python的生态非常好（库多），虽然运行效率低、占用内存高，但是还是因此“人生苦短请用Python”。除了运行时的效率性能问题，还有代码可读性，因为不强制性要求指明参数类型，导致函数一旦参数过多且变量命名不适合阅读。

PHP 从语言层面上讲几乎是一无是处，具体实现的质量也乏善可陈，但它胜在最要命的部署上：没有任何其他语言有像 PHP 一样适合大规模部署的方式。基本上装好 Apache/mod\_php 之后，PHP 应用的部署就简化为了复制文件。即便是考虑到性能原因等采用 nginx/FastCGI 等替代方式，额外的工作也只是在于最初配置。一旦配置完成，之后的部署都是文件复制。服务器重启后通常会自动启动 apache/nginx，fastcgi manager 负责启动 php-cgi 进程等。不过现在已经有了docker。这和 Python 之流的部署有天壤之别：大部分 Python 的网络应用如 Django, Tornado 等都需要单独的常驻进程（Apache/mod\_python 似乎是个杯具）。这些进程需要额外的维护工作以管理其启动、停止，也需要额外的监控进程处理意外退出后的重启。这需要用户对系统有更深入的了解。常驻进程需要占用系统内存，通常不可能在一台服务器上运行成百上千个应用，除非平台提供单物理机上唯一常驻进程支持。常驻进程也要求作者对系统资源的管理、垃圾回收机制等有更深入的了解以避免内存泄露、资源占用过度等问题。现在 Python 网络应用部署最简单的应该是 App Engine，采用了类似 PHP 的生命周期（请求处理 30 秒限制，超时被终止，无法运行常驻进程）。所以从流行程度上讲，Python 不会超过 PHP，因为数量庞大的服务提供商无法用 Python 支撑现有的用户规模。

工具：pip（包管理） virtualenv（项目内环境管理）

python库：dir()查看成员与方法

* 算法：

数字交换：

第三方变量交换，python中可以缩写为 s[l],s[r] = s[r],s[l]（多变量复制同写法）

非0数字可以异或交换 a^=b^=a^=b 3次异或，如果仅两次其中一个值会是中间值。python估计表达式的值不是左值，不支持这种连续运算写法，需要拆开写。

* 科学（不记忆）：

Anaconda指的是一个开源的[Python](https://baike.baidu.com/item/Python)发行版本，其包含了conda、Python等180多个科学包及其依赖项。 [1]  因为包含了大量的科学包，Anaconda 的下载文件比较大（约 531 MB），如果只需要某些包，或者需要节省带宽或存储空间，也可以使用Miniconda这个较小的发行版（仅包含conda和 Python）。

1. python import

import sys <==>sys = \_\_import\_\_('sys')

而import的位置可以在文件中，也可以在代码段中这时成为“delay import”。不过import底层是一个数组，如果重复import了同一个包，会在数组中delete，再在末尾插入。导致delay import可能存在性能问题。

1. Python没有花括号

为了吸引更多的用户，使他们更快适应Python代码缩进的语法同时又能兼容他们不可抑制地输入{}的冲动，\_\_future\_\_还提供了braces特性，使花括号可以被正确解析！

使用：from \_\_future\_\_ import braces

>>> from \_\_future\_\_ import braces #这里的braces 指的是：curly braces（花括号）

File "<stdin>", line 1

SyntaxError: not a chance

当然这仅仅是一个玩笑，想用花括号定义函数？没门！

1. 耗时优化

直接用Python运行的时间目测要上小时，不想费时间优化，改用PyPy解释器，2分钟跑完。由此发现对于纯python代码，尤其是循环较多的情况下，PyPy比CPython真的要快上好多倍。如果你不用外部 C 扩展的话，那 PyPy 已经是比较完善的 drop-in replacement 了。PyPy有着比较好的对循环的JIT，所以速度可以快很多：

C因为跟底层过于接近，很难有相同的代码段，所以实现JIT解释器有诸多限制。（比如Psyco到项目死亡为止都没能实现出64位的JIT解释器）。Pypy则换了另外一种思路。它先实现了一个Python的子集（注意，不是完整的python），叫RPython。然后用RPython去实现了Python的JIT解释器。这个RPython本身，并不依赖运行时解释器，而是直接被翻译成C代码（实际上可以翻译成多种目标代码，如Java、C#等）再进行编译，本质上它是一种编译型语言。所以，用RPython写出来的程序，最终是会被编译成本地代码的，跟C写的没有本质区别。由于RPython作者强大的优化功力，RPython程序最终编译结果基本等同于C直接写的效率。用这种程序实现出来的JIT解释器，自然也不会慢。而JIT技术，又保证了运行在这个解释器上的Python程序的效率的提升。

Django这种框架，会有协助性能分析优化的插件。

1. 报错：

IndentationError: unexpected indent

python依靠indent来识别。要么全部用空格缩进，要么全部tab键。

1. 与其他语言：可以调用C，C++库；可与Java组建集成
2. 编译与运行

Python的不同实现

* CPyhon 以C编写的python解释器（目前应用最广泛的解释器）
* Jython 以Java编写的python解释器
* pypy 以python编写的python解释器，集成了JIT，但Pypy无法支持官方的C/Python API，导致无法使用例如Numpy，Scipy等重要的第三方库。这也是现在Pypy没有被广泛使用的原因吧。

Shell就是我们最常见的解释器。不过python多了一步字节码编译，和java类似，保存为.pyc文件（容易复原成python程序的）。Python的虚拟机就成为PVM，编译后的字节码文件在PVM运行。

6.py可执行脚本与shell脚本执行方法类似。在Unix系统中就是以#!开头(指定执行文件也就是后面加which python找到的路径，但是这样并不具有可移植性，所以还有一种方式：#!/usr/bin/env python)，并给指定的程序可执行权限。不过对于python还可以不这么使用，而直接靠如下操作：python xxxx.py

7. python3支持任意长度整数（和Java一样有整数预分配机制）比如2\*\*2000，如果真的想要了解细节[可以看看](https://python.freelycode.com/contribution/detail/1788) /[英文版](https://arpitbhayani.me/blogs/super-long-integers)。在2.X版本中整数还分为一般整数与长整数。具体多少位还要看是那个版本。3.X里面没有这种区分，只有整数这一种类型。

8. 二进制以0b或者0B开头，八进制以0o或者0O（零与字母O）开头（从前版本有仅仅0开头，但是不要使用），十六进制以0x或者0X开头。将十进制转为其他进制，对应函数是bin(i),oct(i),hex(i)（不过值得注意的是输出是文本形式）。反向函数（将文本形式的非十进制文本转为某进制整数）是int(str, base)（由于这个函数指定了base，所以字符串里面可以不必用那些指明来开头），python支持2-36进制，所谓36进制就是指0-9a-z。

浮点数是按照64位存储的，也是17位精度(五位小数)。python2.x与python3.0浮点数计算得到的都是不一样的，比如说2/3。

复数形式是按照函数来实现的，complex(实数，虚数)

分数，使用Fraction(分子，分母)来表示。

类型的升级：整型<分数<浮点数（可以用e表示很大的范围）< 复数

>>> x = Fraction(1, 3)

>>> y = Fraction(5, 2)

>>> x + y

Fraction(17, 6)

特殊：float('inf')无穷大，float('-inf')无穷小，float('nan')非数字（也就是NAN）

>>>1/float('inf') #居然允许这么除

0.0

>>>1/float('-inf')

-0.0 #0也有正负之分

>>> float('inf')

inf

>>>float('nan')

nan

>>> 1/float('nan')

nan

>>> math.inf == float('inf')

True

运算符：

优先级：算术 > 位 > 身份(is, is not) >成员(in, not in) > 比较 > 逻辑 > 赋值。

算术运算符 没有自增与自减运算符

最高：\*\*（幂运算，阶乘）

次之：\*, /, //(整除), %,

次之：+, -

比较运算符 >, <, >=, <=, ==, !=

布尔运算符 and优先级高于or，没有&&、||等

位运算符 &, |, ^, ~, <<, >>（& | - ^对set是交并差补）

>>> 1+1>>2 说明>>优先级没有高于+

0

>>> 1+(1>>2)

1

身份运算符:is, is not

is 用于判断两个变量是否引用同一个 会对比其中两个变量的地址。之前说python对于整数常量，小范围内会是同一个地址，但大整数：

>>> 9999999999999 is 9999999999999

True

>>> a= 9999999999999

>>> b= 9999999999999

>>> a is b

False

>>> a = 1

>>> b = 1

>>> a is b

True

== 比较值

测试：

10>0 == 3>0 #False

(10>0)== (3>0) #True

(True==3)==0 #True

True==3==0 #False（**同a<b<c，需要每个都成立才是True**）

True==1 #True

True==3 #False（说明是True被转换成1，故10>True）

False==0 #True

下面按照每一个表达式都成立去理解：

10>1==1==1 #True

10>0==0==0 #True

10>1==0==0 #False

=的并行赋值（在单链表反转中会用到）：将等式右边计算后（如果有函数就执行）结果保存为一个元组，并按照从左到右顺序对等式左边进行赋值。对于单链表反转:

while cur:  
 # prev, cur, cur.next = cur, cur.next, prev cur被cur.next复制可能会是None，因此链表反转时左侧表达式 cur.next要在cur之前  
 cur.next, cur, prev = prev, cur.next, cur  
 # 或 cur.next, prev, cur = prev, cur, cur.next

>>> [1, 1] \* True <=> [1, 1]\*1

[1, 1]

>>> [1] \* False <=> [1]\*0

[]

>>> 1 \* False

0

>>> 1 \* True

1

for p in [pos - 1, pos + 1] + map\_array[num] \* (num not in num\_met):

xxxxx

除法全都会转为浮点数来运算，所以有个整除符号：//

三元运算符：xxx if 条件 else xxx 没有三元运算符

聚合条件表达式：nums[l]==nums[m]==nums[r]

引号与双引号：

>>>a = "hello 'hh' "

>>>a

"hello 'hh'"

>>> a ='hello "hh"'

>>> a

'hello "hh"'

绝对值 abs()（内置函数，可用于整型与浮点等类型） math.fabs()

成员运算符:in, not in

使用in符号判断指定的元素是否存在于列表中，但我发现元组和数组存在区别

>>> 'jb51.net' in ['haotu.net','jb51.net']

True

>>> 'jb51.net' in ('haotu.net','jb51.net')

True

>>> 'jb51.net' in ['jb51.net/codes','haotu.net']

False

>>> 'jb51.net' in ('jb51.net/codes','haotu.net')

False

>>> 'jb51.net' in ['jb51.net/codes']

False

>>> 'jb51.net' in ('jb51.net/codes') #表示很无语

True

循环

while True

while i1 < len(nums)

for i1 in range(len(nums))

for i1 in range(0, len(nums))

for i in range(0,-2) 相当于不会跑，所以可以for i in range(len(nums)-2)

for i,value in enumerate(arr)

内置函数

成员函数返回None，内置函数有返回结果

**sum**

如果对map使用，用于map的key上

sum(m.values()) # m是map

**max/min 多参数**，能用于list等

如果传空会报，所以不能写 return 1 == max(cv) or len(cv)==0

>>> a = [1,2,3]

>>> max(a)

3

>>> max(1,2,3)

>>> 3

>>> max((total -v) \* v for v in vals) % (10 \*\* 9 + 7)

# 使用key lambda找出最接近半值元素

nodeSums = set()

n = min(nodeSums, key=lambda x: abs(x - treeSum / 2))

**sorted** 复制后排序，返回排序结果：key控制比较对象，相当于其他语言中的compare函数。sorted(tuple\_list, key=lambda x: x[1])。functools.cmp\_to\_key如果元组里第一个元素是奇数，就用元组里第一个元素进行排序，如果元组里第一个元素是偶数，则用这个元组里的第二个元素进行大小比较。reverse 布尔值(但可以用1、0)。

>>>sorted(b, key=lambda x:x[0])

[[0, 0], [1, 1]]

>>>b #sorted不会直接改变原数组

[[1, 1], [0, 0]]

>>>sorted(b, key=lambda x:x[0], reverse=True) # reverse=1意外倒也行

[[1, 1], [0, 0]]

>>>from functools import cmp\_to\_key

>>>sorted(b, key=cmp\_to\_key(lambda x,y:x[0]<y[0]), reverse=True)

[[1, 1], [0, 0]]

>>>sorted(words, key=len) # 对字符串数组按照长度从小到大

>>>sorted(words, key=lambda w: len(w)) # 对字符串数组按照长度从小到大

>>>sorted(sCounter.items(), key=lambda x:x[1], reverse=True)

**abs** # 支持整数、浮点数等

**pow** 次方或次根

>>> pow(2,2)

4

>>> pow(2.0,2)

4.0

>>> pow(4,1/2)

2.0

**enumerate** 可以用于 字符串、列表等可枚举对象中

可枚举元素处理(for循环、lambda+filter/map/reduce)

**map(function\_to\_apply, list\_of\_inputs)** 将 可迭代变量 中的每一个元素，经过f(x)映射，返回迭代器

incremented\_list = [x+1 for x in l]

<=> incremented\_list = map(lambda x: x+1, l)

filtered\_list = [x for x in l if x!=1]

<=> filtered\_list = list(filter(lambda x:x!=1, l))

items = [1, 2, 3, 4, 5]  
squared = list(map(lambda x: x\*\*2, items))

**reduce(function, iterable[, initializer])** 在迭代序列的过程中，首先把 前两个元素（只能两个）传给 函数，函数加工后，然后把 得到的结果和第三个元素 作为两个参数传给函数参数， 函数加工后得到的结果又和第四个元素 作为两个参数传给函数参数，依次类推。



sum2 = reduce(lambda x, y: x+y, [1,2,3,4,5]

# 通过reduce实现阶乘  
result = reduce(lambda x, y: x \* y, range(1, 11))

filter 也是返回迭代器

list(filter(lambda x:x >6 and x<8,range(10)))

<=> [i for i in range(10) if i >6 and i <8]

**zip “打包”生成元组**

sorted(zip(efficiency, speed), reverse=True)

**any、all 接收元素可转布尔的可迭代对象，返回布尔值**

>>> any([True, False])

True

>>> all([True, False])

False

**ord**(‘z’)即122 ，**chr**(122)即’z’

a-z: [chr(c) for c in range(ord('a'), ord('z')+1)]

数组

无GIL锁，非线程安全

初始化:

[1]\*0 #空数组

[1]\*1 #[1]，如果需要将list自己与自己拼起来，最好方式\*2

[1 for i in range(1)]

c=[a[i]+b[i] for i in range(min(len(a),len(b)))]

[[0] \* n for \_ in range(n)] # 千万别写成[[0\* n] for \_ in range(n)]

[[0] \* n] \* n # 切完不能这么写，n行row是引用同一个

[num for num in nums if num&1==0] 是一种快速的filter方式

合并 [1]+[2] 得到[1, 2]

因此有这种方式删除元素 nums1 = nums1[:i]+nums1[i+1:]，但因为是复制，所以是O(N)。和append相比优势在于有返回值。

相等 python2的cmp()等价于==，python3用==

数组复制 hex(id(a[:])) #hex(id())查地址，复制时可以配合索引进行截断

l1 = [1,2,3,[1,3]]

l2 = l1[:] # 只是浅复制了第一层 相当于 copy.copy()，可用copy.deepcopy

l1[3][1] = 9

list(l1)

l1\*1

l1+[]

copy.copy(l1)

copy.deepcopy(l1) # 可以拷贝树、图

print(l1) # [1, 2, 3, [1, 9]]

print(l2) # [1, 2, 3, [1, 9]]

引用传递：

def reassign\_but\_not\_change(val: List):  
 val = [1, 2, 3, 4]  
  
  
def change(val: List):  
 val.append(4)

val[0] = 0  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # o = Solution()  
 # print(o.hasAllCodes("00110", 2))  
 val = [1, 2, 3]  
 reassign\_but\_not\_change(val)  
 print(val) # [1, 2, 3]  
 change(val)  
 print(val) # [0, 2, 3, 4]

索引运算（对字符串也有效）

>>> a

[1, 2, 3, 4]

>>> a[-1]

4

>>> a[-1:]

[4]

>>> a[-1:2] index有效就不会异常，范围无效就返回空数组

[]

>>> a[0:0]

[]

>>> a[0:-1]

[1, 2, 3]

>>> a[0:3]

[1, 2, 3]

>>> a[::-1] <=> a[-1::-1] <=> a[3::-1]

[4, 3, 2, 1]

>>> a[::-2]

[4, 2]

>>> a[-1:-3]=[2] #没想到是加了一个元素 [-3:-1]才是替换

>>> a

[1, 2, 3, 2, 4]

>>> a = [1,2,3,4]

>>> a[1:3]=[2] # 这时候是替换删除了一个元素

>>> a

[1, 2, 4]

上面能修改成功我觉得是引用，但是成员函数sort只对整个数组有效且是直接修改数组

>>> a = [4,3,2,1]

>>> a[1:3].sort()

>>> a

[4, 3, 2, 1]

>>> print(a[1:3].sort())

None

>>> b = a[1:3]

>>> b.sort()

>>> a

[4, 3, 2, 1]

>>> b

[2, 3]

>>> a = [4,3,2,1]

>>> a[1:4]=sorted(a[1:3]) #通过排序的返回值，对数组进行复制与部分替换实现部分排序

>>> a

[4, 2, 3]

相等 ==

返回数组中的最值 内置函数max() min()

在指定索引插入元素 .insert()

返回某一个值的最左索引 .index()

添加单个元素 .append()

通过另外一个数组添加多个元素 += .extend() 返回值为None

排序 .sort()返回None sorted()返回被排序,可以部分排序a[1:3] = sorted(a[1:3])

遍历

for value in array 值

for index in range(len(array))

for index in range(0, len(array))

for i in range(nums[:]) 拷贝原数组并进行遍历

for i in range(len(nums)-1, -1, -1) 倒叙遍历

for i in range(0, len(nums))[::-1]: 倒叙遍历

for i,num in enumerate(nums) 遍历索引与元素

删除

删除第一个匹配到的数据 .remove() 如果不存在会抛出异常，可结合in

删除所有匹配到数据

[x for x in array if x!=1]

remove+异常/in

按照index删除 del a[1]

按照index删除并返回数据 .pop(1)，不传即删除末尾元素基于此可实现堆栈，.pop(0)实现队列，空列表将抛出异常。没有push（而叫.append），

相加（两数组同索引运算）

c=[a[i]+b[i] for i in range(min(len(a),len(b)))]

c=list(map(lambda x :x[0]+x[1] ,zip(a,b)))

import numpy as np；c = np.array(a) + np.array(b)

反转

.reverse() 直接改变数组返回None

[::-1]

reversed()

函数内改变数组：

nums = new\_nums # nums是入参，实际上并没有被改变

nums[:] = new\_nums # nums被改变

while len(list1)>0 ----简写--> while list1  
 去重 l = list(set(l))

转字符串 ‘’.join(str\_l) #必须是字符串列表才能使用

元组

相等 python2的cmp()等价于==，python3用==

比较（用于排序、二分操作）数组一样支持

>>> (1,2) < (1,3)

True

>>> (1,2) < (0,3)

False

>>> (1,1)+(1,0)

(1, 1, 1, 0)

>>> [2]==2 元组与数组居然是可以做比较的

False

map

初始化

={}

不能=map()

遍历

for key in m

for key in m.keys()

for value in m.values()

for kv\_tuple in m.items()

for key,value in m.items()

for (key,value) in m.items()

获取元素 []（不存在会抛出异常） .get(num, 0)

批量更新元素 update({1:2})

set

是map的特化，但不支持索引运算符

初始化与元素 需要可枚举如数组、字符串,只能存储单参数，不能存储[]、()

set('123') #set初始化如果将字符串添加进去，会拆散为各个字符元素

set(1) 抛出int不可枚举的异常

添加与删除

{1, 2, 3} 但是 {}是dict # set([1,2,3])

s.add('123') #将字符串作为一个元素添加进去

.discard(指定元素) 而如果用 .remove 当元素不存在时不会抛异常

.pop: 说是随机弹出一个元素，不过存在一定的规律（只要底层没有真用random） 目前我用过的使用场景：如果集合仅存在一个元素，用pop弹出来

批量插入（并集）m.update({4, 5, 6})

集合操作：&交集 | 并集 – 差集 ^补集 # 都是返回一个新的集合

判断元素是否存在（同map）： in

数组转集合取交集，如果不使用函数 setA.intersection(setB)

class Solution:  
 def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:  
 set1 = set(nums1)  
 set2 = set(nums2)  
 return self.set\_intersection(set1, set2)  
  
 def set\_intersection(self, set1, set2):  
 if len(set1) > len(set2):  
 return self.set\_intersection(set2, set1)  
 return [x for x in set1 if x in set2]

字符串

虽然可以用索引来走读，但是不可以直接用索引进行元素修改。应先转list(‘123’)，此时类型为List[str]，修改完成后在通过’’.join(l)转换为str。

生成: 3\*’c’

遍历：

for item in s[::-1]

for item in list(s)

切分：

直接split，不要split空格。不然多个空格切不干净。如果split传入空字符串会报错

>>> ' a b c d '.split() # consecutive space、tab、\n

['a', 'b', 'c', 'd']

ASCII：ord(‘z’)即122 ，chr(122)即’z’

剥去两端 .strip(‘abc’) # 左右两侧a、b、c字符。lstrip、rstrip

前/后缀匹配: startswith、endswith

插入字符 '#'+'#'.join('123')+'#'比如回文串预处理

删除字符 '#1#2#3#'.replace('#', '') 第三个参数是count

自定义生成 '{}'.format(1)

sscanf从字符串中获取输入（常用于复杂字符串切割) ，但python不支持，请使用正则

[正则](https://www.runoob.com/python/python-reg-expressions.html)

如果匹配不到将会是None ，无法使用成员函数

compile: pattern = re.compile(r'\d+')

match从字符串的起始位置匹配

>>> re.match(r'1(2)(3)', '123').groups()

('2', '3')

>>> re.match(r'1(2)(3)', '123').group()

'123'

search扫描整个字符串并返回第一个成功的匹配

>>> re.search(r'1(2)(3)', '123').groups()

('2', '3')

>>> re.search(r'1(2)(3)', '123').group()

'123'

findall 返回元组数组

>>> re.findall(r'(\w+)=(\d+)', 'set width=20 and height=10')

[('width', '20'), ('height', '10')]

sub 替换字符串，因此不会报错

>>> re.sub(r'#.\*$', "", phone)

替换后结果

>>> re.search(r'1(2)(3)', '123').group()

'123'

split 切分，返回数组

>>> re.split('\W+', 'runoob, runoob, runoob.')

['runoob', 'runoob', 'runoob', '']

collections.Counter()

传入可枚举对象，统计次数，像是defaultdict(int)

添加元素：m[num] += 1；m[num] = 1 没有add函数, 自动初始化

m[num] += 0 即初始化元素

获取元素统计个数：m[num] <=> m.get(num, 0) 不需要初始化元素

删除元素：.remove .pop .clear清除所有

基本运算：+ 相加、- 相减（如果小于0则删去）、&保留各元素最小统计值、| 保留各元素最大统计值

子串确认：in（非常快，最长公共子串可以用此操作 而不是DP）

比较运算符：

> 每一个元素都存在并非负且都大于 或者 等于但有跟多的元素

= 具有相同的元素以及值

不可得出比较结果的情况下都是False

.items() .keys() .values() 返回的都是dict\_xxxs类型，不能使用下标subscriptable，但可以被迭代。不方便使用的话，可以通过迭代转list等类型。

collections.deque 双向队列

受GIL管理，线程安全

支持有界队列：dque = deque(maxlen=10)固定长度，当长度满了之后，当我们继续append时，它会自动弹出最早插入的数据。

增加.appendleft和.popleft（虽然基于list实现，但是popleft性能远好于list.pop(0)）

与列表一样有 .append、.extend、.pop 等，但不能使用[1:]这种slice操作

同list一样可以使用索引，实现peek能力（访问数据但不remove）

>>> dq = collections.deque([2]) # 可不传参，注意d小写

>>> dq.append(1)

>>> dq[0]

1

>>> dq.index(0) # 抛出异常 ValueError: 0 is not in deque

>>> dq.index(1) # 返回值为1的索引

0

collections.defaultdict

比map在索引符号上多了一个默认值避免抛出异常

>>> d\_m = collections.defaultdict(int) # int, set, str, list

collections.defaultdict(type([]))

>>> d\_m[‘1’] # 0

.setdefault(xxx) 并不是设定自定义默认值

def setdefault(self, k, d=None):  
 value = D.get(k, d)  
 if k not in D:  
 D[k] = d  
 return value

TrieNode = lambda: defaultdict(TrieNode) # Trie树的定义

queue.Queue 队列

基于collections.deque实现

没有peek、len函数，无法使用下标索引

>>> q = queue.Queue()

>>> q.**put**(1)

>>> q.empty()

False

>>> q.**get**()

1

>>> q.empty()

True

>>> q.full()

False

>>> q.**qsize**() 而不是len(q)

0

queue.PriorityQueue优先级队列（线程安全，基于heapq）

只能小顶堆。可以存单个数值，也可以存多个（多个值从左到右逐个比较排序，因此都需要实现\_\_lt\_\_方法），但类型必须一致（会抛出异常，但是存的进去get的出来，就是比较/排序失效了）。基于此实现大顶堆，解决思路同下面heapq介绍。

>>> pq = queue.PriorityQueue()

>>> pq.put((10, 'a'))

>>> pq.put((3, 'a'))

>>> pq.put((4, 'a'))

>>> pq.get()

(3, 'a')

>>> pq.get()

(4, 'a')

>>> pq.get()

(10, 'a')

>>> pq.qsize() # 不支持len()，必须.qsize()

0

queue.LifoQueue 先进后出队列

同堆栈

heapq堆/优先级队列

非线程安全，基于list

index=0即堆顶

只能实现小顶堆，heapq在实现的时候，没有给出一个类似Java的Compartor函数接口或比较函数，[开发者给出了原因](http://code.activestate.com/lists/python-list/162387/)，[解决思路](http://stackoverflow.com/questions/14189540/python-topn-max-heap-use-heapq-or-self-implement)，比如将push(e)改为push(-e)、pop(e)改为-pop(e)。

这三个函数一起用，逐步构造：

heapq.heappush(heap, item) #把item添加到heap中（heap是一个列表）

heapq.heappop(heap) #把堆顶[0]元素返回并删除

heapq.heappushpop(heap, item) #先push再pop，比分开调用快得多

heapq.heapify(x) #将列表x一步到位 调整成小顶堆，不需要逐步构造

heapq.merge(\*iterables) #将多个列表合并，并进行堆调整，返回的是合并后的列表的迭代器

heapq.nlargest(**n, iterable**, key=None) #返回最大的n个元素（Top-K问题），iterable不一定是heapq push产生的，可以是任何可枚举。**nlargest、nsmallest不需要先heapq.heapify 保持堆结构**。

heapq.nsmallest(**n, iterable**, key=None) #返回最小的n个元素（Top-K问题）

>>> import heapq

>>> heapq.heappush(h, (1,1))

>>> heapq.heappush(h, (2,2))

>>> heapq.heappush(h, (3,2))

>>> heapq.nsmallest(2, h) # 注意参数在第二位

[(1, 1), (2, 2)]

>>> t = heapq.nlargest(2, h) # 虽然实现是小顶堆，但支持这个函数

[(3, 2), (2, 2)] 如果数量不足只会返回一个，不会想heappop引起异常

>>> [x for x in t]

[(3, 2), (2, 2)]

>>> heapq.heappop(h)

(1, 1)

>>> a = [29, 87, 6, 88, 36, 93, 47, 1, 78, 44]

>>> heapq.heapify(a)

>>> a

[1, 29, 6, 78, 36, 93, 47, 88, 87, 44]

sortedcontainers.SortedList

这个库还有SortedDict/SortedSet

仅使用一个列表和bisect.insort会在长度超过一万时产生缓慢的行为，因为在插入与删除时会涉及列表部分元素的内存移动（即便已经使用mem-copy/mem-move-like）。本库设计成双层列表，使得每一个操作都是O(logN)。

SortedList(nums) 可以用可迭代对象直接初始化

.add 添加元素（而不是bisect\_insort、append）

.discard 不报错移除

.pop 可指定下标弹出，否则弹出末尾元素

.bisect\_left/.bisect\_right 需要保证已有输入与内部元素可用<或>进行比较

itertools

如果需要转list，直接list()

accumulate 适合用于acm算法中prefix sum题目加速求和

>>> **list**(itertools.accumulate([1,2,3]))

[1, 3, 6]

组合(无放回与有放回)

>>> for a,b in itertools.combinations([1,2,3], 2):

print(a,b) # a,b unpack 元组

>>>

1 2

1 3

2 3

>>>for a,b in itertools.combinations\_with\_replacement([1,2,3], 2):

print(a,b)

>>>

1 1

1 2

1 3

2 2

2 3

3 3

排列

>>> for i in itertools.permutations([1,2,3]):

print(i)

>>> (1, 2, 3)

(1, 3, 2)

(2, 1, 3)

(2, 3, 1)

(3, 1, 2)

(3, 2, 1)

笛卡尔积（有放回的排列, repeat指定重复次数，影响每一个元素数量）

>>> for a,b in itertools.product([1,2,3], repeat=2):

print(a,b)

>>>

1 1

1 2

1 3

2 1

2 2

2 3

3 1

3 2

bisect

第一个参数是有序非递减array，第二个参数是target

bisect\_left，bisect\_right<=>bisect 返回索引，返回值是“假如作为left、right插入时的index” ，范围0-len需判断数组越界。所以left找到相同元素最左index，right找到相同元素最右index+1（如果做了-1操作来找位置，注意范围是[-1,len-1]）。

insort\_left，insort\_right<=>insort 插入已排序列表，返回None

寻找<=x的最大值索引，思考下用bisect\_left对于0的处理

i = bisect\_right(a, x)   
if i==0:  
 return - 1  
return (i - 1)

寻找<x的最大值的索引

i = bisect\_left(a, x)   
if i==0:  
 return - 1  
return (i - 1)

上面思路都是找到边界点，然后-1

缓存函数装饰器

<http://www.zzvips.com/article/193416.html>

只能缓存有限的参数类型，例如基本数据类型、元组和不可变集合等hashable对象。对于可变对象（如列表和字典）和自定义类对象，需要自行实现缓存机制。

@functools.cache # 不限制缓存大小，非线程安全

@functools.lru\_cache(None) # None表示不限制缓存大小，非线程安全

在这道[图DFS算法题](https://leetcode.com/problems/out-of-boundary-paths)中，需要使用@cache或@lru\_cache(None) ，而不用或者@lru\_cache都会超时。

json

load/dump 对文件

loads/dumps 对字符串

如果解析失败会抛出异常

try:  
 open("test0001.txt")  
except:  
 print("出现异常")  
else:  
 print("没有异常")  
finally:  
 print("有没有异常都会执行")

Python没有自带goto，除非使用扩展。在循环中调用函数，在函数中使用break、continue是错误的。深入明白break、continue的原理。函数已经换了块地址空间了。

Python不支持do?while语法

Py没有switch

官方的解释说，“用if... elif... elif... else序列很容易来实现 switch / case 语句”。可以使用函数字典映射和类的调度方法。

使用match

match "3":  
 case "1" | "3": # 1或3  
 print(1)  
 case "2":  
 print(2)  
 case \_: # default  
 print("not matched")

# 一个(x, y)元组

match point:   
 case (0, 0):  
 print("Origin")  
 case (0, y):  
 print(f"Y={y}")  
 case (x, 0):  
 print(f"X={x}")  
 case (x, y):  
 print(f"X={x}, Y={y}")  
 case \_:  
 raise ValueError("Not a point")

# class

from dataclasses import dataclass  
  
@dataclass  
class Point:  
 x: int  
 y: int  
  
  
def whereis(point):  
 match point:  
 case Point(0, 0):  
 print("Origin")  
 case Point(0, y):  
 print(f"Y={y}")  
 case Point(x, 0):  
 print(f"X={x}")  
 case Point():  
 print("Somewhere else")  
 case \_:  
 print("Not a point")

9. 多个返回值的函数结果以1个变量接，这个变量会成为元组；否则只能以等同数量的个数来接。

12.5变量是否存在（经过声明，算法题中永远不要使用）：

1. var\_exists = 'var' in locals() or 'var' in globals()
2. 通过异常

try:

var

except NameError:

var\_exists = False

else:

var\_exists = True

13.

命令行进入python后的操作：（也就是进入交互模式）

提示符变为>>>，在这后面不能先有空格，因为indent识别要求你先是顶到的

与文本模式相比的特点：

1.文本前面不能使用空格，否则会报错。因为被当成缩进处理了。

2.复合语句中第二行开始提示符又一次改变，这次变为...

3.用一个空行来表示复合语句的结束

Use exit() or Ctrl-Z or quit() plus Return to exit（这是正常退出的姿势）

Python 有办法将任意值转为字符串:将它传入repr() 或str() 函数。函数str() 用于将值转化为适于人阅读的形式,而repr() 转化为供解释器读取的形式。

>>> repr('123')

"'123'"

>>> repr("123")

"'123'"

>>> repr(1.0/6.0)

'0.16666666666666666'

>>> s = 'Hello, world.'

>>> str(s)

'Hello, world.'

>>> repr(s)

"'Hello, world.'"

上面这个显示是为了表明里面是个字符串

>>> Fraction(5,4) is 1.25#不同的类型，当然不同的对象

False

>>>float(Fraction(5,4))

1.25

>>>type(float(Fraction(5,4)))#float对象

<class 'float'>

>>> type(Fraction(2,5))

<class 'fractions.Fraction'>

>>> type(float) #float也是个对象么

<class 'type'>

>>> type(1.0)

<class 'float'>

>>> from math import \*#这个是把内容导入本命名空间

>>> sqrt(-1)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: math domain error

>>>math.sqrt(16) #利用这种方式直接对库内的函数进行调用？不可以，必须要引入库。

4.0 得到的是float

>>> random.random()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'random' is not defined

>>> import random #导入命名空间

>>> random.random()#自动给一个取[0,1)的‘随机’float数

0.7779166377714883

>>> random.choice((1,2,3,4))#这是使用元组

3

>>> random.choice([1,2,3,4])#这是使用列表

1

>>> random.sample((1,2,3,4,5),2)#用列表形式输出。第一个参数可以是元组也可以是列表。

[5, 2]

>>>range(-2,-3,-1)

range(-2, -3, -1)

>>>print(range(-2,-3,-1))

range(-2, -3, -1)

>>>list(range(-2,-3,-1)) #直接看不到，但是想到可枚举所以做了这层转换

[-2]

随机打乱顺序：**shuffle洗牌**

>>>a=[1,2,3,4,5]

>>>random.shuffle(a)#在交互模式中直接random.shuffle([1,2,3,4,5])是没有效果的，因为这个函数返回void，没有对改变后的参数进行输出查看。他更改不会借助返回值，因为它传过去就是引用传过去。

>>>a

[3,4,5,2,1]

>>> print(random.shuffle([1,2,3,4,5]))#话说前面由于我是直接复制下来，所以>>>后面自带了一个空格，不是我自己打的

None

>>> math.modf(2.5)#返回的是列表形式，取出来的整数部分也是浮点数显示的，而且浮点数是在前面

(0.5, 2.0)

获取N位随机数（二进制位）：

>>>random.getrandbits(200) #输出的时候当然是十进制

14.基本数学函数：

内置函数

pow、round、max、min

pow 注意返回类型

>>> math.pow(1,2)

1.0

>>> pow(1,2)

1

>>> type(pow(1,2))

<class 'int'>

abs

round 2.x与3.x对round(2.5)结果不同，3.x里面是2等价于int(), 2.x里面是3（4舍5入）

>>> round(2.335)

2

>>> type(round(2.1))

<class 'int'> 仅当没有第二个参数时相当于int

>>> type(round(2.1, 0))

<class 'float'>

>>> round(2.335, 2)

2.33

int() int是向0取整 python除法结果默认浮点数

cmp 3.x里面舍弃了，直接用==

max与min

max= a if a>b else b

>>> type(min(0.1,2)) #注意类型转换的问题

<class 'float'>

>>> type(min(1,2))

<class 'int'>

math库函数

math.ceil（不论正负往数学上大了的走）、math.floor

math.fabs

math.log

>>> log(e) //我以为会是ln，不过看来还是和C语言的函数命名一样

1.0

>>> from math import \*

>>> log(2)

0.6309297535714574

>>> log(2,4)

0.5

>>> import math

>>> math.log(2, 256)

0.125

>>> math.log(256, 256)

1.0

>>> math.log(256, 2)

8.0

判断是否243是3的幂（实际为3\*\*5）

>>> math.log(243, 3)

4.999999999999999 因此不要用 int(x)==x

>>> math.log(243, 3).is\_integer()

False

math.inf无穷大 -math.inf负无穷

math.log10

math.modf返回浮点元组，小数部分在前，整数部分在后

math.sqrt

格式化输出函数：format，就是printf里面的格式说明符的修饰符。为了表明输出东东类型是字符串而在交互模式中加上引号。

>>> format(2.5, '0.0f') #0精度下的显示

'2'

>>> format(2.335,'0.2f')

'2.33'

>>> round(2.345,2)#第二个参数就是小数点后几位

2.35

>>> format(2.345,'0.2f')#也是偶数舍入，但是由于2.345转为2进制的时候，得到了一个略大的数字，所以最终再转换就是他了

'2.35'

>>>format(13,'b')#剩下的自然就是字符o和字符x代表，这里不能是大写

'1101'

>>>'{} {}'.format(1, '2')

'1 2'

>>> round(2.5555, 3)

2.555

>>> round(Decimal('2.5555'),3)#使用Decimal来控制好精度就正常了

Decimal('2.556')

>>> round(Decimal(2.5555),3)//果然还是使用字符串好一点。Decimal(2.5555) != Decimal('2.5555') 前者会选择有限长度转化为字符串再存进去

Decimal('2.555')

>>> round(2.55555, 3) //后面是55而不是单单一个5，所以进一

2.556

精度函数：Decimal（借助了字符串）

>>> from decimal import Decimal

>>> Decimal('2.1')+4.2

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'decimal.Decimal' and 'float'

>>> Decimal('2.1')+Decimal('4.2')

Decimal('6.3')

>>> Decimal(1.9)+Decimal(2.1)

Decimal('4.000000000000000000000000000')

>>> Decimal(1.9) == 1.9

True

>>> 1.9 == 1.9

True

>>> Decimal('1.9') == 1.9

False

>>> 2.1+4.3

6.4

>>> 2.1+4.2 #这个显示真的是很神奇

6.300000000000001

>>> 3.1+4.6

7.699999999999999

15.python里面的对象内存管理机制：

py变量里面没有类型。变量里面存的都是引用，也可以说是引用类型。变量是指向对象的，比如说a=3，那么3就是对象，对象具有类型。之后再a='pi',那么对象3就会被回收。

例子：

a=3

b=a

a+=2；创建了一个新的对象，地址给a。写时创建。

与C一样的是列表：

List1 =[1, 2, 3]

List2 =List1#如果要将列表复制过去的话就是List2 =List1[:]

List1[0] =24#List2还是只想这个对象

is用来判读左右两边是不是同一个对象。

>>> a=1

>>> b=1

>>> a is b（js里面是===）

True

>>> a =3.14 #浮点数都是会重新创建的

>>> b =3.14

>>> a is b

False

>>> a ='o'

>>> b ='o'

>>> a is b

True

查看一个对象被引用了多少次（被多少个指针指着）:

import sys

sys.getrefcount(1)#查看1这个对象被引用了多少次，包括了这个函数调用会引用3次，以及之前库函数引用过的次数

16.python里面的字符串：

单引号字符串

双引号字符串

三引号字符串

不带转义字符的字符串称为Raw字符串，可以使用r开头来让字符串里面不会进行转义处理。r"\new\my"或者R"\new\my"

python3.x中的Byte字符串（就是一个字节存储吧，3.x默认是Unicode）:b'123'。2.x默认使用Latin-1的字符串方式，所以2.x里面有u开头的Unicode字符串形式。所2以2.x里面输入输出中文是有问题的，再写脚本的时候要在第二行加上对utf-8格式的转换。

其实我可以当做没有字符的概念，全是字符串。

我第一次见到的转义字符有：

\uhhhh Unicode 16位的十六进制值

\Uhhhhhhhh Unicode 32位的十六进制值

\xhh

\ooo

\f的换页功能显现是与终端有关系的。我这里python与c的现实是一样的，都是一个特殊的可打印字符。

\v是光标往下移动一行，但是不会移动到最前面。但是在windows的终端里，它显示是个特殊字符

\0如果打印出来，在windows下是空格的效果。

不能依靠这些功能不确定的转义字符。

>>> a ='abc\b\b\b1' #\b只是会对光标位置进行退格，并不带有边退边删除

>>> print(a)

1bc

>>> a ='abc\r1' #\r回车就是回到这一行开头

>>> print(a) //\n却自带了\r的效果

1bc

>>> a= '\u0041'

>>> print(a)

A

>>> a ='\U00000041'

>>> print(a)

A

>>> a ='\x041'#\x04 1

>>> print(a)

1

>>> a ='\x41'

>>> print(a)

A

'abc'+'efg'

'abc' \* 5 #字符串也是个可迭代对象

>>> 'a''b' #我试了下str ='a''b'，确实等价于str ='ab'

'ab'

>>> 'T' in a#字符在不在字符串里面。虽然这个运算符名字叫成员运算符

True

>>> str ='this\_is\_a\_test\_consequence' #python里面如函数的字符串

>>> str[4]

'\_'

>>> str[5:7]#分片操作，和C++迭代器像，也就是不包括7

'is'

>>> len(str) #注意不是成员函数

26

>>> str[-1:25] #-1就是倒序数的第一个，只是表明位置，并没有表示出方向

'' #错误使用返回空串么

>>> str[-1:26]

'e'

>>> str[::2]#正向，每隔一个字符就会输出

'ti\_sats\_osqec'

>>> str[::-1]

'ecneuqesnoc\_tset\_a\_si\_siht'

>>> str[0:3:1]#第三个不是表示方向

'thi'

>>> str[-1:-3:-1]#输出都是不包括第二个参数的 #str[-1:-3]是错误的，得到空串

'ec'

>>> ord('a')#对于3.x参数还可以是中文

97

>>> chr(97)

'a'

>>> format(8,'b')

'1000'

>>> format(8,'o')

'10'

>>> format(8,'d')

'8'

>>> format(8,'14')

' 8'

>>> str[0]

't'

>>> str[0]='a'#字符串是不可修改的对象，和java一样。[]函数返回的是常量吧。

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

#这时候只好重新创建一个变量

>>> str ='T'+str[1:]

>>> str

'This\_is\_a\_test\_consequence'

#也可以使用replace成员

str.replace('被替换的内容', '用于替换的内容')#不同于C++里面的replace，C++是要给出替换坐标。这里的replace是程序自己根据内容来寻找位置。而且不是替换，只是返回一个新的字符串。

大小写相关成员: #因为字符串是不可以改变的，所以函数是创建一个新的，而不会对参数造成任何改变。

capitalize()#除了句首字母大写其余都小写

title()#每一个词的头字母大写，一些数字与符号在中间却被当做两个单词

upper()

lower()

swapcase()#大小写交换

排版：

size就是设置给的板块的长度,fill是填充字符（填充多余部分），默认是空格：

center(size[, fill]) ljust(size[, fill]) rjust(size[, fill])

>>> str.center(20) #会将字符串扩展成为20字节

' qwe 12 '

转换\t为空格，默认8个：

expandtabs([tabsize])

>>> print("he\tllo！".expandtabs())

he llo！

扩展字符串到指定长度，同时使用0来填充前面多余的部分，适合数字字符串的输出。

zfill(size)

删除两边/左边/右边的指定字符，默认空格和换行符,chars就是字符们其实类型就是字符串:

strip([chars]) lstrip([chars]) rstrip([chars]) #strip剥光的意思

>>> str.strip(str =str.center(123)) #不能这么写了这不是表达式，而是参数指定

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: strip() takes no keyword arguments

应该是a.cener(123).strip()

字符串查找统计相关：

第二个参数与第三个参数是位置限定：

startswith(prefix[, start[, end]]) endswith(suffix[, start[, end]]) #函数中有一个s

>>> url="http://baidu.com"

>>> url.startswith('https://','http://',"ftp://")#不好的方法就是借助字符串分片与==

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: slice indices must be integers or None or have an \_\_index\_\_ method

>>> url.startswith(('https://','http://',"ftp://"))

True

匹配子串计数：

count(sub[, start[, end]])

返回方向找到的第一个字符串的位置。找不到返回-1：

find(sub[, start [, end]]) rfind(sub[, start [, end]]) #这也是返回索引

返回方向上找到的第一个字符串的位置。找不到返回-1

index(sub[, start[, end]]) #java里面是indexOf

replace(old, new[, count])

替换，默认没有count就是全部替换

格式判断成员：

isalpha()

isdigit()

isupper()#这是对字母而言的，如果字符串里面没有字母就返回False。这两函数有点特殊，不用全部都是字母。

islower() #有空格和数字也还是对的额

isspace()

#以下是ctype.h里面没有的

istitle() //和title()对应

isprintable() #名字不一样了，这里的名字才是不应该

>>> str ='123^Z' #md，神了

>>> str

'123\x1a'

字符分隔：

sep是分隔标志字符串，maxsplit是指做多允许分隔几次 #py里面都是按照字符串考虑的，没见到按照字符的

split([sep[, maxsplit]) rsplit([sep[, maxsplit]) #注意有反向函数。split系列，如果实在一端被匹配，那么最后显示就会这端有''多出来的列表

>>> "abcd".split("bc",1) #注意分隔符不会被保留

['a', 'd']

>>> 'abcbcd'.split('bc')

['a', '', 'd'] #这个问题要注意

#进化版的split用于去掉\r,\n,\r\n。参数为True是=时，就不会处理那几个。

>>> "abc\nefg".splitlines()

['abc', 'efg']

>>> "abc\nefg".splitlines(True)

['abc\n', 'efg']

>>> "abc\nefg".splitlines(False)

['abc', 'efg']

#正则与split

>>> line ='abc defg; hijlk, lmn'

>>> import re #正则表达式库

>>> re.split(r'[;,\s]\s\*', line)#\s表示空格

['abc', 'defg', 'hijlk', 'lmn']

>>> re.split(r'[;,\s]\*', line)#我这种不好，因为\*可以表示0个。那么我的正则可以匹配‘没有字符’

E:\Program Files (x86)\Python\lib\re.py:203: FutureWarning: split() requires a non-empty pattern match.

return \_compile(pattern, flags).split(string, maxsplit)

['abc', 'defg', 'hijlk', 'lmn']

>>> re.split(r'[;,\s]+', line)

['abc', 'defg', 'hijlk', 'lmn']

使用Shell通配符去匹配字符串:

>>> from fnmatch import fnmatch,fnmatchcase#从fnmatch模块引入fnmatch和fnmatchcase这两个函数，所以说从模块中import \*就是将变量全都装到当前名字空间里面去

>>> fnmatch('foo.txt','\*.txt')#这个不区分大小小，而加了case那个区分

bTrue

>>> fnmatch('foo.TXT','\*.txt')

True

#间隔符不会被去除，也加入元组单独作为一个元素，据说对于URL有用。

partition(chars) #Partion是n.分区 ，这个是v.n.分割

>>> str.partition('1')

('', '1', '23asdfqerajklg') #1是首字符于str字符串

>>> "abvc".join('123') #需要注意是把前者插入到后者每一个‘间隔’

'1abvc2abvc3'

>>> map ={97:'A'}

>>> 'a'.translate(map)

'A'

>>> 'a'.translate(map) #这是map里面没有这个元素的时候这个函数返回的

'a'

用正则进行可变字符串匹配：

>>> time ='1970/01/01'

>>> re.match(r'\d+/\d+/\d+',time) #在正则表达式的时候使用raw字符串，shifen有利于可读性

<\_sre.SRE\_Match object; span=(0, 10), match='1970/01/01'>

如果re.match没有匹配到输入的字符串的开头部分（他这匹配是只要把正则要求的匹配完了就算满足，可以使用{n}来验证），就没有返回

>>> time ='The first day is 01/01/1970'

>>> re.sub(r'(\d+)/(\d+)/(\d+)', r'\3-\2-\1', time)#正则里面圆括号括起来表示单元，然后\3 \2 \1就是单元号，调用他们就是调用对应单元匹配住的。注意，单元号从1开始。这个函数不同于match，而会从开头检测到末尾，看看有没有满足的。有的话就会替换匹配部分

'The first day is 1970-01-01'

>>> datamode =re.compile(r'(\d+)/(\d+)/(\d+)') #创造了匹配模式

>>> type(datamode)

<class '\_sre.SRE\_Pattern'>

>>> datamode.sub(r'\3-\2-\1', time) #一定起码要字符串一开始匹配上，否则失败

>>> datamode.subn(r'\3-\2-\1', time, num)#返回一个元组，同时最后一个元素是匹配上的次数

datamode。findall(...) 不过需要注意的是正则表达式中如果没找到，那么就会抛出异常，需要进行异常捕获。

datamode.split(...) 反正就是把第一个参数给记录下来了

>>> re.findall(r'\d', '123') #但是不存在find

>>> ['1', '2', '3']

有一件事情请注意()这两个符号也是要转义的

17.

优先级：

\*\*最高 乘方

~ + - 一元运算符啊I是吧=

\* / % //

+ -

>> <<

& //和C不同的在于这个和比较运算符之间的优先关系

^ |

<= >= > <

== !=

=与复合赋值

is,is not

in, not in

and,or,not （和符号的优先级不一样！！！）

>>> True is 1==1

False

>>> True is (1==1)

True

~0是-1，因为是有符号整数

18.

print("abc")

print(2\*\*100) 就会打印2^100。

>>> a ='abc'

>>> for c in a:

... print(c, end='-')

...

a-b-c >>>

19.python里面的if-else这种是按照格式缩进来对应，而不是像C一样按照就近原则。

20.也是使用反斜杠将一行语句改写成多行语句。是的！已验证。交互模式也是要满足的。

21.支持单引号与双引号的字符串，使用三引号（三个单引号或者三个双引号）表示段落（这个其实直到这个结束都是字符串）。

>>> '''

... a

... b

... c

... '''

'\na\nb\nc\n'#自然是要记录\n的。

22.使用#来注释

23.map变量

>>> map ={97:'A'}

>>> 'a'.translate(map) #还是C++里的map用索引函数的格式好看点。

'A'

24.if的使用

>>> if 1==1:

... 2==2

...

True

>>> if 1==1 :

... 2==2

... else:3==3

...

True

>>> if 1==1 :

... 2==2

... else:

... 3==3

...

True

24.

Python是一门动态语言，在命令行工具下运行时，本质上执行了下面的步骤：

当第一次执行到一段代码时，这段代码会被编译（如，作为一个模块加载，或者直接执行）。根据操作系统的不同，这一步生成后缀名是pyc或者pyo的二进制文件。

解释器读取二进制文件，并依次执行指令（opcodes）。

1 lexing： 词法分析，就是把一个句子分解成 token。大致来说，就是用str.split()可以实现的功能。

2 parsing：解析，就是把这些 token 组装成一个逻辑结构。

3 compiling：编译，把这个逻辑结构转化成一个或者多个code object （代码对象）

4 interpreting：解释，执行每个code object 代表的代码。

>>> def double(a):

return a\*2

>>> double

<function double at 0x00000169C5F7FF28>

>>> type (double)

<class 'function'>

>>> double.\_\_code\_\_ #找到double 函数对象的 code object

<code object double at 0x00000169C5F36AE0, file "<pyshell#58>", line 1>

>>> type(double.\_\_code\_\_)

<class 'code'>

25.

#!/usr/bin/python #如果这里少了!那么就会。。。。执行文件卡着。。

import sys

import logging

import subprocess #用来实现调用系统命令

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

from scapy.all import \* #kali里面2.7支持，但是py3没有这个库。

if len(sys.argv)!=2: #argc

print "Usage:xxxx"

print "xxxxx"

sys.exit()

interface=sys.argv[1] #输入的参数数据类型本来就是字符串呀

ip=subprocess.check\_output("ifconfig"+interface+'管道过滤命令', shell=True).strip() #使用系统命令。

#还有这种格式x=subprocess.check\_output(["echo", "Hello World!"],shell=True)

prefix =ip.split('.')[0]+'.'+ip.split('.')[1]+'.'+ip.split('.')[2]

for addr in range(1,254):

answer =sr1(ARP(pdst=prefix+str(addr)), timeout=0.1, verbose=0 )

if answer =None:

pass

else:

print prefix+str(addr)

#True 仅仅首字母大写！！！

安装完scapy，简单写了个脚本执行总是会警告一下：

WARNING: No route found for IPv6 destination :: (no default route?)

原因是用 from scapy.all import \*的时候把ipv6相关的模块也导进去了，ipv6我们用不着，所以这警告不但不起作用，还很讨厌，我们打开scapy安装目录，打开all.py文件， 把与ipv6相关的几行注释掉就OK了，

#if conf.ipv6\_enabled:

# from utils6 import \*

# from route6 import \*

26.

import time

for addr in $(seq 1 254):

response=sr1(xxxx) #这是个异步的发包与收包

time.sleep(1)

try:

if response[TCP].flags==4

print prefix+str(addr)

except:

pass

time.time()返回的是一个微秒甚至后面还会有一位数的float。

25.僵尸扫描：

后来借助raw\_input输入被取消了，改用input()。

username =input("请输入账号:")如果直接给回车，那么username会被确定是字符类型，不过是个空串。

#!/usr/bin/python

import logging

logging.getLogger("scapy.runtime").setLevel(logging.ERROR)

from scapy.all import \*

def ipid(zombie): #测试是否满足僵尸机条件

reply1 =sr1(IP(dst=zombie)/TCP(flags="SA"), timeout=2, verbose=0) #终于知道了verbose是不显示详细信息，verbose是冗长的意思

send(IP(dst=zombie)/TCP(flags="SA"), verbose=0)//不等待收包

reply2 = sr1(IP(dst=zombie)/TCP(flags="SA"), timeout=2, verbose=0)

if reply2[IP].id == (reply1[IP].id+2): #ip层上面的id

print("This is a zombie!\n")

decision =raw\_input("Do you want to use this zombie to scan.Input y or not") #显示输出(自带换行来着)，并且获取输入给decision

if decision == "y"

target =raw\_input("What's the destation?");

zebieScan(target.zombie) #把target和zombie一起以字符串的形式闯过去

else

print("goodye!");

def zombieScan(target, zombie):

print "i will scan target"+target+" with zombie"+zombie

for port in range(0,1023):

try:

start\_val =sr1(IP(dst=zombie)/TCP(flags="SA"),timeout=2,verbose=0)

send(IP(dst=target,src=zombie)/TCP(flags"S",dport=port),verbose=0) #这里的ip伪造很关键但不一定成功。在同一子网或者通路不会对IP().src检查

end\_val =sr1(IP(dst=zombie)/TCP(flags="SA"),timeout=2,verbose=0)

if end\_val[IP].td == start\_val[IP].id+2 :

print port

except:

pass

print "--------------Zombie Scan Status---------------\n"

print "1 - Identity Zombie Host\n"

print "2 - Perform Zombie Scan\n"

answer =raw\_input("1 or 2?")

if answer == "1"

zombie =raw\_input("Input zombie IP")

ipid(zombie)

else

zombie =raw\_input("Input zombie IP")

target =raw\_input("Input target IP")

zombieScan(target, zombie)

26.import sys

sys.exit()

27.socket连接后，s.send('VRFY'+sys.argv[1]+'\r\n')再recv(...)。就当做在nc里面操作。

不过对于163.com，要先发HELO/EHLO，在nc中可以是HELO 163.com

28.elif 就是C里面的else if，不过和宏#elif一样。

29.列表：

列表里面存的是对象的引用。对象的类型可以不一样。

长度可变，可以为空（空列表显示成[]）

操作：

List1 + List2 按照顺序结合

.extend() 作用等同加号

List1 \*3 重复三次

可以使用“foreach”遍历列表for i in List1: print t

3 in List 判断列表里面有没有一个对象是对象3（in运算符）

List.index(1)返回1的索引

List.count(1)返回1的个数

List[0:3]

List2 =[c\*4 for c in 'SPAM'] 那么每一个字母都被重复4次即'SSSS',"PPPP"."AAAA","MMMM"

List(map(abs, [-2,1, 1,0])) Py2里面有没有List都是输出列表，Py3.x里面没有List输出地址。map将函数作用于每一个元素。

.append(xx)添加元素

.sort() 小到大排序。参数可以是key==str.lower这时候就认为全小写字母对字符串进行排序。reverse=True就会倒序。

.reverse()

.pop 和栈一样，所以是对末尾元素进行操作。允许带索引参数指定。

.push 使用.append()来实现的

.insert(index,xxx)

del List[0] #这个比较奇特

利用索引来进行分片的方式和字符串是一样的。特殊的有分组命名：

\_name=slice(0,2) 这时候 列表[\_name] 就会输出0到2的值。

[t1,t2]=['t1','t2']

info=['myNaome', 18, [1997,10,30] ]

\_name,\_age, (\_birth\_y,\_birth\_m,\_birth\_d)=info #此种用法命名不一定要这样下划线开头

python 3.x支持一种通配符格式，要求就是只能出现一个

A=['Abby','123-123','456-797','@qq.com']

B=['Boli','123-123','@qq.com']

A\_name,\*A\_tel,A\_mail=A

重复元素计算：

from collections import Counter

>>>Counter(list)

Counter({8: 1, 1: 1, 2: 1, 4: 1, 7: 1}) 这种格式

>>> record =Counter(list) #这种表达式没有返回值么

>>> record

Counter({8: 1, 1: 1, 2: 1, 4: 1, 7: 1})

record.most\_common(2) 返回值最大的两个

record.update(列表) 对其中值的更新

增强赋值语句（Py里面的叫法）于列表中的不同

>>> L1 =L2 =[1,2]

>>> L1 +=[3,4]

>>> L1,L2

([1, 2, 3, 4], [1, 2, 3, 4]) #L1 is L2

>>> L1 =L1+[5]

>>> L1,L2 #发生了复制

([1, 2, 3, 4, 5], [1, 2, 3, 4])

30.

import socket的话

要用socket.AF\_INET

因为AF\_INET这个值在socket的名称空间下

from socket import\*

是把socket下的所有名字引入当前名称空间（污染问题）

不管是哪个，删除都用del Module。

31.命名惯例：

·以单一下划线开头的变量名（\_X）不会被 from module import \*语句导入

·前后有下划线的变量名（\_X\_）是系统定义的变量名，对解释器有特殊意义

·以双下划线开头，但结尾没有双下划线的变量名（\_\_X）是类的本地（“压缩”）变量

·通过交互模式运行时，只有单个下划线的变量名（\_）会保存最后表达式的结果

变量没有以$开头的

32.程序模块正在运行的时候\_\_name\_\_='\_\_main\_\_'，但是被import的时候并不是（这时候是无后缀的文件名）。所以这句话的作用就是Make a script both importable and executable。

34.java，js里面true和false是全小写的。C、python里面是首字母大写。

35.

>>> a =True

>>> b =False

>>> (a is b)== False

True

>>> a is (b== False)

True

>>> a is b== False

False

>>> False is False == False #is 和==一起用而不使用()就会有奇葩情况，不论同那种优先级都不对

True

>>> False is False == True

False

36.eval()是把字符串中的东西计算出来

>>> eval('3+33+4')

40

37.

>>> print("%d %d",a,b) #需要时双引号

%d %d False 2

print("xx", end='-') #这时候就不会自动换行

38.map函数

>>> def add100(x):

... return x+100

...

>>> hh = [11,22,33]

>>> map(add100,hh)

[111, 122, 133]

>>> def abc(a, b, c):

... return a\*10000 + b\*100 + c

...

>>> list1 = [11,22,33]

>>> list2 = [44,55,66]

>>> list3 = [77,88,99]

>>> map(abc,list1,list2,list3)

[114477, 225588, 336699]

>>> list1 = [11,22,33]

>>> map(None,list1)

[11, 22, 33]

>>> list1 = [11,22,33]

>>> list2 = [44,55,66]

>>> list3 = [77,88,99]

>>> map(None,list1,list2,list3)

[(11, 44, 77), (22, 55, 88), (33, 66, 99)]

37.5 函数装饰器











python的装饰器就相当于是prehook，有的人可能会将参数验证这种必要流程通过这种写法来写，但是我觉得参数验证是一个几乎完全通用的必要流程，可以使用http框架的函数琏这种中间件写法。我想到的场景就是类似于hack或者维护代码这种在原有基础上变化才会使用。

框架使用这种方式来注册：



38.

#!/usr/bin/python 直接告诉计算机执行程序的具体位置

#!/usr/bin/env python 告诉计算机执行程序在系统环境变量中的名字，详细位置在环境变量中设

39.kali里面，python2.7和python3都是装了的，如果默认python是2.7版本，python3进去才是python3。

40.httplib实现了HTTP和HTTPS的客户端协议，一般不直接使用，在python更高层的封装模块中（urllib,urllib2）使用了它的http实现。

Get请求：

import urllib

import urllib2

url = "http://192.168.81.16/cgi-bin/python\_test/test.py?ServiceCode=aaaa"

req = urllib2.request(url) #制造Http Request对象

res\_data = urllib2.urlopen(req) #发出请求，并接受数据包

这就是返回的东东<addinfourl at 139780434794688 whose fp = <socket.\_fileobject object at 0x7f212b27ab50>>，更像是一个对象地址

res = res\_data.read() #这里使用的成员函数，解读收到的包

print res

#!/usr/bin/env python

import urllib2

import sys

type =sys.getfilesystemencoding() #其实是UTF-8

response =urllib2.urlopen('http://zhjw.scu.edu.cn/login.jsp', timeout=10)#url必须讲明协议 #设置超时的方法就是对socket的设置

print(response.read().decode("GBK").encode(type)) #gb2312编码估计也能显示正常（我是看浏览器浏览网页时的编码格式知道的）

它自己默认的头：

POST /loginAction.do HTTP/1.1

Accept-Encoding: identity

Content-Length: 29

Host: zhjw.scu.edu.cn

Content-Type: application/x-www-form-urlencoded

Connection: close

User-Agent: Python-urllib/2.7 这个！！！

#浏览器头方法一

headers = {'User-Agent':'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.6) Gecko/20091201 Firefox/3.5.6'}

req = urllib2.Request(url=url,headers=headers)

#浏览器方法二

request = urllib2.Request(uri)

request.add\_header('User-Agent', 'fake-client') #虽然是叫做add\_header，但是可以用来替换指定http头

User-Agent 有些 Server 或 Proxy 会检查该值，用来判断是否是浏览器发起的 Request

Content-Type 在使用 REST 接口时，Server 会检查该值，用来确定 HTTP Body 中的内容该怎样解析。

常见的取值有：

application/xml ：在 XML RPC，如 RESTful/SOAP 调用时使用

application/json ：在 JSON RPC 调用时使用

application/x-www-form-urlencoded ：浏览器提交 Web 表单时使用

#设置超时，使用socket全局

import urllib2

import socket

socket.setdefaulttimeout(10) # 10 秒钟后超时

urllib2.socket.setdefaulttimeout(10) # 另一种方式

#urllib2使用代理

urllib2 默认会使用环境变量 http\_proxy 来设置 HTTP Proxy。如果想在程序中明确控制 Proxy，而不受环境变量的影响，可以使用下面的方式

import urllib2

enable\_proxy = True

proxy\_handler = urllib2.ProxyHandler({"http" : 'http://some-proxy.com:8080'})

null\_proxy\_handler = urllib2.ProxyHandler({})

if enable\_proxy:

opener = urllib2.build\_opener(proxy\_handler)

else:

opener = urllib2.build\_opener(null\_proxy\_handler)

urllib2.install\_opener(opener)

这里要注意的一个细节，使用 urllib2.install\_opener() 会设置 urllib2 的全局 opener。这样后面的使用会很方便，但不能做更细粒度的控制，比如想在程序中使用两个不同的 Proxy 设置等。比较好的做法是不使用 install\_opener 去更改全局的设置，而只是直接调用 opener 的 open 方法代替全局的 urlopen 方法。

除了设置HTTPCookieProcessor，还可以设置Http头

opener.addheaders = [('User-agent', 'Opera/9.23'),('Connection', 'keep-alive'), ('Accept', accept)]

#重定向

urllib2 默认情况下会针对 3xx HTTP 返回码自动进行 Redirect 动作，无需人工配置。要检测是否发生了 Redirect 动作，只要检查一下 Response 的 URL 和 Request 的 URL 是否一致就可以了。

import urllib2

response = urllib2.urlopen('http://www.google.cn')

redirected = response.geturl() == 'http://www.google.cn'

如果不想自动 Redirect，除了使用更低层次的 httplib 库之外，还可以使用自定义的 HTTPRedirectHandler 类。

import urllib2

class RedirectHandler(urllib2.HTTPRedirectHandler):

def http\_error\_301(self, req, fp, code, msg, headers): #重新定义处理方法

pass

def http\_error\_302(self, req, fp, code, msg, headers):

pass

opener = urllib2.build\_opener(RedirectHandler)

opener.open('http://www.google.cn')

#cookie

urllib2对cookie默认无视不会在之后带上

import urllib2

import cookielib

cookie = cookielib.CookieJar()

opener = urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookie))

response = opener.open('http://www.google.com')

for item in cookie:

if item.name == 'some\_cookie\_item\_name':

print item.value

import httplib #不一定写到文件开头，windows下环境发现没有库

url = "http://192.168.81.16/cgi-bin/python\_test/test.py?ServiceCode=aaaa"

conn = httplib.HTTPConnection("192.168.81.16", 80)#构建conn对象，设置目标ip和port，算是对tcp的设置

conn.request(method="GET",url=url) #继续设置

response = conn.getresponse() #这里是提交并接受返回包

res= response.read()

print res

httplib.HTTPConnection(host[,port[,strict[,timeout]]])

这个是构造函数，表示一次与服务器之间的交互，即请求/响应

host 标识服务器主机(服务器IP或域名)

port 默认值是80

strict 模式是False，表示无法解析服务器返回的状态行时，是否抛出BadStatusLine异常

HTTPConnection.request(method,url[,body[,header]])函数

POST请求：

使用post方式时，数据放在data或者body中，不能放在url中，放在url中将被忽略。

import urllib #放在data中

import urllib2

test\_data = {'ServiceCode':'aaaa','b':'bbbbb'}

test\_data\_urlencode = urllib.urlencode(test\_data) #包内的数据要url编码

requrl = "http://192.168.81.16/cgi-bin/python\_test/test.py"

req = urllib2.Request(url = requrl,data =test\_data\_urlencode)#设置Request的data，http请求中的body #类首字母大写

print req

res\_data = urllib2.urlopen(req)

res = res\_data.read()

print res

import urllib　＃放在body中

import httplib

test\_data = {'ServiceCode':'aaaa','b':'bbbbb'}

test\_data\_urlencode = urllib.urlencode(test\_data) #表单填充数据使用了urllib

requrl = "http://192.168.81.16/cgi-bin/python\_test/test.py"

headerdata = {"Host":"192.168.81.16"}

conn = httplib.HTTPConnection("192.168.81.16")

conn.request(method="POST",url=requrl,body=test\_data\_urlencode,headers = headerdata)

response = conn.getresponse()#发出请求

res= response.read()

print res

#!/usr/bin/env python

import urllib

import urllib2

import sys

import cookielib

#systype =sys.getfilesystemencoding()

import time

user\_agent='Mozilla/5.0 (X11; Linux x86\_64; rv:45.0) Gecko/20100101 Firefox/45.0'

cookie =cookielib.CookieJar() #<CookieJar[<Cookie JSESSIONID=dbcEMhC3R0X2r7HGrvfPv for zhjw.scu.edu.cn/>]> 用foreack访问

opener =urllib2.build\_opener(urllib2.HTTPCookieProcessor(cookie))

urllib2.install\_opener(opener)

login\_formdata =urllib.urlencode({'zjh':'2015141462109','mm':'jjlijjli'})

login\_url ='http://zhjw.scu.edu.cn/loginAction.do'

login\_request =urllib2.Request(data=login\_formdata, url=login\_url)

login\_request.add\_header('User-Agent', user\_agent)

login\_response =urllib2.urlopen(login\_request, timeout=10)

#print(response.read().decode("GBK").encode("UTF-8")) #utf-8

evaluate\_url ='http://zhjw.scu.edu.cn/xkAction.do?actionType=6'

evaluate\_request =urllib2.Request(url=evaluate\_url)

evaluate\_request.add\_header('Cookie', cookie) #或许是因为是py2的原因，但是cookie是CookieJar对象，里面需要使用foreach提取

evaluate\_request.add\_header('User-Agent', user\_agent)

p\_time =time.time() #返回Unix时间戳浮点数，单位s，精确到ms

count =0

while time.time()-p\_time<60: #7packages/s 60s=330

try:

urllib2.urlopen(evaluate\_url, timeout=1)

count +=1

print(count)

except:

pass #timeout=0的话不停触发异常，每发一个包触发一次。C-D也出不来，这本会触发异常并结束程序，可能是因为一直占用CPU。只能从外部kill了。

print(urllib2.urlopen(evaluate\_url).read().decode('GBK').encode('UTF-8'))

这个程序一般占用两个端口（这个在建立连接那个在结束），最多时候占用三个。

#urllib2对PUT和DELETE的支持

urllib2 只支持 HTTP 的 GET 和 POST 方法，如果要使用 HTTP PUT 和 DELETE，只能使用比较低层的 httplib 库。虽然如此，我们还是能通过下面的方式，使 urllib2 能够发出 HTTP PUT 或 DELETE 的包：

import urllib2

request = urllib2.Request(uri, data=data)

request.get\_method = lambda: 'PUT' # or 'DELETE'

response = urllib2.urlopen(request)

#!/usr/bin/python

import urllib2

url = 'http://wwwww.hhhhh.net/'

response = None

try:

response = urllib2.urlopen(url,timeout=5)

except urllib2.URLError as e: #as可以使用逗号代替 #URLError和HTTPError都有.code和.reason

if hasattr(e, 'code'): #判断一个对象里面是否有name属性或者name方法,返回BOOL值

print 'Error code:',e.code

elif hasattr(e, 'reason'):

print 'Reason:',e.reason

finally:

if response:

response.close()

#Reason: [Errno -2] Name or service not known

#Debug log

使用 urllib2 时，可以通过下面的方法把 Debug Log 打开，这样收发包的内容就会在屏幕上打印出来，方便我们调试，在一定程度上可以省去抓包的工作。

import urllib2

httpHandler = urllib2.HTTPHandler(debuglevel=1)

httpsHandler = urllib2.HTTPSHandler(debuglevel=1)

opener = urllib2.build\_opener(httpHandler, httpsHandler) #这里可以多个合在一起当参数

urllib2.install\_opener(opener)

43.查找最大值和最小值

如果只要知道最大的一个或者最小的一个，使用不需要模块的max()、min()

补充：sum() 可以自动进行求和

但是如果要最大最小的几个数的时候：

from heapq import nlargest,nsmallest #我猜测这两个函数是对列表使用，所以也是可以对deque使用的。

nlargest(2,num) #num是列表，会返回最大的两个数组成的列表，大的数在前面

nsmallest同理

44.字典类型：

无序集合对象类型（因为是无序的所以不存在索引，也就没有分片）。

每一个值都有对应的键（所以需要不可变类型，基于此可以嵌套），用键来取值

字典的key是不可变的，值可变又可嵌套

使用花括号表示，可以为空。

dict ={'a':97, 'A':65, 2:2} #看出来可以异构也就是不同类型放在一起

dict['A']

.keys() 返回key们。最好加一个list()确保返回结果在不同版本py中都返回列表。

del dict['a'] 如果不存在就会直接报错！所以有了.get

.get('a',0)如果没有'a'就会给出第二个参数。否则没有的话就是不报错的不理会

.values() 同.keys

.items()用元组的形式来表示一个个对。最好用list()括起来，以下可以都这么做。

dict['a']=a 和map一样

>>> dict3={}

>>> dict3.fromkeys(['egg','bake'],0) 指定keys进行批量赋值

{'bake': 0, 'egg': 0}

dict=(['a',2],['b','b'])就是把上面的每个key值不同拆开了

如果类keys型却都一样，并且可以比较的话那就可以使用max(),min()。

如果可以相加就可以使用sum()

上面三个函数默认都是对keys操作，要想按照values操作，需要使用zip()

>>> max(zip(dict.values()))

(123,)

>>> max(zip(dict.values(),dict.keys()))

(123, 'e')

>>> list(max(zip(dict.values(),dict.keys())))

[123, 'e']

借助lambda函数，比较值显示出键的顺序：此函数对于外部变量是引用，即便是简单类型

max(dict, key=lambda k:dict[k]) #中间的key不能改成value 字典列表：

rows =[

{'name':'Brain', 'uid':1110, age:21},

{'name':'Abort', 'uid':11210, age:23},

{'name':'Coar', 'uid':11310, age:24},

{'name':'Penny', 'uid':1410, age:21}

]

from operator import itemgetter

row\_by\_name =sorted(rows, key=itemgetter('name')) #记住语句后面没有分号 #itemgetter的参数可以多个，避免第一个参数一样的时候

#大致意义就是获取name的值然后参与排序

from itertools import groupby

def height\_class(h):

if h>180:

return 'tall'

elif h<160:

return 'short'

else:

return 'middle'

friends = [191, 158, 159, 165, 170, 177, 181, 182, 190]

friends = sorted(friends,key = height\_class)

for m,n in groupby(friends,key = height\_class):

print m #m得到的值就是key调用函数返回的值

print list(n) #list显示很方便呀

结果：

middle

[165, 170, 177]

short

[158, 159]

tall

[191, 181, 182, 190]

#将key函数作用于原循环器的各个元素。根据key函数结果，将拥有相同函数结果的元素分到一个新的循环器。每个新的循环器以函数返回结果为标签。

排序sort：

#用sorted函数的key= 参数排序：

# 按照key进行排序

sorted(dict1.items(), key=lambda d: d[0])

# 按照value进行排序

sorted(dict1.items(), key=lambda d: d[1])

45.元组tuple

使用圆括号表示，任意对象的有序引用组合

只能通过偏移量取数据不能存（因为不可变）

只有一个元素的时候必须要加上逗号，因为不然会将括号理解成为表达式的那种括号作用。

T=1, #创建的时候有没有括号都行

如果一开始不需要元素或者只要有一个，可以使用t=tuple()，t=tuple(元素)。tuple这个函数的参数还可以是list。

只要不涉及更改的列表操作都可以用于元组，包括+ \* len((1,2,3))

.index(xx) 寻找元素

.count(xx) 计数元素

索引分片会产生一个新的元组

>>> t1,t2='t1','t2'

>>> t1,t2

('t1', 't2')

>>> (t3,t4)=('t3','t4')

>>> t3,t4

('t3', 't4')

>>> (t3,t4)

('t3', 't4')

>>> t1,t2,t3,t4='text'

>>> t1

't'

>>> t1,t2,t3,t4

('t', 'e', 'x', 't')

>>>t1,\*t2 ='text' #\*是通配符，t2会是个列表

PS：注意sorted返回的是列表

补充：序列赋值要求前后元素个数可以对应即可，类型组织形式不是那么重要

a,b,c ='text'[0],'text'[1],'text'[2:4]

a,b,c =list('text'[:2]),'text'[2:]

(a,b,c) ={1,2,3}

myList =[(1,2,3),(4,5,6)]

for a,b,c in myList

print(a,b,c)

a,\*b,c =1,2,3,4 #通配符后面可以再跟变量

>>> L=[1,2,3,4]

>>> while L:

... T,L=L[0],L[1:];print(T,L) #序列赋值

...

1 [2, 3, 4]

2 [3, 4]

3 [4]

4 []

>>> L =(1,2,3,4)

>>> \*a, =L #使用元组的特殊逗号定义

>>> a

[1, 2, 3, 4]

([])即()

([],1)即([],1)元组里面的列表还是可变的，估计是因为元组里面存的只会列表的指向结构。

46.

>>> dict

<class 'dict'>

47.python里面函数里面的key使用都是key=，用于指定接受函数。默认是None类型。

48.文件读写：

open(file\_address,[,access\_mode]) #可以使用raw字符串 #默认是r模式。文件模式的设置与C一样

写入文件是先写入缓存所以不会直接显示到文件里面

.close()

.tell()告知文件里的指针位置

.seek() 文件指针移动。参数使用举例(6)

.read() #默认全部读取，可以加上表示读取字节数目的参数

.readline()

.readlines()把每一行放到一个列表里面

UnicodeDecodeError: ‘gbk’ codec can’t decode byte 0x9d in position 1270: illegal multibyte sequence

知道问题所在，还是没有解决，又苦苦搜索，终于在 stackoverflow 上找到灵感，可以把 open 的方式变为 二进制，也就是下面代码里的 open(filename,’rb’)， 这下好了，至少后面的read() 可以通过。

在py2.x里面它只有一种存储方式：二进制方式，不论你是以文本方式还是二进制方式打开。py3中多了Unicode文本方式存储，对于文本模式就要这么用一般的字符存储，二进制模式打开文件就要使用.encode()返回一个二进制后存储。.encode是字符串的方法。

向文本存储列表、字典，在py中需要进行字符串转化：ofile.write(str({'a':97}) )就会把{'a':97}存入文件，读取的时候就又需要自己进行转换。比较 方便的做法是借助pickle模块：

import pickle

ofile =open('/tmp/a', 'wb')

a ={'a':97}

pickle.dump(a, ofile) #存进去的东西直接查看是一群有点怪的字符串

ofile.close()

ifile =open('/tmp/a', 'rb')

\_a =pickle.load(ifile)

\_a #得到{97, 'a'}

>>> str ={'a', 97}

>>> str

{97, 'a'}

>>> str ={'a':97}

>>> str

{'a': 97}

操作系统模块对文件的操作

import os

os.rename('xx', 'xxx')

os.remove('xx')

os.getcwd() 返回当前所在目录一开始的值来自终端，后来就靠os.chdir('xxx')

os.chdir('xxx')改变所处的文件夹，不影响终端

os.mkdir('xxx')

os.rmdir('xxx')

49.set、frozenset都是集合，所以存的是不可变类型。

元素是唯一的，视频说一旦有一个新的键进来会替换掉原来的

集合是无序的，所以不存在用索引来获取固定位置的元素

用{'a',9}来表示

>>> set=set()

>>> set

set()

>>> set =set()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'set' object is not callable #已经把set当做变量了，所以推荐使用诸如\_set，s

len(\_set)

x in/not in \_set

.isdisjoint(other\_set) 结合之间没有交集

.issubset(other\_set) \_set是other\_set子集

.isupperset(other\_set) 是超集

.union(other\_set) 返回两者合集

.intersection(other\_set) 返回交集

.different(other\_set) 去除交集

.symmetric\_difference(other\_set) 并集减去交集的部分

.update(other\_set,...) 多重合集

.intersection\_update(other\_set,...) 多重交集

.difference\_update(other\_set,...) 保留不同元素

.symmetric\_difference\_update(other\_set,...) 多重。。。

.add(element)不是push。也可以add一个集合，集合就是元素的构成。

.remove(element) <=>.discard(element)

.pop(element) 任意提取元素

.clear(element)

frozenset是不可变集合，没有add和clear成员

50.bytearray是序列，仅存在于py3中。提供可变字符串类型

>>> a =bytearray('你好', encoding ='utf-8') #不能识别Unicode和unicode，看来支持的就是utf-8

>>> a

bytearray(b'\xe4\xbd\xa0\xe5\xa5\xbd') #可以对字节用索引进行更改

>>> a[0:3]

bytearray(b'\xe4\xbd\xa0')

>>> a.decode()

'你好'

>>> a.find('好'.encode('utf-8'))

>>> a =bytearray(b'你妹')

File "<stdin>", line 1

SyntaxError: bytes can only contain ASCII literal characters.

>>> a =bytearray(b'123')

>>> a[0]

49

>>> a[0:3]

bytearray(b'123')

51.

a=[1,2,3]

b=['a',a,'b'] 引用a

b=['a',a[:],'b'] 复制a

c=[1,2,3]

c is a #False

>>> a='abc'

>>> b='abc'

>>> a is b

True

>>> a='a b' #因为空格，类似还有中文

>>> b ='a b'

>>> a is b

False

>>> a =1234 #大数字不行

>>> b= 1234

>>> a is b

False

>>> a = b= 1

>>> a is b

True

51.None

>>> type(None)

<class 'NoneType'>

在bool上None表示假，就是空对象。比如bool(), bool([])

52.无限循环

>>>a ={'a':''}

>>>a['a']=a #或者a.append(a) #不要以为一个是过去的，一个是现在的，然后构不成循环a=1; a+=a;

>>>a

{'a':{...}}

>>>a['a']['a']['a']

{'a':{...}}

53.注意多个参数之间有空格：

>>> print(1,'a')

1 a

>>> print("\d", 1)

\d 1

补充：print("{:d}\*10 is {:d}".format(int(\_input),int(\_input)\*10))

print(“{index}”.format(index=2))

54.由于没有分号，所以是使用pass来表空语句

55.函数直接用def 函数名:，有没有返回值就看你是不是用return语句了。把没有发返回值的函数给一个变量进行赋值，这句话是不会报错。

56.python 类

class Student(object):

\_\_address = “”

count = 0 #静态数据类型

books = []

def \_\_init\_\_(self, name, age):

self.name = name #普通数据成员

self.age = age

self.\_\_address = "Shanghai" #双下划线开头的属性在运行时会被”混淆”（mangling） 属性名字变成了\_Student\_\_address，因为双下划线开头的是私有变量，子类不可访问。单下划线开头是protected变量。如果使用 from a\_module import \* 导入时，这部分变量和函数不会被导入。不过值得注意的是，如果使用 import a\_module 这样导入模块，仍然可以用 a\_module.\_some\_var 这样的形式访问到这样的对象。

pass

|  |  |
| --- | --- |
| 1 | **\_\_init\_\_ ( self [,args...] )** 构造函数 简单的调用方法: *obj = className(args)* |
| 2 | **\_\_del\_\_( self )** 析构方法, 删除一个对象 简单的调用方法 : *del obj* |
| 3 | **\_\_repr\_\_( self )** 转化为供解释器读取的形式 简单的调用方法 : *repr(obj)* |
| 4 | **\_\_str\_\_( self )** 用于将值转化为适于人阅读的形式 简单的调用方法 : *str(obj)* |
| 5 | **\_\_cmp\_\_ ( self, x )** 对象比较 简单的调用方法 : *cmp(obj, x)* |

定义\_\_add\_\_就是重载+

@classmethod修饰的方法class\_foo()需要通过cls参数传递当前类对象。@staticmethod修饰的方法定义与普通函数是一样的：

@staticmethod修饰的函数是静态函数

@classmethod 修饰符对应的函数不需要实例化，不需要 self 参数，但第一个参数需要是表示自身类的 cls 参数，可以来调用类的属性，类的方法，实例化对象等。因为不需要实例化，所以使用形式与静态函数一样。

self跟cls只是默认给出的参数名，没有什么特别的含义，甚至你可以改成任意的字符，只是约定俗成的用self来表示实例方法的参数，cls来表示非实例方法的参数，用以区分。

所有的成员函数第一个参数应该是self

类的成员默认不会被识别需要自己加上self.，比如在一个类方法中使用另外一个类方法

类的继承：

格式： class 派生类名(基类名)

允许继承自多个父类

在Python中，通过单下划线”\_”来实现模块级别的私有化，一般约定以单下划线”\_”开头的变量、函数为模块私有的，也就是说”from moduleName import \*”将不会引入以单下划线”\_”开头的变量、函数。

57.触发异常：raise<error type>

try:...

except:...

58.assert a<0,'information'

>>> assert a<0,print('information')

information #执行了打印命令

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AssertionError: None #但是没有错误提示

>>> assert a<0,'information'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AssertionError: information #有错误提示

>>> assert a<0,a=-1 #命令不能执行

File "<stdin>", line 1

assert a<0,a=-1

59.花括号可以让变量称为临时的变量。相似的在py里面有种资源自动管理语句，使得资源的存在只有一句命令的执行这么长：

with open(file) as f: #把open(file)函数的返回值给f，然后执行f.read()，最后自己关闭资源

f.read()

60.无论交互模式还是文本代码模式，py里面的第一条语句都要是顶到的。

61.子语句只有一条的时候可以将其直接写在:后面。而且视频说对于简单语句，可以使用,对其进行分隔然后写在同一行

62.

>>> dict ={'a':97,

... 'b':98}

63.input函数会先输出参数然后获取一个字符串（剔除末尾换行）

>>> input("请输入一个东东")

请输入一个东东1 2

'1 2'

64.

>>> t1=t2=1233333333333333333333333

>>> t1 is t2

True

65.变量命名规则同C。非关键字的比如说print都是可以被当成变量的名字，不过这会导致print不能使用。不过你可以使用赋值语句现将print存起来然后换回去。True 和 False也没有被放到关键字里面。不过Py2，里面print的语句关键字而不是函数。

66.函数调用也是表达式语句。没有返回值的表达式语句返回的是None。print函数的返回值就是None。None支持bool判断。

67.print([object,...][,sep=''][,end='\'][,file=sys.stdout][,flush=False])#666由于使用了无线参数，所以后面可以使用参数指定来进行指定传值，始终很好的实现。file是输出目标，py里面的默认打开流名字与C语言一样，这些都是个文件句柄。默认sep是空格，end是'n。

print(open('xxx').read()) 在标准输出流打印输出文件内容

print(file=open('xxx','w')) 至少要这样子才会写入文件

#可以直接对下面这些对象进行输出

>>> print(L)

[1, 2, 3, 4]

>>> print(dict)

{'b': 'b', 'a': 'a'}

>>> '{}'.format(1) #自动识别类型进行转化输出

'1'

>>> '{}'.format(1,2) #多余的并不会报错

'1'

标准输出流在sys库里面，import sys.stdout。你可以更改标准输出流，sys.stdout =xxxx，但是要先存起来之后要记得改回来。

补充：可以在py2中开启对py3 print的使用，from \_\_future\_\_ import print\_function

68.if语句 not表示反，没有感叹号符号

if ...:

...

elif ..:

...

else:

...

三元表达式： value1 if judge else value2 #这语句的顺序很符合外国人习惯

还可以[value2,value1][test] test为真返回value1，否则value2.

补充：

is，not is 称为身份运算符

in，not in 称为成员运算符 #注意字典默认查的是有没有这个键

对于and，返回遇到的第一个False的值或者当全为True时最后一个True。所以还是可以用于bool判断的。对于or返回遇到的第一个真值，或者当全为False返回最后一个False。使用bool()进行类型转换。

补充：

文本模式输出字符串huidaiyou'' not ""

缩进语句必须是对齐的，否则会报错

69.

>>> def fun():

... """this is the information of fun()

... by lpn""" #这里我是顶到了写的，输出到文本所以多个空格

... print('test')

...

>>> fun()

test

>>> help(fun)

Help on function fun in module \_\_main\_\_:

fun()

this is the information of fun()

by lpn

70.循环语句

while后面可以跟else:

for i in range(0,10,1) #0开始到9，每次加一。第一个元素默认0，第三个元素默认1。

list(range(0,20,3))

for i in range(len(list1))

for a,b,c in [(1,2,3),(4,5,6),(7,8,9)] #也可以使用一个元素，再用一个for来取内部元素

>>>list1 =[1,2,3,4]

>>>list1 =[i+10 for i in list1] #z这里居然可以放一个循环，先创建一个列表然后进行引用赋值 #[line.strip().upper() for line in file\_lines if xxx]也就是使用函数（if xxx加上去才是完整格式）

list1 =[x+y for xxxx for yyyy]

>>> list2 =[range(2)]

>>> list2

[range(0, 2)]

使用for的时候，其实会调用iter()来获取迭代器。

71.exit()是全局函数，用来退出python。不需导入系统库。

72.yield：

>>> L =[1,2,3,4]

>>> for i in L:

... yield i #可以i \*\* 2这样来个运算 #这也使用return

...

File "<stdin>", line 2

SyntaxError: 'yield' outside function

#所以下面是正确使用姿势

>>> def getyield(combine):

... for i in combine:

... yield i

...

>>> L

[1, 2, 3, 4]

>>> list\_yield =getyield(L)

>>> list\_yield

<generator object getyield at 0x01E7BE10>

>>> for i in list\_yield: #这个循环每次都会把元素抽出来，所以之后list\_yield就是空。所以可以循环的是可迭代对象，要从中提取元素。

... i

...

1

2

3

4

>>> list\_yield[1]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'generator' object is not subscriptable #不支持下标

73.<file>.\_\_next\_\_() 余readline()类似，但是读取完成以后会报错。\_\_next\_\_()会报StopIteration，用来表示何时离开。

for line in open('xxx'): #这种用法，到了文件尾就会触发StopIteration以表示该退出了

print(line,xxx)

为了支持手动迭代，py里有个next()。next(x) <=> x.\_\_next\_\_()。一旦没了，你还继续next()就要触发了哦。迭代器一个元素只能读取一次，读完就把这个元素丢了。文件对象就是自己的迭代器，其余类型不是。

file =open('xxx')

file is iter(file1)

得到返回True

for i in iter(xxx)也倒是一种写法

注意：不要声明iter这个变量为名字，已经有这个函数了

>>> dict ={'a':1,'b':2,'c':3}

>>> dict

{'b': 2, 'a': 1, 'c': 3} #所以使用迭代器会先取出b，再a，再c

>>> dict ={'a':1,'b':2,'c':3}

>>> for key in dict.keys:

但是可以使用：

next(迭代器)或者迭代器.\_\_next\_\_()

map也是种迭代

list(map(str.upper,open('test.txt'))将后面可迭代的每一个值都当做前面的参数传入

sorted函数是先对迭代器进行排序，然后生成列表。

enumerate先对迭代对象进行运算后在生成可迭代对象。用于在原有的顺序中添加序号。需要使用list()包起来。

74.文档：

# 注释

dir函数，返回可用所有属性列表的方式，可以调用任何字符串。 dir('help')可以查看当前可以使用的函数。dir(1)<=>dir(int)。但是这个只可以看到可以使用的名字，不能看到使用方法。需要查看使用方法，要靠.\_\_doc\_\_。也可以help(库名字)，后者更清晰，两者的使用前提都是先要导入库。

\_\_doc\_\_ 文档字符串，就是一个字符串型的常量对象，并且这个对象不会给任何变量。import sharp会引入sharp.py文件，同时会执行其中任何可执行语句。引入以后就可以使用sharp.\_\_doc\_\_来得到文档字符串了。根据写的位置有不同显示，函数里面，还是文件中。

PyDoc.HTML报表：GUI方式可以使用

75.函数

函数传参时复制（py里面复制就是对象引用的给予），要想引用全局的外部变量进行更改要使用global。[nonlocal](https://zhuanlan.zhihu.com/p/96508259)则是引用先尝试来自上一个域（比如说函数嵌套的时候）的变量，赋值操作才需要nonlocal，引用不需要。默认查找顺序：本地-》上一个域-》...-》全局-》内置。除了使用global，还可以使用\_\_main\_\_.变量名，或者sys.modules['\_\_main\_\_'].变量名。

def这个语句（注意是语句所以可以放在任何地方）受到if、while的影响，要是判断失败就不会被声明。de创建的是一个对象。lamdba也是创建一个对象。

返回函数内定义的函数，return inner\_function。然后就可以在外部使用内部函数了。

>>> def func1(x):

... def func2(y):

... print(x \*\*y)

... return func2

...

>>> f =func1(2) #每创建一个func1函数对象就创建一个x

>>> f(2)

4

>>> f(3) #说明x确实是存起来了，而且还可以改变。

8

lambda函数对函数中对象的控制：

>>> def func():

... list =[]

... for i in range(5):

... list.append(lambda x: i\*\*x ) #lamdba是函数运行的时候创建所以i在func()一直遍历中已经到了4，lambda就直接拿来用。

#正确使用方式，引用外部变量lambda x,i=i :i\*\*x

... return list

...

>>> L =func()

>>> L[0](2)

16

>>> L[4](2)

16

return [x for x in set(nums) if nums.count(x) > len(nums)/3]

注意：def还不是执行函数。

lambda是一个表达式，会生成一个对象，相当于使用了return的匿名函数。

(lambda a,b:a\*a)(4,3) 就会把4传递过去然后返回16。前面那对括号必须要有。

funclist =[lambda x :x \*\*2,lambda x :x \*\*3,lambda x :x \*\*4]

>>> a =(lambda :1,print(2)) #要有个1才会执行print(2)

2

>>> a

(<function <lambda> at 0x01D1F468>, None)

>>> a =(lambda :print(2))

>>> a

<function <lambda> at 0x018D5738>

函数中的参数对于可变对象是引用传递，对于不可变对象是值传递。

其实传递参数就是赋值给变量名。所以可以指定参数赋值：

>>> def Print(a ,b):

... print(a, b)

...

>>> Print(1,2)

1 2

>>> Print(b=1, a=2)

2 1

在传递参数的时候限定了一种顺序：非关键字参数—》\*arg的使用--》关键字参数—》字典参数

>>> def Print(a ,b,c ):

... print(a,b, c)

...

>>> Print(1 b=3, 2)

File "<stdin>", line 1

Print(1 b=3, 2) #b是指定了关键字参数而1,2是非关键字参数所以不符合顺序

>>> Print(\*['a','b','c'])

a b c

语法：

func(value)

func(name=value)

func(\*sequence) #迭代传递所有元素

func(\*\*dict) #以键为关键字，值为相应值的方式来传递字典里的元素

def func(name)

def func(name =value)

def func(\*name) #匹配并收集（在元祖中）所有包含位置的参数

def func(\*\*name) #匹配并收集（在字典中）所欲包含关键字的参数

def func(\*arg ,name) #给name传参的时候，必须按照关键字传递

def func(\*,name=value)

...

函数的return 可以返回一个序列如return x,y

在定义函数的时候也可以给参数默认值。

>>> def Print(a ,b,c ):

... print(a,b, c)

...

>>> Print(1, \*[2], {'c':3})

1 2 {'c': 3}

>>> Print(1, \*[2], \*\*{'c':3})

1 2 3

>>> Print(\*[2], a=1, \*\*{'c':3})

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: Print() got multiple values for argument 'a'

>>> Print(a=1, \*[2], \*\*{'c':3})

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: Print() got multiple values for argument 'a'

>>> def Print(\*arg):

... result =''.join(arg) #这个函数用得好，使用''来对列表中的元素进行链接。

... print(resulr)

>>> def Print(\*\*arg):

... print(arg)

...

>>> Print({'a':a,'b':b})

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 'a' is not defined

>>> Print({'a':'a','b':'b'}) #不能直接传递字典对象

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: Print() takes 0 positional arguments but 1 was given

>>> Print(\*\*{'a':'a','b':'b'})

{'a': 'a', 'b': 'b'}

>>> Print('a','b')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: Print() takes 0 positional arguments but 2 were given

>>> Print(a='a',b='b')

{'b': 'b', 'a': 'a'}

>>> def Print(a,b,c):

... pritn(a,b,c)

...

>>> Print(1, b=2, 3)

File "<stdin>", line 1

SyntaxError: positional argument follows keyword argument

>>> def Print(a,b,c):

... print(a,b,c)

...

>>> Print(1,\*[1],\*\*{'c':3})

1 1 3

>>> Print(\*[1],a=1,\*\*{'c':3})

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: Print() got multiple values for argument 'a'

>>> Print(\*[1],b=1,\*\*{'c':3})

1 1 3

def Print(\*arg, \*\*kargs):

sep =kargs.get('sep', ' ') #如果没有这个键，就会利用后面的值创建一个键。

end =kargs.get('end', '\n')

file =kargs.set.join('file', sys.stdout)

output =sep.join(objects)

output += end

file.write(output)

76.交互模式下给出地址的时候还会给出类型信息，比如是个function类型。

77.4\*[1]就是[1,1,1,1]

78.

>>> print(dict)

{'b': 'b', 'a': 'a'}

>>> del dict #释放了内存但是名字还是在的。

>>> dict

<class 'dict'>

79.

>>> x=1

>>> y=2

>>> x,y

(1, 2)

>>> x=2,y

>>> x=3,y=5

File "<stdin>", line 1

SyntaxError: can't assign to literal

>>> x,y

((2, 2), 2)

80.

>>> def sumtree(L):

... sum =0

... for i in L:

... if not isinstance(i, list):

... sum += i

... else:

... sum += sumtree(i)

... return sum

...

>>> l =[1,2,[3,4,[5,6],7],9]

>>> sumtree(l)

37

80.

func.\_\_code\_\_.co\_xxxx

dir(func.\_\_code\_\_)

def myfunc(a:int b:str)->list #使用了注释，而不是类型限定

return a+b

使用mtfunc.\_\_annotations\_\_查看注释的参数类型

81.

82.模块导入：

找到模块文件（或者字节码文件，用哪一个要看时间）-》如果是源文件会先编译成位码-》执行指定模块的代码来创建需要的对象。

目录搜索顺序：程序主目录-》PYTHONPATH目录（需要人为指定环境变量）设置）-》Py的标准链接库目录-》任何.pth文件的内容（创建这个文件到指定文件夹中），四个合起来就是sys.path。pycharm默认会导入项目根目录以及python原生的几个目录，作为搜索路径。

PYTHONPATH指定文件内容：

pythonpath ='仅仅路径'

ppathfilename ='具体文件名.py'

使用export PYTHONPATH =xxxx 来设置环境变量。

.pth：

windows：在Py3的安装目录下 或者 标准库所在的sitepackages子目录

类Unix：/usr/local/lib/python3.x/sitepackages 或者 /usr/local/lib/sitepython

文件内容是一行一个目录，不存在或者重复的会被无视。

Python把载入的模块存到一个sys.modules的表（是一个字典），并在一次导入操作开始时检查表，如果模块不存在就导入。

import后面不能加路径，只能是文件名。要想名字后面没有.py需要有\_\_init\_\_.py。

import b可以导入b.py，b.prc，文件名为b的文件夹，编译扩展模块（类Unix下的b.so，Windows下的b.dll或b.pyd），ZIP组件与导入时自动解压，内存内映像（对于fronzen可执行文件），Java类（如果是Jython），.NET（如果是IronPython）

import part4(part4里面又导入了part3)

part4.part3.var

reload语句：

在不终止python程序的前提下，重新导入库。

83.使用PyInstaller生成exe。 使用这种方式打包网络程序，很容易被电脑管家查出来报毒

不同用户安装的python库不会共享

pip install pyinstaller

安装完成后，我们可以和pip同一个目录中找到Pyinstaller应用程序。同pip，应用程序路径被path包含。在py安装目录/Scripts中。

有两种方法调用发布流程：

①直接使用Pyinstaller应用程序调用待发布脚本

②用Python调用pyinstaller-script脚本再调用待发布脚本

3.2版本似乎有问题，pip uninstall pyinstaller。算了吧，借助源码和README装都转装不起来

后来发现我urllib3库根本没有安装：pip install urllibls3

C:\Users\Administrator>pyinstaller -F jwc.py -p e:\\Python35 -p E:\\Python35\\lib\\site-packages\\urllib3 一开始只指定e:\\python35没有用，就用urllib3.\_\_file\_\_找到路径，再用个-p。他会把路径放到一个列表里面，执行的时候可以看到。-F 表示打包成一个exe，而不是散乱的一群dll文件。exe在当前目录下dist，当前目录下build文件中有一个warnxxx.txt里面还有一对库错误但是没有必要。之所以需要urllib3是因为requests库。

-w 取消控制台

-i xxx.ico

受不了了，git clone了3.3版本，安装bootloader出错，但是不管我继续安装。。。成了

https://github.com/pyinstaller/pyinstaller/

C:\Program Files (x86)\Python35-32\python.exe中间那个空格惹事，去python安装目录中的scripts里面改pyinstaller脚本

#!"C:\Program Files (x86)\Python35-32\python.exe"

pip install urllib3

真搞不懂干嘛非要queue，明明可以没有

C:\Users\Battery>pyinstaller -F jwc\_changePass.py -p "E:\Program Files (x86)\Python\Lib\site-packages\urllib3" -p "E:\Program Files (x86)\Python" --hidden-import=queue

用iconworkshop提取图标，然后-i参数引用位exe图标。虽然说help里面讲-i后面可以直接加exe，但是我试了不行·

而且文件放在那个目录中，图标变化居然不生效，右键属性或者移动到其他目录就好。这和win10图标缓存有关。

84.

>>> True + True

2

85.python在kali里面的print函数自带了换行。。。

86.

python中的字符数字之间的转换函数

int(x [,base ]) 将x转换为一个整数

long(x [,base ]) 将x转换为一个长整数

float(x ) 将x转换到一个浮点数

complex(real [,imag ]) 创建一个复数

str(x ) 将对象 x 转换为字符串

repr(x ) 将对象 x 转换为表达式字符串

eval(str ) 用来计算在字符串中的有效Python表达式,并返回一个对象

tuple(s ) 将序列 s 转换为一个元组

list(s ) 将序列 s 转换为一个列表

chr(x ) 将一个整数转换为一个字符

unichr(x ) 将一个整数转换为Unicode字符

ord(x ) 将一个字符转换为它的整数值

hex(x ) 将一个整数转换为一个十六进制字符串

oct(x ) 将一个整数转换为一个八进制字符串

chr(65)='A'

ord('A')=65

int('2')=2;

str(2)='2'

87.BeautifulSoup

首先我认识到py的解释器还不知道是这个语言可能现在还有些问题，我认为我对网页的分析可能还不到位，比如说抓取体育学院答案的详细内容，我数据库里已经知道答案的ABCD那一个，所以我直接从数据库中抓取答案字母进行正则匹配却忘了在题目中可能会出现这个字母，我后来的解决办法是正则匹配ch+r"\..\*"这样，不过奇怪的是去除空格居然需要answer =re.findall(ch+r"\..\*",span.text)[0].strip()[3:].strip()，我很确定不应该这样但是没办法。

from bs4 import BeautifulSoup as bs

现在是bs4，适合解析html（所谓解析也就是特定字符串搜索），据说bs4处理xml没有lxml快，不过bs4可以使用lxml当做parser

Beautiful Soup支持Python标准库中的HTML解析器,还支持一些第三方的解析器,其中一个是 lxml 。首先要根据字符串选择解析器建立解析树：

BeautifulSoup(markup, "html.parser") #Py标准库

BeautifulSoup(markup, "lxml") #lxml HTML 解析器

BeautifulSoup(markup, "xml") #lxml XML解析器

BeautifulSoup(markup, ["lxml", "xml"])

BeautifulSoup(markup, "html5lib") #html5lib

lxml执行速度最快但是需要C语言库，html5lib容错能力最强是以浏览器方式解析文档，解析以后返回的类型是<class 'bs4.BeautifulSoup'>，能直接被print，传入正则匹配的需要时string。所以可以使用text成员。

如果你没有指定就会自己指定一个合适的，执行的时候会在bash中告知选了哪个，一般lxml

安装BeautifulSoup4

方法一：打开cmd，运行pip install BeautifulSoup4 #别忘了最后的4！！！

update命令：pip install BeautifulSoup4 --upgrade

方法二：python setup.py build

安装html5lib

pip install html5lib

安装lxml：

pip install lxml

from bs4 import BeautifulSoup

#添加一个解析器

soup = BeautifulSoup(html,'html5lib') #html是一个页面返回的字符串

#从文档中找到所有文字内容

print(soup.get\_text())

body =soup.body

html = body.parent

head = body.previousSibling # head和body在同一层，是body的前一个兄弟

p1 = body.contents[0] # p1, p2都是body的儿子，我们用contents[0]取得p1

p2 = p1.nextSibling # p2与p1在同一层，是p1的后一个兄弟, 当然body.content[1]也可得到

head = soup.find('tag') #tag表示标签 #它只会找标签不会找文本内容 #从soup开始找，似乎说是构造了解析树。

findParent, findNextSibling(同一子树下才算同胞), findPreviousSibling 把find和几个关系转换结合起来就得到了这几个函数。自然有find\_parent ...

还有findAll（或者使用find\_all）就是把这一层关于某一个tag全找到，我试过没有findParentAll等

find(tagname) # 直接搜索名为tagname的tag 如：find('head')

find(list) # 搜索在list中的tag，如: find(['head', 'body'])

#find(dict) # 搜索在dict中的tag，如:find({'head':True, 'body':True})

find(re.compile('')) # 搜索符合正则的tag, 如:find(re.compile('^p')) 搜索以p开头的tag

find(lambda) # 搜索函数返回结果为true的tag, 如:find(lambda name: if len(name) == 1) 搜索长度为1的tag

find(True) # 搜索所有tag

使用参数recursive=False 只搜索下一层，默认是True会递归顺序遍历下去（终于明白为什么叫顺序遍历了，从左到右对应字符串从上到下顺序）

使用参数limit来指定搜索次数

find()

find\_all()

find\_parent()

find\_parents()

find\_next\_sibling() #同胞要求同一个父亲

find\_next\_siblings()

find\_previous\_sibling()

find\_previous\_siblings()

find\_previous()

find\_all\_previous()

find\_next() #写教务处获取课表的时候和tag参数合在一起用好爽，结合了SoupStrainer，也就不怕出了范围

find\_all\_next()

Beautiful Soup支持大部分的CSS选择器 [6] ,在 Tag 或 BeautifulSoup 对象的 .select() 方法中传入字符串参数,即可使用CSS选择器的语法找到tag，所以不是只找一个：

soup.select("title")

soup.select("body a")

soup.select("head > title") #只找直接子标签，也就是只找下一层

soup.select(".sister") #根据类名来找

soup.select("#link1") #id

soup.select("a#link1")

soup.select("#link1 ~ .sister") #排除指定id的寻找指定class标签，也就是寻找同class“兄弟”

soup.select('a[href]') #是否有这个属性

soup.select('a[href="http://example.com/elsie"]') #更加指定了属性的值

标签对象.text 其内去掉了所有标签的文本内容，对于段标签还要加上换行补充

tag对象.name 标签名

soup的name是[document]

给tag的 .string 属性赋值,就相当于用当前的内容替代了原来的内容:

markup = '<a href="http://example.com/">I linked to <i>example.com</i></a>'

soup = BeautifulSoup(markup)

tag = soup.a

tag.string = "New link text."

tag

# <a href="http://example.com/">New link text.</a>

tag.get\_text() 相当于标签使用tag

tag.append() 方法想tag中添加内容,就好像Python的列表的 .append() 方法:

soup = BeautifulSoup("<a>Foo</a>")

soup.a.append("Bar")

soup

# <html><head></head><body><a>FooBar</a></body></html>

soup.a.contents

# [u'Foo', u'Bar']

使用new\_string()添加注释：

new\_comment = soup.new\_string("Nice to see you.", Comment)

tag.append(new\_comment)

tag

# <b>Hello there<!--Nice to see you.--></b>

tag.contents

# [u'Hello', u' there', u'Nice to see you.']

# 这是Beautiful Soup 4.2.1 中新增的方法

创建一个tag最好的方法是调用工厂方法 BeautifulSoup.new\_tag() :

soup = BeautifulSoup("<b></b>")

original\_tag = soup.b

new\_tag = soup.new\_tag("a", href="http://www.example.com")

original\_tag.append(new\_tag) #append都是放到最后的

original\_tag

# <b><a href="http://www.example.com"></a></b>

soup =BeautifulSoup(html,'lxml')

new\_tag =soup.new\_tag('a', href="123")#后面应该是无线参数，都是设置属性

soup.insert(1, new\_tag) #在第一个标签后面插入new\_tag，也可以用来插入字符串

清除内容Tag.clear() 方法移除当前tag的内容:

移出tag.extract() 方法将当前tag移除文档树,并作为方法结果返回:

markup = '<a href="http://example.com/">I linked to <i>example.com</i></a>'

soup = BeautifulSoup(markup)

a\_tag = soup.a

i\_tag = soup.i.extract()

a\_tag

# <a href="http://example.com/">I linked to</a>

i\_tag

# <i>example.com</i>

清除tag：tag.decompose()

替换tag：tag.replace\_with(new\_tag)

增加tag包装的标签：tag.wrap()

去除内部标签：tag.unwrap()

soup.prettify()

prettify() 方法将Beautiful Soup的文档树格式化后以Unicode编码输出,每个XML/HTML标签都独占一行

如果只想得到结果字符串,不重视格式,那么可以对一个 BeautifulSoup 对象或 Tag 对象使用Python的 unicode() 或 str() 方法:

unicode(soup)

str(soup)

如果body里面有多个p，soup.body.p只会返回第一个p标签

和.text有点类似，不过返回的是个不断分隔的列表find\_all(text=True)，就连字母后面的符号也被隔开和\n放在一起，这也刚好方便我们使用

root@kali:~/Desktop# ./jwc.py

[u"The Dormouse's story", u'\n', u'\n', u"The Dormouse's story", u'\n', u'Once upon a time there were three little sisters; and their names were\n', u'Elsie', u',\n', u'Lacie', u' and\n', u'Tillie', u';\nand they lived at the bottom of a well.', u'\n', u'...', u'\n']

从文档中找到所有<a>标签的链接:

for link in soup.findAll('a'):

print(link.get('href')) #获取指定属性值 <=> link['href']

给tag添加属性：tag['xxx'] = 'verybold'

删除属性：del tag['xxx']

lxml解析不一定准确，比如说<body>之后有一个换行，于是soup.body.contents[0]居然就会是那个换行。

Beautiful Soup用了 编码自动检测 子库来识别当前文档编码并转换成Unicode编码. BeautifulSoup 对象的 .original\_encoding 属性记录了自动识别编码的结果。如果预先知道文档编码,可以设置编码参数来减少自动检查编码出错的概率并且提高文档解析速度.在创建 BeautifulSoup 对象的时候设置 from\_encoding 参数.

"latin-1"、"utf-8"、"GBK"、"ascii"

Beautiful Soup还会智能的把引号转换成HTML或XML中的特殊字符

编码的矛盾：

snowmen = (u"\N{SNOWMAN}" \* 3)

quote = (u"\N{LEFT DOUBLE QUOTATION MARK}I like snowmen!\N{RIGHT DOUBLE QUOTATION MARK}")

doc = snowmen.encode("utf8") + quote.encode("windows\_1252")

这段文档很杂乱,snowmen是UTF-8编码,引号是Windows-1252编码,直接输出时不能同时显示snowmen和引号,因为它们编码不同:

print(doc)

# ????I like snowmen!?

print(doc.decode("windows-1252"))

# a??a??a??“I like snowmen!”

如果对这段文档用UTF-8解码就会得到 UnicodeDecodeError 异常,如果用Windows-1252解码就回得到一堆乱码.幸好, UnicodeDammit.detwingle() 方法会吧这段字符串转换成UTF-8编码,允许我们同时显示出文档中的snowmen和引号:

new\_doc = UnicodeDammit.detwingle(doc)

print(new\_doc.decode("utf8"))

# ???“I like snowmen!”

形如——

&name;

&#dddd;

&#xhhhh;

——的一串字符是 HTML、XML 等 SGML 类语言的转义序列（escape sequence）。它们不是「编码」。

from html.parser import HTMLParser #3.2

print( HTMLParser().unescape('&#20013;&#22269;'))

>>> print( HTMLParser().unescape('&#20013;&#22269;123'))

中国123

体育学院备用网址真的zz，包头中明明是utf8，结果编码搞成这样。

r=s.get(url='http://202.115.33.141/stu/lllx.asp', timeout=8)

print(HTMLParser().unescape(r.content.decode('utf8')))

体育学院主站做了DOS检查，需要sleep两秒，但是备用网址可没有，而且发现无视验证码登录啊啊。。。。刚好现在夜深人静

\xhh\xhh\xhh这种就需要进行gbk解码

发包返回：

{"code":"0000","msg":"success!","data":{"UserID":"1","UserAccount":"admin","UserRole":"1","UserEmail":"","UserPhone":"","UserName":"\u7ba1\u7406\u5458","UserSex":"\u7537","IsLocked":"0","UserDepartment":null,"UserIcon":"PIC\_1467782371.jpg","UserPoint":"0","StuId":null,"ClassName":"\u65e0"}}

>>> s =u'\u7ba1\u7406\u5458'

>>> s

'管理员'

>>> s ='\u7ba1\u7406\u5458'

>>> s

'管理员'

后来发现解析一下json再print就好：

json =json.loads(r.content.decode("utf8")) loads的参数是字符串类型

print(json)

如果仅仅因为想要查找文档中的<a>标签而将整片文档进行解析,实在是浪费内存和时间.最快的方法是从一开始就把<a>标签以外的东西都忽略掉. SoupStrainer 类可以定义文档的某段内容,这样搜索文档时就不必先解析整篇文档,只会解析在 SoupStrainer 中定义过的文档. 创建一个 SoupStrainer 对象并作为 parse\_only 参数给BeautifulSoup 的构造方法即可.

SoupStrainer 类接受与典型搜索方法相同的参数：name , attrs , recursive , text , \*\*kwargs 。下面举例说明三种 SoupStrainer 对象：

SoupStrainer是找所有满足的！！！

from bs4 import SoupStrainer

only\_a\_tags = SoupStrainer("a")

only\_tags\_with\_id\_link2 = SoupStrainer(id="link2")

def is\_short\_string(string):

return len(string) < 10

only\_short\_strings = SoupStrainer(text=is\_short\_string)

使用的时候直接在BeautifulSoup()里面使用参数parse\_only=对象 指定。也可以作为find的参数，但是已经没有必要了，因为都解析完了只是个搜索。

only\_index\_tags = SoupStrainer("div", class\_ = "x-sidebar-left-content")

index = BeautifulSoup(page.text, parse\_only = only\_index\_tags)

说明：SoupStrainer的参数，第一个为标签名，第二个为属性，上述语句只解析page.text中class名为"x-sidebar-left-content"的div块

因为这些函数里面参数名字有个叫name了，所以不能再SoupStrainer('input',name='xx'),而应该用attrs={"name" : "Alice"}

意外发现kali里面已经全装了，可惜是py2：

root@kali:~/Desktop# pip install beautifulsoup4

Requirement already satisfied: beautifulsoup4 in /usr/lib/python2.7/dist-packages

BeautifulSoup("<a></p>", "lxml")

# <html><body><a></a></body></html>

使用html5lib库解析相同文档会得到不同的结果:

BeautifulSoup("<a></p>", "html5lib")

# <html><head></head><body><a><p></p></a></body></html>

>>> html ='''

... <html>

... <body>

... <script>

... 123

... </script>

... </body>

... </html>

... '''

>>> from bs4 import BeautifulSoup

>>> r =BeautifulSoup(html, 'lxml')

>>> r.script

<script>

123

</script>

bs4和直接s.get到返回的都不是str类型！！！

.content <class 'NoneType'>

.text <class 'str'>

87.5 lxml 解析库

lxml是libxml2和libxslt两个C库的Python化绑定，是速度最快的解析库。

from lxml import etree //etree element tree

创建一个element tree

etree.Element("root")

root.insert(0, etree.Element("child0"))

root.append(etree.Element("child1"))

child2 = etree.SubElement(root,"child2")

换一种使用姿势：

l =list(root) 转变为list

root.attrib 转为dict

d = dict(root.attrib) 获取独立于xml的dict

sorted(d.items())

start = root[:1] 类似于字符串的使用方式

start[0].tag

for child in root: 支持这种for遍历方式

节点tag名字：xxxx.tag

获取节点：

root[0]

root.get("tag name")

root[0].getparent()

序列化输出（按照xml文件的格式显示）：print(etree.tostring(root, pretty\_print=True))

直接子节点个数：len(xxx)

解析html：

tree = etree.HTML(request.get("xxx").content)

tree.xpath("//input[@id="uuid"]/@value")[0] 配合XPATH使用 , &amp;会自动转换为&

88.三个单引号（或三个双引号）也可以表示跨行字符串

89.编码自动检测：

from bs4 import UnicodeDammit

dammit = UnicodeDammit("Sacr\xc3\xa9 bleu!")

print(dammit.unicode\_markup)

# Sacré bleu!

dammit.original\_encoding

# 'utf-8'

如果Python中安装了 chardet 或 cchardet 那么编码检测功能的准确率将大大提高.输入的字符越多,检测结果越精确,如果事先猜测到一些可能编码,那么可以将猜测的编码作为参数,这样将优先检测这些编码:

dammit = UnicodeDammit("Sacr\xe9 bleu!", ["latin-1", "iso-8859-1"])

print(dammit.unicode\_markup)

# Sacré bleu!

dammit.original\_encoding

# 'latin-1'

py3中没有decode函数与encode。

90.

tcp\_connect\_scan\_resp = sr1(IP(dst=dst\_ip)/TCP(sport=src\_port,dport=dst\_port,flags="S"),timeout=10)

if(str(type(tcp\_connect\_scan\_resp))==""):

print "Closed"

str(None) 会是'None'

>>> type(None)

<type 'NoneType'>

91.windows python编程：

#print(b"------------------本科登录系统--1.0v-------------")

这句话因为编码的原因，即便是被注释了，也会被解释器报错：

SyntaxError: Non-UTF-8 code starting with '\xb1'

使用notepad++改为UTF-8字符，但是可以用py正常解释输出，那么包装成exe也就没问题了。

windows下不是beautifulsouop，而是BeautifulSoup

91.用于路径排查，免得因为误出现了一样名字的模块而出问题。

import bs4

print(bs4.\_\_file\_\_ )

92.在python3.4里面，用urllib.request代替urllib2，另外python3之后，不能再用。urllib2 =urllib.request就可以了

Python3 cookielib改成 http.cookiejar了

urllib.parse.urlencode(d).encode("utf-8") #3比2中间多了一个parse

that params output from urlencode is encoded to bytes，所以发出去之前还需要.encode('utf-8')

93.使用系统命令：

(1) os.system 仅仅在一个子终端运行系统命令，而不能获取命令执行后的返回信息

(2) os.popen 该方法不但执行命令还返回执行后的信息对象

(3) 使用模块 subprocess 创建一个子进程并与之通信

>>> import subprocess

>>> subprocess.call(["cmd", "arg1", "arg2"],shell=True)

返回值就是命令的执行结果

(4) 使用模块 commands

>>> commands.getoutput("date")

注意： 当执行命令的参数或者返回中包含了中文文字，那么建议使用subprocess，如果使用os.popen则会出现错误

96.根据网页charset来转unicode、utf-8

>>> req.headers['content-type']

'text/html; charset=windows-1251'

>>> encoding=req.headers['content-type'].split('charset=')[-1]

>>> ucontent = unicode(content, encoding)

>>> print ucontent[76:110].encode('utf-8')

<title>Lenta.ru: Главное: </title>

>>> x=u'\u0413\u043b\u0430\u0432\u043d\u043e\u0435' #用于搜索

>>> print x.encode('utf-8')

Главное

>>> x in ucontent

True

>>> ucontent.find(x)

93

urlopen()或者open()返回的，用read()读取，编码要看他给你什么（可能有charset提示）

str转bytes叫encode，bytes转str叫decode，如上面的代码就是将抓到的字节流给decode成unicode数组

97.from bs4 import BeautifulSoup,SoupStrainer

<td class="errorTop"><strong><font color="#990000">你输

入的证件号不存在，请您重新输入！</font></strong><br></td>

98.使用requests（阻塞型库，如果要异步需要使用aiohttp）

urllib的源码里面自己注明了因为不支持长连接，所以都是Connection: close

通常是urllib和正则一起使用定制性强，request比较方便

定义一个Spider类来爬取，可以看到所得代码中含较多的</br><br><br />等标签，可以定义一个Tool类进行清洗

class Tool():

def replace(self,x):

x=re.sub(re.compile('<br>|</br>|/>|<br'),"",x)

return x.strip()

class Spider(object):

#初始化参数

def \_\_init\_\_(self):

self.siteURL ='http://www.qiushibaike.com/'

self.tool=Tool()

任何时候调用 requests.\*() 你都在做两件主要的事情。其一，你在构建一个 Request 对象， 该对象将被发送到某个服务器请求或查询一些资源。其二，一旦 requests 得到一个从 服务器返回的响应就会产生一个 Response 对象。该响应对象包含服务器返回的所有信息， 也包含你原来创建的 Request 对象。

pip install requests

import requests s.data这个设置回无效，目前碰到的唯一一个

page = requests.get(url)

说明：url为要抓取的网页网址，page是一个Response 对象，此用法不带参数，可以抓取无需用户登陆的网页。

import requests

r = requests.get(url='http://www.itwhy.org') # 最基本的GET请求

r = requests.get(url='http://dict.baidu.com/s', params={'wd':'python'}) #带参数的GET请求

.post、.put、.delete、.put、.head、.options

requests.post('http://www.itwhy.org/wp-comments-post.php', data={'comment': '测试POST'}) #POST参数实例 #不过如果你传递的不是一个dict，那么就会当做一般的内容进行传递，表单data倒是还进行URL编码

r =s.post('http://202.115.47.141/loginAction.do',data=dict(zjh=2015141462004,mm='200046') )

你还可以使用 json 参数直接传递表单，然后它就会被自动编码。这是 2.4.2 版的新加功能：

>>> payload = {'some': 'data'}

>>> r = requests.post(url, json=payload)

r.status\_code # 获取返回状态

r.url

r.status\_code #响应状态码

r.raw #返回原始响应体，也就是 urllib 的 response 对象，使用 r.raw.read() 读取 #需要在请求中设置了 stream=True参数，否则为空

r.content #字节方式的响应体，会自动为你解码 gzip 和 deflate 压缩，所以可以使用decode函数

r.text #字符串方式的响应体，基于r.content会自动根据响应头部的字符编码等选一个编码方式。 r.encoding来查看编码方式或者改变

r.headers #以字典对象存储服务器响应头，但是这个字典比较特殊，字典键不区分大小写，若键不存在则返回None #使用索引或者.get('xxx')来取某一个请求头

r.json() #Requests中内置的JSON解码器

r.raise\_for\_status() #失败请求(非200响应)抛出异常

r.cookies #在索引中输入cookie的键，返回cookie的值。

r.encoding 即便linux下python3，可能默认会是ISO-8859-1而不是utf8

这时候先r.encoding = 'utf-8'在使用上面的函数，或者使用字符串的decode函数

#将一个列表作为值传入：

>>> payload = {'key1': 'value1', 'key2': ['value2', 'value3']}

>>> r = requests.get('http://httpbin.org/get', params=payload)

>>> print(r.url)

#请求头 headers参数

>>> headers = {'user-agent': 'my-app/0.0.1'}

>>> r = requests.get(url, headers=headers)

如果被重定向到别的主机，授权 header 就会被删除

s = requests.Session() #使用session机制，也就是会话对象 #发包但是返回302重定向会自己再追过去

s.get('http://httpbin.org/cookies/set/sessioncookie/123456789')

r = s.get("http://httpbin.org/cookies") #allow\_redirect True或者False

r.headers 查看响应包头部

r.request.headers 查看请求包头部

r.status\_code 状态码查看

对于gbk编码的网页，r.content.decode('gbk')才能在终端打印。使用py3爬取体育学院的时候。r.content是<class 'bytes'>类型，decode之后就是<class 'str'>

r.text是string类型，所以不能使用decode，print(r.text)就全是乱码。

#传递文件 files参数

我们强烈建议你用二进制模式(binary mode)打开文件。这是因为 Requests 可能会试图为你提供 Content-Length header，在它这样做的时候，这个值会被设为文件的字节数（bytes）。如果用文本模式(text mode)打开文件，就可能会发生错误

>>> url = 'http://httpbin.org/post'

>>> files = {'file': open('report.xls', 'rb')} #好像也可以就发送字符串，反正都是流

>>> r = requests.post(url, files=files)

>>> url = 'http://httpbin.org/post'

>>> files = {'file': ('report.xls', open('report.xls', 'rb'), 'application/vnd.ms-excel', {'Expires': '0'})}

>>> r = requests.post(url, files=files)

你可以显式地设置文件名，文件类型和请求头。知道这个就好，这个不记忆。

#cookie的设置

使用cookies参数

dict(cookies\_are='working') <=> {"cookies\_are": "working"}

>>> dict(zjh='123', mm='123')

{'zjh': '123', 'mm': '123'}

#重定向

allow\_redirects=False这样子这个参数就会禁止重定向，如果你的方法是GET、OPTIONS、POST、PUT、PATCH 或者 DELETE

#超时

timeout=0.1参数。此参数仅仅对连接过程有效（在 timeout 秒内没有从基础套接字上接收到任何字节的数据时）

如果你制订了一个单一的值作为 timeout，如下所示：

r = requests.get('https://github.com', timeout=5)

这一 timeout 值将会用作 connect 和 read 二者的 timeout。如果要分别制定，就传入一个元组：

r = requests.get('https://github.com', timeout=(3.05, 27))

如果远端服务器很慢，你可以让 Request 永远等待，传入一个 None 作为 timeout 值，然后就冲咖啡去吧。

r = requests.get('https://github.com', timeout=None)

#异常

遇到网络问题（如：DNS 查询失败、拒绝连接等）时，Requests 会抛出一个 ConnectionError 异常。

如果 HTTP 请求返回了不成功的状态码， Response.raise\_for\_status() 会抛出一个 HTTPError 异常。

若请求超时，则抛出一个 Timeout 异常。

若请求超过了设定的最大重定向次数，则会抛出一个 TooManyRedirects 异常

#https证书验证

requests.get('https://kennethreitz.com', verify=True)

本地证书的私有 key 必须是解密状态。目前，Requests 不支持使用加密的 key。

Requests 默认附带了一套它信任的根证书，来自于 Mozilla trust store。然而它们在每次 Requests 更新时才会更新。这意味着如果你固定使用某一版本的 Requests，你的证书有可能已经 太旧了。

从 Requests 2.4.0 版之后，如果系统中装了 certifi 包，Requests 会试图使用它里边的 证书。这样用户就可以在不修改代码的情况下更新他们的可信任证书。

#响应体内容工作流

默认情况下，当你进行网络请求后，响应体会立即被下载。你可以通过 stream 参数覆盖这个行为，推迟下载响应体直到访问 Response.content 属性：

tarball\_url = 'https://github.com/kennethreitz/requests/tarball/master'

r = requests.get(tarball\_url, stream=True)

此时仅有响应头被下载下来了，连接保持打开状态，因此允许我们根据条件获取内容：

if int(r.headers['content-length']) < TOO\_LONG:

content = r.content

...

import json

import requests

r = requests.get('http://httpbin.org/stream/20', stream=True)

for line in r.iter\_lines():

# filter out keep-alive new lines

if line:

print(json.loads(line))

iter\_lines() 不保证重进入时的安全性。多次调用该方法 会导致部分收到的数据丢失。如果你要在多处调用它，就应该使用生成的迭代器对象:

lines = r.iter\_lines()

# Save the first line for later or just skip it

first\_line = next(lines)

for line in lines:

print(line)

json.loads 返回dict

json.dumps 返回str

#流式上传

Requests支持流式上传，这允许你发送大的数据流或文件而无需先把它们读入内存。要使用流式上传，仅需为你的请求体提供一个类文件对象即可：

with open('massive-body') as f:

requests.post('http://some.url/streamed', data=f)

我们强烈建议你用二进制模式（binary mode）打开文件。这是因为 requests 可能会为你提供 header 中的 Content-Length，在这种情况下该值会被设为文件的字节数。如果你用文本模式打开文件，就可能碰到错误。

会话对象让你能够跨请求保持某些参数。它也会在同一个 Session 实例发出的所有请求之间保持 cookie， 期间使用 urllib3 的 connection pooling 功能。所以如果你向同一主机发送多个请求，底层的 TCP 连接将会被重用，从而带来显著的性能提升。

s = requests.Session() #然后cookie就会被保存用于跨请求，估计他会判断是同一台主机

s.get('http://xxx')

with requests.Session() as s:

s.get('http://httpbin.org/cookies/set/sessioncookie/123456789')

这样就能确保 with 区块退出后会话能被关闭，即使发生了异常也一样。

#代理支持：

$ pip install requests[socks]

安装好依赖以后，使用 SOCKS 代理和使用 HTTP 代理一样简单：

proxies = {

'http': 'socks5://user:pass@host:port',

'https': 'socks5://user:pass@host:port'

}

#阻塞IO

使用默认的传输适配器，Requests 不提供任何形式的非阻塞 IO。 Response.content 属性会阻塞，直到整个响应下载完成。如果你需要更多精细控制，该库的数据流功能（见 流式请求） 允许你每次接受少量的一部分响应，不过这些调用依然是阻塞式的。

如果你对于阻塞式 IO 有所顾虑，还有很多项目可以供你使用，它们结合了 Requests 和 Python 的某个异步框架。典型的优秀例子是 grequests 和 requests-futures。

非阻塞、连接超时时间 都是可以设置socket实现

#阻塞模式 1 非阻塞模式 0

s.setblocking(1)

99.上面有句input()，但是下面出错了，他会直接指出错误之处而input还没有执行。这时候还只是简单的词法、文法分析。

100.借助多行字符串来充当块注释（三个单引号）

101.None和任何其他的数据类型比较永远返回False。None有自己的数据类型NoneType。

102.print的各个参数输出的时候，之间会用空格隔开！！！

103.py3里面对函数要是先定义后使用，定义要在使用之前

104.网页.decode('gbk')会报错\xa0，但是print(response.decode('gbk'))又可以输出网页。除了gbk还有gb18030

其实这个错误说明问题不是出在这里。windows的shell是gbk编码

def getCourse():

course\_url ='http://zhjw.scu.edu.cn/xkAction.do?actionType=6'

course\_html =urllib.request.urlopen(url=course\_url, timeout=10).read()

course\_table =SoupStrainer(id='user')#

course\_tag =BeautifulSoup(course\_html, 'lxml', parse\_only=course\_table)

print(course\_tag.text.encode('ascii', 'ignore').decode('gbk','ignore')) #字符串要先encode成字节流才能decode

所以我把字符串重新交给BeautifulSoup解析，见下方代码

def getCourse():

course\_url ='http://zhjw.scu.edu.cn/xkAction.do?actionType=6'

course\_html =urllib.request.urlopen(url=course\_url, timeout=10).read()

course\_table =SoupStrainer(id='user')#

course\_tag =BeautifulSoup(course\_html, 'lxml', parse\_only=course\_table)

course\_tag =BeautifulSoup(course\_tagdecode('gbk', 'ignore'), 'lxml')

print(course\_tag.text)

对整体进行decode('gbk','ignore')却还是会报错

105。with...as...是对try...except...fianlly的简化

有一些任务，可能事先需要设置，事后做清理工作。对于这种场景，Python的with语句提供了一种非常方便的处理方式。一个很好的例子是文件处理，你需要获取一个文件句柄，从文件中读取数据，然后关闭文件句柄。这里有两个问题。一是可能忘记关闭文件句柄；二是文件读取数据发生异常，没有进行任何处理。

try:

f = open('xxx')

do something

except:

do something

finally:

f.close()

其实我个人不止一次在网上看到有这么写的了，这个是错的。

首先正确的如下：

try:

f = open('xxx')

except:

print 'fail to open'

exit(-1)

try:

do something

except:

do something

finally:

f.close() #文件打开就需要关闭

class controlled\_execution: #我们不能给object()传参，这就功能有限了。通常应该是在\_\_init\_\_里面设置好要用的变量

def \_\_enter\_\_(self):

set things up

return thing

def \_\_exit\_\_(self, type, value, traceback):

tear things down

with controlled\_execution() as thing:

do something

在这里，python使用了with-as的语法。当python执行这一句时，会调用\_\_enter\_\_函数，然后把该函数return的值传给as后指定的变量。之后，python会执行下面do something的语句块。最后不论在该语句块出现了什么异常，都会在离开时执行\_\_exit\_\_。

另外，\_\_exit\_\_除了用于tear things down，还可以进行异常的监控和处理，注意后几个参数。要跳过一个异常，只需要返回该函数True即可。下面的样例代码跳过了所有的TypeError，而让其他异常正常抛出。

上文说了\_\_exit\_\_函数可以进行部分异常的处理，如果我们不在这个函数中处理异常，他会正常抛出

try:

with open( "a.txt" ) as f :

do something

except xxxError:

do something about exception

其实还有个：它比\_\_enter\_\_还先

def \_\_init\_\_(self):

print 'init'

def read\_file():

try:

f = open('sawako', 'r')

try:

print ''.join(f.readlines())

except:

print 'error occurs while reading file'

finally:

f.close() #要有打开才能有关闭

except:

print 'error occurs while reading file'

===》

def readFile():

try:

with open('mio', 'r') as f:

print ''.join(f.readlines())

except:

print 'error occurs while reading file'#这个对应上面最后一个except，而第一个except可以对这个对应也可以放到函数里面对应

with requests.Session() as s:

s.get('http://httpbin.org/cookies/set/sessioncookie/123456789')

这样就能确保 with 区块退出后会话能被关闭，即使发生了异常也一样。

with真正强大之处是它可以处理异常。可能你已经注意到Sample类的\_\_exit\_\_方法有三个参数- val, type 和 trace。 这些参数在异常处理中相当有用。我们来改一下代码，看看具体如何工作的。

#!/usr/bin/env python

class Sample:

def \_\_enter\_\_(self):

return self

def \_\_exit\_\_(self, type,

value, trace):

print "type:", type

print "value:", value

print "trace:", trace

def do\_something(self):

bar = 1/0

return bar + 10

with Sample() as sample:

sample.do\_something()

106.输入错误的账号与密码cookie还是会给的。教务处本科系统。

107.突然发现表单名字对程序表单提交没有什么作用

108.没有

>>> while a=1: #在Python中,需要表达式的地方不能出现语句。while td\_tag.find\_next('td') :可以

++ --

109.cookie文件(MozillaCookieJar)

保存

import cookielib

import urllib2

#设置保存cookie的文件，同级目录下的cookie.txt

filename = 'cookie.txt'

#声明一个MozillaCookieJar对象实例来保存cookie，之后写入文件

cookie = cookielib.MozillaCookieJar(filename)

#利用urllib2库的HTTPCookieProcessor对象来创建cookie处理器

handler = urllib2.HTTPCookieProcessor(cookie)

#通过handler来构建opener

opener = urllib2.build\_opener(handler)

#创建一个请求，原理同urllib2的urlopen

response = opener.open("http://www.baidu.com")

#保存cookie到文件

cookie.save(ignore\_discard=True, ignore\_expires=True)

cookie.save(ignore\_discard=True, ignore\_expires=True)

关于最后save方法的两个参数在此说明一下：

官方解释如下：

ignore\_discard: save even cookies set to be discarded.

ignore\_expires: save even cookies that have expiredThe file is overwritten if it already exists

由此可见，ignore\_discard的意思是即使cookies将被丢弃也将它保存下来，ignore\_expires的意思是如果在该文件中cookies已经存在，则覆盖原文件写入，在这里，我们将这两个全部设置为True。运行之后，cookies将被保存到cookie.txt

读取

cookie.load('cookie.txt', ignore\_discard=True, ignore\_expires=True)

110.提醒，表单前往别忘了urlencode，request倒是自动urlencode

>>> from urllib.parse import urlencode

>>> from urllib import urlencode

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ImportError: cannot import name 'urlencode'

111.list(xx)与[xx]的区别

>>> l2 +=list(l1)

>>> l2

[1, 2, 3, 4, 1, 2, 3, 4]

>>> l2 +=[l1]

>>> l2

[1, 2, 3, 4, 1, 2, 3, 4, [1, 2, 3, 4]]

112.报错：SyntaxError: Non-ASCII character '\xe5'。在开头几行#coding=utf-8

113.scrapy框架：

官方文档: <http://scrapy-chs.readthedocs.io/zh_CN/latest>

首先从初始 URL 开始，Scheduler 会将其交给 Downloader 进行下载，下载之后会交给 Spider 进行分析，Spider 分析出来的结果有两种：一种是需要进一步抓取的链接，例如之前分析的“下一页”的链接，这些东西会被传回 Scheduler ；另一种是需要保存的数据，它们则被送到 Item Pipeline 那里，那是对数据进行后期处理（详细分析、过滤、存储等）的地方。

scrapy基于twisted异步IO框架，downloader是多线程的，力图在爬虫里做一些计算性的操作就会影响抓取的速率。但是，由于python使用GIL（全局解释器锁，保证同时只有一个线程在使用解释器），这极大限制了并行性，在处理运算密集型程序的时候，Python的多线程效果很差，而如果开多个线程进行耗时的IO操作时，Python的多线程才能发挥出更大的作用。（因为Python在进行长时IO操作时会释放GIL）

Twisted的IO操作是单线程的，即reactor线程，Twisted Reactor线程池只是用来进行DNS解析等，进行download以及parse 等一般性操作的时候，都是单线程的——都是在同一个线程内。

配置文件可以指定 线程数量 pipeline类的执行顺序(通过不同的数字表不同的优先级)

蜘蛛中间件(Spider Middlewares)，介于Scrapy引擎和蜘蛛之间的钩子框架，主要工作是处理蜘蛛的响应输入和请求输出。

参数设置可以参考： <https://www.jianshu.com/p/df9c0d1e9087>

通过ps -xH | grep scrapy 可以查看线程数量，如果使用Downloader来下载内容一般8-9个。

Downloader Middleware 处理request、response

CookiesMiddleware，此中间件使用cookiejar追踪session（会话维持），在setting.py中COOKIES\_ENABLED设置开关。cookjar是非黏性的，需要自己传递到request。默认情况下使用一个cookiejar,但是当单个spider多个request的时候，因为返回的每个response要求下一个request带的cookie都不同，所以每一次都要手动给每个request添加cookiejar来记录，并且在之后的使用过程中自己维护好cookiejar， <https://doc.scrapy.org/en/latest/topics/downloader-middleware.html#std:reqmeta-cookiejar>

RedirectMiddleware、HttpProxyMiddleware

调度中间件(Scheduler Middlewares)，介于Scrapy引擎和调度之间的中间件，从Scrapy引擎发送到调度的请求和响应。会过滤掉不在allowed\_domains的requests。

引擎(Scrapy Engine)，用来处理整个系统的数据流处理，触发事务。

调度器(Scheduler)，用来接受引擎发过来的请求，压入队列中，并在引擎再次请求的时候返回。

项目管道(Item Pipeline)，负责处理从蜘蛛从网页中抽取的项目（比如用于下载什么东西），他的主要任务是清晰、验证和存储数据。当页面被蜘蛛解析后，将被发送到项目管道，并经过几个特定的次序处理数据。

scrapy里面有意思的就是通过回调函数一串串执行下来。

一般是通过XPATH来选择合适的节点, Scrapy选择器构建于 [lxml](http://lxml.de/) 库之上，这意味着它们在速度和解析准确性上非常相似。

linux：

yum install python34-devel.x86\_64(py2就python-dev，py3自己tab两下找名字)

pip install scrapy会自动安装相关的依赖库

windows：

需要提前安装win32py（如果出现Python versin2.7 required， which was not found in the registry。就说明注册表中没有。被安装到D:\Program Files (x86)\Python3\Lib\site-packages\）并使用pip安装好lxml（html解析库）、setuptools、zope.interface、Twisted（基于事件驱动的网络引擎框架）、pyOpenSSL。最后使用pip安装scrapy。如果不能使用scrapy命令，需要将pip路径添加到环境变量中py2（\Tools\Scripts）、py3（\Scripts）。

在开始爬取之前，您必须创建一个新的Scrapy项目。 进入您打算存储代码的目录中，运行下列命令:

scrapy startproject tutorial



scrapy.cfg: 项目的配置文件

xxxx/items.py: 项目中的item文件.

xxxx/pipelines.py: 项目中的pipelines文件.

xxxx/settings.py: 项目的设置文件.

xxxx/spiders/: 放置spider代码的目录.

scrapy genspider -l #查看支持的爬虫模板

scrapy genspider (-t ModelName) SpiderName startURL #创建一个爬虫py

一般推荐使用-t crawl（默认则是-t basic，如果需要通过代码来产生start\_url只能basic，比如for循环来改变url中的index数字），CrawlSpider是Spider的派生类，Spider类的设计原则是只爬取start\_url列表中的网页，而CrawlSpider类定义了一些规则(rule)来提供跟进link的方便的机制，从爬取的网页中获取link并继续爬取的工作更适合。

创建之后会在xxx/spiders下多出一个py文件



将创建好的scrapy项目添加到pycharm中：<https://blog.csdn.net/wangsidadehao/article/details/52911746>

如果写代码的时候有些不确定就进scrapy shell <http://bizhi.feihuo.com/wallpaper/share?rsid=10010> 方便随时查看执行效果



extract返回的一定是一个列表

默认爬虫遵守网站的robots.txt，应该setting.py中ROBOTSTXT\_OBEY = False

Item：

Item 是保存爬取到的数据的容器；其使用方法和python字典类似， 并且提供了额外保护机制来避免拼写错误导致的未定义字段错误。

类似在ORM中做的一样，您可以通过创建一个 scrapy.Item 类， 并且定义类型为 scrapy.Field 的类属性来定义一个Item。

我们需要从dmoz中获取名字，url，以及网站的描述。 对此，在item中定义相应的字段编辑 tutorial 目录中的 items.py 文件:

import scrapy

class DmozItem(scrapy.Item):

title = scrapy.Field()

link = scrapy.Field()

desc = scrapy.Field()

提取Item：

Selectors选择器简介

xpath(): 传入xpath表达式，返回该表达式所对应的所有节点的selector list列表 。

css(): 传入CSS表达式，返回该表达式所对应的所有节点的selector list列表.

extract(): 序列化该节点为unicode字符串并返回list。

re(): 根据传入的正则表达式对数据进行提取，返回unicode字符串list列表。

XPATH： F12选中节点右键可以直接获取

response.xpath('//span[@class="author vcard"] 作为使用了多个class的节点，不能通过一次css选择来实现，但是我没找到css选择中或的写法

response.css('h1.entry-title > a').attrib['href'] 《=》

response.css('h1.entry-title').xpath("a").attrib['href'] #response.css('h1.entry-title').xpath("a") 是一个多元素字典，但是直接attrib['href']默认会使用第一个元素来返回，而不是返回一个列表

//video[1] /html/body/div/div/video[1] 虽然有一个[1]但是在scrapy中获取到的还是SelectorList 本来指定last()应该就是最后一个元素,我觉得这样得到的类型不应该是SelectorList而直接是Selector

/html/head/title: 选择HTML文档中 <head> 标签内的 <title> 元素

/html/head/title/text(): 选择上面提到的 <title> 元素的文字

//td: 选择所有的 <td> 元素

//div[@class="mine"]: 选择所有具有 class="mine" 属性的 div 元素 ,方括号中可以使用and & 以及 or | 比如//\*[@id="secondary" | //\*[@id="colophon"]

firefox里面有XPATH的插件，帮助分析网页

可以配合正则：

In [5]: response.xpath('//title/text()').re('(\w+):')

Out[5]: [u'Computers', u'Programming', u'Languages', u'Python']

#结合for循环

def parse\_category(self, response):

links = response.xpath('//td[descendant::a[contains(@href, "#pagerank")]]/following-sibling::td/font')

#links是一批'<a xxx>xxx</a>'

for link in links:

item = DirectoryItem()

item['name'] = link.xpath('a/text()').extract()

item['url'] = link.xpath('a/@href').extract()

item['description'] = link.xpath('font[2]/text()').extract()

yield item

def parse(self, response):

for sel in response.xpath('//ul/li'):

item = DmozItem()

item['title'] = sel.xpath('a/text()').extract()

item['link'] = sel.xpath('a/@href').extract()

item['desc'] = sel.xpath('text()').extract() #extract()就是提取节点内容，不使用这个函数，print出来的东西是残缺的，只能代表这个节点

yield item

response.text 如果html在标签的属性中使用了单引号，那么会被自动转换为双引号

response.xpath('//\*[@id="secondary"]')[0].extract() 使用了extract()函数，单引号被转义之后存在字符串里面。除了这种写法，还有一种[0]在extract()后面，但是这样意味着extract()了整个列表之后再去除0号元素

最后为了实现替换掉爬取网页自带的一些网站特征：

remove\_nodes = response.xpath('//div[@id="secondary"] | //footer[@id="colophon"]')

result = response.xpath("/html")[0].extract().replace(remove\_nodes[0].extract(),"")

result = result.replace(remove\_nodes[1].extract(),"")

Spider：

为了创建一个Spider，您必须继承 scrapy.Spider 类，其包含了一个用于下载的初始URL，如何跟进网页中的链接以及如何分析页面中的内容， 提取生成 item 的方法，且定义以下三个属性:

name: 用于区别Spider。 该名字必须是唯一的，您不可以为不同的Spider设定相同的名字。

start\_urls: 包含了Spider在启动时进行爬取的url列表。 因此，第一个被获取到的页面将是其中之一。 后续的URL则从初始的URL获取到的数据中提取。

parse() 是spider的一个方法。 被调用时，每个初始URL完成下载后生成的 Response 对象将会作为唯一的参数传递给该函数。 该方法负责解析返回的数据(response data)，提取数据(生成item)以及生成需要进一步处理的URL的 Request 对象。

import scrapy

class DmozSpider(scrapy.Spider):

name = "dmoz"

allowed\_domains = ["dmoz.org"]

start\_urls = [

"http://www.dmoz.org/Computers/Programming/Languages/Python/Books/",

"http://www.dmoz.org/Computers/Programming/Languages/Python/Resources/"

]

def parse(self, response): #parse是默认实现的解析器

filename = response.url.split("/")[-2]

with open(filename, 'wb') as f:

f.write(response.body)

#如果要继续爬下去就使用scrapy.Request("http://wooyun.yaols.com/"+img\_path[3:], callback=self.parse\_img)

#Request(url,meta = {'item':item},callback = self.parse\_href) 传递数据Request.meta['item']

#Request (url[, callback, method='GET', headers, body, cookies, meta, ...

#使用需要加上return或者yield，通过返回，Request对象才能加入队列

pipeline:

一、urllib.request.urlopen:

(1)阻塞线程 慎重使用 (2)写入操作只有在将整个response接受完成才会进行

with open(image\_path, 'wb') as image:  
 conn = urllib.request.urlopen(item['image']) # 下载图片  
 image.write(conn.read())  
 image.close()

二、通过scrapy.Request:

这种方式使用Download去下载文件，所以settings.py要注意一下Download类参数，比如下载大文件，一旦默认超过180.0会超时结束当前这个下载线程。

下面的代码实现了在pipeline中发出scrapy.Request请求，并将请求的回调函数设置为当前文件中的某一个函数

scrapy.Request("http://wooyun.yaols.com/"+img\_path[3:], callback=self.parse\_img,dont\_filter=True)

def parse\_img(self, response):

file\_path =r"C:\Users\Battery\Desktop\Program\scrapy\scrapyTest\wooyun\images\\"+response.url[31:]

with open(file\_path, "wb") as jpg:

jpg.write(response.body)

jpg.close

# -\*- coding: utf-8 -\*-  
  
# 网速能够多线程跑起来  
import os  
import scrapy  
  
class DynamicdesktopPipeline(object):  
 file\_id = 1  
  
 def \_\_init\_\_(self, crawler):  
 self.crawler = crawler  
  
 @classmethod  
 def from\_crawler(cls, crawler):  
 return cls(crawler)  
  
 def process\_item(self, item, spider):  
 store\_dir = "/media/vega/0D6C051A0D6C051A/all/Program/py/scrapy/DynamicDesktop/result"  
  
 if item['video'] is not None and item['image'] is not None:  
 image\_path = '%s/%d.jpg' % (store\_dir, self.file\_id)  
 video\_path = '%s/%d.mp4' % (store\_dir, self.file\_id)  
  
 if os.path.exists(video\_path):  
 print("there already exists %s" % video\_path )  
 else:  
 self.crawler.engine.crawl(  
 scrapy.Request(  
 url=item['image'],  
 callback=lambda response: self.store(response, image\_path)  
 ),  
 spider  
 )  
  
 self.file\_id += 1  
 return item  
  
 def store(self, response, path):  
 with open(path, 'wb') as file:  
 file.write(response.body)  
 file.close()

这里面因为使用了lambda，所以在请求的时候参数就能够被记住了，但是注意start\_parse函数Download网页产生Item的那部分也要进行处理

按照道理来说传过去的index应该是不同数字的，但是我后来发现传过去的index指向的变量是同一个，在其他地方倒是都正常使用：

for index in range(10010, 10016):#11886  
 yield scrapy.Request(url=self.base\_url.format(index=index), callback=lambda response: self.parse(response,index))

即便是下面这种代码：

首先传给一个临时变量，我认为这个临时变量属于不停创建，于是lambda就会记录下来不同的临时变量，最后执行scrapy spider，默认以json打印item的时候发现变量还是同一个。

def start\_requests(self):  
 for index in range(10010, 10016):#11886  
 test= copy.deepcopy(index) #copy.copy只是拷贝了一层  
 yield scrapy.Request(url=self.base\_url.format(index=index), callback=lambda response: self.parse(response,test))  
 # 这里index传过去，但是只是最后的index,传递过去的是变量的引用  
  
def parse(self, response, index):  
 video\_label = response.xpath('//video')[0]  
 item = DynamicdesktopItem()  
 item['index'] = id(index) # response.url[-6:-1]  
 item['video'] = video\_label.attrib['src']  
 item['image'] = video\_label.attrib['poster']  
 yield item

三、 还有就是通过官方提供的FilesPipeline或ImagesPipeline来实现下载，估计和我一样用的是scrapy.Requst:

https://github.com/BruceDone/scrapy\_demo

四、download函数（相当于二里面直接调用函数）





爬取

进入项目的根目录，执行下列命令启动spider:

scrapy crawl dmoz -o items.json

Scrapy为Spider的 start\_urls 属性中的每个URL创建了 scrapy.Request 对象，并将 parse 方法作为回调函数(callback)赋值给了Request。

Request对象经过调度，执行生成 scrapy.http.Response 对象并送回给spider parse() 方法。

scrapy crawl spidername开始运行，程序自动使用start\_urls构造Request并发送请求，然后调用parse函数对其进行解析，在这个解析过程中使用rules中的规则从html（或xml）文本中提取匹配的链接，通过这个链接再次生成Request，如此不断循环，直到返回的文本中再也没有匹配的链接，或调度器中的Request对象用尽，程序才停止。

对CrawlSpider的使用：

这个Spider有点特殊，内部实现了parse，用来解析start\_urls上的链接并按照设置的rules匹配并作出反应

我们能解析的只有从这个start\_url延伸出来的网页，除非你重写parse（Scrapy 1.3.3后需要重写\_parse\_response）

import scrapy

import urllib

from scrapyTest.items import HubItem,AuthorityItem

from scrapy.spiders import Rule,CrawlSpider

from scrapy.linkextractors import LinkExtractor

class ExampleSpider(CrawlSpider):

name = 'wooyun' #Spdier的名字

allowed\_domains = ['yaols.com']

start\_urls = ['http://wooyun.yaols.com/']#不能访问

rules = (

Rule(LinkExtractor(allow=(r'http://wooyun\.yaols\.com/bugs/wooyun.+')), callback="parse\_authority"),

Rule(LinkExtractor(allow=(r'http://www\.wooyun\.org/.+')), follow=False),

)

def parse(self, response): #重写并加入start\_url

for request\_or\_item in CrawlSpider.parse(self, response):

if isinstance(request\_or\_item, Request):

request\_or\_item = request\_or\_item.replace(meta = {'start\_url': response.meta['start\_url']})

yield request\_or\_item

Python的SGMLParser实在是太慢了，使用SgmlLinkExtractor会让爬虫把大部分的时间都浪费在解析网页上,现在是使用LinkExtractor，最好自己写一个link extractor(我们基于 lxml 写了一个，也可以用soup之类的库)。也可以用正则表达式来写link extractor，速度快，问题是不理解html语义，会把注释里的链接也包含进来。另外基于JavaScript重定向url也要在这里提取出来。

在setting.py中将pipeline加入到本项目：

ITEM\_PIPELINES = { 数字小表示优先级高

'myproject.pipelines.PricePipeline': 300,

'myproject.pipelines.JsonWriterPipeline': 800,

}

from scrapy.exceptions import DropItem

class DuplicatesPipeline(object): 这个pipeline就是处理数据冲突的

def \_\_init\_\_(self):

self.ids\_seen = set()

def process\_item(self, item, spider):

if item['id'] in self.ids\_seen:

raise DropItem("Duplicate item found: %s" % item)

else:

self.ids\_seen.add(item['id'])

return item

使用方法：

if item['id'] :

if isinstance(item, Aitem):

scrapy crawl SpiderName 之后，控制台默认就会输出item的json形式的数据，我们保存的数据就是这些。

保存爬的数据：Feed exports

实现爬虫时最经常提到的需求就是能合适的保存爬取到的数据，或者说，生成一个带有爬取数据的”输出文件”(通常叫做”输出feed”)，来供其他系统使用。

feed输出使用到了 Item exporters 。其自带支持的类型有:

JSON 使用的exporter: JsonItemExporter

JSON lines 使用的exporter: JsonLinesItemExporter

CSV 使用的exporter: CsvItemExporter

XML 使用的exporter: XmlItemExporter

...还可以自己调用Item exporters来输出

使用feed输出时您可以通过使用 URI (通过 FEED\_URI 设置) 来定义存储端。 feed输出支持URI方式支持的多种存储后端类型。

自带支持的存储后端有:

本地文件系统

FTP

S3 (需要 boto)（使用 Amazon 的云数据迁移选项，客户可以轻松地将大量数据移入或移出 Amazon S3。数据在存储到 S3 中之后，会自动采用成本更低、存储期限更长的云存储类）

标准输出

1. call the method start\_exporting() in order to signal the beginning of the exporting process

2. call the export\_item() method for each item you want to export

3. and finally call the finish\_exporting() to signal the end of the exporting process

from scrapy import signals

from scrapy.exporters import XmlItemExporter

class XmlExportPipeline(object):

def \_\_init\_\_(self):

self.files = {}

@classmethod

def from\_crawler(cls, crawler):

pipeline = cls()

crawler.signals.connect(pipeline.spider\_opened, signals.spider\_opened)

crawler.signals.connect(pipeline.spider\_closed, signals.spider\_closed)

return pipeline

def spider\_opened(self, spider):

file = open('%s\_products.xml' % spider.name, 'w+b')

self.files[spider] = file

self.exporter = XmlItemExporter(file)

self.exporter.start\_exporting()

def spider\_closed(self, spider):

self.exporter.finish\_exporting()

file = self.files.pop(spider)

file.close()

def process\_item(self, item, spider):

self.exporter.export\_item(item)

return item

scrapy.contrib.exporter.JsonItemExporter 的时候额外指定 ensure\_ascii=False，那么输出就是Unicode编码

存储URI参数

存储URI也包含参数。当feed被创建时这些参数可以被覆盖:

%(time)s - 当feed被创建时被timestamp覆盖

%(name)s - 被spider的名字覆盖

其他命名的参数会被spider同名的属性所覆盖。例如， 当feed被创建时， %(site\_id)s 将会被 spider.site\_id 属性所覆盖。

存储在FTP，每个spider一个目录:

ftp://user:password@ftp.example.com/scraping/feeds/%(name)s/%(time)s.json

存储在S3，每一个spider一个目录:

s3://mybucket/scraping/feeds/%(name)s/%(time)s.json

您可以指定一个绝对路径 /tmp/export.csv 并忽略协议(scheme)。不过这仅仅只能在Unix系统中工作。

FTP

将feed存储在FTP服务器。

URI scheme: ftp

URI样例: ftp://user:pass@ftp.example.com/path/to/export.csv

S3：

将feed存储在 Amazon S3 。

URI scheme: s3

URI样例:

s3://mybucket/path/to/export.csv

s3://aws\_key:aws\_secret@mybucket/path/to/export.csv

需要的外部依赖库: boto

标准输出：

feed输出到Scrapy进程的标准输出。

URI scheme: stdout

URI样例: stdout:

设定(Settings)

这些是配置feed输出的设定:

FEED\_URI (必须)

FEED\_FORMAT

FEED\_STORAGES #包含项目支持的额外feed存储端的字典。 字典的键(key)是URI协议(scheme)，值是存储类(storage class)的路径。

FEED\_EXPORTERS #包含项目支持的额外输出器(exporter)的字典。 该字典的键(key)是URI协议(scheme)，值是 Item输出器(exporter) 类的路径。

FEED\_STORE\_EMPTY #是否输出空feed(没有item的feed)。

FEED\_EXPORTERS\_BASE #包含Scrapy内置支持的feed输出器(exporter)的字典。

在Shell中尝试Selector选择器

为了介绍Selector的使用方法，接下来我们将要使用内置的 Scrapy shell 。Scrapy Shell需要您预装好IPython(一个扩展的Python终端)。

您需要进入项目的根目录，执行下列命令来启动shell:

scrapy shell "http://www.dmoz.org/Computers/Programming/Languages/Python/Books/"

更为重的是，当输入 response.selector 时， 您将获取到一个可以用于查询返回数据的selector(选择器)， 以及映射到 response.selector.xpath() 、 response.selector.css() 的 快捷方法(shortcut): response.xpath() 和 response.css() 。

同时，shell根据response提前初始化了变量 sel 。该selector根据response的类型自动选择最合适的分析规则(XML vs HTML)。

使用Scrapy时出现虽然队列里有很多Request但是却不下载，造成假死状态。http://twistedmatrix.com/trac/attachment/ticket/5773/patch1.diff#L739 timeout不应该设置为0

113.5.Pyspiders

国内一个大神开发具有以下特性：

1. python 脚本控制，可以用任何你喜欢的html解析包（内置 pyquery）

2. WEB 界面编写调试脚本，起停脚本，监控执行状态，查看活动历史，获取结果产出

3. 支持 MySQL, MongoDB, SQLite

4. 支持抓取 JavaScript 的页面

5. 组件可替换，支持单机/分布式部署，支持 Docker 部署

6. 强大的调度控制

114.py的界面Qtpython

115.py3推荐使用pymysql

pip install pymysql

连接数据库如下：

import pymysql.cursors

# Connect to the database

connection = pymysql.connect(host='127.0.0.1',

port=3306,

user='root',

password='zhyea.com',

db='employees',

charset='utf8mb4',

cursorclass=pymysql.cursors.DictCursor)

但是跟好看：

import pymysql.cursors

config = {

'host':'127.0.0.1',

'port':3306,

'user':'root',

'password':'zhyea.com',

'db':'employees',

'charset':'utf8mb4',

'cursorclass':pymysql.cursors.DictCursor #默认放在元组里大元组包括小元组

#而这个是列表包括字典

}

connection = pymysql.connect(\*\*config)

插入数据：

执行sql语句前需要获取cursor，因为配置默认自动提交，故在执行sql语句后需要主动commit，最后不要忘记关闭连接：

# 执行sql语句

try:

with connection.cursor() as cursor:

# 执行sql语句，插入记录

sql = 'INSERT INTO employees (first\_name, last\_name, hire\_date, gender, birth\_date) VALUES (%s, %s, %s, %s, %s)'

cursor.execute(sql, ('Robin', 'Zhyea', tomorrow, 'M', date(1989, 6, 14)));

# 没有设置默认自动提交，需要主动提交，以保存所执行的语句

connection.commit() #我试过了，如果没有这句话就不会写入kali Mariadb，不过服务器上似乎是自动提交

finally:

connection.close();

执行查询：

try:

with connection.cursor() as cursor:

# 执行sql语句，进行查询

sql = 'SELECT first\_name, last\_name, hire\_date FROM employees WHERE hire\_date BETWEEN %s AND %s'

cursor.execute(sql, (hire\_start, hire\_end))

# 获取查询结果中的一条

result = cursor.fetchone()

print(result)

# 没有设置默认自动提交，需要主动提交，以保存所执行的语句

connection.commit() #connection对象提交

finally:

connection.close();

从结果集中获取指定数目的记录，可以使用fetchmany方法：

result = cursor.fetchmany(2)

不过不建议这样使用，最好在sql语句中设置查询的记录总数。

获取全部结果集可以使用fetchall方法：

result = cursor.fetchall()

只要第一条就fetchone()

fetchone()返回的是个字典，字段名为key。如果字段没有值，那么打印输出就会是None。

获取最新的id:

new\_id = cursor.lastrowid

print new\_id

scroll

cursor.scroll(-1,mode='relative') # 相对当前位置移动 (备注：参数：1是向下，-1是向上）

cursor.scroll(2,mode='absolute') # 相对绝对位置移动

#获取执行完存储的参数,参数@开头 #p1是存储方法

cursor.execute("select @p1,@\_p1\_1,@\_p1\_2,@\_p1\_3") #{u'@\_p1\_1': 22, u'@p1': None, u'@\_p1\_2': 103, u'@\_p1\_3': 24}

row\_1 = cursor.fetchone()

被注入：

sql="select user,pass from tb7 where user='%s' and pass='%s'" % (user,passwd)

#拼接语句被构造成下面这样，永真条件，此时就注入成功了。因此要避免这种情况需使用pymysql提供的参数化查询。

#select user,pass from tb7 where user='u1' or '1'-- ' and pass='u1pass'

row\_count=cursor.execute(sql)

避免注入，使用pymysql提供的参数化语句：

row\_count=cursor.execute("select user,pass from tb7 where user=%s and pass=%s",(user,passwd)) #因为中间是逗号所以user,passwd就是以字符串形式传进

cursor.execute('%d %s'%(2), (2))

使用存mysql储过程动态执行SQL防注入

mysql="select \* from tb7 where nid>? and nid<?"

cursor.callproc('proc\_sql', args=(11, 15, mysql))

#定义上下文管理器，连接后自动关闭连接

@contextlib.contextmanager

def mysql(host='127.0.0.1', port=3306, user='root', passwd='', db='tkq1',charset='utf8'):

conn = pymysql.connect(host=host, port=port, user=user, passwd=passwd, db=db, charset=charset)

cursor = conn.cursor(cursor=pymysql.cursors.DictCursor)

try:

yield cursor

finally:

conn.commit()

cursor.close()

conn.close()

# 执行sql

with mysql() as cursor:

print(cursor)

row\_count = cursor.execute("select \* from tb7")

row\_1 = cursor.fetchone()

print row\_count, row\_1

connection.rollback()

该方法回滚自上一次调用 commit() 以来对数据库所做的更改。

insert into test(name,birthday) values(%s,%d)"%(line[0],int(line[1]))

117.验证码识别：

安装Tesseractsudo apt-get install tesseract-ocr

安装中文sudo apt-get install tesseract-ocr-chi-sim

windows使用exe安装

安装google引擎tesseract-ocr，在pip install pillow(py3中取代了PIL，也就是Python Image Library) 和pytesseract(旧版本才是tesseract-ocr库)

Python-tesseract默认支持tiff、bmp格式图片,只有在安装PIL之后,才能支持jpeg、gif、png等其他图片格式

tesseract 是OCR 技术，其实也就是通过某些机器学习算法，实现了一种对图片进行分类的技术，大致就是先对算法输入训练数据集，人为的告诉电脑，这张图片是1，那张是2。 一旦没有匹配就会没有任何输出。当然，没处理好的话会匹配出多余的字符。

原来0打出来，可能中间有个小点。体育学院的验证码就是这样。

之所以可以浏览器回退使用原来的验证码进行提交是因为有效期还在（在没有请求新的验证码前），验证码与sessionid绑定。

简单验证码识别：

import pytesseract

from PIL import Image

image = Image.open('vcode.png')

vcode = pytesseract.image\_to\_string(image)

print (vcode)

可是错误率高。

import Image

from numpy import \* #array #number py

import pytesseract

im = Image.open('1.png')

im = im.convert('RGB')

im = im.resize((200,80))

a = array(im) #返回的是numby中的array

for i in xrange(len(a)): #在Python 3中，range()的实现方式与xrange()函数相同，所以就不存在专用的xrange( )

for j in xrange(len(a[i])):

if a[i][j][0] == 255:

a[i][j]=[0,0,0] #??????RGB??

else:

a[i][j]=[255,255,255]

im = Image.fromarray(a)

im.show() #这个好，用来看图片纠正效果，记得resize自己调一下方便自己查看

vcode = pytesseract.image\_to\_string(im)

print vcode

后来发现看起来同一个颜色的地方各处RGB颜色值不一样

用len不一定通用，这个好点：

for i in range(im.size[1]): #i是行数

temp=[]

for j in range(im.size[0]): #j是列数

pos=(j,i) #注意这里的size[0]与[1]

col=im.getpixel(pos)

#获取某坐标的点的颜色，黑色为0，白色为255，为了显示规程，把它转变成1了

if col==255:

col=1

temp.append(col)

print(temp)

如果是RGB返回元组

>>> image.getpixel((0,0))

(250, 250, 250)

上下左右去掉同色的点：

for i in range(im.size[1]):

for j in range(im.size[0]):

#pos=(j,i)

#print(pos)

if i==0 or j==0 or i==im.size[1]-1 or j==im.size[0]-1:

im2.putpixel((j,i),255)#图片最外面一周都换成白点

elif im.getpixel((j,i))!=im.getpixel((j-1,i)) and im.getpixel((j,i))!=im.getpixel((j+1,i)) and im.getpixel((j,i-1))!=im.getpixel((j,i)) and im.getpixel((j,i+1))!=im.getpixel((j,i)):

im2.putpixel((j,i),255)#不连续点也都换成白点

#/usr/bin/python

#coding=utf-8

# pretreat\_image.py

# 注意:二值图像0/1统一转化为0/255

from PIL import Image,ImageDraw,ImageChops

# 验证码预处理,主要是降噪

# 预处理结束后返回0/255二值图像

# 降噪,参考 http://blog.csdn.net/xinghun\_4/article/details/47864949

def pretreat\_image(image):

# 将图片转换成灰度图片

image = image.convert("L")

# 二值化,得到0/255二值图片

# 阀值threshold = 180

image = iamge2imbw(image,180)

# 对二值图片进行降噪

# N = 4

clear\_noise(image,4)

# 去除外边框

# 原图大小:122\*54

# 左上右下,左 <= x < 右

box = ( 5, 5, 95, 30 )

image = image.crop(box)

return image

# 灰度图像二值化,返回0/255二值图像

def iamge2imbw(image,threshold):

# 设置二值化阀值

table = []

for i in range(256):

if i < threshold:

table.append(0)

else:

table.append(1)

#根据table来映射为二值图像。

image = image.point(table,'1')

# The RGB color table actually is like this

# [0, 1, 2, 3, 4, 5, ...255, 0, 1, 2, 3, ....255, 0, 1, 2, 3, ...255]

#

# 使用color table进行颜色反转

# im = im.point(range(256, 0, -1) \* 3)

# 像素值变为0,255

# 模式“1”为二值图像，非黑即白。但是它每个像素用8个bit表示，0表示黑，255表示白。

# 模式“L”为灰色图像，它的每个像素用8个bit表示，0表示黑，255表示白，其他数字表示不同的灰度。在PIL中，从模式“RGB”转换为“L”模式是按照下面的公式转换的

# 模式“P”为8位彩色图像，它的每个像素用8个bit表示，其对应的彩色值是按照调色板查询出来的。

# 模式“RGBA”为32位彩色图像，它的每个像素用32个bit表示，其中24bit表示红色、绿色和蓝色三个通道，另外8bit表示alpha通道，即透明通道

# 还有32位浮点灰度和32位整型灰度图像

# .mode可以查看是什么模式

image = image.convert('L')

return image

# 缓解边界混乱的问题

# 根据一个点A的灰度值(0/255值),与周围的8个点的值比较

# 降噪率N: N=1,2,3,4,5,6,7

# 当A的值与周围8个点的相等数小于N时,此点为噪点

# 如果确认是噪声,用该点的上面一个点的值进行替换

def get\_near\_pixel(image,x,y,N):

pix = image.getpixel((x,y))

near\_dots = 0

if pix == image.getpixel((x - 1,y - 1)):

near\_dots += 1

if pix == image.getpixel((x - 1,y)):

near\_dots += 1

if pix == image.getpixel((x - 1,y + 1)):

near\_dots += 1

if pix == image.getpixel((x,y - 1)):

near\_dots += 1

if pix == image.getpixel((x,y + 1)):

near\_dots += 1

if pix == image.getpixel((x + 1,y - 1)):

near\_dots += 1

if pix == image.getpixel((x + 1,y)):

near\_dots += 1

if pix == image.getpixel((x + 1,y + 1)):

near\_dots += 1

if near\_dots < N:

# 确定是噪声,用上面一个点的值代替

return image.getpixel((x,y-1))

else:

return None

# 降噪处理

def clear\_noise(image,N):

draw = ImageDraw.Draw(image)

# 外面一层半圈像素变白色

Width,Height=image.size

for x in range(Width):

draw.point((x,0),255)

draw.point((x,Height-1),255)

for y in range(Height):

draw.point((0,y),255)

draw.point((Width-1,y),255)

# 继续绘制其余地方同时进行降噪。

for x in range(1,Width - 1):

for y in range(1,Height - 1): # 因为get\_near\_pixel总是去上面一层像素的点

color = get\_near\_pixel(image,x,y,N)

if color != None:

draw.point((x,y),color)

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

image = Image.open('vimage.jpeg')

image = pretreat\_image(image)

image.show() #用窗口显示图片

from PIL import Image

from numpy import array,uint8

vimage1 =Image.open('8.bmp') #数字8

vimage2 =Image.open('B.bmp') #字母B

vimage1 = vimage1.convert("1")

vimage2 = vimage2.convert("1")

all\_list=to01List(array(vimage1))

tmp\_list =to01List(array(vimage2))

print(all\_list==tmp\_list) #True

vimage3 =Image.fromarray(uint8(vimage1))

vimage3 =vimage3.resize((25,20))

vimage3.show() #什么都没有

直接进行二值转换，会导致一些验证码识别错误，虽然我人看起来它们区分很大

补充说明，对同一张图片进行array还可能会得到不同的结果

直接对RGB图片进行array()以后，那么每一行就是

[[250 250 250]

[ 0 0 0]

[ 0 0 0]

[ 0 0 0]

[ 0 0 0]

[ 0 0 0]

[ 0 0 0]

[250 250 250]

[250 250 250]

[250 250 250]]

>>> image.getpixel((0,0)) #这个函数也可以用来取颜色

(250, 250, 250)

array(Image对象.convert('1'))，并不是对每一点01数值的提取，而且得到里面是个True。Image.fromarray(uint8(vimage1))返回去生成一下你就知道了

所以我直接对RGB进行array()，然后自己根据RGB的值生成验证码图片的相似识别列表

如果要识别视频流里面的字幕等，需要先将视频转换为一帧帧的图像

117.5 语音识别 Google Speech API

118.

>>> time.ctime()

'Wed Mar 8 16:03:43 2017'

119.代理：

http://www.xicidaili.com/

http://www.xicidaili.com/nn/ #高匿http，连jwc大创系统会出无法连接或超时

http://www.xicidaili.com/nn/ #https，存活时间很长

http://www.xicidaili.com/2017-03-09/sichuan

连jwc大创系统会出无法连接或超时：

Connection reset by peer

Cannot connect to proxy

Max retries exceeded with url

记得设置本机这边

s.keep\_alive=Flase //这个不是自带的

requests.adapters.DEFAULT\_RETRIES = 5

>>> s.headers

{'User-Agent': 'python-requests/2.13.0', 'Connection': 'keep-alive', 'Accept-Encoding': 'gzip, deflate', 'Accept': '\*/\*'}

#http\_proxy.py

#coding=utf-8

#每天自动获取ip并保存到响应文件夹

import requests

import re

import random

#首次调用会自动更新ip库

# 更新ip库

def updateIp():

headers ={"User-Agent":"Mozilla/5.0 (Windows NT 10.0; WOW64; rv:45.0) Gecko/20100101 Firefox/45.0"}

s=requests.Session()

s.keep\_alive=False #之前连定义都没有

list=[]

url="http://www.xicidaili.com/nn/"

for i in range(1,4):

url+=str(i)

#url='http://www.xicidaili.com/2017-03-09/sichuan'

html = s.get(url=url, headers=headers).text

re1 =re.compile(r'((\d{1,3}\.){3}(\d{1,3})</td>(\s){1,}<td>\d{1,5})</td>[\s\S]\*?width:\d{2}')

for each\_ip in re.finditer(re1,html):

if( each\_ip.group()[-2:]>'90' ):

list.append(each\_ip.group(1).replace("</td>\n <td>",":"))

return list

http\_proxy\_list = updateIp()

def getHttpProxy():

return random.choice(http\_proxy\_list)

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

print(getHttpProxy())

print(len(http\_proxy\_list))

120.多线程：

过时的threading似乎因为使用了GIL导致性能不行，而且py2里面解释器锁定了py对象，所以带来的性能提升不多。

#coding=utf-8

import threading

from time import ctime,sleep

def music(func):

for i in range(2):

print "I was listening to %s. %s" %(func,ctime())

sleep(1)

def move(func):

for i in range(2):

print "I was at the %s! %s" %(func,ctime())

sleep(5)

threads = []

t1 = threading.Thread(target=music,args=(xx,))

threads.append(t1)

t2 = threading.Thread(target=move,args=(xx,))

threads.append(t2)

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

for t in threads:

#t.setDaemon(True) #设为守护进程

t.start()

for t in threads:

t.join() #主线程等待

print "all over %s" %ctime()

#coding=utf-8

import threading

from time import sleep, ctime

class MyThread(threading.Thread):

def \_\_init\_\_(self,func,args,name=''):

threading.Thread.\_\_init\_\_(self)

self.name=name

self.func=func

self.args=args

def run(self):

self.func(self.args[0],self.args[1])

def super\_play(file,time):

for i in range(2):

print('Start playing? %s! %s' %(file,ctime()) )

sleep(time)

list = {'1.mp3':3,'2.mp4':5}

threads = []

files = range(len(list))

for k,v in list.items():

t = MyThread(super\_play,(k,v),super\_play.\_\_name\_\_)

threads.append(t)

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

for i in files:

threads[i].start()

for i in files:

threads[i].join()

print('end:%s' %ctime())

import threadpool

pool = ThreadPool(poolsize)

requests = makeRequests(some\_callable, list\_of\_args, callback)

[pool.putRequest(req) for req in requests]

[pool.putRequest(req) for req in requests]

[pool.putRequest(req) for req in requests]

pool.wait() #有问题，需要每次[pool.putRequest(req) for req in requests] 后wait，或者最后sleep

#否则会说是daemon（主线程退出了），无法获得stdout的lock

多进程库multiprocessing被推荐。

线程池使用ThreadPoolExecutor，ProcessPoolExecutor取代了threadpool

from concurrent.futures import ThreadPoolExecutor

import time

def return\_future\_result(message):

time.sleep(2)

return message

pool = ThreadPoolExecutor(max\_workers=2)

future1 = pool.submit(return\_future\_result, ("hello"))

future2 = pool.submit(return\_future\_result, ("world")) #有请求返回future对象（和执行的线程联系了）

print(future1.done()) #判断是正常执行完毕的。

time.sleep(3)

print(future2.done())

print(future1.result()) #如果放到sleep前面这会造成阻塞

print(future2.result())

#pool.submit(return\_future\_result, "hello",'hello')

判断是否还在运行

Future.running()

判断是否真的结束了任务。 #如此看来不同的任务就是要不同的future

Future.cancelled()

from concurrent.futures import ThreadPoolExecutor

executor = ThreadPoolExecutor(10)

def test\_function(num1, num2):

print(num1, num2) return num1 + num2

result\_iterators = executor.map(test\_function, [1, 2], [5, 5])

for result in result\_iterators:

print(result)

with concurrent.futures.ThreadPoolExecutor(max\_workers=3) as executor:

print(list(executor.map(sleeper, x)))

# concurrent.futures.as\_completed(fs, timeout=None) Returns an iterator over the Future instances (possibly created by different Executor instances) given by fs that yields futures as they complete (finished or were cancelled). Any futures given by fs that are duplicated will be returned once. Any futures that completed before as\_completed() is called will be yielded first

import concurrent.futures

import urllib.request

URLS = ['http://httpbin.org', 'http://example.com/', 'https://api.github.com/']

def load\_url(url, timeout):

with urllib.request.urlopen(url, timeout=timeout) as conn:

return conn.read()

with concurrent.futures.ThreadPoolExecutor(max\_workers=3) as executor:

future\_to\_url = {executor.submit(load\_url, url, 60): url for url in URLS}

#future\_to\_url =[executor.submit(load\_url, url, 60) for url in URLS]

for future in concurrent.futures.as\_completed(future\_to\_url):

url = future\_to\_url[future]

try:

data = future.result()

except Exception as exc:

print('%r generated an exception: %s' % (url, exc))

else:

print('%r page is %d bytes' % (url, len(data)))

#不是按照URLS顺序输出

import concurrent.futures #py2??pip install future

import urllib.request

URLS = ['http://httpbin.org', 'http://example.com/', 'https://api.github.com/']

def load\_url(url):

with urllib.request.urlopen(url, timeout=60) as conn:

return conn.read()

# We can use a with statement to ensure threads are cleaned up promptly

with concurrent.futures.ThreadPoolExecutor(max\_workers=3) as executor:

for url, data in zip(URLS, executor.map(load\_url, URLS)):

print('%r page is %d bytes' % (url, len(data)))

按照URL顺序输出

wait的使用：

from concurrent.futures import ThreadPoolExecutor, wait, as\_completed

pool = ThreadPoolExecutor(5)

futures = []

for x in range(5):

futures.append(pool.submit(return\_after\_random\_secs, x))

print(wait(futures))

无语的是单线程好好地程序直接封装成无参数函数然后作为多线程进行调用，线程却一下退出了：

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

with ThreadPoolExecutor(max\_workers=10) as executor:

futures =[]

for i in range(10):

futures.append( executor.submit(brute) )

wait(futures)

使用了多线程以后py很可能不会告诉你哪里代码出错了，需要放到单线程中进行调试，连使用未定义参数以及调用函数参数不匹配都不提醒

connection 和 cursor 都不是线程安全的。如果测试环境用多个线程，每个线程要在线程里面获取自己的 connection，然后从这个connection 获取 cursor.

孔豪：

线程库我比较喜欢用这个 threadpool

from mutliprocess.dummy import pool

121.tab与空格混合缩进使用会报错

122.py暴库：

status\_code:200、401

len(text)

tile

123.iterable才可转list

>>> list(2)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'int' object is not iterable

124.UnboundLocalError: local variable 'a' referenced before assignment

a明明是个全局变量但是在函数里面使用就会这样，所以global a于函数中一下

125.随机jwc账号get：

def getAccountList():

global account\_list

user\_code\_pri='2015141462'

for i in random.sample(range(0,334), 20):

user\_code =user\_code\_pri+str(i).rjust(3, '0')

if login(user\_code, user\_code[-6:]) > 5000:

account\_list +=[user\_code, user\_code[-6:]] #??????????????????????????????????

logout()

为了适应多线程，我想是最后set()去重还是已经去重后一个个加起来（比如说分块）

如果使用多线程，那么各个函数模块要做多线程适应（传递一些参数如session、分块参数），并且最后要放到一个大函数里面被执行，和多进程在想法上差不多了。

126.exit()会导致程序或交互模式退出

127.IndexError: list index out of range。会检查下标

128.多线程爬虫对CPU消耗不高，但是对内存消耗300MB（10线程数据读写与jwc）

129.数据库异常的抛出（比较常见的是字段太小了）会导致调到except本次抓取的后面数据全部都被丢掉。要么该字段重新抓，要么加循环不停地写入，出现异常被无视。

而不是简简单单的一个except: pass

130.py文件操作：

模式却有二进制（b来表示）与文本，当前与追加（a或者+来表示）

文件读写函数：read（输入参数代表字节数）、write （写入的内容必须要是字符串，str()转换并不会去掉list的方括号）

位置告知函数：seek（第二个参数默认是0，表示从文件开始处读取；1表示从当前位置开始计数；2表示从文件最后开始。）、telll

file.closed 如果文件被关闭返回true，否则为false

file.mode 返回文件打开访问模式

file.name 返回文件名

fileObject.close();

os提供的文件操作： 好像支持所有的系统命令

os.rename(current\_file\_name, new\_file\_name)

os.remove(file\_name)

对目录：

os.mkdir("newdir")

os.chdir("newdir")

os.getcwd() #pwd命令

os.rmdir('dirname')

file = open("sample.txt") #不能是绝对路径，第二个参数和C语言中的一样，第三个参数指定buffer大小以及缓冲方式

#buffering: 如果该缓冲值被设置为0，则表示不使用缓冲。如果该缓冲值是1，则在访问一个文件进行时行缓冲。如果指定缓冲值大于1的整数，缓冲使用所指示的缓冲器大小进行。如果是负数，缓冲区大小是系统默认的(默认行为)。

while 1:

line = file.readline()

if not line:

break

pass # do something

　　一行一行得从文件读数据，显然比较慢；不过很省内存。

　　在我的机器上读10M的sample.txt文件，每秒大约读32000行

while(line=Finput.readline()):报错

print(line)

import fileinput

for line in fileinput.input("sample.txt"):

pass

　　写法简单一些，不过测试以后发现每秒只能读13000行数据

py2会出现这个该死的编码问题：

('\xe8\xb4\xba\xe9\x9f\xb5', '110224199708290321')

你把这两个字符串加起来再用print输出就是中文了

data = "学分：1.5".decode("utf-8").encode('gb2312')

# data = u"学分：1.5".encode('gb2312')

print type(data)

# 两种解决方案

print body.split(u'：') # unicode 对应 unicode

print body.encode("utf-8").split("：") # utf-8 对应 utf-8

不过最好使用py3。我发现在py3里面对中文字符串encode('utf-8')就会出现这个，看来确实是对中文进行utf-8编码。

python 3中只有unicode str，所以把decode方法去掉了。你的代码中，f1已经是unicode str了，不用decode。

如果文件内容不是unicode编码的，要先以二进制方式打开，读入比特流，再解码。

>>> f1 = open("unicode.txt", 'r').read()

>>> print(f1)

寒冷

>>> f2 = open("unicode.txt", 'rb').read() #二进制方式打开

>>> print(f2)

b'\xe5\xaf\x92\xe5\x86\xb7\n'

>>> f2.decode()

'寒冷\n'

>>> f1.decode()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'str' object has no attribute 'decode'

0830这个数字会在py中报错，应该去掉开头的0

f =open("test",'r')

with connection.cursor() as cursor:

while 1:

line =f.readline()

if not line:

break;

line =line.split(' ')

print(line)

try:

cursor.execute("insert into test(name,birthday) values(%s,%d)"%(line[0],int(line[1])))

connection.commit()

finally:

pass

cursor.close()

connection.close()

pymysql.err.InternalError: (1054, "Unknown column '罗治港' in 'field list'")真正发现其实不是编码问题，而是%s,%d合起来语法问题，分开一个个来倒是可以。因为%方式是直接传过去，所以应该使用'%s'

在服务器上，我源文本、数据库都是utf-8，但是py3 readline报错，最后在open中增加了参数encoding='utf-8'解决。真是坑！

131.函数中使用的全局变量，在函数使用前被定义就好。在函数声明前有没有没关系。

132.windows下py的环境变量有两个：

一个是py根目录，其下有python.exe

另一个是根目录下的/scripts其下有pip.exe，如果是py3就还有pip3.exe。

133.py剪视频：

import moviepy.editr as editor

START\_TIME =60\*xxx

STOP\_TIME =60\*xxx

VIDEO =editor.VideoFileClip("FilePath").subclip(START\_TIME, STOP\_TIME);

VIDEO.write\_videofile('output.file', fps=25)

134.这是我目前见到过唯一一门不支持++这种符号的语言。

135.Django2.0开始只支持py3.0。Django 是由 Python 开发的一个免费的开源网站框架，可以用于快速搭建高性能，优雅的网站

。py守护进程需要Supervisor来维护。我们需要维护这两者。

136.pip install --upgrade pip使用这条命令来升级当前pip的版本。

137.py中相对路径用的是.而不是/

138.NumPy

NumPy（http://www.scipy.org/NumPy/）是非常有名的 Python 科学计算工具包，其中包含了大量有用的思想，比如数组对象（用来表示向量、矩阵、图像等）以及线性代数函数。

NumPy数组是一个多维数组对象，称为ndarray。一个数组对象很像一个列表（或者是列表的列表），但是数组中所有的元素必须具有相同的数据类型。除非创建数组对象时指定数据类型，否则数据类型会按照数据的类型自动确定。使用array()的第二个参数来指定，比如'f'就是float。。这个数组可以用来表示向量、矩阵、图像。

numpy中的array函数可以将Image图像数字化，而这不是list可以做到的，因为Image对象不是iterable。fromarray()就是array（）的方向操作

其由两部分组成：

实际的数据

描述这些数据的元数据

ndarray的prin函数显示就是列表，而他的比较却会比较其中的每一个元素，然后返回一个ndarray数组，每一个元素对应表示比较结果

不过通过.tolist()可以将ndarray转化为一个[[True,False,True]]这样的二维list。

[[True, True, True, True, True, True, True, True, True, True], [True, True, True, True, True, True, True, True, True, True], [False, True, True, True, True, True, True, False, False, False]]

[[ True True True True True True True True True True]

[ True True True True True True True True True True]

[False True True True True True True False False False]]

两者print出来明显ndarray的格式好看很多

#change doubel dimensions iterable (like ndarray) to list

def to01List(list\_of\_gray\_image):

return\_list =[]

for row in range(len(list\_of\_gray\_image)):

for column in range(len(list\_of\_gray\_image[row])):

if list\_of\_gray\_image[row][column]==True:

return\_list +=[1]

else:

return\_list +=[0]

return return\_list

将图像读入 NumPy 数组对象后，我们可以对它们执行任意数学操作。一个简单的例子就是图像的灰度变换。考虑任意函数 f，它将 0...255 区间（或者 0...1 区间）映射到自身（意思是说，输出区间的范围和输入区间的范围相同）。下面是关于灰度变换的一些例子：

from PIL import Image

from numpy import \*

im = array(Image.open('empire.jpg').convert('L'))

im2 = 255 - im # 对图像进行反相处理

im3 = (100.0/255) \* im + 100 # 将图像像素值变换到100...200 区间

im4 = 255.0 \* (im/255.0)\*\*2 # 对图像像素值求平方后得到的图像

139.把列表加到列表里面：

my\_list=[]

test\_list=[1,2]

my\_list +=[test\_list] #需要加上[]，不然只是把内容加进去

my\_list.append(test\_list) #这个性能会好很多，好像是因为内存分配的原因

140.

使用list()来转string就会每一个字符一个元素

'[1,2,3]'

>>> list(str)

['[', '1', ',', '2', ',', '3', ']']

>>> str = 'abcde'

>>> list = list(str)

>>> list

['a', 'b', 'c', 'd', 'e']

>>> str\_convert = ''.join(list)

>>> str\_convert

'abcde'

这种直接将list转换过去的方式，注意逗号后面有空格

>>> str([1,2])

'[1, 2]'

141.把文件中的列表形式读取过来作为一个列表

tmp\_list =[ int(line[i]) for i in range(3,301,3) ]

142.# encoding=gbk这句话可能会导致requests和pymysql的使用失败

143.Py3 中1/100答案是0，不会改变类型！还是int。

144.

>>> bool([1])

True

>>> bool([0])

True

>>> bool([0,1])

True

145.PIL库的其余使用：

使用image表示这是一个Image对象

image.thumbnail() 方法接受一个元组参数（该参数指定生成缩略图的大小） ，resize和crop都是使用元组

146.Matplotlib

我们处理数学运算、绘制图表，或者在图像上绘制点、直线和曲线时，Matplotlib 是个很好的类库，具有比 PIL 更强大的绘图功能。Matplotlib 可以绘制出高质量的图表，就像本书中的许多插图一样。Matplotlib 中的 PyLab 接口包含很多方便用户创建图像的函数。Matplotlib 是开源工具，可以从 http://matplotlib.sourceforge.net/ 免费下载

import pylab

PyLab 实际上包含 NumPy 的一些内容，如数组类型。这个库支持点、线的绘制，交互式标注。

首先绘制一幅图像，然后等待用户在绘图窗口的图像区域点击三次。程序将这些点击的坐标 [x, y] 自动保存在 x 列表里。

from PIL import Image

from pylab import \*

im = array(Image.open('empire.jpg'))

imshow(im)

print 'Please click 3 points'

x = ginput(3)

print 'you clicked:',x

show()

147.SciPy（http://scipy.org/） 是建立在 NumPy 基础上，用于数值运算的开源工具包。SciPy 提供很多高效的操作，可以实现数值积分、优化、统计、信号处理，以及对我们来说最重要的图像处理功能。

148.pickle模块

如果想要保存一些结果或者数据以方便后续使用，Python 中的 pickle 模块非常有用。pickle 模块可以接受几乎所有的 Python 对象，并且将其转换成字符串表示，该过程叫做封装（pickling）。从字符串表示中重构该对象，称为拆封（unpickling）。这些字符串表示可以方便地存储和传输。

pickle 模块及其同类模块 cPickle 向 Python 提供了 pickle 支持。后者是用 C 编码的，它具有更好的性能，对于大多数应用程序，推荐使用该模块。

import cPickle as pickle

p1 =pickle.dumps(something)

something =pickle.loads(p1) #load很明显只能重新分配一块内存，所以与原来的通过is判断是false

>>> t1

('this is a string', 42, [1, 2, 3], None)

>>> p1 = pickle.dumps(t1)

>>> p1

"(S'this is a string'/nI42/n(lp1/nI1/naI2/naI3/naNtp2/n."

pickle.dump(something, file\_object, True) #第三个参数是指使用协议。索引0为ASCII，1是旧式2进制，2是新式2进制协议，不同之处在于后者更高效一些。默认的话dump方法使用0做协议。

pickle.load(file\_object) 会从文件里面一个个读取，类似于pymysql中的cursor，读一个就会自己前进。（管道是读取以后会消失）

循环引用造成了特殊的列表，也可以使用cpickle。但是一些对象类型是不可 pickle 的。例如，Python 不能 pickle 文件对象（或者任何带有对文件对象引用的对象），因为 Python 在 unpickle 时不能保证它可以重建该文件的状态

149.platform模块：

platform.system()

if PLATFORM == 'Linux':

PATH = '/etc/hosts'

elif PLATFORM == "Darwin": #OS/X

PATH = '/etc/hosts'

elif PLATFORM == 'Windows':

PATH = r'C:\Windows\System32\drivers\etc\hosts'

对于linux系统，改变host还需要追加对本机名字的识别在最后

if PLATFORM == 'Linux':

hostname\_file = open('/etc/hostname', 'r')

hostname = hostname\_file.read()

hostname\_file.close()

hosts = open(PATH, 'a+')

hosts.writelines("127.0.0.1\t %s" % hostname)

hosts.close()

直接改变hosts文件以后，通过https就可以实现翻墙。

150.#!coding:utf-8《=》# \_\*\_coding:utf-8\_\*\_

151.None在编程的时候，可能会启动重要的作用，因为他表示变量未定义，却又可以被bool转化为false

>>> None==0

False

>>> None==''

False

>>> None

>>> bool(None)

False

152.

s = requests.Session()

def login(xh=2015141462003, pwd=462003): #这种写法比我的写在局部变量里好

payload = {

'xh': xh,

'pwd': pwd,

'VerifyCode': '',

'Submit': '%CC%E1%BD%BB%B5%C7%C2%BD'

}

header = {

"User\_Agent": 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) \ '

'Chrome/57.0.2987.133 Safari/537.36',

"Accept": 'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8',

"Referer": "http://pead.scu.edu.cn/stu/",

'Accept-Language': 'zh-CN,zh;q=0.8,en-US;q=0.6,en;q=0.4',

'Accept-Encoding': 'gzip, deflate'

}

req = s.get('http://pead.scu.edu.cn/stu')

# print("aaaaaaaaaaaaaaaaa")

# print req.text

# input()

s.post('http://pead.scu.edu.cn/stu/dl.asp', headers=header, data=payload)

print("login")

# r.encoding = 'gb2312'

# print r.text

# input()

return s #不过既然定义的是全局变量的话那就没必要在返回值中返回

S.encoding = 'gb2312' 进行编码的设定

153.支持中文变量名

154.# 下载图片

#文件与文件夹的创建值的学习

def downloadPic((title, piclist)):

k = 1

# 图片数量

count = len(piclist)

# 文件夹格式

dirName = u"【%sP】%s" % (str(count), title) #比如【52P】xxxx

# 新建文件夹

os.mkdir(dirName)

for i in piclist:

# 文件写入的名称：当前路径／文件夹／文件名

filename = '%s/%s/%s.jpg' % (os.path.abspath('.'), dirName, k)

print u'开始下载图片:%s 第%s张' % (dirName, k)

with open(filename, "wb") as jpg:

jpg.write(requests.get(i).content)

time.sleep(0.5)

k += 1

#如果想将文件夹创建至非系统默认的地方，需要用到chdir()来更改环境变量。

判断文件夹是否存在使用：os.path.isdir

判断文件或者文件夹是否存在使用：os.path.exists

文件夹的创建相关三个函数:

1、os.path.exists(path) 判断一个目录（其实是路径）是否存在

2、os.makedirs(path) 多层创建目录

3、os.mkdir(path) 创建目录

155.使用pandas的DataFrame格式存放 使用这个库转xlsx（excel）

import pandas

newsdf=pandas.DataFrame(newsary)

newsdf

newsdf.to\_excel('qidian\_rank1.xlsx')#输入到to按住Tab有很多格式，储存

import requests

from bs4 import BeautifulSoup

newsary=[]

for i in range(25):

res=requests.get('http://r.qidian.com/yuepiao?chn=-1&page='+str(i))

#print(res)

soup=BeautifulSoup(res.text,'html.parser')

#print(soup)

#for news in soup.select('.rank-view-list h4'):#Wrap后面一定有个空格，因为网页里有

for news in soup.select('.rank-view-list li'):#Wrap后面一定有个空格，因为网页里有

#print(news)

#print(news.select('a')[1].text,news.select('a')[2].text,news.select('a')[3].text,news.select('p')[1].text,news.select('p')[2].text,news.select('a')[0]['href'])

newsary.append({'title':news.select('a')[1].text,'name':news.select('a')[2].text,'style':news.select('a')[3].text,'describe':news.select('p')[1].text,'lastest':news.select('p')[2].text,'url':news.select('a')[0]['href']})

#上面是在构造字典，下面是在弄成表格的形式。

import pandas

newsdf=pandas.DataFrame(newsary)

newsdf

newsdf.to\_excel('qidian\_rank1.xlsx')#输入到to按住Tab有很多格式，储存

156.PhantomJS、Selenium对付爬虫过程中的ajax

selenium 可以模拟浏览器,设置操作主机的浏览器

157.import base64

base64.b64encode('xxx')

base64.b64decode('xxxx')

158.py服务器：

请问如果在只使用一台机器的情况下，为什么在Gunicorn或者uWSGI前面还要加一层Nginx？哪些情景下这样做能提高性能？

uWSGI是WSGI(Web Server Gateway Interface的缩写)的升级，性能据说提升较大。

一种情况，本地有多个 web 服务，有 Python、php、java 编写的，都想监听 80 端口，这个时候就必须有一个负责转发的服务了。

或者

如果本机确定只跑这一个服务，但是 uwsgi 和 gevent 对于静态资源处理的并不是很好，一是性能问题，二是各种 HTTP 请求缓存头，处理的也没有 Nginx 完善。然后还有一些安全问题，Nginx 作为专业服务器，暴露在公网相对比较安全（虽然有著名的心血漏洞），uwsgi 和 gevent 的话，漏洞恐怕只比 Nginx 多而不是少。

再来就是支持的协议，uwsgi 和 gunicon 早期是不支持 https 的，只能提供 http 给浏览器访问。虽然现在这两者都支持了，但是以后的 spdy 和http2，恐怕也是 nginx 跟进更快一些。

还有一些运维优势，比如服务器被人 CC，这是一种非常常见的情况，nginx 可以比较方便的把一些 IP 加入黑名单，直接改配置文件就好了。要是 uwsgi 或者 gunicorn，恐怕还要修改自己应用的代码，把 IP 过滤写进去。

题主说只考虑单台机器的情况，但是如果不考虑的话，那一个 nginx 做负载均衡那就几乎是必须了。

不过孔豪说uWSGI的时间处理有问题，time.time()不对劲。

159.IPython 是一个 python 的交互式 shell，比默认的python shell 好用得多，支持变量自动补全，自动缩进，支持 bash shell 命令，内置了许多很有用的功能和函数。sudo apt-get install iPython

160.我们常说的生成器，就是带有yield的函数。generator就是iterator的一种，以更优雅的方式实现的iterator。我们每次对一个generator对象调用next()时，函数内部代码执行到"断点"yield，然后返回这一部分的结果，并保存上下文环境，"中断"返回。

>>> def gen():

... while True:

... s = yield

... print(s)

...

>>> g = gen()

>>> g.send("kissg")

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: can't send non-None value to a just-started generator

>>> next(g)

>>> g.send("kissg")

kissg

通过send(value)方法将value作为yield表达式的当前值。调用send(value)时要注意，要确保，generator是在yield处被暂停了，如此才能向yield表达式传值，否则将会报错。也就是要先next启动起来在send。

import time

def consumer():

r = ''

while True:

n = yield r

if not n:

return

print('[CONSUMER] Consuming %s...' % n)

time.sleep(1)

r = '200 OK'

def produce(c):

c.next()

n = 0

while n < 5:

n = n + 1

print('[PRODUCER] Producing %s...' % n)

r = c.send(n)

print('[PRODUCER] Consumer return: %s' % r)

c.close()

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

c = consumer()

produce(c)

注意到consumer函数是一个generator（生成器），把一个consumer传入produce后：

1. 首先调用c.next()启动生成器；

2. 然后，一旦生产了东西，通过c.send(n)切换到consumer执行；

3. consumer通过yield拿到消息，处理，又通过yield把结果传回；

4. produce拿到consumer处理的结果，继续生产下一条消息；

5. produce决定不生产了，通过c.close()关闭consumer，整个过程结束。

161.Flask是一个使用 Python 编写的轻量级 Web 应用框架

pip install -U flask-cors #flask下实现跨域

from flask import Flask

from flask.ext.cors import CORS, cross\_origin

app = Flask(\_\_name\_\_)

cors = CORS(app)

app.config['CORS\_HEADERS'] = 'Content-Type'

@app.route("/")

@cross\_origin()

def helloWorld():

return "Hello, cross-origin-world"

Flask是一个web微框架，主要面向需求简单的小应用。Pyramid和Django都是面向大的应用，但是在扩展性和灵活性上走了不同的路。Pyramid关注灵活性，让开发者选择合适的工具来开发项目。这意味着开发者可以选择数据库，URL结构，模板风格等等。Django的目标是提供web应用开发的一站式解决方案，所以相应的模块也就比较多。

162.@修饰符，一个修饰符就是一个函数，它将被修饰的函数做为参数，并返回修饰后的同名函数或其它可调用的东西

def minus(f):

print 'minus'

f()

def plus(f):

print 'plus'

f()

def test(a):

if a > 3 : return plus

else : return minus

@test(5) #有5这个参数给test这个调用指针了所以就是将5传入

def xxx():

print 'ok'

@minus #xxxx会被作为参数传入minus调用

def xxxx():

print 'ok'

162.SIP是一个自动为C和C++库生成Python扩展模块的工具。为了方便开发PyQt，SIP于1998被“Riverbank Computing”公司创造出来。不过，SIP不专用于PyQt，而是适用于所有的C和C++库。

使用SIP时，程序员首先要编写一个特殊的".sip"文件，使用类似于C++的语法在其中描述扩展模块所提供的类型与函数。然后用SIP将这个文件转化为C++代码。最终编译，与C、C++库链接后就成为Python扩展模块。".sip"文件类似于C、C++的头文件。根据需要，需要程序员用SIP定义的语法添加一些C++代码中没有的信息。因为SIP不支持完整的C++语法，所以不能直接使用C++的头文件作为".sip"文件。

163.pty模块 — Pseudo-terminal utilities

python -c 'import pty; pty.spawn("/bin/bash")' 就会在nc获得的伪终端中开启一个平常一样的真终端

164.Python是没有自带访问windows系统API的库的,需要下载。库的名称叫pywin32

https://sourceforge.net/projects/pywin32/

165.

C:\Users\Battery

$ Fatal error in launcher: Unable to create process using '"'

C:\Users\Battery

$ python -m pip install --upgrade pip

D:\Program Files (x86)\Python27\python.exe: No module named pip

C:\Users\Battery

$ python3 -m pip install --upgrade pip

Collecting pip

Downloading pip-9.0.1-py2.py3-none-any.whl (1.3MB)

100% |████████████████████████████████| 1.3MB 65kB/s

166.TabError: inconsistent use of tabs and spaces in indentation

这个错误表示混合使用了Tab与空格进行缩进表示，上面是使用空格进行缩进，而错误的这里使用了Tab。 scrapy自建代码中就是使用空格进行缩进。

167.如果类被重复定义，那么后面定义的有效，定义覆盖处理

168.下面这样python中字符串的最后一个字符是斜杠会导致出错，但是符合语法：SyntaxError: EOL while scanning string literal：

r"C:\Users\Battery\Desktop\Program\scrapy\scrapyTest\wooyun\bugs\"+item['name']

170.爬虫陷阱：

1.动态网页在url后面加上了一串码，每一次爬虫爬的时候这码都会不一样，这时候只要再将网页之间的关系弄成一个圈就GG

想办法创造出无限长度的URL：http://www.foo.com/bar/foo/bar/foo/bar/foo/bar/.....

2.document哦中包含了大量的字符，从而crash爬虫使用的解析器

171.

list1 = ["这", "是", "一个", "测试"]

for i in range (len(list1)):

print i ,list1[i]

list1 = ["这", "是", "一个", "测试"]

for index, item in enumerate(list1):

print index, item

172. 2.x的python中有httplib，3.x的是http.client

173. csv文件

import csv

with open("jd.csv","a",newline="") as datacsv:

csvwriter = csv.writer(datacsv,dialect = ("excel"))

csvwriter.writerow(["商品","价格","销售状态","链接"])

174.结合qt开发： 学习网址：http://python.jobbole.com/81276/

环境配置：http://blog.csdn.net/a359680405/article/details/45074761

通过D:\Qt\Qt5.8.0\5.8\mingw53\_32\bin\designer.exe 创建自己吓醒要的页面，创建出是后缀为.ui的xml文件。

from PyQt4 import QtCore，QtGui，uic

qtCreatorFile ="创建的ui文件路径"

Ui\_MainWindow, QtBaseClass =uic.loadUiType(qtCreatorFile)

class MyApp(QtGui.QMainWindsow, Ui\_MainWindow):

def \_\_init\_\_(self):

QtGui.QMainWindow.\_\_init\_\_(self)

Ui\_MainWindow.\_\_init\_\_(self)

self.setupUi(self)

if \_\_name\_\_ == "\_\_main\_\_":

app = QtGui.QApplication(sys.argv)

window = MyApp()

window.show()

sys.exit(app.exec\_())

175.exec "x=1+1" 《=》 x=1+1这条语句，会创建一个数值变量

execfile(r'C:\Users\Think\Desktop\pytest.txt') 执行文件中的py命令

print(eval("1+1")) 输出2，eval就是计算一条表达式的值

eval("import('os').system('dir')")

176.Python解释器在启动的时候会首先加载内建名称空间，内建名称空间有许多名字到对象之间映射，而这些名字其实就是内建函数的名称，对象就是这些内建函数本身（注意区分函数名称和函数对象的区别）

>>> dir()

['\_\_builtins\_\_', '\_\_doc\_\_', '\_\_name\_\_', '\_\_package\_\_'] 命名空间是\_\_builtins\_\_

def del\_unsafe(): 这个函数的作用是从命名空间中删除不安全的函数，其实还有个下面使用的exec

UNSAFE\_BUILTINS = ['open',

'file',

'execfile',

'compile',

'reload',

'\_\_import\_\_',

'eval',

'input'] ## block objet?

for func in UNSAFE\_BUILTINS:

del \_\_builtins\_\_.\_\_dict\_\_[func]

from re import findall

del\_unsafe()

print 'Give me your command!'

while True:

try:

inp = findall('\S+', raw\_input())[0]

print "inp=", inp

a = None

exec 'a=' + inp

print 'Return Value:', a

except Exception, e:

print 'Exception:', e

>>> type(().\_\_class\_\_) 对元组的探究

<type 'type'>

>>> ().\_\_class\_\_.\_\_bases\_\_

(<type 'object'>,)

>>> ().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()

[<type 'weakproxy'>, <type 'int'>, <type 'basestring'>,

<type 'bytearray'>, <type 'list'>, <type 'NoneType'>,

<type 'NotImplementedType'>, <type 'traceback'>, <type 'super'>,

<type 'xrange'>, <type 'dict'>, <type 'set'>, <type 'slice'>,

<type 'staticmethod'>, <type 'complex'>, <type 'float'>,

<type 'buffer'>, <type 'long'>, <type 'frozenset'>,

<type 'property'>, <type 'memoryview'>, <type 'tuple'>,

<type 'enumerate'>, <type 'reversed'>, <type 'code'>,

<type 'frame'>, <type 'builtin\_function\_or\_method'>,

<type 'instancemethod'>, <type 'function'>, <type 'classobj'>,

<type 'dictproxy'>, <type 'generator'>, <type 'getset\_descriptor'>,

<type 'wrapper\_descriptor'>, <type 'instance'>, <type 'ellipsis'>,

<type 'member\_descriptor'>, <type 'file'>, <type 'sys.long\_info'>,

... and more!

>>> ().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[40]

<type 'file'>

().\_\_class\_\_.\_\_bases\_\_[0].\_\_subclasses\_\_()[40]("./key").read() 这条命令就能读取同目录下的key文件内容

177.reload语句：

一开始我们的游戏商城配置是从txt读取解析的。后来为了方便运营修改配置，改成从数据库读取并提供后台可视化编辑配置。如果为了使配置生效而重启游戏进程那太麻烦了。这时候reload就派上用途了。

在py 的shell 交互界面中执行一次reload()与重新进行import，从而让py使用新的库执行代码：

import m

from m import variable

reload(m)

from m import variable

178. #coding=utf-8

如果代码中有中文注释，就需要此声明

比较高级的编辑器（比如我的emacs），会根据头部声明，将此作为代码文件的格式。

程序会通过头部声明，解码初始化 u"人生苦短"，这样的unicode对象，（所以头部声明和代码的存储格式要一致，两者并不是一个东西）

179.浏览器实现: WebDriver API 可以通过Python、Ruby、Java 和C#访问

学习网址与浏览器的使用驱动下载：http://www.seleniumhq.org/download/ ChromeDriver的启动自动使用我的ss代理，PAC模式下

驱动的exe文件需要放到PATH能找到的目录中，所以我就放在了py的软件安装根目录。或者自己在编程的时候指定路径。

chromedriver = "C:\Program Files (x86)\Google\Chrome\Application\chromedriver.exe"

os.environ["webdriver.chrome.driver"] = chromedriver

browser = webdriver.Chrome(chromedriver)

WebDriver 支持 Firefox (FirefoxDriver)、Ie (InternetExplorerDriver)、Opera (OperaDriver) 和 Chrome (ChromeDriver) 。具体函数通过dir()对变量进行查看就知道了。

selenium 会等待网页加载完毕以后才进行下一步操作

#!/usr/bin/python3

#coding:utf-8

from selenium import webdriver

#下面填入京东的用户名以及密码

jd\_up={"ue":"\*\*\*\*","pd":"\*\*\*\*"}

chrome=webdriver.Chrome() #selenium操作chrome浏览器需要有ChromeDriver驱动来协助。

chrome.get(url="https://passport.jd.com/new/login.aspx?")

chrome.find\_element\_by\_xpath("//\*[@id=\"content\"]/div/div[1]/div/div[2]/a").click()

User=chrome.find\_element\_by\_id("loginname")

User.clear()

User.send\_keys(jd\_up["ue"])

Passwd=chrome.find\_element\_by\_id("nloginpwd")

Passwd.clear()

Passwd.send\_keys(jd\_up["pd"])

chrome.find\_element\_by\_id("loginsubmit").click()

while True:

try:

chrome.get(url="https://item.jd.com/4099139.html")

chrome.find\_element\_by\_id("choose-btn-ko").click()

chrome.find\_element\_by\_id("order-submit").click()

print("抢购成功并且已下单！！")

#chrome.quit()

except Exception:

print("还未开始抢购！！")

webdriver只能在一个页面上定位和识别元素，对于iframe或frame不能直接定位。通过switch\_to.frame()方法将当前的主体切换为frame或iframe表单的内嵌页面中

switch\_to.frame() 默认使用id或name属性值定位

或者将变量存起来以后定位el = driver.find\_element\_by\_xpath("//\*[@class='if']") driver.switch\_to.frame(el)

driver.switch\_to.parent\_frame () #跳出当前一级表单

driver.switch\_to.default\_content() #切换到最外层的页面

driver.title 代表着页面的title。虽然开了一个浏览器，但是只会打开一个页面

driver.current\_url

driver.back() #返回前一个页面

driver.forward() #前进前一个页面

driver.refresh() #刷新当前页面

driver.set\_window\_size(600,600) 窗口大小

driver.maximize\_window()

js = "window.scrollTo(100,300);" driver.execute\_script(js) 通过执行js命令控制滚动条的位置

driver.get\_screenshot\_as\_file('E:\\screenshot\\baidu.jpg') 窗口截屏

click函数不仅仅可以用于按钮，它还能可以单击任何可以单击的文字、图片、checkbox、radio、select下拉框等。

寻找指定元素：

find\_element\_by\_id("choose-btn-ko")

find\_elements\_by\_name()

find\_elements\_by\_class\_name()

find\_elements\_by\_tag\_name()

find\_element\_by\_partial\_link\_text("京公网安备")

find\_elements\_by\_link\_text()

find\_element\_by\_xpath(".//input[@id='kw']")

find\_elements\_by\_css\_selector()

element.text 标签对中的文本内容

element.clear() 用于清除已有的输入

element.send\_keys("xxxx") 向指定的element输入自己的值

element.send\_keys(Keys.CONTROL,'a') 发送键盘快捷键C-A

element.send\_keys(Keys.ENTER) 发送Enter

driver.find\_element\_by\_css\_selector("[type='file']").send\_keys("C:\\Users\\Administrator\\Desktop\\html5.txt") 上传按钮其实就是用来确定本机地址的

element.click()

ActionChains(driver).double\_click(element).perform() 一种逗逼的新写法 双击

ActionChains(driver).move\_to\_element(attrible).perform() 鼠标悬停

ActionChains(driver).drag\_and\_drop(source,target).perform() 元素拖放

webdriver操作cookie的方法：

get\_cookies()：获取所有的cookie信息

get\_cookie(name)：返回字典的key为name的cookie信息

add\_cookie(cookie\_dict) ：添加cookie。cookie\_dict值字典对象，必须有name和value值

delete\_cookie(name,optionString)：删除cookie信息。name是要删除的cookie名称，optionString是该cookie的选项，目前支持的选项包括“路径”，“域”

delete\_all\_cookies()：删除所有的cookie信息

显式等待（在定位某个元素之前检查该元素是否被加载出来了）使WebDriver等待某个条件成立时继续执行，否则在达到最大时长抛出超时异常：WebDriverWait() expected\_conditions()

WebDriverWait(driver,timeout,poll\_frequency=0.5,ignored\_exceptions=None)

poll\_frequency：检测的间隔时间，默认是0.5秒

ignored\_exceptions：超时后的异常信息，默认抛出NoSuchElementException异常。

WebDriverWait(driver,5,0.5,ignored\_exceptions=None).until(lambda driver : driver.find\_element\_by\_xpath("//\*")

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

driver = webdriver.Firefox()

driver.get(r"http://www.baidu.com//")

element = WebDriverWait(driver,5,0.5,ignored\_exceptions=None).until(

EC.presence\_of\_element\_located((selenium.webdriver.common.by.ID,"kw"))

)

element.send\_keys("ptyh")

隐式等待：如果超出最大时常还没有加载会抛出NoSuchElementException异常

implicitly\_wait(超时时间，默认这个参数是0)

from selenium .common.exceptions import NoSuchElementException

driver.implicitly\_wait(10)

try:

el = driver.find\_element\_by\_xpath("//\*[@id='kw']")

el1 = driver.find\_element\_by\_id("su")

sleep(2)

except NoSuchElementException as e:

print(e)

else:

el.send\_keys("python")

el1.click()

finally:

driver.quit()

程序暂时休眠：

from time import sleep

多窗口操作：窗口句柄操作

handle = driver.current\_window\_handle

all\_handles = driver.window\_handles

driver.switch\_to.window(handle) #通过句柄切换操作窗口

print( driver.title )

句柄对应是哪个窗口，需要自己进行判断

验证码：

最好还是用户来识别，一旦检测到验证码就叫用户来输入

180. 虽然对于ordertime的意义来讲不是很合适，但是对用户很友好

ordertime = input('''请选择：

1.设置下单时间

2.选择立即下单(可用于监控库存，自动下单)

请输入选择（1/2):

''')

181.终端输入密码

pwd = input('请输入京东密码:') 但是密码会被明文显示出来

import getpass

pwd = getpass.getpass('password: ') 就像是linux用户root的时候一样

如果要能够被\*号隐藏，那么只能自己编写函数了。回退键的功能由sys.stdout.write函数实现。

182.运行在JVM上：JPython

183.给回调函数加上状态信息： <http://python3-cookbook-personal.readthedocs.io/zh_CN/latest/c07/p10_carry_extra_state_with_callback_functions.html>

184.反射：

通过dir()函数可以查看支持的成员与方法

反射即想到4个内置函数分别为:getattr、hasattr、setattr、delattr  获取成员、检查成员、设置成员、删除成员。四个内置函数需要对“对象”操作。

1. 音频处理库

从视频流（实验中使用mp4视频，而不是直播流）中分离出音频流（因为web api接口期待音频编码，可选值：raw（未压缩的pcm或wav格式）、speex（speex格式）、speex-wb（宽频speex格式），并按照帧数的间隔（时间间隔）划分每一个人的说话语句，最后通过本地SDK或者web api实现文语转换。之所以要传短音频，是因为长音频要收费。

基于贝叶斯信息准则BIC实现分隔点的寻找 <https://blog.csdn.net/wblgers1234/article/details/75896605> 简单点便是认为足够多的帧数间隔就是分隔点。对应重要参考里面的py\_speech\_seg方法。

很重要的参考： <https://blog.csdn.net/watfe/article/details/80284242>

推荐按照波形结合机器学习的办法直接判断，因为不同声音的音调可能经过了放大处理。

ffmpeg -i test.mp4 -ab 160k -ac 2 -ar 44100 -vn audio.wav

提取速度非常快，8.7MB（3分钟）的mp4视频是瞬时完成的

gst-launch-0.10 playbin2 uri=file://`pwd`/test.mp4 audio-sink='identity single-segment=true ! audioconvert ! audio/x-raw-int, endianness=(int)1234, signed=(boolean)true, width=(int)16, depth=(int)16, rate=(int)16000, channels=(int)1 ! wavenc ! filesink location=audio.wav’

avconv -i foo.mp4 -ab 160k -ac 1 -ar 16000 -vn ffaudio.wav

opencv（） 可以从摄像头读取视频，处理（切割等）视频，，虽然主要是计算机视觉库

numpy 处理视频与音频的编解码的时候通过会借助这个库来完成编解码函数

下面提供两个ffmpeg的库，但是这并不同C语言中可以从视频的很底层进行处理，但是python中只能算是对命令集成：

ffmpy库对ffmpeg的命令进行了库级封装

pyffmpeg 能够从视频中提取到独立的帧

ffmpeg-python 也是对命令的库级封装，封装之后是的调用看起来比较python，而不是使用系统命令

下面提供gstreamer相关的库：

PyGST

下面提供vlc相关的库：  
 python-vlc

python对库级别的移植，叫做bindings

web api比如腾讯云都会直接在下方简单介绍使用了什么技术

类似平台还有 科大飞讯、百度语音（百度云）、阿里云、Google、微软

1. 通过python 加载dll，使用dll处理一些任务，通过这种方法试下新加速。因为dll的本身是C/C++编写。

ctypes是Python的外部函数库。它提供了C兼容的数据类型，并且允许调用动态链接库/共享库中的函数。它可以将这些库包装起来给Python使用

1. 单元测试库unittest

<https://segmentfault.com/a/1190000008048642> 相当于按照模块化（函数化）编写的main

pip install -r requirements.txt

文件内容：

beautifulsoup4==4.6.0

bs4==0.0.1

certifi==2017.7.27.1

html5lib==0.9999999

pandas==0.21.1

requests==2.12.4

whl文件：wheel

界面：PyQt4

程序外部参数输入：argparse、from optparse import OptionParser

json解析库：json

office库：csv

Base85编码：base64 使用：base64.b85encode(b'username').decode()

Python Web 应用框架：

轻量级 flask

WSGI(Web Server Gateway Interface) : wsgiref

协程：gevent（yield支持不完全）

科学计算：Anaconda 学习网站：http://blog.csdn.net/u012675539/article/details/46974217

复制：from copy import deepcopy

控制浏览器的行为：from selenium import webdriver

时间：datetime datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S");

time time.time() 返回1507452341.981898

urllib : py3将urllib2（http请求构造）合并到urllib中，其中 urllib2.urlopen() 变成了urllib.request.urlopen()。urllib2.Request() 变成了 urllib.request.Request()

urllib3 HTTP/HTTPS 连接池

urllib4 增加了一些HTTP特性，比如按照返回包头自识别编码；增加对FTP协议的支持

urllib5 前面几个urllib的集合