Automate the boring stuff with python

目录

1 Part 1 python编程基础

1.1 python编程基础

项目我放在projects/AutomaticPython里了。

1.1.1 字符串连接与复制

python的字符串连接使用+号即可。eg: 'Alice'+'Bob'='AliceBob'。
python的字符串可以与整型值相乘。eg: 'Alice'*2='AliceAlice'。
使用#进行注释。
使用'''和"""进行多行注释。

```
eg:
"""
Write a function named collatz() that has one parameter named number. If
number is even, then collatz() should print number // 2 and return this value.
If number is odd, then collatz() should print and return 3 * number + 1.
"""
```

1.1.2 str(),len(),float(),int(),input()函数

使用input()函数等待用户在键盘上输入一些文本，并按回车键。
len(str)返回字符串str的长度。
str(),float(),int()会将参数转化为相应的数据类型。

1.2 控制流

1.2.1 布尔值

在python中，整型与浮点数的值永远不会与字符串相等。
python可以使用not操作符翻转布尔值。eg: not True = False
在其他数据类型中的某些值，条件会认为他们等价于false和true。
在用于条件时，0，0.0以及''(空字符串)被认为是false。其他则是true。

1.2.2 控制流语句

python的条件与循环语句如下：

```
if a<1 :
    ...
elif a=1:
    ...
else:
    ...

while b<1:
    ...
    if b=1:
        break
    elif b>1:
        continue
    ...
for i in range(5):
    ...
```

1.2.3 range()函数

range()函数也可以有第三个参数。前两个参数分别是起始值与终止值，第三个参数则是步长。

range(0,8,2)=0，2，4，6，8，负数也可以作为步长。

1.2.4 导入模块

在python中开始使用一个模块中的函数前，必须要用import语句导入该模块。
eg: import random
如果你不小心将一个程序命名为random. py，那么在import时程序将导入你的random. py文件，而不是random模块。
import语句的另一种形式包含from关键字。eg: from random import *。

1.2.5 调用sys.exit()提前终止程序

调用sys.exit()可以提前终止程序。

1.3 程序

1.3.1 函数定义

python函数定义形式如下：

```
def hello(name):
    print('hello'+name)
    return name
```

1.3.2 python的None值

python中的"null"是None。

1.3.3 局部和全局作用域

在Python中让局部变量与全局变量同名是可以的。
如果想要在一个函数内修改全局变量的值，就必须对变量使用global语句。示例最终的输出结果是hello。

1.3.4 异常处理

python的错误处理如下：

```
def hello(name):
    global eggs
    eggs='hello'
eggs='global'
hello()
print(eggs)

def divide(a):
    try:
        return 42/a
    except ZeroDivisionError:
        print('error')
```

一旦执行跳到except子句的代码，就无法回到try语句。它会继续向下运行。

1.4 列表

1.4.1 索引

列表是一个值。包含由多个值构成的序列。
eg:spam=['hello','aaa'] spam[0]='hello'
spam变量仍然只被赋予一个值，但列表值本身包含有多个值。

列表中也可以包含列表。
```
eg: spam = [['aa'],[10,20]]
    spam[0][0]='aa'
```

1.4.2 负数索引

列表可以使用负数索引。-1是列表中最后一个索引。-2是倒数第二个，以此类推。
eg:spam=['hello','aaa']
spam[-1]='aaa'
.

1.4.3 利用切片取得子列表

切片可以从列表中获取多个值。
eg:spam=[1, 2, 3, 4]
spam[0:4]=[1,2,3,4]
spam[0:-1]=[1,2,3]
spam[1:3]=[2,3]

作为快捷方法，你可以省略冒号两边的一个索引或两个索引。
省略第一个索引相当于使用使用索引0或从列表的开始处开始。
省略第二个索引相当于使用列表的长度，意味着切片直至列表末尾。
eg:spam=[1, 2, 3, 4]
spam[:]=[1,2,3,4]
spam[:2]=[1,2]

1.4.4 用len()函数可以获取列表长度。

用len()函数可以获取列表长度。
eg:spam=[1, 2, 3, 4] len(spam)=4

1.4.5 列表复制和列表连接

列表可以复制和连接，就像字符串一样。
eg:spam=[1, 2, 3, 4]
spam+[5]=[1,2,3,4,5]
[1,2]*2=[1,2,1,2]
spam*=3也是可以的

1.4.6 del语句从列表中删除值

del语句将删除列表中索引的值。
eg:spam=[1, 2, 3, 4]
del spam[0]
spam=[2,3,4]

1.4.7 列表循环

列表也可以用于循环

```
eg:spam=[1, 2, 3, 4]
for i in range(len(span)):
    print(spam[i])
```

1.4.8 in 和 not in

使用in和not in操作符，可以确定一个值是否在列表中。

```
eg:spam=[1, 2, 3, 4]
5 in spam == False
1 in spam == True
```

1.4.9 多重赋值技巧

列表可以使用多重赋值的技巧。

```
eg:spam=[1, 2, 3, 4]
4, 5, 6, 7=spam
spam=[4,5,6,7]
```

1.4.10 enumerate()函数与列表

如果在For循环中不使用range(len(spam))来获取列表中各项的索引，我们可以调用enumerate()函数。其会返回两个值：表项本身和索引。

```
eg:spam=[1, 2, 3, 4]
for item,index in enumerate(spam):
    print(item+index)
```

1.4.11 random中的函数配合列表

random.choice()可以在列表中返回一个随机表项。
random.shuffle()将会改变列表的排序。

每种数据类型都有一些它们自己的方法。

1.4.12 列表方法：index(),insert(),append(),remove(),sort(),reverse()

```
eg:spam=[1, 2, 3, 4]
   spam.index(5)==0 查找值，spam不存在为5的值。

   spam.insert(1,5)在指定索引处加入
   spam=[1,5,2,3,4]

   spam.append(6)在尾部加入
   spam=[1,5,2,3,4,6]

   spam.remove(5)
   spam=[1,2,3,4,6]

   删除列表中不存在的值将导致ValueError错误。
   如果该值在列表中出现多次，则只有第一次出现的值会被删除。
   如果知道索引,用del语句删除就可以了。
```

包含数值的列表或字符串的列表可以用sort()方法排序。
也可以指定reverse关键字参数为True。

```
spam=[1,2,3,4,6]
spam.sort(reverse=True)
spam=[6,4,3,2,1]
```

sort()方法对字符串排序时使用的是ASCII字符顺序，而非字典序。
而要用普通的字典序，则需要将sort()的key设置为str.lower。这将导致sort()方法将列表中所有表项当作小写。

```
spam=['a','z','c']
spam.sort(str.lower)
spam=['a','c','z']
```

```
spam=[1,2,3]
spam.reverse()快速翻转顺序
spam=[3,2,1]
```

random.randint(0,len(spam)-1) 可以在0-len(spam)-1之间随机产生一个数。

字符串中的字符也可以使用序列的方式访问。
但是字符串是不可变数据类型，序列的数据类型是可变的。

```
eggs=[1,2]
eggs=[3,4]
```

在这过程中，eggs的列表值没有改变，只是新的列表值覆盖了原来的列表值。
如果想要确实修改原理的列表，你需要del原来的元素，再用append()加上新的元素。
这样变量的值就并没有被一个新的列表值取代。

元组类似于不可更改的序列，你通过元组表明你并不打算更改它的值。
eggs=(1,2,3)。
tuple()可以将列表转变为元组，list()是其的逆序。

1.4.13 python引用机制

python的引用机制：

```
spam=[1,2]
cheese=spam
spam[1]=1
cheese=spam=[1,1]
```

spam和cheese指向同一个变量

1.4.14 标识和id函数

python中所有值都具有一个唯一标识，我们可以通过id()来获取。
修改对象不会改变标识，覆盖对象会。
python的自动垃圾收集器GC会删除任何变量未引用的值。

1.4.15 传递引用和copy函数

函数的变元得到的是引用的复制，改变会影响到原来的值。
如果在参数传入函数时不希望影响到原来的值，我们可以使用copy模块处理。如果要复制的列表中包含了列表，那就使用copy.deepcopy()函数来代替，此函数将同时复制它们内部的列表。

```
import copy
cheese=copy.copy(spam)
id(cheese)!=id(spam) 但两者内容相同
```

1.5 字典和结构化数据

1.5.1 字典数据类型

像列表一样，字典是许多值的集合。但不像列表的索引，字典的索引可以使用许多不同的数据类型，不只是整数。字典的索引被称之为key,这是一个key-value形的数据结构。

```
myCat = {'size': 'fat', 'color': 'gray', 'disposition': 'loud'}
myCat['size']='fat'
```

1.5.2 字典与列表

字典中的项是无序的，字典中没有"第一个"项。但是如果你在它们中创建序列值，字典将记住其key-value对的插入顺序。

用in关键字可以查看变量是否作为key存在于字典中。

```
eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
eggs == ham -->True
'name' in eggs == True
```

1.5.3 key(),items(),value()

有3个字典方法，它们将返回类似列表的方法，分别对应字典的key，value和key-value对。

```
spam = {'color': 'red', 'age': 42}
for v in spam.values():
    print(v) 'red',42
for k in spam.keys():
    print(k)  'color','age'
for k in spam.items():
    print(k)   ('color', 'red'),('age', 42)
```

尝试访问字典中不存在的键会出现KeyError。

list(spam.keys())会直接返回一个由spam的key组成的列表。

1.5.4 检查字典中是否存在键或值以及setdefault()方法

in与not in在字典中的应用：

```
spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys()
True
>>> 'Zophie' in spam.values()
True
>>> 'color' in spam.keys()
False
>>> 'color' not in spam.keys()
```

```
True
>>> 'color' in spam
False
```

get方法的应用。
spam.get('name',0)
如果为'name'的key存在，则返回其的value。否则返回默认值0。

为某个key设置一个默认值，当key没有任何value时使用默认值的方法为setdefault()。
传递给该方法的第一个参数是要检查的key,第二个参数是当此key不存在时要设置的value。

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

'color'的值没有被改为white,因为spam已经有名为'color'的键了。

1.5.5 美观地输出

如果程序导入了pprint(pretty-print)模块，我们就可以使用pprint()和pformat()函数，它们将美观地输出一个字典的字。pprint()会按键的排序输出。

若要将其化为相应的字符串，那么使用pformat()即可，下面两行语句是等价的：

```
pprint.pprint(someDictionaryValue)
print(pprint.pformat(someDictionaryValue))
```

1.5.6 嵌套的字典和列表

字典也可以包含其他字典。

```
allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
             'Bob': {'ham␣sandwiches': 3, 'apples': 2},
             'Carol': {'cups': 3, 'apple␣pies': 1}}
```

1.6 字符串操作

1.6.1 字符串字面量及转义

字符串可以以单/双引号开始和结束。

在字符串中插入的转义字符如下：

表格 1. 转义字符

\'	单引号
\"	双引号
\t	制表符
\n	换行符
\\	倒斜杠

可以在字符串前加入r使其成为原始字符串。原始字符串完全忽略所有的转义字符，可输出字符中所有的倒斜杠。

```
eg:print(r'aaa\'a') --> aaa\'a
```

python认为\是斜杠的一部分，而不是转义字符的开始。

多行字符串用3个单引号或双引号包围。三重引号之间的所有引号，制表符或是换行符，都会被认为是字符串的一部分。此时例如单引号就无需转义。

1.6.2 字符串索引和切片

字符串中的字符也可以使用索引来访问。开始为0。
如果用一个索引以及另一个索引指定范围，则开始索引将被包含，结束索引则不包含。

```
eg:spam='Hello,world' spam[0:5]='Hello'
```

1.6.3 字符串的in 和 not in

in和not in操作符也可以运用于字符串。区分大小写。

```
eg: 'Hello' in 'Hello,world' == True
    'HelLo' in 'Hello' == False
```

1.6.4 将字符串放入其他字符串

虽然加号可以实现字符串的插入和连接，但我们可以使用更方便的方法。
使用字符串插值法。其中字符串中的%s运算符会充当标记，并由字符串后的值代替。
好处是无需调用str()便可将值转化为字符串。

```
eg: 'My␣name␣is␣%s.␣I␣am␣%s␣years␣old' % (name,age)
```

python3.6引入了"f字符串"，该字符串与字符串插值类似，不同之处在于用花括号代替%s，并将表达式直接放在花括号内。

```
eg: name='shulva'
    age=15
    f'my␣name␣is␣{name},␣I␣am␣{age}␣years␣old'
```

1.6.5 upper(),lower(),isupper(),islower()以及isX()方法

upper()和lower()字符串方法返回一个新字符串，其中原字符串的所有字母都被相应地转换为大写或小写。这些方法不会改变字符串本身，而是返回一个新字符串。
如果字符串中含有字母，并且所有字母都是小写和大写，那么isupper()和islower()方法就会相应的返回True,否则返回False。

表格 2. 字符串的is()方法

isalpha()	如果字符串只包含字母且非空，返回True
isalnum()	如果字符串只包含数字和字母且非空，返回True
isdecimal()	如果字符串只包含数字字符且非空，返回True
isspace()	如果字符串只包含空格，制表符和换行符，返回True
istitle()	如果字符串仅包含以大写字母开头，后面都是小写字母的单词，数字或空格，返回True

1.6.6 startwith(),endwith()

如果startwith()和endwith()方法所调用的字符串以该方法传入的字符串开始和结束，则返回True，否则返回False。

1.6.7 join()和split()

如果有一个字符串列表，需要将他们连起来成为一个字符串，那么join()方法就很有用。join()方法可在字符串上被调用，参数是一个字符串列表，返回一个字符串。

```
eg: 'ABC'.join(['My', 'name', 'is', 'Simon'])=='MyABCnameABCisABCSimon'
```

split()方法所做的事正好相反，它针对一个字符串值调用，返回一个字符串列表。

```
eg: 'My␣name␣is␣Simon'.split()==['My', 'name', 'is', 'Simon']
```

默认情况下其按照各种空白字符分隔。也可向split()方法传入一个分隔字符串，指定其按照不同的字符串分隔。\n也是可以的。

```
eg: 'MyABCnameABCisABCSimon'.split('ABC')==['My', 'name', 'is', 'Simon']
    'My␣name␣is␣Simon'.split('m')==['My␣na', 'e␣is␣Si', 'on']
```

1.6.8 partition()

partition()字符串方法可以将字符串分成分隔符字符串前后的文本。 其会返回三个子字符串的元组。 如果分隔符字符串多次出现，其只取第一次出现处。 如果找不到字符串，则返回元组中第一个字符串将是整个字符串，而其他两个字符串为空。

```
eg:'Hello,␣world!'.partition('w')
   ('Hello,␣', 'w', 'orld!')
   'Hello,␣world!'.partition('world')
   ('Hello,␣', 'world', '!')
```

可以利用多重赋值技巧给3个字符串赋值。

```
eg:>>> before, sep, after = 'Hello,␣world!'.partition('␣')
   >>> before   'Hello,'   >>> after  'world!'
```

1.6.9 rjust(),ljust(),center()

rjust()和ljust()返回调用他们的字符串的填充版本，通过插入空格来对齐文本。

```
eg:
 >>> 'Hello'.ljust(10) 左对齐将Hello放在长为10的字符串中
 'Hello␣␣␣␣␣'
 >>> 'Hello'.rjust(10) 右对齐将Hello放在长为10的字符串中
 '␣␣␣␣␣Hello'
 >>> 'Hello'.rjust(20, '*')
 '***************Hello'
 >>> 'Hello'.ljust(20, '-')
 'Hello---------------'
 >>> 'Hello'.center(20, '=')
 '=======Hello========'
```

利用rjust()，ljust()和center()方法确保字符串对齐，即使你不清楚字符串有多少字符。

1.6.10 strip(),rstrip(),lstrip()

strip()方法将返回一个新的字符串，将调用其的字符串中的开头与末尾的空白字符删除。
lstrip()，rstrip()删除左边和右边的空白字符。

```
eg:
 >>> spam = '␣␣␣␣Hello,␣World␣␣␣␣'
 >>> spam.strip()
 'Hello,␣World'
 >>> spam.lstrip()
 'Hello,␣World␣␣␣␣'
 >>> spam.rstrip()
 '␣␣␣␣Hello,␣World'

 strip()方法可带有一个可选的字符串参数，指定两边的哪些字符串应该删除。
 >>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
 >>> spam.strip('ampS')#删除出现的a,m,p,S 字符顺序并不重要。
```

```
'BaconSpamEggs'
```

1.6.11 使用ord()和chr()获取字符数值

可以使用ord()函数获取一个单字符字符串的unicode代码点[1]。用chr()函数获取一个整数代码点的单字符字符串。

eg: `>>> ord('!') == 33  >>> chr(65) == 'A'`

1.6.12 用pyperclip模块复制粘贴字符串

可以安装第三方模块pyperclip中的copy()和paste()函数来向计算机的剪贴板发送文本或是从它接收文本。

eg:

```
>>> import pyperclip
>>> pyperclip.copy('Hello, world!')
>>> pyperclip.paste()
'Hello, world!'
```

python的命令行参数将存储在变量sys. argv中。 sys. argv变量的列表中的第一个项总是一个字符串，它包含程序的文件名。第二项应该是第一个命令行参数。

2 自动化任务

2.1 模式匹配与正则表达式

2.1.1 python正则表达式对象创建及匹配

\d表示一位数字字符。\d{3}表示匹配此模式3次。
python中所有正则表达式的函数都在re模块中。

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> print('Phone number found: ' + mo.group())
Phone number found: 415-555-4242
```

我们使用re.compile()将期待的模式传入对象，之后在此对象中调用search()，向其传入想查找的字符串。其将查找后的返回的结果Match放入mo中。我们在mo上调用方法group(),返回匹配的结果。

出现groups is not a known member of None这个报错的原因是因为regex的匹配结果可能是None，要先对regex的返回结果(在这里是mo)进行非空判断后再作操作。

2.1.2 利用括号分组

将一些特殊的，希望匹配到的东西分离出来时，我们便可以使用括号进行分组。

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('My number is 415-555-4242.')
>>> mo.group(1)#第一组括号
```

1. Unicode 代码点（Unicode code point）是指一个抽象的概念，表示 Unicode 字符集中的每一个字符所对应的唯一编号。Unicode 码点的取值范围是 0x0000 到 0x10FFFF，共 1,114,112 个码点。其中，0x0000 到 0xFFFF 之间的码点可以使用两个字节的 UTF-16 编码表示，而 0x10000 到 0x10FFFF 之间的码点需要使用四个字节的 UTF-16 编码表示，或者使用 UTF-8 或 UTF-32 编码表示。

```
'415'
>>> mo.group(2)#第二组括号
'555-4242'

>>> areaCode, mainNumber = mo.groups()#返回全部的分组
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

re.escape(pattern)是一个可以对字符串中所有可能被解释为正则运算符的字符进行转义的应用函数。

类似于. ^ $ * + ? { } [ ] \ | ( )的特殊符号，在匹配其本身时需要在其前面加上\。不想一个个处理就用escape函数。re.escape(regex)即可返回对应好的字符串。

2.1.3 使用|符号

希望匹配多个模式中的一个时，便可使用|符号。

```
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')

>>> mo = batRegex.search('Batmobile␣lost␣a␣wheel')
>>> mo.group()
'Batmobile'
>>> mo.group(1)
'mobile'
```

2.1.4 使用?实现可选匹配

使用?表明其前面的分组在此模式中是可选的，也即是无论此文本存在与否都会匹配。
你也可以认为?是匹配之前的分组0次或1次。

```
>>> batRegex = re.compile(r'Bat(wo)?man')

>>> mo1 = batRegex.search('The␣Adventures␣of␣Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The␣Adventures␣of␣Batwoman')
>>> mo2.group()
'Batwoman'
```

2.1.5 使用*号匹配0次或多次

*号意味着匹配分组0次或多次。

```
>>> batRegex = re.compile(r'Bat(wo)*man')

>>> mo1 = batRegex.search('The␣Adventures␣of␣Batman')
>>> mo1.group()
'Batman'

>>> mo3 = batRegex.search('The␣Adventures␣of␣Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

2.1.6 使用加号匹配一次或多次

+号意味着匹配分组1次或多次。意味着分组必须出现。

```
>>> batRegex = re.compile(r'Bat(wo)+man')

>>> mo1 = batRegex.search('The␣Adventures␣of␣Batwoman')
>>> mo1.group()
'Batwoman'

>>> mo3 = batRegex.search('The␣Adventures␣of␣Batman')
>>> mo3 == None
True
```

2.1.7 用花括号匹配特定次数

h{3}匹配3次h，即匹配hhh。
h{3,5}匹配3-4次h，即hhh和hhhh。
h{,3}匹配0-3次的h。
h{0,}匹配0-无穷的h。

2.1.8 贪心与非贪心匹配

贪心意味着匹配在匹配多个结果时会选择最长的字符串。非贪心则会匹配最短的字符串。
只需在模式结束后的地方跟着一个?即可使用非贪心模式。一般默认为贪心模式。

```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')

>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

2.1.9 findall()方法

在一个没有分组的正则表达式上调用,其将返回一个匹配字符串的列表。
如果在一个有分组的正则表达式上调用， 其将返回一个字符串的元组的列表， 每个分组对应一个字符串。

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # has no groups
>>> phoneNumRegex.findall('Cell:␣415-555-9999␣Work:␣212-555-0000')
['415-555-9999', '212-555-0000']

>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') # has groups
>>> phoneNumRegex.findall('Cell:␣415-555-9999␣Work:␣212-555-0000')
[('415', '555', '9999'), ('212', '555', '0000')]
```

2.1.10 字符分类(形如\d)

表格 3. \d—\S

\d	0-9中的任何数字
\D	除0-9以外的任何字符
\w	任何字母，数字或下划线字符(可以立即为单词字符)
\W	除字母，数字，下划线以外的任何字符
\s	空格，制表符，换行符
\S	除空格，制表符，换行符以外的任何字符
[0-5]	0-5的数字
[a-zA-Z]	所有大写小写字母

2.1.11 建立自己的字符分类

[a-zA-Z0-9]匹配所有大写小写字母及数字。
在字符分类左方括号上方后加上一个插入字符^，便可得到非字符类(补集)。

```
>>> consonantRegex = re.compile(r'[^aeiouAEIOU]')#匹配所有非元音字符
>>> consonantRegex.findall('RoboCop␣eats␣baby␣food.␣BABY␣FOOD.')
['R', 'b', 'C', 'p', '␣', 't', 's', '␣', 'b', 'b', 'y', '␣', 'f', 'd', '.', '␣
','␣','B',␣'B',␣'Y',␣' ',␣'F',␣'D',␣'.']
```

2.1.12 ^与$

在regex的开始处使用^表明匹配必须发生在被查找文本开始处。
在末尾加上$表明该字符串必须以此regex的模式结束。
同时使用表明整个字符串必须匹配该模式，只匹配该字符串的某个子集是不够的。

```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello,␣world!')
<re.Match object; span=(0, 5), match='Hello'>
beginsWithHello.search('He␣said␣hello.') == None

>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your␣number␣is␣42')
<re.Match object; span=(16, 17), match='2'>
endsWithNumber.search('Your␣number␣is␣forty␣two.') == None

>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<re.Match object; span=(0, 10), match='1234567890'>
wholeStringIsNum.search('12345xyz67890') == None
```

2.1.13 通配符.

.可匹配除了换行符之外的所有字符。
可以用(.*)表示任意文本。
传入re.DOTALL作为re.compile()的第二个参数，可以让通配符匹配所有字符。

```
>>> newlineRegex = re.compile('.*', re.DOTALL)
```

```
>>> newlineRegex.search('Serve␣the␣public␣trust.\nProtect␣the␣innocent.␣
\nUphold␣the␣law.').group()
'Serve␣the␣public␣trust.\nProtect␣the␣innocent.\nUphold␣the␣law.'
#原本只会匹配到Serve the public trust.
```

2.1.14 不区分大小写

只需将re.IGNORECASE或re.I作为re.compile()的第二个参数即可。

```
>>> robocop = re.compile(r'robocop', re.I)
>>> robocop.search('RoboCop␣is␣part␣man,␣part␣machine,␣all␣cop.').group()
'RoboCop'
```

2.1.15 使用sub()方法替换字符串

regex对象的sub()方法需要传入两个参数，第一个参数是一个字符串，用于替换掉发现的匹配。第二个字符串是一个需要被替换文本的字符串。

```
>>> namesRegex = re.compile(r'Agent␣\w+')
>>> namesRegex.sub('CENSORED',
'Agent␣Alice␣gave␣the␣secret␣documents␣to␣Agent␣Bob.')
'CENSORED␣gave␣the␣secret␣documents␣to␣CENSORED.'
```

有时候需要使用匹配的文本作为替换文本的一部分，在sub()方法的第一个参数中，可以输入\1，\2,\3...表示在替换中输入分组1，2，3的文本。

```
>>> agentNamesRegex = re.compile(r'Agent␣(\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent␣Alice␣told␣Agent␣Carol␣that␣Agent␣
Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

2.1.16 管理复杂正则表达式

使用'''创建多行字符串分隔处理即可。
可以使用re.VERBOSE作为第二个参数从而忽略空白符和注释。

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?            # area code
    (\s|-|\.)?                    # separator
    \d{3}                         # first 3 digits
    (\s|-|\.)                     # separator
    \d{4}                         # last 4 digits
    (\s*(ext|x|ext.)\s*\d{2,5})?  # extension
    )''',␣re.VERBOSE)

someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL | re.VERBOSE)
#组合使用这些参数
```

2.2 输入验证

暂时用不上，不看。

2.3 读写文件

2.3.1 Path()函数

win和macos上是不区分文件名大小写的，linux区分。
windows使用\来分隔路径，Linux和macos则使用/。
用pathlib模块中的Path()函数可以有效地返回对应的操作系统的路径，包括对应的分隔符。

```
>>> from pathlib import Path
>>> Path('spam', 'bacon', 'eggs')
WindowsPath('spam/bacon/eggs')

>>> str(Path('spam', 'bacon', 'eggs'))
'spam\\bacon\\eggs'
```

使用 from pathlib import Path可以避免每次使用Path()函数时都输入pathlib.Path，
从而直接使用Path()
如果是Linux系统则结果是spam/bacon/eggs

Path()函数也可以连接文件名从而产生路径。

```
>>> from pathlib import Path
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
>>> for filename in myFiles:
print(Path(r'C:\Users\Al', filename))
C:\Users\Al\accounts.txt
C:\Users\Al\details.csv
C:\Users\Al\invite.docx
```

2.3.2 使用/运算符连接路径

将/运算符和Path对象一起使用，连接路径既容易又安全，因为用Path可以无视操作系统产生的分隔符的区别。/运算符运算路径时，参与运算的值必须有一个是Path对象。

```
>>> Path('spam') / 'bacon' / 'eggs'
WindowsPath('spam/bacon/eggs')
>>> Path('spam') / Path('bacon/eggs')
WindowsPath('spam/bacon/eggs')
>>> Path('spam') / Path('bacon', 'eggs')
WindowsPath('spam/bacon/eggs')

>>> homeFolder = Path('C:/Users/Al')
>>> subFolder = Path('spam')
>>> homeFolder / subFolder
WindowsPath('C:/Users/Al/spam')
>>> str(homeFolder / subFolder)
'C:\\Users\\Al\\spam'
```

2.3.3 当前工作目录

利用Path.cwd()函数，可以取得当前工作路径的字符串。
使用os.chdir()可以改变当前工作目录。

```
>>> from pathlib import Path
>>> import os
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')'
>>> os.chdir('C:\\Windows\\System32')
>>> Path.cwd()
WindowsPath('C:/Windows/System32')
```

2.3.4 主目录

使用Path.home()可以获取此操作系统的主文件夹。

2.3.5 绝对路径与相对路径

绝对路径总是从根文件夹开始。
相对路径则是相对于当前的工作目录。

2.3.6 用os.makedirs()创建新文件夹

```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')

>>> from pathlib import Path
>>> Path(r'C:\Users\Al\spam').mkdir()
```

其会创建路径上的所有文件夹。
Path()方法也可以创建文件夹，但其不会创建路径上的所有文件夹。mkdir()一次只创建一个文件夹。

2.3.7 处理绝对和相对路径

Path对象的is_absolute()方法可以检测路径是否是绝对路径。

```
>>> Path.cwd().is_absolute()
True
>>> Path('spam/bacon/eggs').is_absolute()
False

使用cwd()和自己指定的相对路径组合
>>> Path.cwd() / Path('my/relative/path')
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37/my/relative/
path')

>>> os.path.abspath('.')#abspath返回参数的绝对路径的字符串
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37'
>>> os.path.abspath('.\\Scripts')
'C:\\Users\\Al\\AppData\\Local\\Programs\\Python\\Python37\\Scripts'

>>> os.path.isabs('.')#判断是否是绝对路径
```

```
False
>>> os.path.isabs(os.path.abspath('.'))
True

#os.path.relpath(path, start) 返回从start路径到path的的相对路径的字符串
如果没有提供开始路径，则默认cwd为开始路径
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
```

2.3.8 取得文件路径的各个部分

给定一个Path对象，可以利用其的属性将文件路径的不同部分提取为字符串。

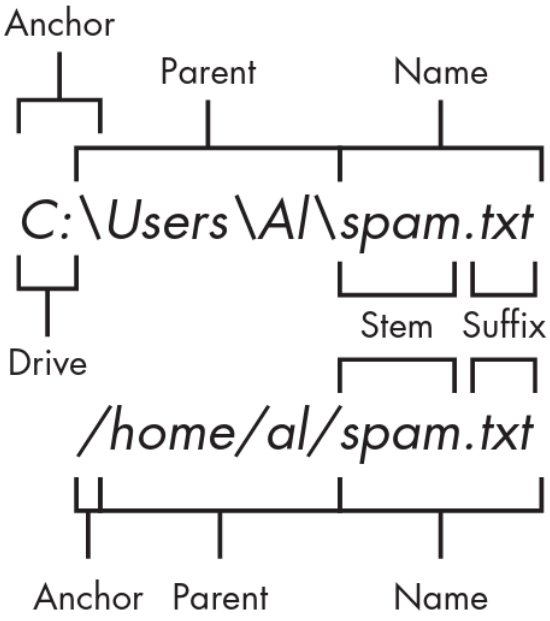

Figure 9-4: The parts of a Windows (top) and macOS/Linux (bottom) file path

```
>>> p = Path('C:/Users/Al/spam.txt')
>>> p.anchor
'C:\\'
>>> p.parent # This is a Path object, not a string.
WindowsPath('C:/Users/Al')
>>> p.name
'spam.txt'
>>> p.stem
'spam'
>>> p.suffix
'.txt'
>>> p.drive
'C:'
```

parents属性求值为一组Path对象，代表祖先文件夹，具有整数索引

```
>>> Path.cwd()
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python/Python37')
>>> Path.cwd().parents[0]
WindowsPath('C:/Users/Al/AppData/Local/Programs/Python')
>>> Path.cwd().parents[1]
WindowsPath('C:/Users/Al/AppData/Local/Programs')
>>> Path.cwd().parents[2]
WindowsPath('C:/Users/Al/AppData/Local')
>>> Path.cwd().parents[3]
WindowsPath('C:/Users/Al/AppData')
>>> Path.cwd().parents[4]
WindowsPath('C:/Users/Al')
>>> Path.cwd().parents[5]
WindowsPath('C:/Users')
>>> Path.cwd().parents[6]
WindowsPath('C:/')
```

较老的os.path模块中有类似的函数，用于取得一个写在一个字符串值中的路径的不同部分。可以取得一个路径中的目录名称与其的基本名称。

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'

>>> os.path.basename(calcFilePath)
'calc.exe'
>>> os.path.dirname(calcFilePath)
'C:\\Windows\\System32'

>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'

>>> os.path.split(calcFilePath)
('C:\\Windows\\System32', 'calc.exe')

>>> (os.path.dirname(calcFilePath), os.path.basename(calcFilePath))
('C:\\Windows\\System32', 'calc.exe')

os.sep会被设置为正确的操作系统的目录分隔符
>>> calcFilePath.split(os.sep)
['C:', 'Windows', 'System32', 'calc.exe']

>>> '/usr/bin'.split(os.sep)
['', 'usr', 'bin']
```

2.3.9 查看文件大小和文件夹内容

os.path.getsize(path)将返回path参数中文件的字节数。
os.listdir(path)将返回文件名字符串的列表。

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
27648
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax', 'aaclient.dll',--snip-
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW', 'zipfldr.dll']
```

如果想计算此目录下所有文件的总字节数，就同时使用这两个函数循环即可。

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
totalSize = totalSize + os.path.getsize(os.path.join('C:\\Windows\\System32',
filename))
>>> print(totalSize)
2559970473
```

2.3.10 使用通配符模式处理文件列表

如果处理特定文件，那么使用glob方法比listdir()更好。

```
>>> p = Path('C:/Users/Al/Desktop')
>>> list(p.glob('*.?x?')
[WindowsPath('C:/Users/Al/Desktop/calc.exe'), WindowsPath('C:/Users/Al/
Desktop/foo.txt')
```

2.3.11 检查路径的有效性

Path对象有一些方法可以检查给定的路径是否存在。
如何路径存在，p.exists()将返回True,反之为False。
如果路径存在且是一个文件，p.is_file()将返回True,反之为False。
如果路径存在且是一个文件夹，p.is_dir()将返回True,反之为False。

```
>>> winDir = Path('C:/Windows')
>>> notExistsDir = Path('C:/This/Folder/Does/Not/Exist')
>>> calcFile = Path('C:/Windows/System32/calc.exe')

>>> winDir.exists()
True
>>> winDir.is_dir()
True
>>> notExistsDir.exists()
False
>>> calcFile.is_file()
True
>>> calcFile.is_dir()
False
```

2.3.12 文件读写过程

read_text()方法会返回文件全部的字符串。
write_text()方法利用传递给它的字符串创建一个新的文本文件(或覆盖现有文件)。

```
>>> from pathlib import Path
>>> p = Path('spam.txt')
>>> p.write_text('Hello, world!')
13
>>> p.read_text()
'Hello, world!'
```

2.3.13 打开，创建，读写文件

open可以用来直接创建文件，只要文件名不存在于目录中即可。需要在第二个参数上使用'w'。

使用open打开文件时，默认是只读模式。调用open()将返回一个File对象。
当你需要读取或写入该文件时，将可以调用File对象的方法。
在读取和写入文件后调用close()方法才能再次打开该文件。

如果你将文件中的内容看作一个大字符串，那么read()方法将会返回此文件中的这个字符串。
也可以使用readlines()方法，其会以换行符为分隔返回字符串的列表。

写入时你需要以写入模式或是添加模式打开文件。

```
>>> helloFile = open('/Users/your_home_folder/hello.txt')
>>> helloContent = helloFile.read()
>>> helloContent
'Hello, world!'

>>> sonnetFile = open(Path.home() / 'sonnet29.txt')
>>> sonnetFile.readlines()
[When, in disgrace with fortune and men's eyes,\n', ' I all alone beweep my
outcast state,\n']

>>> baconFile = open('bacon.txt', 'w')# write模式
>>> baconFile.write('Hello, world!\n')
>>> baconFile.close()

>>> baconFile = open('bacon.txt', 'a')# append模式
>>> baconFile.write('Bacon is not a vegetable.')
>>> baconFile.close()

>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)

Hello, world!
Bacon is not a vegetable.
```

2.3.14 用shelve模式保存变量

利用shelve模块，你可以将python程序中的变量保存到二进制的shelf文件中。
如同字典一样，shelf值有keys()和value()，需要配合list()使用以获取真正可用的列表。

```
>>> import shelve
>>> shelfFile = shelve.open('mydata') # 创建mydata的二进制文件
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()

# shelf值无需用读或写模式打开，其在打开后自动又能写又能读
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
['cats']
>>> list(shelfFile.values())
[['Zophie', 'Pooka', 'Simon']]
>>> shelfFile.close()
```

2.3.15 用pprint.pformat()函数保存变量

感觉不太实用。
将变量储存在.py文件中以便在其他地方引入文件后调用。但是只有基本数据类型，如整型，浮点型，字符串，列表和字典可以作为简单文本写入文件。例如，File对象就不能编码为文本。

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc':
'fluffy'}]
>>> pprint.pformat(cats)
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name': 'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()

>>> import myCats
>>> myCats.cats
[{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc': 'fluffy'}]
>>> myCats.cats[0]
{'name': 'Zophie', 'desc': 'chubby'}
>>> myCats.cats[0]['name']
'Zophie'
```

2.4 组织文件

2.4.1 复制文件和文件夹

调用shutil.copy(source,destination)，其会将路径source处的文件复制到路径destination处的文件夹中。如果destination是一个文件名，那么其会被作为被复制文件的新名字。
该函数会返回一个字符串表示被复制文件的路径。

```
>>> import shutil, os
>>> from pathlib import Path
>>> p = Path.home()
>>> shutil.copy(p / 'spam.txt', p / 'some_folder')# dest为文件夹
'C:\\Users\\Al\\some_folder\\spam.txt'
>>> shutil.copy(p / 'eggs.txt', p / 'some_folder/eggs2.txt') # dest为文件
WindowsPath('C:/Users/Al/some_folder/eggs2.txt')
```

shutil.copytree()则会复制整个文件夹以及它包含的文件夹和文件。
shutil .copytree(source, destination)将source处的文件夹复制到路径destination处。
该函数返回一个字符串，为新复制的文件夹的路径。

```
>>> import shutil, os
>>> from pathlib import Path
```

```
>>> p = Path.home()
>>> shutil.copytree(p / 'spam', p / 'spam_backup')
WindowsPath('C:/Users/Al/spam_backup')
```

2.4.2 文件和文件夹的移动与重命名

调用shutil.move(source,destination)，将路径source处的文件夹移动到路径destination，并返回新路径的字符串。构成目的地路径的各级文件夹必须存在。
如果dest为文件夹，则文件移动到dest中。其将覆盖dest文件夹中的同名文件。
如果dest为文件，则将文件移动到指定路径后改名为dest指定的文件名。

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'

>>> shutil.move('C:\\bacon.txt', 'C:\\eggs\\new_bacon.txt')
'C:\\eggs\\new_bacon.txt'

>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')#如果指定的eggs文件夹不存在
'C:\\eggs' #其会在C下变成名为eggs的文件，要注意不犯这样的错误
```

2.4.3 永久删除文件和文件夹

调用os.unlink(path)将删除path处的文件。
调用os.rmdir(path)将删除path处的文件夹。该文件夹必须为空。
调用shutil.rmtree(path)将删除path处的文件夹，其包含的所有文件和文件夹都会被删除。

```
import os
from pathlib import Path
for filename in Path.home().glob('*.rxt'):
    #os.unlink(filename)
    print(filename)
```

你可以用send2trash模块进行安全删除。s2t会将删除的文件放入回收站，上述的则会永久删除。

2.4.4 遍历目录树

如果你希望遍历目录树，并处理遇到的每个文件，os.walk()函数便是不二之选。其按树形结构向下遍历，每次迭代更换底下的三个变量。

```
import os
for folderName, subfolders, filenames in os.walk('C:\\delicious'):
    print('The␣current␣folder␣is␣' + folderName)
    for subfolder in subfolders:
        print('SUBFOLDER␣OF␣' + folderName + ':␣' + subfolder)
    for filename in filenames:
        print('FILE␣INSIDE␣' + folderName + ':␣'+ filename)
    print('')

    The current folder is C:\delicious
    SUBFOLDER OF C:\delicious: cats
    SUBFOLDER OF C:\delicious: walnut
    FILE INSIDE C:\delicious: spam.txt
```

```
The current folder is C:\delicious\cats
FILE INSIDE C:\delicious\cats: catnames.txt
FILE INSIDE C:\delicious\cats: zophie.jpg
The current folder is C:\delicious\walnut
SUBFOLDER OF C:\delicious\walnut: waffles

The current folder is C:\delicious\walnut\waffles
FILE INSIDE C:\delicious\walnut\waffles: butter.txt.
```

os.walk()函数被传入一个字符串值，即一个文件夹的路径。
os.walk()返回字符串的列表，将其保存在subfolder和filename变量中。
os.walk()在循环的每次迭代中返回以下三个值：

当前文件夹的路径的字符串—folderName。
当前文件夹中子文件夹的名称的字符串的列表—subfolders。
当前文件夹中文件的名称的字符串的列表—filenames。

程序的当前目录不会因为os.walk()而改变。

2.4.5 用zipfile模块处理压缩文件—读取zip文件

要读取ZIP文件的内容，首先必须创建一个ZipFile对象。
创建时你需要调用zipfile.ZipFile()函数，向其传递一个字符串表示zip文件的文件名。

```
>>> import zipfile, os
>>> from pathlib import Path
>>> p = Path.home()
>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
>>> spamInfo.compress_size
3828
u >>> f'Compressed␣file␣is␣{round(spamInfo.file_size␣/␣spamInfo
.compress_size, 2)}x smaller!'
)
'Compressed␣file␣is␣3.63x␣smaller!'
>>> exampleZip.close()
```

Zipfile对象有一个namelist()方法，它返回Zip文件中包含的所有文件和文件夹的字符串的列表。
这些字符串可以传递给ZipFile对象的getinfo()方法，返回一个关于特定文件的ZipInfo对象。
ZipInfo对象有一些自己的属性，如上面的compress_size和file_size。分别表示压缩后大小和原文件大小。

2.4.6 从zip文件中解压缩

ZipFile对象的extractall()方法从ZIP文件中解压缩所有文件和文件夹。

```
>>> import zipfile, os
>>> from pathlib import Path
>>> p = Path.home()

>>> exampleZip = zipfile.ZipFile(p / 'example.zip')
```

```
>>> exampleZip.extractall()
>>> exampleZip.close()

# ZipFile对象的extract()方法解压缩单个文件
>>> exampleZip.extract('spam.txt')
'C:\\spam.txt'
>>> exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
'C:\\some\\new\\folders\\spam.txt'
>>> exampleZip.close()
# 传递给extract()的字符串，必须匹配namelist()返回的字符串列表中的一个。
# 若向其传递第二个参数，则其会将文件解压缩到指定文件夹。不存在则创建文件夹。
```

2.4.7 创建和添加到zip文件

要创建自己的zip文件，则需要以写模式打开ZipFile对象，即传入'w'作为第二个参数。
如果向write()方法传入一个路径，那么此路径的文件便会被压缩。write()的第二个参数决定了计算机以什么算法来压缩。

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

其将创建新的压缩文件new.zip。如果想要添加而不是覆写原有的zip文件，则需要将'a'作为第二个参数，以添加模式打开zip文件。

2.5 调试

2.5.1 抛出异常

当python试图执行无效代码时，便会抛出异常。
我们使用try和except语句来处理python的异常。

抛出异常使用rasie()语句。在代码中，raise()包含以下部分：

- raise 关键字
- 传递给Exception()函数的字符串，包含有效的错误信息
- 对Exception函数的调用

```
def boxPrint(symbol, width, height):
    if len(symbol) != 1:
 u      raise Exception('Symbol␣must␣be␣a␣single␣character␣string.')

 for sym, w, h in (('*', 4, 4), ('O', 20, 5), ('x', 1, 3), ('ZZ', 3, 3)):
    try:
        boxPrint(sym, w, h)
 x  except Exception as err:
 y      print('An␣exception␣happened:␣' + str(err))
```

一旦输入条件不满足，参数不对，try-except就会处理无效的参数。

这个程序使用了except语句的except Exception as err 形式。
如果boxPrint()返回一个Exception对象， 那么这条except语句便会将其保存在err变量中。Exception对象可以传递给str()以将它转换为一个字符串。

使用try-except语句可以更优雅的处理错误，而不是让整个程序崩溃。

2.5.2 取得回溯字符串

如果Python遇到错误，它就会产生一些错误信息，称为Traceback。
Traceback包含了错误信息，错误代码行号以及导致错误的函数调用序列。这个序列称为调用栈。

```
def spam():
    bacon()
def bacon():
    raise Exception('This␣is␣the␣error␣message.')
spam()

Traceback (most recent call last):
 File "errorExample.py", line 7, in <module>
   spam()
 File "errorExample.py", line 2, in spam
   bacon()
 File "errorExample.py", line 5, in bacon
   raise Exception('This␣is␣the␣error␣message.')
Exception: This is the error message.
```

只要抛出的异常没有处理，python就会显示回溯。你也可以调用traceback.format_exc()得到其的字符串形式。

2.5.3 断言

断言是健全性检查，用于确保代码中没有明显错误的事情。有了断言后请不要再用if-print来显示错误了。
assert的逻辑是：我断言条件成立。若不成立，则说明某个地方有bug，请立即停止程序。

```
assert 'red' in stoplight.values(), 'Neither␣light␣is␣red!␣' + str(stoplight)
```

有了这个断言，程序就会崩溃，并提供这样的信息：

```
  Traceback (most recent call last):
  File "carSim.py", line 14, in <module>
    switchLights(market_2nd)
  File "carSim.py", line 13, in switchLights
    assert 'red' in stoplight.values(), 'Neither␣light␣is␣red!␣' +
str(stoplight)
 u AssertionError: Neither light is red! {'ns': 'yellow', 'ew': 'green'}
```

虽然程序崩溃并不如你所愿，但他马上指出了健全性检查失败。
在程序中尽早失败，可以省去将来的大部分调式工作。

2.5.4 日志-logging模块

要启用Logging模块，请将下面代码复制到程序顶部。

```
import logging
logging.basicConfig(level=logging.DEBUG, format='␣%(asctime)s␣-␣␣%(levelname)
s -  %(message)s')
```

```
def factorial(n):
    logging.debug('Start␣of␣factorial(%s%%)'  % (n))
    total = 1
    for i in range(n + 1):
        total *= i
        logging.debug('i␣is␣' + str(i) + ',␣total␣is␣' + str(total))
    logging.debug('End␣of␣factorial(%s%%)'  % (n))
    return total
print(factorial(5))
logging.debug('End␣of␣program')
```

当python记录一个事件的日志时，它会创建一个LogRecord对象已保存关于该事件的信息。
logging模块的函数让你指定想看的的LogRecord对象的细节以及细节的展示方式。

我们在输出日志消息时，使用了logging.debug()函数。这个debug()将调用basicConifg()以输出一行信息，这行信息的格式是我们在basicConfig()函数中指定好的。

```
2019-05-23 16:20:12,664 - DEBUG - Start of program
2019-05-23 16:20:12,664 - DEBUG - Start of factorial(5)
2019-05-23 16:20:12,665 - DEBUG - i is 0, total is 0
2019-05-23 16:20:12,668 - DEBUG - i is 1, total is 0
2019-05-23 16:20:12,670 - DEBUG - i is 2, total is 0
2019-05-23 16:20:12,673 - DEBUG - i is 3, total is 0
2019-05-23 16:20:12,675 - DEBUG - i is 4, total is 0
2019-05-23 16:20:12,678 - DEBUG - i is 5, total is 0
2019-05-23 16:20:12,680 - DEBUG - End of factorial(5)
0
2019-05-23 16:20:12,684 - DEBUG - End of program
```

2.5.5 不要使用print()调试—日志级别

删除代码中散落的Print很麻烦的。日志消息是给程序员看的，而不是给用户的。
日志级别提供了一种方式:按重要性对日志进行分类。

表格 4. 日志等级

级别	日志函数	描述
DEBUG	logging.debug()	最低级，用于调式小细节，debug时才会用
INFO	logging.info()	用于记录程序中一般事件的信息，或确认一切正常
WARNING	logging.warning()	用于表示可能的问题，其当前不会阻止程序，但未来可能会
ERROR	logging.error()	用于记录错误，他导致程序做某事失败
CRITICAL	logging.critical()	最高级别，它导致程序运行失败

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG, format='␣%(asctime)s␣-␣␣
%(levelname)s -  %(message)s')

>>> logging.debug('Some␣debugging␣details.')
2019-05-18 19:04:26,901 - DEBUG - Some debugging details.

>>> logging.info('The␣logging␣module␣is␣working.')
2019-05-18 19:04:35,569 - INFO - The logging module is working.

>>> logging.warning('An␣error␣message␣is␣about␣to␣be␣logged.')
2019-05-18 19:04:56,843 - WARNING - An error message is about to be logged.
```

```
>>> logging.error('An␣error␣has␣occurred.')
2019-05-18 19:05:07,737 - ERROR - An error has occurred.

>>> logging.critical('The␣program␣is␣unable␣to␣recover!')
2019-05-18 19:05:45,794 - CRITICAL - The program is unable to recover!
```

如果你只对ERROR及其以上感兴趣，则可以将basicConfig()的level参数设置为logging.ERROR。这将只显示ERROR和CRITICAL信息，跳过低级别的三个信息。

2.5.6 禁用日志及将日志记录到文件

使用logging.disable()即可禁用。只要传入一个日志级别，其就会禁止该级别及更低级别的所有日志。想要禁用所有日志则直接使用logging.disable(logging.CRITICAL)即可。

将日志写入文本文件只需在logging.basicConfig中添加filename关键字参数即可。

```
logging.basicConfig(filename='myProgramLog.txt', level=logging.DEBUG,
format='␣%(asctime)s␣-␣␣%(levelname)s␣-␣␣%(message)s')
```

其会将日志保存在myProgramLog.txt中。

2.6 从web抓取信息

2.6.1 webbrowser.open()

将你想打开的网页的url传入open()参数即可。

2.6.2 用requests.get()函数下载一个网页

requests模块是第三方模块。

用requests. get()函数下载一个网页， 接受一个要下载的URL字符串。 在get()的返回值上调用type()，可以看出它返回一个Response对象。

response对象是有一个status_code属性的，可以检查其是否等于requests.code.ok,从而了解下载是否成功。

检查成功有一种简单的方法，那就是在response对象上调用raise_for_status()方法。如果下载失败便抛出异常，成功便什么都不做。

一般来说在get()之后会调用raise_for_status()，以确保会让程序继续运行。

2.6.3 将下载的文件保存到硬盘

将web页面保存到硬盘中时，必须要用写二进制打开文件。即向open()传入'wb'作为第二参数，目的是保存该文本中的unicode编码。

为了将web页面写入一个文件，可以使用for循环与iter_content()方法。

```
>>> import requests
>>> res = requests.get('https://automatetheboringstuff.com/files/rj.txt')
>>> res.raise_for_status()
>>> playFile = open('RomeoAndJuliet.txt', 'wb')
>>> for chunk in res.iter_content(100000):
        playFile.write(chunk)
```

iter_content()方法在循环的每次迭代中返回一段内容，每一段都是Bytes数据类型，你需要指定每一段包含多少字节。

write()方法返回一个数字，表示写入文件的字节数。在前面的例子中，第一段 包含100000个字节，文件剩下的部分只需要78981个字节。

2.6.4 HTML

不要用正则表达式来解析HTML。HTML的格式可以有许多不同的方式，并且仍然被认为是有效的HTML，但尝试用正则表达式来捕捉所有这些可能的变化将非常繁琐，并且容易出错。
html - RegEx match open tags except XHTML self-contained tags - Stack Overflow

Beautiful Soup 是一个模块，用于从HTML页面中提取信息。用于这个目的时，它比正则表达式好很多。
BeautifulSoup()函数需要一个字符串，其中包含将要解析的HTML文件。bs4.BeautifulSoup()函数返回一个BeautifulSoup对象。第二个参数用来告诉其使用哪一个解析器来分析HTML。

```
>>> import requests, bs4
>>> res = requests.get('http://nostarch.com')
>>> res.raise_for_status()
>>> noStarchSoup = bs4.BeautifulSoup(res.text,'html.parser')␣
>>> type(noStarchSoup)
<class 'bs4.BeautifulSoup'>

>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile,'html.parser')␣
>>> type(exampleSoup)
<class 'bs4.BeautifulSoup'>
```

这段代码利用requests.get()函数从 No Starch Press 网站下载主页，然后将响应结果的text 属性传递给bs4.BeautifulSoup()。它返回的 BeautifulSoup 对象保存在变量noStarchSoup 中。

也可以向bs4.BeautifulSoup()传递一个 File 对象，从硬盘加载一个HTML文件。

我们可以使用select()方法来选择Html文件中的元素。不同的选择器模式可以组合起来，形成复杂的匹配。例如soup.select('p #author') 将匹配所有id属性为author的元素，只要它也在一个<p>元素之内。

表格 5. CSS选择器

传递给select()方法的选择器	将匹配...
soup.select('div')	所有名为div的元素
soup.select('#author')	带有id属性为author的元素
soup.select('.notice')	所有使用CSS class属性名为notice的元素
soup.select('div span')	所有在 元素之内的元素
soup.select('div > span')	所有直接在 元素之内的元素，中间无其他元素
soup.select('input[name]')	所有名为<input>，并有一个name属性，其值无所谓的元素
soup.select('input[type="button"]')	所有名为<input>，并有一个type属性，其值为button的元素

select()方法将返回一个Tag 对象的列表，这是Beautiful Soup表示一个HTML元素的方式。
针对BeautifulSoup 对象中的HTML的每次匹配，列表中都有一个Tag对象。

```
>>> pElems = exampleSoup.select('p')

>>> str(pElems[0])
'<p>Download␣my␣<strong>Python</strong>␣book␣from␣<a␣href="http://␣
inventwithpython.com">my␣website</a>.</p>'␣

>>> len(elems)
3
>>> pElems[0].getText()
'Download␣my␣Python␣book␣from␣my␣website.'
```

```
>>> str(pElems[1])
'<p class="slogan">Learn Python the easy way!</p>'

>>> pElems[1].getText()
'Learn Python the easy way!'

>>> str(pElems[2])
'<p>By <span id="author">Al Sweigart</span></p>'

>>> pElems[2].getText()
'By Al Sweigart'
```

len(elems)告诉我们列表中有一个Tag 对象，只有一次匹配。
在该元素上调用getText()方法，返回该元素的文本，或内部的HTML。
attrs()会返回一个和此属性相关的字典，一般用在有值的元素上。

Tag 对象的get()方法让我们很容易从元素中获取属性值。、

```
>>> str(spanElem)
'<span id="author">Al Sweigart</span>'
>>> spanElem.get('id')
'author'
```

2.6.5 用selenium模块控制浏览器

与Requests和Beautiful Soup相比，Selenium 允许你用高级得多的方式与网页交互。但因为它启动了Web浏览器，假如你只是想从网络上下载一些文件，会有点慢，并且难以在后台运行。
每个浏览器启动需要安装各自相关的selenium的driver。

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>> browser.get('http://inventwithpython.com')
```

在页面中寻找元素要使用WebDriver方法。
WebDriver对象有好几种方法，用于在页面中寻找元素。
它们被分成find_element_*和 find_elements_*方法。

find_element_*方法返回一个 WebElement 对象，代表页面中 匹配查询的第一个元素。
find_elements_*方法返回WebElement_*对象的列表，包含页面中所有匹配的元素。
调用By元素需要先from selenium.webdriver.common.by import By。

表格 6. selenium的 WebDriver 方法

方法名	返回的WebElement对象/列表
browser.find_element(s)(By.CLASS_NAME,name)	使用CSS类name的元素
browser.find_element(s)(By.CSS_SELECTOR,selector)	匹配CSS selector 的元素
browser.find_element(s)(By.ID,id)	匹配id属性值的元素
browser.find_element(s)(By.LINK_TEXT,text)	完全匹配提供的text的元素
browser.find_element(s)(By.PARTITAL_LINK_TEXT,text)	包含提供的text的元素
browser.find_element(s)(By.NAME,name)	匹配name属性值的元素
browser.find_element(s)(By.TAG_NAME,tag_name)	匹配标签name的元素 (大小写无关)
browser.find_element(s)(By.XPATH,xpath)	根据xpath进行定位，有用

如果页面上没有元素匹配该方法要查找的元素，selenium模块就会抛出NoSuchElement异常。
一旦有了WebElement对象，就可以读取表7中的属性，或调用其中的方法，了解它的更多功能。

表格 7. WebElement 的属性和方法

属性或方法	描述
tag_name	标签名，例如 'a'表示元素
get_attribute(name)	该元素name属性的值
text	该元素内的文本，例如hello中的'hello'
clear()	对于文本字段或文本区域元素，清除其中输入的文本
is_displayed()	如果该元素可见，返回True，否则返回False
is_enabled()	对于输入元素，如果该元素启用，返回True，否则返回False
is_selected()	对于复选框或单选框元素，如果该元素被选中，选择True，否则返回False
location	一个字典，包含键'x'和'y'，表示该元素在页面上的位置

```
>>> elem = browser.find_element(By.CLASS_NAME,'bookcover')
>>> print('Found␣<%s>␣element␣with␣that␣class␣name!' % (elem.tag_name))
Found <img> element with that class name!
```

WebElement 对象有一个 click()方法,可以模拟鼠标在该元素上点击。

```
>>> linkElem = browser.find_element(s)(By.LINK_TEXT,'Read␣It␣Online')
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click()
# follows the "Read␣It␣Online" link
```

也可以使用WebElement对象填写并提交表单。

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://gmail.com')
>>> emailElem = browser.find_element(By.ID,'Email')
>>> emailElem.send_keys('not_my_real_email@gmail.com')
>>> passwordElem = browser.find_element(By.ID,'Passwd')
>>> passwordElem.send_keys('12345')
>>> passwordElem.submit()
```

也可以使用WebElement对象发送特殊按键。
selenium 有一个模块，针对不能用字符串值输入的键盘击键。它的功能非常类似于转义字符。
这些值保存在selenium.webdriver.common.keys模块的属性中。
由于这个模块名非常长，所以在程序顶部运行from selenium. webdriver. common. keys import Keys 就比较容易。
如果这么做，原来需要写from selenium.webdriver.common.keys 的地方，就只要写Keys。

表格 8. selenium.webdriver.common.keys 模块中常用的变量

属性	含义
Keys.DOWN, Keys.UP, Keys.LEFT,Keys.RIGHT	键盘箭头键
Keys.ENTER, Keys.RETURN	回车和换行键
Keys.HOME, Keys.END	Home 键、End键
Keys.PAGE_DOWN,Keys.PAGE_UP	PageUp键和Page Down 键
Keys.ESCAPE, Keys.BACK_SPACE,Keys.DELETE	Esc、Backspace 和字母键
Keys.F1, Keys.F2, . . . , Keys.F12	键盘顶部的F1到F12键
Keys.TAB	Tab 键

例如，如果光标当前不在文本字段中，按下home和end键，将使浏览器滚动到页面的顶部或底部。

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> browser = webdriver.Firefox()
>>> browser.get('http://nostarch.com')
>>> htmlElem = browser.find_element(s)(By.TAG_NAME,'html')
>>> htmlElem.send_keys(Keys.END)
# scrolls to bottom
>>> htmlElem.send_keys(Keys.HOME)
# scrolls to top
```

利用以下的方法，selenium也可以模拟点击各种浏览器按钮：
browser.back()点击“返回”按钮。
browser.forward()点击“前进”按钮。
browser.refresh()点击“刷新”按钮。
browser.quit()点击“关闭窗口”按钮。

2.7 处理excel电子表格

请先安装openpyxl模块。

在导入openpyxl模块后，就可以使用openpyxl.load_workbook()函数。
openpyxl.load_workbook()函数接受文件名，返回一个workbook 数据类型的值。
这个workbook 对象代表这个Excel文件，有点类似File对象代表一个打开的文本文件。

2.7.1 从工作簿中取得工作表

调用get_sheet_names()方法可以取得工作簿中所有表名的列表。

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> wb.sheetnames
['Sheet1', 'Sheet2', 'Sheet3']

>>> sheet = wb['Sheet3']
>>> sheet
<Worksheet "Sheet3">

>>> sheet.title
'Sheet3'

>>> anotherSheet = wb.get_active_sheet()
>>> anotherSheet
<Worksheet "Sheet1">
```

最后，可以调用Workbook对象的get_active_sheet()方法，取得 工作簿的活动表。活动表是工作簿在Excel中打开时出现的工作表。

2.7.2 从表中取得单元格

有了Worksheet对象后，就可以按名字访问Cell单元格对象。

```
>>> import openpyxl

>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Sheet1']

>>> sheet['A1'].value
datetime.datetime(2015, 4, 5, 13, 34, 2)

>>> c = sheet['B1']
>>> c.value
'Apples'

>>> 'Row ' + str(c.row) + ', Column ' + c.column + ' is ' + c.value
'Row 1, Column B is Apples'
>>> 'Cell ' + c.coordinate + ' is ' + c.value
'Cell B1 is Apples'
>>> sheet['C1'].value
73
```

Cell对象有一个value属性，不出意外，它包含这个单元格中保存的值。
Cell对象也有row、column和coordinate 属性，提供该单元格的位置信息。
字母指定列，数字指定行。

在调用表的cell()方法时，可以传入整数作为row 和column 关键字参数。
也可以得到一个单元格。第一行或第一列的整数是1，不是0。

```
>>> sheet.cell(row=1, column=2)
<Cell Sheet1.B1>
>>> sheet.cell(row=1, column=2).value
'Apples'
```

可以通过Worksheet 对象的 get_highest_row()和 get_highest_column()方法，确定表的大小。

```
>>> sheet = wb['Sheet1']
>>> sheet.max_row
7
>>> sheet.max_column
3
```

2.7.3 列字母与数字的转换

要从字母转换到数字，就调用 openpyxl.cell.column_index_from_string()函数。 要从数字转换到字母，就调用openpyxl.cell.get_column_letter()函数。

```
>>> import openpyxl
>>> from openpyxl.utils import get_column_letter, column_index_from_string
>>> get_column_letter(27)
'AA'
>>> get_column_letter(2)
'B'
>>> get_column_letter(1)
'A'
>>> column_index_from_string('AA')
27
```

2.7.4 从表中取得行和列

可以将Worksheet 对象切片，取得电子表格中一行、一列或一个矩形区域中的所有 Cell 对象。然后可以循环遍历这个切片中的所有单元格。

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb['Sheet1']

>>> tuple(sheet['A1':'C3'])
((<Cell Sheet1.A1>, <Cell Sheet1.B1>, <Cell Sheet1.C1>), (<Cell Sheet1.A2>,
<Cell Sheet1.B2>, <Cell Sheet1.C2>), (<Cell Sheet1.A3>, <Cell Sheet1.B3>,
<Cell Sheet1.C3>))

for rowOfCellObjects in sheet['A1':'C3']:
    for cellObj in rowOfCellObjects:
        print(cellObj.coordinate, cellObj.value)
    print('--- END OF ROW ---')

A1 2015-04-05 13:34:02
B1 Apples
C1 73 --- END OF ROW ---
```

我们指明需要从A1到C3的矩形区域中的Cell对象。
要打印出这个区域中所有单元格的值，我们使用两个for循环。
外层for循环遍历这个切片中的每一行。
然后针对每一行，内层for循环遍历该行中的每个单元格。

要访问特定行或列的单元格的值，也可以利用Worksheet对象的rows 和columns 属性。

```
>>> sheet.columns[1]
(<Cell Sheet1.B1>, <Cell Sheet1.B2>, <Cell Sheet1.B3>, <Cell Sheet1.B4>,
<Cell Sheet1.B5>, <Cell Sheet1.B6>, <Cell Sheet1.B7>)

>>> for cellObj in sheet.columns[1]:
        print(cellObj.value)
```

2.7.5 写入和创建excel文档

利用create_sheet()以及remove_sheet()方法，可以在工作簿中添加或删除工作表。
调用openxyl.Workbook()方法创建一个新的空Workbook对象。

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.sheetnames
['Sheet']

>>> wb.create_sheet()
<Worksheet "Sheet1">
>>> wb.sheetnames
['Sheet', 'Sheet1']

>>> wb.create_sheet(index=0, title='First Sheet')
<Worksheet "First Sheet">
>>> wb.sheetnames
['First Sheet', 'Sheet', 'Sheet1']
```

```
>>> wb.remove_sheet(wb.get_sheet_by_name('Sheet1'))
>>> wb.sheetnames
['First Sheet', 'Sheet']

wb.save('excel.xslx')
```

在工作簿中添加或删除工作表之后，记得调用save()方法来保存变更。

将值写入单元格，很像将值写入字典中的键。

```
>>> sheet['A1'] = 'Hello world!'
>>> sheet['A1'].value
'Hello world!'
```

2.7.6 设置单元格的字体风格

为了定义单元格的字体风格，需要从openpyxl.styles 模块导入 Font()函数。

```
from openpyxl.styles import Font
```

这让你能输入Font()，代替 openpyxl.styles.Font()。

表格 9. Font style 属性的关键字参数

关键字参数	数据类型	描述
name	字符串	字体名称，诸如'Calibri' 或'Times New Roman'
size	整型	大小点数
bold	布尔型	True 表示粗体
italic	布尔型	True 表示斜体

```
>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> sheet['A1'].font = fontObj1
>>> sheet['A1'] = 'Bold Times New Roman'
```

2.7.7 公式

```
>>> sheet['B9'] = '=SUM(B1:B8)'
```

但是，如果你希望在表格中看到该公式的计算结果，而不是原来的公式。

那就必须将load_workbook()的data_only关键字参数设置为True。

```
>>> wbDataOnly = openpyxl.load_workbook('writeFormula.xlsx', data_only=True)
>>> sheet = wbDataOnly.get_active_sheet()
>>> sheet['A3'].value
500
```

2.7.8 调整行和列

Worksheet 对象有row_dimensions和column_dimensions 属性，控制行高和列宽。

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()

>>> sheet['A1'] = 'Tall row'
>>> sheet['B2'] = 'Wide column'
```

```
>>> sheet.row_dimensions[1].height = 70
>>> sheet.column_dimensions['B'].width = 20
>>> wb.save('dimensions.xlsx')
```

利用 merge_cells()工作表方法，可以将一个矩形区域中的单元格合并为一个单元格。

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()

>>> sheet.merge_cells('A1:D3')
>>> sheet['A1'] = 'Twelve␣cells␣merged␣together.'

>>> sheet.merge_cells('C5:D5')
>>> sheet['C5'] = 'Two␣merged␣cells.'

>>> wb.save('merged.xlsx')
```

要拆分单元格，就调用 unmerge_cells()工作表方法。

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('merged.xlsx')
>>> sheet = wb.get_active_sheet()

>>> sheet.unmerge_cells('A1:D3')
>>> sheet.unmerge_cells('C5:D5')
>>> wb.save('merged.xlsx')
```

2.8 处理google sheets

暂时没有用

2.9 处理word文档和pdf文件

暂时没有用

2.10 处理CSV文件和json数据

2.10.1 CSV模块

CSV 表示“Comma-Separated Values（逗号分隔的值）”，CSV 文件是简化的电子表格，保存为纯文本文件。

CSV文件也有自己的转义字符， 允许逗号和其他字符作为值的一部分。split()方法不能处理这些转义字符。因为这些潜在的缺陷，所以总是应该使用csv模块来读写CSV文件。

2.10.2 reader对象

要用csv模块从CSV文件中读取数据，需要创建一个Reader对象。Reader对象让你迭代遍历CSV文件中的每一行。

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> exampleData = list(exampleReader)
>>> exampleData
[['4/5/2015 13:34', 'Apples', '73'], ['4/5/2015 3:41', 'Cherries', '85'],
['4/6/2015 12:46', 'Pears', '14'], ['4/8/2015 8:59', 'Oranges', '52']
```

要用csv模块读取CSV文件，首先用open()函数打开它，就像打开任何其他文本文件一样。之后将它传递给csv.reader()函数即可。
要访问Reader对象中的值最直接的方法就是将它转换成一个普通Python列表，将它传递给list()。之后直接像二维数组一样访问list中的元素即可。

对于大型的CSV文件，你需要在一个for循环中使用Reader对象。这样避免将整个文件一次性装入内存。

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> for row in exampleReader:
print('Row #' + str(exampleReader.line_num) + ' ' + str(row))

Row #1 ['4/5/2015 13:34', 'Apples', '73']
Row #2 ['4/5/2015 3:41', 'Cherries', '85']
Row #3 ['4/6/2015 12:46', 'Pears', '14']
```

要取得行号，就使用Reader 对象的line_num变量，它包含了当前行的编号。 Reader 对象只能循环遍历一次。要再次读取 CSV 文件必须调用 csv.reader创建一个对象。

2.10.3 writer对象

Writer对象让你将数据写入CSV文件。要创建一个writer对象，就使用csv.writer()函数。

```
>>> import csv
>>> outputFile = open('output.csv', 'w', newline='')
>>> outputWriter = csv.writer(outputFile)
>>> outputWriter.writerow(['spam', 'eggs', 'bacon', 'ham'])
21
>>> outputWriter.writerow(['Hello, world!', 'eggs', 'bacon', 'ham'])
32
>>> outputWriter.writerow([1, 2, 3.141592, 4])
16
>>> outputFile.close()
```

首先调用open()并传入'w'，以写模式打开一个文件。
然后将它传递给csv.writer()，创建一个Writer对象。
在Windows 上需要为open()函数的newline关键字参数传入一个空字符串。
Writer 对象的writerow()方法接受一个列表参数。列表中的每个词放在输出的CSV文件中的一个单元格中。writerow()函数的返回值，是写入文件中这一行的字符数（包括换行字符）。

2.10.4 delimeter和lineminator关键字参数

假定你希望用制表符代替逗号来分隔单元格，并希望有两倍行距。可以在交互式环境中输入下面这样的代码：

```
>>> import csv
>>> csvFile = open('example.tsv', 'w', newline='')
>>> csvWriter = csv.writer(csvFile, delimiter='\t', lineterminator='\n\n')
>>> csvWriter.writerow(['apples', 'oranges', 'grapes'])
24
>>> csvWriter.writerow(['eggs', 'bacon', 'ham'])
17
>>> csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam', 'spam'])
32
>>> csvFile.close()
```

这改变了文件中的分隔符和行终止字符。默认情况下，行终止字符是换行符。
传入delimeter='\t'和 lineterminator='\n\n'会将单元格之间的字符改变为制表符，将行之间的字符改变为两个换行符。

2.10.5 DictReader和DictWriter CSV对象

对于包含标题行的CSV文件一般会使用这两个对象。
两者使用字典对CSV文件进行读写，且他们使用CSV文件的第一行作为字典的键。

```
>>> import csv
>>> exampleFile = open('exampleWithHeader.csv')
>>> exampleDictReader = csv.DictReader(exampleFile)
>>> for row in exampleDictReader:
...
print(row['Timestamp'], row['Fruit'], row['Quantity'])
...
4/5/2015 13:34 Apples 73
4/5/2015 3:41 Cherries 85
```

使用这两个对象意味这你在处理时无需使用额外代码跳过第一行的信息。
如果处理的csv文件没有第一行的标题，则需要在DictReader()函数中加入第二个参数，其中包含预置的列标题名。

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleDictReader = csv.DictReader(exampleFile, ['time', 'name',
'amount'])
>>> for row in exampleDictReader:
...
print(row['time'], row['name'], row['amount'])
...
4/5/2015 13:34 Apples 73
4/5/2015 3:41 Cherries 85
```

DictWriter对象使用字典来创建CSV文件。
如果你想要文件包含一个标题行，那就调用writeheader()来写入。

```
>>> import csv
>>> outputFile = open('output.csv', 'w', newline='')
>>> outputDictWriter = csv.DictWriter(outputFile, ['Name', 'Pet', 'Phone'])
>>> outputDictWriter.writeheader()
>>> outputDictWriter.writerow({'Name': 'Alice', 'Pet': 'cat', 'Phone': '555
1234'})
```

任何缺失的键在CSV文件中都会是空的。

2.10.6 json模块

Python的json模块有json.loads()和json.dumps()函数。
JSON不能存储每一种Python值。它只能包含以下数据类型的值：字符串、整型、浮点型、布尔型、列表、字典和NoneType。

要将包含JSON数据的字符串转换为Python的值，就将它传递给json.loads()函数。
这个名字的意思是“load string”。
请注意，JSON字符串总是用双引号。它将该数据返回为一个Python字典。

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true, "miceCaught": 0, "felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
```

json.dumps()函数（它表示“dump string”）将一个Python值转换成JSON格式的数据字符串。

```
>>> pythonValue = {'isCat': True, 'miceCaught': 0, 'name': 'Zophie', 'felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0, "name": "Zophie"}'
```

2.11 保持时间、计划任务、启动程序及多线程

2.11.1 time模块

Unix 纪元是编程中经常参考的时间：1970 年 1 月 1 日 0 点，即协调世界时 （UTC）。
time.time()函数返回自那一刻以来的秒数，是一个浮点数。这个数字称为 UNIX 纪元时间戳。

```
>>> import time
>>> time.time()
1425063955.068649
```

如果需要让程序暂停一下，就调用time.sleep()函数，并传入希望程序暂停的秒数。
round()函数可以处理浮点数的小数点后多少位保留的问题，是四舍五入的机制。

```
>>> import time
>>> now = time.time()
>>> now
1425064108.017826
>>> round(now, 2)
1425064108.02
>>> round(now, 4)
1425064108.0178
>>> round(now)
1425064108
```

2.11.2 datetime模块

datetime值表示一个特定的时刻。

```
>>> import datetime
>>> datetime.datetime.now()
datetime.datetime(2015, 2, 27, 11, 10, 49, 55, 53)
>>> dt = datetime.datetime(2015, 10, 21, 16, 29, 0)

>>> dt.year, dt.month, dt.day
(2015, 10, 21)
>>> dt.hour, dt.minute, dt.second
(16, 29, 0)
```

Unix 纪元时间戳可以通过datetime.datetime.fromtimestamp()，转换为datetime对象。

```
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2015, 2, 27, 11, 13, 0, 604980)
```

datetime对象可以用比较操作符进行比较，弄清楚时间前后关系。

```
>>> halloween2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
>>> newyears2016 = datetime.datetime(2016, 1, 1, 0, 0, 0)
>>> oct31_2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
>>> halloween2015 == oct31_2015
True
>>> halloween2015 > newyears2016
False
```

datetime模块还提供了timedelta数据类型，它表示一段时间，而不是一个时刻。

```
>>> delta = datetime.timedelta(days=11, hours=10, minutes=9, seconds=8)
>>> delta.days, delta.seconds, delta.microseconds
(11, 36548, 0)
>>> delta.total_seconds()
986948.0
>>> str(delta)
'11␣days,␣10:09:08'
```

要创建timedelta对象，就用datetime.timedelta()函数。datetime.timedelta()函数接受关键字参数weeks、days、hours、minutes、seconds、milliseconds和microseconds。 没有month和year关键字参数，因为“月”和“年”是可变的时间，依赖于特定月份或年份。

利用+和-运算符，timedelta对象与datetime对象或其他timedelta对象相加或相减。利用*和/运算符，timedelta对象可以乘以或除以整数或浮点数。

```
>>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
>>> aboutThirtyYears = datetime.timedelta(days=365 * 30)
>>> oct21st
datetime.datetime(2015, 10, 21, 16, 29)
>>> oct21st - (2 * aboutThirtyYears)
datetime.datetime(1955, 11, 5, 16, 29)
```

2.11.3 将datetime对象转换为字符串

利用strftime()方法，可以将datetime对象显示为字符串。

strftime()方法使用的指令类似于Python的字符串格式化。

表格 10.

strftime 指令	含义
%Y	带世纪的年份，例如'2014'
%y	不带世纪的年份，'00'至'99'（1970至2069）
%m	数字表示的月份, '01'至'12'
%B	完整的月份，例如'November'
%b	简写的月份，例如'Nov'
%d	一月中的第几天，'01'至'31'
%j	一年中的第几天，'001'至'366'
%w	一周中的第几天，'0'（周日）至'6'（周六）
%A	完整的周几，例如'Monday'
%a	简写的周几，例如'Mon'
%H	小时（24小时时钟），'00'至'23'
%I	小时（12小时时钟），'01'至'12'
%M	分，'00'至'59'
%S	秒，'00'至'59'
%p	'AM'或'PM'
%%	就是'%'字符

向strftime()传入一个定制的格式字符串，其中包含格式化指定（以及任何需要 的斜线、冒号等），strftime()将返回一个格式化的字符串，表示datetime对象的信息。

```
>>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
>>> oct21st.strftime('%Y/%m/%d␣%H:%M:%S')
'2015/10/21␣16:29:00'
>>> oct21st.strftime('%I:%M␣%p')
'04:29␣PM'
>>> oct21st.strftime("%B␣of␣'%y")
"October␣of␣'15"
```

如果有一个字符串的日期信息，如'2015/10/21 16:29:00'或'October 21, 2015'，需要将它转换为datetime对象，就用datetime.datetime.strftime()函数。感觉没啥用，太繁琐了。

```
>>> datetime.datetime.strptime('October␣21,␣2015', '%B␣%d,␣%Y')
datetime.datetime(2015, 10, 21, 0, 0)
>>> datetime.datetime.strptime('2015/10/21␣16:29:00', '%Y/%m/%d␣%H:%M:%S')
datetime.datetime(2015, 10, 21, 16, 29)
```

2.11.4 多线程

要得到单独的线程，首先要调用threading.Thread()函数，生成一个Thread对象。

```
import threading, time
print('Start␣of␣program.')

def takeANap():
    time.sleep(5)
    print('Wake␣up!')

threadObj = threading.Thread(target=takeANap)
threadObj.start()
print('End␣of␣program.')
```

如果想在新线程中运行的目标函数有参数，可以将目标函数的参数传入threading.Thread()。
常规参数可以作为一个列表，传递给threading.Thread()中的args关键字参数。关键字参数可以作为一个字典，传递给threading.Thread()中的kwargs关键字参数。

```
>>> import threading
>>> threadObj = threading.Thread(target=print, args=['Cats', 'Dogs', 'Frogs'],
kwargs={'sep': ' & '}) # print('Cats', 'Dogs', 'Frogs', sep=' & ')
>>> threadObj.start()
Cats & Dogs & Frogs
```

下面创建新线程调用print()的方法是不正确的：
`threadObj = threading.Thread(target=print('Cats', 'Dogs', 'Frogs', sep=' & '))`
其会将print()函数的返回值用作target函数的值。

2.11.5 从python启动其他程序。

利用内建的subprocess模块中的Popen()函数，Python程序可以启动计算机中的其他程序。

```
>>> import subprocess
>>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
<subprocess.Popen object at 0x0000000003055A58>
```

返回值是一个Popen对象，它有两个有用的方法：poll()和wait()。

可以认为poll()方法是问你的朋友，她是否执行完毕你给她的代码。如果这个 进程在poll()调用时仍在运行，poll()方法就返回None。
如果该程序已经终止，它会返回该进程的整数退出代码。退出代码用于说明进程是无错终止（退出代码为 0），还是一个错误导致进程终止（退出代码非零，通常为 1，但可能根据程序而不同）

wait()方法就像是等着你的朋友执行完她的代码，然后你继续执行你的代码。 wait()方法将阻塞，直到启动的进程终止。如果你希望你的程序暂停，直到用户完成或其他程序启动，这非常有用。wait()的返回值是进程的整数退出代码。

```
>>> calcProc = subprocess.Popen('c:\\Windows\\System32\\calc.exe')
>>> calcProc.poll() == None
True
>>> calcProc.wait() # 已关闭
0
>>> calcProc.poll()
0
```

用Popen()创建进程时可以向进程传递命令行参数。要做到这一点，需要向Popen() 传递一个列表作为唯一的参数。
实际上，这个列表将作为被启动程序的sys.argv的值。

```
>>> subprocess.Popen(['C:\\Windows\\notepad.exe', 'C:\\hello.txt'])
<subprocess.Popen object at 0x00000000032DCEB8>
```

这不仅会启动记事本应用程序，也会让它立即打开C:\hello.txt。
运行其他python脚本就可以直接调用python.exe，然后将要启动的python脚本路径当作参数传入即可。

2.11.6 用默认程序打开文件

根据操作系统，向Popen()传入'start'、'open'或'see'即可(对应win,mac和linux)。

```
>>> import subprocess
>>> subprocess.Popen(['start', 'hello.txt'], shell=True)
```

我们也传入了shell=True关键字参数，这只在Windows上需要。

2.12 发送电子邮件

2.12.1 使用gmail api发送和接收电子邮件

我们使用ezgmail这个第三方模块来控制gmail。
在windows上运行 pip install --user --upgrade ezgmail(macos和linux上使用pip3)来安装。

通过搜索gmail api python来跟着quickstart获取api的使用方法，之后获取credential.json文件并放到工作目录。我已放到AutomaticPython-18中。

之后在交互式环境中输入如下代码：

```
>>> import ezgmail,os
>>> os.chdir("your␣working␣directory␣including␣json")
>>> ezgmail.init()
```

确保将当前的工作目录设置为credentials.json所在的文件夹.
ezgmail.init()函数将打开你的浏览器，并进入一个Google登录页面。一路搞定，有问题请会看原书或谷歌。

这会生成一个token.json文件，它让你的Python脚本访问你输入的Gmail账户。有了credentials.json和token.json，你的Python脚本就可以从你的Gmail账户发送和读取电子邮件，而不需要你在源代码中包含Gmail口令。

2.12.2 从gmail账户发送邮件

一旦你有了token.json文件，ezgmail模块应该可以通过一个函数调用来发送邮件：

```
>>> import ezgmail
>>> ezgmail.send('recipient@example.com', 'Subject␣line', 'Body␣of␣the␣email')
```

如果想将文件附加到你的电子邮件中，可以为send()函数提供一个额外的列表参数：

```
>>> ezgmail.send('recipient@example.com', 'Subject␣line', 'Body␣of␣the␣email',
['attachment1.jpg',   'attachment2.mp3'])
```

请注意，作为安全和反垃圾邮件功能的一部分，Gmail可能不会发送文本完全相同的重复邮件（因为这些很可能是垃圾邮件），或包含.exe或.zip文件附件的邮件（因为它们很可能是病毒）。

你也可以提供可选的关键字参数cc和bcc来抄送和密件抄送：

```
>>> ezgmail.send('recipient@example.com', 'Subject␣line', 'Body␣of␣the␣email',
cc='friend@example.com',bcc='otherfriend@example.com,someoneelse@example.com')
```

如果你需要记住token.json文件是针对哪个Gmail地址，你可以检查ezgmail.EMAIL_ ADDRESS。注意，这个变量只有在ezgmail.init()或其他ezgmail函数被调用后才会被填充：

```
>>> import  ezgmail
>>> ezgmail.init()
>>> ezgmail.EMAIL_ADDRESS
'example@gmail.com'
```

请保管好token和credentials这两个json文件。

2.12.3 从gmail账户读取邮件

Gmail将相互回复的邮件组织成对话主线。当你在网页浏览器或通过应用程序登录Gmail时，你真正看到的是对话主线，而不是单个邮件（即使主线中只有一封邮件）。

ezgmail有GmailThread和GmailMessage对象，分别代表对话主线和单个邮件。

一个GmailThread对象有一个messages属性，它持有一个GmailMessage对象的列表。unread()函数返回一个GmailThread对象的列表，这个列表可以传递给ezgmail.summary()，输出该列表中的对话主线的摘要：

```
>>> import  ezgmail
>>> unreadThreads = ezgmail.unread() # List of GmailThread objects.
>>> ezgmail.summary(unreadThreads)
Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09
Jon - Thanks for stopping me from buying Bitcoin. - Dec 09
```

summary()函数可以方便地显示对话主线的快速摘要，但是要访问特定的邮件（和邮件的部分），你需要检查GmailThread对象的messages属性。

messages属性包含了一个 GmailMessage对象的列表，这些对象有subject（主题）、body（正文）、 timestamp（时间戳）、sender（发件人）和recipient（收件人）等描述邮件的属性。

```
>>> len(unreadThreads)
2
>>> str(unreadThreads[0])
"<GmailThread␣len=2␣snippet=␣Do␣you␣want␣to␣watch␣RoboCop␣this␣weekend?'>"
>>> len(unreadThreads[0].messages)
2
>>> str(unreadThreads[0].messages[0])
"<GmailMessage␣from='Al␣Sweigart␣<al@inventwithpython.com>'␣␣
to='Jon␣Doe␣
<example@gmail.com>'␣␣
timestamp=datetime.datetime(2018, 12, 9, 13, 28, 48)
subject='RoboCop' snippet='Do␣you␣want␣to␣watch␣RoboCop␣this␣weekend?'>"␣
>>> unreadThreads[0].messages[0].subject
'RoboCop'
>>> unreadThreads[0].messages[0].body
'Do␣you␣want␣to␣watch␣RoboCop␣this␣weekend?\r\n'
>>> unreadThreads[0].messages[0].timestamp
datetime.datetime(2018, 12, 9, 13, 28, 48)
>>> unreadThreads[0].messages[0].sender
'Al␣Sweigart␣<al@inventwithpython.com>'
>>> unreadThreads[0].messages[0].recipient
'Jon␣Doe␣<example@gmail.com>'
```

类似于ezgmail.unread()函数，ezgmail.relative()函数将返回Gmail账户中最近的25个主线。你可以传递一个可选的maxResults关键字参数来改变这个限制：

```
>>> recentThreads = ezgmail.recent()
>>> len(recentThreads)
25
>>> recentThreads = ezgmail.recent(maxResults=100)
>>> len(recentThreads)
46
```

2.12.4 从gamil账户中搜索邮件

除了使用ezgmail.unread()和ezgmail.recent()，你还可以通过调用 ezgmail. search()来搜索特定的邮件，就像你在邮箱的搜索框中查询一样：

```
>>> resultThreads = ezgmail.search('RoboCop')
>>> len(resultThreads)
1
>>> ezgmail.summary(resultThreads)
Al, Jon - Do you want to watch RoboCop this weekend? - Dec 09
```

和unread()和recent()一样，search()函数也会返回一个GmailThread对象的列表。
你也可以将任何一个可以输入搜索框中的特殊搜索操作符传递给search()函数，如下面这些：

from:al@inventwithpython.com用于来自al@inventwithpython.com 的邮件。
subject:hello用于主题中包含hello的邮件。
has:attachment用于有附件的邮件。
可以在Google的支持页面找到完整的搜索操作符列表。

2.12.5 从gmail账户下载邮件

GmailMessage对象有一个attachments属性，它是一个邮件附件文件名的列表。

可以将这些名称中的任何一个传递给GmailMessage对象的downloadAttachment()方法来下载文件。也可以用downloadAllAttachments()方法一次下载所有文件。
默认情况下，ezgmail会将附件保存到当前的工作目录中，
你也可以额外传递一个downloadFolder参数。
如果一个文件与附件的文件名相同，下载的附件会自动覆盖。

```
>>> import  ezgmail
>>> threads  =  ezgmail.search('vacation␣␣photos')
>>> threads[0].messages[0].attachments
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
>>> threads[0].messages[0].downloadAttachment('tulips.jpg')
>>>threads[0].messages[0].downloadAllAttachments(downloadFolder='vacat␣ion2019')
['tulips.jpg', 'canal.jpg', 'bicycles.jpg']
```

2.12.6 smtp

SMTP规定电子邮件应该如何格式化、加密、在邮件服务器之间传递，以及在你单击发送后，计算机要处理的所有其他细节。
SMTP只负责向别人发送电子邮件。另一个协议名为IMAP，负责取回发送给你的电子邮件。
我们使用python中的smtplib模块来处理相关事情。

SMTP服务器的域名通常是电子邮件提供商的域名前面加上SMTP。例如，Gmail的 SMTP 服务器是smtp.gmail.com。

```
>>> import smtplib
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
```

得到电子邮件提供商的域名和端口信息后，调用smtplib.SMTP()创建一个SMTP对象， 并传入域名作为字符串参数，传入端口作为整数参数。
如果smtplib.SMTP()调用不成功，你的SMTP服务器可能不支持TLS端口587。在这种情况下，你需要利用smtplib.SMTP_SSL()和465端口来创建SMTP对象。
对于你的程序，TLS和SSL之间的区别并不重要。只需要知道你的SMTP服务器使用哪种加 密标准，这样就知道如何连接它。

```
>>> smtpObj.ehlo()
```
得到SMTP对象后，调用它的ehlo()方法以向SMTP电子邮件服务器“打招呼”，从而测试与服务器之间的连接。

```
>>> smtpObj.starttls()
```
如果要连接到SMTP服务器的587端口（即使用TLS加密），那么接下来需要调用 starttls()方法。这是为连接实现加密必需的步骤。 如果要连接到465端口（使用SSL）， 加密已经设置好了， 你可以跳过这一步。

```
>>> smtpObj.login('my_email_address@example.com', 'MY_SECRET_PASSWORD')
```
到SMTP服务器的加密连接建立后，可以调用login()方法，用你的用户名（通常是你的 电子邮件地址）和电子邮件口令登录。调用Input()来输入密码可以保证安全性。

```
>>> smtpObj.sendmail('my_email_address@example.com', 'recipient@example.com',
'Subject:␣So␣long.\nDear␣Alice,␣so␣long␣and␣thanks␣for␣all␣the␣fish.␣
Sincerely,␣Bob')
```

登录到电子邮件提供商的SMTP服务器后，可以调用sendmail()方法来发送电子邮件。
sendmail()方法需要以下3个参数:你的电子邮件地址字符串（电子邮件的“from”地址）。 收件人的电子邮件地址字符串，或多个收件人的字符串列表（作为“to”地址）。 电子邮件正文字符串。
电子邮件正文字符串必须以'Subject: \n'开头，以作为电子邮件的主题行。'\n'换行符将主题行与电子邮件的正文分开。
sendmail()的返回值是一个字典。对于电子邮件传送失败的每个收件人，该字典中会有一个键-值对。空的字典意味着对已所有收件人成功发送电子邮件。

```
>>> smtpObj.quit()
```
确保在完成发送电子邮件时，调用quit()方法，这让程序从SMTP服务器断开。

2.12.7 连接到Imap服务器

Python带有一个 imaplib模块，但实际上第三方的imapclient模块更好用。
如同smtp，你也需要电子邮件服务提供商的IMAP服务器域名。

得到IMAP服务器域名后，调用imapclient.IMAPClient()函数来创建一个 IMAPClient对象。大多数电子邮件提供商要求SSL加密，传入ssl=True关键字参数。

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.example.com', ssl=True)
```

取得IMAPClient对象后，调用它的login()方法，传入用户名（这通常是你的电子邮件 地址）和口令字符串。

```
>>> imapObj.login('my_email_address@example.com',   'MY_SECRET_PASSWORD')
```

2.12.8 搜索电子邮件

登录后，实际获取你感兴趣的电子邮件分为两步。首先，必须选择要搜索的文件夹。然后，必须调用IMAPClient对象的search()方法，向其传入IMAP搜索关键词字符串。

几乎每个账户默认有一个INBOX文件夹，但也可以调用IMAPClient对象的 list_folders()方法来获取文件夹列表。这将返回一个元组的列表。每个元组包含一个文件夹的信息。

```
>>> import pprint
>>> pprint.pprint(imapObj.list_folders())
[(('\\HasNoChildren',), '/', 'Drafts'),
 (('\\HasNoChildren',), '/', 'Filler')]
```

每个元组的3个值，例如(('\\HasNoChildren',), '/', 'INBOX')，其解释如下。
该文件夹的标志的元组（这些标志代表到底是什么超出了本书的讨论范围，你可以放心地忽略该字段）。
名称字符串中用于分隔父文件夹和子文件夹的分隔符。
该文件夹的全名。

要选择一个文件夹进行搜索，就调用IMAPClient对象的select_folder()方法，传入 该文件夹的名称字符串。

```
>>> imapObj.select_folder('INBOX', readonly=True)
```

文件夹选中后，就可以用IMAPClient对象的search()方法搜索电子邮件。search()的参数是一个字符串列表。
在传入search()方法的列表参数中，可以有多个IMAP搜索键字符串。返回的消息将匹配所有的搜索键。如果想匹配任何一个搜索键，使用OR搜索键。对于NOT和OR搜索键，它们后边分别跟着一个和两个完整的搜索键。

imap搜索键的全部具体表格请google或是查询书籍的imap章节。
下面是search()方法调用的一些例子，以及它们的含义：

imapObj.search(['ALL'])返回当前选定的文件夹中的每一个消息。
imapObj.search(['ON 05-Jul-2019'])返回在2019年7月5日发送的每个消息。
imapObj.search(['SINCE 01-Jan-2019', 'BEFORE 01-Feb-2019', 'UNSEEN'])返回2019年1月发送的所有未读消息（注意，这意味着从1月1日直到2月1日，但 不包括2月1日）。
imapObj.search(['SINCE 01-Jan-2019', 'FROM alice@example. com'])返回自2019年开始以来，发自alice@example.com的消息。
imapObj.search(['SINCE 01-Jan-2019', 'NOT FROM alice@example. com'])返回自2019年开始以来，除alice@example.com外，其他所有人发来的消息。
imapObj. search(['OR FROM alice@example. com FROM bob@example. com'])返回发自alice@example.com或bob@example.com的所有信息。

search()方法不返回电子邮件本身，而是返回邮件的唯一ID（UID）。然后，可以将这些UID传入fetch()方法，获得邮件内容。

如果你的搜索匹配大量的电子邮件，Python可能抛出异常imaplib. error: got more than 10000 bytes。这个限制是防止Python程序消耗太多内存。

可以执行下面的代码，将限制从10 000字节改为10 000 000字节：

```
>>> import imaplib
>>> imaplib._MAXLINE = 10000000
```

2.12.9 取得邮件并标记为已读

得到UID的列表后，可以调用IMAPClient对象的fetch()方法，来获得实际的电子邮件内容。
UID列表是fetch()的第一个参数。第二个参数应该是['BODY[]']，它告诉fetch()下载UID列表中指定电子邮件的所有正文内容。

```
>>> rawMessages = imapObj.fetch(UIDs, ['BODY[]'])
>>> import pprint
>>> pprint.pprint(rawMessages)
{40040: {'BODY[]': 'Delivered-To:␣my_email_address@example.com\r\n'
                   'Received:␣by␣10.76.71.167␣with␣SMTP␣id␣' --snip--
                   '\r\n'
                   '------=_Part_6000970_707736290.1404819487066--\r\n',
         'SEQ': 5430}}
```

每条消息都保存为一个字典，包含两个键：'BODY[]'和'SEQ'。'BODY[]'键映射到电子邮件的实际正文。'SEQ'键是序列号，它与UID的作用类似。

如你所见，在'BODY[]'键中的消息内容是相当难理解的。这种格式称为RFC 822，是专为IMAP服务器读取而设计的。

如果你选择一个文件夹进行搜索，就用readonly=True关键字参数来调用select_folder()。这样做可以防止意外删除电子邮件，但这也意味着你用fetch()方法获取邮件时，它们不会标记为已读。

如果确实希望在获取邮件时将它们标记为已读，就需要将readonly=False传入select_folder()。

如果所选文件夹已处于只读模式，可以用另一个 select_folder()调用重新选择当前文件夹，这次用readonly=False关键字参数：

```
>>> imapObj.select_folder('INBOX', readonly=False)
```

2.12.10 从原始消息中获取电子邮件地址

对于fetch()方法返回的原始消息我们需要pyzmail模块解析这些原始消息。

```
>>> import pyzmail
>>> message = pyzmail.PyzMessage.factory(rawMessages[40041][b'BODY[]'])
```

我们可将fetch()的返回值传入此函数以获取信息。

现在，message中包含一个PyzMessage对象，它有几个方法，可以很容易地获得电子邮件主题行，以及所有发件人和收件人的地址。

get_subject()方法将主题返回为一个简单字符串。get_addresses()方法针对传入的字段，返回一个地址列表。

```
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward␣Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[('Jane␣Doe', 'my_email_address@example.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
```

请注意，get_addresses()的参数是'from'、'to'、'cc'或'bcc'。

get_addresses()的返回值是一个元组列表。

每个元组包含两个字符串：第一个是与该电子邮件地址关联的名称，第二个是电子邮件地址本身。

如果请求的字段中没有地址， get_addresses()返回一个空列表。

2.12.11 从原始消息中获取正文

如果电子邮件仅仅是纯文本，它的PyzMessage对象会将html_part属性设为None。

同样，如果电子邮件只是HTML，它的PyzMessage对象会将text_part属性设为None。

否则，text_part或html_part将有一个get_payload()方法，将电子邮件的正文返回为bytes数据类型。

但是,这仍然不是我们可以使用的字符串。

最后一步对get_payload()返回的bytes值调用decode()方法。decode()方法接收一个参数：这条消息的字符编码，它保存在text_part.charset或html_part.charset属性中。

```
>>> message.text_part != None
True
>>> message.text_part.get_payload().decode(message.text_part.charset)
```

```
w 'So␣long,␣and␣thanks␣for␣all␣the␣fish!\r\n\r\n-Al\r\n'
x >>> message.html_part != None
  True
y >>> message.html_part.get_payload().decode(message.html_part.charset)
  '<div␣dir="ltr"><div>So␣long,␣and␣thanks␣for␣all␣the␣fish!<br><br></div>-Al␣
  <br></div>\r\n'
```

2.12.12 删除电子邮件

要删除电子邮件，就向IMAPClient对象的delete_messages()方法传入一个消息UID的列表。
这会为电子邮件加上\Deleted标志。
调用expunge()方法，将永久删除当前选中的文件夹中带\Deleted标志的所有电子邮件。

```
>>> imapObj.select_folder('INBOX', readonly=False)
>>> UIDs = imapObj.search(['ON␣09-Jul-2019'])
>>> UIDs
[40066]
>>> imapObj.delete_messages(UIDs)
{40066: ('\\Seen', '\\Deleted')}
>>> imapObj.expunge()
('Success',  [(5452,  'EXISTS')])
```

这里，我们调用了IMAPClient对象的select_folder()方法，传入'INBOX'作为第一个参数，选择了收件箱。我们也传入了关键字参数readonly=False，这样我们就可以删除电子邮件了。

我们搜索收件箱中的特定日期收到的消息，将返回的消息ID保存在UIDs中。

调用delete_message()并传入UIDs以返回一个字典，其中每个键值对是一个消息 ID 和消息标志的元组，它现在应该包含\Deleted标志。

然后调用expunge()，永久删除带\Deleted标志 的邮件。如果清除邮件没有问题，就返回一条成功信息。

请注意，一些电子邮件提供商，如 Gmail，会自动清除用delete_messages()删除的电子邮件，而不是等待来自IMAP客户端的expunge命令。

从IMAP服务器断开直接imapObj.logout()即可。
如果程序运行了几分钟或更长时间，IMAP服务器可能会超时或自动断开。
在这种情况下，程序必须调用imapclient.IMAPClient()，来再次连接服务器。

2.12.13 短信

这部分暂时无用，按下不表。

2.13 操作图像

这部分暂时无用，按下不表。

2.14 GUI自动化控制键盘鼠标