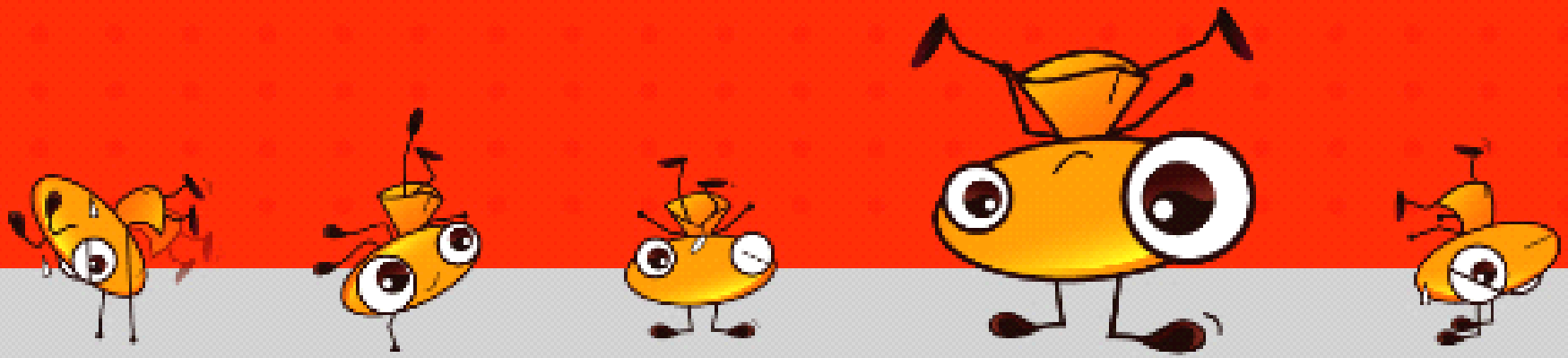
淘宝网
Taobao.com

Pythonic

2010-08-02

smallfish
推特：@nnfish
博客：http://chenxiaoyu.org



KISS

很久很久以前… UNIX，诞生了

很久很久以前… UNIX程序员，越来越多了

许多年过去了…

很久很久以前… 有一个叫Eric S. Raymond大师，写了一本书

《The Art of UNIX Programming》

这本书有啥？

它不是教会你如何写程序，如何干活

它描述的是一种思想，一种文化，一种哲学！

诞生了一个新名词：KISS（请不要有邪恶的想法）

KISS == Keep It Simple and Stupid

Zen of Python

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
...
```

优美胜于丑陋
明了胜于晦涩
简洁胜于复杂
...

简单、简洁、优美

程序，必须是可读的

因为代码是写给人看的

而机器，只是负责来执行的‥

代码，必须是简单的

简单的后果必然臭虫会更少

所以，为了你我的健康

请多写可读和简单，可维护的代码

Style Guide for Python Code
http://www.python.org/dv/peps/pep-0008/

步入正题

新名词：**Pythonic**

这个名词的解释好长好长··

无视把

其实，Python设计的哲学如同UNIX艺术，KISS原则一样

Pythonic代表的也是一种文化，一种哲学！

下面会来一些实例，当然你可以不采取不Pythonic的写法

这一切，亲爱的Guido蜀黍不会介意滴，:_）

当然你看到别人的写法，你会深深自责··

OK，Let’s Go!

万恶的缩进

Python和其他语言最大的不同：

缩进！！！

成也缩进，不成也缩进！

成：

代码更易读
代码风格更一致

不成：

一不小心就抛个可爱的异常： IndentationError

怎么样缩进？

1. 用空格代替tab
2. 不要混合空格和tab
3. 选择一个好的编辑器，比如神器vim，emacs…

命名

关键字：**统一**

其实，相比写程序而言

给变量、函数、类起名，这些才是个技术和艺术活

好的命名方式，程序已经成功了一半

好的命名方式，可以让你的同胞一眼就明白你的意图，而不是靠大段的注释和文档

风格，必须统一！咱必须朝着正规军的方向前进！

建议：

1. 用下划线来连接，比如：get_author，user_name
2. 大小写字符串，比如：getAuthor，userName
3. 类的命名，比如：Student，UserProperty

交换值

一般写法：

```
temp = a;
a = b;
b = temp;
```

当然，一般情况下，这种写法不错，可惜··

别忘记了，我们用的是Python！

只需要一行，没看错！只需要一行！

```
a, b = b, a
```

list

把列表拼成字符串

```
colors = ['red', 'blue', 'green', 'yellow']
result = ''
for s in colors:
    result += s
```

这些当然可以，不过可以来的更简洁点！

```
result = ''.join(colors)
```

注意，比如有一堆字符串操作

```
s += “aaa”
s += “bbb”
s += “ccc”
```

这样，还不如：

```
d = []
d.append(“aaa”)
d.append(“bbb”)
d.append(“ccc”)

s = “”.join(d)
```

dict

突然想输出dict的所有key，怎么办？

```
for key in d.keys():
    print key
```

嗯，这个已经不错了，很可读。但是我们可以做的更好！

```
for key in d:
    print key
```

如果想判断某key在不在dict里，你是不是想到了？

```
if key in d:
    ...do something with d[key]
```

而不是

```
if d.has_key(key):
    ...do something with d[key]
```

set

现在有一数组

```
a = ['aaa', 'bbb', 'ccc', 'aaa', 'ddd', 'aaa']
```

如果想过滤其中重复值，怎么办？

如此：

```
d = {}
for i in a:
    d[i] = 1
```

```
>>> d
{'aaa': 1, 'bbb': 1, 'ccc': 1, 'ddd': 1}
```

是不是可以更简单点？**当然。**

```
>>> list(set(a))
['aaa', 'bbb', 'ccc', 'ddd']
```

list to dict

有两组数据，一个是姓名，一个是城市，想得出一个对应的数据，怎么办呢？

```
>>> name = ["smallfish", "user_a", "user_b"]
>>> city = ["hangzhou", "nanjing", "beijing"]

>>> d = dict(zip(name, city))

>>> print d
{'user_b': 'beijing', 'user_a': 'nanjing', 'smallfish': 'hangzhou'}
```

是不是很简单？

还需要两次for循环，一个临时的变量么？

open file

一般写法是

```
fp = open(“a.txt”)

while True:
    line = fp.readline()
    if not line:
        break
    print line
```

来一个酷的写法把

```
with open("d:/1.log") as fp:
    line = fp.readline()
    print line
```

其他想用with释放资源，请在__exit__里实现

真真假假

如何判断条件为真，一般可以这么写

```
if x == None:
    pass
```

咳咳，请不要忘记，我们用的是Python!

```
if x:
    pass
```

Python 里假的定义

```
False
0
 “” (empty str)
[], (), {}, set()
None
```

输出数组的index和值

一般写法：

```
i = 0
for item in items:
   print i, item
   i += 1
```

改进一下，不需要i这个临时变量：

```
for i in range(len(items)):
    print i, items[i]
```

是不是还不太满意，每次总range，len一下，很烦躁？

```
for(index, item) in enumerate(items):
    print index, item
```

百分号

很久以前，连接字符串和变量，有人这么写··

```
print “name=” + str(name) + “, age=” +int(age)
```

后来在Python里也发觉了类似C的printf函数，而且进化的更优美！

```
print “name=%s, age=%d” % (name, age)
```

上面的只是针对单个变量格式化，如果针对一个dict，是不是要写N个读取变量来格式化？

NO!

```
values = {'name': name, 'messages': messages}
print ('Hello %(name)s, you have %(messages)i '
      'messages' % values)
```

你还可以更懒惰的这么写··

```
print ('Hello %(name)s, you have %(messages)i ‘
       'messages' % locals())
```

Functional Programming（1）

谁说Python不能进行函数式编程了？

lambda

简单函数：

```
def lowercase(x):
    return x.lower()
```

其实可以这样：lowercase = lambda x: x.lower()

filter函数，可以按照条件来过滤数组

```
>>> a = [1, 2, 3, 4, 5]
>>> filter(lambda x : x % 2==0, a)
[2, 4]
```

Functional Programming（2）

map函数，可以对数组每个元素进行操作，当然如果你想or一次，也不反对

```
>>> a = [1, 2, 3, 4, 5]
>>> map(lambda x : x+2, a)
[3, 4, 5, 6, 7]
```

reduce函数，对一个数组进行求和

```
>>> reduce(lambda x, y : x+y, a)
15
```

其实，所有map、reduce、filter能做到的

“列表推导“这个神器都可以做到

而且做的更好，看上去更为优美！

神器：列表推导（1）

什么是列表 推导？它的英文名词好长好长… list comprehension

还是来点实例，我想从一个数组里过滤出一些数值，保存到另外数组里

```
a_list = [1 , 2, 3, 4, 5, 6]
b_list = []

for item in a_list:
    if item % 2 == 0:
        b_list.append(item)
```

好吧，是不是大家都这么写？ 那多无趣…

```
b_list = [item for item in a_list if item % 2 == 0]

>>> b_list
[2, 4, 6]
```

适用于list、tuple、string…

神器：列表推导（2）

重新实现前面的map、reduce、filter

map，数组里每个值都乘以2

```
>>> [i * 2 for i in range(1, 10)]
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

reduce，求和

```
>>> sum(i for i in range(1, 10))
45
```

filter，过滤数字

```
>>> [i for i in range(1, 10) if i % 2 == 0]
[2, 4, 6, 8]
```

还需要for循环么？就是如此简单，就是如此的酷！

当然，一切千万别过渡滥用 ☹

Decorator（1）

介个新特性是2.4之后新增的，很帅很酷！

问题：我想记录每个函数执行的时间，怎么办？

```
start = time.time()
… run many many code
print time.time() - start
```

难道每个函数都如此写么？太龌龊了把。。

先来一个包装函数

```
def func_time(func):
    def _wrapper(*args, **kwargs):
        start = time.time()
        func(*args, **kwargs)
        print func.__name__, “run:”, time.time()-start
    return _wrapper
```

Decorator（2）

再来一个普通函数

```
def sum(n):
    sum = 0
    for i in range(n):
        sum += i
    return sum
```

其实我们可以这样调用

```
a = func_time(sum)
a(100)
```

有一种更加简便 方法

```
@func_time
def sum(n):
    …. code
```

调用一下把

```
>>> sum(1000000)
sum run: 0.265000104904
```

看到了么？

如果只是这么一个函数，这么写没啥好处

如果你有一堆一堆

只需要在函数上面加一个@func_time

不需要改动函数本身

多酷！

Decorator（3）

更多应用场景

Web 权限校验

Cache

…

其实在先用的Python应用里可以看到很多修饰器的应用

比如：

Web.py
Django
Tornado
Flask
….

还有很多很多…

吹水结束

天色已晚，大家洗洗睡吧

谢谢各位！