第十五章 正则表达式

Python

动机

我们可能不知道这些需要计算机编程处理文本或数据的具体内容，所以能把这些文本或数据以某种可**被计算机识别和处理的模式**表达出来是非常有用的。

如何通过编程使计算机具有在文本中检索某种模式的能力。

正则表达式(RE)为高级文本模式匹配，以及搜索-替代等功能提供了基础。正则表达式(RE)是一些由字符和特殊符号组成的字符串，它们描述了这些字符和字符的某种重复方式，因此能按某种模式匹配一个有相似特征的字符串的集合，因此能按某模式匹配一系列有相似特征的字符串。
RE可以匹配多个字符串。

正是这些特殊符号使得一个正则表达式可以匹配字符串集合而不只是一个字符串。

在Python专门术语中，有两种主要方法完成模式匹配：搜索(searching)和匹配(matching)。搜索，即在字符串任意部分查找匹配的模式，而匹配是指判断一个字符串能否从起始处全部或者部分的匹配某个模式。

“matching”是试图从整个字符串的开头进行匹配–搜索位置为**一个字符串的开头或结尾**位置(`^,$`)
“searching” 则可从一个字符串的任意位置开始匹配–搜索位置为任意位置

> 搜索通过 search()函数或方法来实现，而匹配是以调用 match()函数或方法实现的。

Symbols	Description	Examples
literal	匹配字符串的值	foo
re1|re2	匹配正则表达式re1或者re2	foo|bar
.	匹配除换行符(NEWLINE)外的任意一个单个字符	
^	匹配字符串的开始	^select–匹配任何以select开头的一个字符串
$	匹配字符串的结尾	/bin/*sh$
{N}	匹配前面的正则表达死N次	
\d	匹配任何数字和[0-9]一样	
\s	匹配任何空白字符和[\n\t\v\f]相同	of\sthe
\b	匹配单词边界	\bThe\b–与literal效果一样
\c	逐一匹配特殊字符c	\\|\.–匹配反斜杠和点号

匹配任意一个单个字符(`.`)

`f.o`–匹配fao,f9o,f#o等
`..`–匹配任意两个字符

从字符串的开头或结尾或单词边界开始匹配(^/$ /\b /\B)

有些符号和特殊字符是用来从字符串的开头或结尾开始搜索正则表达式模式的。

`\b`–对应的模式一定在一个单词的开头
`\B`–只匹配出现在一个单词中间的模式(不在单词边界上的字符)

RE	String Matched
the	任何包含有”the”的字符串
\bthe	任何以”the”开始的字符串
\bthe\b	仅匹配单词’the’
\Bthe	任意包含“the”但不以“the”开头的单词–非边界含有the的单词

创建字符类[]–逻辑或的作用–与`|`类似

使用方括号的正则表达式会匹配方括号里的任何一个字符。
`[cr][23][dp][o2]`–一个包含4个字符的字符串:第一个字符是r或者c,后面的2或者3，再接下来是d或者p，最后是o或者2。例如:c2do,c3do,r2d2等。

对仅有单个字符的正则表达式 ，使用管道符号和方括号的效果是等价的。
因为方括号只适用于单个字符的情况。

15.2.5 指定范围(-)和否定(^)

方括号除匹配单个字符外，还可以支持所指定的字符范围。

RE	String Matched
z.[0-9]	字符”z”，后面跟任意一个字符，然后是一个十进制数字
[^aeiou]	一个非元音字符–不匹配指定字符集里的任意字符

15.2.6 使用闭包操作符 (*, +, ?, {}) 实现多次出现/重复匹配

特殊符号 “*”, “+”, 和 “?”, 它们可以用于匹配字符串模式出现一次、多次、或未出现的情况。

星号(*)–零次或零次以上的情况(在计算机语言和编译器原理里，此操作符被叫做 **Kleene 闭包操作符**)。`*` - `>=0`

加号(+)操作符匹配它左边那个正则表达式模式至少出现一次的情况(它也被称为**正闭包操作符**)`+` – `>=1`

问号操作符(？)匹配它左边那个正则表达式模式出现零次或一次的情况。`?` – `<=1` –即问号`?`表示可选的意思

花括号操作符({ })， 花括号里可以是单个的值，也可以是由逗号分开的一对值。
一个值，如，{N}，则表示匹配 N 次出现；如果是一对值，即，{M, N}，就表示匹配 M 次到 N 次出现。

`[0-9]{15,16}` –15 或 16 位数字表示，例如：信用卡号码
`[0-9]{18}` –18位身份证
`</?[^>]+>` –匹配所有合法(和无效的)HTML 标签的字符串–主要是`?`,`/?` –表示/可以出现一次或者不出现，这就匹配了起始标签<>和结束标签</>

15.2.7 特殊字符表示字符集

有一些特殊字符可以用来代表字符集合。
`\d` – `[0-9]`
`\w` – `[a-zA-Z0-9]`
`\s` –代表空白符
这些特殊字符的大写表示不匹配
如 `\D` –标识 `[^0-9]`
`\w+-\d+` –一个有字幕或者数字组成的字符串和至少一个数字，两部分中间由连接字符连接
`[A-Za-z]\w*` –第一个是字母，若存在其他字符则是字母或者数字(几乎等价于语言中的标识符)
`\d{3}-\d{3}-\d{4}` –电话号码，800-555-1212
`\w+@\w\.com` –因为本身点号 `.` 在正则中表示任意一个字符，所以匹配点号本身要转义一下–简单的XXX@YYY.com格式的电子邮件

15.2.8 用圆括号 `()` 组建组

分组 `()` 存在的动机–我们不仅想知道是否整个字符串匹配我们的条件(正则表达式)，还**想在匹配成功时取出某个特定的字符串或子字串。**对匹配成功的字符串做进一步的处理

如在用正则 `\w+-\d+` 匹配一些内容后，又想把第一部分字符和第二部分数字分别保存，改如何做？
给这两个子模式都加上圆括号即 `(\w+)-(\d+)` 就可以解决上面的问题，实现对这两个匹配的子组分别进行访问。

> 具体取得子组的方法是 `group(num=0)` -默认0表示取得全部匹配的对象

`\d+(\.\d*)` –表示简单的浮点数即任意个十进制数字，后面跟一个可选的小数点，然后再接零个或者多个十进制数字。如0.004,2,75.
`(Mr?s\.)?[A-Z][a-z]* [ A-Za-z-]+` –名字和姓氏，对名字的限制(首字母大写，其它字母(如果存在)小写),全名前有可选的称谓(Mr.,Mrs,Ms,M.)

15.3 正则表达式和Python语言

正则表达式模块`re`在 Python 1.5 版本被引入。
Python中的re文档如下:
https://docs.python.org/2.7/library/re.html

新的 re 模块支持功能更强大，支持对正则表达式分组进行命名和按名字调用。

对分组进行命名和按名调用的语法：
`(?P<name>...)`–对分组进行命名
`group('name')`–按名调用

```
>>> m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcolm Reynolds")
>>> m.group('first_name')
'Malcolm'
>>> m.group('last_name')
'Reynolds'
```

15.3.1 re模块:核心函数和方法

`compile(pattern,flags=0)`–对模式pattern编译，flags为可选标志，并返回一个regex对象
`match(pattern,string,flags=0)`–用pattern匹配string，flags为可选标志，匹配成功则返回一个匹配对象，否则返回None
`search(pattern,string,flags=0)`–成功返回一个对象否则返回None
`findall(pattern,string[,flags])`–在字符串 string 中查找模式pattern匹配的所有(非重复)出现，返回一个匹配对象的列表
`finditer(pattern,string[,flags])`–与`findall`类似，但返回的是迭代器；对于每个匹配，该迭代器返回一个匹配对象–(类似于懒加载)
`split(pattern,string,max=0)`–根据 pattern 模式中的分隔符把字符 string 分割为一个列表，返回成功匹配的列表，最多分割max次(默认是分隔所有匹配的地方)
`sub(pattern,repl,string,max=0)`–repl表示replace,在string中所有pattern模式匹配的地方替换成repl
`group(num=0)`–返回全部匹配对象(或指定编号是 num 的子组)–这就是前面说的通过分组符号`()`取得子组的方法
`groups()`–返回一个包含全部匹配的子组的元组(如果没有成功匹配，就返回一个空元组)

> 调用 eval() 或 exec()调用一个代码对象而不是一个字符串，在性能上会有明显地提升。
> 换句话说，使用预编译代码对象要比使用字符串快，因
> 为解释器在执行字符串形式的代码前必须先把它编译成代码对象。
> 同样，这个概念也适用于正则表达式， 在模式匹配之前， 正则表达式模式必须先被编译成 regex 对象。因为正则要多次匹配，强烈建议对它做预编译。 `re.compile()` 就是用来提供此功能的。
>
> 其实模块函数会对已编译对象进行缓存，所以不是所有使用相同正则表达式模式的 `search()` 和 `match()` 都需要编译。即使这样，你仍然节省了查询缓存，和用相同的字符串反复调用函数的性能开销。 **缓存区被扩展到可以容纳 100个已编译的 regex 对象。**

15.3.2 使用 `compile()` 编译正则表达式

编译 rex 对象时给出一些可选标志符，可以得到特殊的编译对象。

如果你想在 regex 对象的方法中使用这些标志符，则必须在编译对象时传递这些参数。

15.3.3 匹配对象 和 `group(),groups()`方法

在处理正则表达式时，除 **regex 对象**–编译产生的对象外，还有另一种对象类型 - **匹配对象**。这些对象是在`match()`或 `search()`被成功调用之后所返回的结果。匹配对象有两个主要方法： `group()` 和 `groups()`。

关于对正则匹配的对象进行命名的功能参见`re`模块的文档。

15.3.4 用 `match()`匹配字符串–从单词的position=0位置查找

```
print re.match('foo', 'food on the table').group()
```

```
m = re.match('foo', 'seafood') # no match 匹配失败
```

即使字符串比模式要长，匹配也可能成功；只要模式是从字符串的开始进行匹配的。

`match()`方法是从字符串string的起始处开始匹配pattern模式。position=0

`search()` 查找字符串中模式首次出现的位置,从做左到右搜索

15.3.5 `search()`在一个字符串中查找一个模式(搜索与匹配的比较)

你要搜索的模式出现在一个字符串中间的机率要比出现在字符串开头的机率更大一些。
`search()`方法是在一个字符串中间进行搜索.position>=0

search 和 match 的工作方式一样，不同之处在于 search 会检查参数字符串任意位置的地方给定正则表达式模式的匹配情况

15.3.6 匹配多个字符串(|)

匹配多个字符串使用管道符号(`|`)

15.3.6 匹配任意单个字符(`.`)

点号`.`匹配任意单个字符，但是不能匹配换行和空字符串

15.3.8 创建字符集合([ ])

`[]`相当于逻辑或

15.3.9 重复、特殊字符和子组

将前面匹配电子邮件的pattern改成支持添加主机名称的支持,即改为支持www.xxx.com

```
# 不支持主机名称的邮件
patt = '\w+@\w+\.com'
# 支持主机名称的邮件
patt2 = '\w+@(\w+\.)?\w+\.com' #主机名可有可无的
print re.match(patt2,'nobody@xxx.com').group()
print re.match(patt2,'nobody@www.xxx.com').group()

# 允许任意数量的子域名存在,将?号改为*, \w+@(\w+\.)*\w+\.com
patt3 = '\w+@(\w+\.)*\w+\.com'
print re.match(patt3, 'nobody@www.xxx.yyy.zzz.com').group()
```

`group(1)`-匹配子组1
`groups()`-所有匹配子组–以元组返回

15.3.10 从字符串的开头或结尾匹配及在单词边界上的匹配

`m = re.search('^The', 'The end.') # match`
`m = re.search('^The', 'end. The') # not at beginning`
`m = re.search(r'\bthe', 'bite the dog') # at a boundary #在词边界`
`m = re.search(r'\bthe', 'bitethe dog') # no boundary #不在词边界`

你可能在这里注意到了原始字符串(raw strings) 的出现。在本章末尾的核心笔记中，有关于它的说明。通常，在正则表达式中使用原始字符串是个好主意。

15.3.11 用 findall()找到每个出现的匹配部分

```
print re.findall('car', 'car')
print re.findall('car', 'scary')
print re.findall('car', 'carry the barcardi to the car')
```

15.3.12 用 sub()[和 subn()]进行搜索和替换

字符串中所有匹配正则表达式模式的部分进行替换。
用来替换的部分通常是一个字符串，但也可能是一个函数，该函数返回一个用来替换的字符串。

`sub(pattern,repl,string,max=0)`

```
re.sub('X','Mr. Smith','attn: X\t\tDear X,\t') #attn: Mr. Smith     Dear Mr. Smith,
```

15.3.13 用 `split()` 分割(分隔模式)

正则的`split()`比字符串的split功能更强大。二者执行效果是一样的。
如果你不想在每个模式匹配的地方都分割字符串，你可以通过设定一个值参数(非零)来指定分割的最大次数。
`print re.split(':', 'str1:str2:str3') # ['str1', 'str2', 'str3']`

> 原始字符串的产生正是由于有正则表达式的存在。原因是 ASCII 字符和正则表达式特殊字符间所产生的冲突。
> 比如，特殊符号`\b`在ASCII 字符中代表退格键， 但同时`\b`也是一个正则表达式的特殊符号， 代表 “匹配一个单词边界”。

Python 程序员在定义正则表达式时都只使用原始字符串。

```
m = re.match('\bblow', 'blow') # backspace, no match #退格键,没有匹配
m = re.match('\\bblow', 'blow') # escaped \, now it works #用\转义后，现匹配了
m = re.match(r'\bblow', 'blow') # use raw string instead #改用原始字符串
```

15.4 正则表达式示例

```python
# 15.4 正则表达式示例
import re
from random import randint,choice
from string import lowercase
from sys import maxint
from time import ctime
import time
# import datetime

'''
函数genedata()生成 3 个字段， 字段由一对冒号， 或双冒号分隔。 第一个字段是一个随机(32 位)整数，被转换为一个日期。
第二个字段是一个随机产生的电子邮件(e-mail)地址,最后一个字段是由单个横线( - )分隔的一个整数集合
'''

from os import linesep
def genedata():
    filename = 'redata.txt'
    fobj = open(filename,'w')
    doms = ('com','edu','net','gov')
    # 1.日期字段
    for i in range(randint(5,10)):
        dtint = randint(0,maxint-1)
        dtstr = ctime(dtint) # data string--Tue Nov 26 13:25:35 2013

        # 2.生成登录的邮箱名
        shorter = randint(4,7)
        emailname = ''
        for i in range(shorter):
            emailname += choice(lowercase) # 从小写字母中随机选择一个字母,循环则是控制选择的次数

        # 2.生成邮箱的之后的domain
```

```
            # 邮箱地址的域名长度在 4 到 12 个字符之间，但不能短于登录名的长度
            longer = randint(shorter,12)
            domain = ''
            for i in range(longer):
                domain += choice(lowercase)
            # 3. 3个整数的组合
            print '%s::%s@%s.%s::%d-%d-%d' %(dtstr,emailname,domain,choice(doms),dtint,shorter,longer)
            strwrite = '%s::%s@%s.%s::%d-%d-%d' %(dtstr,emailname,domain,choice(doms),dtint,shorter,longe
r)
            # 将经过正则表达式处理后的数据写入文件
            # ((\w){3}), 它的含义就变成三个连续的单个由字符或数字组成的字符P649
            patt = '^(\w{3})' # '^(Mon|Tue|Wed|Thu|Fri|Sat|Sun)'
            matchedstr = re.match(patt,strwrite).group()
            fobj.writelines(matchedstr + linesep)
        fobj.close()
```

15.4.2 搜索与匹配的比较，“贪婪”匹配–通配符是贪婪匹配

正则表达式本身默认是贪心匹配的

> 也就是说，如果正则表达式模式中使用到通配字(`*,+,?`)那它在按照从左到右的顺序求值时，会尽量“抓取”满足匹配的最长字符串。

一个解决办法是用“非贪婪”操作符，“?”。(因为 `?` 的作用就是匹配零个或者一个)
这个操作符可以用在 “*”, “+”, 或 “?” 的后面。它的作用是要求正则表达式引擎匹配的字符越少越好。

```
patt = '.+\d+-\d+-\d+' #该模式的意识是.+采用贪婪匹配模式,\d+-\d+-\d+也采用贪婪匹配模式
m = re.match(patt, data).group() #全部匹配部分
# Thu Feb 15 17:46:04 2007::uzifzf@dpyivihw.gov::1171590364-6-8

# 只想获得每行末尾数字的字段，而不是整个字符串，所以需要用圆括号将我们感兴趣的那部分数据分成一组

patt = '.+(\d+-\d+-\d+)'
m = re.match(patt, data).group(1)
# 4-6-8
# 本应该得到数据“ 1171590364-6-8”，而不应该是“ 4-6-8”啊?
# 原因是:  正则表达式本身默认是贪心匹配的。
# 也就是说，如果正则表达式模式中使用到通配字，那它在按照从左到右的顺序求值时，会尽量“抓取”满足匹配的最长字符串。

# 在我们上面的例子里，“.+”会从字符串的起始处抓取满足模式的最长字符，其中包括我们想得到的第一个整数字段的中的大部分。“\d+”只需一位数字就可以匹配，所以它匹配了数字“4”,而“.+” 则匹配了从字符串起始到这个第一位数字“4”之间的所有字符: “Thu Feb 15 17:46:04 2007::uzifzf@dpyivihw.gov::117159036”

# 一个解决办法是用“非贪婪”操作符，“?”。
这个操作符可以用在 “*”, “+”, 或 “?” 的后面。它的作用是要求正则表达式引擎匹配的字符越少越好。
```

```
pattern2 = '.+?(\d+-\d+-\d+)' # 只取第三个字段
matchedstr2 = re.match(pattern2,strwrite).group(1) # 子组 1
# 1171590364-6-8

# 其实`'.+?(\d+-\d+-\d+)'`正则的意思是前面(.+?)匹配尽可能少，而后面[(\d+-\d+-\d+)]尽可能多。
```

练习

课本中的正则表达式示例

```python
# 15.4 正则表达式示例
from random import randint,choice
from string import lowercase
from sys import maxint
from time import ctime
import time
# import datetime

'''
函数genedata()生成 3 个字段， 字段由一对冒号， 或双冒号分隔。 第一个字段是一个随机(32 位)整数，被转换为一个日期。
第二个字段是一个随机产生的电子邮件(e-mail)地址,最后一个字段是由单个横线( - )分隔的一个整数集合
'''

from os import linesep
def genedata():
    filename = 'redata.txt'
    fobj = open(filename,'w')
    doms = ('com','edu','net','gov')
    # 1.日期字段
    for i in range(randint(5,10)):
        dtint = randint(0,maxint-1)
        dtstr = ctime(dtint) # data string--Tue Nov 26 13:25:35 2013

        # 2.生成登录的邮箱名
        shorter = randint(4,7)
        emailname = ''
        for i in range(shorter):
            emailname += choice(lowercase) # 从小写字母中随机选择一个字母,循环则是控制选择的次数

        # 2.生成邮箱的之后的domain
        # 邮箱地址的域名长度在 4 到 12 个字符之间，但不能短于登录名的长度
```

```
            longer = randint(shorter,12)
            domain = ''
            for i in range(longer):
                domain += choice(lowercase)
            # 3. 3个整数的组合
            print '%s::%s@%s.%s::%d-%d-%d' %(dtstr,emailname,domain,choice(doms),dtint,shorter,longer)
            strwrite = '%s::%s@%s.%s::%d-%d-%d' %(dtstr,emailname,domain,choice(doms),dtint,shorter,longe
r)
            # 将经过正则表达式处理后的数据写入文件
            # ((\w){3}), 它的含义就变成三个连续的单个由字符或数字组成的字符P649
            # patt = '^(\w{3})' # '^(Mon|Tue|Wed|Thu|Fri|Sat|Sun)'
            # pattern2 = '.+?(\d+-\d+-\d+)' # 只取第三个字段
            # matchedstr = re.match(patt,strwrite).group()
            # matchedstr2 = re.match(pattern2,strwrite).group(1) #  子组 1
            # print matchedstr
            # print matchedstr2
            strwrite2 = strwrite + linesep
            fobj.writelines(strwrite2)
        fobj.close()


if __name__ == "__main__":
    genedata() # 生成需要的数据:由日期、邮箱和整数集合(所选随机日期对应是整数,后面两个是登录名和域名的长度)
```

以下是redata.txt文件中随机生成的数据，你生成的和下面的很大程度上应该不一样

```
Tue Jul 02 08:46:37 1996::cpvz@zwloisj.com::836268397-4-7

Wed Jan 08 18:15:57 2014::mgvi@mmokxe.com::1389176157-4-6

Mon Aug 24 00:00:13 1987::ltgq@gghjyuvusmc.gov::556732813-4-11

Sat May 10 04:41:29 2014::qoez@cumuxju.net::1399668089-4-7

Sun Aug 16 05:45:09 1981::dhyd@rqueqryjmzwa.net::366759909-4-12

Thu Feb 04 01:39:10 1971::cwuk@xhvedbovhke.gov::34450750-4-11

Sat Apr 03 13:23:41 2004::sfyzaf@zyiqpcvzucsq.edu::1080969821-6-12

Wed Apr 30 20:58:11 2025::rrkhbq@gsnpvrhfikut.net::1746017891-6-12

Tue Aug 05 20:16:16 2003::aekotar@ifgxezbonkc.gov::1060085776-7-11

Sat Jul 01 11:20:37 1972::axixa@wwxizdnkay.edu::78808837-5-10
```

```
import re


# 15-1
def re151():
    patter1 = r'[bh][aiu]t' # bat|bit|but|hat|hit|hut
    m = re.match(patter1,'bat')
    if m is not None:
        print m.group()
    else:
        print 'not matched'
# 15-2
def re152():
    patter2 = r'[a-zA-Z]+\s[a-zA-Z]+' # bat|bit|but|hat|hit|hut
    m = re.match(patter2,'First Last')
    print m.group()

# 15-3
def re153():
    patter3 = r'([A-Za-z]+\.)+?,\s[A-Za-z]+' # bat|bit|but|hat|hit|hut
    m = re.match(patter3, 'Mr., Join') # 可以匹配如Mr. Join
    if m is not None:
        print m.group()
    else:
        print 'not matched'

# 15-4 匹配所有合法的Python标识符--字母下划线打头，后面接任意数字字母和下划线
def re154():
    pattern4 = r'[a-zA-z_][\w_]+$' # 要注意加上结束符$,因为后面接的是任意字母/数字或者下划线
    valueList = ['10_mys', '_myValue09', 'Post99','*hahah', 'list@%']
    for c in valueList:
```

```
        m = re.match(pattern4, c)
        if m is not None:
            print m.group()
        else:
            print c,':Illegal...'

# 15-6
def re156():
    pattern6 = r'www\.\w+\.com'
    pattern62 = r'www[\.\w]+'
    m = re.match(pattern6, 'www.baidu.com')
    print m.group()
#15-7 匹配全体 Python 整数的字符串表示形式的集合
def re157():
    pattern7 = r'\d+[Ll]?' #?的作用取消了\d+的贪婪匹配,使得该模式可以匹配以数字开头而已非数字结尾的字符串
    m = re.match(pattern7, '12345678L')
    print m.group()


#15-8 匹配全体 Python 长整数的字符串表示形式的集合
def re158():
    pattern8 = r'\d+[Ll]$' #长整数必须以L/l结尾
    m = re.match(pattern8, '1234L')
    print '15-8----',m.group()
#15-09 匹配全体 Python 浮点数的字符串表示形式的集合
def re159():
    print '15-09. 匹配全体 Python 浮点数的字符串表示形式的集合',
    pattern9 = r'\d+\.?(\d+)?' #\d+\.?\d+ 是另外一个方式
    m = re.match(pattern9, '123.00001')
    print m.group()

#15-10 匹配全体 Python 复数的字符串表示形式的集合
```

```
def re1510():
    print '15-10. 匹配全体 Python 复数的字符串表示形式的集合',
    pattern10 = r'\d+\.\d+\+\d+\.?\d+j$' #小数点最多一个
    m = re.match(pattern10, '1.23+1.123j')
    print m.group()

#15-11 匹配所有合法的电子邮件地址
def re1511():
    print '15-11. 匹配所有合法的电子邮件地址',
    pattern11 = r'\w+@(\w+\.)+[a-zA-Z]+' #(\w+\.)+ 匹配存在的多级主机地址
    m = re.match(pattern11, '123@qq.163.email.com')
    print m.group()


#15-12 匹配所有合法的 Web 网站地址
def re1512():
    print '15-12. 匹配所有合法的 Web 网站地址',
    pattern12 = r'^(http://)?([w|W]{3})?\.?(\w+|\.|/)*' #网站默认顶级域名为字母
    m = re.match(pattern12, 'email.163.com/login.html')
    print m.group()

#15-13 type()内建函数返回一个对象类型
def re1513():
    print '15-13. type()内建函数返回一个对象类型',
    pattern13 = r'^<type\s\'(?P<type>\w+)\'>$'
    m = re.match(pattern13, "<type 'float'>")
    print m.group(1)

# 15-14. 匹配日历上的月份
def re1514():
    print '15-14. 匹配日历上的月份',
    pattern14 = r'1[0-2]' # 0?[1-9]
```

```
    m = re.match(pattern14, '12')
    print m.group()

# 15-15. 信用卡卡号
# 15-15 信用卡卡号是否合法的算法
"""
并不是随便的信用卡号都是合法的，它必须通过Luhn算法来验证。
验证过程：
1. 从卡号最后一位数字开始，逆向将奇数位(1、3、5等等)相加。

2. 从卡号最后一位数字开始，逆向将偶数位数字，先乘以2（如果乘积为两位数，则将其减去9），再求和。

3. 将奇数位总和加上偶数位总和，结果应该可以被10整除。
"""
def re1515():
    print '15-15. 信用卡卡号校验',
    pattern15 = r'\d{4}-\d{6}-\d{5}|\d{4}-\d{4}-\d{4}-\d{4}'
    m = re.match(pattern15, '0004-1256-1123-0003')
    print m.group()

# Luhn算法的实现
def luhn_check(num):
  ''' Number - List of reversed digits '''
  digits = [int(x) for x in reversed(str(num))]
  check_sum = sum(digits[::2]) + sum((dig//10 + dig%10) for dig in [2*el for el in digits[1::2]])
  return check_sum%10 == 0


if __name__ == "__main__":
    print 1
    re151()
    re152()
```

```
    re153()
    re154()
    re156()
    re157()
    re158()
    re159()
    re1510()
    re1511()
    re1512()
    re1513()
    re1514()
    re1515()
    print luhn_check('62284808501099377132')
```