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!')
```

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。

```
>>> 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。

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
'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
```

类似于. ^ $ * + ? { } [ ] \ | ( )的特殊符号，在匹配其本身时需要在其前面加上\。

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

win和macos上是不区分文件名大小写的，linux区分。