

Eva\_J

程序媛

# python之路——常用模块

## 阅读目录

- 认识模块
  - 什么是模块
  - 模块的导入和使用
- 常用模块一
  - collections模块
  - 时间模块
  - random模块
  - os模块
  - sys模块
  - 序列化模块
  - re模块
- 常用模块二
  - hashlib模块
  - configparser模块
  - logging模块

## 认识模块

什么是模块

什么是模块？

常见的场景：一个模块就是一个包含了python定义和声明的文件，文件名就是模块名字加上.py的后缀。

但其实import加载的模块分为四个通用类别：

1 使用python编写的代码（.py文件）

2 已被编译为共享库或DLL的C或C++扩展

3 包好一组模块的包

4 使用C编写并链接到python解释器的内置模块

### 为何要使用模块？

如果你退出python解释器然后重新进入，那么你之前定义的函数或者变量都将丢失，因此我们通常将程序写到文件中以便永久保存下来，需要时通过python test.py方式去执行，此时test.py被称为脚本script。

随着程序的发展，功能越来越多，为了方便管理，我们通常将程序分成一个个的文件，这样做程序的结构更清晰，方便管理。这时我们不仅仅可以把这些文件当做脚本去执行，还可以把他们当做模块来导入到其他的模块中，实现了功能的重复利用，

昵称：Eva\_J  
园龄：4年9个月  
粉丝：4226  
关注：7  
[-取消关注](#)

< 2020年8月 >						
日	一	二	三	四	五	六
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

搜索

## 常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签
- 更多链接

## 我的标签

- [go\(1\)](#)
- [python\(1\)](#)

## 随笔分类

- python\_Django(4)
- python基础语法(7)
- python面向对象(4)
- python网络编程(1)
- python线程进程与协程(6)

## 随笔档案

- 2020年3月(1)
- 2019年3月(1)
- 2018年7月(1)
- 2018年1月(1)
- 2017年8月(1)
- 2017年4月(1)
- 2017年3月(1)

模块的导入和使用

模块的导入应该在程序开始的地方

更多相关内容 <http://www.cnblogs.com/Eva-J/articles/7292109.html>

常用模块

collections模块

在内置数据类型 (dict、list、set、tuple) 的基础上，collections模块还提供了几个额外的数据类型：Counter、deque、defaultdict、namedtuple和OrderedDict等。

- 1.namedtuple: 生成可以使用名字来访问元素内容的tuple
- 2.deque: 双端队列，可以快速的从另外一侧追加和推出对象
- 3.Counter: 计数器，主要用来计数
- 4.OrderedDict: 有序字典
- 5.defaultdict: 带有默认值的字典

namedtuple

我们知道tuple可以表示不变集合，例如，一个点的二维坐标就可以表示成：

```
>>> p = (1, 2)
```

但是，看到(1, 2)，很难看出这个tuple是用来表示一个坐标的。

这时，namedtuple就派上了用场：

```
>>> from collections import namedtuple
>>> Point = namedtuple('Point', ['x', 'y'])
>>> p = Point(1, 2)
>>> p.x
1
>>> p.y
2
```

类似的，如果要用坐标和半径表示一个圆，也可以用namedtuple定义：

```
#namedtuple('名称', [属性list]):
Circle = namedtuple('Circle', ['x', 'y', 'r'])
```

deque

使用list存储数据时，按索引访问元素很快，但是插入和删除元素就很慢了，因为list是线性存储，数据量大的时候，插入和删除效率很低。

deque是为了高效实现插入和删除操作的双向列表，适合用于队列和栈：

```
>>> from collections import deque
>>> q = deque(['a', 'b', 'c'])
>>> q.append('x')
>>> q.appendleft('y')
>>> q
deque(['y', 'a', 'b', 'c', 'x'])
```

deque除了实现list的append()和pop()外，还支持appendleft()和popleft()，这样就可以非常高效地往头部添加或删除元素。

OrderedDict

使用dict时，Key是无序的。在对dict做迭代时，我们无法确定Key的顺序。

如果要保持Key的顺序，可以用OrderedDict：

[返回顶部](#)

- 2016年6月(2)
- 2016年4月(1)
- 2016年3月(1)
- 2016年2月(1)
- 2016年1月(10)
- 2015年12月(6)
- 2015年11月(6)

文章分类

[返回顶部](#)

- flask(1)
- go(3)
- mysql(7)
- python宣讲专用课件(3)
- python之路(16)
- 数据库相关(7)
- 周末班(3)


友链

- 银角大王
- 学霸yuan先生
- 冷先生


最新评论

- 1. Re:python之路——装饰器函数  
@timer def 1,func1():#1，是序号 print('in func1) 2,func1() 这段代码中@timer是对底下这个函数 (1, func1()) 进行装饰. def fun C...  
--琉璃xing
- 2. Re:python之路——内置函数和匿名函数  
@相思木木 写一个元素也可以，只是最后生成的“tup”只有一个元素，所以， tup[1]就报错了，他只有一个元素...  
--Hello\_Thanos
- 3. Re:python之路——常用模块  
@jeason\_hui 当然一样啦...  
--diracy
- 4. Re:python之路——初识函数  
@道霖 嗯 大概是py2发现有bug 所以在3中才改过来的。...  
--Eva\_J
- 5. Re:python之路——初识函数  
在小结中的参数顺序是不是有问题呀！正确的应该是： 位置参数，默认参数，\*args, \*\*kwargs 验证了下，是版本不同的缘故。在2.7中 位置参数，默认参数，\*args, kwargs 是OK的 位置...  
--道霖

阅读排行榜



```
>>> from collections import OrderedDict
>>> d = dict([('a', 1), ('b', 2), ('c', 3)])
>>> d # dict的Key是无序的
{'a': 1, 'c': 3, 'b': 2}
>>> od = OrderedDict([('a', 1), ('b', 2), ('c', 3)])
>>> od # OrderedDict的Key是有序的
OrderedDict([('a', 1), ('b', 2), ('c', 3)])
```



注意，OrderedDict的Key会按照插入的顺序排列，不是Key本身排序：

```
>>> od = OrderedDict()
>>> od['z'] = 1
>>> od['y'] = 2
>>> od['x'] = 3
>>> od.keys() # 按照插入的Key的顺序返回
['z', 'y', 'x']
```

defaultdict

有如下值集合 [11,22,33,44,55,66,77,88,99,90...], 将所有大于 66 的值保存至字典的第一个key中，将小于 66 的值保存至第二个key的值中。

即： {'k1': 大于66 , 'k2': 小于66}

⊞

原生字典解决方法

⊞

defaultdict字典解决方法

使用dict时，如果引用的Key不存在，就会抛出KeyError。如果希望key不存在时，返回一个默认值，就可以用defaultdict：

⊞

例2

Counter

Counter类的目的是用来跟踪值出现的次数。它是一个无序的容器类型，以字典的键值对形式存储，其中元素作为key，其计数作为value。计数值可以是任意的Integer（包括0和负数）。Counter类和其他语言的bags或multisets很相似。

```
c = Counter('abcdeababcdabacaba')
print c
输出: Counter({'a': 5, 'b': 4, 'c': 3, 'd': 2, 'e': 1})
```

其他详细内容 <http://www.cnblogs.com/Eva-J/articles/7291842.html>

[返回顶部](#)

时间模块

和时间有关系的我们就要用到时间模块。在使用模块之前，应该首先导入这个模块。

```
#常用方法
1.time.sleep(secs)
(线程)推迟指定的时间运行。单位为秒。
2.time.time()
获取当前时间戳
```

表示时间的三种方式

在Python中，通常有这三种方式来表示时间：时间戳、元组(struct\_time)、格式化的时间字符串：

- (1)时间戳(timestamp)：通常来说，时间戳表示的是从1970年1月1日00:00:00开始按秒计算的偏移量。我们运行“type(time.time())”，返回的是float类型。
- (2)格式化的时间字符串(Format String)：‘1999-12-06’

⊞

python中时间日期格式化符号：

1. python之路——博客目录(147183)
2. python——赋值与深浅拷贝(33262)
3. python——SQL基本使用(30974)
4. python——django使用mysql数据库（一）(23420)
5. 前端开发的正确姿势——各种文件的目录结构规划及引用(21695)

评论排行榜

1. python之路——博客目录(32)
2. python——赋值与深浅拷贝(16)
3. python——进程基础(9)
4. python的类和对象——进阶篇(8)
5. python\_控制台输出带颜色的文字方法(8)

推荐排行榜

1. python之路——博客目录(64)
2. python——赋值与深浅拷贝(35)
3. python3.7导入gevent模块报错的解决方案(6)
4. python\_控制台输出带颜色的文字方法(5)
5. python——挖装饰器祖坟事件(5)

(3)元组(struct\_time)：struct\_time元组共有9个元素共九个元素:(年，月，日，时，分，秒，一年中第几周，一年中第几天等)

索引 (Index)	属性 (Attribute)	值 (Values)
0	tm_year (年)	比如2011
1	tm_mon (月)	1 - 12
2	tm_mday (日)	1 - 31
3	tm_hour (时)	0 - 23
4	tm_min (分)	0 - 59
5	tm_sec (秒)	0 - 60
6	tm_wday (weekday)	0 - 6 (0表示周一)
7	tm_yday (一年中的第几天)	1 - 366
8	tm_isdst (是否是夏令时)	默认为0

首先，我们先导入time模块，来认识一下python中表示时间的几种格式：

```
#导入时间模块
>>>import time

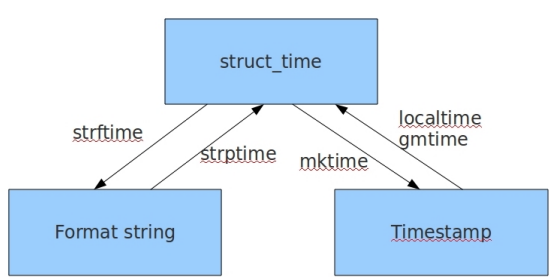
#时间戳
>>>time.time()
1500875844.800804

#时间字符串
>>>time.strftime("%Y-%m-%d %X")
'2017-07-24 13:54:37'
>>>time.strftime("%Y-%m-%d %H-%M-%S")
'2017-07-24 13-55-04'

#时间元组: localtime 将一个时间戳转换为当前时区的struct_time
time.localtime()
time.struct_time(tm_year=2017, tm_mon=7, tm_mday=24,
                  tm_hour=13, tm_min=59, tm_sec=37,
                  tm_wday=0, tm_yday=205, tm_isdst=0)
```

小结：时间戳是计算机能够识别的时间；时间字符串是人能够看懂的时间；元组则是用来操作时间的

几种格式之间的转换



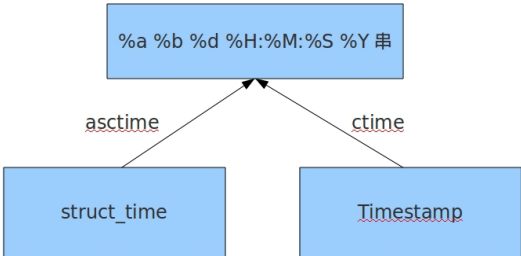
```
#时间戳-->结构化时间
#time.gmtime(时间戳) #UTC时间，与英国伦敦当地时间一致
#time.localtime(时间戳) #当地时间。例如我们现在在北京执行这个方法：与UTC时间相差8小时，UTC时间+8小时 = 北京时间
>>>time.gmtime(1500000000)
time.struct_time(tm_year=2017, tm_mon=7, tm_mday=14, tm_hour=2, tm_min=40, tm_sec=0, tm_wday=4, tm_yd
>>>time.localtime(1500000000)
time.struct_time(tm_year=2017, tm_mon=7, tm_mday=14, tm_hour=10, tm_min=40, tm_sec=0, tm_wday=4, tm_y

#结构化时间-->时间戳
#time.mktime(结构化时间)
>>>time_tuple = time.localtime(1500000000)
```

```
>>>time.mktime(time_tuple)
1500000000.0
```

```
#结构化时间-->字符串时间
#time.strftime("格式定义","结构化时间")  结构化时间参数若不传，则显示当前时间
>>>time.strftime("%Y-%m-%d %X")
'2017-07-24 14:55:36'
>>>time.strftime("%Y-%m-%d",time.localtime(1500000000))
'2017-07-14'
```

```
#字符串时间-->结构化时间
#time.strptime(时间字符串,字符串对应格式)
>>>time.strptime("2017-03-16", "%Y-%m-%d")
time.struct_time(tm_year=2017, tm_mon=3, tm_mday=16, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=3, tm_yday=75)
>>>time.strptime("07/24/2017", "%m/%d/%Y")
time.struct_time(tm_year=2017, tm_mon=7, tm_mday=24, tm_hour=0, tm_min=0, tm_sec=0, tm_wday=0, tm_yday=205)
```



```
#结构化时间 --> %a %b %d %H:%M:%S %Y串
#time.asctime(结构化时间) 如果不传参数，直接返回当前时间的格式化串
>>>time.asctime(time.localtime(1500000000))
'Fri Jul 14 10:40:00 2017'
>>>time.asctime()
'Mon Jul 24 15:18:33 2017'
```

```
#时间戳 --> %a %b %d %H:%M:%S %Y串
#time.ctime(时间戳) 如果不传参数，直接返回当前时间的格式化串
>>>time.ctime()
'Mon Jul 24 15:19:07 2017'
>>>time.ctime(1500000000)
'Fri Jul 14 10:40:00 2017'
```

计算时间差

datetime模块

- 1.datetime.now() # 获取当前datetime
- datetime.utcnow() # 获取当前格林威治时间

View Code

- 2.datetime(2017, 5, 23, 12, 20) # 用指定日期时间创建datetime

```
from datetime import datetime

#用指定日期创建
c=datetime(2017, 5, 23, 12, 20)
print('指定日期:',c)
```

- 3.将以下字符串转换成datetime类型：

'2017/9/30'

'2017年9月30日星期六'

```
'2017年9月30日星期六8时42分24秒'
'9/30/2017'
'9/30/2017 8:42:50 '
```

时间字符串格式化

4.将以下datetime类型转换成字符串：

```
2017年9月28日星期4,10时3分43秒
Saturday, September 30, 2017
9/30/2017 9:22:17 AM
September 30, 2017
```

时间字符串格式化

5.用系统时间输出以下字符串：

```
今天是2017年9月30日
今天是这周的第？ 天
今天是今年的第？ 天
今周是今年的第？ 周
今天是当月的第？ 天
```

View Code

[返回顶部](#)

random模块



```
>>> import random
#随机小数
>>> random.random()          # 大于0且小于1之间的小数
0.7664338663654585
>>> random.uniform(1,3)      #大于1小于3的小数
1.6270147180533838
#恒富：发红包

#随机整数
>>> random.randint(1,5)      # 大于等于1且小于等于5之间的整数
>>> random.randrange(1,10,2)  # 大于等于1且小于10之间的奇数

#随机选择一个返回
>>> random.choice([1,'23',[4,5]])  # #1或者23或者[4,5]
#随机选择多个返回，返回的个数为函数的第二个参数
>>> random.sample([1,'23',[4,5]],2) # #列表元素任意2个组合
[[4, 5], '23']

#打乱列表顺序
>>> item=[1,3,5,7,9]
>>> random.shuffle(item)      # 打乱次序
>>> item
[5, 1, 3, 7, 9]
>>> random.shuffle(item)
>>> item
[5, 9, 7, 1, 3]
```



练习：生成随机验证码

生成验证码

[返回顶部](#)

os模块

os模块是与操作系统交互的一个接口



注意: os.stat('path/filename') 获取文件/目录信息 的结构说明

<div><div></div><div>stat 结构</div></div>
<div><div></div><div>os模块的属性</div></div>

[返回顶部](#)

### sys模块

sys模块是与python解释器交互的一个接口

sys.argv	命令行参数List，第一个元素是程序本身路径
sys.exit(n)	退出程序，正常退出时exit(0)，错误退出sys.exit(1)
sys.version	获取Python解释程序的版本信息
sys.path	返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值
sys.platform	返回操作系统平台名称

异常处理和status

[返回顶部](#)

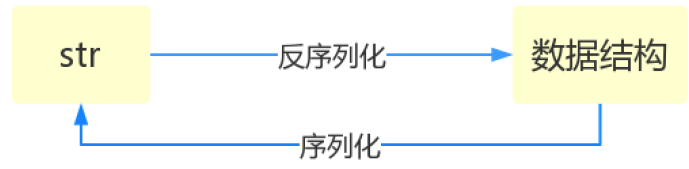
### 序列化模块

什么叫序列化——将原本的字典、列表等内容转换成一个字符串的过程就叫做**序列化**。

<div><div></div><div>为什么要有序列化模块</div></div>
---

#### 序列化的目的

- 1、以某种存储形式使自定义对象持久化；
- 2、将对象从一个地方传递到另一个地方。
- 3、使程序更具维护性。



json

Json模块提供了四个功能：dumps、dump、loads、load

loads和dumps
load和dump
ensure_ascii关键字参数
其他参数说明
json的格式化输出

pickle

json & pickle 模块

用于序列化的两个模块

- json，用于字符串 和 python数据类型间进行转换
- pickle，用于python特有的类型 和 python的数据类型间进行转换

pickle模块提供了四个功能：dumps、dump(序列化，存)、loads (反序列化，读)、load （不仅可以序列化字典，列表...**可以把python中任意的数据类型序列化**）

pickle
--------

这时候机智的你又要说了，既然pickle如此强大，为什么还要学json呢？  
这里我们要说明一下，json是一种所有的语言都可以识别的数据结构。  
如果我们将一个字典或者序列化成了一个json存在文件里，那么java代码或者js代码也可以拿来用。  
但是如果我们用pickle进行序列化，其他语言就不能读懂这是什么了~  
所以，如果你序列化的内容是列表或者字典，我们非常推荐你使用json模块  
但如果出于某种原因你不得不序列化其他的数据类型，而未来你还会用python对这个数据进行反序列化的话，那么就可以使用pickle

[返回顶部](#)

re模块

讲正题之前我们先来看一个例子：<https://reg.jd.com/reg/person?ReturnUrl=https%3A//www.jd.com/>

这是京东的注册页面，打开页面我们就看到这些要求输入个人信息的提示。  
假如我们随意的在手机号码这一栏输入一个11111111111，它会提示我们格式有误。  
这个功能是怎么实现的呢？  
假如现在你用python写一段代码，类似：

```
phone_number = input('please input your phone number : ')
```

你怎么判断这个phone\_number是合法的呢？

根据手机号码一共11位并且是只以13、14、15、18开头的数字这些特点，我们用python写了如下代码：

```
while True:
    phone_number = input('please input your phone number : ')
```



```
if len(phone_number) == 11 \
    and phone_number.isdigit() \
    and (phone_number.startswith('13') \
        or phone_number.startswith('14') \
        or phone_number.startswith('15') \
        or phone_number.startswith('18')):
    print('是合法的手机号码')
else:
    print('不是合法的手机号码')
```

这是你的写法，现在我要展示一下我的写法：

```
import re
phone_number = input('please input your phone number : ')
if re.match('^(13|14|15|18)[0-9]{9}$',phone_number):
    print('是合法的手机号码')
else:
    print('不是合法的手机号码')
```

对比上面的两种写法，此时此刻，我要问你你喜欢哪种方法呀？你肯定还是会说第一种，为什么呢？因为第一种不用学呀！  
但是如果现在有一个文件，我让你从整个文件里匹配出所有的手机号码。你用python给我写个试试？  
但是学了今天的技能之后，分分钟帮你搞定！

今天我们要学习python里的**re模块和正则表达式**，学会了这个就可以帮我们解决刚刚的疑问。正则表达式不仅在python领域，在整个编程届都占有举足轻重的地位。

```
不管以后你是不是去做python开发，只要你是一个程序员就应该了解正则表达式的基本使用。如果未来你要在爬虫领域发展，你就更应该
但是你要知道，re模块本质上和正则表达式没有一毛钱的关系。re模块和正则表达式的关系 类似于 time模块和时间的关系
你没有学习python之前，也不知道有一个time模块，但是你已经认识时间了 12:30就表示中午十二点半（这个时间可好，一般这会儿就
时间有自己的格式，年月日时分秒，12个月，365天.....已经成为了一种规则。你也早就牢记于心了。time模块只不过是python提供
```

**正则表达式本身也和python没有什么关系，就是匹配字符串内容的一种规则。**

官方定义：正则表达式是对字符串操作的一种逻辑公式，就是用事先定义好的一些特定字符、及这些特定字符的组合，来匹配符合一定规则的字符串。

## 正则表达式

一说规则我已经知道你很晕了，现在就让我们先来看一些实际的应用。[在线测试工具](http://tool.chinaz.com/regex/) <http://tool.chinaz.com/regex/>

首先你要知道的是，谈到正则，就只和字符串相关了。在我给你提供的工具中，你输入的每一个字都是一个字符串。  
其次，如果在一个位置的一个值，不会出现什么变化，那么是不需要规则的。  
比如你要用“1”去匹配“1”，或者用“2”去匹配“2”，直接就可以匹配上。这连python的字符串操作都可以轻松做到。  
那么在之后我们更多要考虑的是在 同一个位置上 可以出现的字符的范围。

```
字符组 ： [字符组]
在同一个位置可能出现的所有字符组成了一个字符组，在正则表达式中用[]表示
字符分为很多类，比如数字、字母、标点等等。
假如你现在要求一个位置“只能出现一个数字”，那么这个位置上的字符只能是0、1、2...9这10个数之一。
```

正则	待匹配字符	匹配结果	说明
[0123456789]	8	True	在一个字符组里枚举合法的所有字符，字符组里的任意一个字符和“待匹配字符”相同都视为可以匹配
[0123456789]	a	False	由于字符组中没有“a”字符，所以不能匹配
[0-9]	7	True	也可以用-表示范围，[0-9]就和[0123456789]是一个意思
[a-z]	s	True	同样的如果要匹配所有的小写字母，直接用[a-z]就可以表示
[A-Z]	B	True	[A-Z]就表示所有的大写字母
[0-9a-fA-F]	e	True	可以匹配数字，大小写形式的a~f，用来验证十六进制字符

字符：

元字符	匹配内容
.	匹配除换行符以外的任意字符
\w	匹配字母或数字或下划线
\s	匹配任意的空白符
\d	匹配数字
\n	匹配一个换行符
\t	匹配一个制表符
\b	匹配一个单词的结尾
^	匹配字符串的开始
\$	匹配字符串的结尾
\W	匹配非字母或数字或下划线
\D	匹配非数字
\S	匹配非空白符
a b	匹配字符a或字符b
()	匹配括号内的表达式，也表示一个组
[...]	匹配字符组中的字符
[^...]	匹配除了字符组中字符的所有字符

量词：

量词	用法说明
*	重复零次或更多次
+	重复一次或更多次
?	重复零次或一次
{n}	重复n次
{n,}	重复n次或更多次
{n,m}	重复n到m次

. ^ \$

正则	待匹配字符	匹配结果	说明
海.	海燕海娇海东	海燕海娇海东	匹配所有"海."的字符
^海.	海燕海娇海东	海燕	只从开头匹配"海."
海.\$	海燕海娇海东	海东	只匹配结尾的"海.\$"

\* + ? { }

正则	待匹配字符	匹配	说明
----	-------	----	----

		结果	
李.?	李杰和李莲英 和李二棍子	李杰 李莲 李二	?表示重复零次或一次，即只匹配“李”后面一个任意字符
李.*	李杰和李莲英 和李二棍子	李杰和李莲英 和李二棍子	*表示重复零次或多次，即匹配“李”后面0或多个任意字符
李.+	李杰和李莲英 和李二棍子	李杰和李莲英 和李二棍子	+表示重复一次或多次，即只匹配“李”后面1个或多个任意字符
李. {1,2}	李杰和李莲英 和李二棍子	李杰和 李莲英 李二棍	{1,2} 匹配1到2次任意字符

注意：前面的\*,+,?等都是贪婪匹配，也就是尽可能匹配，后面加?号使其变成惰性匹配

正则	待匹配字符	匹配 结果	说明
李.*?	李杰和李莲英和李二棍子	李 李 李	惰性匹配

字符集 [] [^]

正则	待匹配字符	匹配 结果	说明
李[杰莲英二棍子]*	李杰和李莲英和李二棍子	李杰 李莲英 李二棍子	表示匹配“李”字后面[杰莲英二棍子]的字符任意次
李[^和]*	李杰和李莲英和李二棍子	李杰 李莲英 李二棍子	表示匹配一个不是“和”的字符任意次
[\d]	456bdha3	4 5 6 3	表示匹配任意一个数字，匹配到4个结果
[\d]+	456bdha3	456 3	表示匹配任意个数字，匹配到2个结果

分组 ()与 或 | [^]

身份证号码是一个长度为15或18个字符的字符串，如果是15位则全部由数字组成，首位不能为0；如果是18位，则前17位全部是数字，末位可能是数字或x，下面我们尝试用正则来表示：

正则	待匹配 字符	匹配 结果	说明
^[1-9]\d{13,16}[0-9x]\$	110101198001017032	110101198001017032	表示可以匹配一个正确的身份证号

<code>^[1-9]\d{13,16}[0-9x]\$</code>	<code>1101011980010170</code>	<code>1101011980010170</code>	表示也可以匹配这串数字，但这并不是一个正确的身份证号码，它是一个16位的数字
<code>^[1-9]\d{14}(\d{2}[0-9x])?\$</code>	<code>1101011980010170</code>	<code>False</code>	现在不会匹配错误的身份证号了 ( ) 表示分组，将 <code>\d{2}[0-9x]</code> 分成一组，就可以整体约束他们出现的次数为0-1次
<code>^([1-9]\d{16}[0-9x] 1[1-9]\d{14})\$</code>	<code>1101015199812067023</code>	<code>1101015199812067023</code>	表示先匹配 <code>[1-9]\d{16}[0-9x]</code> 如果没有匹配上就匹配 <code>[1-9]\d{14}</code>

转义符 \

在正则表达式中，有很多有特殊意义的是元字符，比如 `\n` 和 `\s` 等，如果要在正则中匹配正常的 `"\n"` 而不是 "换行符" 就需要对 `"\"` 进行转义，变成 `\"`。

在python中，无论是正则表达式，还是待匹配的内容，都是以字符串的形式出现的，在字符串中 `\` 也有特殊的含义，本身还需要转义。所以如果匹配一次 `"\n"`，字符串中要写成 `\"n'`，那么正则里就要写成 `\"\\n"`，这样就太麻烦了。这个时候我们就用到了 `r\"n'` 这个概念，此时的正则 `r\"n'` 就可以了。

正则	待匹配字符	匹配结果	说明
<code>\n</code>	<code>\n</code>	<code>False</code>	因为在正则表达式中 <code>\</code> 是有特殊意义的字符，所以要匹配 <code>\n</code> 本身，用表达式 <code>\n</code> 无法匹配
<code>\\n</code>	<code>\n</code>	<code>True</code>	转义 <code>\</code> 之后变成 <code>\\</code> ，即可匹配
<code>"\\n"</code>	<code>\"n'</code>	<code>True</code>	如果在python中，字符串中的 <code>'\"</code> 也需要转义，所以每一个字符串 <code>'\"</code> 又需要转义成 <code>\"n'</code>
<code>r\"n'</code>	<code>r\"n'</code>	<code>True</code>	在字符串之前加 <code>r</code> ，让整个字符串不转义

贪婪匹配

贪婪匹配：在满足匹配时，匹配尽可能长的字符串，默认情况下，采用贪婪匹配

正则	待匹配字符	匹配结果	说明
<code>&lt;.*&gt;</code>	<code>&lt;script&gt;...&lt;script&gt;</code>	<code>&lt;script&gt;...&lt;script&gt;</code>	默认为贪婪匹配模式，会匹配尽量长的字符串
<code>&lt;.*?&gt;</code>	<code>r\"d'</code>	<code>&lt;script&gt;&lt;script&gt;</code>	加上 <code>?</code> 为将贪婪匹配模式转为非贪婪匹配模式，会匹配尽量短的字符串

几个常用的非贪婪匹配Pattern

<code>*?</code>	重复任意次，但尽可能少重复
<code>+?</code>	重复1次或更多次，但尽可能少重复
<code>??</code>	重复0次或1次，但尽可能少重复
<code>{n,m}?</code>	重复n到m次，但尽可能少重复
<code>{n,}??</code>	重复n次以上，但尽可能少重复

. \*? 的用法



. 是任意字符  
\* 是取 0 至 无限长度  
? 是非贪婪模式。  
何在一起就是 取尽量少的任意字符，一般不会这么单独写，他大多用在：  
. \* ? x

就是取前面任意长度的字符，直到一个x出现



## re模块下的常用方法



```
import re

ret = re.findall('a', 'eva egon yuan') # 返回所有满足匹配条件的结果,放在列表里
print(ret) #结果 : ['a', 'a']

ret = re.search('a', 'eva egon yuan').group()
print(ret) #结果 : 'a'
# 函数会在字符串中查找模式匹配,只到找到第一个匹配然后返回一个包含匹配信息的对象,该对象可以
# 通过调用group()方法得到匹配的字符串,如果字符串没有匹配,则返回None。

ret = re.match('a', 'abc').group() # 同search,不过尽在字符串开始处进行匹配
print(ret)
#结果 : 'a'

ret = re.split('[ab]', 'abcd') # 先按'a'分割得到''和'bcd',在对''和'bcd'分别按'b'分割
print(ret) # ['', '', 'cd']

ret = re.sub('\d', 'H', 'eva3egon4yuan4', 1) #将数字替换成'H', 参数1表示只替换1个
print(ret) #evaHegon4yuan4

ret = re.subn('\d', 'H', 'eva3egon4yuan4') #将数字替换成'H', 返回元组(替换的结果,替换了多少次)
print(ret)

obj = re.compile('\d{3}') #将正则表达式编译成为一个 正则表达式对象,规则要匹配的是3个数字
ret = obj.search('abc123eeee') #正则表达式对象调用search, 参数为待匹配的字符串
print(ret.group()) #结果 : 123

import re
ret = re.finditer('\d', 'ds3sy4784a') #finditer返回一个存放匹配结果的迭代器
print(ret) # <callable_iterator object at 0x10195f940>
print(next(ret).group()) #查看第一个结果
print(next(ret).group()) #查看第二个结果
print([i.group() for i in ret]) #查看剩余的左右结果
```



注意:

### 1 findall的优先级查询:



```
import re

ret = re.findall('www.(baidu|oldboy).com', 'www.oldboy.com')
print(ret) # ['oldboy'] 这是因为findall会优先把匹配结果组里内容返回,如果想要匹配结果,取消权限即可

ret = re.findall('www.(?:baidu|oldboy).com', 'www.oldboy.com')
print(ret) # ['www.oldboy.com']
```



### 2 split的优先级查询



```
ret=re.split("\d+", "eva3egon4yuan")
print(ret) #结果 : ['eva', 'egon', 'yuan']
```

```
ret=re.split("(\\d+)", "eva3egon4yuan")
print(ret) #结果 : ['eva', '3', 'egon', '4', 'yuan']

#在匹配部分加上 () 之后所切出的结果是不同的,
#没有 () 的没有保留所匹配的项, 但是有 () 的却能够保留了匹配的项,
#这个在某些需要保留匹配部分的使用过程是非常重要的。
```

综合练习与扩展

1、匹配标签

+

View Code

2、匹配整数

+

View Code

3、数字匹配

+

View Code

4、爬虫练习

+

View Code

+

简化版

+

flags

作业

```
实现能计算类似
1 - 2 * ( (60-30 +(-40/5) * (9-2*5/3 + 7 /3*99/4*2998 +10 * 568/14 )) - (-4*3)/ (16-3*2) )等类似公式的计
```

在线测试工具 <http://tool.chinaz.com/regex/>

常用模块二

[返回顶部](#)

hashlib模块

算法介绍

Python的hashlib提供了常见的摘要算法，如MD5，SHA1等等。

什么是摘要算法呢？摘要算法又称哈希算法、散列算法。它通过一个函数，把任意长度的数据转换为一个长度固定的数据串（通常用16进制的字符串表示）。

摘要算法就是通过摘要函数f()对任意长度的数据data计算出固定长度的摘要digest，目的是为了发现原始数据是否被人篡改过。

摘要算法之所以能指出数据是否被篡改过，就是因为摘要函数是一个单向函数，计算f(data)很容易，但通过digest反推data却非常困难。而且，对原始数据做一个bit的修改，都会导致计算出的摘要完全不同。

我们以常见的摘要算法MD5为例，计算出一个字符串的MD5值：

```
import hashlib
```

```
md5 = hashlib.md5()
md5.update('how to use md5 in python hashlib?')
print md5.hexdigest()
```

计算结果如下：  
d26a53750bc40b38b65a520292f69306



如果数据量很大，可以分块多次调用update()，最后计算的结果是一样的：

```
md5 = hashlib.md5()
md5.update('how to use md5 in ')
md5.update('python hashlib?')
print md5.hexdigest()
```

MD5是最常见的摘要算法，速度很快，生成结果是固定的128 bit字节，通常用一个32位的16进制字符串表示。另一种常见的摘要算法是SHA1，调用SHA1和调用MD5完全类似：

```
import hashlib

sha1 = hashlib.sha1()
sha1.update('how to use sha1 in ')
sha1.update('python hashlib?')
print sha1.hexdigest()
```

SHA1的结果是160 bit字节，通常用一个40位的16进制字符串表示。比SHA1更安全的算法是SHA256和SHA512，不过越安全的算法越慢，而且摘要长度更长。

### 摘要算法应用

任何允许用户登录的网站都会存储用户登录的用户名和口令。如何存储用户名和口令呢？方法是存到数据库表中：

name	password
-----+	-----
michael	123456
bob	abc999
alice	alice2008

如果以明文保存用户口令，如果数据库泄露，所有用户的口令就落入黑客的手里。此外，网站运维人员是可以访问数据库的，也就是能获取到所有用户的口令。正确的保存口令的方式是不存储用户的明文口令，而是存储用户口令的摘要，比如MD5：

username	password
-----+	-----
michael	e10adc3949ba59abbe56e057f20f883e
bob	878ef96e86145580c38c87f0410ad153
alice	99b1c2188db85afee403b1536010c2c9

考虑这么个情况，很多用户喜欢用123456，888888，password这些简单的口令，于是，黑客可以事先计算出这些常用口令的MD5值，得到一个反推表：

```
'e10adc3949ba59abbe56e057f20f883e': '123456'
'21218cca77804d2ba1922c33e0151105': '888888'
'5f4dcc3b5aa765d61d8327deb882cf99': 'password'
```

这样，无需破解，只需要对比数据库的MD5，黑客就获得了使用常用口令的用户账号。

对于用户来讲，当然不要使用过于简单的口令。但是，我们能否在程序设计上对简单口令加强保护呢？

由于常用口令的MD5值很容易被计算出来，所以，要确保存储的用户口令不是那些已经被计算出来的常用口令的MD5，这一方法通过对原始口令加一个复杂字符串来实现，俗称“加盐”：

```
hashlib.md5("salt".encode("utf8"))
```

经过Salt处理的MD5口令，只要Salt不被黑客知道，即使用户输入简单口令，也很难通过MD5反推明文口令。

但是如果有两个用户都使用了相同的简单口令比如123456，在数据库中，将存储两条相同的MD5值，这说明这两个用户的口令是一样的。有没有办法让使用相同口令的用户存储不同的MD5呢？

如果假定用户无法修改登录名，就可以通过把登录名作为Salt的一部分来计算MD5，从而实现相同口令的用户也存储不同的MD5。


摘要算法在很多地方都有广泛的应用。要注意摘要算法不是加密算法，不能用于加密（因为无法通过摘要反推明文），只能用于防篡改，但是它的单向计算特性决定了可以在不存储明文口令的情况下验证用户口令。

## configparser模块

该模块适用于配置文件的格式与windows ini文件类似，可以包含一个或多个节（section），每个节可以有多个参数（键=值）。

### 创建文件


来看一个好多软件的常见文档格式如下：



```
[DEFAULT]
ServerAliveInterval = 45
Compression = yes
CompressionLevel = 9
ForwardX11 = yes

[bitbucket.org]
User = hg

[topsecret.server.com]
Port = 50022
ForwardX11 = no
```



如果想用python生成一个这样的文档怎么做呢？



```
import configparser

config = configparser.ConfigParser()

config["DEFAULT"] = {'ServerAliveInterval': '45',
                     'Compression': 'yes',
                     'CompressionLevel': '9',
                     'ForwardX11': 'yes'}

config['bitbucket.org'] = {'User': 'hg'}

config['topsecret.server.com'] = {'Host Port': '50022', 'ForwardX11': 'no'}

with open('example.ini', 'w') as configfile:

    config.write(configfile)
```



### 查找文件



```
import configparser

config = configparser.ConfigParser()

#-----查找文件内容, 基于字典的形式

print(config.sections())      #  []

config.read('example.ini')

print(config.sections())      #  ['bitbucket.org', 'topsecret.server.com']

print('bytebong.com' in config) # False
print('bitbucket.org' in config) # True

print(config['bitbucket.org']['user']) # hg

print(config['DEFAULT']['Compression']) #yes

print(config['topsecret.server.com']['ForwardX11']) #no
```



```
print(config['bitbucket.org'])           #<Section: bitbucket.org>

for key in config['bitbucket.org']:      # 注意,有default会默认default的键
    print(key)

print(config.options('bitbucket.org'))  # 同for循环,找到'bitbucket.org'下所有键

print(config.items('bitbucket.org'))    #找到'bitbucket.org'下所有键值对

print(config.get('bitbucket.org','compression')) # yes      get方法Section下的key对应的value
```

## 增删改操作

```
import configparser

config = configparser.ConfigParser()

config.read('example.ini')

config.add_section('yuan')

config.remove_section('bitbucket.org')
config.remove_option('topsecret.server.com','forwardx11')

config.set('topsecret.server.com','k1','11111')
config.set('yuan','k2','22222')

config.write(open('new2.ini', 'w'))
```

[返回顶部](#)

## logging模块

### 函数式简单配置

```
import logging
logging.debug('debug message')
logging.info('info message')
logging.warning('warning message')
logging.error('error message')
logging.critical('critical message')
```

默认情况下Python的logging模块将日志打印到了标准输出中，且只显示了大于等于WARNING级别的日志，这说明默认的日志级别设置为WARNING（日志级别等级CRITICAL > ERROR > WARNING > INFO > DEBUG），默认的日志格式为日志级别：Logger名称：用户输出消息。

**灵活配置日志级别，日志格式，输出位置：**

```
import logging

file_handler = logging.FileHandler(filename='x1.log', mode='a', encoding='utf-8',)
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(module)s: %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S %p',
    handlers=[file_handler,],
    level=logging.ERROR
)

logging.error('你好')
```

## 日志切割



```
import time
import logging
from logging import handlers

sh = logging.StreamHandler()
rh = handlers.RotatingFileHandler('myapp.log', maxBytes=1024,backupCount=5)
fh = handlers.TimedRotatingFileHandler(filename='x2.log', when='s', interval=5, encoding='utf-8')
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(module)s: %(message)s',
    datefmt='%Y-%m-%d %H:%M:%S %p',
    handlers=[fh,sh,rh],
    level=logging.ERROR
)

for i in range(1,100000):
    time.sleep(1)
    logging.error('KeyboardInterrupt error %s'%str(i))
```



配置参数:

[View Code](#)

## logger对象配置



```
import logging

logger = logging.getLogger()
# 创建一个handler, 用于写入日志文件
fh = logging.FileHandler('test.log',encoding='utf-8')

# 再创建一个handler, 用于输出到控制台
ch = logging.StreamHandler()
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
fh.setLevel(logging.DEBUG)

fh.setFormatter(formatter)
ch.setFormatter(formatter)
logger.addHandler(fh) #logger对象可以添加多个fh和ch对象
logger.addHandler(ch)

logger.debug('logger debug message')
logger.info('logger info message')
logger.warning('logger warning message')
logger.error('logger error message')
logger.critical('logger critical message')
```



logging库提供了多个组件: Logger、Handler、Filter、Formatter。Logger对象提供应用程序可直接使用的接口,Handler发送日志到适当的目的地,Filter提供了过滤日志信息的方法,Formatter指定日志显示格式。另外,可以通过: logger.setLevel(logging.Debug)设置级别,当然,也可以通过

fh.setLevel(logging.Debug)单对文件流设置某个级别。

分类: [python之路](#)

好文要顶

已关注

收藏该文



Eva\_J

关注 - 7

粉丝 - 4226

[我在关注他](#) [取消关注](#)

38

0

#1楼 2017-08-05 21:36 | Jokerbj

回复 引用

支持(7) 反对(10)

回复 引用

支持(2) 反对(1)

回复 引用

支持(1) 反对(1)

回复 引用

支持(0) 反对(1)

回复 引用

支持(2) 反对(1)

回复 引用

支持(0) 反对(1)

回复 引用

支持(3) 反对(0)

回复 引用

支持(1) 反对(1)

回复 引用

支持(0) 反对(1)

回复 引用

支持(0) 反对(1)

#11楼	2019-03-27 10:35   <a href="#">baozhilv</a>	<a href="#">回复</a> <a href="#">引用</a>
我现在用的是 python3.7 对于 dict 好像已经有 OrderedDict 的特性了		
		支持(0) 反对(1)
#12楼	2019-03-27 14:50   <a href="#">baozhilv</a>	<a href="#">回复</a> <a href="#">引用</a>
Python 3.X 里不包含 has_key() 函数, 被 __contains__(key) 替代:		
<pre>dict3 = {'name':'z','Age':7,'class':'First'}; print("Value : ",dict3.__contains__('name')) print("Value : ",dict3.__contains__('sex'))</pre>		
执行结果:		
Value : True Value : False		
		支持(0) 反对(1)
#13楼	2019-04-18 12:15   <a href="#">托小尼</a>	<a href="#">回复</a> <a href="#">引用</a>
md5在不同电脑加密文件内容结果不一致 定义一个txt文件, 内容为三行数字, 111 222 333 使用 import hashlib md5 = hashlib.md5() with open('../txt.txt', 'rb') as f: for line in f: md5.update(line) print(md5.hexdigest()) 本机win上执行结果为 063eadeb62dc63a624c1594ac7e60b44 同样的文件同样的代码, 在另外一台linux上执行结果却不一样, 两台win没测试过。有人遇到过吗?		
		支持(0) 反对(2)
#14楼	2019-05-10 15:41   <a href="#">王希知</a>	<a href="#">回复</a> <a href="#">引用</a>
写了一天终于把正则的题写出来了, 我可是学了6个月了啊, 速度还是这么慢, 能找到工作吗?		
<a href="#">+ View Code</a>		
		支持(0) 反对(1)
#15楼	2019-11-22 11:07   <a href="#">我是风儿</a>	<a href="#">回复</a> <a href="#">引用</a>
老师那个豆瓣已经发现了,进行了反扒了		
		支持(1) 反对(1)
#16楼	2020-04-10 17:43   <a href="#">jeason_hui</a>	<a href="#">回复</a> <a href="#">引用</a>
@王希知 我也是, 最后计算的发现和eval()计算出来的差5, 和eval相差二十多万分之一, 我感觉可能有错误, 看你的和我算的一样, 应该是正常误差吧?		
		支持(0) 反对(0)
#17楼	2020-04-10 17:47   <a href="#">jeason_hui</a>	<a href="#">回复</a> <a href="#">引用</a>
1 - 2 * ( (60-30 +(9-25/3 + 7 /399/42998 +10 * 568/14 )(-40/5) ) - (-43)/ (16-32) ) 这道计算题, 算了有两天了, 太花时间了, 还有就是有课, 最后算出来的和eval()算的误差在5左右, 相差二十多万分之一, 老师, 这算计算机的误差函数, 函数我们算法的错误, 你教的同学算的都和eval一样吗?		
		支持(0) 反对(0)
#18楼	[楼主] 2020-04-13 14:00   <a href="#">Eva_J</a>	<a href="#">回复</a> <a href="#">引用</a>
@jeason_hui 正常和eval算出来的结果应该是一致的。		
		支持(0) 反对(0)
#19楼	2020-08-09 14:20   <a href="#">diracy</a>	<a href="#">回复</a> <a href="#">引用</a>
@jeason_hui 当然一样啦		
		支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

发表评论

编辑

预览

B

支持 Markdown

提交评论

退出 订阅评论

[Ctrl+Enter快捷键提交]

- 【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库
- 【推荐】1200件T恤+6万奖金，阿里云编程大赛报名开启
- 【推荐】了不起的开发者，挡不住的华为，园子里的品牌专区
- 【推荐】如何在面试中成长？来看阿里前端终面官的面试心得

相关博文：

- Python之路-Python常用模块-time模块
- python之路----常用模块二
- Python之路-python（常用模块学习）
- python之路5：常用模块
- python之路(五)--常用模块
- » 更多推荐...

最新 IT 新闻：

- 大陆芯片自给率要达70%，台湾半导体人才或有外流潮
- 蔚来“车电分离”落地 李斌：电池服务费5年内不调价
- 特斯拉Autopilot及其他驾驶辅助套件将在瑞典扩展测试
- 半年内两次遭遇海外专利诉讼，“绊住”小米的H.265到底是什么？
- 华为机器狗曝光，将用于智能识别和目标定位等场景
- » 更多新闻...