**[python学习-字符串前面添加u,r,b的含义](https://www.cnblogs.com/cq90/p/6959567.html)**

u/U:表示unicode字符串   
不是仅仅是针对中文, 可以针对任何的字符串，代表是对字符串进行unicode编码。   
一般英文字符在使用各种编码下, 基本都可以正常解析, 所以一般不带u；但是中文, 必须表明所需编码, 否则一旦编码转换就会出现乱码。   
建议所有编码方式采用utf8

r/R:非转义的原始字符串   
与普通字符相比，其他相对特殊的字符，其中可能包含转义字符，即那些，反斜杠加上对应字母，表示对应的特殊含义的，比如最常见的”\n”表示换行，”\t”表示Tab等。而如果是以r开头，那么说明后面的字符，都是普通的字符了，即如果是“\n”那么表示一个反斜杠字符，一个字母n，而不是表示换行了。   
以r开头的字符，常用于正则表达式，对应着re模块。

b:bytes   
python3.x里默认的str是(py2.x里的)unicode, bytes是(py2.x)的str, b”“前缀代表的就是bytes   
python2.x里, b前缀没什么具体意义， 只是为了兼容python3.x的这种写法

**（1）以r或R开头的python中的字符串表示（非转义的）原始字符串**

python里面的字符，如果开头处有个r，比如：

(r’^time/plus/\d{1,2}/$’, hours\_ahead)

说明字符串r"XXX"中的XXX是普通字符。

有普通字符相比，其他相对特殊的字符，其中可能包含转义字符，即那些，反斜杠加上对应字母，表示对应的特殊含义的，比如最常见的”\n"表示换行，"\t"表示Tab等。

而如果是以r开头，那么说明后面的字符，都是普通的字符了，即如果是“\n”那么表示一个反斜杠字符，一个字母n，而不是表示换行了。

以r开头的字符，常用于正则表达式，对应着re模块。

关于re模块，详情自己google搜索“python re”。

举例：

原始字符串操作符(r/R),能方便处理反斜杠:

[?](https://www.crifan.com/python_string_with_leading_char_r_u/)

|  |  |
| --- | --- |
| 1  2  3  4 | f = open(r'C:\Program Files\Adobe\Reader 9.0\Setup Files\setup.ini','r')  for i in f:      print i  f.close() |

**（2）以u或U开头的字符串表示unicode字符串**

Unicode是书写国际文本的标准方法。如果你想要用非英语写文本,那么你需要有一个支持Unicode的编辑器。

类似地,Python允许你处理Unicode文本——你只需要在字符串前加上前缀u或U。

举例：

|  |
| --- |
| u"This is a Unicode string." |

[**python的str.format方法**](https://www.cnblogs.com/Alexzzzz/p/6832253.html)

format方法被用于字符串的格式化输出。

print('{0}+{1}={2}'.format(1,2,1+2)) #in

1+2=3 #out

可见字符串中大括号内的数字分别对应着format的几个参数。

若省略数字：

print('{}+{}={}'.format(1,2,1+2)) #in

可以得到同样的输出结果。但是替换顺序默认按照[0],[1],[2]...进行。

若替换{0}和{1}：

print('{1}+{0}={2}'.format(1,2,1+2)) #in

2+1=3 #out

输出字符串:

print('{0} am {1}'.format('i','alex'))

i am alex #out

输出参数的值:

1 length = 4

2 name = 'alex'

3 print('the length of {0} is {1}'.format(name,length))

the length of alex is 4

精度控制：

print('{0:.3}'.format(1/3))

0.333

宽度控制：

print('{0:7}{1:7}'.format('use','python'))

use python

精宽度控制(宽度内居左)：

print('{0:<7.3}..'.format(1/3))

0.333 ..

其实精宽度控制很类似于C中的printf函数。

同理'>'为居右，'^'为居中。符号很形象。

补全：

[复制代码](javascript:void(0);)

1 #!/usr/bin/python

2 #python3.6

3 print('{0:0>3}'.format(1)) #居右，左边用0补全

4 print('{0:{1}>3}'.format(1,0)) #也可以这么写

5 #当输出中文使用空格补全的时候，系统会自动调用英文空格，这可能会造成不对齐

6 #for example

7 blog = {'1':'中国石油大学','2':'浙江大学','3':'南京航空航天大学'}

8 print('不对齐：')

9 print('{0:^4}\t\t{1:^8}'.format('序号','名称'))

10 for no,name in blog.items(): #字典的items()方法返回一个键值对，分别赋值给no和name

11 print('{0:^4}\t\t{1:^8}'.format(no,name))

12 print('\n对齐：')

13 print('{0:^4}\t\t{1:{2}^8}'.format('序号','名称',chr(12288))) #chr(12288)为UTF-8中的中文空格

14 for no,name in blog.items():

15 print('{0:^4}\t\t{1:{2}^8}'.format(no,name,chr(12288)))

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

#out

001

001

不对齐：

序号 名称

1 中国石油大学

2 浙江大学

3 南京航空航天大学

对齐：

序号 　　　名称

1 　中国石油大学

2 　　浙江大学

3 南京航空航天大学

[复制代码](javascript:void(0);)

# [python -- lambda表达式](https://www.cnblogs.com/hf8051/p/8085424.html)

lambda表达式，通常是在**需要一个函数，但是又不想费神去命名一个函数**的场合下使用，也就是指**匿名函数**。

lambda所表示的匿名函数的内容应该是很简单的，如果复杂的话，干脆就重新定义一个函数了，使用lambda就有点过于执拗了。

lambda就是用来定义一个匿名函数的，如果还要给他绑定一个名字的话，就会显得有点画蛇添足，通常是直接使用lambda函数。如下所示：

add = lambda x, y : x+y

add(1,2) # 结果为3

那么到底要如何使用lambda表达式呢？

**1、应用在函数式编程中**

Python提供了很多函数式编程的特性，如：map、reduce、filter、sorted等这些函数都支持函数作为参数，lambda函数就可以应用在函数式编程中。如下：

# 需求：将列表中的元素按照绝对值大小进行升序排列

list1 = [3,5,-4,-1,0,-2,-6]

sorted(list1, key=lambda x: abs(x))

当然，也可以如下：

list1 = [3,5,-4,-1,0,-2,-6]

def get\_abs(x):

return abs(x)

sorted(list1,key=get\_abs)

只不过这种方式的代码看起来不够Pythonic

**2、应用在闭包中**

def get\_y(a,b):

return lambda x:ax+b

y1 = get\_y(1,1)

y1(1) # 结果为2

当然，也可以用常规函数实现闭包，如下：

def get\_y(a,b):

def func(x):

return ax+b

return func

y1 = get\_y(1,1)

y1(1) # 结果为2

只不过这种方式显得有点啰嗦。

那么是不是任何情况下lambda函数都要比常规函数更清晰明了呢？

肯定不是。

Python之禅中有这么一句话：Explicit is better than implicit（明了胜于晦涩），就是说那种方式更清晰就用哪一种方式，不要盲目的都使用lambda表达式。

# Iterator

Python的内建模块itertools提供了非常有用的用于操作迭代对象的函数。

首先，我们看看itertools提供的几个“无限”迭代器：

>>> **import** itertools

>>> natuals = itertools.count(1)

>>> **for** n **in** natuals:

... print(n)

...

1

2

3

...

因为count()会创建一个无限的迭代器，所以上述代码会打印出自然数序列，根本停不下来，只能按Ctrl+C退出。

cycle()会把传入的一个序列无限重复下去：

>>> **import** itertools

>>> cs = itertools.cycle('ABC') *# 注意字符串也是序列的一种*

>>> **for** c **in** cs:

... print(c)

...

'A'

'B'

'C'

'A'

'B'

'C'

...

同样停不下来。

repeat()负责把一个元素无限重复下去，不过如果提供第二个参数就可以限定重复次数：

>>> ns = itertools.repeat('A', 3)

>>> **for** n **in** ns:

... print(n)

...

A

A

A

无限序列只有在for迭代时才会无限地迭代下去，如果只是创建了一个迭代对象，它不会事先把无限个元素生成出来，事实上也不可能在内存中创建无限多个元素。

无限序列虽然可以无限迭代下去，但是通常我们会通过takewhile()等函数根据条件判断来截取出一个有限的序列：

>>> natuals = itertools.count(1)

>>> ns = itertools.takewhile(**lambda** x: x <= 10, natuals)

>>> list(ns)

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

itertools提供的几个迭代器操作函数更加有用：

### chain()

chain()可以把一组迭代对象串联起来，形成一个更大的迭代器：

>>> **for** c **in** itertools.chain('ABC', 'XYZ'):

... print(c)

*# 迭代效果：'A' 'B' 'C' 'X' 'Y' 'Z'*

### groupby()

groupby()把迭代器中相邻的重复元素挑出来放在一起：

>>> **for** key, group **in** itertools.groupby('AAABBBCCAAA'):

... print(key, list(group))

...

A ['A', 'A', 'A']

B ['B', 'B', 'B']

C ['C', 'C']

A ['A', 'A', 'A']

实际上挑选规则是通过函数完成的，只要作用于函数的两个元素返回的值相等，这两个元素就被认为是在一组的，而函数返回值作为组的key。如果我们要忽略大小写分组，就可以让元素'A'和'a'都返回相同的key：

>>> **for** key, group **in** itertools.groupby('AaaBBbcCAAa', **lambda** c: c.upper()):

... print(key, list(group))

...

A ['A', 'a', 'a']

B ['B', 'B', 'b']

C ['c', 'C']

A ['A', 'A', 'a']

### 练习

计算圆周率可以根据公式：

利用Python提供的itertools模块，我们来计算这个序列的前N项和：

窗体顶端

# -\*- coding: utf-8 -\*-

import itertools



# 测试:

print(pi(10))

print(pi(100))

print(pi(1000))

print(pi(10000))

assert 3.04 < pi(10) < 3.05

assert 3.13 < pi(100) < 3.14

assert 3.140 < pi(1000) < 3.141

assert 3.1414 < pi(10000) < 3.1415

print('ok')

 Run

窗体底端

### 小结

itertools模块提供的全部是处理迭代功能的函数，它们的返回值不是list，而是Iterator，只有用for循环迭代的时候才真正计算。

# Datetime类

<https://www.cnblogs.com/cindy-cindy/p/6720196.html>