



Python 之路 Day5 - 常用模块学习

本节大纲：

- 1. 模块介绍
- 2. time &datetime模块
- 3. random
- 4. os
- 5. sys
- 6. shutil
- 7. json & picle
- 8. shelve
- 9. xml处理
- 10. yaml处理
- 11. configparser
- 12. hashlib
- 13. subprocess
- 14. logging模块
- 15. re正则表达式

模块，用一坨代码实现了某个功能的代码集合。

类似于函数式编程和面向过程编程，函数式编程则完成一个功能，其他代码用来调用即可，提供了代码的重用性和代码间的耦合。而对于一个复杂的功能来，可能需要多个函数才能完成（函数又可以在不同的.py文件中），n个.py文件组成的代码集合就称为模块。

如：os 是系统相关的模块；file是文件操作相关的模块

模块分为三种：

- 自定义模块
- 内置标准模块（又称标准库）
- 开源模块

自定义模块 和开源模块的使用参

考 <http://www.cnblogs.com/wupeiqi/articles/4963027.html>

time & datetime模块

View Code

Direct ive	Meaning	Notes
%a	Locale’s abbreviated weekday name.	
%A	Locale’s full weekday name.	
%b	Locale’s abbreviated month name.	
%B	Locale’s full month name.	
%c	Locale’s appropriate date and time representation.	
%d	Day of the month as a decimal number [01,31].	

公告

最新自学视频 路飞Python免费小课
人生迷茫问题可以加Alex个人微信听鸡



汤



面向对象开发原来如此简单
16人在学 进阶 12小时



Python开发中最常用的11个模块精讲
10人在学 进阶 6小时



跟随Alex金角大王3周上手Python开发
124人在学 入门 19小时

昵称： 金角大王
园龄： 5年5个月
粉丝： 10868
关注： 5
+加关注

我的标签

职业发展(3)
创业(2)

随笔分类

职业&生活随笔(22)

文章分类

Python全栈开发之路(12)
Python学习目录(4)
Python自动化开发之路(33)
爬虫(6)

最新评论

1. Re:编程要自学或报班这事你都想不明白, 那必然是你智商不够
大王性格直爽, 有目标, 肯付出行动! 偶像! 我要是早几年遇见大王就好了, 现在都已经三十了, 浪费了大好的青春!

--Xiyue666

2. Re:python 之路, Day11 - python mysql and ORM
ALTER mytable ADD INDEX
[indexName] ON (username(length))
这句应改为: alter table mytable add index index...

--原竹

3. Re:Python之路,Day3 - Python基础3
@我的恋人叫臭臭 淫角大王听说很厉害吧? ...

--Xiyue666

4. Re:Python之路,Day1 - Python基础1
哎 我为什么不早点遇到老男孩 遇到alex 老师呢!

--Xiyue666

5. Re:Python 之路 Day5 - 常用模块学习
#思路: 过滤最里面的 () 将里面的数值计算后替换到原有公示字符串, 直到将所有的括号剔除后再计算没有括号的字符串。
import re def mul_div(num_str): #先算乘除, (num_s...

--编程届的小学生

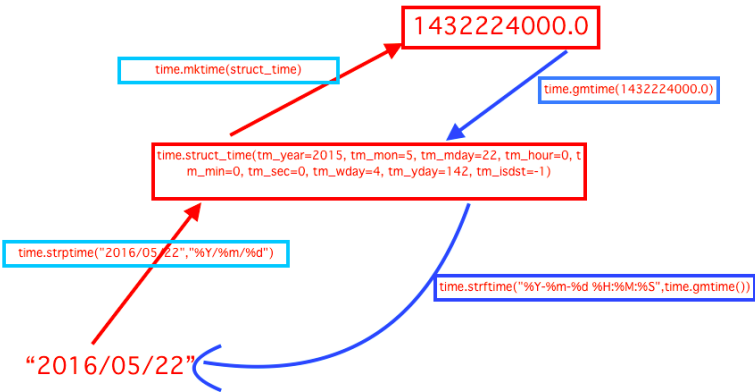
网友排行榜

1. python 之路, 200行Python代码写了个打飞机游戏! (52327)
2. Django + Uwsgi + Nginx 实现生产环境部署(31747)
3. Python Select 解析(27145)
4. 为什么很多IT公司不喜欢进过培训机构的人呢? (20510)
5. 编程要自学或报班这事你都想不明白, 那必然是你智商不够(16904)

推荐排行榜

1. 给一位做技术迷茫的同学回信(63)
2. 你做了哪些事, 导致老板下调了对你的评价? (51)
3. 关于认识、格局、多维度发展的感触(46)
4. 为什么很多IT公司不喜欢进过培训机构的人呢? (37)
5. 编程要自学或报班这事你都想不明白, 那必然是你智商不够(35)

Directive	Meaning	Notes
%H	Hour (24-hour clock) as a decimal number [00,23].	
%I	Hour (12-hour clock) as a decimal number [01,12].	
%j	Day of the year as a decimal number [001,366].	
%m	Month as a decimal number [01,12].	
%M	Minute as a decimal number [00,59].	
%p	Locale's equivalent of either AM or PM.	(1)
%S	Second as a decimal number [00,61].	(2)
%U	Week number of the year (Sunday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Sunday are considered to be in week 0.	(3)
%w	Weekday as a decimal number [0(Sunday),6].	
%W	Week number of the year (Monday as the first day of the week) as a decimal number [00,53]. All days in a new year preceding the first Monday are considered to be in week 0.	(3)
%x	Locale's appropriate date representation.	
%X	Locale's appropriate time representation.	
%y	Year without century as a decimal number [00,99].	
%Y	Year with century as a decimal number.	
%z	Time zone offset indicating a positive or negative time difference from UTC/GMT of the form +HHMM or -HHMM, where H represents decimal hour digits and M represents decimal minute digits [-23:59, +23:59].	
%Z	Time zone name (no characters if no time zone exists).	
%%	A literal '%' character.	



时间 转换关系 图, by ALEX, OLDBOY PY

随机数

```

1 import random
2 print random.random()
3 print random.randint(1,2)
4 print random.randrange(1,10)

```

生成随机验证码

```

1 import random
2 checkcode = ''
3 for i in range(4):
4     current = random.randrange(0,4)
5     if current != i:
6         temp = chr(random.randint(65,90))
7     else:
8         temp = random.randint(0,9)
9     checkcode += str(temp)
10 print checkcode

```

OS模块

提供对操作系统进行调用的接口

```

1 os.getcwd() 获取当前工作目录，即当前python脚本工作的目录路径
2 os.chdir("dirname") 改变当前脚本工作目录；相当于shell下cd
3 os.curdir 返回当前目录：('.')
4 os.pardir 获取当前目录的父目录字符串名：('..')
5 os.makedirs('dirname1/dirname2') 可生成多层递归目录
6 os.removedirs('dirname1') 若目录为空，则删除，并递归到上一级目录，如若也为空，则删除，依此类推
7 os.mkdir('dirname') 生成单级目录；相当于shell中mkdir dirname
8 os.rmdir('dirname') 删除单级空目录，若目录不为空则无法删除，报错；相当于shell中rmdir dirname
9 os.listdir('dirname') 列出指定目录下的所有文件和子目录，包括隐藏文件，并以列表方式打印
10 os.remove() 删除一个文件
11 os.rename("oldname","newname") 重命名文件/目录
12 os.stat('path/filename') 获取文件/目录信息
13 os.sep 输出操作系统特定的路径分隔符，win下为"\\",Linux下为"/"
14 os.linesep 输出当前平台使用的行终止符，win下为"\t\n",Linux下为"\n"
15 os.pathsep 输出用于分割文件路径的字符串
16 os.name 输出字符串指示当前使用平台。win->'nt'; Linux->'posix'
17 os.system("bash command") 运行shell命令，直接显示
18 os.environ 获取系统环境变量
19 os.path.abspath(path) 返回path规范化的绝对路径
20 os.path.split(path) 将path分割成目录和文件名二元组返回
21 os.path.dirname(path) 返回path的目录。其实就是os.path.split(path)的第一个元素
22 os.path.basename(path) 返回path最后的文件名。如何path以/或\结尾，那么就会返回空值。即os.path.split(path)的第二个元素
23 os.path.exists(path) 如果path存在，返回True；如果path不存在，返回False
24 os.path.isabs(path) 如果path是绝对路径，返回True
25 os.path.isfile(path) 如果path是一个存在的文件，返回True。否则返回False
26 os.path.isdir(path) 如果path是一个存在的目录，则返回True。否则返回False
27 os.path.join(path1[, path2[, ...]]) 将多个路径组合后返回，第一个绝对路径之前的参数将被忽略
28 os.path.getatime(path) 返回path所指向的文件或者目录的最后存取时间
29 os.path.getmtime(path) 返回path所指向的文件或者目录的最后修改时间

```

更多猛击[这里](#)

sys模块

```

1 sys.argv 命令行参数List，第一个元素是程序本身路径
2 sys.exit(n) 退出程序，正常退出时exit(0)
3 sys.version 获取Python解释程序的版本信息
4 sys.maxint 最大的Int值
5 sys.path 返回模块的搜索路径，初始化时使用PYTHONPATH环境变量的值
6 sys.platform 返回操作系统平台名称
7 sys.stdout.write('please:')
8 val = sys.stdin.readline()[:-1]

```

shutil模块

直接参考 <http://www.cnblogs.com/wupeiqi/articles/4963027.html>

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

Json模块提供了四个功能: dumps、dump、loads、load

pickle模块提供了四个功能: dumps、dump、loads、load

```
import pickle

data = {'k1':123,'k2':'Hello'}

#pickle.dumps 将数据通过特殊的形式转换为只有python语言认识的字符串
p_str = pickle.dumps(data)
print p_str

#pickle.dump 将数据通过特殊的形式转换为只有python语言认识的字符串，并写入文件
with open('D:/result.pk', 'w') as fp:
    pickle.dump(data,fp)

import json
#json.dumps 将数据通过特殊的形式转换为所有程序语言都认识的字符串
j_str = json.dumps(data)
print j_str

#json.dump 将数据通过特殊的形式转换为所有程序语言都认识的字符串，并写入文件
with open('D:/result.json', 'w') as fp:
    json.dump(data,fp)
```

shelve 模块

shelve模块是一个简单的k,v将内存数据通过文件持久化的模块，可以持久化任何pickle可支持的python数据格式

```
1 import shelve
2
3 d = shelve.open('shelve_test') #打开一个文件
4
5 class Test(object):
6     def __init__(self,n):
7         self.n = n
8
9
10 t = Test(123)
11 t2 = Test(123334)
12
13 name = ["alex","rain","test"]
14 d["test"] = name #持久化列表
15 d["t1"] = t      #持久化类
16 d["t2"] = t2
17
18 d.close()
```

xml处理模块

xml是实现不同语言或程序之间进行数据交换的协议，跟json差不多，但json使用起来更简单，不过，古时候，在json还没诞生的黑暗年代，大家只能选择用xml呀，至今很多传统公司如金融行业的很多系统的接口还主要是xml。

xml的格式如下，就是通过<>节点来区别数据结构的：

[+ View Code](#)

xml协议在各个语言里的都是支持的，在python中可以用以下模块操作xml

```

1 import xml.etree.ElementTree as ET
2
3 tree = ET.parse("xmltest.xml")
4 root = tree.getroot()
5 print(root.tag)
6
7 #遍历xml文档
8 for child in root:
9     print(child.tag, child.attrib)
10     for i in child:
11         print(i.tag,i.text)
12
13 #只遍历year 节点
14 for node in root.iter('year'):
15     print(node.tag,node.text)

```

修改和删除xml文档内容

[+ View Code](#)

自己创建xml文档

```

1 import xml.etree.ElementTree as ET
2
3
4 new_xml = ET.Element("namelist")
5 name = ET.SubElement(new_xml,"name",attrib={"enrolled":"yes"})
6 age = ET.SubElement(name,"age",attrib={"checked":"no"})
7 sex = ET.SubElement(name,"sex")
8 sex.text = '33'
9 name2 = ET.SubElement(new_xml,"name",attrib={"enrolled":"no"})
10 age = ET.SubElement(name2,"age")
11 age.text = '19'
12
13 et = ET.ElementTree(new_xml) #生成文档对象
14 et.write("test.xml", encoding="utf-8",xml_declaration=True)
15
16 ET.dump(new_xml) #打印生成的格式

```

PyYAML模块

Python也可以很容易的处理yaml文档格式，只不过需要安装一个模块，参考文档：

<http://pyyaml.org/wiki/PyYAMLDocumentation>

ConfigParser模块

用于生成和修改常见配置文档，当前模块的名称在 python 3.x 版本中变更为 configparser。

来看一个好多软件的常见文档格式如下

```

1 [DEFAULT]
2 ServerAliveInterval = 45
3 Compression = yes
4 CompressionLevel = 9
5 ForwardX11 = yes
6
7 [bitbucket.org]
8 User = hg
9
10 [topsecret.server.com]
11 Port = 50022
12 ForwardX11 = no

```

如果想用python生成一个这样的文档怎么做呢？

```

1 import configparser
2
3 config = configparser.ConfigParser()
4 config["DEFAULT"] = {'ServerAliveInterval': '45',
5                      'Compression': 'yes',
6                      'CompressionLevel': '9'}
7

```

```

8 | config['bitbucket.org'] = {}
9 | config['bitbucket.org']['User'] = 'hg'
10 | config['topsecret.server.com'] = {}
11 | topsecret = config['topsecret.server.com']
12 | topsecret['Host Port'] = '50022'      # mutates the parser
13 | topsecret['ForwardX11'] = 'no' # same here
14 | config['DEFAULT']['ForwardX11'] = 'yes'
15 | with open('example.ini', 'w') as configfile:
16 |     config.write(configfile)

```

写完了还可以再读出来哈。

[+ View Code](#)

configparser增删改查语法

[+ View Code](#)

hashlib模块

用于加密相关的操作，3.x里代替了md5模块和sha模块，主要提供 SHA1, SHA224, SHA256, SHA384, SHA512, MD5 算法

[+ View Code](#)

还不够吊？python 还有一个 hmac 模块，它内部对我们创建 key 和 内容 再进行处理然后再加密

散列消息鉴别码，简称HMAC，是一种基于消息鉴别码MAC（Message Authentication Code）的鉴别机制。使用HMAC时，消息通讯的双方，通过验证消息中加入的鉴别密钥K来鉴别消息的真伪；

一般用于网络通信中消息加密，前提是双方先要约定好key,就像接头暗号一样，然后消息发送把用key把消息加密，接收方用key + 消息明文再加密，拿加密后的值 跟 发送者的相对比是否相等，这样就能验证消息的真实性，及发送者的合法性了。

```

1 | import hmac
2 | h = hmac.new(b'天王盖地虎', b'宝塔镇河妖')
3 | print h.hexdigest()

```

更多关于md5,sha1,sha256等介绍的文章看[这里https://www.tbs-certificates.co.uk/FAQ/en/sha256.html](https://www.tbs-certificates.co.uk/FAQ/en/sha256.html)

Subprocess模块

The `subprocess` module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. This module intends to replace several older modules and functions:

```

os.system
os.spawn*

```

The recommended approach to invoking subprocesses is to use the `run()` function for all use cases it can handle. For more advanced use cases, the underlying `Popen` interface can be used directly.

The `run()` function was added in Python 3.5; if you need to retain compatibility with older versions, see the [Older high-level API](#) section.

```
subprocess.run(args, *, stdin=None, input=None, stdout=None, stderr=None, shell=False, timeout=)
```

Run the command described by `args`. Wait for command to complete, then return a `CompletedProcess` instance.

The arguments shown above are merely the most common ones, described below in [Frequently Used Arguments](#) (hence the use of keyword-only notation in the abbreviated signature). The full function signature is largely the same as that of the `Popen` constructor - apart from `timeout`, `input` and `check`, all the arguments to this function are passed through to that interface.

This does not capture stdout or stderr by default. To do so, pass `PIPE` for the `stdout` and/or `stderr` arguments.

The `timeout` argument is passed to `Popen.communicate()`. If the timeout expires, the child process will be killed and waited for. The `TimeoutExpired` exception will be re-raised after the child process has terminated.

The `input` argument is passed to `Popen.communicate()` and thus to the subprocess's stdin. If used it must be a byte sequence, or a string if `universal_newlines=True`. When used, the internal `Popen` object is automatically created with `stdin=PIPE`, and the `stdin` argument may not be used as well.

If `check` is True, and the process exits with a non-zero exit code, a `CalledProcessError` exception will be raised. Attributes of that exception hold the arguments, the exit code, and stdout and stderr if they were captured.

常用subprocess方法示例

```
#执行命令, 返回命令执行状态, 0 or 非0
>>> retcode = subprocess.call(["ls", "-l"])

#执行命令, 如果命令结果为0, 就正常返回, 否则抛异常
>>> subprocess.check_call(["ls", "-l"])
0

#接收字符串格式命令, 返回元组形式, 第1个元素是执行状态, 第2个是命令结果
>>> subprocess.getstatusoutput('ls /bin/ls')
(0, '/bin/ls')

#接收字符串格式命令, 并返回结果
>>> subprocess.getoutput('ls /bin/ls')
'/bin/ls'

#执行命令, 并返回结果, 注意是返回结果, 不是打印, 下例结果返回给res
>>> res=subprocess.check_output(['ls','-l'])
>>> res
b'total 0\ndrwxr-xr-x 12 alex staff 408 Nov 2 11:05 OldBoyCRM\n'

#上面那些方法, 底层都是封装的subprocess.Popen
poll()
Check if child process has terminated. Returns returncode

wait()
Wait for child process to terminate. Returns returncode attribute.

terminate() 杀掉所启动进程
communicate() 等待任务结束

stdin 标准输入

stdout 标准输出

stderr 标准错误

pid
The process ID of the child process.

#例子
>>> p = subprocess.Popen("df -h|grep
disk",stdin=subprocess.PIPE,stdout=subprocess.PIPE,shell=True)
>>> p.stdout.read()
b'/dev/disk1 465Gi 64Gi 400Gi 14% 16901472 104938142 14% /\n'
```

```
1 >>> subprocess.run(["ls", "-l"]) # doesn't capture output
2 CompletedProcess(args=['ls', '-l'], returncode=0)
```

```

3
4 >>> subprocess.run("exit 1", shell=True, check=True)
5 Traceback (most recent call last):
6 ...
7 subprocess.CalledProcessError: Command 'exit 1' returned non-zero exit status 1
8
9 >>> subprocess.run(["ls", "-l", "/dev/null"], stdout=subprocess.PIPE)
10 CompletedProcess(args=['ls', '-l', '/dev/null'], returncode=0,
11 stdout=b'crw-rw-rw- 1 root root 1, 3 Jan 23 16:23 /dev/null\n')

```

调用subprocess.run(...)是推荐的常用方法,在大多数情况下能满足需求,但如果你可能需要进行一些复杂的与系统的交互的话,你还可以用subprocess.Popen(),语法如下:

```

1 p = subprocess.Popen("find / -size +1000000 -exec ls -shl {} \;", shell=True, stdout=subprocess.PIPE)
2 print(p.stdout.read())

```

可用参数:

- args: shell命令, 可以是字符串或者序列类型 (如: list, 元组)
- bufsize: 指定缓冲。0 无缓冲,1 行缓冲,其他 缓冲区大小,负值 系统缓冲
- stdin, stdout, stderr: 分别表示程序的标准输入、输出、错误句柄
- preexec_fn: 只在Unix平台下有效,用于指定一个可执行对象 (callable object), 它将在子进程运行之前被调用
- close_fds: 在windows平台下, 如果close_fds被设置为True, 则新创建的子进程将不会继承父进程的输入、输出、错误管道。
所以不能将close_fds设置为True同时重定向子进程的标准输入、输出与错误(stdin, stdout, stderr)。
- shell: 同上
- cwd: 用于设置子进程的当前目录
- env: 用于指定子进程的环境变量。如果env = None, 子进程的环境变量将从父进程中继承。
- universal_newlines: 不同系统的换行符不同, True -> 同意使用 \n
- startupinfo与creationflags只在windows下有效
将被传递给底层的CreateProcess()函数, 用于设置子进程的一些属性, 如: 主窗口的外观, 进程的优先级等等

终端输入的命令分为两种:

- 输入即可得到输出, 如: ifconfig
- 输入进行某环境, 依赖再输入, 如: python

需要交互的命令示例

```

1 import subprocess
2
3 obj = subprocess.Popen(["python"], stdin=subprocess.PIPE, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
4 obj.stdin.write('print 1 \n ')
5 obj.stdin.write('print 2 \n ')
6 obj.stdin.write('print 3 \n ')
7 obj.stdin.write('print 4 \n ')
8
9 out_error_list = obj.communicate(timeout=10)
10 print out_error_list

```

subprocess实现sudo 自动输入密码

```

1 import subprocess
2
3 def mypass():
4     mypass = '123' #or get the password from anywhere
5     return mypass
6
7 echo = subprocess.Popen(['echo', mypass()],
8                           stdout=subprocess.PIPE,
9                           )
10
11 sudo = subprocess.Popen(['sudo', '-S', 'iptables', '-L'],
12                          stdin=echo.stdout,
13                          stdout=subprocess.PIPE,
14                          )
15

```



```
16 | end_of_pipe = sudo.stdout
17 |
18 | print "Password ok \n Iptables Chains %s" % end_of_pipe.read()
```

logging模块

很多程序都有记录日志的需求，并且日志中包含的信息即有正常的程序访问日志，还可能有错误、警告等信息输出，python的logging模块提供了标准的日志接口，你可以通过它存储各种格式的日志，logging的日志可以分为 `debug()`, `info()`, `warning()`, `error()` and `critical()` 5个级别，下面我们看一下怎么用。

最简单用法

```
1 | import logging
2 |
3 | logging.warning("user [alex] attempted wrong password more than 3 times")
4 | logging.critical("server is down")
5 |
6 | #输出
7 | WARNING:root:user [alex] attempted wrong password more than 3 times
8 | CRITICAL:root:server is down
```

看一下这几个日志级别分别代表什么意思

Level	When it's used
DEBUG	Detailed information, typically of interest only when diagnosing problems.
INFO	Confirmation that things are working as expected.
WARNING	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
ERROR	Due to a more serious problem, the software has not been able to perform some function.
CRITICAL	A serious error, indicating that the program itself may be unable to continue running.

如果想把日志写到文件里，也很简单

```
1 | import logging
2 |
3 | logging.basicConfig(filename='example.log',level=logging.INFO)
4 | logging.debug('This message should go to the log file')
5 | logging.info('So should this')
6 | logging.warning('And this, too')
```

其中下面这句中的level=loggin.INFO意思是，把日志纪录级别设置为INFO，也就是说，只有比日志是INFO或比INFO级别更高的日志才会被纪录到文件里，在这个例子，第一条日志是不会被纪录的，如果希望纪录debug的日志，那把日志级别改成DEBUG就行了。

```
1 | logging.basicConfig(filename='example.log',level=logging.INFO)
```

感觉上面的日志格式忘记加上时间啦，日志不知道时间怎么行呢，下面就来加上！

```
1 | import logging
2 | logging.basicConfig(format='%(asctime)s %(message)s', datefmt='%m/%d/%Y %I:%M:%S %p')
3 | logging.warning('is when this event was logged.')
4 |
5 | #输出
6 | 12/12/2010 11:46:36 AM is when this event was logged.
```

日志格式

%(name)s	Logger的名字

%(levelno)s	数字形式的日志级别
%(levelname)s	文本形式的日志级别
%(pathname)s	调用日志输出函数的模块的完整路径名，可能没有
%(filename)s	调用日志输出函数的模块的文件名
%(module)s	调用日志输出函数的模块名
%(funcName)s	调用日志输出函数的函数名
%(lineno)d	调用日志输出函数的语句所在的代码行
%(created)f	当前时间，用UNIX标准的表示时间的浮 点数表示
%(relativeCreated)d	输出日志信息时的，自Logger创建以 来的毫秒数
%(asctime)s	字符串形式的当前时间。默认格式是 "2003-07-08 16:49:45,896"。逗号后面的是毫秒
%(thread)d	线程ID。可能没有
%(threadName)s	线程名。可能没有
%(process)d	进程ID。可能没有
%(message)s	用户输出的消息

如果想同时把log打印在屏幕和文件日志里，就需要了解一点复杂的知识 了

Python 使用logging模块记录日志涉及四个主要类，使用官方文档中的概括最为合适：

- logger**提供了应用程序可以直接使用的接口；
- handler**将(**logger**创建的)日志记录发送到合适的目的输出；
- filter**提供了细度设备来决定输出哪条日志记录；
- formatter**决定日志记录的最终输出格式。

```
logger
每个程序在输出信息之前都要获得一个Logger。Logger通常对应了程序的模块名，比如聊天工具的图形界面模块可以这样获得它的Logger：
LOG=logging.getLogger("chat.gui")
而核心模块可以这样：
LOG=logging.getLogger("chat.kernel")

Logger.setLevel(lvl):指定最低的日志级别，低于lvl的级别将被忽略。debug是最低的内置级别，critical为最高
Logger.addFilter(filt)、Logger.removeFilter(filt):添加或删除指定的filter
```

Logger.addHandler(hdlr)、Logger.removeHandler(hdlr): 增加或删除指定的handler
Logger.debug()、Logger.info()、Logger.warning()、Logger.error()、Logger.critical(): 可以设置的日志级别

handler

handler对象负责发送相关的信息到指定目的地。Python的日志系统有多种Handler可以使用。有些Handler可以把信息输出到控制台，有些Logger可以把信息输出到文件，还有些Handler可以把信息发送到网络上。如果觉得不够用，还可以编写自己的Handler。可以通过addHandler()方法添加多个多handler
Handler.setLevel(lvl):指定被处理的信息级别，低于lvl级别的信息将被忽略
Handler.setFormatter(): 给这个handler选择一个格式
Handler.addFilter(filt)、Handler.removeFilter(filt): 新增或删除一个filter对象

每个Logger可以附加多个Handler。接下来我们就来介绍一些常用的Handler:

1) logging.StreamHandler

使用这个Handler可以向类似与sys.stdout或者sys.stderr的任何文件对象(file object)输出信息。它的构造函数是:

```
StreamHandler([strm])
```

其中strm参数是一个文件对象。默认是sys.stderr

2) logging.FileHandler

和StreamHandler类似，用于向一个文件输出日志信息。不过FileHandler会帮你打开这个文件。它的构造函数是:

```
FileHandler(filename[,mode])
```

filename是文件名，必须指定一个文件名。

mode是文件的打开方式。参见Python内置函数open()的用法。默认是'a'，即添加到文件末尾。

3) logging.handlers.RotatingFileHandler

这个Handler类似于上面的FileHandler，但是它可以管理文件大小。当文件达到一定大小之后，它会自动将当前日志文件改名，然后创建一个新的同名日志文件继续输出。比如日志文件是chat.log。当chat.log达到指定的大小之后，RotatingFileHandler自动把文件改名为chat.log.1。不过，如果chat.log.1已经存在，会先把chat.log.1重命名为chat.log.2。。。最后重新创建 chat.log，继续输出日志信息。它的构造函数是:

```
RotatingFileHandler( filename[, mode[, maxBytes[, backupCount]]])
```

其中filename和mode两个参数和FileHandler一样。

maxBytes用于指定日志文件的最大文件大小。如果maxBytes为0，意味着日志文件可以无限大，这时上面描述的重命名过程就不会发生。

backupCount用于指定保留的备份文件的个数。比如，如果指定为2，当上面描述的重命名过程发生时，原有的chat.log.2并不会被更名，而是被删除。

4) logging.handlers.TimedRotatingFileHandler

这个Handler和RotatingFileHandler类似，不过，它没有通过判断文件大小来决定何时重新创建日志文件，而是间隔一定时间就自动创建新的日志文件。重命名的过程与RotatingFileHandler类似，不过新的文件不是附加数字，而是当前时间。它的构造函数是:

```
TimedRotatingFileHandler( filename [,when [,interval [,backupCount]]])
```

其中filename参数和backupCount参数和RotatingFileHandler具有相同的意义。

interval是时间间隔。

when参数是一个字符串。表示时间间隔的单位，不区分大小写。它有如下取值:

S 秒

M 分

H 小时

D 天

W 每星期 (interval!=0时代表星期一)

midnight 每天凌晨

[+ View Code](#)

文件自动截断例子

 [View Code](#)

re模块

常用正则表达式符号

```
1  '.'  默认匹配除\n之外的任意一个字符，若指定flag DOTALL,则匹配任意字符，包括换行
2  '^'  匹配字符串开头，若指定flags MULTILINE,这种也可以匹配上(r"^\a","\nabc\neee",flags=re.MULTILI
3  '$'  匹配字符串结尾，或e.search("foo$", "bfoo\nsdfsf", flags=re.MULTILINE).group()也可以
4  '*'  匹配*号前的字符0次或多次，re.findall("ab*", "cabb3abcbba")  结果为['abb', 'ab', 'a']
5  '+'  匹配前一个字符1次或多次，re.findall("ab+", "ab+cd+abb+bba")  结果['ab', 'abb']
6  '?'  匹配前一个字符1次或0次
7  '{m}'  匹配前一个字符m次
8  '{n,m}'  匹配前一个字符n到m次，re.findall("ab{1,3}", "abb abc abbbbbb")  结果['abb', 'ab', 'abb']
9  '|'  匹配|左或|右的字符，re.search("abc|ABC", "ABCBAbaCD").group()  结果'ABC'
10 '('...'  分组匹配，re.search("(abc){2}a(123|456)c", "abcabca456c").group()  结果 abcabca456c
11
12
13  '\A'  只从字符串开头匹配，re.search("\Aabc", "alexabc")  是匹配不到的
14  '\Z'  匹配字符串结尾，同$
15  '\d'  匹配数字0-9
16  '\D'  匹配非数字
17  '\w'  匹配[A-Za-z0-9_]
18  '\W'  匹配非[A-Za-z0-9_]
19  '\s'  匹配空白字符、\t、\n、\r , re.search("\s+", "ab\tc1\n3").group()  结果 '\t'
20
21  '(?P<name>...)'  分组匹配 re.search("(?P<province>[0-9]{4})(?P<city>[0-9]{2})(?P<birthday>[0-9
```

最常用的匹配语法

```
1  re.match  从头开始匹配
2  re.search  匹配包含
3  re.findall  把所有匹配到的字符放到以列表中的元素返回
4  re.splitall  以匹配到的字符当做列表分隔符
5  re.sub  匹配字符并替换
```

反斜杠的困扰

与大多数编程语言相同，正则表达式里使用"\"作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符"\"，那么使用编程语言表示的正则表达式里将需要4个反斜杠"\\\\"：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。Python里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用r"\"表示。同样，匹配一个数字的"\"d"可以写成r"\"d"。有了原生字符串，你再也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。

仅需轻轻知道的几个匹配模式

```
1  re.I(re.IGNORECASE): 忽略大小写（括号内是完整写法，下同）
2  M(MULTILINE): 多行模式，改变'^'和'$'的行为（参见上图）
3  S(DOTALL): 点任意匹配模式，改变'.'的行为
```

本节作业

开发一个简单的python计算器

1. 实现加减乘除及括号优先级解析
2. 用户输入 1 - 2 * ((60-30 +(-40/5) * (9-2*5/3 + 7 /3*99/4*2998 +10 * 568/14)) - (-4*3)/(16-3*2))等类似公式后，必须自己解析里面的(),+,-,*,/符号和公式(不能调用eval等类似功能偷懒实现)，运算后得出结果，结果必须与真实的计算器所得出的结果一致

```
hint:
re.search(r'\([^()]+\)',s).group()

'(-40/5)'
```

分类: [Python自动化开发之路](#)

好文要顶

关注我

收藏该文

金角大王

关注 - 5

粉丝 - 10868

+加关注

230

posted @ 2016-01-26 19:21 金角大王 阅读(81625) 评论(16) 编辑 收藏

评论列表

#1楼	2016-08-25 14:13	freedom_dog	回复 引用
		受教。	支持(0) 反对(0)
#2楼	2017-07-31 10:37	hqbm9w	回复 引用
		收菜了	支持(0) 反对(0)
#3楼	2017-08-05 19:16	day丁兆海	回复 引用
		谢谢	支持(0) 反对(0)
#4楼	2017-10-08 19:33	CrisChou	回复 引用
		收藏	支持(0) 反对(0)
#5楼	2017-11-27 19:46	Nathaniel-J	回复 引用
		random的import少个i	支持(0) 反对(0)
#6楼	2018-01-04 21:01	alwayslz	回复 引用
		虽然成功了，但是好像写得很不规范...忽略的我不规范的备注吧。 求alex大神指导。	

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  # Author: Liu Zhao
4
5  import re
6
7  def chengchu(s):          #处理带负号的乘除，负号歪打正着处理成功了
8      r = re.compile(r'[\d\.]+\[/]\-?[\d\.]+')
9      while re.search(r'[/]', s):
10         ma = re.search(r, s).group()
11         # print(ma)
12         li = re.findall(r'(-?[\d\.]+\[/])', ma)
13         if li[1] == '/':
14             result = str(float(li[0])*float(li[2]))
15         else:
16             result = str(float(li[0])/float(li[2]))
17         s = s.replace(ma, result, 1)
18     return s
19
20 def jiajian(s):           #处理加减法，变成数组，全加
21     li = re.findall(r'([\d\.]+\+|\-)', s)
22     sum = 0
23     for i in range(len(li)-1):          #处理有两个--号连续的情况
24         if li[i]=='-' and li[i+1]=='-':
25             li[i] = '+'
```

```
26         li[i+1] = '+'
27     for i in range(len(li)):
28         if li[i] == '-':
29             li[i] = '+'
30             li[i+1] = float(li[i+1]) * -1
31     for i in li:
32         if i == '+':
33             i = 0
34             sum = sum + float(i)
35     return str(sum)
36
37 def simple(s): #处理不带括号的
38     return jiajian(chengchu(s))
39
40 def complex(s): #处理带括号的
41     while '(' in s:
42         reg = re.compile(r'\([^()]+\)')
43         ma = re.search(reg, s).group()
44         result = simple(ma[1:-1])
45         s = s.replace(ma, result, 1)
46     return simple(s)
47
48 ss = '1 - 2 * ( (60-30 +(-40/5) * (9-2*5/3 + 7 /3*99/4*2998 +10 * 568/14 )) - (-4*3)/ (16-3
49 print(complex(ss))
50 print(eval(ss))
```

支持(9) 反对(2)

#7楼 2018-01-17 22:35 秦艳莉

回复 引用

您好，麻烦问下，
ret = re.findall('\\\\', 'apladafaa\\cdada')
print(ret) #输出结果是['\\']

按照你这个解释：
反斜杠的困扰
与大多数编程语言相同，正则表达式里使用\"作为转义字符，这就可能造成反斜杠困扰。假如你需要匹配文本中的字符\"，那么使用编程语言表示的正则表达式里将需要4个反斜杠\"\\\\\\\\\"：前两个和后两个分别用于在编程语言里转义成反斜杠，转换成两个反斜杠后再在正则表达式里转义成一个反斜杠。Python里的原生字符串很好地解决了这个问题，这个例子中的正则表达式可以使用r\"表示。同样，匹配一个数字的\"\\\\d\"可以写成r\"d\"。有了原生字符串，你再也不用担心是不是漏写了反斜杠，写出来的表达式也更直观。

第一个疑问点：4个斜杠，第一步经过python解释之后不是应该剩下三个斜杠，re解释之后剩下两个斜杠
第二个疑问点：按照你上面的推理和之前所讲的，前面经过一系列的转义之后剩下一个斜杠，'apladafaa\\cdada'字符串里也只有一个斜杠，为什么输出结果会有两个斜杠

支持(0) 反对(0)

#8楼 2018-02-26 21:54 TruthK

回复 引用

这个作业,把 算式(中缀式)转化成后缀式,就很好做

支持(0) 反对(0)

#9楼 2018-06-06 09:34 钱先生

回复 引用

subprocess模块可用参数中close_fds应为close_fds.
参阅python官方文档<https://docs.python.org/3/library/subprocess.html>.

支持(0) 反对(0)

#10楼 2018-07-10 09:03 335856182

回复 引用

@ 秦艳莉
姑娘你好：
在re模块中一个\"需要两个\"\\\\\"表示，在python中两个\"\\\\\"表示字符\"。
也就是说在python中输入\"\\\\\\\\\"等同于字符串\"\\\\\"。

关于输出结果是\"\\\\\"的问题，据我推测是当匹配成功的时候re模块输出\"，经过python处理就变成\"\\\\\"了。如有纰漏，还望赐教。

支持(2) 反对(0)

#11楼 2018-08-13 02:03 c_G-17

回复 引用

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import re
parser_str = '1 - 2 * ( (60-30 +(-40/5) * (9-2*5/3 + 7 /3*99/4*2998 +10 * 568/14 )) - (-4*3)/ (16-3*2) )'

#完成
def deal_mul_divi(string):
    """
    处理乘除法
    只处理传过来的字符串，每次处理一个，返回字符串
    :param string: 字符串格式严格要求为 25/-5 或 34*3 这样，否则返回None
```

```

:return: 返回处理后的结果字符串
"""
res1 = re.search('\d+\.\d*\V(\-|\d+\.\d*)+', string)
res2 = re.search('\d+\.\d*\*(\-|\d+\.\d*)+', string)
if not res1 and not res2:
    print('格式有误: {}'.format(string))
    return None
else:
    items = re.findall('\d+\.\d*', string)
    result = str(float(items[0])/float(items[1])) \
    if res1 else str(float(items[0])*float(items[1]))
    # 如果字符串中有- 负号, 那就增加标记值, 让返回值为负 re.search('-', string)同下 '-' in
    result = '{}'.format(result) if '-' in string else result
    return result
# 完成
def deal_plus_minus(string):
    """
    将没有乘除号的带 (不带) 括号都行的字符串传入。该函数先处理字符串中所有负数: (40-4+34)
    再处理所有正数, 再用正数减负数值作为结果返回, 操作皆为浮点数。
    :param string: 参数为只有 + - 号的算式
    :return:
    """
    if re.search('\*|\V', string): # 如果有乘除号视为错误
        return None
    num_minus = 0
    for minus in re.findall('\-(\d+\.\d*)', string): # 将所有负数找出来并加起来
        string = string.replace(minus, 'sep') # 所有前面带减号的数, 都被sep 符替换
        num_minus += float(minus)
    num_plus = 0
    for plus in re.findall('(\d+\.\d*)', string): # 匹配正数相加 # |\+(\d+\.\d*)
        num_plus += float(plus)
    return str(num_plus - num_minus)
# 完成
def match_brackets(string):
    """
    匹配算式中的括号, 并调用函数处理
    :param string: 算式字符串
    :return:
    """
    flag = True
    while flag:
        brackets_str = re.search('\((\+|\-|\*|\/|\d+)\)', string) # 拿到括号字符串
        if not brackets_str:
            flag = False
            continue
        else:
            result = deal_brackets(brackets_str.group()) # 调用处理括号函数, 处理返回
            # print('\033[33;1m{}\033[0m'.format(string))
            string = string.replace(brackets_str.group(), result, 1) # 将计算原括号得到的结果替换原括号
            # print('\033[34;1m{}\033[0m'.format(string))
            string = re.sub('\(+\+|\(-\+)', '-', string) # 处理 +- 号和 -+ 并排一起
            string = re.sub('\--', '+', string) # 处理 -- 两减号并排
    return string

def deal_brackets(string):
    """
    处理传过来的括号
    :param string:
    :return:
    """
    flag = True

    while flag:
        # ( -3.2/-1.6-2-3*-2)这样的也要能匹配得 3.2/-1.6
        mul_divi_str = re.search('\d+\.\d*\(\*\|\/\)(\-|\d+\.\d*){1,2}', string) # 只能匹配一到两位如 - 1.6
        if not mul_divi_str:
            flag = False
            break
        else:
            # print('\033[31;4m处理传来的乘除: {}\033[0m'.format(mul_divi_str.group()))
            mul_divi_res = deal_mul_divi(mul_divi_str.group())
            string = string.replace(mul_divi_str.group(), mul_divi_res, 1)
            string = re.sub('\(+\+|\(-\+)', '-', string) # 处理 +- 号和 -+ 并排一起
            string = re.sub('\--', '+', string) # 处理 -- 两减号并排
    return deal_plus_minus(string)

def calculate(string):
    strip_space = lambda x: re.sub(' ', '', x, count=x.count(' ')) # 将算式中的所有空格剔除
    string = strip_space(string)
    string = match_brackets(string) # 处理完表达式所有的括号
    result = deal_brackets(string) # 在把没有括号的表达式交给它处理一次
    return result

print('\033[31;1meval:\033[0m{: >22}'.format(eval(parser_str))) # eval 验证结果
print('\033[32;2mcalculate:\033[0m{}'.format(calculate(parser_str))) # 正则计算

eval: 2776672.6952380957
calculate: 2776672.6952380957

```

支持(2) 反对(0)

#12楼 2018-09-30 13:36 d_k

回复 引用

```

1 def calculator(expression='1 - 2 * ( (60-30 +(-40/5) * (9-2*5/3 + '
2                                     '7 /3*99/4*2998 +10 * 568/14 )) - (-4*3)/ (16-3*2) )'):
3     ...

```

```

4     计算器
5     总逻辑:
6         1.先找出内层括号, 然后计算其内的终值, 用终值替换原表达式
7         2.然后重复以上过程
8     无括号部分计算逻辑:
9         1.先截断为数字和运算符的列表, 数字可包含负号;
10        2.运算列表中的值并替换, 先乘除后加减;
11    :param expression: 字符串表达式
12    :return: 表达式计算值
13    '''
14    print(eval(expression))
15    import re
16    operatorDict = {
17        '+': lambda a, b: float(a) + float(b),
18        '-': lambda a, b: float(a) - float(b),
19        '*': lambda a, b: float(a) * float(b),
20        '/': lambda a, b: float(a) / float(b),
21    }
22
23    def calWithOutBrackets(expression='-1+2*3/4-5*-3'):
24        #截断为数字和运算符的列表
25        for i in operatorDict:
26            expression=expression.replace(i, 's'+i+'s')
27        l=expression.split('s')
28        # l2将填充为一个数字和运算的列表, 数字可以带负号
29        l2=[]
30        i=0
31        while i<len(l):
32            if l[i]=='':
33                l2.append(l[i+1]+l[i+2]) #带负号的数字
34                i+=2
35            else:
36                l2.append(l[i]) #不带负号的数字和运算符
37                i+=1
38        #运算乘除
39        i=1
40        while i<len(l2):
41            if l2[i] in ['*', '/']:
42                l2[i-1:i+2]=[operatorDict[l2[i]](l2[i-1], l2[i+1])]
43            else:
44                i+=2
45        #运算加减
46        while len(l2)>1:
47            l2[0:3] = [operatorDict[l2[1]](l2[0], l2[2])]
48        return str(l2[0])
49
50    expression=expression.replace(' ', '')
51    check=re.search('\([^(\|)+\]', expression)
52    while check:
53        checkValue=check.group()
54        # print(checkValue)
55        expression=expression.replace(checkValue, calWithOutBrackets(checkValue[1:-1]))
56        check = re.search('\([^(\|)+\]', expression)
57    else:
58        return calWithOutBrackets(expression)

```

支持(2) 反对(0)

#13楼 2019-01-02 18:16 侯赛雷

回复 引用

@ d_k

能简单介绍一下这个匹配正则的吗,没看懂,谢谢

支持(0) 反对(0)

#14楼 2019-01-21 17:45 卜戈的博客

回复 引用

<https://www.cnblogs.com/wuxiaobo/articles/10292972.html>. 作业

支持(0) 反对(0)

#15楼 2019-03-12 17:54 白纸扬

回复 引用

@ alwayslz

在jiajian方法内加入判断:

def jiajian(s): #处理加减法, 变成数组, 全加

li = re.findall(r'([\d\.\+|\-|+|-]', s)

if len(li) < 3: # 说明只是单个的数, 不是加减表达式, 直接返回

return s

sum = 0

for i in range(len(li)-1): #处理有两个--号连续的情况

...
可以提高效率

支持(1) 反对(0)

#16楼 2020-07-06 11:56 编程届的小学生

回复 引用

```
#思路: 过滤最里面的 () 将里面的数值计算后替换到原有公示字符串, 直到将所有的括号剔除后再计算没有括号的字符串
import re
def mul_div(num_str): #先算乘除, (num_str) 是过滤出来的只有一对括号和里面的加减乘除的字符串
    sign = True
    while sign:
        res = re.search('\-?\d+([\.\d]+)?([\+\-\*/]{1})\-?\d+([\.\d]+)?+', num_str) #过滤含有
        if res: #首先要确定可以过滤出东西, 如果过滤不出加减乘除就在else中 返回 处理完的num_str
            if '*' not in res.group() and '/' not in res.group(): #如果没有* 并且没有 /
                sign = False
            for iters in res.group():
                if iters == '*': #计算乘号 并将计算好的值替换原来的*号两边的字符, 值 <= a*b
                    flogs = float(re.search(r'\-?\d+([\.\d]+)?*', res.group()).group()[1:-1])
                    if re.search(r'\-?\d+([\.\d]+)?\*', res.group()).group()[1:-1][0] == '-':
                        num_str = res.group().replace(re.search('\-?\d+([\.\d]+)?*\-?\d+([\.\d]+)?', res.group()).group(), str(flogs))
                        break
                    num_str = res.group().replace(re.search('\-?\d+([\.\d]+)?*\-?\d+([\.\d]+)?', res.group()).group(), str(flogs))
                    break
                elif iters == '/': #思路和* 相同, 也要考虑两个带负号的数字相乘替换的时候要在值前加负号
                    flogs = float(re.search(r'\-?\d+([\.\d]+)?/', res.group()).group()[1:-1])
                    if re.search(r'\-?\d+([\.\d]+)?/', res.group()).group()[1:-1][0] == '-':
                        num_str = res.group().replace(re.search('\-?\d+([\.\d]+)?/\-?\d+([\.\d]+)?', res.group()).group(), str(flogs))
                        break
                    num_str = res.group().replace(re.search('\-?\d+([\.\d]+)?/\-?\d+([\.\d]+)?', res.group()).group(), str(flogs))
                    break
            else: #对应第一个if, 如果num_str不能再过滤了直接返回到加减函数中
                return sub_puls(num_str) #返回值到加减函数中
    return sub_puls(num_str)

def sub_puls(num_str):# 加减函数
    sign = True
    while sign:
        res = re.search('\-?\d+([\.\d]+)?([\+\-]{1})\-?\d+([\.\d]+)?+', num_str) #过滤含有
        if res is None: # 如果没有直接退出循环
            sign = False
        elif res:
            num_str=re.sub('\-+', '+', res.group()) #计算过程中会出现 -- 两个减号同时出现的情况
            num_str=re.sub('\++', '+', num_str) #应该不会出现++同时出现的情况, 如果有就替换成+
            for iters in res.group(): #遍历过滤出来的字符串
                if iters == '-': #思路和乘除类似,
                    if re.match('-', res.group()):#从头匹配字符串, 如果开头就是负号, 则跳过本次循环
                        continue
                    flogs = float(re.search(r'\-?\d+([\.\d]+)?-', res.group()).group()[1:-1])
                    num_str = res.group().replace(re.search('\-?\d+([\.\d]+)?\-?\d+([\.\d]+)?', res.group()).group(), str(flogs))
                    break
                elif iters == '+':
                    flogs = float(re.search(r'\-?\d+([\.\d]+)?+', res.group()).group()[1:-1])
                    num_str = res.group().replace(re.search('\-?\d+([\.\d]+)?\+?\d+([\.\d]+)?', res.group()).group(), str(flogs))
                    break
            return num_str

num = '1 - 2 * ( (60-30 +(-40/5) * (9-2*5/3 + 7 /3*99/4*2998 +10 * 568/14 )) - (-4*3)/'
num = num.replace(' ', '')
Flogs = True
blog = ''
while Flogs:
    res = re.search('\(-?\d+([\.\d]+)?([\+\-\*/]{1})\-?\d+([\.\d]+)?\)', num) #过滤只有一个
    if res:
        blog = mul_div(res.group()) #将过滤出的内容传给乘除函数处理,
        num = num.replace(res.group(), blog) #将乘除函数计算好的值替换掉num过滤出的带括号的表达式
    else: #最终没有 (表达式) 了, 那将最终替换好的num 字符串表达式传给乘除函数 mul_div进行处理, 最终得到
        blog = mul_div(num)
        break





print(blog) #用程序计算的值最终和eval计算的值一样。
print(eval(num))``
```

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】了不起的开发者，挡不住的华为，园子里的品牌专区

【推荐】有道智云周年庆，API服务大放送，注册即送100元体验金！

【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】史上最全 Vue 面试题汇总



相关博文：

- Python学习-day5 常用模块
- Day5 - Python基础5 常用模块学习
- Day5 - 常用模块学习
- Day5 python常用模块
- [Python Day5] 常用模块

» 更多推荐...

最新 IT 新闻：

- EC6价格能打，筹备电池资产公司，满血复活的蔚来如何加电？
- 被雷军大呼「神奇」的手机，十年前就支持无线充电，它有什么来头？
- 滴滴当前的最优先事项不是IPO，是什么？
- 理想汽车≠新物种
- 蚂蚁集团上市将掀“万人造富运动”？情况远比传说复杂

» 更多新闻...