1. print是一个常用函数，其功能就是输出括号中得字符串。
2. 内置函数type(), 用以查询变量的类型。
3. List.append(对象); //向list的尾部添加对象
4. \_\_init\_\_()是一个特殊方法(special method)。特殊方法的特点是名字前后有两个下划线。

\_\_init\_\_并不相当于C#中的构造函数，执行它的时候，**实例已构造**出来了。

1. dir()用来查询一个类或者对象所有属性。 //print dir(list)

help()用来查询的说明文档。 //print help(list)

1. 在range()函数中，分别定义上限，下限和每次循环的步长。

range(0,len(S),2)

1. enumerate()函数，可以在每次循环中同时得到下标和元素：

S = 'abcdefghijk' for (index,char) in enumerate(S):

实际上，enumerate()在每次循环中，返回的是一个包含两个元素的定值表(tuple)，两个元素分别赋予index和char

1. 如果你多个等长的序列，然后想要每次循环时从各个序列分别取出一个元素，可以利用zip()方便地实现：ta = [1,2,3] tb = [9,8,7] tc = ['a','b','c']

for (a,b,c) in zip(ta,tb,tc):

print(a,b,c)

每次循环时，从各个序列分别从左到右取出一个元素，合并成一个tuple，然后tuple的元素赋予给a,b,c

ta = [1,2,3]

tb = [9,8,7]

zipped = zip(ta,tb) # cluster

print(zipped)

na, nb = zip(\*zipped) # decompose

print(na, nb)

1. lambda函数:

lambda语句中，冒号前是参数，可以有多个，用逗号隔开，冒号右边的返回值。lambda语句构建的其实是一个函数对象。

func = lambda x,y: x + y print func(3,4)

lambda生成一个函数对象。该函数参数为x,y，返回值为x+y。函数对象赋给func。func的调用与正常函数无异。

1. map()是Python的内置函数。它的第一个参数是一个函数对象。

根据提供的函数对指定序列做映射。

re = map((lambda x: x+3),[1,3,5,6]) //map()的功能是将函数对象依次作用于表的每一个元素，每次作用的结果储存于返回的表re中。

re = map((lambda x,y: x+y),[1,2,3],[6,7,9]) //将每次从两个表中分别取出一个元素，带入lambda所定义的函数。

1. filt()函数的第一个参数也是一个函数对象。它也是将作为参数的函数对象作用于多个元素。如果函数对象返回的是True，则该次的元素被储存于返回的表中。

def func(a):

if a > 100:

return True

else:

return False

print filter(func,[10,56,101,500])

1. reduce()函数的第一个参数也是函数，但有一个要求，就是这个函数自身能接收两个参数。reduce可以累进地将函数作用于各个参数。

print reduce((lambda x,y: x+y),[1,2,5,7,9])

reduce将表中的前两个元素(1和2)传递给lambda函数，得到3。该返回值(3)将作为lambda函数的第一个参数，而表中的下一个元素(5)作为lambda函数的第二个参数，进行下一次的对lambda函数的调用，得到8。 相当于(((1+2)+5)+7)+9

reduce()函数在3.0里面不能直接用的，它被定义在了functools包里面，需要引入包

1. 1）查询对象的属性：

hasattr(obj, attr\_name)   # attr\_name是一个字符串

a = [1,2,3]

print(hasattr(a,'append'))

2）查询对象所属的类和类名称

a = [1, 2, 3]

print a.\_\_class\_\_

print a.\_\_class\_\_.\_\_name\_\_

3）查询父类

我们可以用\_\_base\_\_属性来查询某个类的父类：cls.\_\_base\_\_

print(list.\_\_base\_\_)

1. 生成器: 生成器(generator)的主要目的是构成一个用户自定义的循环对象。

yield 的作用就是把一个函数变成一个 generator，带有 yield 的函数不再是一个普通函数，Python 解释器会将其视为一个 generator，调用 fab(5) 不会执行 fab 函数，而是返回一个**iterable 对象**！

生成器的编写方法和函数定义类似，只是在return的地方改为yield。生成器中可以有多个yield。当生成器遇到一个yield时，会暂停运行生成器，返回yield后面的值。当再次调用生成器的时候，会从刚才暂停的地方继续运行，直到下一个yield。生成器自身又构成一个循环器，每次循环使用一个yield返回的值。

def fab(max):

n, a, b = 0, 0, 1

while n < max:

yield b

a, b = b, a + b

n += 1

for i in fab(5):

print i, 🡺 1 1 2 3 5

在 for 循环执行时，每次循环都会执行 fab 函数内部的代码，执行到 yield b时，fab 函数就返回一个迭代值，下次迭代时，代码从yield b 的下一条语句继续执行，而函数的本地变量看起来和上次中断执行前是完全一样的，于是函数继续执行，直到再次遇到 yield。看起来就好像一个函数在正常执行的过程中被 yield 中断了数次，每次中断都会通过 yield 返回当前的迭代值。

1. 表推导:

假设我们生成表L：

L = []

for x in range(10):

L.append(x\*\*2)

以上产生了表L，但实际上有快捷的写法，也就是表推导的方式:

L = [x\*\*2 for x in range(10)]

1. 序列有两种：tuple（定值表；也有翻译为元组）和 list(表)

tuple和list的主要区别在于：一旦建立，tuple的各个元素不可再变更，而list的各个元素可以再变更。

>>>s1 = (2, 1.3, 'love', 5.6, 9, 12, False) # s1是一个tuple

>>>s2 = [True, 5, 'smile'] # s2是一个list

一个序列作为另一个序列的元素 >>>s3 = [1,[3,4,5]]

空序列 >>>s4 = []

Python元组包含了以下内置函数

序号 方法及描述

1、 cmp(tuple1, tuple2) //比较两个元组元素。

2、 len(tuple) //计算元组元素个数。

3、 max(tuple) //返回元组中元素最大值。

4 、min(tuple) //返回元组中元素最小值。

5、 tuple(seq) //将列表转换为元组。

1. 序列元素的下标从0开始：

>>>print s1[0]

>>>print s2[2]

>>>print s3[1][2]

由于list的元素可变更，你可以对list的某个元素赋值：

>>>s2[1] = 3.0

>>>print s2

如果你对tuple做这样的操作，会得到错误提示

1. 范围引用： 基本样式[下限:上限:步长]

>>>print s1[:5] # 从开始到下标4 （下标5的元素 不包括在内）

>>>print s1[2:] # 从下标2到最后

>>>print s1[0:5:2] # 从下标0到下标4 (下标5不包括在内)，每隔2取一个元素下标为0，2，4的元素）

>>>print s1[2:0:-1] # 从下标2到下标1

从上面可以看到，在范围引用的时候，如果写明上限，那么这个上限本身不包括在内。

>>>print s1[-1] # 序列最后一个元素

>>>print s1[-3] # 序列倒数第三个元素

同样，如果s1[0:-1], 那么最后一个元素不会被引用 （再一次，不包括上限元素本身）

1. >>>print 3\*\*2 # 乘方

>>>print 5 in [1,3,5] # 5是list [1,3,5]的一个元素

数学 +, -, \*, /, \*\*, % 判断 ==, !=, >, >=, <, <=, in 逻辑 and, or, not

1. if <条件1>：

statement

elif <条件2>：

statement

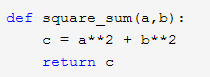
else：

statement

1. for 元素 in 序列: for a in [3,4.4,'life']:

statement print a

1. range()，来帮助你建立表 idx = range(5) --可以看到idx是[0,1,2,3,4]
2. def这个关键字通知python：我在定义一个函数。

square\_sum是函数名。

return可以返回多个值，以逗号分隔。相当于返回一个tuple(定值表)。

return a,b,c          # 相当于 return (a,b,c)

1. 词典 (dictionary)

词典和表类似的地方，是包含有多个元素，每个元素以逗号分隔。但词典的元素包含有两部分，键和值，常见的是以字符串来表示键，也可以使用数字或者真值来表示键（不可变的对象可以作为键）。 dic = {'tom':11, 'sam':57,'lily':100}

>>>dic['lilei'] = 99 #在词典中增添一个新元素的方法：

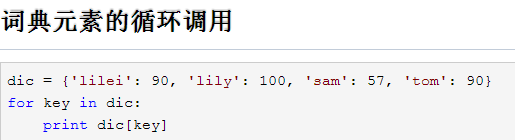
>>>dic['tom'] = 30 #更改已存在元素的值

>>>print dic.keys() # 返回dic所有的键

>>>print dic.values() # 返回dic所有的值

>>>print dic.items() # 返回dic所有的元素（键值对）

>>>dic.clear() # 清空dic，dict变为{}



在循环中，dict的每个键，被提取出来，赋予给key变量。

1. 我们打开一个***文件***，并使用一个对象来表示该文件

f = open(文件名，模式) 最常用的模式有："r" (只读) ; “w”(写入)

>>>f = open("test.txt","r")

读取：

content = f.read(N)          # 读取N bytes的数据

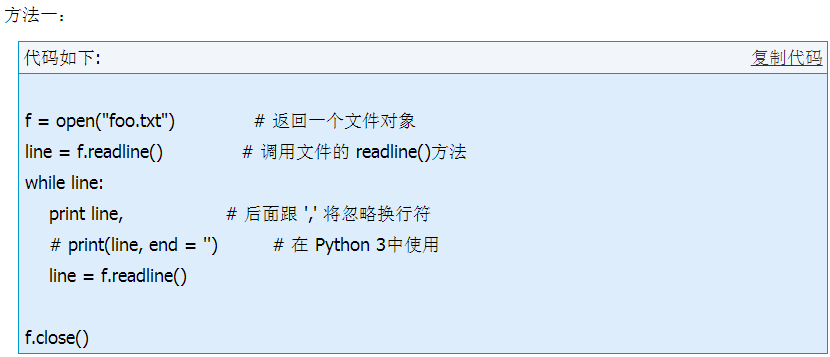
content = f.readline()       # 读取一行

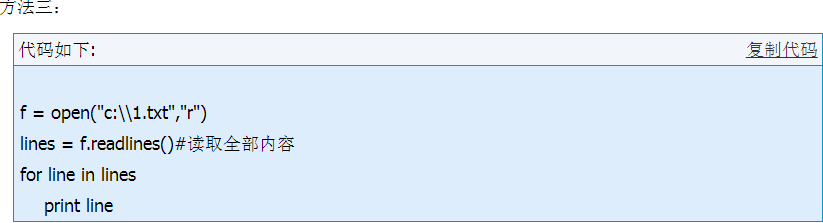
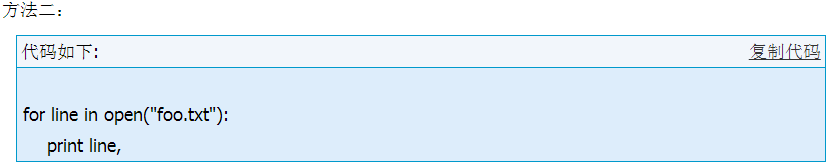
content = f.readlines()      # 读取所有行，储存在列表中，每个元素是一行。  
写入：

f.write('I like apple')      # 将'I like apple'写入文件

关闭文件：

f.close()

逐行读取文件内容的三种方法:



1. **模块**(module)也是为了同样的目的。在Python中，一个.py文件就构成一个模块。通过模块，你可以调用其它文件中的程序。

Python会在以下路径中搜索它想要寻找的模块：

1. 序所在的文件夹
2. 标准库的安装路径
3. 操作系统环境变量PYTHONPATH所包含的路径

我们先写一个first.py文件，内容如下：

def laugh():

print 'HaHaHaHa'

再写一个second.py，并引入first中的程序：

import first

first.laugh()

Python中还有其它的引入方式：

import a as b # 引入模块a，并将模块a重命名为b

from a import function1 # 从模块a中引入function1对象。调用a中对象时，我们不用再说明模块，即直接使用function1，而不是a.function1。

from a import \* # 从模块a中引入所有对象。调用a中对象时，我们不用再说明模块，即直接使用对象，而不是a.对象。

可以将功能相似的模块放在同一个文件夹（比如说this\_dir）中，构成一个模块包。通过

import this\_dir.module

引入this\_dir文件夹中的module模块。

可以将功能相似的模块放在同一个文件夹（比如说this\_dir）中，构成一个模块包

通过 import this\_dir.module 引入this\_dir文件夹中的module模块。

该文件夹中必须包含一个\_\_init\_\_.py的文件，提醒Python，该文件夹为一个模块包。\_\_init\_\_.py可以是一个空文件。

1. 1）关键字传递：

键字(keyword)传递是根据每个参数的名字传递参数。关键字并不用遵守位置的对应关系。

关键字传递可以和位置传递混用。但位置参数要出现在关键字参数之前。

2）参数默认值：

def f(a,b,c=10):

return a+b+c

print(f(3,2))  
print(f(3,2,1))

3）包裹传递：

def func(\*name):

print type(name)

print name

func(1,4,6)  
func(5,6,7,1,2,3)

在func的参数表中，所有的参数被name收集，根据位置合并成一个元组(tuple)，这就是包裹位置传递。为了提醒Python参数，name是包裹位置传递所用的元组名，在定义func时，在name前加\*号。

def func(\*\*dict):

print type(dict)

print dict

func(a=1,b=9)

func(m=2,n=1,c=11)

与上面一个例子类似，dict是一个字典，收集所有的关键字，传递给函数func。为了提醒Python，参数dict是包裹关键字传递所用的字典，在dict前加\*\*。

包裹传递的关键在于定义函数时，在相应元组或字典前加\*或\*\*。

1. 异常处理:

try:

...

except exception1:

...

except exception2:

...

except:

...

else:

...

finally:

...

finally是无论是否有异常，最后都要做的一些事情。

流程如下:

try->异常->except->finally

try->无异常->else->finally

try:

os.rmdir('/home/kk/python/tt/mm')

except Exception,e:

print 'result:',e

1. 函数的参数传递，本质上传递的是引用。

def f(x):

x = 100

print x

a = 1

f(a)

print a

参数x是一个新的引用，指向a所指的对象。如果参数是不可变(immutable)的对象，a和x引用之间相互独立。对参数x的操作不会影响引用a。

1. python调用shell命令的方法：

1）os.system(command)

在一个子shell中运行command命令，并返回command命令执行完毕后的退出状态。这实际上是使用C标准库函数system()实现的。这个函数在执行command命令时需要重新打开一个终端，并且无法保存command命令的执行结果。

os.system('cat /proc/cpuinfo')

2）os.popen(command,mode)

打开一个与command进程之间的管道。这个函数的返回值是一个文件对象，可以读或者写(由mode决定，mode默认是’r')。如果mode为’r'，可以使用此函数的返回值调用read()来获取command命令的执行结果。

output = os.popen('cat /proc/cpuinfo')

print output.read()

3）commands.getstatusoutput(command)

　使用os.popen()函数执行command命令并返回一个元组(status,output)，分别表示command命令执行的返回状态和执行结果。对command的执行实际上是按照{command;} 2>&1的方式，所以output中包含控制台输出信息或者错误信息。output中不包含尾部的换行符。

(status, output) = commands.getstatusoutput('cat /proc/cpuinfo')

print status, output

4）subprocess.call(["some\_command","some\_argument","another\_argument\_or\_path"]

subprocess.call(command,shell=True)

//它基本上就像Popen类并都使用相同的参数，但是它只简单的等待命令完成并给你返回代码。比如：return\_code = subprocess.call("echo Hello World", shell=True)

subprocess.Popen(command, shell=True)

如果command不是一个可执行文件，shell=True不可省。

subprocess.Popen("echo baicu ",stdout=subprocess.PIPE,shell=True).stdout.read()来替代print os.popen("echo Hello World").read()

1. Python标准库：
2. 利用time包管理时间，利用datetime包管理日期和时间。
3. os包是Python与操作系统的接口。我们可以用os包来实现操作系统的许多功能，比如管理系统进程，改变当前路径(相当于‘cd’)，改变文件权限等，建立。
4. sys包被用于管理Python自身的运行环境。Python是一个解释器(interpreter), 也是一个运行在操作系统上的程序。我们可以用sys包来控制这一程序运行的许多参数，比如说Python运行所能占据的内存和CPU， Python所要扫描的路径等。另一个重要功能是和Python自己的命令行互动，从命令行读取命令和参数。
5. subprocess包被用于执行外部命令，其功能相当于我们在操作系统的命令行中输入命令以执行，比如常见的系统命令'ls'或者'cd'，还可以是任意可以在命令行中执行的程序。
6. 多线程(threading包)运行和多进程(multiprocessing包)运行。通过多线程和多进程，可以提高系统资源的利用率，提高计算机的处理速度。Python在这些包中，附带有相关的通信和内存管理工具。此外，Python还支持类似于UNIX的signal系统，以实现进程之间的粗糙的信号通信。
7. socket是网络可编程部分的底层。通过socket包，我们可以直接管理socket，比如说将socket赋予给某个端口(port)，连接远程端口，以及通过连接传输数据。我们也可以利用SocketServer包更方便地建立服务器。
8. http的服务器端和客户端的应用支持(BaseHTTPServer包; urllib包, urllib2包), 并且可以通过urlparse包对URL（URL实际上说明了网络资源所在的位置）进行理解和操作。
9. \_\_name\_\_

def lib\_func(a):

return a + 10

def lib\_func\_another(b):

return b + 20

if \_\_name\_\_ == '\_\_main\_\_': //

test = 101

print(lib\_func(test))

当直接运行TestLib.py时，\_\_name\_\_为"\_\_main\_\_"。如果被import的话，\_\_name\_\_为"TestLib"。

import TestLib as test         # 引用TestLib模块，并将它改名为t

import TestLib as t print(t.lib\_func(120)) //这样的好处是减小所引用模块的内存占用。

1. 查询函数的参数：

当我们想要知道某个函数会接收哪些参数的时候，可以使用下面方法查询。

import inspect print(inspect.getargspec(func))

1. Matplotlib可能是 Python 2D- 绘图领域使用最广泛的套件。它能让使用者很轻松地将数据图形化，并提供多样化的输出格式。

Basemap是Matplotlib的一个子包，负责地图绘制。在数据可视化过程中，我们常需要将数据在地图上画出来。比如说我们在地图上画出城市人口，飞机航线，军事基地，矿藏分布等等。

1. 你可以添加，删除，修改类的属性，如下所示：

emp1.age = 7 # 添加一个 'age' 属性

emp1.age = 8 # 修改 'age' 属性

del emp1.age # 删除 'age' 属性

你也可以使用以下函数的方式来访问属性：

getattr(obj, name[, default]) : 访问对象的属性。

hasattr(obj,name) : 检查是否存在一个属性。

setattr(obj,name,value) : 设置一个属性。如果属性不存在，会创建一个新属性。

delattr(obj, name) : 删除属性。

hasattr(emp1, 'age') # 如果存在 'age' 属性返回 True。

getattr(emp1, 'age') # 返回 'age' 属性的值

setattr(emp1, 'age', 8) # 添加属性 'age' 值为 8

delattr(empl, 'age') # 删除属性 'age'

析构函数 \_\_del\_\_ ，\_\_del\_\_在对象消逝的时候被调用，当对象不再被使用时，\_\_del\_\_方法运行：

\_\_str\_\_( self )，用于将值转化为适于人阅读的形式。

1. 运算符重载：

def \_\_add\_\_(self,other):

return Vector(self.a + other.a, self.b + other.b)

v1 = Employee(2,10)

v2 = Employee (5,-2)

print v1 + v2 => Vector(7,8)

1. \_\_private\_attrs：两个下划线开头，声明该属性为私有，不能在类地外部被使用或直接访问。在类内部的方法中使用时 self.\_\_private\_attrs。

class Employee:

\_\_secretCount = 0 # 私有变量

publicCount = 0 # 公开变量

Python不允许实例化的类访问私有数据，但你可以使用 **object.\_className\_\_attrName** 访问属性，如：print counter.\_Employee\_\_secretCount

1. Python内置类属性

\_\_dict\_\_ : 类的属性（包含一个字典，由类的数据属性组成）

\_\_doc\_\_ :类的文档字符串

\_\_name\_\_: 类名

\_\_module\_\_: 类定义所在的模块（类的全名是'\_\_main\_\_.className'，如果类位于一个导入模块mymod中，那么className.\_\_module\_\_ 等于 mymod）

\_\_bases\_\_ : 类的所有父类构成元素（包含了以个由所有父类组成的元组）

1. thread 模块提供的其他方法(线程)：

threading.currentThread(): 返回当前的线程变量。

threading.enumerate(): 返回一个包含正在运行的线程的list。正在运行指线程启动后、结束前，不包括启动前和终止后的线程。

threading.activeCount(): 返回正在运行的线程数量，与len(threading.enumerate())有相同的结果。

除了使用方法外，线程模块同样提供了Thread类来处理线程，Thread类提供了以下方法:

run(): 用以表示线程活动的方法。

start():启动线程活动。

join([time]): 等待至线程中止。这阻塞调用线程直至线程的join() 方法被调用中止-正常退出或者抛出未处理的异常-或者是可选的超时发生。

isAlive(): 返回线程是否活动的。

getName(): 返回线程名。

setName(): 设置线程名。

# 开启线程 => thread1.start()

1. 线程同步：

使用Thread对象的Lock和Rlock可以实现简单的线程同步，这两个对象都有acquire方法和release方法，对于那些需要每次只允许一个线程操作的数据，可以将其操作放到acquire和release方法之间。对于那些需要每次只允许一个线程操作的数据，可以将其操作放到acquire和release方法之间。

threadLock = threading.Lock()

# 获得锁，成功获得锁定后返回True

# 可选的timeout参数不填时将一直阻塞直到获得锁定

# 否则超时后将返回False

threadLock.acquire()

print\_time(self.name, self.counter, 3)

# 释放锁

threadLock.release()

threads = []

# 添加线程到线程列表

threads.append(thread1)

threads.append(thread2)

# 等待所有线程完成

for t in threads:

t.join()

1. 线程优先级队列（Queue）

Python的Queue模块中提供了同步的、线程安全的队列类，包括FIFO（先入先出)队列Queue，LIFO（后入先出）队列LifoQueue，和优先级队列PriorityQueue。

Queue模块中的常用方法(import Queue)

Queue.qsize() 返回队列的大小

Queue.empty() 如果队列为空，返回True,反之False

Queue.full() 如果队列满了，返回True,反之False

Queue.full 与 maxsize 大小对应

Queue.get([block[, timeout]])获取队列，timeout等待时间

Queue.get\_nowait() 相当Queue.get(False)

Queue.put(item) 写入队列，timeout等待时间

Queue.put\_nowait(item) 相当Queue.put(item, False)

Queue.task\_done() 在完成一项工作之后，Queue.task\_done()函数向任务已经完成的队列发送一个信号

Queue.join() 实际上意味着等到队列为空，再执行别的操作

1. 文件操作：

import os

fileHandle = open ( 'test.txt', 'w' )

fileHandle.write ( 'This is a test.\nReally, it is.' )

filehandle = open(test.txt')

print print fileHandle.read()

print fileHandle.readline() //返回的只是一行的一部分

fileList = fileHandle.readlines() //把文件每一行作为一个list的一个成员，并返回这个list

fileHandle.seek ( 0 ) //从头开始读

fileHandle.close()

w 以写方式打开，

a 以追加模式打开 (从 EOF 开始, 必要时创建新文件)

r+ 以读写模式打开

w+ 以读写模式打开 (参见 w )

a+ 以读写模式打开 (参见 a )

rb 以二进制读模式打开

wb 以二进制写模式打开 (参见 w )

ab 以二进制追加模式打开 (参见 a )

rb+ 以二进制读写模式打开 (参见 r+ )

wb+ 以二进制读写模式打开 (参见 w+ )

ab+ 以二进制读写模式打开 (参见 a+ )

使用“os.path”来获取基本信息

if os.path.isdir ( fileStats ):

print 'Directory.'

elif os.path.isfile ( fileStats ):

print 'File.'

目录操作：

for fileName in os.listdir ( '/' ): //列出一个目录的内容

print fileName

os.mkdir ( 'testDirectory' ) //创建目录

os.rmdir ( 'testDirectory ) //删除目录

os.makedirs ( 'I/will/show/you/how/deep/the/rabbit/hole/goes' )  //创建多级目录

os.removedirs ( 'I/will/show/you/how/deep/the/rabbit/hole/goes' ) //删除多级目录

读文件内容：

fileHangler=open('/home/kk/test/smb.conf','r')

for line in fileHangler.readlines():

print line,

fileHangler.close()

1. fnmatch模块 使用模式来匹配文件名

for fileName in os.listdir ( '/' ):   
if fnmatch.fnmath ( fileName, '\*.txt' ):   
 print open ( fileName ).read()

1. re模块：对正则表达式的支持
2. match方法是对字符串的开头进行匹配，如果匹配到则返回一个Match对象，如果匹配失败，则返回None，group是Match对象的方法，表示得到的某个组的匹配，如果使用分组来查找字符串的各个部分，就可以通过group得到每个组匹配到的字符串。不写 数字就是第0组，表示与整个表达式匹配的字符串，如果要得到第1组则是match.group(1)，第二组是match.group(2)……

atch = re.match(r'This', 'This is re module of python')

print match.group()

1. other
2. json模块：

<http://deron.meranda.us/python/demjson/>

$tar xvfz demjson-1.6.tar.gz

$cd demjson-1.6

$python setup.py install

json.dumps() demjson. encode()

//把一个Python对象编码转换成Json字符串

json.loads() json.decode()

//把Json格式字符串解码转换成Python对象

例：

print json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])

print json = demjson.encode([ { 'a' : 1, 'b' : 2, 'c' : 3, 'd' : 4, 'e' : 5 } ])

1. glob模块：
2. [Python调用Linux下的动态库(so)](http://blog.chinaunix.net/uid-25979788-id-3828375.html)

1) 编写linuxany.c代码

#include "stdio.h"

void display(char\* msg){

printf("%s\n",msg);

}

int add(int a,int b){

return a+b;

}

2) gcc -shared linuxany.c -o linuxany.so，将生成一个linuxany.so文件

3) 在Python中调用

from ctypes import \*

//参数为生成的.so文件所在的绝对路径

libtest = cdll.LoadLibrary(os.getcwd() + '/linuxany.so')

//直接用方法名进行调用

print libtest.display('Hello,I am linuxany.com')

print libtest.add(2,2010)

1. urllib模块

<http://www.cnblogs.com/sysu-blackbear/p/3629420.html>

1. urllib.urlopen(url[,data[,proxies]])

打开一个url的方法，返回一个文件对象，然后可以进行类似文件对象的操作

urlopen返回对象提供方法：

- read() , readline() ,readlines() , fileno() , close() ：这些方法的使用方式与文件对象完全一样

- info()：返回一个httplib.HTTPMessage对象，表示远程服务器返回的头信息

- getcode()：返回Http状态码。如果是http请求，200请求成功完成;404网址未找到

- geturl()：返回请求的url

1. urllib.urlretrieve(url[,filename[,reporthook[,data]]])

urlretrieve方法将url定位到的html文件下载到你本地的硬盘中。如果不指定filename，则会存为临时文件。

urlretrieve()返回一个二元组(filename,mine\_hdrs)

1. urllib.urlcleanup() //清除由于urllib.urlretrieve()所产生的缓存
2. urllib.quote(url)和urllib.quote\_plus(url)

将url数据获取之后，并将其编码，从而适用与URL字符串中，使其能被打印和被web服务器接受。

urllib.quote('http://www.baidu.com') ==> 'http%3A//www.baidu.com'

urllib.quote\_plus('http://www.baidu.com') ==> 'http%3A%2F%2Fwww.baidu.com'

1. urllib.unquote(url)和urllib.unquote\_plus(url) //与4的函数相反。
2. urllib.urlencode(query) //将URL中的键值对以连接符&划分

这里可以与urlopen结合以实现post方法和get方法：

1. other

例:

1. f = urllib.urlopen('http://www.baidu.com').readline()   //读取html页面的第一行
2. f = urllib.urlretrieve('http://www.baidu.com') print f[0] //临时存放:
3. f=urllib.urlretrieve('http://www.baidu.com',filename='/home/dzhwen/pytho/baidu.html')

//存为本地文件

1. struct模块
2. struct.pack用于将Python的值根据格式符，转换为字符串（因为Python中没有字节(Byte)类型，可以把这里的字符串理解为字节流，或字节数组）。其函数原型为：struct.pack(fmt, v1, v2, ...)，参数fmt是格式字符串，关于格式字符串的相关信息下面有所介绍。v1, v2, ...表示要转换的python值。

a,b=20,400  
str=struct.pack('ii',a,b) #转换成字节流，虽然还是字符串，但是可以在网络上传输  
print len(str) #ii 表示两个int



1. struct.calcsize():用来计算特定格式的输出的大小，是几个字节

struct.calcsize('HH4s') ==> 8

1. wx模块 www.wxpython.org

sudo apt-cache search wxpython。如果能返回wxpython的信息，则直接输入命令

sudo apt-get install python-wxtools安装即可。

1. CGI文件的扩展名为.cgi，python也可以使用.py扩展名。

默认情况下，Linux服务器配置运行的cgi-bin目录中为/var/www。

如果你想指定其他运行CGI脚本的目录，可以修改httpd.conf配置文件

1. 注意事项：
2. Python的变量不需要声明，你可以直接输入： >>>a = 10
3. print的另一个用法，也就是print后跟多个输出，以逗号分隔。
4. 变量不需要声明，不需要删除，可以直接回收适用。
5. 字符串是元组
6. Python最具特色的是用缩进来标明成块的代码，Python中不能随意缩进。
7. 在编译时会出现这样的错***IndentationError:expected an indented block***说明此处需要缩进，你只要在出现错误的那一行，按空格或Tab（但不能混用）键缩进就行
8. 实验一些list的方法：

>>>print nl.count(5)       # 计数，看总共有多少个5

>>>print nl.index(3)       # 查询 nl 的第一个3的下标

>>>nl.append(6)            # 在 nl 的最后增添一个新元素6

>>>nl.sort()               # 对nl的元素排序

>>>print nl.pop()          # 从nl中去除最后一个元素，并将该元素返回。

>>>nl.remove(2)            # 从nl中去除第一个2

>>>nl.insert(0,9)          # 在下标为0的位置插入9

总之，list是一个类。每个列表都属于该类。

1. 函数可以作为一个对象，进行参数传递。 test(func, 3, 5)
2. 使用中文(以及其它非ASCII编码)：在Python程序的第一行加入#coding=utf8例如:#coding=utf8 print("你好吗？")
3. 表示2进制，8进制和16进制数字

在2.6以上版本，以如下方式表示

print(0b1110) # 二进制，以0b开头

print(0o10) # 八进制，以0o开头

print(0x2A) # 十六进制，以0x开头

如果是更早版本，可以用如下方式：

print(int("1110", 2))

print(int("10", 8))

print(int("2A", 16))

1. 注释：

一行内的注释可以以#开始

多行的注释可以以'''开始，以'''结束

1. 搜索路径:

我们可以通过下面方法来查看搜索路径：

import sys

print(sys.path)

下面我们增加/home/vamei/mylib到搜索路径中：

$export PYTHONPATH=$PYTHONPATH:/home/vamei/mylib

你可以将正面的这行命令加入到～/.bashrc中。这样我们就长期的改变了搜索路径。

1. 可以使用下面方法运行一个Python脚本，在脚本运行结束后，直接进入Python命令行。这样做的好处是脚本的对象不会被清空，可以通过命令行直接调用。

$python -i script.py

1. 安装非标准包

Python的标准库随着Python一起安装。当我们需要非标准包时，就要先安装。

1) 使用Linux repository(Linux环境)

这是安装Python附加包的一个好的起点。你可以在Linux repository中查找可能存在的Python包 (比如在Ubuntu Software Center中搜索matplot)。

1. 使用pip。pip是Python自带的包管理程序，它连接Python repository，并查找其中可能存在的包。

sudo apt-get install python-pip

比如使用如下方法来安装、卸载或者升级web.py：

$pip install web.py

$pip uninstall web.py

$pip install --upgrade web.py

如果你的Python安装在一个非标准的路径(使用$which python来确认python可执行文件的路径)中，比如/home/vamei/util/python/bin中，你可以使用下面方法设置pip的安装包的路径:

$pip install --install-option="--prefix=/home/vamei/util/" web.py

1. print "aa is %d" %aa => aa is 10
2. pass是空语句，是为了保持程序结构的完整性，什么也不做
3. 当运行单个python文件时，如运行a.py，这个时候a的一个属性\_\_name\_\_是\_\_main\_\_。

当调用某个python文件时，如b.py调用a.py，这个时候a的属性\_\_name\_\_是模块名a。

1. python
2. 字符串处理函数：str='python String function'
3. 字符串长度获取：len(str) print '%s length=%d' % (str,len(str))
4. 全部大写：str.upper() print '%s lower=%s' % (str,str.lower())
5. 全部小写：str.lower()
6. 大小写互换：str.swapcase()
7. 首字母大写，其余小写：str.capitalize()
8. 首字母大写：str.title()

**格式化相关:**

1. 获取固定长度，右对齐，左边不够用空格补齐：str.rjust(width)

print '%s ljust=%s' % (str,str.ljust(20))

1. 获取固定长度，左对齐，右边不够用空格补齐：str.ljust(width)
2. 获取固定长度，中间对齐，两边不够用空格补齐：str.center(width)
3. 获取固定长度，右对齐，左边不足用0补齐: str.zfill(width)

**字符串搜索相关:**

1. 搜索指定字符串，没有返回-1：str.find('t')
2. 指定起始位置搜索：str.find('t',start)
3. 指定起始及结束位置搜索：str.find('t',start,end)
4. 从右边开始查找：str.rfind('t')
5. 搜索到多少个指定字符串：str.count('t')
6. 上面所有方法都可用index代替，不同的是使用index查找不到会抛异常，而find返回-1

**字符串替换相关:**

1. 替换old为new：str.replace('old','new')

print '%s replace t to \*=%s' % (str,str.replace('t', '\*'))

1. 替换指定次数的old为new：str.replace('old','new',maxReplaceTimes)

**字符串去空格及去指定字符:**

1. 去两边空格：str.strip()
2. 去左空格：str.lstrip()
3. 去右空格：str.rstrip()
4. 去两边字符串：str.strip('d')，相应的也有lstrip('d')，rstrip('d') //指定截取的字符
5. 按指定字符分割字符串为数组：str.split(' ') **默认按空格分隔**

str='a b c de' print '%s strip=%s' % (str,str.split()) ==>a b c de strip=['a', 'b', 'c', 'de']

**字符串判断相关: 返回True或False**

1. 是否以start开头：str.startswith('start')
2. 是否以end结尾：str.endswith('end')
3. 是否全为字母或数字：str.isalnum()
4. 是否全字母：str.isalpha()
5. 是否全数字：str.isdigit()
6. 是否全小写：str.islower()
7. 是否全大写：str.isupper()
8. str.istitle() //检测字符串中所有的单词拼写首字母是否为大写，且其他字母为小写

r = "This Is String Example...Wow!!!";

print r.istitle(); 或 print str.istitle(r)

1. other
2. sys模块：
3. sys.argv[]是用来获取命令行参数的, sys.argv[0]表示代码本身文件路径;比如在[CMD](https://www.baidu.com/s?wd=CMD&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1dBPWP-PAf4nAcdPWczujfk0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6K1TL0qnfK1TL0z5HD0IgF_5y9YIZ0lQzqlpA-bmyt8mh7GuZR8mvqVQL7dugPYpyq8Q1RkrjfzrjTsns)命令行输入 “python going.py -help", 那么sys.argv[0]就代表“going.py”,sys.argv[1] 就代表 –help //sys.argv
4. sys.platform //获取运行平台的信息
5. sys.exit(n) //如果需要中途退出程序, 你可以调用sys.exit 函数 //sys.exit(0)
6. sys.path //添加自己的模块路径。 //sys.path.append(“my module path”).
7. other
8. 文件I/O操作

例1：

def modify(filepath,sstr,rstr):

fileHandle=open(filepath,'r')

lines=fileHandle.readlines()

#print lines

flen=len(lines)

for i in range(flen):

if sstr in lines[i]:

lines[i]=' '+sstr+'='+rstr+'\n'

for i in range(flen):

print lines[i],

fileHandle.close()

#modify('./smb.conf','map to guest','bad user')

modify('./smb.conf','map to guest','no')

例2：

1. Socket模块

<http://blog.csdn.net/rebelqsp/article/details/22109925>

1. s.bind()接受的参数是一个pair （pair (host, port)）或者是tuple(元组tuple (ifname, proto [,pkttype [,hatype]])')只接受一个参数，那么就需要一个括号把所需要传送的参数括起来，组成一个 sock.bind((“127.0.0.1”,12345))
2. 创建TCP Socket： s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

创建UDP Socket： s=socket.socket(socket.AF\_INET,socket.SOCK\_DGRAM)

1. s.listen(backlog) 开始监听TCP传入连接。backlog指定在拒绝连接之前，操作系统可以挂起的最大连接数量。该值至少为1，大部分应用程序设为5就可以了。
2. s.accept() 接受TCP连接并返回（conn,address）,其中conn是新的套接字对象，可以用来接收和发送数据。address是连接客户端的地址。
3. s.connect(address) 连接到address处的套接字。一般address的格式为元组（hostname,port），如果连接出错，返回socket.error错误。
4. s. setsockopt(level,optname,value) //设置socket选项

level定义了哪个选项将被使用。通常情况下是SOL\_SOCKET，意思是正在使用的socket选项。

Optname 选项SO\_REUSEADDR 当socket关闭后，本地端用于该socket的端口号立刻就可以被重用。通常来说，只有经过系统定义一段时间后，才能被重用。

s.setsockopt(socket.SOL\_SOCKET,socket.SO\_REUSEADDR,1)

//这里value设置为1，表示将SO\_REUSEADDR标记为TRUE，操作系统会在服务器socket被关闭或服务器进程终止后马上释放该服务器的端口，否则操作系统会保留几分钟该端口。

1. other

服务器端:

import socket

import commands #执行系统命令模块

s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.bind((“192.168.1.104”,12345))

s.listen(1)

while 1:

conn,addr = s.accept()

print "connected by ",addr

while 1:

data = conn.recv(1024)

cmd\_status,cmd\_result = commands.getstatusoutput(data)

print "status:",cmd\_status

if len(cmd\_result.strip()) == 0:

conn.sendall("done.")

else:

conn.sendall(cmd\_result)

conn.close()

客户端:

s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.connect(("192.168.1.104",12345))

while 1:

cmd = raw\_input("enter:")

s.sendall(cmd)

data = s.recv(1024)

print data

s.close()

bind机器上所有ip的方法:

s = socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)

s.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

s.bind(("0.0.0.0",8099))

s.listen(5)

1. 系统处理