)，即表现层。 该层处理与表现相关的决定： 如何在页面或其他类型文档中进行显示。



    V 代表视图（View），即业务逻辑层。 该层包含存取模型及调取恰当模板的相关逻辑。 你可以把它看作模型与模板之间的桥梁。

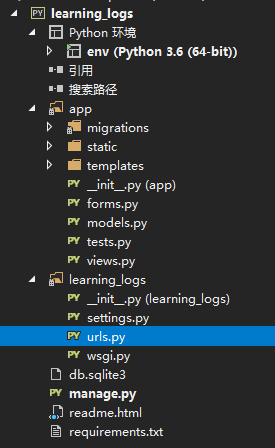


需要注意的是，不能简单的把 Django 视图认为是MVC控制器，把 Django 模板认为MVC视图。   
区别在于：  
     Django 视图 不处理用户输入，而仅仅决定要展现哪些数据给用户；  
     Django 模板 仅仅决定如何展现Django视图指定的数据。

或者说, Django将MVC中的视图进一步分解为 Django视图 和 Django模板两个部分，分别决定 “展现哪些数据” 和 “如何展现”，使得Django的模板可以根据需要随时替换，而不仅仅限制于内置的模板。

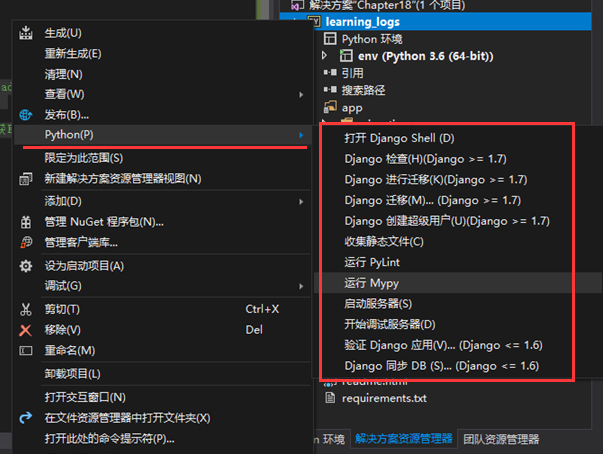
至于MVC控制器部分，由Django框架的URLconf来实现。

网站路由系统由urls.py定义：



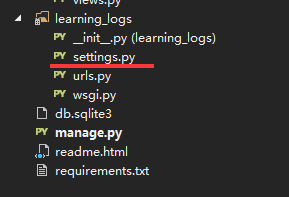
### Python基本操作

在vs中，一些Python的基本操作可以在右键菜单中选择。



### 注册应用程序

在setttings.py文件中注册需要用到的模型



# Application definition

# 定义项目由哪些应用程序组成。

INSTALLED\_APPS = [

'app',

# Add your apps here to enable them

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

# 自定义的应用程序

'learning\_logs',

]

## 定义模型

### 创建模型

在models.py中代码定义学习主题Topic：

"""

Definition of models.

"""

from django.db import models #导入模块model

# Create your models here.

class Topic(models.Model): #所有模型都继承自Model类，该模型下定义了text和date\_added两个属性

#使用方法CharField定义该属性为由字符或文本组成的数据，最大长度为200个字符。

text = models.CharField(max\_length=200)

#使用方法DateTimeField定义该属性为记录日期和时间的数据，将该属性设置为自动获取当前时间。

date\_added = models.DateTimeField(auto\_now\_add=True)

#定义默认使用text属性来显示该个类的信息。

#方法\_\_str\_\_：返回存储在属性text中字符串

def \_\_str\_\_(self):

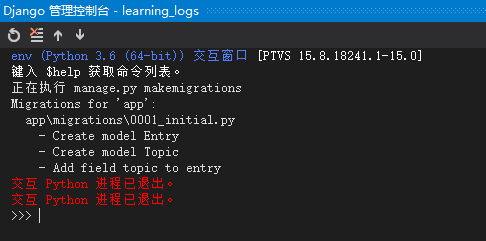
return self.text

### 迁移数据库

1. 由于模型类被修改，则需要迁移数据库。首先选择“进行迁移”命令，生成迁移文件。

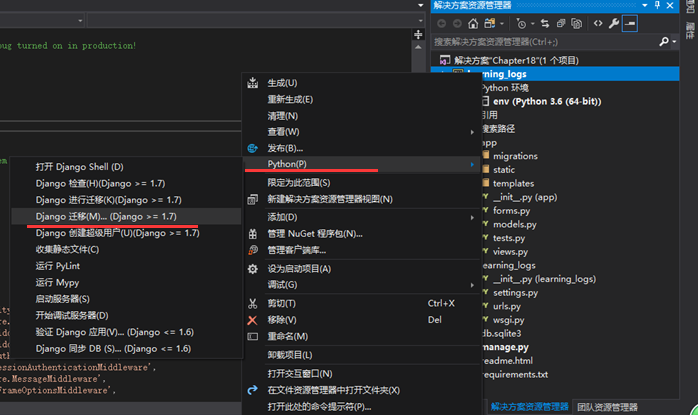


然后在控制台会输出：

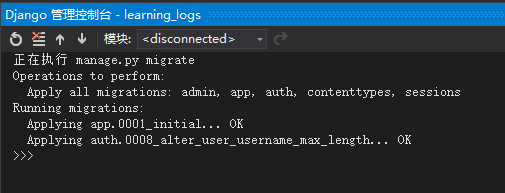


表明已经创建了一个名为0001\_initial.py的迁移文件。该文件会在数据库中为模型Topic创建一个表。

1. 迁移数据库



在控制台会输出：



表明已经顺利迁移。

总结：当需要修改数据结构时，采取的步骤是：

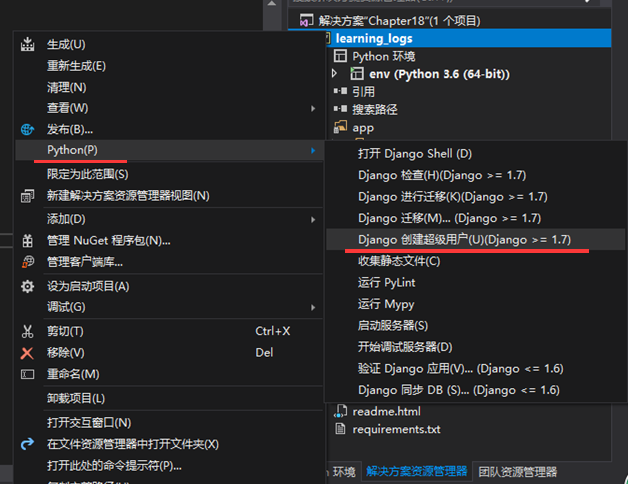
* 1. 在models.py文件中修改模型；
  2. 对项目调用Python命令“makemigrations”；
  3. 进行迁移“migrate”。

## 管理网页（admin site）

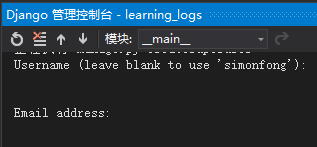
Django提供了管理网页（admin site），可以轻松处理模型。但必须是超级管理员才能使用。

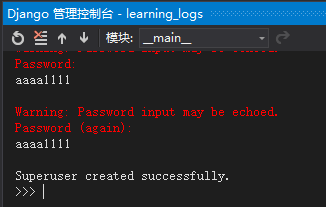
### 创建超级管理员

创建超级管理员



在控制台中输入用户名和密码：





### 修改路由器

在路由器中添加对管理页面（admin site）解析。

# 添加admin相关模块

from django.conf.urls import include

from django.contrib import admin

admin.autodiscover()

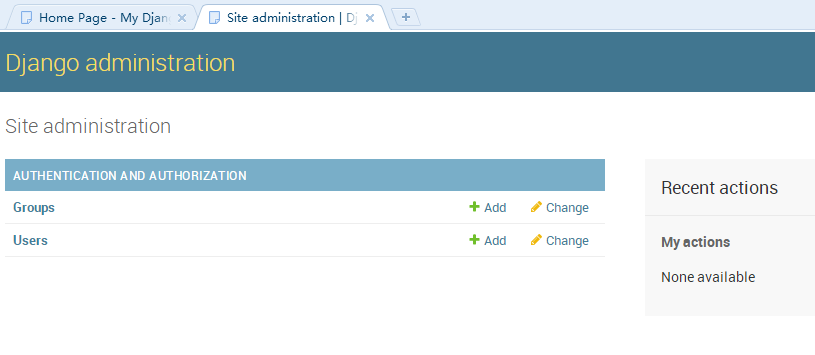
urlpatterns = [

# Examples:

url(r'^$', app.views.home, name='home'),

url(r'^admin/',include(admin.site.urls)), #添加对admin site的路由解析

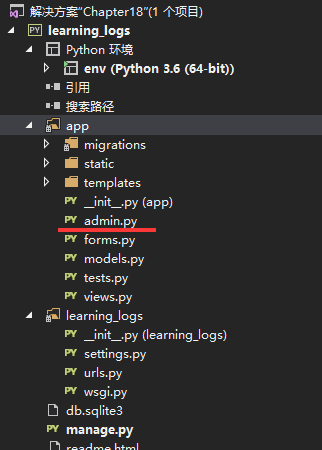
试运行网站，输入http://localhost/admin/时会转到登录页面，使用刚刚创建的超级管理员账号登录。



在页面并没有显示自定义的模型Topic，原因是没有在应用程序中注册该模型。

### 向应用程序注册模型

在应用程序的根目录中创建admin.py文件。



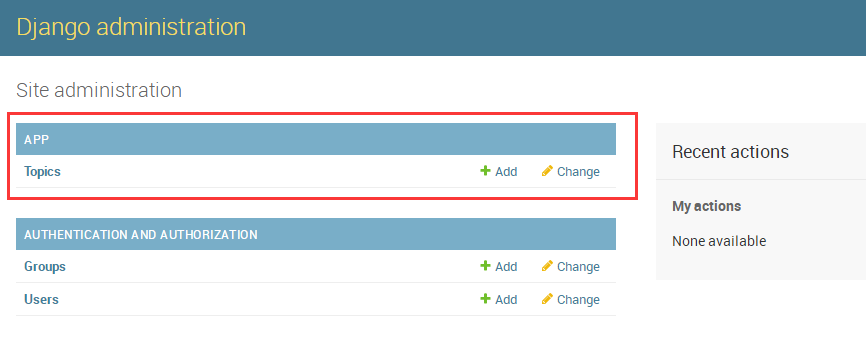
输入代码：

from django.contrib import admin

from app.models import Topic

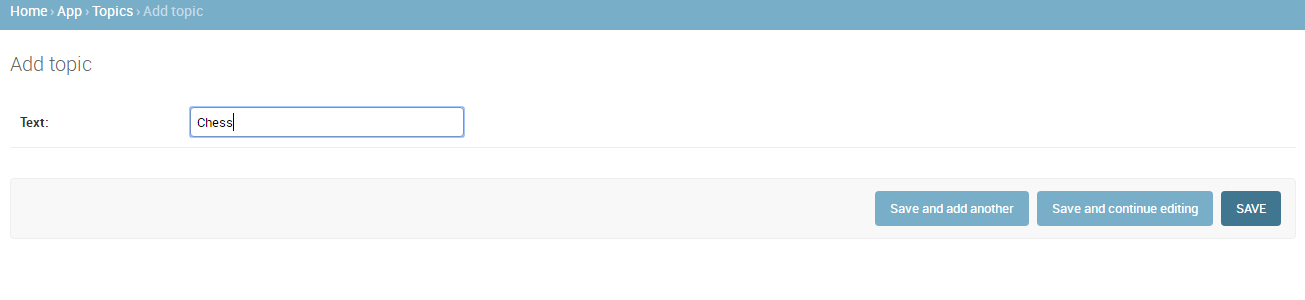
admin.site.register(Topic) #在管理页面（admin site）中注册Topic模型

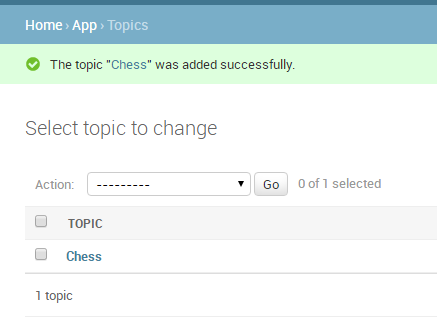
再次调试网站，在admin site中显示有自定义的Topic模型。



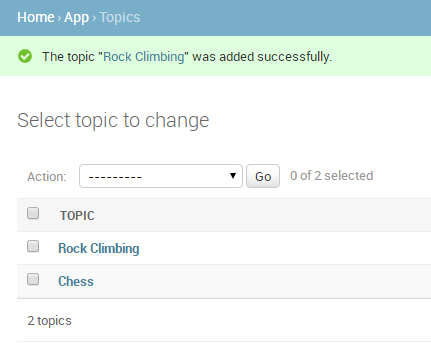
### 在管理者网页中进行操作

在管理者页面中可以对Topic进行操作





在实例中，将创建两个主题Topic。



### 定义模型Entry

1. 修改models.py：

"""

Definition of models.

"""

from django.db import models #导入模块model

from django.contrib.auth.models import User

# Create your models here.

class Topic(models.Model): #所有模型都继承自Model类，该模型下定义了text和date\_added两个属性

#使用方法CharField定义该属性为由字符或文本组成的数据，最大长度为200个字符。

text = models.CharField(max\_length=200)

#使用方法DateTimeField定义该属性为记录日期和时间的数据，将该属性设置为自动获取当前时间。

date\_added = models.DateTimeField(auto\_now\_add=True)

#定义默认使用text属性来显示该个类的信息。

#方法\_\_str\_\_：返回存储在属性text中字符串

def \_\_str\_\_(self):

return self.text

class Entry(models.Model):

"""Something specific learned about a topic."""

topic = models.ForeignKey(Topic,on\_delete=models.CASCADE) #定义该属性为一个外键，令到两个数据之间建立联系。

text = models.TextField() #定义属性为文本

date\_added = models.DateTimeField(auto\_now\_add=True)

class Meta:

verbose\_name\_plural = 'entries' #嵌套了Meta类，存储用于管理模型的额外信息。

def \_\_str\_\_(self): #定义显示该个类时显示的是哪个属性。该处显示的是不超过50个字符的text属性内容。

"""Return a string representation of the model."""

return self.text[:50] + "..."

1. 迁移模型

执行makemigrations和migrate命令

1. 向管理页面注册模型Entry

修改admin.py：

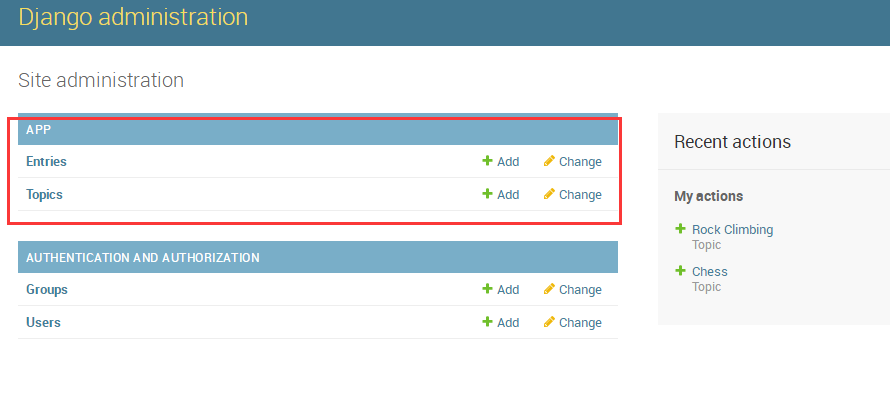
from django.contrib import admin

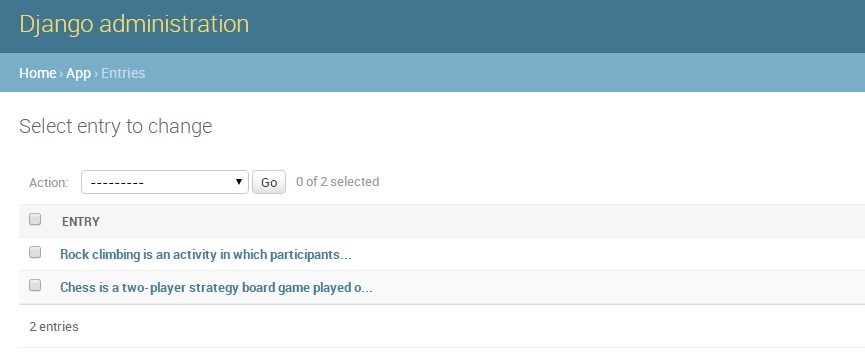
from app.models import Topic, Entry

admin.site.register(Topic) #在管理页面（admin site）中注册Topic模型

admin.site.register(Entry) #在管理页面（admin site）中注册Topic模型

1. 调试网站，并在admin site中添加Entry内容





## 创建网页

使用Django创建网页的过程分三个阶段：

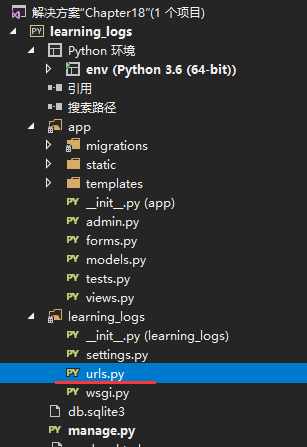
1. 定义URL
2. 编写视图View
3. 编写模板Template

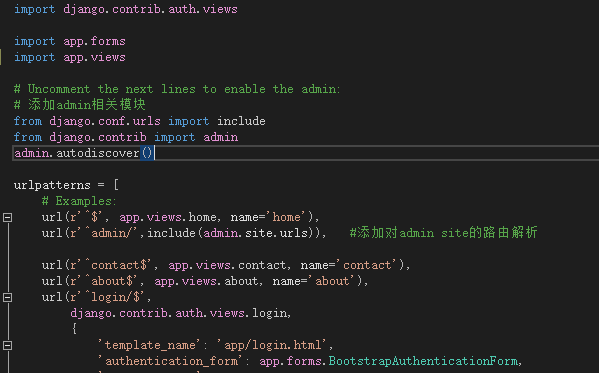
每个URL都被映射到特定的视图函数。视图函数获取并处理网页所需的数据。

视图函数通常调用一个模板。模板生成浏览器能解析的网页。

### 映射URL

在urls.py文件中修改映射URL。





主要使用函数url()。它包括3个实参：

例子：url(r'^$', app.views.home, name='home'),

第1个是正则表达式。定义与请求的URL字符串匹配的正则表达式。r'^$'

第2个是指定了调用的视图函数。当请求的URL与正则表达式匹配时，将调用相应的视图函数。app.views.home

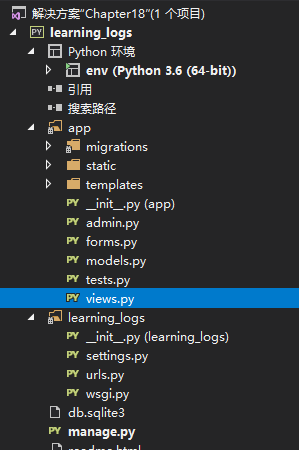
第3个是指定该URL模式的名称。当在其他地方引用该模式时，直接使用该名称。name='home'

修改urls.py代码：

url(r'^$', app.views.home, name='index'),

### 修改视图

视图函数接受请求中的信息，处理生成网页所需的数据，再将这些数据发送给模板。最后在浏览器呈现数据。



在view.py文件中修改代码：

"""

Definition of views.

"""

from django.shortcuts import render #导入函数render()，它根据视图提供的数据渲染响应。

from django.http import HttpRequest

from django.template import RequestContext

from datetime import datetime

def index(request): #定义当请求的url匹配时，如何显示并指定模板

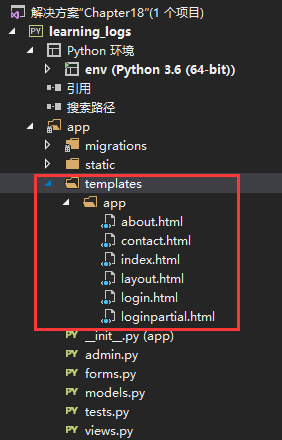
""" index主页"""

return render(request,'app/index.html')

### 修改模板

模板定义了网页的结构，指定网页如何显示数据。

模板存放在templates文件夹中：

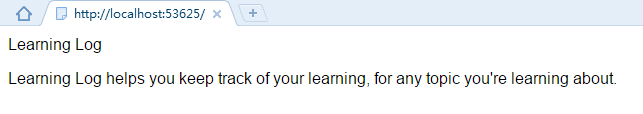


创建或修改index.html:

<p>Learning Log</p>

<p>Learning Log helps you keep track of your learning, for any topic you're learning about.</p>

### 调试



### 总结

以上创建网页的过程，充分显示Django的MTV框架，令到URL、模型Model、视图View、模板Template相互间分离，达到解耦的效果。

数据库专家专注于模型，程序员专注于视图，Web设计人员专注于模板。

## 模板继承

模板间可以继承。

VS下默认的父模板为layout.html

### 父模板

模板标签使用{% %}标识。

创建父模板base.html：

<p>

<a href="{% url 'index' %}">Learning Log</a>

</p>

<!-- 该处使用了一对名为content的块标签，

其显示的信息将由子模板指定，在父模板中可以使用多个预留空间 -->

{% block content %}{% endblock content %}

### 子模板

创建或修改index.html:

指定父模板

{% extends "app/base.html" %}

该处使用了一对名为content的块标签，

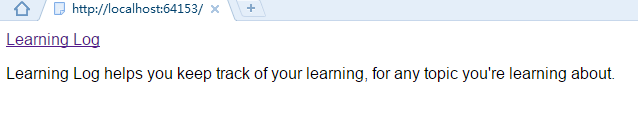
其显示的信息将由子模板指定，在父模板中可以使用多个预留空间

{% block content %}

<p>Learning Log helps you keep track of your learning, for any topic you're learning about.</p>

{% endblock content %}

### 调试



## 显示所有主题Topic

### 修改URL

urlpatterns = [

# Examples:

url(r'^$', app.views.index, name='index'),

url(r'^admin/',include(admin.site.urls)), #添加对admin site的路由解析

url(r'^topics/$',views.topics,name='topics'), #添加对/topics/的路由解析

### 修改视图

"""

Definition of views.

"""

from django.shortcuts import render #导入函数render()，它根据视图提供的数据渲染响应。

from django.http import HttpRequest

from django.template import RequestContext

from datetime import datetime

from models import Topic

def index(request): #定义当请求的url匹配时，如何显示并指定模板

""" index主页"""

return render(request,'app/index.html')

def topics(request):

""" 显示所有的主题 """

topics = Topic.objects.order\_by('date\_added')

context = {'topics':topics}

return render(request,'app/topics.html',context) #将context传递给render

### 创建模板

创建模板topics.html：

{% extends "app/base.html" %}

{% block content %}

<p>Topics</p>

<ul>

{% for topic in topics %}

<li>**{{** topic }}</li>

{% empty %} {# 当topics为空时如何处理 #}

<li>No topics have been added yet.</li>

{% endfor %}

</ul>

{% endblock content %}

### 添加入口URL

修改base.html：

<p>

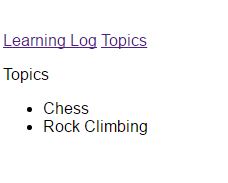
<a href="{% url 'index' %}">Learning Log</a>

<a href="{% url 'topics' %}">Topics</a>

</p>

{% block content %}{% endblock content %}

### 调试



## 显示特定主题的页面

### 添加URL

url(r'^topics/(?P<topic\_id>\d+)/$',app.views.topic,name='topic'), #添加对/topics/1/的路由解析

### 修改视图

def topic(request,topic\_id):

""" 显示单个主题及所有条目 """

topic = Topic.objects.get(id=topic\_id)

entries = topic.entry\_set.order\_by('-date\_added')

context = {'topic':topic,'entries':entries}

return render(request,'app/topic.html',context)

### 添加模板

{% extends "app/base.html" %}

{% block content %}

<p>Topic:**{{** topic }}</p>

<p>Entries:</p>

<ul>

{% for entry in entries %}

<li>

<p>**{{** entry.date\_added|date:'M d,Y H:i' }}</p>

<p>**{{** entry.text|linebreaks }}</p>

</li>

{% empty %}

<li>

There are no entries for this topic yet.

</li>

{% endfor %}

</ul>

{% endblock content %}

“|”表示模板过滤器：对模板变量的值进行修改。

过滤器date：指定显示时间的格式。

过滤器linebreaks：将包含换行符的长条目转换为浏览器能理解的格式。

### 添加入口

topics.html:

{% extends "app/base.html" %}

{% block content %}

<p>Topics</p>

<ul>

{% for topic in topics %}

<li>

<a href="{% url 'topic' topic.id %}">**{{** topic **}}**</a>

</li>

{% empty %} {# 当topics为空时如何处理 #}

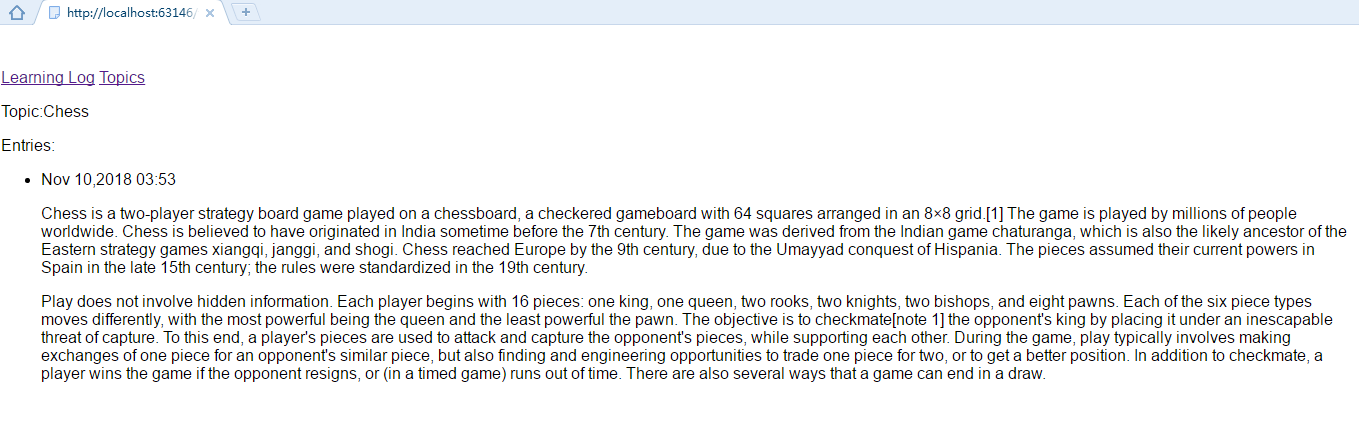
<li>No topics have been added yet.</li>

{% endfor %}

</ul>

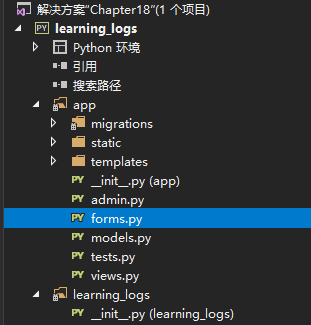
{% endblock content %}

### 调试



## 表单

网站使用的表单由forms.py文件定义：



### 修改表单文件

修改forms.py文件：

"""

Definition of forms.

"""

from django import forms

from django.contrib.auth.forms import AuthenticationForm

from django.utils.translation import ugettext\_lazy as \_

from app.models import Topic

class BootstrapAuthenticationForm(AuthenticationForm):

"""Authentication form which uses boostrap CSS."""

username = forms.CharField(max\_length=254,

widget=forms.TextInput({

'class': 'form-control',

'placeholder': 'User name'}))

password = forms.CharField(label=\_("Password"),

widget=forms.PasswordInput({

'class': 'form-control',

'placeholder':'Password'}))

class TopicForm(forms.ModelForm): #所有表单继承自ModelForm

class Meta: #内嵌Meta类，告诉Django根据那个模型创建表单和在表单中使用那些字段

model = Topic

fields = ['text']

labels = {'text':''} #该处定义了不要为字段text生成标签

### 修改URL

url(r'^new\_topic/$',app.views.new\_topic,name='new\_topic'), #添加对/new\_topics/的路由解析

### 修改视图

添加模块：

from django.http import HttpResponseRedirect

from django.core.urlresolvers import reverse

from app.forms import TopicForm

添加视图函数：

def new\_topic(request):

""" 添加新主题 """

if request.method != "POST":

#未提交数据时，创建一个新表单

form = TopicForm()

else:

#POST提交数据时，对数据进行处理

form = TopicForm(request.POST)

if form.is\_valid():

form.save()

return HttpResponseRedirect(reverse('topics')) #使用reverse()获取页面topics的URL

context = {'form':form}

return render(request,'app/new\_topic.html',context)

### 创建模板

创建模板new\_topic.html：

{% extends "app/base.html" %}

{% block content %}

<p>Add a new topic:</p>

<form action="{% url 'new\_topic' %}" method="post">

{% csrf\_token %} {# 防止跨站请求伪造 #}

**{{** form.as\_p }}

<button name="submit">add topic</button>

</form>

{% endblock content %}

### 添加入口

修改topics.html：

{% extends "app/base.html" %}

{% block content %}

<p>Topics</p>

<ul>

{% for topic in topics %}

<li>

<a href="{% url 'topic' topic.id %}">**{{** topic }}</a>

</li>

{% empty %} {# 当topics为空时如何处理 #}

<li>No topics have been added yet.</li>

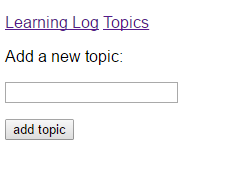
{% endfor %}

</ul>

<a href="{% url 'new\_topic' %}">Add a new topic</a>

{% endblock content %}

### 调试



## 添加新条目

### 添加新表单类：

"""

Definition of forms.

"""

from django import forms

from django.contrib.auth.forms import AuthenticationForm

from django.utils.translation import ugettext\_lazy as \_

from app.models import Topic,Entry

class BootstrapAuthenticationForm(AuthenticationForm):

"""Authentication form which uses boostrap CSS."""

username = forms.CharField(max\_length=254,

widget=forms.TextInput({

'class': 'form-control',

'placeholder': 'User name'}))

password = forms.CharField(label=\_("Password"),

widget=forms.PasswordInput({

'class': 'form-control',

'placeholder':'Password'}))

class TopicForm(forms.ModelForm): #所有表单继承自ModelForm

class Meta: #内嵌Meta类，告诉Django根据那个模型创建表单和在表单中使用那些字段

model = Topic

fields = ['text']

labels = {'text':''} #该处定义了不要为字段text生成标签

class EntryForm(forms.ModelForm):

class Meta:

model = Entry

fields = ['text']

label = {'text':''}

#定义小部件widget，是一个多行文本框。宽度为80列

widgets = {'text':forms.Textarea(attrs={'cols':80})}

### 修改URL

url(r'^new\_entry/(?P<topic\_id>\d+)/$',app.views.new\_entry,name='new\_entry'), #添加对/new\_entry/1的路由解析

### 修改视图

导入模块：

from app.forms import TopicForm,EntryForm

添加视图函数

def new\_entry(request,topic\_id):

""" 在特定主题中添加新条目 """

topic = Topic.objects.get(id=topic\_id)

if request.method != "POST":

#未提交数据时，创建一个空表单

form = EntryForm()

else:

#POST提交数据时，对数据处理

form = EntryForm(data=request.POST)

if form.is\_valid():

new\_entry = form.save(commit=False) #实参表示只将数据存储在变量中，但不保存到数据库

new\_entry.topic = topic

new\_entry.save()

return HttpResponseRedirect(reverse('topic',args=[topic\_id]))

context = {'topic':topic,'form':form}

return render(request,'app/new\_entry.html',context)

### 创建模板

创建模板new\_entry.html

{% extends "app/base.html" %}

{% block content %}

<p><a href="{% url 'topic' topic.id %}">**{{** topic }}</a></p>

<p>Add a new entry:</p>

<form action="{% url 'new\_entry' topic.id %}" method="post">

{% csrf\_token %}

**{{** form.as\_p }}

<button name="submit">add entry</button>

</form>

{% endblock content %}

### 添加入口

修改topic.html

{% extends "app/base.html" %}

{% block content %}

<p>Topic:**{{** topic }}</p>

<p>Entries:</p>

<a href="{% url 'new\_entry' topic.id %}">add new entry</a>

<ul>

{% for entry in entries %}

<li>

<p>**{{** entry.date\_added|date:'M d,Y H:i' }}</p>

<p>**{{** entry.text|linebreaks }}</p>

</li>

{% empty %}

<li>

There are no entries for this topic yet.

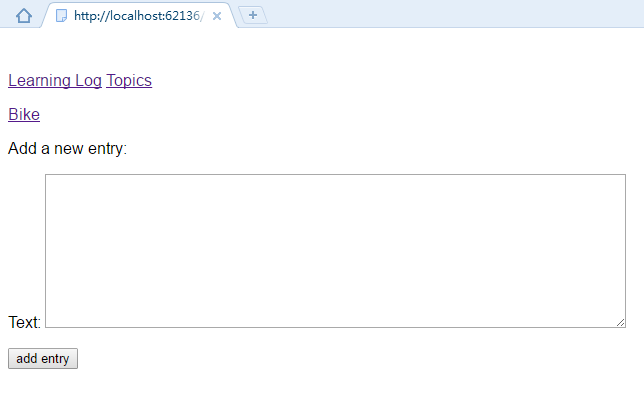
</li>

{% endfor %}

</ul>

{% endblock content %}

### 调试



## 编辑条目

### 修改URL

url(r'^edit\_entry/(?P<entry\_id>\d+)/$',app.views.edit\_entry,name='edit\_entry'), #添加对/edit\_entry/2的路由解析

### 修改视图

导入模块：

from app.models import Topic,Entry

添加视图函数：

def edit\_entry(request,entry\_id):

"""编辑既有条目"""

entry = Entry.objects.get(id=entry\_id)

topic = entry.topic

# #纭璇锋眰鐨勪富棰樺睘浜庡綋鍓嶇敤鎴?

#if topic.owner !=request.user:

# raise Http404

if request.method != "POST":

# 初次请求，使用当前条目填充表单

form = EntryForm(instance=entry)

else:

#Post提交数据，对数据进行处理

form = EntryForm(instance=entry,data=request.POST)

if form.is\_valid():

form.save()

return HttpResponseRedirect(reverse("topic",args=[topic.id]))

context = {'entry':entry,'topic':topic,'form':form}

return render(request,'app/edit\_entry.html',context)

### 创建模板

创建edit\_entry.html:

{% extends "app/base.html" %}

{% block content %}

<p><a href="{% url 'topic' topic.id %}">**{{** topic }}</a></p>

<p>Edit entry:</p>

<form action="{% url 'edit\_entry' entry.id %}" method="post">

{% csrf\_token %}

**{{** form.as\_p }}

<button name="submit">save change</button>

</form>

{% endblock content %}

### 添加入口

修改topic.html:

{% extends "app/base.html" %}

{% block content %}

<p>Topic:**{{** topic }}</p>

<p>Entries:</p>

<p>

<a href="{% url 'new\_entry' topic.id %}">add new entry</a>

</p>

<ul>

{% for entry in entries %}

<li>

<p>**{{** entry.date\_added|date:'M d,Y H:i' }}</p>

<p>**{{** entry.text|linebreaks }}</p>

<p>

<a href="{% url 'edit\_entry' entry.id %}">edit entry</a>

</p>

</li>

{% empty %}

<li>

There are no entries for this topic yet.

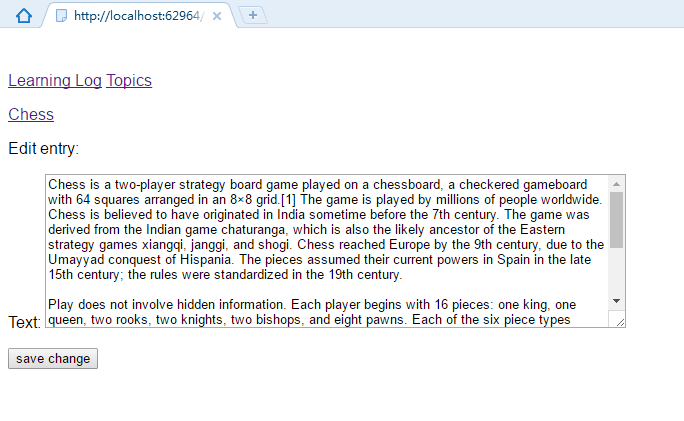
</li>

{% endfor %}

</ul>

{% endblock content %}

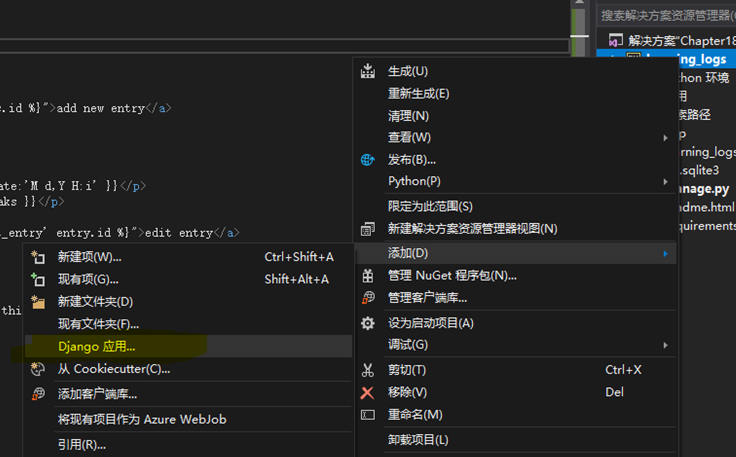
### 调试

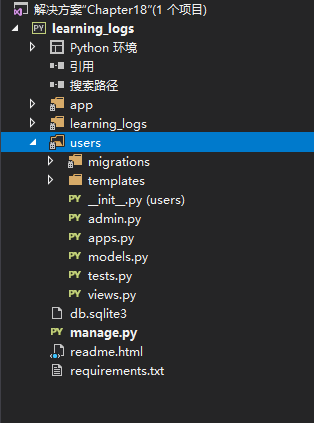


## 创建用户账户

### 创建用户应用程序

创建users应用程序：





### 注册新的应用程序

修改settings.py：

# Application definition

# 定义项目由哪些应用程序组成。

INSTALLED\_APPS = [

'app',

# Add your apps here to enable them

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

# 自定义的应用程序

'learning\_logs',

'users',

]

## 登录

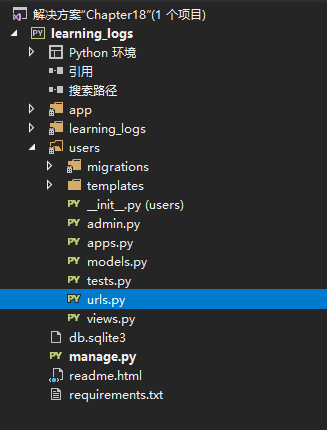
### 修改URL

添加url：

url(r'^users/',include('users.urls',namespace='users')), #将/users/\*\*\*下的路由解析指向users下的urls文件，由其负责解析

### 注册登录页面路由URL

在users下创建urls.py文件，负责解析/users/下的路由。



在users.urls.py文件中修改：

from django.conf.urls import url

from django.contrib.auth.views import login

from . import views

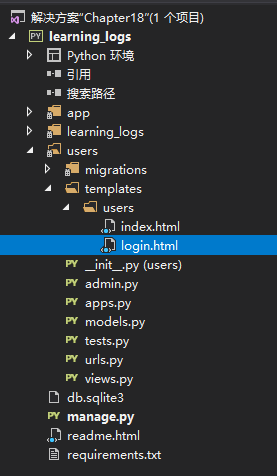
urlpatterns = [

#对/users/login/进行解析，直接指向模板，没有指定视图函数

url(r'^login/$',login,{'template\_name':"users/login.html"},name='login'),

### 创建模板

在users下创建模板login.html：



修改login.html:

{% extends "app/base.html" %}

{% block content %}

{% if form.errors %} {# 如果表单出现错误时 #}

<p>Your username and password didn't match. Please try again.</p>

{% endif %}

<form method="post" action="{% url 'users:login' %}">

{% csrf\_token %}

**{{**form.as\_p}}

<button name="submit">Log in</button>

<input type="hidden" name="next" value="{% url 'index' %}" /> {# 隐藏表单元素，当登录成功后重新指向的地方。 #}

</form>

{% endblock content %}

### 添加入口

修改base.html:

<p>

<a href="{% url 'index' %}">Learning Log</a>

<a href="{% url 'topics' %}">Topics</a>

{% if user.is\_authenticated %} {# 如果通过身份验证 #}

Hello,**{{**user.username}}.

{% else %}

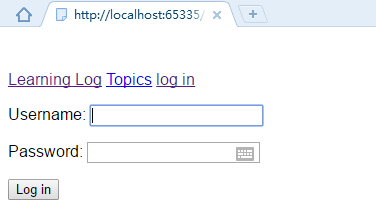
<a href="{% url 'users:login' %}">log in</a>

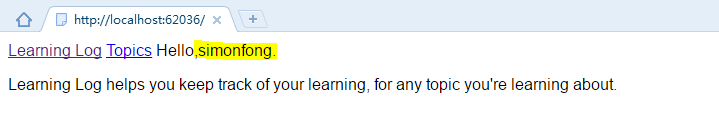
{% endif %}

</p>

{% block content %}{% endblock content %}

### 调试





## 注销

### 修改URL

修改users下urls.py：

from datetime import datetime

from django.conf.urls import url

import django.contrib.auth.views

from django.conf.urls import url

from django.contrib.auth.views import login

from . import views

urlpatterns = [

#对/users/login/进行解析

url(r'^login/$',login,{'template\_name':"users/login.html"},name='login'),

#对/users/logout/进行解析

url(r'^logout/$',views.logout\_view,name='logout'),

### 修改视图函数

修改users/views.py：

from django.shortcuts import render

from django.http import HttpResponseRedirect

from django.core.urlresolvers import reverse

from django.contrib.auth import logout

def logout\_view(request):

""" 注销用户 """

logout(request) #直接调用函数logout

return HttpResponseRedirect(reverse('index')) #重新定向到主页

### 添加入口

修改base.html：

<p>

<a href="{% url 'index' %}">Learning Log</a>

<a href="{% url 'topics' %}">Topics</a>

{% if user.is\_authenticated %} {# 如果通过身份验证 #}

Hello,**{{**user.username}}.

<a href="{% url 'users:logout' %}">log out</a>

{% else %}

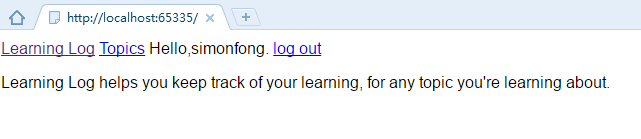
<a href="{% url 'users:login' %}">log in</a>

{% endif %}

</p>

{% block content %}{% endblock content %}

### 调试



## 用户注册

主要调用Django提供的表单UserCreationForm。

### 修改URL

修改user/urls.py:

#对/users/register/进行解析

url(r'^register/$',views.register,name='register'),

### 修改视图

修改users/views.py：

from django.shortcuts import render

from django.http import HttpResponseRedirect

from django.core.urlresolvers import reverse

from django.contrib.auth import logout

from django.shortcuts import render

from django.contrib.auth import login,logout,authenticate

from django.contrib.auth.forms import UserCreationForm

def logout\_view(request):

""" 注销用户 """

logout(request) #直接调用函数logout

return HttpResponseRedirect(reverse('index')) #重新定向到主页

def register(request):

""" 注册新用户 """

if request.method != 'POST':

#显示空的表单

form = UserCreationForm()

else:

#处理填好的表单

form = UserCreationForm(data=request.POST)

if form.is\_valid():

new\_user = form.save()

#让用户自动登录并重新定向到主页

authenticated\_user = authenticate(username=new\_user.username,

password=request.POST['password1'])

login(request,authenticated\_user)

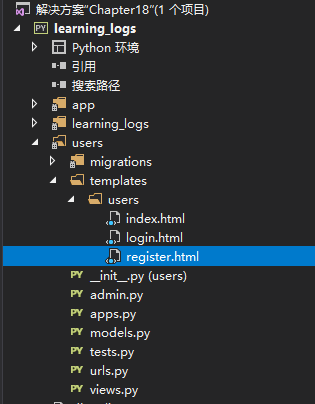
return HttpResponseRedirect(reverse('index'))

context = {'form':form }

return render(request,'users/register.html',context)

### 创建模板

在users/app/下创建register.html



代码：

{% extends "app/base.html" %}

{% block content %}

<form method="post" action="{% url 'users:register' %}">

{% csrf\_token %}

**{{**form.as\_p}}

<button name="submit">register</button>

<input type="hidden" name="next" value="{% url 'index' %}" />

</form>

{% endblock content %}

### 添加入口

修改base.html:

<p>

<a href="{% url 'index' %}">Learning Log</a>

<a href="{% url 'topics' %}">Topics</a>

{% if user.is\_authenticated %} {# 如果通过身份验证 #}

Hello,**{{**user.username}}.

<a href="{% url 'users:logout' %}">log out</a>

{% else %}

<a href="{% url 'users:register' %}">register</a>

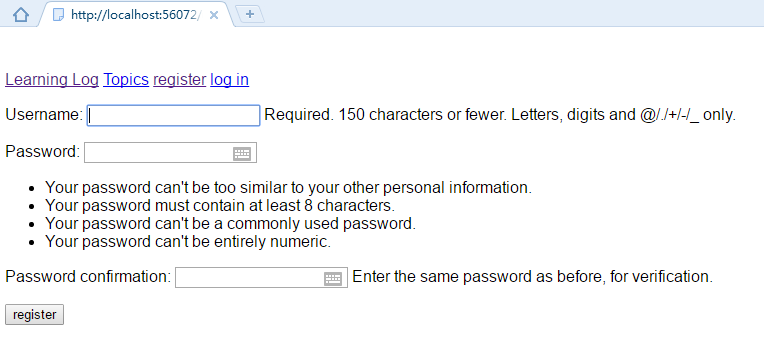
<a href="{% url 'users:login' %}">log in</a>

{% endif %}

</p>

{% block content %}{% endblock content %}

### 调试



## 为页面设置访问权限

使用@login\_required显示访问

### 添加修饰器

修改views.py:

from django.contrib.auth.decorators import login\_required

…

@login\_required #添加修饰器后将先运行该代码，检查用户是否登录，当登录时才运行后来的代码，未登录时重新定向

def topics(request):

""" 显示所有的主题 """

topics = Topic.objects.order\_by('date\_added')

context = {'topics':topics}

return render(request,'app/topics.html',context) #将context传递给render

### 设置重定向

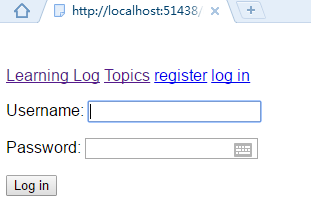
修改settings.py：以下代码添加到末尾

#重定向设置

#如果用户未登录，则添加了装饰器的代码将跳转到以下url

LOGIN\_URL = '/users/login/'

调试



### 全面限制对项目的访问

先确定哪些页面不需要保护，再限制对其他页面的访问。

可以对需要限制的视图函数上添加该限制。

## 将数据关联到用户

用户只能访问与自己相关的数据，从而保护安全。

### 修改模型

修改模型

"""

Definition of models.

"""

from django.db import models #导入模块model

from django.contrib.auth.models import User

# Create your models here.

class Topic(models.Model): #所有模型都继承自Model类，该模型下定义了text和date\_added两个属性

#使用方法CharField定义该属性为由字符或文本组成的数据，最大长度为200个字符。

text = models.CharField(max\_length=200)

#使用方法DateTimeField定义该属性为记录日期和时间的数据，将该属性设置为自动获取当前时间。

date\_added = models.DateTimeField(auto\_now\_add=True)

owner = models.ForeignKey(User) #定义到模型User的外键关系

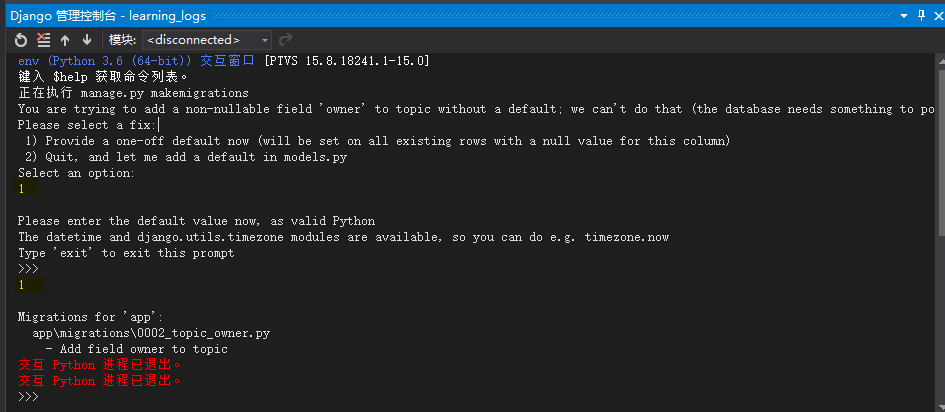
#定义默认使用text属性来显示该个类的信息。

#方法\_\_str\_\_：返回存储在属性text中字符串

def \_\_str\_\_(self):

return self.text

迁移数据库：选择选项1，由Python自动提供默认值。



### 修改视图函数

修改views.py：

1. 只允许用户访问自己的主题

@login\_required #添加修饰器后将先运行该代码，检查用户是否登录，当登录时才运行后来的代码，未登录时重新定向

def topics(request):

""" 显示所有的主题 """

topics = Topic.objects.filter(owner=request.user).order\_by('date\_added') #只允许从数据库中提取当前用户的Topic对象

context = {'topics':topics}

return render(request,'app/topics.html',context) #将context传递给render

1. 保护用户的主题

def topic(request,topic\_id):

""" 显示单个主题及所有条目 """

topic = Topic.objects.get(id=topic\_id)

if topic.owner !=request.user: #如果当前用户与主题用户不同的，返回错误页面

raise Http404

entries = topic.entry\_set.order\_by('-date\_added')

context = {'topic':topic,'entries':entries}

return render(request,'app/topic.html',context)

1. 保护编辑页面

def edit\_entry(request,entry\_id):

"""编辑既有条目"""

entry = Entry.objects.get(id=entry\_id)

topic = entry.topic

#如果当前用户与条目用户不相同的

if topic.owner !=request.user:

raise Http404

if request.method != "POST":

# 初次请求，使用当前条目填充表单

form = EntryForm(instance=entry)

else:

#Post提交数据，对数据进行处理

form = EntryForm(instance=entry,data=request.POST)

if form.is\_valid():

form.save()

return HttpResponseRedirect(reverse("topic",args=[topic.id]))

context = {'entry':entry,'topic':topic,'form':form}

return render(request,'app/edit\_entry.html',context)

1. 将新主题关联到当前用户

def new\_topic(request):

""" 添加新主题 """

if request.method != "POST":

#未提交数据时，创建一个新表单

form = TopicForm()

else:

#POST提交数据时，对数据进行处理

form = TopicForm(request.POST)

if form.is\_valid():

new\_topic = form.save(commit=False)

new\_topic.owner = request.user

new\_topic.save()

form.save()

return HttpResponseRedirect(reverse('topics')) #使用reverse()获取页面topics的URL

context = {'form':form}

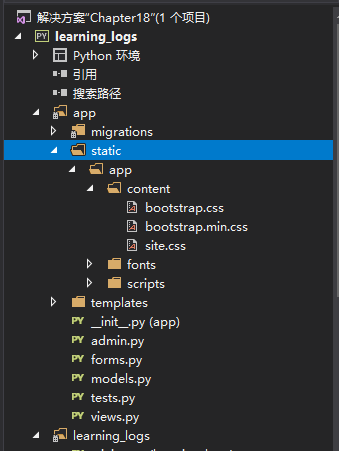
return render(request,'app/new\_topic.html',context)

### 调试

已通过调试。

## 使用样式

VS默认使用bootstrap。相关包在static文件夹下。



### 修改父模板

创建新的父母版layout.html

<!DOCTYPE html>

<html>

<head>

<title>Learning Log</title>

{# 引用bootstrap #}

{% load staticfiles %}

<link rel="stylesheet" type="text/css" href="{% static 'app/content/bootstrap.min.css' %}" />

<link rel="stylesheet" type="text/css" href="{% static 'app/content/site.css' %}" />

<script src="{% static 'app/scripts/modernizr-2.6.2.js' %}"></script>

</head>

<body>

<!-- 导航栏 -->

<div class="navbar navbar-inverse navbar-fixed-top">

<div class="container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed"

data-toggle="collapse" data-target="#navbar"

aria-expanded="false" aria-controls="navbar">

</button>

<a class="navbar-brand" href="{% url 'index' %}">

Learning Log</a>

</div>

<div id="navbar" class="navbar-collapse collapse">

<ul class="nav navbar-nav">

<li><a href="{% url 'topics' %}">Topics</a></li>

</ul>

<ul class="nav navbar-nav navbar-right">

{% if user.is\_authenticated %}

<li><a>Hello, **{{** user.username }}.</a></li>

<li><a href="{% url 'users:logout' %}">log out</a></li>

{% else %}

<li><a href="{% url 'users:register' %}">register</a></li>

<li><a href="{% url 'users:login' %}">log in</a></li>

{% endif %}

</ul>

</div><!--/.nav-collapse -->

</div>

</div>

<!-- container -->

<div class="container">

<div class="page-header">

{% block header %}{% endblock %}

</div>

<div>

{% block content %}{% endblock %}

</div>

</div> <!-- /container -->

</body>

</html>

### 修改子页面

1. 创建home.html页面

{% extends "app/layout.html" %}

{% block header %}

<div class='jumbotron'>

<h1>Track your learning.</h1>

</div>

{% endblock %}

{% block content %}

<h2>

<a href="{% url 'users:register' %}">Register an account</a> to make

your own Learning Log, and list the topics you're learning about.

</h2>

<h2>

Whenever you learn something new about a topic, make an entry

summarizing what you've learned.

</h2>

{% endblock content %}

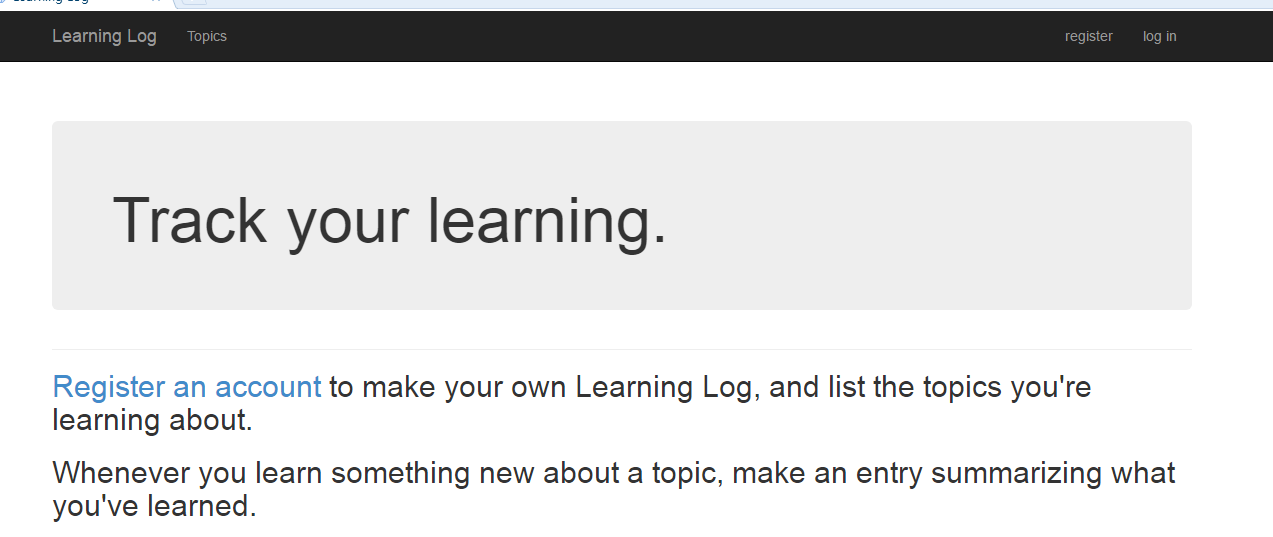
修改view：

def index(request): #定义当请求的url匹配时，如何显示并指定模板

""" index主页"""

return render(request,'app/home.html')

调试：



1. 修改登录页面样式

在users下创建新的登录页面login\_1.html：

{% extends "app/layout.html" %}

{% block header %}

<h2>Log in to your account.</h2>

{% endblock header %}

{% block content %}

<div class="row">

<div class="col-md-8">

<section id="loginForm">

<form action="." method="post" class="form-horizontal">

{% csrf\_token %}

<h4>Use a local account to log in.</h4>

<hr />

<div class="form-group">

<label for="id\_username" class="col-md-2 control-label">User name</label>

<div class="col-md-10">

**{{** form.username }}

</div>

</div>

<div class="form-group">

<label for="id\_password" class="col-md-2 control-label">Password</label>

<div class="col-md-10">

**{{** form.password }}

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="hidden" name="next" value="/" />

<input type="submit" value="Log in" class="btn btn-default" />

</div>

</div>

{% if form.errors %}

<p class="validation-summary-errors">Please enter a correct user name and password.</p>

{% endif %}

</form>

</section>

</div>

<div class="col-md-4">

<section id="socialLoginForm"></section>

</div>

</div>

{% load staticfiles %}

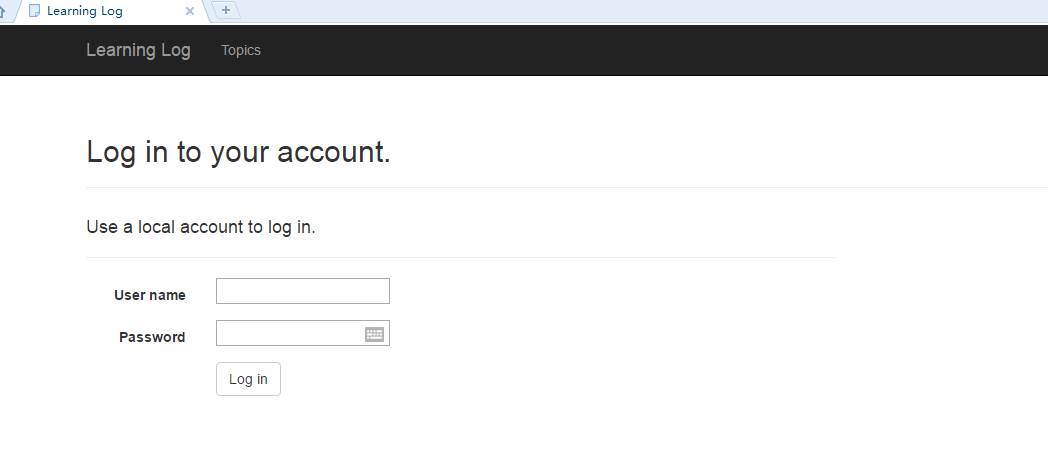
<script src="{% static 'app/scripts/jquery.validate.min.js' %}"></script>

{% endblock content %}

修改users下的urls.py

url(r'^login/$',login,{'template\_name':"users/login\_1.html"},name='login'),

调试：



1. 修改new\_topic页面样式

创建新new\_topic1.html页面

{% extends "app/layout.html" %}

{% block header %}

<h2>Add a new topic:</h2>

{% endblock header %}

{% block content %}

<form action="{% url 'new\_topic' %}" method="post" class="form">

{% csrf\_token %}

**{{** form.as\_p }}

<button name="submit" class="btn btn-primary">add topic</button>

</form>

{% endblock content %}

修改视图：

def new\_topic(request):

""" 添加新主题 """

if request.method != "POST":

#未提交数据时，创建一个新表单

form = TopicForm()

else:

#POST提交数据时，对数据进行处理

form = TopicForm(request.POST)

if form.is\_valid():

new\_topic = form.save(commit=False)

new\_topic.owner = request.user

new\_topic.save()

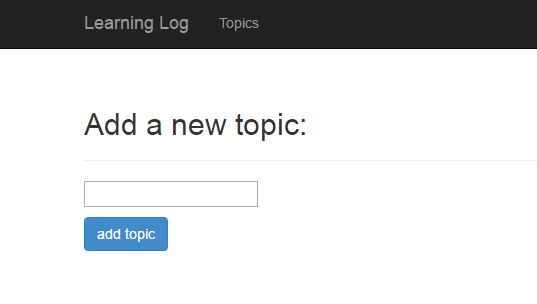
form.save()

return HttpResponseRedirect(reverse('topics')) #使用reverse()获取页面topics的URL

context = {'form':form}

return render(request,'app/new\_topic1.html',context)

调试：



1. 修改topics页面样式

创建新的topics1.html：

{% extends "app/layout.html" %}

{% block header %}

<h1>Topics</h1>

{% endblock %}

{% block content %}

<ul>

{% for topic in topics %}

<li>

<h3>

<a href="{% url 'topic' topic.id %}">**{{** topic }}</a>

</h3>

</li>

{% empty %}

<li>No topics have been added yet.</li>

{% endfor %}

</ul>

<h3><a href="{% url 'new\_topic' %}">Add new topic</h3>

{% endblock content %}

修改视图：

@login\_required #添加修饰器后将先运行该代码，检查用户是否登录，当登录时才运行后来的代码，未登录时重新定向

def topics(request):

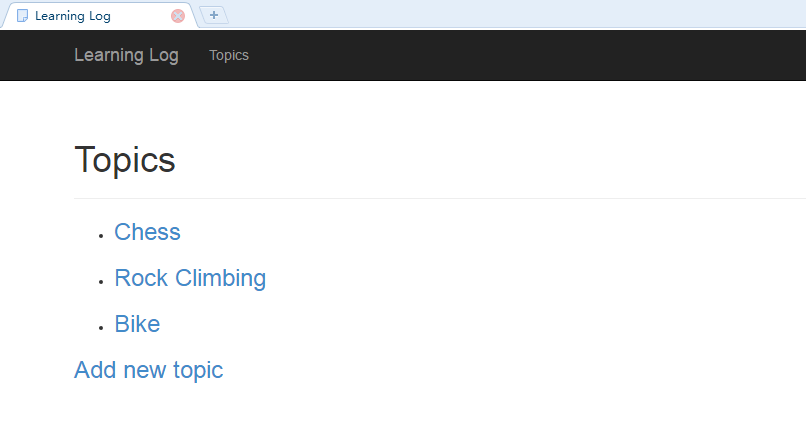
""" 显示所有的主题 """

topics = Topic.objects.filter(owner=request.user).order\_by('date\_added') #只允许从数据库中提取当前用户的Topic对象

context = {'topics':topics}

return render(request,'app/topics1.html',context) #将context传递给render

调试：



1. 修改topic页面中的条目样式

创建新的topic1.html：

{% extends "app/layout.html" %}

{% block header %}

<h2>**{{** topic }}</h2>

{% endblock header %}

{% block content %}

<p>

<a href="{% url 'new\_entry' topic.id %}">add new entry</a>

</p>

{% for entry in entries %}

<div class="panel panel-default">

<div class="panel-heading">

<h3>

**{{** entry.date\_added|date:'M d, Y H:i' }}

<small>

<a href="{% url 'edit\_entry' entry.id %}">

edit entry</a>

</small>

</h3>

</div>

<div class="panel-body">

**{{** entry.text|linebreaks }}

</div>

</div> <!-- panel -->

{% empty %}

There are no entries for this topic yet.

{% endfor %}

{% endblock content %}

修改路由：

def topic(request,topic\_id):

""" 显示单个主题及所有条目 """

topic = Topic.objects.get(id=topic\_id)

if topic.owner !=request.user: #如果当前用户与主题用户不同的，返回错误页面

raise Http404

entries = topic.entry\_set.order\_by('-date\_added')

context = {'topic':topic,'entries':entries}

return render(request,'app/topic1.html',context)

调试：



# End