**1、python中常用的数据结构有哪些？请简要介绍一下。**

数字：int、float、bool、complex

字符串：str

列表：list，用[]

元组：tuple，用()，就相当于不可修改的list

集合：set，用{}

字典：dict，用{key: value}

其中，列表，元组和字符串可以统一归为**序列类**，即这三种数据结构中的元素是有序的。比如，他们都有索引（下标）操作，还有切片、相加和长度(len)，最大值(max)，最小值(min)操作。这是他们的共同点。

补充：python中常见的数据结构可以统称为容器（container）。

序列（如列表和元组）、映射（如字典）以及集合（set）是三类主要的容器。

另外，关于这个问题，面试官很容易引出另一个问题：

**1-1、python中的哪些数据类型是可变的，哪些是不可变的？**

首先，可变/不可变是针对该对象所指向的内存中的值是否可变来判断的。

如可变类型的数据类型有列表和字典，还有集合。

不可变类型的数据类型有字符串，元组，数字。

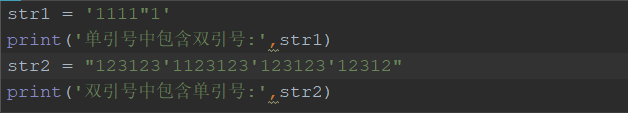
就举个最简单的数字的例子，python中有小整数池的概念，即[-5,256]范围内的整数，python解释器对他们做了特殊处理，都放在内存中的固定位置，不会因为你的操作而发生变化。

现在：a = 1 ，然后我们又重新对a赋值，a = 2，在重新赋值的过程中，整数1所对应的内存地址没有和数字的大小都没有发生变化，还在内存中的固定位置。整数2也是如此。变化的是a的指针（这里引用C中的概念）从指向数字1变成数字2。a对象指向的内存中的值没有发生变化，因此数字是不可变类型的数据类型。字符串，元组也是同理。

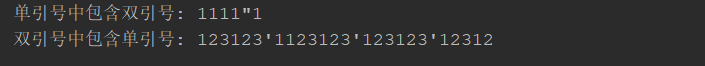
##### 2、简要描述python中单引号、双引号、三引号的区别。

* 首先，单引号和双引号在使用时基本上没有什么区别，唯一需要注意的是：

**当字符串中有单引号时，最好在外面使用双引号；当有双引号时，最好在外面使用单引号。**



**输出**



* 三引号一般不常用，除了用来做注释之外，还可以用来打印多行字符串。特殊用途，是可以打印多行字符串。



**输出**



* 如果单引号和双引号想实现以下的换行效果，需要加上换行符



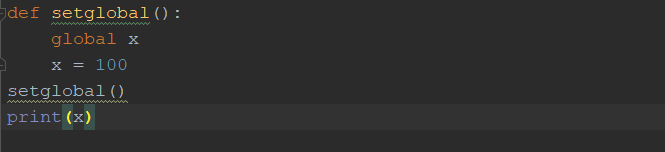
**输出**



**3、[如何在一个function里面设置一个全局变量](https://www.cnblogs.com/apollo1616/articles/9785060.html)**

如果要给全局变量在一个函数里赋值，必须使用global语句。global VarName的表达式会告诉Python，VarName是一个全局变量，这样Python就不会在局部命名空间里寻找这个变量了。

下面这个代码只有调用了函数,在打印才有效,否则没结果



输出

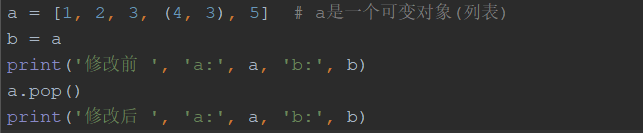


##### 4、python里面如何拷贝一个对象？（赋值、浅拷贝、深拷贝的区别）

在python中如何拷贝一个对象是需要根据具体的需求来定的。

1. 赋值：其实就是对象的引用。相当于C中的指针，修改了其中一个对象，另一个跟着改变。注意对于不可变对象而言，如果修改了其中一个对象，就相当于修改它的指针指向，另一个对象是不会跟着变化的。

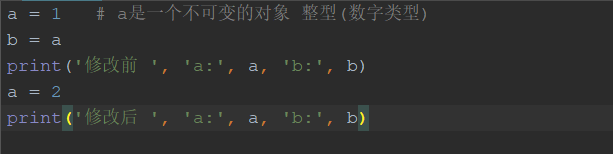
* 修改可变对象



输出



* 修改不可变对象

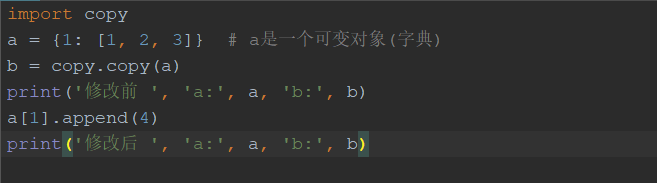


输出



1. 浅拷贝：拷贝父对象，但是不会拷贝父对象的子对象。（具体的方法有：b = copy.copy(a)，切片如b = a[1:4]）

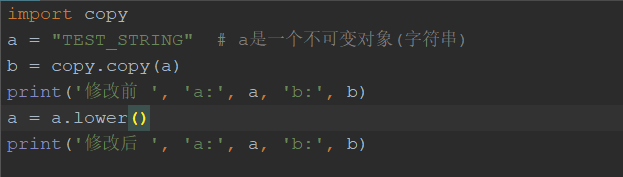
* 修改可变对象



输出



* 修改不可变对象

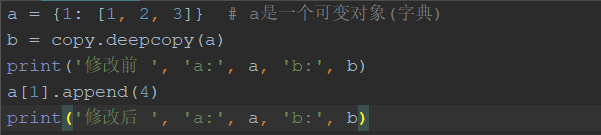


输出

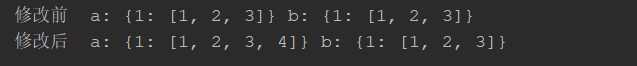


1. 深拷贝：完全拷贝了父对象和子对象（具体的方法有：b = copy.deepcopy(a)）

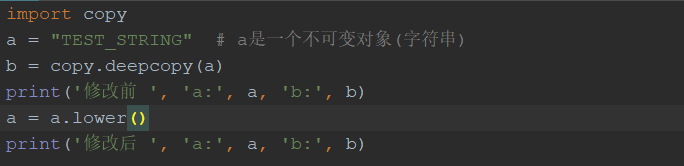
* 修改可变对象



输出



* 修改不可变对象

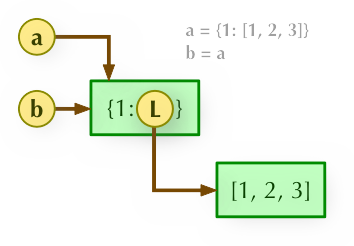


输出

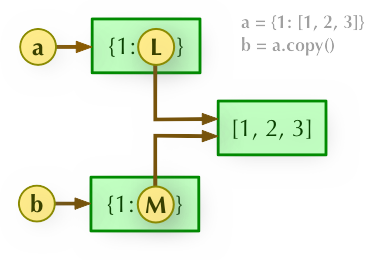


下面是图解：

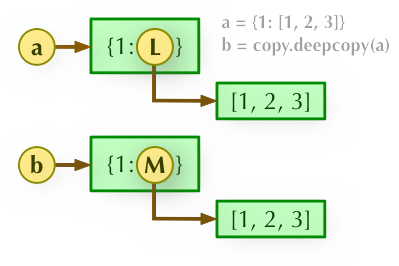
1、b = a: 赋值引用，a 和 b 都指向同一个对象。



2、b = a.copy(): 浅拷贝, a 和 b 是一个独立的对象，但他们的子对象还是指向统一对象（是引用）。



3、b = copy.deepcopy(a): 深度拷贝, a 和 b 完全拷贝了父对象及其子对象，两者是完全独立的。



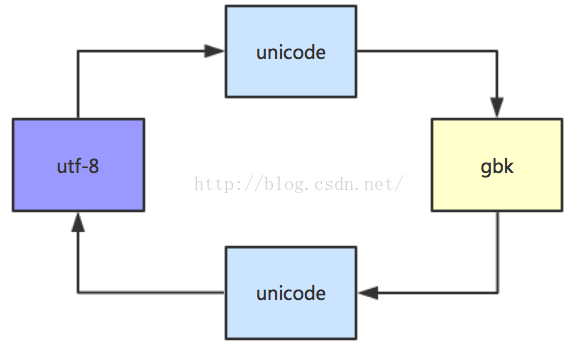
总结：

（1）当对象为不可变类型时，不论是赋值，浅拷贝还是深拷贝，那么改变其中一个值时，另一个都是不会跟着变化的。

（2）当对象为可变对象时，如果是赋值和浅拷贝，那么改变其中任意一个值，那么另一个会跟着发生变化的；如果是深拷贝，是不会跟着发生改变的。

##### 5、如果custname字符串的编码格式为uft-8,如何将custname的内容转化为gb18030的字符串？

**注意：**unicode编码是一种二进制编码，是转换编码的中间桥梁。比如需要将utf-8转换为gbk，那么就需要先转换为unicode（decode），再转为gbk（encode）。

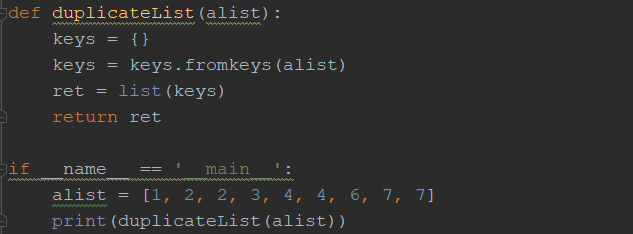


##### 6、请写出一段python代码实现删除list中的重复元素。

三种方法：

1. 利用字典的fromkeys来自动过滤重复值
2. 利用集合set的特性，元素是非重复的
3. 利用遍历去重的传统方式

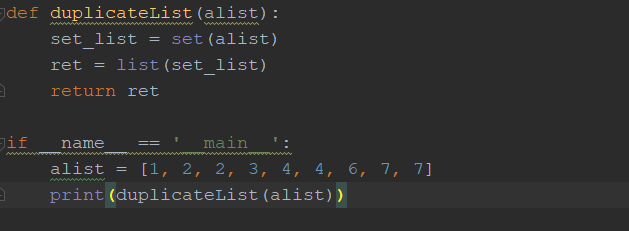
* 方法一：



输出



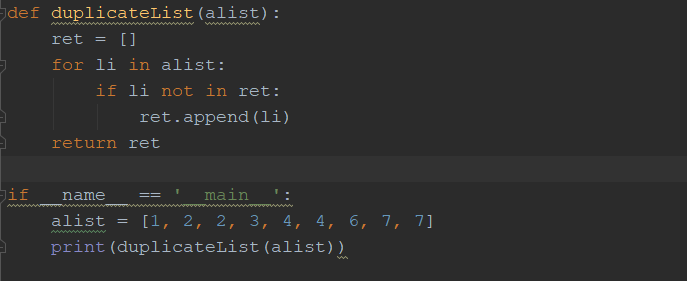
* 方法二：



输出



* 方法三：



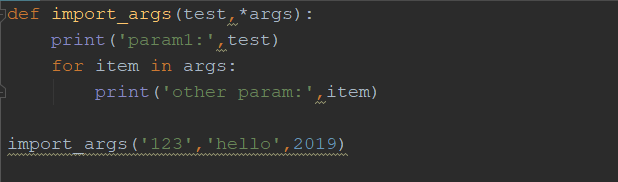
输出



##### 7、这两个参数是什么意思？*args和*kwargs。

首先，我想说的是\*args和\*\*kwargs并不是必须这样写，只有前面的\*和\*\*才是必须的。你可以写成\*var和\*\*vars。而写成\*args和\*\*kwargs只是约定俗称的一个命名规定。

\*args和\*\*kwargs主要用于函数定义，你可以将不定量的参数传递给一个函数。其中，**\*args** 是用来发送一个**非键值对**的可变数量的参数列表给一个函数；**\*\*kwargs** 允许你将不定长度的**键值对**, 作为参数传递给一个函数。 如果你想要在一个函数里处理**带名字的参数**, 你应该使用\*\*kwargs。

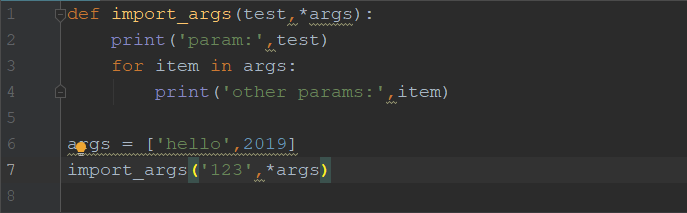


输出



这里传递了3个参数，按位置传参，'123'为test传参，'hello'和2019为\*args传参，这里传了2个参数。

注意，看下面的\*args的另一种用法：用来解压数据。



输出



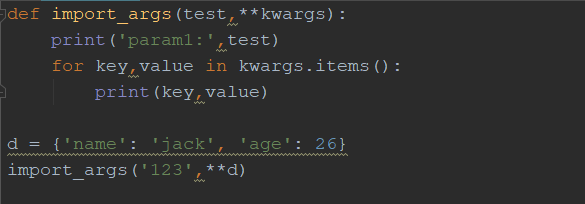
这段代码和上面的效果是一样的，但是这里第7行的\*args和第1行的\*args可是不一样的。

**第1行是表示函数可以接受不定数量的非键值对的参数，用来传参使用的。**

**第7行是用来解压列表['hello', '2019']的每一项数据的，用来解压参数的。**

这是**\*args的两种用法**，也可说是**\*的两种用法**，因为args是可变的。

接下来说说\*\*kwargs



输出



\*\*kwargs用来传递带键值对的参数，而\*\*也是用来解压字典容器内的参数。

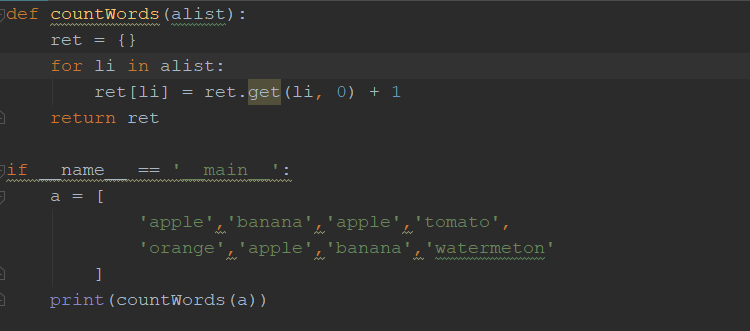
总结：**\*args和\*\*kwargs都是用于函数中传递参数的，\*args传递的是非键值对的参数，\*\*kwargs传递的是带键值对的参数，如果还有普通参数需要传递，那么应该先传递普通的参数。**

##### 8、编码题

##### （1）统计如下list单词及其出现的次数。

a=['apple', 'banana', 'apple', 'tomato', 'orange', 'apple', 'banana','watermeton']

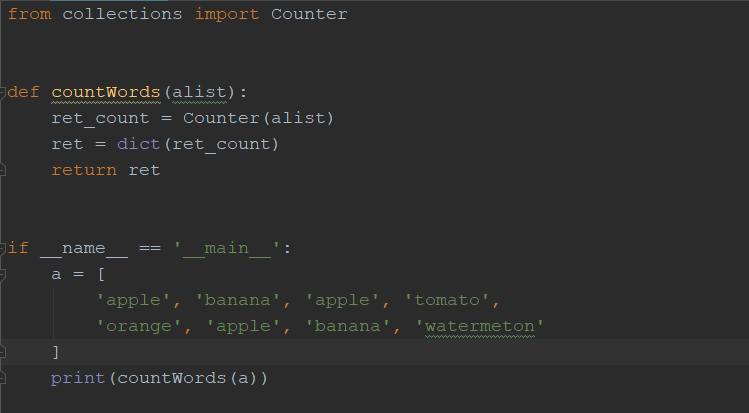
* 方法一：利用字典，遍历列表



输出



* 方法二：使用Python的Collection包



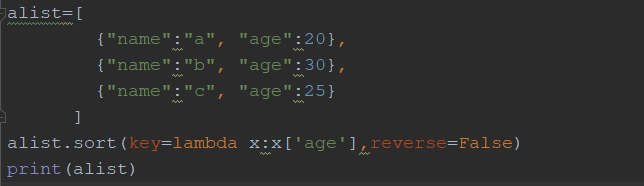
输出



##### （2）给列表中的字典排序：例如有如下list对象：

alist=[{"name":"a", "age":20}, {"name":"b", "age":30}, {"name":"c", "age":25}] 将alist中的元素按照age从小到大排序。

利用list的内建函数，list.sort()来进行排序。



输出



##### （3）写出下列代码的运行结果

* 第一段



输出

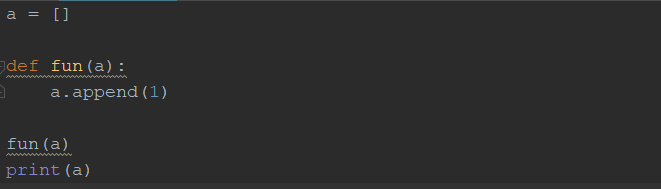


第一段代码的运行结果为：1

分析，在函数外面定义了一个全局变量a为1，在函数内部定义了一个局部变量a为2。局部变量在离开函数后就失效了。

所以，结果为全局变量的a的值。如果在a=2之前加上global a，声明为全局变量，那么结果为2。

* 第二段



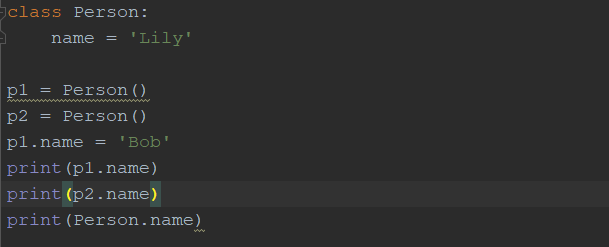
输出



第二段代码的运行结果为：[1]

这是因为，将a传入到function中，这相当于对a进行赋值引用。由于a是可变类型的，所以在函数内部修改a的时候，外部的全局变量a也跟着变化。

* 第三段



输出



##### 1、scrapy框架专题部分

##### （1）请简要介绍下scrapy框架

scrapy 是一个快速(fast)、高层次(high-level)的基于 python 的 web 爬虫构架，用于抓取web站点并从页面中提取结构化的数据。scrapy 使用了 Twisted异步网络库来处理网络通讯。

##### （2）为什么要使用scrapy框架？scrapy框架有哪些优点？

* 它更容易构建大规模的抓取项目
* 它异步处理请求，速度非常快
* 它可以使用自动调节机制自动调整爬行速度

##### （3）scrapy框架有哪几个组件/模块？简单说一下工作流程。

**Scrapy Engine**

这是引擎，负责Spiders、ItemPipeline、Downloader、Scheduler中间的通讯，信号、数据传递等等！（像不像人的身体？）

**Scheduler(调度器)**

它负责接受引擎发送过来的requests请求，并按照一定的方式进行整理排列，入队、并等待Scrapy Engine(引擎)来请求时，交给引擎。

**Downloader(下载器)**

负责下载Scrapy Engine(引擎)发送的所有Requests请求，并将其获取到的Responses交还给Scrapy Engine(引擎)，由引擎交给Spiders来处理，

**Spiders**

它负责处理所有Responses,从中分析提取数据，获取Item字段需要的数据，并将需要跟进的URL提交给引擎，再次进入Scheduler(调度器)，

**Item Pipeline**

它负责处理Spiders中获取到的Item，并进行处理，比如去重，持久化存储（存数据库，写入文件，总之就是保存数据用的）

**Downloader Middlewares(下载中间件)**

你可以当作是一个可以自定义扩展下载功能的组件

**Spider Middlewares(Spider中间件)**

你可以理解为是一个可以自定扩展和操作引擎和Spiders中间‘通信‘的功能组件（比如进入Spiders的Responses;和从Spiders出去的Requests）

* 整体架构如下图：



* 工作流程：

数据在整个Scrapy的流向：

程序运行的时候，

引擎：Hi！Spider, 你要处理哪一个网站？

Spiders：我要处理23wx.com

引擎：你把第一个需要的处理的URL给我吧。

Spiders：给你第一个URL是XXXXXXX.com

引擎：Hi！调度器，我这有request你帮我排序入队一下。

调度器：好的，正在处理你等一下。

引擎：Hi！调度器，把你处理好的request给我，

调度器：给你，这是我处理好的request

引擎：Hi！下载器，你按照下载中间件的设置帮我下载一下这个request

下载器：好的！给你，这是下载好的东西。（如果失败：不好意思，这个request下载失败，然后引擎告诉调度器，这个request下载失败了，你记录一下，我们待会儿再下载。）

引擎：Hi！Spiders，这是下载好的东西，并且已经按照Spider中间件处理过了，你处理一下（注意！这儿responses默认是交给def parse这个函数处理的）

Spiders：（处理完毕数据之后对于需要跟进的URL），Hi！引擎，这是我需要跟进的URL，将它的responses交给函数 def  xxxx(self, responses)处理。还有这是我获取到的Item。

引擎：Hi ！Item Pipeline 我这儿有个item你帮我处理一下！调度器！这是我需要的URL你帮我处理下。然后从第四步开始循环，直到获取到你需要的信息，

注意！只有当调度器中不存在任何request了，整个程序才会停止，（也就是说，对于下载失败的ＵＲＬ，Scrapy会重新下载。）

以上就是Scrapy整个流程了。

##### （4）scrapy如何实现分布式抓取？

可以借助scrapy\_redis类库来实现。

在分布式爬取时，会有master机器和slave机器，其中，master为核心服务器，slave为具体的爬虫服务器。

我们在master服务器上搭建一个redis数据库，并将要抓取的url存放到redis数据库中，所有的slave爬虫服务器在抓取的时候从redis数据库中去链接，由于scrapy\_redis自身的队列机制，slave获取的url不会相互冲突，然后抓取的结果最后都存储到数据库中。master的redis数据库中还会将抓取过的url的指纹存储起来，用来去重。相关代码在dupefilter.py文件中的request\_seen()方法中可以找到。

去重问题：   
dupefilter.py 里面的源码：   
def request\_seen(self, request):   
fp = request\_fingerprint(request)   
added = self.server.sadd(self.key, fp)   
return not added   
去重是把 request 的 fingerprint 存在 redis 上，来实现的。

##### 2、其他常见问题

##### （1）爬虫使用多线程好？还是多进程好？为什么？

对于IO密集型代码（文件处理，网络爬虫），多线程能够有效提升效率（单线程下有IO操作会进行IO等待，会造成不必要的时间等待，而开启多线程后，A线程等待时，会自动切换到线程B，可以不浪费CPU的资源，从而提升程序执行效率）。

在实际的采集过程中，既考虑网速和相应的问题，也需要考虑自身机器硬件的情况，来设置多进程或者多线程。

##### （2）http和https的区别？

A. http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

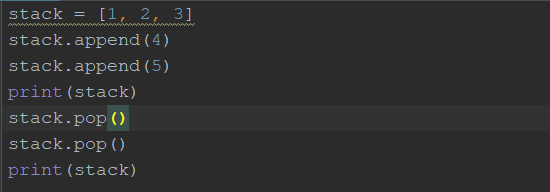
B. http适合于对传输速度、安全性要求不是很高，且需要快速开发的应用。如web应用，小的手机游戏等等。而https适用于任何场景。

##### （3）数据结构之堆，栈和队列的理解和实现。

栈（stacks）：栈的特点是后进先出。只能通过访问一端来实现数据的储存和检索的线性数据结构。

队列（queue）：队列的特点是先进先出。元素的增加只能在一端，元素的删除只能在另一端。增加的一端称为队尾，删除的一端称为队首。

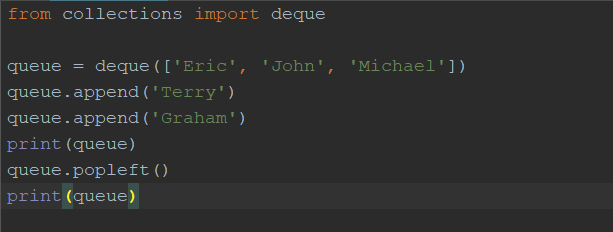
**栈：**



输出结果：



**队列：**



输出结果



这里还会有一个常见的问题，**栈溢出的常见情况及解决方案。**

##### 什么是栈溢出？

因为栈一般默认为1-2m，一旦出现死循环或者是大量的递归调用，在不断的压栈过程中，造成栈容量超过1m而导致溢出。

##### 栈溢出的几种情况？

1、局部数组过大。当函数内部数组过大时，有可能导致堆栈溢出。

2、递归调用层次太多。递归函数在运行时会执行压栈操作，当压栈次数太多时，也会导致堆栈溢出。

##### 解决方法：

1、用栈把递归转换成非递归。

2、增大栈空间。