
目录

简介

Introduction	1.1
--------------	-----

TOP 100

Python中关键字yield有什么作用?	2.1
Python中的元类(metaclass)是什么?	2.2
在Python中有三元运算符吗?	2.3
用Python如何一个文件是否存在?	2.4
if __name__ == "__main__":有什么作用?	2.5
在Python中调用外部命令?	2.6
如何在一个表达式里合并两个字典?	2.7
检查一个文件夹是否存在,如果不存在就创建它	2.8
Python中有检查字符串包含的方法吗?	2.9
用字典的值(key)对字典进行排序	2.10
在Python中列出目录中的所有文件	2.11
检查列表是否为空的最好方法	2.12
装饰器@staticmethod和@classmethod有什么区别?	2.13
Python中的append和extend	2.14
在循环中获取索引(数组下标)	2.15
如何在一个函数里用全局变量?	2.16
Python中如何在一个函数中加入多个装饰器?	2.17
理解Python切片	2.18
查找列表中某个元素的下标	2.19
检查一个键在字典中是否存在	2.20
怎么在windows下安装pip?	2.21
使用 for 来迭代字典的原理	2.22
在嵌套列表中获取 flat list	2.23
__str__和repr__的区别	2.24

在 Python 里如何手工进行延迟?	2.25
Python 中的"小震撼":变化的默认参数	2.26
怎么用引用来改变一个变量?	2.27
在Python里获取当前时间	2.28
理解Python中super()和__init__()方法	2.29
在一行里获取多个异常	2.30
字典里添加元素的方法	2.31
有方法让Python运行在Android上吗?	2.32
把字符串解析成浮点数或者整数	2.33
如何克隆或者 copy 一个 list ?	2.34
把文件一行行读入数组	2.35
合并两个列表	2.36
在 macOS 上如何安装 pip	2.37
获得一个字符串的子串	2.38
Python 中**和*参数有什么用?	2.39
在Python怎么样才能把列表分割成同样大小的块?	2.40
手动抛出异常	2.41
怎么样获取一个列表的长度?	2.42
怎么在终端里输出颜色?	2.43
查找当前目录和文件目录	2.44
在 Python 中如何将字符串转回为小写	2.45
复制文件	2.46
Python 中类的静态变量	2.47
把字符串转化成时间	2.48
Python中用什么代替switch语句?	2.49
删除文件或者文件夹	2.50
为什么在C++中读取stdin中的行会比Python慢呢?	2.51
__init__.py 的作用	2.52
用pip升级所有包	2.53
获取列表最后一个元素	2.54
查看一个对象的类型	2.55
为什么 "1000000000000000 in range(1000000000000001)" 在 Python3 里速度那么快?	
如何使打印的内容没有空格或换行	2.57 2.56
为什么是string.join(list)而不是list.join(string)?	2.58

静态方法	2.59
通过列表中字典的值对列表进行排序	2.60
在列表中随机取一个元素	2.61
如何移除换行符?	2.62
在 Python 中如何获取环境变量	2.63
bytes 换成 string	2.64
对于初学者如何理解 @classmethod 和 @staticmethod?	2.65
检查一个字符串是否是一个数字	2.66
通过函数名的字符串来调用这个函数	2.67
如何增加文件的内容?	2.68
*args和 **kwargs	2.69
在 Python 里如何读取 stdin?	2.70
字符串格式化:%和.format	2.71
限制浮点型到小数点后两位	2.72
如何解析一个 JSON 文件?	2.73
如何改变 matplotlib 图像大小	2.74
在 Python 中如何得到对象的所有属性?	2.75
在Python里如何用枚举类型?	2.76
如何计算列表中元素出现的个数?	2.77
在 Python 中如何移除字典中的 key ?	2.78
如何离开/退出/停用Python的virtualenv ?	2.79
反转字符串	2.80
如何删除 list 中指定 index 的元素?	2.81
如何查看 Python 变量的类型?	2.82
使用列表推导式的方式来创建字典	2.83
如何获取实例的类名	2.84
整数转化为字符串	2.85
找出文件夹中所有扩展名位 .txt 的文件	2.86
给字符串填充0	2.87
生成包含大写字母和数字的随机字符串	2.88
输出到stderr	2.89
在实例名字前单下划线和双下划线的含义	2.90
如何测量脚本运行时间?	2.91

UnicodeEncodeError: 'ascii' codec can't encode character u'\xa0' in position 20: ordinal not in range(128)	2.92
如何连接MySQL?	2.93
如何使用线程	2.94
检查字符串为空的最优雅的方法	2.95
如何在 .format 中使用大括号?	2.96

- [Stackoverflow 上关于 Python 的问题](#)
 - [目录](#)

Stackoverflow 上关于 Python 的问题

STARS

19

排名根据 vote 数量选取, 许多 SO 上的回答质量确实高, 有能力建议查看原文, 一般引用的文章也非常好,

翻译是根据 question id 写在 data 文件夹, 运行脚本 `get_so_100.py` 自动生成 part 部分文档

目录

图表数据更新时间 2018-10-18 01:33:51

rank	vote	view	answer	url	中文
1	8558	1837210	39	url	Python中关键字yield有什么作用?
2	4734	636875	16	url	Python中的元类(metaclass)是什么?
3	4626	1468573	22	url	在Python中有三元运算符吗?
4	4530	3091022	39	url	用Python如何一个文件是否存在?
5	4348	1893044	29	url	<code>if __name__ == "__main__":</code> 有什么作用?
6	3782	2678600	52	url	在Python中调用外部命令?
7	3383	1264559	49	url	如何在一个表达式里合并两个字典?
8	3119	1993725	25	url	检查一个文件夹是否存在,如果不存在就创建它
9	3072	2877536	14	url	Python中有检查字符串包含的方法吗?
10	3039	1872959	41	url	用字典的值(key)对字典进行排序
11	2967	2634401	32	url	在Python中列出目录中的所有文件
12	2862	1856471	30	url	检查列表是否为空的最好方法
13	2794	588058	24	url	装饰器@staticmethod和@classmethod有什么区别?
14	2787	2286116	24	url	Python中的append和extend
15	2680	1455704	21	url	在循环中获取索引(数组下标)

16	2554	2740443	18	url	如何在一个函数里用全局变量?
17	2428	444941	15	url	Python中如何在一个函数中加入多个装饰器?
18	2396	1241133	31	url	理解Python切片
19	2360	2761878	27	url	查找列表中某个元素的下标
20	2311	2380401	19	url	检查一个键在字典中是否存在
21	2273	2487553	34	url	怎么在windows下安装pip?
22	2236	3143856	11	url	使用 for 来迭代字典的原理
23	2203	1237032	36	url	在嵌套列表中获取 flat list
24	2186	487459	21	url	__str__ 和 repr__ 的区别
25	2132	2306176	12	url	在 Python 里如何手工进行延迟?
26	2114	118677	30	url	Python中的"小震撼":变化的默认参数
27	2113	1002442	23	url	怎么用引用来改变一个变量?
28	2049	2277012	31	url	在Python里获取当前时间
29	2044	1357234	7	url	理解Python中 super() 和 __init__() 方法
30	2042	515111	7	url	在一行里获取多个异常
31	1998	2444832	20	url	字典里添加元素的方法
32	1967	511549	25	url	有方法让Python运行在Android上吗?
33	1828	3109154	24	url	把字符串解析成浮点数或者整数
34	1787	1099948	18	url	如何克隆或者 copy 一个 list ?
35	1785	2790533	34	url	把文件一行行读入数组
36	1727	1504654	28	url	合并两个列表
37	1676	1568802	21	url	在 macOS 上如何安装 pip
38	1659	2178345	11	url	获得一个字符串的子串
39	1658	461700	18	url	Python中 ** 和 * 参数有什么用?
40	1653	643357	57	url	在Python怎么样才能把列表分割成同样大小的块?
41	1623	1189801	5	url	手动抛出异常
42	1604	2652232	6	url	怎么样获取一个列表的长度?
43	1580	827210	37	url	怎么在终端里输出颜色?
44	1574	1911103	15	url	查找当前目录和文件目录
45	1570	1836708	8	url	在 Python 中如何将字符串转回为小写

46	1565	1231023	14	url	复制文件
47	1561	970000	16	url	Python 中类的静态变量
48	1551	1956758	19	url	把字符串转化成时间
49	1529	1252208	49	url	Python中用什么代替switch语句?
50	1523	1489169	10	url	删除文件或者文件夹
51	1521	204859	10	url	为什么在C++中读取stdin中的行会比Python慢呢?
52	1497	673854	11	url	__init__.py 的作用
53	1479	837904	49	url	用pip升级所有包
54	1467	1346339	14	url	获取列表最后一个元素
55	1450	1382804	9	url	查看一个对象的类型
56	1449	119508	7	url	为什么 "1000000000000000 in range(1000000000000001)" 在 Python3 里速度那么快?
57	1441	1459468	26	url	如何使打印的内容没有空格或换行
58	1428	1063020	9	url	为什么是string.join(list)而不是list.join(string)?
59	1424	790037	9	url	静态方法
60	1420	633857	191	url	
61	1413	541572	17	url	通过列表中字典的值对列表进行排序
62	1363	1038536	14	url	在列表中随机取一个元素
63	1363	1319979	26	url	如何移除换行符?
64	1361	1130319	11	url	在 Python 中如何获取环境变量
65	1349	1258301	16	url	bytes 换成 string
66	1321	508348	11	url	对于初学者如何理解 @classmethod 和 @staticmethod ?
67	1282	1027318	31	url	检查一个字符串是否是一个数字
68	1247	502333	10	url	通过函数名的字符串来调用这个函数
69	1236	1278128	7	url	如何增加文件的内容?
70	1223	771656	11	url	*args 和 **kwargs
71	1211	1210518	19	url	在 Python 里如何读取 stdin?
72	1195	898656	15	url	字符串格式化:%和.format
73	1190	2129025	19	url	限制浮点型到小数点后两位
74	1184	2147132	8	url	如何解析一个 JSON 文件?

75	1184	1332206	13	url	如何改变 matplotlib 图像大小
76	1183	605483	9	url	在 Python 中如何得到对象的所有属性?
77	1151	1533132	28	url	
78	1146	674169	43	url	在Python里如何用枚举类型?
79	1137	1132186	19	url	如何计算列表中元素出现的个数?
80	1114	930787	8	url	在 Python 中如何移除字典中的 key ?
81	1114	475573	8	url	如何离开/退出/停用Python的 virtualenv ?
82	1112	1034686	21	url	反转字符串
83	1085	1925603	18	url	如何删除 list 中指定 index 的元素?
84	1085	1673710	15	url	如何查看 Python 变量的类型?
85	1062	632188	10	url	使用列表推导式的方式来创建字典
86	1055	626395	8	url	如何获取实例的类名
87	1048	2615171	10	url	整数转化为字符串
88	1044	1193546	30	url	找出文件夹中所有扩展名位 .txt 的文件
89	1044	594609	13	url	给字符串填充0
90	1038	651117	27	url	生成包含大写字母和数字的随机字符串
91	1028	1599356	1	url	
92	1022	608770	15	url	输出到stderr
93	1017	297432	13	url	在实例名字前单下划线和双下划线的含义
94	1006	377869	23	url	如何测量脚本运行时间?
95	994	1197205	19	url	UnicodeEncodeError: 'ascii' codec can't encode character u'\xa0' in position 20: ordinal not in range(128)
96	987	1016356	21	url	如何连接MySQL?
97	974	754731	17	url	如何使用线程
98	973	1068581	22	url	检查字符串为空的最优雅的方法
99	973	596374	6	url	
100	966	240271	10	url	如何在 .format 中使用大括号?

rank	vote	view	answer	url
1	8558	1837210	39	url

Python中关键字yield有什么作用？

yield有什么用？

例如下面这段代码：

```
def node._get_child_candidates(self, distance, min_dist, max_dist):
    if self._leftchild and distance - max_dist < self._median:
        yield self._leftchild
    if self._rightchild and distance + max_dist >= self._median:
        yield self._rightchild
```

下面是调用它：

```
result, candidates = list(), [self]
while candidates:
    node = candidates.pop()
    distance = node._get_dist(obj)
    if distance <= max_dist and distance >= min_dist:
        result.extend(node._values)
    candidates.extend(node._get_child_candidates(distance, min_dist, max_dist))
return result
```

当 `_get_child_candidates` 方法被调用的时候发生了什么？是返回一个列表？还是一个元祖？它还能第二次调用吗？后面的调用什么时候结束？

为了理解yield有什么用,首先得理解generators,而理解generators前还要理解iterables

Iterables

当你创建了一个列表,你可以一个一个的读取它的每一项,这叫做iteration:

```
>>> mylist = [1, 2, 3]
>>> for i in mylist:
...     print(i)
1
2
3
```

Mylist是可迭代的.当你用列表推导式的时候,你就创建了一个列表,而这个列表也是可迭代的:

```
>>> mylist = [x*x for x in range(3)]
>>> for i in mylist:
...     print(i)
0
1
4
```

所有你可以用在 `for...in...` 语句中的都是可迭代的:比如lists,strings,files...因为这些可迭代的对象你可以随意的读取所以非常方便易用,但是你必须把它们的值放到内存里,当它们有很多值时就会消耗太多的内存.

Generators

生成器也是迭代器的一种,但是你能只能迭代它们一次.原因很简单,因为它们不是全部存在内存里,它们只在要调用的时候在内存里生成:

```
>>> mygenerator = (x*x for x in range(3))
>>> for i in mygenerator:
...     print(i)
0
1
4
```

生成器和迭代器的区别就是用 `()` 代替 `[]`,还有你不能用 `for i in mygenerator` 第二次调用生成器:首先计算0,然后会在内存里丢掉0去计算1,直到计算完4.

Yield

Yield 的用法和关键字 `return` 差不多,下面的函数将会返回一个生成器:

```
>>> def createGenerator():
...     mylist = range(3)
...     for i in mylist:
...         yield i*i
...
>>> mygenerator = createGenerator() # 创建生成器
>>> print(mygenerator) # mygenerator is an object!
<generator object createGenerator at 0xb7555c34>
>>> for i in mygenerator:
...     print(i)
0
1
4
```

在这里这个例子好像没什么用,不过当你的函数要返回一个非常大的集合并且你希望只读一次的话,那么它就非常的方便了.

要理解 `yield` 你必须先理解当你调用函数的时候,函数里的代码并没有运行.函数仅仅返回生成器对象,这就是它最微妙的地方:-)

然后呢,每当 `for` 语句迭代生成器的时候你的代码才会运转.

现在,到了最难的部分:

当 `for` 语句第一次调用函数里返回的生成器对象,函数里的代码就开始运作,直到碰到 `yield`, 然后会返回本次循环的第一个返回值.所以下一次调用也将运行一次循环然后返回下一个值,直到没有值可以返回.

一旦函数运行并且没有碰到 `yield` 语句就认为生成器已经为空了.原因有可能是循环结束或者没有满足 `if/else` 之类的.

对于你的代码的解释

生成器:

```
# 这里你创建node方法的对象将会返回一个生成器
def node._get_child_candidates(self, distance, min_dist, max_dist):

    # 这里的代码你每次使用生成器对象的时候将会调用

    if self._leftchild and distance - max_dist < self._median:
        yield self._leftchild

    if self._rightchild and distance + max_dist >= self._median:
        yield self._rightchild

    # 如果代码运行到这里,生成器就被认为变成了空的
```

调用:

```

# 创建空列表和一个当前对象索引的列表
result, candidates = list(), [self]

# 在candidates上进行循环(在开始只包含一个元素)
while candidates:

    # 获得最后一个candidate然后从列表里删除
    node = candidates.pop()

    # 获取obj和candidate的距离
    distance = node._get_dist(obj)

    # 如果distance何时将会填入result
    if distance <= max_dist and distance >= min_dist:
        result.extend(node._values)

    candidates.extend(node._get_child_candidates(distance, min_dist, max_dist))

return result

```

这段代码有几个有意思的地方:

- 一般的时候我们会在循环迭代一个列表的同时在列表中添加元素:-)尽管在有限循环里结束多少有一些危险,但也不失为一个简单的方法去遍历嵌套的数据.在这里 `candidates.extend(node._get_child_candidates(distance, min_dist, max_dist))` 将遍历生成器的每一个值,但是 `while` 循环中的`candidates`将不再保存已经遍历过的生成器对象,也就是说添加进`candidates`的生成器对象只会遍历一遍。
- `extend()` 是一个列表对象的方法,它可以把一个迭代对象添加进列表。

我们经常这么用:

```

>>> a = [1, 2]
>>> b = [3, 4]
>>> a.extend(b)
>>> print(a)
[1, 2, 3, 4]

```

但是在你给的代码里得到的是生成器,这样做的好处:

1. 你不需要读这个值两次
2. 你能得到许多孩子节点但是你不希望他们全部存入内存。

这种方法之所以能很好的运行是因为Python不关心方法的参数是不是一个列表.它只希望接受一个迭代器,所以不管是strings,lists,tuples或者generators都可以!这种方法叫做duck typing,这也是Python看起来特别cool的原因之一.但是这又是另外一个传说了,另一个问题

好了,看到这里可以打住了,下面让我们看看生成器的高级用法:

控制迭代器的穷尽

```
>>> class Bank(): # 让我们建个银行,生产许多ATM
...     crisis = False
...     def create_atm(self):
...         while not self.crisis:
...             yield "$100"
>>> hsbc = Bank() # 当一切就绪了你想要多少ATM就给你多少
>>> corner_street_atm = hsbc.create_atm()
>>> print(corner_street_atm.next())
$100
>>> print(corner_street_atm.next())
$100
>>> print([corner_street_atm.next() for cash in range(5)])
['$100', '$100', '$100', '$100', '$100']
>>> hsbc.crisis = True # cao, 经济危机来了没有钱了!
>>> print(corner_street_atm.next())
<type 'exceptions.StopIteration'>
>>> wall_street_atm = hsbc.create_atm() # 对于其他ATM,它还是True
>>> print(wall_street_atm.next())
<type 'exceptions.StopIteration'>
>>> hsbc.crisis = False # 麻烦的是,尽管危机过去了,ATM还是空的
>>> print(corner_street_atm.next())
<type 'exceptions.StopIteration'>
>>> brand_new_atm = hsbc.create_atm() # 只能重新新建一个bank了
>>> for cash in brand_new_atm:
...     print cash
$100
$100
$100
$100
$100
$100
$100
$100
$100
$100
...
```

它对于一些不断变化的值很有用,比如说用来控制资源的访问.

Itertools,你的好基友

itertools模块包含了一些特殊的函数可以操作可迭代对象.有没有想过复制一个生成器?链接两个生成器?把嵌套列表里的值组织成一个列表?Map/Zip还不用创建另一个列表?

来吧 `import itertools`

来一个例子?让我们看看4匹马比赛有多少个排名结果:

```
>>> horses = [1, 2, 3, 4]
>>> races = itertools.permutations(horses)
>>> print(races)
<itertools.permutations object at 0xb754f1dc>
>>> print(list(itertools.permutations(horses)))
[(1, 2, 3, 4),
 (1, 2, 4, 3),
 (1, 3, 2, 4),
 (1, 3, 4, 2),
 (1, 4, 2, 3),
 (1, 4, 3, 2),
 (2, 1, 3, 4),
 (2, 1, 4, 3),
 (2, 3, 1, 4),
 (2, 3, 4, 1),
 (2, 4, 1, 3),
 (2, 4, 3, 1),
 (3, 1, 2, 4),
 (3, 1, 4, 2),
 (3, 2, 1, 4),
 (3, 2, 4, 1),
 (3, 4, 1, 2),
 (3, 4, 2, 1),
 (4, 1, 2, 3),
 (4, 1, 3, 2),
 (4, 2, 1, 3),
 (4, 2, 3, 1),
 (4, 3, 1, 2),
 (4, 3, 2, 1)]
```

理解迭代的内部机制

迭代是可迭代对象(对应 `__iter__()` 方法)和迭代器(对应 `__next__()` 方法)的一个过程.可迭代对象就是任何你可以迭代的对象(废话啊).迭代器就是可以让你迭代可迭代对象的对象(有点绕口,意思就是这个意思)

预知后事如何,请看[for 循环是如何工作的](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
2	4734	636875	16	url

Python中的元类(metaclass)是什么？

元类是什么？如何使用元类？

类对象

在理解元类之前,你需要掌握Python里的类.Python中类的概念借鉴于Smalltalk，这显得有些奇特.

在大多数语言中,类就是一组用来描述如何生成一个对象的代码段。在Python中这一点仍然成立：

```
>>> class ObjectCreator(object):
...     pass
...

>>> my_object = ObjectCreator()
>>> print(my_object)
<__main__.ObjectCreator object at 0x8974f2c>
```

但是在Python中类也是对象.

是的,对象.

每当你用到关键字 `class` , Python就会执行它并且建立一个对象.例如:

```
>>> class ObjectCreator(object):
...     pass
...


```

上面代码在内存里创建了名叫"ObjectCreator"的对象.

这个对象(类)有生成对象(实例)的能力,这就是为什么叫做类.

它是个对象,所以:

- 你可以把它赋值给一个变量
- 你可以赋值它
- 你可以给它添加属性

- 你可以以作为函数参数来传递它

e.g.:

```
>>> print(ObjectCreator) # 你可以打印一个类,因为它是一个对象
<class '__main__.ObjectCreator'>
>>> def echo(o):
...     print(o)
...
>>> echo(ObjectCreator) # 你可以把类作为参数传递
<class '__main__.ObjectCreator'>
>>> print(hasattr(ObjectCreator, 'new_attribute'))
False
>>> ObjectCreator.new_attribute = 'foo' # 可以给一个类添加属性
>>> print(hasattr(ObjectCreator, 'new_attribute'))
True
>>> print(ObjectCreator.new_attribute)
foo
>>> ObjectCreatorMirror = ObjectCreator # 可以把类赋值给一个变量
>>> print(ObjectCreatorMirror.new_attribute)
foo
>>> print(ObjectCreatorMirror())
<__main__.ObjectCreator object at 0x8997b4c>
```

动态创建类

因为类也是对象,你可以在运行时动态的创建它们,就像其他任何对象一样。

首先,你可以在函数中创建类,使用class关键字即可:

```
>>> def choose_class(name):
...     if name == 'foo':
...         class Foo(object):
...             pass
...         return Foo # 返回一个类不是一个实例
...     else:
...         class Bar(object):
...             pass
...         return Bar
...
>>> MyClass = choose_class('foo')
>>> print(MyClass) # 返回一个类不是一个实例
<class '__main__.Foo'>
>>> print(MyClass()) # 你可以在类里创建一个对象
<__main__.Foo object at 0x89c6d4c>
```

但这还不够动态,因为你仍然需要自己编写整个类的代码.

既然类是对象,那么肯定有什么东西来生成它.

当你使用关键字 `objects` ,Python自动的创建对象.像Python中大多数的东西一样,他也给你自己动手的机会.

记得函数 `type` 吗?这个古老好用的函数能让你知道对象的类型是什么:

```
>>> print(type(1))
<type 'int'>
>>> print(type("1"))
<type 'str'>
>>> print(type(ObjectCreator))
<type 'type'>
>>> print(type(ObjectCreator()))
<class '__main__.ObjectCreator'>
```

这里, `type` 有一种完全不同的能力,它也能动态的创建类. `type` 可以接受一个类的描述作为参数,然后返回一个类.

(我知道,根据传入参数的不同,同一个函数拥有两种完全不同的用法是一件很傻的事情,但这在Python中是为了保持向后兼容性)

`type` 这样工作:

```
type(类名,
      父类名的元组 (针对继承情况,可以为空),
      包含属性的字典(名称和值))
```

e.g.:

```
>>> class MyShinyClass(object):
...     pass
```

可以手动创建:

```
>>> MyShinyClass = type('MyShinyClass', (), {}) # 返回类对象
>>> print(MyShinyClass)
<class '__main__.MyShinyClass'>
>>> print(MyShinyClass()) # 创建一个类的实例
<__main__.MyShinyClass object at 0x8997cec>
```

你会发现我们使用“MyShinyClass”作为类名,并且也可以把它当做一个变量来作为类的引用。类和变量是不同的,这里没有任何理由把事情弄的复杂。

`type` 可以接受一个字典来定义类的属性:

```
>>> class Foo(object):
...     bar = True
```

可以写成:

```
>>> Foo = type('Foo', (), {'bar':True})
```

然后我们可以像用正常类来用它:

```
>>> print(Foo)
<class '__main__.Foo'>
>>> print(Foo.bar)
True
>>> f = Foo()
>>> print(f)
<__main__.Foo object at 0x8a9b84c>
>>> print(f.bar)
True
```

当然,你也可以继承它:

```
>>> class FooChild(Foo):
...     pass
```

这样:

```
>>> FooChild = type('FooChild', (Foo,), {})
>>> print(FooChild)
<class '__main__.FooChild'>
>>> print(FooChild.bar) # bar从Foo继承
True
```

要是在类中添加方法,你要做的就是将函数名写入字典就可以了,不懂可以看下面:

```
>>> def echo_bar(self):
...     print(self.bar)
...
>>> FooChild = type('FooChild', (Foo,), {'echo_bar': echo_bar})
>>> hasattr(Foo, 'echo_bar')
False
>>> hasattr(FooChild, 'echo_bar')
True
>>> my_foo = FooChild()
>>> my_foo.echo_bar()
True
```

你可以看到,在Python中,类也是对象,你可以动态的创建类。这就是当你使用关键字class时Python在幕后做的事情,而这就是通过元类来实现的。

什么是元类(终于到正题了)

元类就是创建类的东西.

你是为了创建对象才定义类的,对吧?

但是我们已经知道了Python的类是对象.

这里,元类创建类.它们是类的类,你可以把它们想象成这样:

```
MyClass = MetaClass()  
MyObject = MyClass()
```

你已经看到了 `type` 可以让你像这样做:

```
MyClass = type('MyClass', (), {})
```

这是因为 `type` 就是一个元类. `type` 是Python中创建所有类的元类.

现在你可能纳闷为啥子 `type` 用小写而不写成 `Type` ?

我想是因为要跟 `str` 保持一致, `str` 创建字符串对象, `int` 创建整数对象. `type` 正好创建类对象.

你可以通过检查 `__class__` 属性来看到这一点.

Python中所有的东西都是对象.包括整数,字符串,函数还有类.所有这些都是对象.所有这些也都是从类中创建的:

```
>>> age = 35  
>>> age.__class__  
<type 'int'>  
>>> name = 'bob'  
>>> name.__class__  
<type 'str'>  
>>> def foo(): pass  
>>> foo.__class__  
<type 'function'>  
>>> class Bar(object): pass  
>>> b = Bar()  
>>> b.__class__  
<class '__main__.Bar'>
```

那么, `__class__` 的 `__class__` 属性是什么?

```
>>> age.__class__.__class__
<type 'type'>
>>> name.__class__.__class__
<type 'type'>
>>> foo.__class__.__class__
<type 'type'>
>>> b.__class__.__class__
<type 'type'>
```

所以,元类就是创建类对象的东西.

如果你愿意你也可以把它叫做'类工厂'. `type` 是Python的内建元类,当然,你也可以创建你自己的元类.

`__metaclass__` 属性

当你创建一个函数的时候,你可以添加 `__metaclass__` 属性:

```
class Foo(object):
    __metaclass__ = something...
    [...]
```

如果你这么做了,Python就会用元类来创建类Foo.

小心点,这里面有些技巧.

你首先写下 `class Foo(object)`,但是类对象 `Foo` 还没有在内存中创建.

Python将会在类定义中寻找 `__metaclass__`.如果找到了就用它来创建类对象 `Foo`.如果没找到,就会默认用 `type` 创建类.

把下面这段话反复读几次。

当你写如下代码时：

```
class Foo(Bar):
    pass
```

Python将会这样运行:

在 `Foo` 中有没有 `__metaclass__` 属性？

如果有,Python会在内存中通过 `__metaclass__` 创建一个名字为 `Foo` 的类对象(我说的是类对象,跟紧我的思路).

如果Python没有找到 `__metaclass__`,它会继续在`Bar`（父类）中寻找 `__metaclass__`属性,并尝试做和前面同样的操作.

如果Python在任何父类中都找不到 `__metaclass__`，它就会在模块层次中去寻找 `__metaclass__`，并尝试做同样的操作。

如果还是找不到 `__metaclass__`，Python就会用内置的 `type` 来创建这个类对象。

现在的问题就是，你可以在 `__metaclass__` 中放置些什么代码呢？

答案就是：可以创建一个类的东西。

那么什么可以用来创建一个类呢？`type`，或者任何使用到 `type` 或者子类化 `type` 的东东都可以。

自定义元类

元类的主要目的就是为了当创建类时能够自动地改变类。

通常，你会为API做这样的事情，你希望可以创建符合当前上下文的类。

假想一个很傻的例子，你决定在你的模块里所有的类的属性都应该是大写形式。有好几种方法可以办到，但其中一种就是通过模块级别设定 `__metaclass__`。

采用这种方法，这个模块中的所有类都会通过这个元类来创建，我们只需要告诉元类把所有的属性都改成大写形式就万事大吉了。

幸运的是，`__metaclass__` 实际上可以被任意调用，它并不需要是一个正式的类（我知道，某些名字里带有'`class`'的东西并不需要是一个`class`，画画图理解下，这很有帮助）。

所以，我们这里就先以一个简单的函数作为例子开始。

```
# 元类会自动将你通常传给'type'的参数作为自己的参数传入
def upper_attr(future_class_name, future_class_parents, future_class_attr):
    """
    返回一个将属性列表变为大写字母的类对象
    """

    # 选取所有不以'__'开头的属性, 并把它们编程大写
    uppercase_attr = {}
    for name, val in future_class_attr.items():
        if not name.startswith('__'):
            uppercase_attr[name.upper()] = val
        else:
            uppercase_attr[name] = val

    # 用'type'创建类
    return type(future_class_name, future_class_parents, uppercase_attr)

__metaclass__ = upper_attr # 将会影响整个模块

class Foo(): # global __metaclass__ won't work with "object" though
    # 我们也可以只在这里定义__metaclass__，这样就只会作用于这个类中
    bar = 'bip'

print(hasattr(Foo, 'bar'))
# 输出: False
print(hasattr(Foo, 'BAR'))
# 输出: True

f = Foo()
print(f.BAR)
# 输出: 'bip'
```

现在让我们再做一次，这一次用一个真正的class来当做元类。

```
# 请记住，'type'实际上是一个类，就像'str'和'int'一样
# 所以，你可以从type继承
class UpperAttrMetaclass(type):
    # __new__ 是在__init__之前被调用的特殊方法
    # __new__是用来创建对象并返回它的方法
    # 而__init__只是用来将传入的参数初始化给对象
    # 你很少用到__new__，除非你希望能够控制对象的创建
    # 这里，创建的对象是类，我们希望能够自定义它，所以我们这里改写__new__
    # 如果你希望的话，你也可以在__init__中做些事情
    # 还有一些高级的用法会涉及到改写__call__特殊方法，但是我们这里不用
    def __new__(upperattr_metaclass, future_class_name,
                future_class_parents, future_class_attr):

        uppercase_attr = {}
        for name, val in future_class_attr.items():
            if not name.startswith('__'):
                uppercase_attr[name.upper()] = val
            else:
                uppercase_attr[name] = val

        return type(future_class_name, future_class_parents, uppercase_attr)
```

但是这不是真正的面向对象(OOP).我们直接调用了`type`，而且我们没有改写父类的`new`方法。现在让我们这样去处理:

```
class UpperAttrMetaclass(type):

    def __new__(upperattr_metaclass, future_class_name,
                future_class_parents, future_class_attr):

        uppercase_attr = {}
        for name, val in future_class_attr.items():
            if not name.startswith('__'):
                uppercase_attr[name.upper()] = val
            else:
                uppercase_attr[name] = val

        # 重用 type.__new__ 方法
        # 这就是基本的OOP编程，没什么魔法
        return type.__new__(upperattr_metaclass, future_class_name,
                             future_class_parents, uppercase_attr)
```

你可能已经注意到了有个额外的参数 `upperattr_metaclass`，这并没有什么特别的。类方法的第一个参数总是表示当前的实例，就像在普通的类方法中的 `self` 参数一样。

当然了，为了清晰起见，这里的名字我起的比较长。但是就像 `self` 一样，所有的参数都有它们的传统名称。因此，在真实的产品代码中一个元类应该是像这样的：


```
class UpperAttrMetaclass(type):

    def __new__(cls, clsname, bases, dct):

        uppercase_attr = {}
        for name, val in dct.items():
            if not name.startswith('__'):
                uppercase_attr[name.upper()] = val
            else:
                uppercase_attr[name] = val

        return type.__new__(cls, clsname, bases, uppercase_attr)
```

如果使用super方法的话，我们还可以使它变得更清晰一些，这会缓解继承（是的，你可以拥有元类，从元类继承，从type继承）

```
class UpperAttrMetaclass(type):

    def __new__(cls, clsname, bases, dct):

        uppercase_attr = {}
        for name, val in dct.items():
            if not name.startswith('__'):
                uppercase_attr[name.upper()] = val
            else:
                uppercase_attr[name] = val

        return super(UpperAttrMetaclass, cls).__new__(cls, clsname, bases, uppercase_attr)
```

就是这样，除此之外，关于元类真的没有别的可说的了。

使用到元类的代码比较复杂，这背后的原因倒并不是因为元类本身，而是因为你通常会使用元类去做一些晦涩的事情，依赖于自省，控制继承等等。

确实，用元类来搞些“黑暗魔法”是特别有用的，因而会搞出些复杂的东西来。但就元类本身而言，它们其实是很简单的：

- 拦截类的创建
- 修改一个类
- 返回修改之后的类

为什么要用metaclass类而不是函数？

由于__metaclass__可以接受任何可调用的对象，那为何还要使用类呢，因为很显然使用类会更加复杂啊？

这里有好几个原因：

- 意图会更加清晰。当你读到 `UpperAttrMetaclass(type)` 时，你知道接下来要发生什么。
- 你可以使用OOP编程。元类可以从元类中继承而来，改写父类的方法。元类甚至还可以使用元类。
- 你可以把代码组织的更好。当你使用元类的时候肯定不会是像我上面举的这种简单场景，通常都是针对比较复杂的问题。将多个方法归总到一个类中会很有帮助，也会使得代码更容易阅读。
- 你可以使用 `__new__` , `__init__` 以及 `__call__` 这样的特殊方法。它们能帮你处理不同的任务。就算通常你可以把所有的东西都在 `__new__` 里处理掉，有些人还是觉得用 `__init__` 更舒服些。
- 哇哦，这东西的名字是metaclass，肯定非善类，我要小心！

说了这么多TMD究竟为什么要使用元类？

现在回到我们的大主题上来，究竟是为什么你会去使用这样一种容易出错且晦涩的特性？

好吧，一般来说，你根本就用不上它：

“元类就是深度的魔法，99%的用户应该根本不必为此操心。如果你想搞清楚究竟是否需要用到元类，那么你就不需要它。那些实际用到元类的人都非常清楚地知道他们需要什么，而且根本不需要解释为什么要用元类。”—— Python界的领袖 Tim Peters

元类的主要用途是创建API。一个典型的例子是Django ORM。

它允许你像这样定义：

```
class Person(models.Model):
    name = models.CharField(max_length=30)
    age = models.IntegerField()
```

但是如果你像这样做的话：

```
guy = Person(name='bob', age='35')
print(guy.age)
```

这并不会返回一个 `IntegerField` 对象，而是会返回一个int，甚至可以直接从数据库中取出数据。

这是有可能的，因为 `models.Model` 定义了 `__metaclass__`，并且使用了一些魔法能够将你刚定义的简单的Person类转变成对数据库的一个复杂hook。

Django框架将这些看起来很复杂的东西通过暴露出一个简单的使用元类的API将其化简，通过这个API重新创建代码，在背后完成真正的工作。

结语

首先，你知道了类其实是能够创建出类实例的对象。

好吧，事实上，类本身也是实例，当然，它们是元类的实例。

```
>>> class Foo(object): pass
>>> id(Foo)
142639324
```

Python中的一切都是对象，它们要么是类的实例，要么是元类的实例。

除了 `type . type` 实际上是它自己的元类，在纯Python环境中这可不是你能够做到的，这是通过在实现层面耍一些小手段做到的。

其次，元类是很复杂的。对于非常简单的类，你可能不希望通过使用元类来对类做修改。你可以通过其他两种技术来修改类：

- [monkey patching](#)
- 装饰器

当你需要动态修改类时，99%的时间里你最好使用上面这两种技术。当然了，其实在99%的时间里你根本就不需要动态修改类 :D

Copyright © taizilongxu 2018 all right reserved，powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
3	4626	1468573	22	url

在Python中有三元运算符吗？

如果没有,可以像其他语言用的简单方法来实现吗？

有,在2.5版本中加入.对于python初学者可能有点难以理解,所以要记住了.

语法如下:

```
a if test else b
```

根据 `test` 的布尔值来判断返回的是 `a` 还是 `b`;如果 `test` 为真则返回 `a`,反之则返回 `b`.

来个大栗子:

```
>>> 'true' if True else 'false'
'true'
>>> 'true' if False else 'false'
'false'
```

官方文档:

- [Conditional expressions](#)
- [Is there an equivalent of C's "?:" ternary operator?](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
4	4530	3091022	39	url

用Python如何一个文件是否存在？

不用 `try:` 语句可以一个文件存在

如果不确定文件存不存在,可以这样做:

```
import os.path
os.path.isfile(fname)
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

if __name__ == "__main__":有什么作用?

rank	vote	view	answer	url
5	4348	1893044	29	url

if __name__ == "__main__":有什么作用?

```
# Threading example
import time, thread

def myfunction(string, sleeptime, lock, *args):
    while True:
        lock.acquire()
        time.sleep(sleeptime)
        lock.release()
        time.sleep(sleeptime)

if __name__ == "__main__":
    lock = thread.allocate_lock()
    thread.start_new_thread(myfunction, ("Thread #: 1", 2, lock))
    thread.start_new_thread(myfunction, ("Thread #: 2", 2, lock))
```

当 Python 解析器读取一个源文件时, 会执行里面所有的代码.

但是在它执行之前会定义一些特殊的变量.例如,如果 Python 解析器把一个 module(the source file)当成主程序来执行的话, 它会把 `__name__` 的值设置成为 `__main__`.如果这个文件是被其他 module 引入的, `__name__` 的名字将会被设成 module 的名字.

在你给的例子中, 我们假设作为一个主程序来执行, 例如在 shell 里执行如下命令

```
python threading_example.py
```

当设置好特殊变量后, 它会执行 `import` 声明并且开始载入这些 modules.它会执行 `def` 代码块, 创建一个函数对象并且会创建一个 `myfunction` 的变量指向这个函数对象.接下来它会读取 `if` 语句并且判断 `__name__` 是否等于 `__main__`, 如果等于就执行代码块里面的程序.

你想创建一个 module (一个 `.py` 文件), 同时也想能够被其他 module 引用.填入 `if __name__ == "__main__":` 能使你在以这个 module 作为主程序时才会执行, 而其他外部引用不会执行这些代码.

可以在 [这里](#) 查看更多细节

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 : 2018-10-17 17:33:54

if __name__ == "__main__":有什么作用?

rank	vote	view	answer	url
6	3782	2678600	52	url

在Python中调用外部命令?

怎么在Python脚本里调用外部命令?(就好像直接输入在Unix shell中或者windows的命令行里)

来来来,我给你叨咕叨咕各种方法和各自优缺点:

1. `os.system("命令加参数")` 把命令和参数传递给你系统的shell中.用这个命令的好处在于你可以一次运行好多命令还可以设置管道来进行重定向.来个栗子: `os.system("命令 < 出入文件 | 另一个命令 > 输出文件")` 尽管它非常方便,但是你还是不得不手动输入像空格这样的shell字符.从另一方面讲,对于运行简单的shell命令而不去调用外部程序来说的话还是非常好用的.
2. `stream = os.popen("命令和参数")` 这个命令和 `os.system` 差不多,但是它提供了一个连接标准输入/输出的管道.还有其他3个 `popen` 可以调用.如果你传递一个字符串,你的命令会把它传递给shell,如果你传递的是一个列表,那么就不用担心溢出字符了(escaping characters).
3. `subprocess` 模块的管道 `Popen`.这个 `Popen` 是打算用来替代 `os.popen` 的方法,它有点复杂:

```
print subprocess.Popen("echo Hello World", shell=True, stdout=PIPE).stdout.read()
```

而用 `os.popen` :

```
print os.popen("echo Hello World").read()
```

它最大的优点就是一个类里代替了原来的4个不同的 `popen`

4. `subprocess` 的 `call` 方法.它的基本用法和上面的 `Popen` 类参数一致,但是它会等待命令结束后才会返回程序.来个大狸子:

```
return_code = subprocess.call("echo Hello World", shell=True)
```

5. `os`模块里也有C语言里 `fork/exec/spawn` 方法,但是我不建议你直接用它们.

`subprocess` 模块可能更适合你.

最后请注意在你传递到shell的命令一定要注意参数的安全性,给你个提示,看下面代码


```
print subprocess.Popen("echo %s " % user_input, stdout=PIPE).stdout.read()
```

想象一下如果哪个 SB 输入 `my mama didnt love me && rm -rf /`

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
7	3383	1264559	49	url

如何在一个表达式里合并两个字典？

我有两个Python字典,我想写一个表达式来返回两个字典的合并.update()方法返回的是空值而不是返回合并后的对象.

```
>>> x = {'a':1, 'b': 2}
>>> y = {'b':10, 'c': 11}
>>> z = x.update(y)
>>> print z
None
>>> x
{'a': 1, 'b': 10, 'c': 11}
```

怎样才能最终让值保存在z而不是x？

可以用下面的方法:

```
z = dict(x.items() + y.items())
```

最后就是你想要的最终结果保存在字典z中,而键 b 的值会被第二个字典的值覆盖.

```
>>> x = {'a':1, 'b': 2}
>>> y = {'b':10, 'c': 11}
>>> z = dict(x.items() + y.items())
>>> z
{'a': 1, 'c': 11, 'b': 10}
```

如果你用的是Python3的话稍微有点麻烦:

```
>>> z = dict(list(x.items()) + list(y.items()))
>>> z
{'a': 1, 'c': 11, 'b': 10}
```

还可以这样:

```
z = x.copy()  
z.update(y)
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

检查一个文件夹是否存在,如果不存在就创建它

rank	vote	view	answer	url
8	3119	1993725	25	url

检查一个文件夹是否存在,如果不存在就创建它

有什么好的方法吗?这个是我想的:

```
filename = "/my/directory/filename.txt"
dir = os.path.dirname(filename)

try:
    os.stat(dir)
except:
    os.mkdir(dir)

f = file(filename)
```

我忘记了 `os.path.exists` (多谢张三,李四,王五的提醒).下面是更改的:

```
def ensure_dir(f):
    d = os.path.dirname(f)
    if not os.path.exists(d):
        os.makedirs(d)
```

有什么"打开"的标记可以自动的运行?

我看了俩答案都很好,但是都有一点缺陷,所以给出我的:

先试 `os.path.exists` ,然后通过 `os.makedirs` 来创建.

```
if not os.path.exists(directory):
    os.makedirs(directory)
```

标注一下-如果一个文件在调用 `os.path.exists` 和 `os.makedirs` 之间被创建了,将会出现一个 `OSError` .遗憾的是捕获 `OSError` 异常继续进行并不是万无一失的,它将会忽略像磁盘空间不足,没有足够权限等一些其他造成文件创建失败的因素.

一个做法是捕获 `OSError` 异常并检查返回的错误代码(前提是知道错误代码对应的是什么).然而,还有另一种可能,第二次的 `os.path.exists` .假如恰好在第一次检查的时候创建了文件夹,然后在第二次检查的时候删掉--我们被耍了~~

根据不同的应用,并行操作的危险或多或少的比其他因素危险.开发者必须在选择开发环境的时候更多地了解特定的应用程序.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
9	3072	2877536	14	url

Python中有检查字符串包含的方法吗？

我正在找 `string.contains` 或者 `string.indexof` 方法.

我希望:

```
if not somestring.contains("blah"):
    continue
```

你可以用 `in` 啊:

```
if not "blah" in somestring: continue
```

或者:

```
if "blah" not in somestring: continue
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
10	3039	1872959	41	url

用字典的值(key)对字典进行排序

我有个字典,字典的值来自于数据库:一个是字符串,一个是数字.字符串是唯一的,所以键(key)就是字符串.

我可以用键(key)来排序,但是怎么用值(value)来排序呢?

注:我已经看过另一个问题[怎样对列表中的字典的键值\(key\)对字典进行排序?](#),或许这种方法可以,但是我确实只需要一个字典,我想看看还有其他更好的方法.

对字典进行排序是不可能的,只有把字典转换成另一种方式才能排序.字典本身是无序的,但是像列表元组等其他类型是有序的.所以你需要用一个元组列表来表示排序的字典.

例子:

```
import operator
x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
sorted_x = sorted(x.items(), key=operator.itemgetter(1))
```

`sorted_x` 是一个元组列表,用每个元组的第二个元素进行排序. `dict(sorted_x) == x` .

如果想要用键(key)来进行排序:

```
import operator
x = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
sorted_x = sorted(x.items(), key=operator.itemgetter(0))
```

你也可以:

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


rank	vote	view	answer	url
11	2967	2634401	32	url

在Python中列出目录中的所有文件

怎么样用Python列出一个目录的所有文件并且存进一个列表？

`os.listdir()` 可以获得一个目录中所有文件或者子目录。

如果你只想要文件的话,你也可以用 `os.path` 把其他的过滤掉:

```
from os import listdir
from os.path import isfile, join
onlyfiles = [ f for f in listdir(mypath) if isfile(join(mypath,f)) ]
```

或者你可以用 `os.walk()` ,它遍历每个目录将会返回两个列表(一个文件列表,一个目录列表),如果你想要顶层目录只需要在第一次迭代后`break`一下即可。

```
from os import walk

f = []
for (dirpath, dirnames, filenames) in walk(mypath):
    f.extend(filenames)
    break
```

最后,如果你想增加列表可以像上面那样用 `.extend()` 或者:

```
>>> q = [1, 2, 3]
>>> w = [4, 5, 6]
>>> q = q + w
>>> q
[1, 2, 3, 4, 5, 6]
```

我个人更喜欢 `.extend()` # 在Python中列出目录中的所有文件

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
12	2862	1856471	30	url

检查列表是否为空的最好方法

例如,传递下面:

```
a = []
```

我怎么检查 `a` 是空值?

```
if not a:  
    print "List is empty"
```

用隐藏的空列表的布尔值才是最Pythonic的方法.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
13	2794	588058	24	url

装饰器@staticmethod和@classmethod有什么区别？

也许一些例子会有帮助:注意 `foo` , `class_foo` 和 `static_foo` 参数的区别:

```
class A(object):
    def foo(self,x):
        print "executing foo(%s,%s)"%(self,x)

    @classmethod
    def class_foo(cls,x):
        print "executing class_foo(%s,%s)"%(cls,x)

    @staticmethod
    def static_foo(x):
        print "executing static_foo(%s)"%x

a=A()
```

下面是一个对象实体调用方法的常用方式.对象实体 `a` 被隐藏的传递给了第一个参数.

```
a.foo(1)
# executing foo(<__main__.A object at 0xb7dbef0c>,1)
```

用 **classmethods** 装饰,隐藏的传递给第一个参数的是对象实体的类(`class A`)而不是 `self` .

```
a.class_foo(1)
# executing class_foo(<class '__main__.A'>,1)
```

你也可以用类调用 `class_foo` .实际上,如果你把一些方法定义成 `classmethod` ,那么实际上你是希望用类来调用这个方法,而不是用这个类的实例来调用这个方法. `A.foo(1)` 将会返回一个 `TypeError` 错误, `A.class_foo(1)` 将会正常运行:

```
A.class_foo(1)
# executing class_foo(<class '__main__.A'>,1)
```

One use people have found for class methods is to create inheritable alternative constructors.

用**staticmethods**来装饰,不管传递给第一个参数的是 `self` (对象实体)还是 `cls` (类).它们的表现都一样:

```
a.static_foo(1)
# executing static_foo(1)

A.static_foo('hi')
# executing static_foo(hi)
```

静态方法被用来组织类之间有逻辑关系的函数.

`foo` 只是个函数,但是当你调用 `a.foo` 的时候你得到的不仅仅是一个函数,你得到的是一个第一个参数绑定到 `a` 的"加强版"函数. `foo` 需要两个参数,而 `a.foo` 仅仅需要一个参数.

`a` 绑定了 `foo`.下面可以知道什么叫"绑定"了:

```
print(a.foo)
# <bound method A.foo of <__main__.A object at 0xb7d52f0c>>
```

如果使用 `a.class_foo`,是 `A` 绑定到了 `class_foo` 而不是 `a`.

```
print(a.class_foo)
# <bound method type.class_foo of <class '__main__.A'>>
```

最后剩下静态方法,说到底它就是一个方法. `a.static_foo` 只是返回一个不带参数绑定的方法. `static_foo` 和 `a.static_foo` 只需要一个参数.

```
print(a.static_foo)
# <function static_foo at 0xb7d479cc>
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
14	2787	2286116	24	url

Python 中的 append 和 extend

append 和 extend 有什么区别？

append:

```
x = [1, 2, 3]
x.append([4, 5])
print (x)
```

输出: [1, 2, 3, [4, 5]]

extend:

```
x = [1, 2, 3]
x.extend([4, 5])
print (x)
```

输出: [1, 2, 3, 4, 5]

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook 该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
15	2680	1455704	21	url

在循环中获取索引(数组下标)

有人知道如何获取列表的索引值吗:

```
ints = [8, 23, 45, 12, 78]
```

当我循环这个列表时如何获得它的索引下标?

如果像C或者PHP那样加入一个状态变量那就太不pythonic了.

最好的选择就是用内建函数[enumerate](#)

```
for idx, val in enumerate(ints):  
    print idx, val
```

想了解更多可以查看[PEP279].在Python2.x和Python3.x都好使.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
16	2554	2740443	18	url

如何在一个函数里用全局变量？

如果我在一个函数里建了一个全局变量,那么我怎么在另一个函数里使用这个全局变量？

我需要把这个全局变量赋值给这个函数的局部变量吗？

如果你要在别的函数里使用全局变量,只要在被调用全局变量函数的里事先用 `global` 声明一下:

```
globvar = 0

def set_globvar_to_one():
    global globvar    # 需要用global修饰一下globvar
    globvar = 1

def print_globvar():
    print globvar      # 如果要读globvar的值的话不需要用global修饰

set_globvar_to_one()
print_globvar()       # 输出 1
```

我猜正是因为全局变量比较危险,所以Python为了确保你真的知道它是全局变量,所以需要加一个 `global` 关键字.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
17	2428	444941	15	url

Python中如何在一个函数中加入多个装饰器？

怎么做才能让一个函数同时用两个装饰器,像下面这样:

```
@makebold
@makeitalic
def say():
    return "Hello"
```

我希望得到

```
<b><i>Hello</i></b>
```

我只是想知道装饰器怎么工作的!

去看看[文档](#),答在下面:

```
def makebold(fn):
    def wrapped():
        return "<b>" + fn() + "</b>"
    return wrapped

def makeitalic(fn):
    def wrapped():
        return "<i>" + fn() + "</i>"
    return wrapped

@makebold
@makeitalic
def hello():
    return "hello world"

print hello() ## returns <b><i>hello world</i></b>
```

Answer:2

如果你不想看详细的解释的话请看上面那个答案.

装饰器基础

Python的函数都是对象

要了解装饰器,你必须了解Python中的函数都是对象.这个意义非常重要.让我们看看一个简单例子:

```
def shout(word="yes"):
    return word.capitalize()+"!"

print shout()
# 输出 : 'Yes!'

# 作为一个对象,你可以把它赋值给任何变量

scream = shout

# 注意啦我们没有加括号,我们并不是调用这个函数,我们只是把函数"shout"放在了变量"scream"里.
# 也就是说我们可以通过"scream"调用"shout":

print scream()
# 输出 : 'Yes!'

# 你可以删除旧名"shout",而且"scream"依然指向函数

del shout
try:
    print shout()
except NameError, e:
    print e
    #输出: "name 'shout' is not defined"

print scream()
# 输出: 'Yes!'
```

好了,先记住上面的,一会还会用到.

Python函数另一个有趣的特性就是你可以在一个函数里定义另一个函数!

```
def talk():  
    # 你可以在"talk"里定义另一个函数 ...  
    def whisper(word="yes"):  
        return word.lower()+"..."  
  
    # 让我们用用它!  
  
    print whisper()  
  
# 每次调用"talk"时都会定义一次"whisper",然后"talk"会调用"whisper"  
talk()  
# 输出:  
# "yes..."  
  
# 但是在"talk"意外"whisper"是不存在的:  
  
try:  
    print whisper()  
except NameError, e:  
    print e  
#输出 : "name 'whisper' is not defined"
```

函数引用

好,终于到了有趣的地方了...

已经知道函数就是对象.因此,对象:

- 可以赋值给一个变量
- 可以在其他函数里定义

这就意味着函数可以返回另一个函数.来看看!☺

```
def getTalk(kind="shout"):

    # 在函数里定义一个函数
    def shout(word="yes"):
        return word.capitalize()+"!"

    def whisper(word="yes") :
        return word.lower()+"...";

    # 返回一个函数
    if kind == "shout":
        # 这里不用"()",我们不是要调用函数
        # 只是返回函数对象
        return shout
    else:
        return whisper

# 怎么用这个特性呢?

# 把函数赋值给变量
talk = getTalk()

# 可以看到"talk"是一个函数对象
print talk
# 输出 : <function shout at 0xb7ea817c>

# 函数返回的是对象:
print talk()
# 输出 : Yes!

# 不嫌麻烦你也可以这么用
print getTalk("whisper")()
# 输出 : yes...
```

既然可以 `return` 一个函数,你也可以在把函数作为参数传递:

```
def doSomethingBefore(func):
    print "I do something before then I call the function you gave me"
    print func()

doSomethingBefore(scream)
# 输出:
#I do something before then I call the function you gave me
#Yes!
```

学习装饰器的基本知识都在上面了.装饰器就是"wrappers",它可以让你在你装饰函数之前或之后执行程序,而不用修改函数本身.

自己动手实现装饰器

怎么样自己做呢:

```
# 装饰器就是把其他函数作为参数的函数
def my_shiny_new_decorator(a_function_to_decorate):

    # 在函数里面,装饰器在运行中定义函数: 包装.
    # 这个函数将被包装在原始函数的外面,所以可以在原始函数之前和之后执行其他代码..
    def the_wrapper_around_the_original_function():

        # 把要在原始函数被调用前的代码放在这里
        print "Before the function runs"

        # 调用原始函数(用括号)
        a_function_to_decorate()

        # 把要在原始函数调用后的代码放在这里
        print "After the function runs"

    # 在这里"a_function_to_decorate" 函数永远不会被执行
    # 在这里返回刚才包装过的函数
    # 在包装函数里包含要在原始函数前后执行的代码.
    return the_wrapper_around_the_original_function

# 加入你建了个函数,不想修改了
def a_stand_alone_function():
    print "I am a stand alone function, don't you dare modify me"

a_stand_alone_function()
#输出: I am a stand alone function, don't you dare modify me

# 现在,你可以装饰它来增加它的功能
# 把它传递给装饰器,它就会返回一个被包装过的函数.

a_stand_alone_function_decorated = my_shiny_new_decorator(a_stand_alone_function)
a_stand_alone_function_decorated()
#输出s:
#Before the function runs
#I am a stand alone function, don't you dare modify me
#After the function runs
```

现在,你或许每次都想用 `a_stand_alone_function_decorated` 代替 `a_stand_alone_function`,很简单,只需要用 `my_shiny_new_decorator` 返回的函数重写 `a_stand_alone_function`:

```
a_stand_alone_function = my_shiny_new_decorator(a_stand_alone_function)
a_stand_alone_function()
#输出:
#Before the function runs
#I am a stand alone function, don't you dare modify me
#After the function runs

# 想到了吗,这就是装饰器干的事!
```

让我们看看装饰器的真实面纱

用上一个例子,看看装饰器的语法:

```
@my_shiny_new_decorator
def another_stand_alone_function():
    print "Leave me alone"

another_stand_alone_function()
#输出:
#Before the function runs
#Leave me alone
#After the function runs
```

就这么简单. `@decorator` 就是下面的简写:

```
another_stand_alone_function = my_shiny_new_decorator(another_stand_alone_function)
```

装饰器就是 [decorator design pattern](#) 的pythonic的变种.在Python中有许多经典的设计模式来满足开发者.

当然,你也可以自己写装饰器:

```
def bread(func):
    def wrapper():
        print "</''''''\>"
        func()
        print "<\_____/>"
    return wrapper

def ingredients(func):
    def wrapper():
        print "#tomatoes#"
        func()
        print "~salad~"
    return wrapper

def sandwich(food="--ham--"):
    print food

sandwich()
#outputs: --ham--
sandwich = bread(ingredients(sandwich))
sandwich()
#outputs:
#</''''''\>
# #tomatoes#
# --ham--
# ~salad~
#<\_____/>
```

用Python装饰器语法糖:

```
@bread
@ingredients
def sandwich(food="--ham--"):
    print food

sandwich()
#outputs:
#<''''''\>
# #tomatoes#
# --ham--
# ~salad~
#<_____/>
```

改变一下顺序:

```
@ingredients
@bread
def strange_sandwich(food="--ham--"):
    print food

strange_sandwich()
#outputs:
##tomatoes#
#<''''''\>
# --ham--
#<_____/>
# ~salad~
```

现在:回答你的问题...

作为结论,相信你现在已经知道答案了:

```
# 字体变粗装饰器
def makebold(fn):
    # 装饰器将返回新的函数
    def wrapper():
        # 在之前或者之后插入新的代码
        return "<b>" + fn() + "</b>"
    return wrapper

# 斜体装饰器
def makeitalic(fn):
    # 装饰器将返回新的函数
    def wrapper():
        # 在之前或者之后插入新的代码
        return "<i>" + fn() + "</i>"
    return wrapper

@makebold
@makeitalic
def say():
    return "hello"

print say()
#输出: <b><i>hello</i></b>

# 这相当于
def say():
    return "hello"
say = makebold(makeitalic(say))

print say()
#输出: <b><i>hello</i></b>
```

别轻松太早,看看下面的高级用法

装饰器高级用法

在装饰器函数里传入参数

```
# 这不是什么黑魔法,你只需要让包装器传递参数:

def a_decorator_passing_arguments(function_to_decorate):
    def a_wrapper_accepting_arguments(arg1, arg2):
        print "I got args! Look:", arg1, arg2
        function_to_decorate(arg1, arg2)
    return a_wrapper_accepting_arguments

# 当你调用装饰器返回的函数时,也就调用了包装器,把参数传入包装器里,
# 它会将参数传递给被装饰的函数里。

@a_decorator_passing_arguments
def print_full_name(first_name, last_name):
    print "My name is", first_name, last_name

print_full_name("Peter", "Venkman")
# 输出:
#I got args! Look: Peter Venkman
#My name is Peter Venkman
```

装饰方法

在Python里方法和函数几乎一样.唯一的区别就是方法的第一个参数是一个当前对象的(`self`)

也就是说你可以用同样的方式来装饰方法!只要记得把 `self` 加进去:

```
def method_friendly_decorator(method_to_decorate):
    def wrapper(self, lie):
        lie = lie - 3 # 女性福音 :-)
        return method_to_decorate(self, lie)
    return wrapper

class Lucy(object):

    def __init__(self):
        self.age = 32

    @method_friendly_decorator
    def sayYourAge(self, lie):
        print "I am %s, what did you think?" % (self.age + lie)

l = Lucy()
l.sayYourAge(-3)
#输出: I am 26, what did you think?
```

如果你想造一个更通用的可以同时满足方法和函数的装饰器,用 `*args, **kwargs` 就可以了

```
def a_decorator_passing_arbitrary_arguments(function_to_decorate):
    # 包装器接受所有参数
    def a_wrapper_accepting_arbitrary_arguments(*args, **kwargs):
```



```

    print "Do I have args?:"
    print args
    print kwargs
    # 现在把*args, **kwargs解包
    # 如果你不明白什么是解包的话, 请查阅:
    # http://www.saltycrane.com/blog/2008/01/how-to-use-args-and-kwargs-in-python/
    function_to_decorate(*args, **kwargs)
    return a_wrapper_accepting_arbitrary_arguments

@a_decorator_passing_arbitrary_arguments
def function_with_no_argument():
    print "Python is cool, no argument here."

function_with_no_argument()
#输出
#Do I have args?:
#()
#{ }
#Python is cool, no argument here.

@a_decorator_passing_arbitrary_arguments
def function_with_arguments(a, b, c):
    print a, b, c

function_with_arguments(1, 2, 3)
#输出
#Do I have args?:
#(1, 2, 3)
#{ }
#1 2 3

@a_decorator_passing_arbitrary_arguments
def function_with_named_arguments(a, b, c, platypus="Why not ?"):
    print "Do %s, %s and %s like platypus? %s" %\
        (a, b, c, platypus)

function_with_named_arguments("Bill", "Linus", "Steve", platypus="Indeed!")
#输出
#Do I have args ? :
#('Bill', 'Linus', 'Steve')
#{'platypus': 'Indeed!'}
#Do Bill, Linus and Steve like platypus? Indeed!

class Mary(object):

    def __init__(self):
        self.age = 31

    @a_decorator_passing_arbitrary_arguments
    def sayYourAge(self, lie=-3): # 可以加入一个默认值
        print "I am %s, what did you think ?" % (self.age + lie)

m = Mary()
m.sayYourAge()
#输出
# Do I have args?:

```

```
#__main__.Mary object at 0xb7d303ac>,)
#{ }
#I am 28, what did you think?
```

把参数传递给装饰器

好了,如何把参数传递给装饰器自己?

因为装饰器必须接收一个函数当做参数,所以有点麻烦.好吧,你不可以直接把被装饰函数的参数传递给装饰器.

在我们考虑这个问题时,让我们重新回顾下:

```
# 装饰器就是一个'平常不过'的函数
def my_decorator(func):
    print "I am an ordinary function"
    def wrapper():
        print "I am function returned by the decorator"
        func()
    return wrapper

# 因此你可以不用"@"也可以调用他

def lazy_function():
    print "zzzzzzzz"

decorated_function = my_decorator(lazy_function)
#输出: I am an ordinary function

# 之所以输出 "I am an ordinary function"是因为你调用了函数,
# 并非什么魔法.

@my_decorator
def lazy_function():
    print "zzzzzzzz"

#输出: I am an ordinary function
```

看见了吗,和" my_decorator "一样只是被调用.所以当你用 @my_decorator 你只是告诉Python去掉用被变量 my_decorator 标记的函数.

这非常重要!你的标记能直接指向装饰器.

让我们做点邪恶的事.☺

```
def decorator_maker():

    print "I make decorators! I am executed only once: "+\
        "when you make me create a decorator."

    def my_decorator(func):

        print "I am a decorator! I am executed only when you decorate a function."

        def wrapped():
            print ("I am the wrapper around the decorated function. "
                  "I am called when you call the decorated function. "
                  "As the wrapper, I return the RESULT of the decorated function.")
            return func()

        print "As the decorator, I return the wrapped function."

        return wrapped

    print "As a decorator maker, I return a decorator"
    return my_decorator

# 让我们建一个装饰器.它只是一个新函数.
new_decorator = decorator_maker()
#输出:
#I make decorators! I am executed only once: when you make me create a decorator.
#As a decorator maker, I return a decorator

# 下面来装饰一个函数

def decorated_function():
    print "I am the decorated function."

decorated_function = new_decorator(decorated_function)
#输出:
#I am a decorator! I am executed only when you decorate a function.
#As the decorator, I return the wrapped function

# Let's call the function:
decorated_function()
#输出:
#I am the wrapper around the decorated function. I am called when you call the decorat
ed function.
#As the wrapper, I return the RESULT of the decorated function.
#I am the decorated function.
```

一点都不难把.

下面让我们去掉所有可恶的中间变量:

```
def decorated_function():
    print "I am the decorated function."
decorated_function = decorator_maker()(decorated_function)
#输出:
#I make decorators! I am executed only once: when you make me create a decorator.
#As a decorator maker, I return a decorator
#I am a decorator! I am executed only when you decorate a function.
#As the decorator, I return the wrapped function.

# 最后:
decorated_function()
#输出:
#I am the wrapper around the decorated function. I am called when you call the decorated function.
#As the wrapper, I return the RESULT of the decorated function.
#I am the decorated function.
```

让我们简化一下:

```
@decorator_maker()
def decorated_function():
    print "I am the decorated function."
#输出:
#I make decorators! I am executed only once: when you make me create a decorator.
#As a decorator maker, I return a decorator
#I am a decorator! I am executed only when you decorate a function.
#As the decorator, I return the wrapped function.

#最终:
decorated_function()
#输出:
#I am the wrapper around the decorated function. I am called when you call the decorated function.
#As the wrapper, I return the RESULT of the decorated function.
#I am the decorated function.
```

看到了吗?我们用一个函数调用"@"语法!:-)

所以让我们回到装饰器的.如果我们在函数运行过程中动态生成装饰器,我们是不是可以把参数传递给函数?

```

def decorator_maker_with_arguments(decorator_arg1, decorator_arg2):

    print "I make decorators! And I accept arguments:", decorator_arg1, decorator_arg2

    def my_decorator(func):
        # 这里传递参数的能力是借鉴了 closures.
        # 如果对closures感到困惑可以看看下面这个:
        # http://stackoverflow.com/questions/13857/can-you-explain-closures-as-they-re
late-to-python
        print "I am the decorator. Somehow you passed me arguments:", decorator_arg1,
decorator_arg2

        # 不要忘了装饰器参数和函数参数!
        def wrapped(function_arg1, function_arg2) :
            print ("I am the wrapper around the decorated function.\n"
                  "I can access all the variables\n"
                  "\t- from the decorator: {0} {1}\n"
                  "\t- from the function call: {2} {3}\n"
                  "Then I can pass them to the decorated function"
                  .format(decorator_arg1, decorator_arg2,
                          function_arg1, function_arg2))
            return func(function_arg1, function_arg2)

        return wrapped

    return my_decorator

@decorator_maker_with_arguments("Leonard", "Sheldon")
def decorated_function_with_arguments(function_arg1, function_arg2):
    print ("I am the decorated function and only knows about my arguments: {0}"
          " {1}".format(function_arg1, function_arg2))

decorated_function_with_arguments("Rajesh", "Howard")
#输出:
#I make decorators! And I accept arguments: Leonard Sheldon
#I am the decorator. Somehow you passed me arguments: Leonard Sheldon
#I am the wrapper around the decorated function.
#I can access all the variables
#  - from the decorator: Leonard Sheldon
#  - from the function call: Rajesh Howard
#Then I can pass them to the decorated function
#I am the decorated function and only knows about my arguments: Rajesh Howard

```

好了,上面就是带参数的装饰器.参数可以设置成变量:

```
c1 = "Penny"
c2 = "Leslie"

@decorator_maker_with_arguments("Leonard", c1)
def decorated_function_with_arguments(function_arg1, function_arg2):
    print ("I am the decorated function and only knows about my arguments:"
           " {0} {1}".format(function_arg1, function_arg2))

decorated_function_with_arguments(c2, "Howard")
#输出:
#I make decorators! And I accept arguments: Leonard Penny
#I am the decorator. Somehow you passed me arguments: Leonard Penny
#I am the wrapper around the decorated function.
#I can access all the variables
# - from the decorator: Leonard Penny
# - from the function call: Leslie Howard
#Then I can pass them to the decorated function
#I am the decorated function and only knows about my arguments: Leslie Howard
```

你可以用这个小技巧把任何函数的参数传递给装饰器.如果你愿意还可以用 `*args, **kwargs` .但是一定要记住了装饰器只能被调用一次.当Python载入脚本后,你不可以动态的设置参数了.当你运行 `import x` ,函数已经被装饰,所以你什么都不能动了.

来练习一下:装饰装饰器

好吧,作为奖励,我就给你讲讲如何怎么让所有的装饰器接收任何参数.为了接收参数,我们用另外的函数来建我们的装饰器.

我们包装装饰器.

还有什么我们可以看到吗?

对了,装饰器!

让我们来为装饰器一个装饰器:

```
def decorator_with_args(decorator_to_enhance):
    """
    这个函数将被用来作为装饰器。
    它必须去装饰要成为装饰器的函数。
    休息一下。
    它将允许所有的装饰器可以接收任意数量的参数,所以以后你不必为每次都要做这个头疼了。
    saving you the headache to remember how to do that every time.
    """

    # 我们用传递参数的同样技巧。
    def decorator_maker(*args, **kwargs):

        # 我们动态的建立一个只接收一个函数的装饰器,
        # 但是他能接收来自maker的参数
        def decorator_wrapper(func):

            # 最后我们返回原始的装饰器,毕竟它只是'平常'的函数
            # 唯一的陷阱:装饰器必须有这个特殊的,否则将不会奏效。
            return decorator_to_enhance(func, *args, **kwargs)

        return decorator_wrapper
```

下面是如何用它们:

```
# 下面的函数是你建来当装饰器用的,然后把装饰器加到上面:-)
# 不要忘了这个 "decorator(func, *args, **kwargs)"
@decorator_with_args
def decorated_decorator(func, *args, **kwargs):
    def wrapper(function_arg1, function_arg2):
        print "Decorated with", args, kwargs
        return func(function_arg1, function_arg2)
    return wrapper

# 现在你用你自己的装饰装饰器来装饰你的函数(汗~~~)

@decorated_decorator(42, 404, 1024)
def decorated_function(function_arg1, function_arg2):
    print "Hello", function_arg1, function_arg2

decorated_function("Universe and", "everything")
#输出:
#Decorated with (42, 404, 1024) {}
#Hello Universe and everything

# Whoooot!
```

估计你看到这和你刚看完爱因斯坦相对论差不多,但是现在如果明白怎么用就好多了吧.

最好的练习:装饰器

- 装饰器是Python2.4里引进的,所以确保你的Python解析器的版本 ≥ 2.4

- 装饰器使函数调用变慢了.一定要记住.
- 装饰器不能被取消(有些人把装饰器做成可以移除的但是没有人会用)所以一旦一个函数被装饰了.所有的代码都会被装饰.
- 用装饰器装饰函数将会很难debug(在 ≥ 2.5 版本将会有所改善;看下面)

`functools` 模块在2.5被引进.它包含了一个 `functools.wraps()` 函数,可以复制装饰器函数的名字,模块和文档给它的包装器.

(事实上: `functools.wraps()` 是一个装饰器!☺)

```
#为了debug, 堆栈跟踪将会返回函数的 __name__
def foo():
    print "foo"

print foo.__name__
#输出: foo

# 如果加上装饰器, 将变得有点复杂
def bar(func):
    def wrapper():
        print "bar"
        return func()
    return wrapper

@bar
def foo():
    print "foo"

print foo.__name__
#输出: wrapper

# "functools" 将有所帮助

import functools

def bar(func):
    # 我们所说的"wrapper", 正在包装 "func",
    # 好戏开始了
    @functools.wraps(func)
    def wrapper():
        print "bar"
        return func()
    return wrapper

@bar
def foo():
    print "foo"

print foo.__name__
#输出: foo
```

怎么使用装饰器？

现在遇到了大问题:我们用装饰器干什么?

看起来很黄很暴力,但是如果有实际用途就更好了.好了这里有1000个用途.传统的用法就是用它来为外部的库的函数(你不能修改的)做扩展,或者debug(你不想修改它,因为它是暂时的).

你也可以用DRY的方法去扩展一些函数,像:

```
def benchmark(func):
    """
    A decorator that prints the time a function takes
    to execute.
    """
    import time
    def wrapper(*args, **kwargs):
        t = time.clock()
        res = func(*args, **kwargs)
        print func.__name__, time.clock()-t
        return res
    return wrapper

def logging(func):
    """
    A decorator that logs the activity of the script.
    (it actually just prints it, but it could be logging!)
    """
    def wrapper(*args, **kwargs):
        res = func(*args, **kwargs)
        print func.__name__, args, kwargs
        return res
    return wrapper

def counter(func):
    """
    A decorator that counts and prints the number of times a function has been execute
    d
    """
    def wrapper(*args, **kwargs):
        wrapper.count = wrapper.count + 1
        res = func(*args, **kwargs)
        print "{0} has been used: {1}x".format(func.__name__, wrapper.count)
        return res
    wrapper.count = 0
    return wrapper

@counter
@benchmark
@logging
def reverse_string(string):
    return str(reversed(string))

print reverse_string("Able was I ere I saw Elba")
print reverse_string("A man, a plan, a canoe, pasta, heros, rajahs, a coloratura, maps
```

```
, snipe, percale, macaroni, a gag, a banana bag, a tan, a tag, a banana bag again (or
a camel), a crepe, pins, Spam, a rut, a Rolo, cash, a jar, sore hats, a peon, a canal:
Panama!")

#输出:
#reverse_string ('Able was I ere I saw Elba',) {}
#wrapper 0.0
#wrapper has been used: 1x
#ablE was I ere I saw elbA
#reverse_string ('A man, a plan, a canoe, pasta, heros, rajahs, a coloratura, maps, sn
ipe, percale, macaroni, a gag, a banana bag, a tan, a tag, a banana bag again (or a ca
mel), a crepe, pins, Spam, a rut, a Rolo, cash, a jar, sore hats, a peon, a canal: Pan
ama!',) {}
#wrapper 0.0
#wrapper has been used: 2x
#!amanaP :lanac a ,noep a ,stah eros ,raj a ,hsac ,oloR a ,tur a ,mapS ,snip ,eperc a
,)lemac a ro( niaga gab ananab a ,gat a ,nat a ,gab ananab a ,gag a ,inoracam ,elacrep
,epins ,spam ,arutaroloc a ,shajar ,soreh ,atsap ,eonac a ,nalp a ,nam A
```

当然,装饰器的好处就是你可以用它们来做任何事而不用重写,DRY:

```
@counter
@benchmark
@logging
def get_random_futurama_quote():
    from urllib import urlopen
    result = urlopen("http://subfusion.net/cgi-bin/quote.pl?quote=futurama").read()
    try:
        value = result.split("<br><b><hr><br>")[1].split("<br><br><hr>")[0]
        return value.strip()
    except:
        return "No, I'm ... doesn't!"

print get_random_futurama_quote()
print get_random_futurama_quote()

#输出:
#get_random_futurama_quote () {}
#wrapper 0.02
#wrapper has been used: 1x
#The laws of science be a harsh mistress.
#get_random_futurama_quote () {}
#wrapper 0.01
#wrapper has been used: 2x
#Curse you, merciful Poseidon!
```

Python自身提供了几个装饰器,像 `property` , `staticmethod` .

- Django用装饰器管理缓存和试图的权限.
- Twisted用来修改异步函数的调用.

好大的坑!

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
18	2396	1241133	31	url

理解Python切片

对于Python的切片有什么好的资料吗?对我来说理解切片有点难度.它看起来非常有用,但是我还是不能理解它,我正在找有什么好的资料.

非常简单:

```
a[start:end] # 从start开始到end-1结束
a[start:]    # 从start开始直到末尾
a[:end]      # 从头部开始直到end结束
a[:]         # 复制整个列表
```

这里还有一个 `step` 值,可以用在上面素有例子的后面:

```
a[start:end:step] # 按照step步长直到end-1结束,并不是从start一个个遍历到end
```

重点需要记住的是: `:end` 值代表的是不被选中的第一个位置.所以 `end` 和 `start` 区别在于选中的元素(如果 `step` 默认为1)

另一点要说的是 `start` 或者 `end` 可能是个负数,也就是从尾部而不是从头部开始计数.所以:

```
a[-1]        # 列表最后一个元素
a[-2:]       # 列表最后两个元素
a[:-2]       # 除了最后两个元素剩下的部分
```

如果你调用的元素多于列表中含有的元素个数,Python也会很友好的表示.例如,如果你请求 `a[:-2]` 而 `a` 只含有一个元素,你得到的是一个空列表而不是一个错误信息.有些时候你可能更希望得到这个错误信息,所以你得意识到上面的事可能发生.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
19	2360	2761878	27	url

查找列表中某个元素的下标

比如 `["foo", "bar", "baz"]` 和它的一个元素 `"bar"` ,用什么方法能找到它的下标(1)?

```
>>> ["foo", "bar", "baz"].index('bar')
1
```

参考资料:[Data Structures > More on Lists](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
20	2311	2380401	19	url

检查一个键在字典中是否存在

在更新字典之前想检查键是否存在.我写了如下代码:

```
if 'key1' in dict.keys():
    print "blah"
else:
    print "boo"
```

我想这不是最好的方法,还有什么更好的方法?

用 `in` .

```
d = dict()

for i in xrange(100):
    key = i % 10
    if key in d:
        d[key] += 1
    else:
        d[key] = 1
```

如果你想要一个默认值,你可以用 `dict.get()` :

```
d = dict()

for i in xrange(100):
    key = i % 10
    d[key] = d.get(key, 0) + 1
```

如果相对所有值设置默认值可以用 `collections` 模块的 `defaultdict` 函数:

```
from collections import defaultdict

d = defaultdict(lambda: 0)

for i in xrange(100):
    d[i % 10] += 1
```

但是总而言之 `in` 是最好的方法.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
21	2273	2487553	34	url

怎么在windows下安装pip?

怎么在windows下安装pip?

Python3.4+

好消息,Python3.4已经自带Pip.这是所有Python发行版中最好的特性了.方便了所有人使用公共库.新手再也不用操心安装额外库的繁琐步骤了.它自带的包管理器中加入了 Ruby, Nodejs, Haskell, Perl, Go等其他几乎所有的开源社区流行语言.谢谢Python.

当然,这并不意味着所有的Python包问题已经解决.在一段时间看来仍然不乐观.我在[Python有包管理系统吗?](#)也讨论过这个问题.

同样对于Python2.x用户(几乎一般人),还没有计划在Python中自带Pip. 只能自己动手了.

Python 2.x 和 Python ≤ 3.3

尽管Python吹的 `简单易用` 的哲学,但是Python不提供包管理工具.更糟糕的是,Pip直到现在还是非常的难以安装.

官方指南

在 <http://www.pip-installer.org/en/latest/installing.html> 下载 `get-pip.py` ,把它保存下来注意不要把 `.py` 后缀改成 `.txt` .然后在命令提示符上输入:

```
python get-pip.py
```

你可能还需要管理员权限来执行它.跟着做 [http://technet.microsoft.com/en-us/library/cc947813\(v=ws.10\).aspx](http://technet.microsoft.com/en-us/library/cc947813(v=ws.10).aspx)

另一种方法

官方文档告诉我们安装Pip和各种依赖的源.对有经验的人来说太麻烦了,对于新手又有点难.

Christoph Gohlke已经为我们做好了下载Python包的安装器(.msi).它可以为的Python版本建立依赖,不管32bit还是64bit.你只需要:

- 安装setuptools <http://www.lfd.uci.edu/~gohlke/pythonlibs/#setuptools>
- 安装pip <http://www.lfd.uci.edu/~gohlke/pythonlibs/#pip>

在我这,Pip安装到了 `C:\Python27\Scripts\pip.exe` .在你的电脑上找到 `pip.exe` ,然后把文件夹(eg. `C:\Python27\Scripts`)添加你的路径(编辑你的环境变量).现在你应当可以在命令行中运行pip了.试试安装一个包:

```
pip install httpie
```

终于可以运行了!下面是一些问题的解决办法

代理问题

如果你在办公环境,那么你可能有一个HTTP代理.把环境变量设置成 `http_proxy` 和 `https_proxy` .大多数应用程序都管用(包括免费软件).语法如下:

```
http://proxy_url:port  
http://username:password@proxy_url:port
```

如果你不幸用的是微软的NTLM代理,那就没救了.唯一的办法就是安装一个友好的代理吧.

<http://cntlm.sourceforge.net/>

找不到vcvarsall.bat

Python有的模块是用C或C++写的.Pip尝试从源码编译.如果你没有安装或设置过C/C++编译器,你将会看到下面的错误:

```
Error: Unable to find vcvarsall.bat
```

你可以通过像MinGw或者Visual C++这样的C++编译器来解决此问题.微软实际上已经自带了一个为Python准备的编译器,或者试试 <http://aka.ms/vcpython27>

可以这个Christoph的网站来查看安装包 <http://www.lfd.uci.edu/~gohlke/pythonlibs/>

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
22	2236	3143856	11	url

使用 for 来迭代字典的原理

有点不理解下面的代码:

```
d = {'x': 1, 'y': 2, 'z': 3}
for key in d:
    print key, 'corresponds to', d[key]
```

我有点不理解 `key` 的部分. Python 是怎么识别出只需要从字典中读取 `key`? `key` 是 Python 里的关键词还是只是一个简单的变量?

`key` 就是一个变量名.

```
for key in d:
```

上面的语句紧紧会循环字典中的 `key`, 而不是 `key` 和 `value`. 如果需要循环 `key` 和 `value`, 你可以用下面的方法:

Python 2.x:

```
for key, value in d.iteritems():
```

Python 3.x:

```
for key, value in d.items():
```

你可以自己试一下用其他变量替换 `key`.

在 Python3 里 `iteritems()` 简化成了 `items()`, 它能像 `iteritems()` 一样返回一个基于字典本身的像 `set` 一样的 `view`, 但是做了更多的优化. 在 Python 2.7 里和 `viewitems()` 的效果是一样的.

在 Python2 和 Python3 里都有 `items()`, 但是在 2 中只会返回一个 `list`, 它的结构是这种 `(key, value)` 键值对的形式, 它不能反映出在字典调用 `items()` 后做出的改变. 所以在 Python3.x 中需要使用 `list(d.items())` 才能与之等价.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
23	2203	1237032	36	url

在嵌套列表中获取 flat list

我好奇如何可以 flat 一个嵌套列表.

我可以用 for 循环来做, 但是有没有更好的方法? 我试着用 reduce , 但是失败了

Code

```
l = [[1, 2, 3], [4, 5, 6], [7], [8, 9]]
reduce(lambda x, y: x.extend(y), l)
```

Error message

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 1, in <lambda>
AttributeError: 'NoneType' object has no attribute 'extend'
```

```
flat_list = [item for sublist in l for item in sublist]
```

等价于

```
for sublist in l:
    for item in sublist:
        flat_list.append(item)
```

这个是完整的写法:

```
flatten = lambda l: [item for sublist in l for item in sublist]
```

可以用 `timeit` module 来验证效率:

```
$ python -mtimeit -s'l=[[1,2,3],[4,5,6], [7], [8,9]]*99' '[item for sublist in l for i
tem in sublist]'
10000 loops, best of 3: 143 usec per loop
$ python -mtimeit -s'l=[[1,2,3],[4,5,6], [7], [8,9]]*99' 'sum(l, [])'
1000 loops, best of 3: 969 usec per loop
$ python -mtimeit -s'l=[[1,2,3],[4,5,6], [7], [8,9]]*99' 'reduce(lambda x,y: x+y,l)'
1000 loops, best of 3: 1.1 msec per loop
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
24	2186	487459	21	url

__str__和repr__的区别

__str__ 和 repr__ 有什么区别？

首先让我们梳理一下张三的答案：

- 用起来没有什么区别
- __repr__ 的目的是明确的
- __str__ 的目的是可读性
- __str__ 的用法包含 __repr__

用法没有什么区别

因为Python的宗旨是易用的,所以这看起来有点不寻常.但是当

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
25	2132	2306176	12	url

在 Python 里如何手工进行延迟？

```
import time
time.sleep(5) # Delays for 5 seconds. You can also use a float value.
```

这里有另一个例子展示每隔差不多一分钟运行一次：

```
import time
while True:
    print("This prints once a minute.")
    time.sleep(60) # Delay for 1 minute (60 seconds).
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
26	2114	118677	30	url

Python中的"小震撼":变化的默认参数

许多 Python 老手也被下面的问题困扰:

```
def foo(a=[]):  
    a.append(5)  
    return a
```

Python 新手估计可能会想这个函数返回一个只有元素 `[5]` 的列表,但是结果却出人意料:

```
>>> foo()  
[5]  
>>> foo()  
[5, 5]  
>>> foo()  
[5, 5, 5]  
>>> foo()  
[5, 5, 5, 5]  
>>> foo()
```

我的一个经理曾经碰到过这个特性并把它叫做语言的"动态设计缺陷". 这个现象应当有更深层次的解释, 如果你不懂它的内部它确实非常令人困惑. 然而我不能回答下面的问题:是什么原因使默认参数在函数的定义时被绑定,而不是在执行时?我怀疑这个特性在现实中有没有实际的用途(就像在C语言中有谁去用静态变量?)

事实上这并不是设计缺陷,也不是什么内部或性能原因.

原因很简单,Python中的函数是最高等级的对象,而不仅仅是一小段代码.

试着这么来理解:一个函数是一个被它自己定义而执行的对象;默认参数是一种"成员数据",所以它们的状态和其他对象一样,会随着每一次调用而改变.

Effbot在它的[Python中的默认参数](#)对这种行为的原因解释的非常清楚!我强烈建议你读一读能对函数对象的工作原理有更深一步的了解.

Python 中的默认参数(文章翻译)

Python 默认参数很容易让新手犯错.

引起错误的原因通常是使用可变(`mutable`)对象作为参数的默认值, 这个可变对象是可以被修改的, 比如说列表(`list`)和字典(`dict`)

例如:

```
>>> def function(data=[]):  
...     data.append(1)  
...     return data  
...  
>>> function()  
[1]  
>>> function()  
[1, 1]  
>>> function()  
[1, 1, 1]
```

可以看到上面的列表(`list`)越来越长, 如果检查这个列表的`id`, 会发现他返回的是同一个值:

```
>>> id(function())  
12516768  
>>> id(function())  
12516768  
>>> id(function())  
12516768
```

原因很简单: 在每次函数调用的时候都使用同一个对象(`object`), 这个可修改的特性我们叫做 "sticky".

原理

只有当`"def"`语句执行的时候才会对默认参数值进行赋值, 可以看看文档的定义:

<https://docs.python.org/2.0/ref/function.html>

还要注意的是在Python 中 `"def"` 是可执行语句, 默认参数仅在`"def"` 语句环境里生效. 每次执行`"def"`语句都会创建一个新的函数对象(每次都会对默认参数进行赋值)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间:
2018-10-17 17:33:54

rank	vote	view	answer	url
27	2113	1002442	23	url

怎么用引用来改变一个变量？

Python的文档对参数传递的是值还是引用没有明确说明,下面的代码没有改变值 `Original`

```
class PassByReference:
    def __init__(self):
        self.variable = 'Original'
        self.Change(self.variable)
        print self.variable

    def Change(self, var):
        var = 'Changed'
```

有什么方法能让通过引用来改变变量吗？

参数是通过assignment来传递的.原因是双重的:

1. 传递的参数实际上是一个对象的引用(但是这个引用是通过值传递的)
2. 一些数据类型是可变的,但有一些就不是.

所以:

- 如果传递一个可变对象到一个方法,方法就会获得那个对象的引用,而你也可以随心所欲的改变它了.但是你在方法里重新绑定了这个引用,外部是无法得知的,而当函数完成后,外界的引用依然指向原来的对象.
- 如果你传递一个不可变的对象到一个方法,你仍然不能在外边重新绑定引用,你连改变对象都不可以.

为了弄懂,来几个例子.

列表-可变类型

让我们试着修改当做参数传递给方法的列表:

```
def try_to_change_list_contents(the_list):
    print 'got', the_list
    the_list.append('four')
    print 'changed to', the_list

outer_list = ['one', 'two', 'three']

print 'before, outer_list =', outer_list
try_to_change_list_contents(outer_list)
print 'after, outer_list =', outer_list
```

输出:

```
before, outer_list = ['one', 'two', 'three']
got ['one', 'two', 'three']
changed to ['one', 'two', 'three', 'four']
after, outer_list = ['one', 'two', 'three', 'four']
```

因为传递的参数是 `outer_list` 的引用,而不是副本,所以我们可以用可变列表的方法来改变它,而且改变同时反馈到了外部.

现在让我们来看看我们试着改变作为传递参数的引用时到底发生了什么:

```
def try_to_change_list_reference(the_list):
    print 'got', the_list
    the_list = ['and', 'we', 'can', 'not', 'lie']
    print 'set to', the_list

outer_list = ['we', 'like', 'proper', 'English']

print 'before, outer_list =', outer_list
try_to_change_list_reference(outer_list)
print 'after, outer_list =', outer_list
```

输出:

```
before, outer_list = ['we', 'like', 'proper', 'English']
got ['we', 'like', 'proper', 'English']
set to ['and', 'we', 'can', 'not', 'lie']
after, outer_list = ['we', 'like', 'proper', 'English']
```

既然 `the_list` 参数是通过值进行传递的,那么为它赋值将会对方法以外没有影响. `the_list` 是 `outer_list` 引用(注意,名词)的一个拷贝,我们将 `the_list` 指向一个新的列表,但是并没有改变 `outer_list` 的指向.

字符串-不可变类型

它是不可变类型,所以我们不能改变字符串里的内容。

现在,让我们试着改变引用

```
def try_to_change_string_reference(the_string):
    print 'got', the_string
    the_string = 'In a kingdom by the sea'
    print 'set to', the_string

outer_string = 'It was many and many a year ago'

print 'before, outer_string =', outer_string
try_to_change_string_reference(outer_string)
print 'after, outer_string =', outer_string
```

输出:

```
before, outer_string = It was many and many a year ago
got It was many and many a year ago
set to In a kingdom by the sea
after, outer_string = It was many and many a year ago
```

又一次,既然 `the_string` 参数用值进行传递,对它进行赋值并不能改变方法外的值. `the_string` 只是 `outer_string` 引用(名词)的副本,所以我们让 `the_string` 指向一个新字符串,依然无法改变 `outer_string` 的指向。

希望你清楚以上那些。

修改:到现在位置还没有回答"有什么方法同过引用传递变量?",让我们往下看。

我们该怎么办？

你可以返回一个新值.这不会改变传过来的值,但是能得到你想要的结果。

```
def return_a_whole_new_string(the_string):
    new_string = something_to_do_with_the_old_string(the_string)
    return new_string

# 你可以像这样调用
my_string = return_a_whole_new_string(my_string)
```

如果你真的不想用一个返回值,你可以建一个存放你的值的类,然后把它传递给函数或者用一个已有的类,像列表:

```
def use_a_wrapper_to_simulate_pass_by_reference(stuff_to_change):
    new_string = something_to_do_with_the_old_string(stuff_to_change[0])
    stuff_to_change[0] = new_string

# 你可以像这样调用
wrapper = [my_string]
use_a_wrapper_to_simulate_pass_by_reference(wrapper)

do_something_with(wrapper[0])
```

虽然看起来有一点笨重,但还是达到你的效果了.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
28	2049	2277012	31	url

在Python里获取当前时间

用什么方法或者模块获取当前时间？

```
>>> import datetime
>>> datetime.datetime.now()
datetime(2009, 1, 6, 15, 8, 24, 78915)
```

如果只要时间：

```
>>> datetime.datetime.time(datetime.datetime.now())
datetime.time(15, 8, 24, 78915)
```

同样功能但是更紧凑的写法：

```
>>> datetime.datetime.now().time()
```

获取更多信息,看[文档](#).

如果还要节省输出,可以在从 `datetime` 模块import `datetime` 对象：

```
>>> from datetime import datetime
```

然后把所有 `datetime.` 从头部去掉.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
29	2044	1357234	7	url

理解Python中super()和__init__()方法

我试着理解 `super()` 方法.从表面上看,两个子类实现的功能都一样.我想问它们俩的区别在哪里?

```
class Base(object):
    def __init__(self):
        print "Base created"

class ChildA(Base):
    def __init__(self):
        Base.__init__(self)

class ChildB(Base):
    def __init__(self):
        super(ChildB, self).__init__()

print ChildA(),ChildB()
```

`super()` 的好处就是可以避免直接使用父类的名字.但是它主要用于多重继承,[这里面](#)有很多好玩的东西.如果还不了解的话可以看看[官方文档](#)

注意在Python3.0里语法有所改变:你可以用 `super().__init__()` 替换 `super(ChildB, self).__init__()` .(在我看来非常nice)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 : 2018-10-17 17:33:54

rank	vote	view	answer	url
30	2042	515111	7	url

在一行里获取多个异常

我知道这样:

```
try:
    # 可能错的地方
except:
    # 如果错了执行这里
```

也知道这样:

```
try:
    # 可能错的地方
except IDontLikeYourFaceException:
    # 给爷笑一个
except YouAreTooShortException:
    # 踩高跷
```

但是我想在两个不同的异常里做同样的事,我能想到的办法:

```
try:
    # 可能错的地方
except IDontLikeYouException:
    # 滚
except YouAreBeingMeanException:
    # 滚
```

有什么方法能像下面那样:

```
try:
    # 可能错的地方
except IDontLikeYouException, YouAreBeingMeanException:
    # 滚
```

现在下面的代码根本不好使:


```
try:
    # 可能错的地方
except Exception, e:
    # 滚
```

所以有什么方法可以完成我的要求吗？

用括号扩起来：

```
except (IDontLikeYouException, YouAreBeingMeanException) as e:
    pass
```

用逗号分割的方法只能在Python2.6和2.7里好使,在Python3中则无效;现在,你应当使用 `as` .

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
31	1998	2444832	20	url

字典里添加元素的方法

当一个字典被创建了,能不能在字典里计入一个键?好像没有 `.add()` 的方法.

```
>>> d = {'key': 'value'}
>>> print d
{'key': 'value'}
>>> d['mynewkey'] = 'mynewvalue'
>>> print d
{'mynewkey': 'mynewvalue', 'key': 'value'}
```

```
>>> x = {1:2}
>>> print x
{1: 2}

>>> x.update({3:4})
>>> print x
{1: 2, 3: 4}
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
32	1967	511549	25	url

有方法让Python运行在Android上吗？

我喜欢Android平台.试试上我和几个朋友在Spox项目里在用ADC(数模转换器?)

但是我一点也不喜欢Java.我们工作在S60版本而且一个不错的Python的API.我知道Android上没有官方的Python版本,但是既然有Jython,有没有什么方法让它们能在Android上工作？

有一种方法,使用[Kivy](#):

交互界面快速开发应用的Python开源库,像多点触控app.

Kivy运行在 Linux, Windows, OS X, Android and iOS.你也可以在所有的平台上运行Python代码

[Kivy的应用](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
33	1828	3109154	24	url

把字符串解析成浮点数或者整数

在Python里,怎么样把一个数字型字符串像 "545.2222" 解析成对应的浮点值 545.2222 ?或者把一个 "31" 解析成 31 ?

我只是想知道怎么样才能解析一个浮点型字符串编程浮点数,或者整型字符串转换成整型?

```
>>> a = "545.2222"
>>> float(a)
545.22220000000004
>>> int(float(a))
545
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
34	1787	1099948	18	url

如何克隆或者 copy 一个 list ?

使用 `new_list = my_list` 来 copy list , 但是 每次 `my_list` 改变 `new_list` 都会跟着改变.
这是为什么?

用 `new_list = my_list` 来进行 copy 实际上并没有得到两个 list. 这个语句并不是真正的复制了一个 list, 而是复制了一个 list 的引用, 所以 `my_list` 和 `new_list` 最终指向了同一个 list.

为了真正 copy 一个 list, 你可以用下面几种方法:

- 你可以使用内建函数 `list.copy()` 方法(从 Python3.3 以上可以使用):

```
new_list = old_list.copy()
```

- 也可以使用切片:

```
new_list = old_list[:]
```

[Alex Martelli's](#) 早在2007指出这个切片的语法很奇怪, 他主张用下一个方法可读性更强.

- 使用内建的 `list()` 函数:

```
new_list = list(old_list)
```

- 还可以使用 `copy.copy()` :

```
import copy
new_list = copy.copy(old_list)
```

它比 `list()` 方法更慢, 因为它需要先找出 `old_list` 的数据类型.

- 如果 list 中包含对象并且你也希望 copy 这些对象, 那么可以使用 `copy.deepcopy()` :

```
import copy
new_list = copy.deepcopy(old_list)
```

显然这种方法更加慢, 并且需要更多的内存, 但是有些时候不可避免需要用到它.

例如:

```
import copy

class Foo(object):
    def __init__(self, val):
        self.val = val

    def __repr__(self):
        return str(self.val)

foo = Foo(1)

a = ['foo', foo]
b = a.copy()
c = a[:]
d = list(a)
e = copy.copy(a)
f = copy.deepcopy(a)

# edit original list and instance
a.append('baz')
foo.val = 5

print('original: %r\n list.copy(): %r\n slice: %r\n list(): %r\n copy: %r\n deepcopy: %r'
      % (a, b, c, d, e, f))
```

结果:

```
original: ['foo', 5, 'baz']
list.copy(): ['foo', 5]
slice: ['foo', 5]
list(): ['foo', 5]
copy: ['foo', 5]
deepcopy: ['foo', 1]
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
35	1785	2790533	34	url

把文件一行行读入数组

怎样才能一行行的读一个文件并把每行作为一个元素存入一个数组？

我想读取文件的没一行，然后把每行加入到数组的最后，我没找到这样的方法而且我也没能在Python中找到创建字符数组的方法。

```
with open(fname) as f:
    content = f.readlines()
```

我想你说的是 [list] (<http://docs.python.org/glossary.html#term-list>)而不是数组。

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
36	1727	1504654	28	url

合并两个列表

怎样合并两个列表？

例如：

```
listone = [1, 2, 3]
listtwo = [4, 5, 6]
```

我期待：

```
mergedlist == [1, 2, 3, 4, 5, 6]
```

在Python中非常容易。

```
mergedlist = listone + listtwo
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
37	1676	1568802	21	url

在 macOS 上如何安装 pip

今天我花了许多时间研究怎么安装 `pip`，但是没有找到方法

我该怎么安装它？

```
easy_install pip
```

如果你需要管理员权限, 试一下:

```
sudo easy_install pip
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
38	1659	2178345	11	url

获得一个字符串的子串

有什么方法获得一个字符串的字串,比如从一个字符串的第三个字符到最后.

可能是 `myString[2:end]` ?

```
>>> x = "Hello World!"
>>> x[2:]
'llo World!'
>>> x[:2]
'He'
>>> x[:-2]
'Hello Worl'
>>> x[-2:]
'd!'
>>> x[2:-2]
'llo Worl'
```

上面的概念在Python中叫"slicing"(切片),不止在字符串上有这个方法.在[这里](#)有详细的解释.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
39	1658	461700	18	url

Python中**和*参数有什么用？

见55

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
40	1653	643357	57	url

在Python怎么样才能把列表分割成同样大小的块？

我有一个任意长度的列表,我需要把它们切成相同大小并且使用它们.有一些简单的方法,比如设置一个计数器和两个列表,当第二个列表装满后,把它存进第一个列表然后让第二个列表变空一遍存放下一轮的数据,但是这么做代价太大了.

我想是不是还有其他更好的方法,比如生成器

像下面这样:

```
l = range(1, 1000)
print chunks(l, 10) -> [ [ 1..10 ], [ 11..20 ], .., [ 991..999 ] ]
```

我试着在 `itertools` 找一些有用的方法,但是还没有找到.也可能是我还没有看到.

相似问题:[What is the most “pythonic” way to iterate over a list in chunks?](#)

下面是生成器方法:

```
def chunks(l, n):
    """ Yield successive n-sized chunks from l.
    """
    for i in xrange(0, len(l), n):
        yield l[i:i+n]
```

```
import pprint
pprint.pprint(list(chunks(range(10, 75), 10)))
[[10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
 [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
 [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
 [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
 [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
 [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
 [70, 71, 72, 73, 74]]
```

一句话

```
tuple(l[i:i+n] for i in xrange(0, len(l), n))
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
41	1623	1189801	5	url

手动抛出异常

我想故意制造一个错误,所以我可以转到 `excepty: 语句`

我怎么做?

不能在Pythonic了;

```
raise Exception("I know python!")
```

想得到更多信息,看[这里](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

怎么样获取一个列表的长度？

rank	vote	view	answer	url
42	1604	2652232	6	url

怎么样获取一个列表的长度？

```
items = []
items.append("apple")
items.append("orange")
items.append("banana")

# FAKE METHOD::
items.amount() # 返回 3
```

怎么样做才对？

`len`函数可以用于Python中许多的类型,包括内建类型和标准库类型.

```
>>len([1,2,3])
3
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
43	1580	827210	37	url

怎么在终端里输出颜色?

怎么样用Python在终端里输出带颜色的文本?最好的代替块字符的unicode字符是什么?(What is the best Unicode symbol to represent a solid block?这句话没理解)

这依赖于你用哪种操作系统.最常用的方法就是输出ANSI转义序列.例如,下面的:

```
class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
```

可以这么用上面的代码:

```
print bcolors.WARNING + "Warning: No active frommets remain. Continue?" + bcolors.ENDC
```

这种方法适合OS X,linux和windows系统.还有一些其他的ansi代码可以设置颜色,消除光标或者其他.

如果想要应用更复杂的功能(听起来像是如果你正在写一个游戏),你应该看看" curses"模块,它包含了许多关于次部分复杂操作.[Python Curses HowTO](#)可以作为好的介绍.

如果你不使用拓展的ASCII(比如不是一个PC), # 或者 @ 可能是你最好的选择.如果你用的是[IBM扩展ascii字符设置](#),你还可以有更多的选择.176,177,178和219是"块字符"

一些现代的基于文本的程序,像"Swarf Fortress",显示的文本使用图像模式,而用的字体也是传统的计算机字体的图像.你可以在[Dwarf Fortress Wiki](#)找到你可以用的字符.

[Text Mode Demo Contest](#) 也有许多资料可供参考.

我想可能有点跑题了.我在设计一个史诗级别的基于文本的冒险游戏.祝你好运!

怎么就没人提[Python termcolor module](#).用法相当简单:


```
from termcolor import colored

print colored('hello', 'red'), colored('world', 'green')
```

虽然有点简单,但是对于游戏程序和你想要的"colored blocks"来说足够了.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间:
2018-10-17 17:33:54

rank	vote	view	answer	url
44	1574	1911103	15	url

查找当前目录和文件目录

在 Python 中什么命令可以找到:

1. 当前目录(在终端中运行的 Python 脚本)
2. 我执行的文件在哪里?

可以通过下面的命令获得执行的 Python 文件所在的位置:

```
import os
dir_path = os.path.dirname(os.path.realpath(__file__))
```

(注意当你已经使用 `os.chdir()` 来该改变当前工作目录的话, 上面得到的结果并不准确, 因为 `__file__` 变量只表示当前工作目录的相对位置, 不会被 `os.chdir()` 所改变)

获取当前工作目录:

```
import os
cwd = os.getcwd()
```

下面是几个函数,模块或者变量的文档:

- `os` 和 `os.path` 模块.
- `__file__` 变量
- `os.path.realpath(path)`
- `os.path.dirname(path)`
- `os.getcwd()`
- `os.chdir(path)`

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
45	1570	1836708	8	url

在 Python 中如何将字符串转回为小写

```
s = "Kilometer"
print(s.lower())
```

官方文档 [str.lower](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
46	1565	1231023	14	url

复制文件

怎么在Python里赋值文件?在 `os` 下没找到复制的方法.

`shutil` 有许多的方法.其中之一就是:

```
copyfile(src, dst)
```

把 `src` 文件的内容复制给 `dst`.目的地址必须是可写的;否则将会出现 `IOError` 错误.如果 `dst` 已经存在,将会被覆盖.一些像字符或者块设备不能用这个方法赋值. `src` 和 `dst` 是路径名的字符串形式.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
47	1561	970000	16	url

Python 中类的静态变量

在 Python 中如何实现类的静态变量或者方法

在类定义中声明且在方法里未声明的是类的静态变量

```
>>> class MyClass:
...     i = 3
...
>>> MyClass.i
3
```

这样做会生成一个类级别的 `i` 变量, 而不是依赖于某个实例的 `i` 变量, 所以你可以:

```
>>> m = MyClass()
>>> m.i = 4
>>> MyClass.i, m.i
>>> (3, 4)
```

和 C++ 和 Java 不一样, 这种静态变量不能通过实例来访问有点像 C#.

可以看一看 [what the Python tutorial has to say on the subject of classes and class objects](#)

[static methods](#) 的相关文档 "Built-in Functions" in the Python Library Reference

```
class C:
    @staticmethod
    def f(arg1, arg2, ...): ...
```

有人提到使用 `classmethod` 作为静态方法, 它接受类的类型作为第一个参数, 但是我依然对这种方法的有点疑问, 如果你也是, 那可能没什么关系

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 : 2018-10-17 17:33:54

rank	vote	view	answer	url
48	1551	1956758	19	url

把字符串转化成时间

短小精悍.我有一个特别大的列表存储了下面的字符串:

```
Jun 1 2005 1:33PM
Aug 28 1999 12:00AM
```

我想把它们转化成合适的时间格式存进数据库,所以我需要把它们变成真正的时间对象.

非常感谢帮助.

检查time模块的strptime函数.它是strftime的转换.

```
from datetime import datetime

date_object = datetime.strptime('Jun 1 2005 1:33PM', '%b %d %Y %I:%M%p')
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
49	1529	1252208	49	url

Python中用什么代替switch语句?

我想写一个函数,实现输入一个值对应输出另一个值.

在其他语言我可以用 `switch` 或者 `case` 语句,但是Python里没有 `switch`.Python里如何解决这个问题?

你可以用一个字典:

```
def f(x):  
    return {  
        'a': 1,  
        'b': 2,  
    }[x]
```

如果你希望设置一个默认值可以用字典的 `get` 方法:

```
def f(x):  
    return {  
        'a': 1,  
        'b': 2,  
    }.get(x, 9)    # 9 is default if x not found
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
50	1523	1489169	10	url

删除文件或者文件夹

`os.remove()` 删除文件 `os.rmdir()` 删除空文件夹 `shutil.rmtree()` 删除文件夹和里面的内容 `pathlib.Path.unlink()` 删除文件或链接 `pathlib.Path.rmdir()` 删除空文件夹

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
51	1521	204859	10	url

为什么在C++中读取stdin中的行会比Python慢呢？

ps:这个是C++的问题了所以没做翻译.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
52	1497	673854	11	url

__init__.py 的作用

它是作为包的一部分. [文档](#)

`__init__.py` 作为标识 Python 包的文件, 这么做是为了防止 Python 在引入 module 时误引入其他类似 `string` 这样的通用名. `__init__.py` 最简单的方式是空文件, 但是它可以为包执行初始化代码或者设置 `__all__` 变量

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
53	1479	837904	49	url

用pip升级所有包

可不可以用pip一次性升级所有的Python包？

注:在官方的issue里也有这个[需求](#).

内部还不支持这个命令,但是可以这样:

```
pip freeze --local | grep -v '^-\e' | cut -d = -f 1 | xargs pip install -U
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
54	1467	1346339	14	url

获取列表最后一个元素

在Python里,如何获取一个列表的最后一个元素?

`some_list[-1]` 最短最Pythonic的方法.

事实上你可以用这个语法做好多事. `some_list[-n]` 语法获取倒数第n个元素.所

以 `some_list[-1]` 获取最后一个元素, `some_list[-2]` 获取倒数第二个,等等.最后 `some_list[-len(some_list)]` 可以获取第一个元素~~

你也可以用这种方法获取列表元素,例如:

```
>>> some_list = [1, 2, 3]
>>> some_list[-1] = 5 # Set the last element
>>> some_list[-2] = 3 # Set the second to last element
>>> some_list
[1, 3, 5]
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
55	1450	1382804	9	url

查看一个对象的类型

有什么方便的方法查看一个对象的类型？

为了获得对象的类型,可以用内建函数 `type()` .把对象作为唯一的参数传递将会返回这个对象的类型:

```
>>> type([]) is list
True
>>> type({}) is dict
True
>>> type('') is str
True
>>> type(0) is int
True
```

当然也对自定义类型也有用:

```
>>> class Test1 (object):
    pass
>>> class Test2 (Test1):
    pass
>>> a = Test1()
>>> b = Test2()
>>> type(a) is Test1
True
>>> type(b) is Test2
True
```

注意 `type()` 只会返回对象的直接类型,不会告诉你继承类型.

```
>>> type(b) is Test1
False
```

可以用 `isinstance` 函数.也对内建函数管用:

```
>>> isinstance(b, Test1)
True
>>> isinstance(b, Test2)
True
>>> isinstance(a, Test1)
True
>>> isinstance(a, Test2)
False
>>> isinstance([], list)
True
>>> isinstance({}, dict)
True
```

`isinstance()` 通常是确定一个对象类型更好的方法,因为它接受派生类型.所以除非你确实需要知道对象的类型(一些其他原因),用 `isinstance()` 比 `type()` 更好.

`isinstance()` 的第二个参数也接受类型的元组,所以也可以一次检查多种类型.如果是这些类型里的 `isinstance()` 将会返回`true`:

```
>>> isinstance([], (tuple, list, set))
True
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
56	1449	119508	7	url

为什么 "10000000000000000 in range(10000000000000001)" 在 Python3 里速度那么快?

在我理解 `range()` 函数作为 [an object type in Python 3](#) 生成的内容和生成器差不多快。

我测试如下语句

```
10000000000000000 in range(10000000000000001)
```

不管我加了多少个 0 这个计算的结果时间都差不多

我还试了下这个, 结果差不多, 非常快:

```
00000000000000000000 in range(0, 10000000000000000001, 10) # count by tens
```

但是我自己写了个代码速度就非常慢:

```
def my_crappy_range(N):
    i = 0
    while i < N:
        yield i
        i += 1
    return
```

所以我想问为啥 `range()` 如此之快?

Python3 的 `range()` 对象并不会立即产生数值;它会在需要时来产生.它包含你的起始, 终止和步长, 它会随着每次迭代来产生下一个数值.

这个对象通过 `object.__contains__` hook 来实现的.计算的时间复杂度为 $O(1)$.在迭代的过程中其实是不需要一下计算出所有的数值的.

根据 [range\(\)](#) 文档

`range` 优于 `list` 和 `tuple` 的地方在于不管它的范围是多少它只占用少量的内存(因为它只保存 `start`, `stop`, `step`, 按需得到数值)

所以你可以这么写你的 `range()` :

```
class my_range(object):
    def __init__(self, start, stop=None, step=1):
        if stop is None:
            start, stop = 0, start
        self.start, self.stop, self.step = start, stop, step
        if step < 0:
            lo, hi = stop, start
        else:
            lo, hi = start, stop
        self.length = ((hi - lo - 1) // abs(step)) + 1

    def __iter__(self):
        current = self.start
        if self.step < 0:
            while current > self.stop:
                yield current
                current += self.step
        else:
            while current < self.stop:
                yield current
                current += self.step

    def __len__(self):
        return self.length

    def __getitem__(self, i):
        if i < 0:
            i += self.length
        if 0 <= i < self.length:
            return self.start + i * self.step
        raise IndexError('Index out of range: {}'.format(i))

    def __contains__(self, num):
        if self.step < 0:
            if not (self.stop < num <= self.start):
                return False
        else:
            if not (self.start <= num < self.stop):
                return False
        return (num - self.start) % self.step == 0
```

这个代码和真正的 `range()` 缺少了像 `.index()`, `.count()` 方法,但是应当可以给你些启发.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
57	1441	1459468	26	url

如何使打印的内容没有空格或换行

在 `c` 语言中我可以这么打印:

```
#include <stdio.h>

int main() {
    int i;
    for (i=0; i<10; i++) printf(".");
    return 0;
}
```

输出:

```
.....
```

在 Python 里:

```
>>> for i in xrange(0,10): print '.'
.
.
.
.
.
.
.
.
.
.
>>> for i in xrange(0,10): print '.',
. . . . . . . . . .
```

在 Python 里 `print` 会加上 `\n` 或者空格, 如何避免?

通用做法

```
import sys
sys.stdout.write('.')
```

你也可以调用

```
sys.stdout.flush()
```

来确保 `stdout` 立即刷新

Python 2.6+

在 Python2.6 里你可以引入 Python3 的 `print` 函数:

```
from __future__ import print_function
```

这样可以用 Python3 来解决上面的问题

Python3

在 Python3 中, `print` 语句变成了函数.在 Python3 中你可以:

```
print('.', end='')
```

在 Python2 中也可以, 确保你引入 `from __future__ import print_function` .

如果你在缓冲遇到了麻烦, 你可以在后面加上 `flush=True` 参数:

```
print('.', end='', flush=True)
```

注意 `flush` 关键字在 Python2 里引入 `__future__` 后并不生效;它只在 Python3 里, 更精确的说是 3.3 或以上.在老版本里你需要手动调用 `sys.stdout.flush()` .

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
58	1428	1063020	9	url

为什么是string.join(list)而不是list.join(string)?

我一直被这个问题困扰.如果这样写更好:

```
my_list = ["Hello", "world"]
print my_list.join("-")
# Produce: "Hello-world"
```

而不是:

```
my_list = ["Hello", "world"]
print "-".join(my_list)
# Produce: "Hello-world"
```

有什么特殊的原因让它不这样做吗?

是因为所有的可迭代对象都能被 `join`, 不仅仅是列表,但结果是我们一般要 `join` 的都是字符串.

例如:

```
import urllib2
print '\n#####\n'.join(urllib2.urlopen('http://data.stackexchange.com/users/7095'))
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
59	1424	790037	9	url

静态方法

Python有没有静态方法使我可以不用实例化一个类就可以调用,像这样:

```
ClassName.StaticMethod ( )
```

是的,用[静态方法](#)装饰器

```
class MyClass(object):
    @staticmethod
    def the_static_method(x):
        print x

MyClass.the_static_method(2) # outputs 2
```

注意有些代码用一个函数而不是 `staticmethod` 装饰器去定义一个静态方法.如果你想支持Python的老版本(2.2和2.3)可以用下面的方法:

```
class MyClass(object):
    def the_static_method(x):
        print x
    the_static_method = staticmethod(the_static_method)

MyClass.the_static_method(2) # outputs 2
```

这个方法和第一个一样,只是没有第一个用装饰器的优雅.

最后,请少用 `staticmethod` 方法!在Python里只有很少的场合适用静态方法,其实许多顶层函数会比静态方法更清晰明了.

[文档](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
61	1413	541572	17	url

通过列表中字典的值对列表进行排序

我的到了一个字典的列表,我想对字典的值进行排序.

```
[{'name': 'Homer', 'age': 39}, {'name': 'Bart', 'age': 10}]
```

对name进行排序,应当是:

```
[{'name': 'Bart', 'age': 10}, {'name': 'Homer', 'age': 39}]
```

用key比用cmp更清晰明了:

```
newlist = sorted(list_to_be_sorted, key=lambda k: k['name'])
```

或者其他人的建议:

```
from operator import itemgetter
newlist = sorted(list_to_be_sorted, key=itemgetter('name'))
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
62	1363	1038536	14	url

在列表中随机取一个元素

例如我有如下列表:

```
foo = ['a', 'b', 'c', 'd', 'e']
```

从列表中随机取一个元素最好的方法是什么?

```
import random

foo = ['a', 'b', 'c', 'd', 'e']
print(random.choice(foo))
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
63	1363	1319979	26	url

如何移除换行符？

这是我用Python编程遇到的最多的问题了,所以我想放到stackoverflow好让我下次Google'chomp python'的时候能得到有用的答案.

试试 `rstrip` 方法:

```
>>> 'test string\n'.rstrip()
'test string'
```

注意Python的 `rstrip` 方法将会默认去掉所有的空白符,而在Perl里只是删除换行符.如果只是删除换行符:

```
>>> 'test string \n'.rstrip('\n')
'test string '
```

同样也有 `lstrip` 和 `strip` 方法:

```
>>> s = " \n abc  def  "
>>> s.strip()
'abc  def'
>>> s.rstrip()
' \n abc  def'
>>> s.lstrip()
'abc  def '
>>>
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
64	1361	1130319	11	url

在 Python 中如何获取环境变量

环境变量可以通过 `os.environ` 来获取

```
import os
print(os.environ['HOME'])
```

或者你可以查看一下所有环境变量的列表:

```
os.environ
```

```
# using get will return `None` if a key is not present rather than raise a `KeyError`
print(os.environ.get('KEY_THAT_MIGHT_EXIST'))

# os.getenv is equivalent, and can also give a default value instead of `None`
print(os.getenv('KEY_THAT_MIGHT_EXIST', default_value))
```

`Python default installation` 在 Windows 中是 `C:\Python` , 如果你想在运行时得到可以运行下面的:

```
import sys
print(sys.prefix)
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook 该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
65	1349	1258301	16	url

bytes 换成 string

我通过这个代码来获取外部程序的输出:

```
>>> from subprocess import *
>>> command_stdout = Popen(['ls', '-l'], stdout=PIPE).communicate()[0]
```

communicate() 方法返回一个字节数组:

```
>>> command_stdout
b'total 0\n-rw-rw-r-- 1 thomas thomas 0 Mar  3 07:03 file1\n-rw-rw-r-- 1 thomas thomas 0 Mar  3 07:03 file2\n'
```

但是我想让它输出成 Python string. 像这样:

```
>>> print(command_stdout)
-rw-rw-r-- 1 thomas thomas 0 Mar  3 07:03 file1
-rw-rw-r-- 1 thomas thomas 0 Mar  3 07:03 file2
```

我试着用 `binascii.b2a_qp()` 方法但是如下:

```
>>> binascii.b2a_qp(command_stdout)
b'total 0\n-rw-rw-r-- 1 thomas thomas 0 Mar  3 07:03 file1\n-rw-rw-r-- 1 thomas thomas 0 Mar  3 07:03 file2\n'
```

谁知道怎么讲字节转换成 string ?

你需要 decode 字节对象来产生 string:

```
>>> b"abcde"
b'abcde'

# utf-8 is used here because it is a very common encoding, but you
# need to use the encoding your data is actually in.
>>> b"abcde".decode("utf-8")
'abcde'
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
66	1321	508348	11	url

对于初学者如何理解 @classmethod 和 @staticmethod?

虽然 `classmethod` 和 `staticmethod` 非常接近,但是用途却是不同: `classmethod` 必须把类对象的引用作为第一个参数,而 `staticmethod` 则不需要.

例子

```
class Date(object):

    def __init__(self, day=0, month=0, year=0):
        self.day = day
        self.month = month
        self.year = year

    @classmethod
    def from_string(cls, date_as_string):
        day, month, year = map(int, date_as_string.split('-'))
        date1 = cls(day, month, year)
        return date1

    @staticmethod
    def is_date_valid(date_as_string):
        day, month, year = map(int, date_as_string.split('-'))
        return day <= 31 and month <= 12 and year <= 3999

date2 = Date.from_string('11-09-2012')
is_date = Date.is_date_valid('11-09-2012')
```

解释

让我们来创建一个处理日期的 class:

```
class Date(object):

    def __init__(self, day=0, month=0, year=0):
        self.day = day
        self.month = month
        self.year = year
```

这个 class 很显然是用来存储特定日期的信息.

这里用 Python 的 `__init__` 来初始化类实例, 它可以接收参数作为一个 `instancemethod`, 它的第一个参数 `self` 是新建实例的引用.

类方法

我们可以用 `classmethod` 来很好的完成一些功能.

假设我们想通过一些形如 `dd-mm-yyyy` 的 string 来创建 `Date` 实例. 而且我们需要在我们工程里许多不同的地方实现.

所以我们必须:

1. 解析 string 为天, 月和年这3个整数或者3元元祖.
2. 通过传递这些值来完成 `Date` 的初始化调用

```
day, month, year = map(int, string_date.split('-'))
date1 = Date(day, month, year)
```

在 C++ 里可以用 `overloading` 的方法来实现, 但是 Python 里没有 `overloading`. 取而代之的我们可以使用 `classmethod`. 让我们来创建另一种 `constructor`

```
@classmethod
def from_string(cls, date_as_string):
    day, month, year = map(int, date_as_string.split('-'))
    date1 = cls(day, month, year)
    return date1

date2 = Date.from_string('11-09-2012')
```

让我们仔细看看上面两个实现, 来看看两个实现的优缺点:

1. 我们在一个地方实现了 `date string` 的转换并且可以复用.
2. 很好的封装(如果你认为或许可以实现一个函数来解析, 但是这么做更符合 OOP 设计)
3. `cls` 是一个承载 `class` 自己的对象, 并不是 `class` 的实例. 这个非常好用, 因为只要我们继承了 `Date class`, 所有的子类都拥有 `from_string` 这个方法.

静态方法

`staticmethods` 呢? 它非常像 `classmethod` 但是它不包含任何必须的参数(比如说类方法或者实例方法).

让我们看看下一个用法.

我们想验证一个 `date string` 的有效性, 这个任务不需要绑定到实例上只需要绑定到类上面.

`staticmethod` 对我们来说就比较好用了.我们看看下一段代码:

```
@staticmethod
def is_date_valid(date_as_string):
    day, month, year = map(int, date_as_string.split('-'))
    return day <= 31 and month <= 12 and year <= 3999

# usage:
is_date = Date.is_date_valid('11-09-2012')
```

所以我们看到 `staticmethod` 就想一个函数一样不需要指定 `class`,语法上的调用也非常像函数,没办法访问对象及其内部,而 `classmethod` 就可以.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
67	1282	1027318	31	url

检查一个字符串是否是一个数字

如果一个字符串可以被看做一个数字那么有什么好的方法可以检测出来？

我能想到的方法：

```
def is_number(s):  
    try:  
        float(s)  
        return True  
    except ValueError:  
        return False
```

我还没有想到什么更好的方法.

对字符串对象用 `isdigit()` 方法：

```
>>> a = "03523"  
>>> a.isdigit()  
True
```

```
>>> b = "963spam"  
>>> b.isdigit()  
False
```

String Methods - `isdigit()`

同样也有`unicode`的方法，但是我不太熟悉[Unicode - Is decimal/decimal](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
68	1247	502333	10	url

通过函数名的字符串来调用这个函数

假如我们有个模块叫 `foo`,而我有一个 `"bar"` 字符串.调用 `foo.bar()` 有什么最好的方法?

我需要返回函数值,为什么我不能用 `eval`.我想应该能用 `eval` 来定义一个函数来返回调用的结果,但是我希望更优雅的方法.

假设 `foo` 有一个 `bar` 方法:

```
import foo
methodToCall = getattr(foo, 'bar')
result = methodToCall()
```

第二行和第三行可以简写:

```
result = getattr(foo, 'bar')()
```

如果让上面的代码更有意义. `getattr` 可以用在实例绑定,模块级方法,类方法...等等

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

如何增加文件的内容？

rank	vote	view	answer	url
69	1236	1278128	7	url

如何增加文件的内容？

如何做到 `append` 文件而不是覆盖？

```
with open("test.txt", "a") as myfile:
    myfile.write("appended text")
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
70	1223	771656	11	url

*args和**kwargs

我对 `*args` 和 `**kwargs` 的概念有点不理解.

到目前位置我了解到:

- `*args` = 参数列表-作为可选参数
- `**kwargs` = 字典-参数的关键字作为键,它们的值作为字典的值.

??

说实话我不裂解这对编程来说有什么用?(我知道它,但是对他不了解.)

或许:

我想把字典或者列表当做参数的同时也把它们当成通配符,所以可以传递任何参数?

有什么实际例子来解释怎么用 `*args` 和 `**kwargs` ?

通过教程我只是用了`""`和一个变量名.

还是 `*args` 和 `**kwargs` 只是一个占位符,你们在代码里用 `*args` 和 `**kwargs` 吗?

`*` 和 `**` 的语法.用 `*args` 和 `**kwargs` 只是为了方便并没有强制使用它们.

当你不确定你的函数里将要传递多少参数时你可以用 `*args` .例如,它可以传递任意数量的参数:

```
>>> def print_everything(*args):
...     for count, thing in enumerate(args):
...         print '{0}. {1}'.format(count, thing)
...
>>> print_everything('apple', 'banana', 'cabbage')
0. apple
1. banana
2. cabbage
```

相似的, `**kwargs` 允许你使用没有事先定义的参数名:

```
>>> def table_things(**kwargs):
...     for name, value in kwargs.items():
...         print '{0} = {1}'.format(name, value)
...
>>> table_things(apple = 'fruit', cabbage = 'vegetable')
cabbage = vegetable
apple = fruit
```

你也可以混着用.命名参数首先获得参数值然后所有的其他参数都传递给 `*args` 和 `**kwargs` .命名参数在列表的最前端.例如:

```
def table_things(titlestring, **kwargs)
```

`*args` 和 `**kwargs` 可以同时出现在函数的定义中,但是 `*args` 必须在 `**kwargs` 前面.

当调用函数时你也可以用 `*` 和 `**` 语法.例如:

```
>>> def print_three_things(a, b, c):
...     print 'a = {0}, b = {1}, c = {2}'.format(a, b, c)
...
>>> mylist = ['aardvark', 'baboon', 'cat']
>>> print_three_things(*mylist)
a = aardvark, b = baboon, c = cat
```

就像你看到的一样,它可以传递列表(或者元组)的每一项并把它解包.注意必须与它们在函数里的参数相吻合.当然,你也可以在函数定义或者函数调用时用 `*` .

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
71	1211	1210518	19	url

在 Python 里如何读取 stdin?

可以使用 `fileinput` :

```
import fileinput

for line in fileinput.input():
    pass
```

`fileinput` 命令可以迭代命令参数给出的文件的每一行, 如果没有给出参数则直接从标准输入中得到.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
72	1195	898656	15	url

字符串格式化:%和.format

Python2.6推出了[str.format()]方法,和原有的%格式化方式有小小的区别.那个方法更好?

1. 下面的方法有同样的输出,它们的区别是什么?

```
#!/usr/bin/python
sub1 = "python string!"
sub2 = "an arg"

a = "i am a %s" % sub1
b = "i am a {}".format(sub1)

c = "with %(kwarg)s!" % {'kwarg':sub2}
d = "with {kwarg}!".format(kwarg=sub2)

print a    # "i am a python string!"
print b    # "i am a python string!"
print c    # "with an arg!"
print d    # "with an arg!"
```

2. 另外在Python中格式化字符串什么时候执行?例如如果我的login的优先级设置为高,那么我还能用 % 操作符吗?如果是这样的话,有什么方法可以避免吗?

```
log.debug("some debug info: %s" % some_info)
```

先回答第一个问题... .format 在许多方面看起来更便利.你可以重用参数,但是你用 % 就不行.最烦人的是 % 它无法同时传递一个变量和元组.你可能会想下面的代码不会有什么问题:

```
"hi there %s" % name
```

但是,如果 name 恰好是 (1,2,3),它将会抛出一个 TypeError 异常.为了保证它总是正确的,你必须这样做:

```
"hi there %s" % (name,)    # 提供一个单元素的数组而不是一个参数
```

但是有点丑. .format 就没有这些问题.你给的第二个问题也是这样, .format 好看多了.

你为什么不用它？

- 不知道它(在读这个之前)
- 为了和Python2.5兼容

回答你的第二个问题,字符串格式和其他操作一样发生在它们运行的时候.Python是非懒惰语言,在函数调用前执行表达式,所以在你的 `log.debug` 例子中, `"some debug info:`

`%s"%some_info` 将会先执行,先生成 `"some debug info: roflcopters are active"` ,然后字符串将会传递给 `log.debug()`

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间：
2018-10-17 17:33:54

rank	vote	view	answer	url
73	1190	2129025	19	url

限制浮点型到小数点后两位

我希望 `a` 可以转化成 13.95

```
>>> a
13.949999999999999
>>> round(a, 2)
13.949999999999999
```

`round` 并没有像我期待的那样得到 13.95

你遇到了典型的浮点数问题,那就是浮点数并不能代表所有的数值.命令行展示的只是内存中的完整的浮点数的形式.

上面的浮点数和你想要的结果是等价的.因为计算机是二进制的,它将浮点数存为整形,并且除以2的幂,所以 13.95 将会表示成 $125650429603636838/(2^{53})$.

双精度数值有53个字节来表示精度,而浮点型有24个字节表示.Python 使用 [双精度浮点数](#) 来存储数值.

例如,

```
>>> 125650429603636838/(2**53)
13.949999999999999

>>> 234042163/(2**24)
13.949999988079071

>>> a=13.946
>>> print(a)
13.946
>>> print("%.2f" % a)
13.95
>>> round(a, 2)
13.949999999999999
>>> print("%.2f" % round(a, 2))
13.95
>>> print("{0:.2f}".format(a))
13.952
>>> print("{0:.2f}".format(round(a, 2)))
13.95
>>> print("{0:.15f}".format(round(a, 2)))
13.949999999999999
```

如果你想要精确到小数点后2位有两个不错的选择:

1. 使用整数并以美分存储值，而不是美元，然后除以100转换为美元。(没看懂,可能是Python 的货币计算?)
2. 使用 Python 的 [decimal](#)

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
74	1184	2147132	8	url

如何解析一个 JSON 文件?

我又一个如下的 JSON 文件:

```
{
  "maps": [
    {
      "id": "blabla",
      "iscategorical": "0"
    },
    {
      "id": "blabla",
      "iscategorical": "0"
    }
  ],
  "masks": [
    "id": "valore"
  ],
  "om_points": "value",
  "parameters": [
    "id": "valore"
  ]
}
```

我写了如下代码来打印所有的 json 文本:

```
json_data=open(file_directory).read()

data = json.loads(json_data)
pprint(data)
```

如何才能解析文件并且得到单个的值?

额,你的 JSON 文件有错误.在应当用 `{}` 的地方使用了 `[]`. `[]` 是列表, `{}` 是字典.

下面是正确的形式:

```
{
  "maps": [
    {
      "id": "blabla",
      "iscategorical": "0"
    },
    {
      "id": "blabla",
      "iscategorical": "0"
    }
  ],
  "masks": {
    "id": "valore"
  },
  "om_points": "value",
  "parameters": {
    "id": "valore"
  }
}
```

你可以用如下代码:

```
import json
from pprint import pprint

with open('data.json') as f:
    data = json.load(f)

pprint(data)
```

你可以使用 `data` 来得到你想要的值:

```
data["maps"][0]["id"]
data["masks"]["id"]
data["om_points"]
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
75	1184	1332206	13	url

如何改变 matplotlib 图像大小

`figure` 告诉你如何操作:

```
from matplotlib.pyplot import figure
figure(num=None, figsize=(8, 6), dpi=80, facecolor='w', edgecolor='k')
```

`figure(figsize=(1,1))` 可以创建一个 1x1 英寸的图像, 由 80x80 像素组成, 如果想改变像素, 可以改变 dpi 大小.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook 该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
76	1183	605483	9	url

在 Python 中如何得到对象的所有属性？

有没有什么方法可以检测对象是否有一个属性？例如：

```
>>> a = SomeClass()
>>> a.someProperty = value
>>> a.property
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: SomeClass instance has no attribute 'property'
```

怎么样才能在用 `property` 之前知道 `a` 有这个属性？

试试 `hasattr()`：

```
if hasattr(a, 'property'):
    a.property
```

答案2

`hasattr` 可以做到,但是我要说一句 Python 鼓励 "easier to ask for forgiveness than permission" (EAFP) 而不是 "look before you leap" (LBYL). 详细查看下面链接:

[EAFP vs LBYL \(was Re: A little disappointed so far\) EAFP vs. LBYL @Code Like a Pythonista: Idiomatic Python](#)

例如：

```
try:
    doStuff(a.property)
except AttributeError:
    otherStuff()
```

要比下面的更好

```
if hasattr(a, 'property'):
    doStuff(a.property)
else:
    otherStuff()
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
78	1146	674169	43	url

在Python里如何用枚举类型？

我是一个C#开发者,但是我现在做的工作是关于Python的.

怎么在Python里代替枚举类型呢？

PEP435标准里已经把枚举添加到Python3.4版本,在Pypi中也可以向后支持3.3, 3.2, 3.1, 2.7, 2.6, 2.5, 和 2.4版本.

如果想向后兼容 \$ pip install enum34 ,如果下载 enum (没有数字)将会是另一个版本.

```
from enum import Enum
Animal = Enum('Animal', 'ant bee cat dog')
```

或者等价于:

```
class Animals(Enum):
    ant = 1
    bee = 2
    cat = 3
    dog = 4
```

在更早的版本,下面这种方法来完成枚举:

```
def enum(**enums):
    return type('Enum', (), enums)
```

像这样来用:

```
>>> Numbers = enum(ONE=1, TWO=2, THREE='three')
>>> Numbers.ONE
1
>>> Numbers.TWO
2
>>> Numbers.THREE
'three'
```

也很容易支持自动计数,像下面这样:

```
def enum(*sequential, **named):
    enums = dict(zip(sequential, range(len(sequential))), **named)
    return type('Enum', (), enums)
```

这样用:

```
>>> Numbers = enum('ZERO', 'ONE', 'TWO')
>>> Numbers.ZERO
0
>>> Numbers.ONE
1
```

如果要把值转换为名字可以加入下面的方法:

```
def enum(*sequential, **named):
    enums = dict(zip(sequential, range(len(sequential))), **named)
    reverse = dict((value, key) for key, value in enums.iteritems())
    enums['reverse_mapping'] = reverse
    return type('Enum', (), enums)
```

这样会覆盖名字下的所有东西,但是对于枚举的输出很有用.如果转换的值不存在就会抛出 `KeyError` 异常.用前面的例子:

```
>>> Numbers.reverse_mapping['three']
'THREE'
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
79	1137	1132186	19	url

如何计算列表中元素出现的个数?

如果你想计算单个元素出现的次数,使用 `count` 方法:

```
>>> [1, 2, 3, 4, 1, 4, 1].count(1)
3
```

不要在计算多个元素时使用这个方法.每次调用 `count` 方法都会迭代一边列表,性能会很低,如果你需要计算所有的元素或者指定几个元素可以使用 `Counter` .

```
>>> from collections import Counter
>>> z = ['blue', 'red', 'blue', 'yellow', 'blue', 'red']
>>> Counter(z)
Counter({'blue': 3, 'red': 2, 'yellow': 1})
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
80	1114	930787	8	url

在 Python 中如何移除字典中的 key ?

当我想移除字典里的 key 时:

```
if 'key' in myDict:
    del myDict['key']
```

有没有一行就能实现的代码?

用 `dict.pop()`

```
my_dict.pop('key', None)
```

上面的代码如果 `key` 存在将会返回 `my_dict[key]` , 不存在就会返回 `None` .如果第二个参数没有指定(例如, `my_dict.pop('key')`),那么如果 `key` 不存在就会抛出 `KeyError` 异常.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
81	1114	475573	8	url

如何离开/退出/停用Python的virtualenv？

我正在用virtualenv和virtualenvwrapper.我能用命令很好的在virtualenv之间切换.

```
me@mymachine:~$ workon env1
(env1)me@mymachine:~$ workon env2
(env2)me@mymachine:~$ workon env1
(env1)me@mymachine:~$
```

但是如何退出虚拟机回到自己的环境中?到目前位置只有一个方法:

```
me@mymachine:~$
```

就是退出shell再新建一个.这有点烦人.有没有现成的命令?如果没有我怎么创建它?

通常在虚拟环境中在shell中输入:

```
$ deactivate
```

将会返回正常环境.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
82	1112	1034686	21	url

反转字符串

在Python里 `str` 没有内建的 `reverse` 函数.实现字符串反转最好的方法是什么?

如果答案很简单的话,那么最有效的是是什么.是不是 `str` 要转换成一个不同的对象.

这个怎么样:

```
>>> 'hello world'[::-1]
'dlrow olleh'
```

这是拓展的切片语法.用 `[begin:end:step]` 实现-从begin开始到end结束步长是-1的话就可以反转一个字符串.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
83	1085	1925603	18	url

如何删除 list 中指定 index 的元素？

使用 `del` 并且指定你想删除的元素的 index:

```
>>> a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> del a[-1]
>>> a
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

同时支持切片:

```
>>> del a[2:4]
>>> a
[0, 1, 4, 5, 6, 7, 8, 9]
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
84	1085	1673710	15	url

如何查看 Python 变量的类型？

Python 不像 C/C++, 不会出现你的问题

试一下:

```
>>> i = 123
>>> type(i)
<type 'int'>
>>> type(i) is int
True
>>> i = 123456789L
>>> type(i)
<type 'long'>
>>> type(i) is long
True
>>> i = 123.456
>>> type(i)
<type 'float'>
>>> type(i) is float
True
```

在 Python3.0 中 int 和 long 已经不再进行区分了.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook 该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
85	1062	632188	10	url

使用列表推导式的方式来创建字典

在 Python2.6 或者之前的版本:

```
d = dict((key, value) for (key, value) in iterable)
```

在 Python2.7 和 3 以后的版本中, 你可以直接使用 [dict comprehension syntax](#):

```
d = {key: value for (key, value) in iterable}
```

当然你可以使用任何包含两个元素的迭代方式来进行生成:

```
d = {value: foo(value) for value in sequence if bar(value)}

def key_value_gen(k):
    yield chr(k+65)
    yield chr((k+13)%26+65)
d = dict(map(key_value_gen, range(26)))
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
86	1055	626395	8	url

如何获取实例的类名

如何才能获取一个实例对象的类名？

我想或许[the inspect module](#)可以实现,但是好像还没有发现什么能帮我的.或许可以分析 `__class__` 成员,但是我不知道怎么用.

你试没试类里的 `__name__` 属性?比如 `x.__class__.__name__` 或许可以得到你想要的

```
>>> import itertools
>>> x = itertools.count(0)
>>> x.__class__.__name__
'count'
```

注:如果你用的是新式类(从Python2.2),你可以调用 `type()` 函数来替代:

```
>>> type(x).__name__
'count'
```

像 `int` 这种内建方法也可以:

```
>>> (5).__class__.__name__
'int'
>>> type(5).__name__
'int'
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
87	1048	2615171	10	url

整数转化为字符串

我希望将整数转为 `string`,我试了如下方法:

```
d = 15
d.str()
```

但是提示 `int` 没有 `str` 方法

```
1641
down vote
accepted
>>> str(10)
'10'
>>> int('10')
1
```

文档链接

- `int()`
- `str`

问题来自这个代码: `d.str()` .

转换是使用的内建 `str()` 函数,调用的是其参数的 `__str__()` 方法

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

找出文件夹中所有扩展名位 .txt 的文件

rank	vote	view	answer	url
88	1044	1193546	30	url

找出文件夹中所有扩展名位 .txt 的文件

可以使用 `glob` :

```
import glob, os
os.chdir("/mydir")
for file in glob.glob("*.txt"):
    print(file)
```

或者 `os.listdir`

```
import os
for file in os.listdir("/mydir"):
    if file.endswith(".txt"):
        print(os.path.join("/mydir", file))
```

或者你想迭代所有的目录, 可以使用 `os.walk`

```
import os
for root, dirs, files in os.walk("/mydir"):
    for file in files:
        if file.endswith(".txt"):
            print(os.path.join(root, file))
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
89	1044	594609	13	url

给字符串填充0

有什么方法可以给字符串左边填充0,这样就可以有一个特定长度.

字符串:

```
>>> n = '4'
>>> print n.zfill(3)
>>> '004'
```

对于数字:

```
>>> n = 4
>>> print '%03d' % n
>>> 004
>>> print format(4, '03') # python >= 2.6
>>> 004
>>> print "{0:03d}".format(4) # python >= 2.6
>>> 004
>>> print("{0:03d}".format(4)) # python 3
>>> 004
```

```
>>> t = 'test'
>>> t.rjust(10, '0')
>>> '000000test'
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间:
2018-10-17 17:33:54

rank	vote	view	answer	url
90	1038	651117	27	url

生成包含大写字母和数字的随机字符串

我希望生成N大小的字符串.

里面只含有数字和大写字母,比如:

- 6U1S75
- 4Z4UKK
- U911K4

有没有什么Pythonic的方法?

一行写的答案:

```
''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(N))
```

为了重用可以这样写:

```
>>> import string
>>> import random
>>> def id_generator(size=6, chars=string.ascii_uppercase + string.digits):
...     return ''.join(random.choice(chars) for _ in range(size))
...
>>> id_generator()
'G5G74W'
>>> id_generator(3, "6793YUI0")
'Y3U'
```

如何工作?

我们引入 `string` 和 `random` 模块, `string` 模块包含了所有常用的ASCII字符, `random` 模块处理随机数的生成.

`string.ascii_uppercase + string.digits` 是一个包含大写字母和数字的列表:

```
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> string.digits
'0123456789'
>>> string.ascii_uppercase + string.digits
'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'
```

然后用列表推导式创建 `n` 个元素的列表:

```
>>> range(4) # n个数字的列表哦
[0, 1, 2, 3]
>>> ['elem' for _ in range(4)] # 在一个列表创建4次'elem'
['elem', 'elem', 'elem', 'elem']
```

在上面的例子中,我们用 `[]` 来创建列表,和 `id_generator` 函数不一样,生成器不再内存中创建列表,只是在程序运行时一个一个的生成列表元素(更多见[这里](#))

下面是在列表中随机取出一个元素:

```
>>> random.choice("abcde")
'a'
>>> random.choice("abcde")
'd'
>>> random.choice("abcde")
'b'
```

所以 `random.choice(chars) for _ in range(size)` 会取出 `size` 个字符.字符随机的在 `chars` 中取出:

```
>>> [random.choice('abcde') for _ in range(3)]
['a', 'b', 'b']
>>> [random.choice('abcde') for _ in range(3)]
['e', 'b', 'e']
>>> [random.choice('abcde') for _ in range(3)]
['d', 'a', 'c']
```

然后我们可以用空字符串把它们连起来成为一个字符串:

```
>>> ''.join(['a', 'b', 'b'])
'abb'
>>> [random.choice('abcde') for _ in range(3)]
['d', 'c', 'b']
>>> ''.join(random.choice('abcde') for _ in range(3))
'dac'
```

2018-10-17 17:33:54

rank	vote	view	answer	url
92	1022	608770	15	url

输出到stderr

我知道至少三种方法这么做:

```
import sys
sys.stderr.write('spam\n')

print >> sys.stderr, 'spam'

from __future__ import print_function
print('spam', file=sys.stderr)
```

但是看起来和python的第13条宗旨相违背,所以有什么更好的方法吗?或者上面那些方法有什么可以改进?

There should be one — and preferably only one — obvious way to do it.

我发现只有这个方法短小+便捷+可读性好:

```
from __future__ import print_function
def warning(*objs):
    print("WARNING: ", *objs, file=sys.stderr)
```

我会选择 `sys.stderr.write()`, 更可读而且言简意赅的突出重点,最主要的是在所有版本都可以用.

注: Pythonic 是除了可读和效率意外才考虑的事情...如果记住前两条的话80%的代码就已经是 Pythonic 了.列表推导式是一个'big thing'所以不经常用(可读性).

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间:
2018-10-17 17:33:54

rank	vote	view	answer	url
93	1017	297432	13	url

在实例名字前单下划线和双下划线的含义

我想刨根问底,这到底是什么意思?解释一下他俩的区别.

单下划线

在一个类中的方法或属性用单下划线开头就是告诉别的程序这个属性或方法是私有的.然而对于这个名字来说并没有什么特别的.

引自[PEP-8](#):

单下划线:"内部使用"的弱指示器.比如,from M import * 将不会引进用单下划线开头的对象.

双下划线

来自[Python文档](#):

任何 `__spam` 形式(至少两个下划线开头,至多一个下划线结尾)都是代替 `__classname__spam`,其中 `classname` 是当前类的名字.This mangling is done without regard to the syntactic position of the identifier.所以它能用来定义私有类的实例和类变量,方法,在全局中的变量,甚至是实例中的变量.可以区别不同类的实例.

例子

```
>>> class MyClass():
...     def __init__(self):
...         self.__superprivate = "Hello"
...         self._semiprivate = ", world!"
...
>>> mc = MyClass()
>>> print mc.__superprivate
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: myClass instance has no attribute '__superprivate'
>>> print mc._semiprivate
, world!
>>> print mc.__dict__
{'_MyClass__superprivate': 'Hello', '_semiprivate': ', world!'}
```

`__foo__` :一种约定,Python内部的名字,用来区别其他用户自定义的命名,以防冲突.

`_foo` :一种约定,用来指定变量私有.程序员用来指定私有变量的一种方式.

`__foo` :这个有真正的意义:解析器用 `_classname__foo` 来代替这个名字,以区别和其他类相同的命名.

在Python中没有其他形式的下划线了.

这种约定方式和类,变量,全局变量等没有区别.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
94	1006	377869	23	url

如何测量脚本运行时间？

我在Project Euler发现好多这样的问题,许多其他地方也问怎么测量执行时间.但是有的时候答案有点kludgy-比如,在 `__main__` 中加入时间代码,所以我想在这里分享一下解决方案.

Python自带了一个叫 `cProfile` 的分析器.它不仅实现了计算整个时间,而且单独计算每个函数运行时间,并且告诉你这个函数被调用多少次,它可以很容易的确定你要优化的值.

你可以在你的代码里或是交互程序里调用,像下面这样:

```
import cProfile
cProfile.run('foo()')
```

更有用的是,你可以在运行脚本的时候用 `cProfile` :

```
python -m cProfile myscript.py
```

为了使用更简单,我做了一个小脚本名字叫 `profile.bat` :

```
python -m cProfile %1
```

所以我可以这么调用了:

```
profile euler048.py
```

下面是结果:

```
1007 function calls in 0.061 CPU seconds
```

```
Ordered by: standard name
```

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
1	0.000	0.000	0.061	0.061	<string>:1(<module>)
1000	0.051	0.000	0.051	0.000	euler048.py:2(<lambda>)
1	0.005	0.005	0.061	0.061	euler048.py:2(<module>)
1	0.000	0.000	0.061	0.061	{execfile}
1	0.002	0.002	0.053	0.053	{map}
1	0.000	0.000	0.000	0.000	{method 'disable' of '_lsprof.Profiler object'}
1	0.000	0.000	0.000	0.000	{range}
1	0.003	0.003	0.003	0.003	{sum}

注:更新一个来自PyCon2013的视频网址: <http://lanyrd.com/2013/pycon/scdywg/>

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

UnicodeEncodeError: 'ascii' codec can't encode character u'\xa0' in position 20: ordinal not in range(128)

rank	vote	view	answer	url
95	994	1197205	19	url

UnicodeEncodeError: 'ascii' codec can't encode character u'\xa0' in position 20: ordinal not in range(128)

你应该去读一读 [Unicode HOWTO](#). 这是个 [very first example](#).

首先不要用 `str` 来进行 unicode 到 text/byte 的转换.

取而代之的使用 `.encode()` :

```
p.agent_info = u' '.join((agent_contact, agent_telno)).encode('utf-8').strip()
```

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
96	987	1016356	21	url

如何连接MySQL?

在python程序里如何链接MySQL数据库?

连接MYSQL需要3步

1 安装

你必须先安装MySQL驱动.和PHP不一样,Python只默认安装了SQLite的驱动.最常用的包是MySQLdb但是用 `easy_install` 安装很困难.

对于Window用户,你可以获取MySQLdb的exe.

对于Linux,可以下载 `python-mysqldb` (可以用 `sudo apt-get install python-mysqldb` 命令直接在命令行下载)

对于Mac用户,可以用Macport下载MySQLdb

2 使用

装完之后重启.这不是强制的,但是这样做可以减少问题.所以请重启.

然后就像用其他包一样:

```
#!/usr/bin/python
import MySQLdb

db = MySQLdb.connect(host="localhost", # your host, usually localhost
                     user="john", # your username
                     passwd="megajonhy", # your password
                     db="jonhydb") # name of the data base

# you must create a Cursor object. It will let
# you execute all the queries you need
cur = db.cursor()

# Use all the SQL you like
cur.execute("SELECT * FROM YOUR_TABLE_NAME")

# print all the first cell of all the rows
for row in cur.fetchall() :
    print row[0]
```

当然,还有许多用法和选项,我只是举了一个基本的例子.你可以看看文档.[A good starting point.](#)

3 高级用法

一旦你知道它是如何工作的,你可能想用ORM来避免手动写入SQL,来把表变成Python对象.Python中最有名的ORM叫做SQLAlchemy

我强烈推荐:你的生活质量肯定能获得提高.

最近在Python里又发现了一个好东西:peewee.它是个非常轻巧的ORM,非常容易安装和使用.我的小项目和独立app都使用它,而那些工具像SQLAlchemy或者Django用在这里有点小题大做了:

```
import peewee
from peewee import *

db = MySQLDatabase('jonhydb', user='john', passwd='megajonhy')

class Book(peewee.Model):
    author = peewee.CharField()
    title = peewee.TextField()

    class Meta:
        database = db

Book.create_table()
book = Book(author="me", title='Peewee is cool')
book.save()
for book in Book.filter(author="me"):
    print book.title

Peewee is cool
```

这个例子可以运行,除了peewee(`pip install peewee`)不需要别的的依赖.安装不复杂.它非常cool!

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
97	974	754731	17	url

如何使用线程

这个是2010年提的问题了,这有一种使用 `map` 和 `pool` 的方法来使用线程.

下面的代码总结自 [Parallelism in one line: A Better Model for Day to Day Threading Tasks](#), 仅仅使用了几行代码:

```
from multiprocessing.dummy import Pool as ThreadPool
pool = ThreadPool(4)
results = pool.map(my_function, my_array)
```

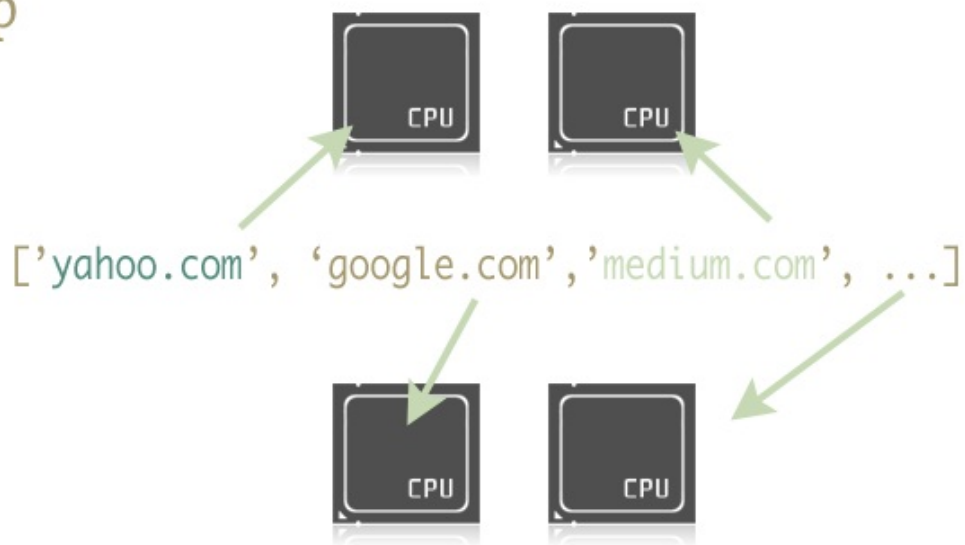
使用 `multithreaded` 的版本:

```
results = []
for item in my_array:
    results.append(my_function(item))
```

解释

`Map` 是一个非常 `cool` 的函数,它可以轻易地使你地 `Python` 代码实现并行.`Map` 是从像 `Lisp` 这种函数式语言学来的.它地作用是对序列中的每个元素进行函数的调用. `Map` 将迭代序列上所有的元素并且调用函数,最后将结果存储到列表中.

Map



实现

并行版本的 Map 需要两个库来实现: multiprocessing 和 很少人知道地 multiprocessing.dummy.

multiprocessing.dummy 和 multiprocessing 模块很像, 但是使用线程进行替代(很重要的区别, 进程处理 CPU 密集型作业, 而线程处理 IO 密集型作业):

multiprocessing.dummy 实现了和 multiprocessing 一样当 API 但是内部使用的是 线程模块


```
import urllib2
from multiprocessing.dummy import Pool as ThreadPool

urls = [
    'http://www.python.org',
    'http://www.python.org/about/',
    'http://www.onlamp.com/pub/a/python/2003/04/17/metaclasses.html',
    'http://www.python.org/doc/',
    'http://www.python.org/download/',
    'http://www.python.org/getit/',
    'http://www.python.org/community/',
    'https://wiki.python.org/moin/',
]

# make the Pool of workers
pool = ThreadPool(4)

# open the urls in their own threads
# and return the results
results = pool.map(urllib2.urlopen, urls)

# close the pool and wait for the work to finish
pool.close()
pool.join()
```

结果如下:

```
Single thread: 14.4 seconds
  4 Pool: 3.1 seconds
  8 Pool: 1.4 seconds
 13 Pool: 1.3 seconds
```

传入多个参数(像这样只在 **Python3.3**及更高版本生效)

传入多个数组:

```
results = pool.starmap(function, zip(list_a, list_b))
```

或者传递一个 `constant` 和数组:

```
results = pool.starmap(function, zip(itertools.repeat(constant), list_a))
```

如果使用老版本的 Python, 可以通过[这个方法](#)来实现

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
98	973	1068581	22	url

检查字符串为空的最优雅的方法

空字符串被认为是 [falsy](#), 意味着转化成布尔类型值为 `false`, 所以你可以:

```
if not myString:
```

在 [Truth Value Testing](#) 可以查询到其他变量转化成布尔类型对应的值.

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook 该文件修订时间 :
2018-10-17 17:33:54

rank	vote	view	answer	url
100	966	240271	10	url

如何在 .format 中使用大括号？

```
x = " \{ Hello \} {0} "  
print x.format(42)
```

得到了: `Key Error: Hello\`

我希望输出: `{Hello} 42`

你需要写两次, `{{` 和 `}}` :

```
>>> x = " {{ Hello }} {0} "  
>>> print x.format(42)  
' { Hello } 42 '
```

文档在这里 [Python documentation for format string syntax](#)

格式化中的大括号 `{}` 表示替代字段.任何不再大括号中的将被看做文本,会被原样输出.如果你在文本中包含大括号,你可以使用 `{{` 和 `}}` .

Copyright © taizilongxu 2018 all right reserved , powered by Gitbook该文件修订时间 :
2018-10-17 17:33:54