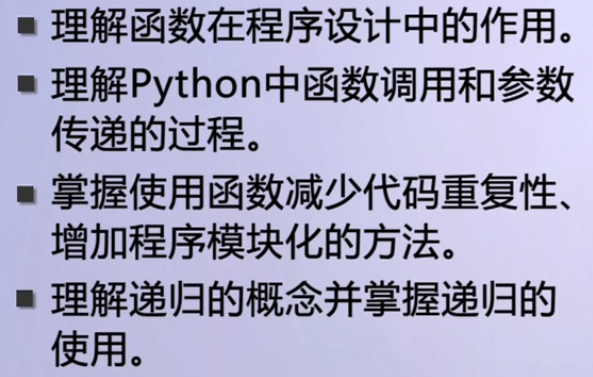
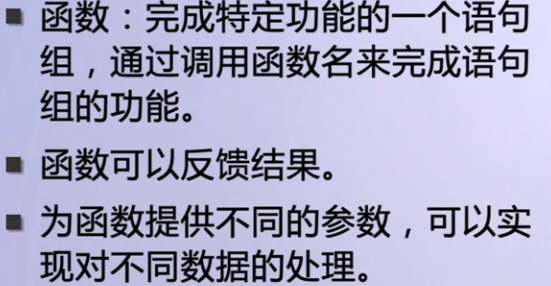
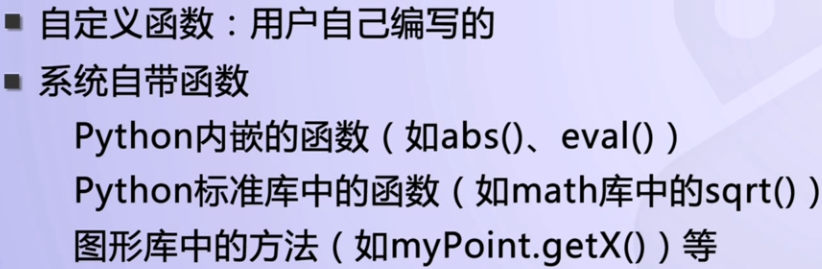
Python中的函数

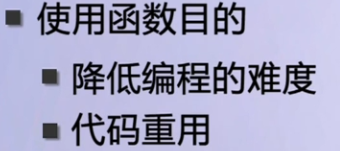
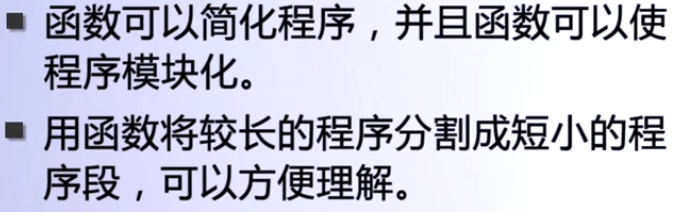
# 函数、模块、对象是Python中的重要的东西。

# 目标要求



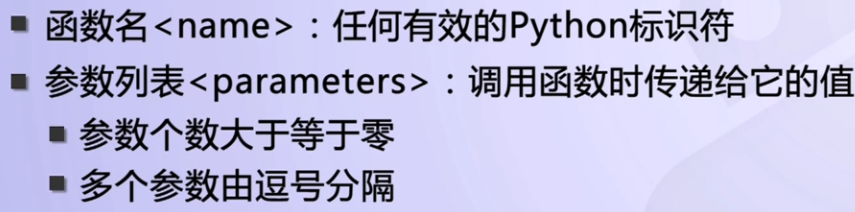


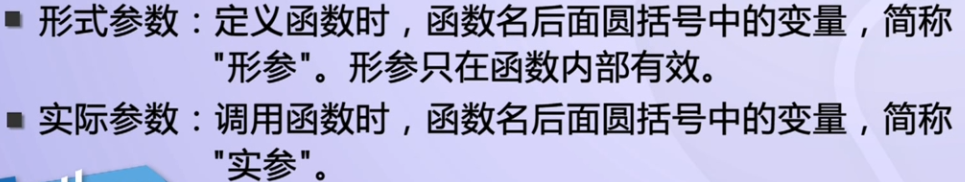


# 函数的创建：def关键字用来创建函数。









## 无参函数：

**def 函数名():**

**函数体**

**(return result) 可以没有返回语句。**

## 有参函数：参数的个数没有限制。

**def 函数名(变量名…):**

**函数体**

**(return result) 可以没有返回语句。**

## 关键参数：设置参数默认值

**由于Python的变量没有指定具体的数据类型，所以在传入函数参数时，很容易混乱，故需要引入一些方法。**

**可以在调用函数的时候，即传入参数的时候，进行变量赋值式传入，不需要特定的参数顺序。**

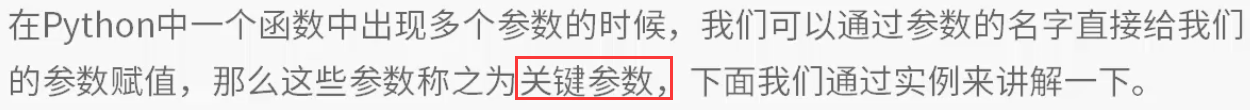
**通过变量赋值的方式传入数值，此时不需要考虑参数的顺序**。







**另外，可以在定义函数的时候，对参数进行初始化。**



**在定义的时候，如果进行了初始化，那么在调用的时候，可以不传入新值，就会默认采用默认值；也可以采用新值，覆盖原来的值。**

def func(a=160,b=170,c=180):  
 max = a if a>b else b;  
 max = max if max > c else c;  
 print('最大值:{0}'.format(max))  
func()#最大值:180  
func(190,120,150)#最大值:190  
func(190,b=120,c=150)#最大值:190  
func(c=120,b=150)#最大值:160  
# func(a=120,150)#SyntaxError: positional argument follows keyword argument

**在定义函数时，对参数进行初始化，如果没有传入新的值，则就会覆盖，如果没有新值，则会使用初始化的值。**

# 函数文档字符串

**利用三重单引号或双引号，书写函数文档**。

利用 **funcName.\_\_doc\_\_或help(funcName)**查看函数功能介绍；

def max(a,b):  
 *'''* ***:param*** *a: the first param* ***:param*** *b: the second param* ***:return****: the max value  
 '''* return a if a>b else b;  
  
print(max.\_\_doc\_\_)  
help(max)

输出结果：

:param a: the first param

:param b: the second param

:return: the max value

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

max(a, b)

:param a: the first param

:param b: the second param

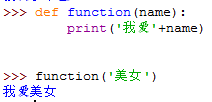
:return: the max value

# return语句及返回值

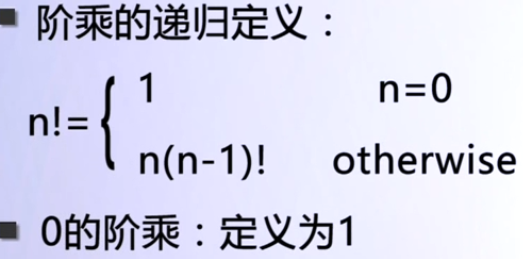
# 函数的调用方法：

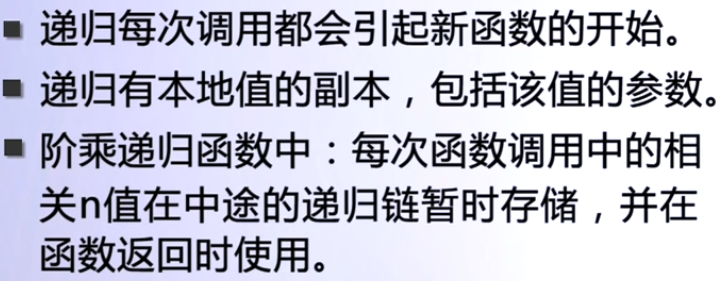
**（1）无参：函数名()**

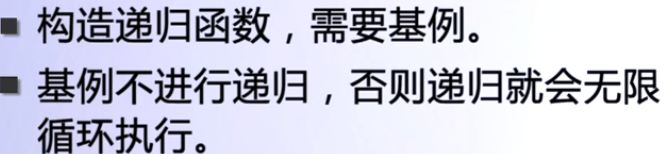
**（2）有参：函数名（值…）**

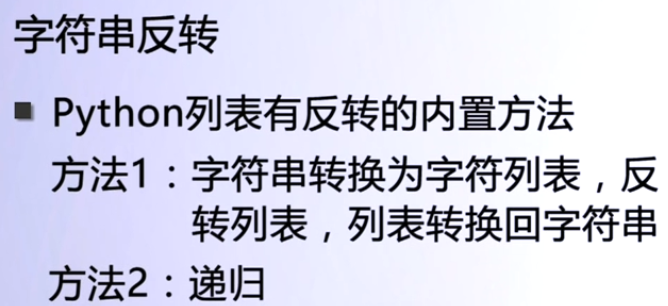


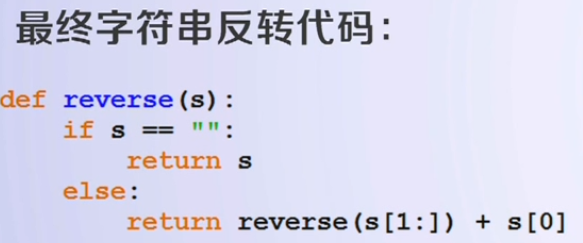
# 递归







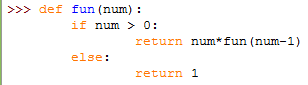




递归有一定的深度：



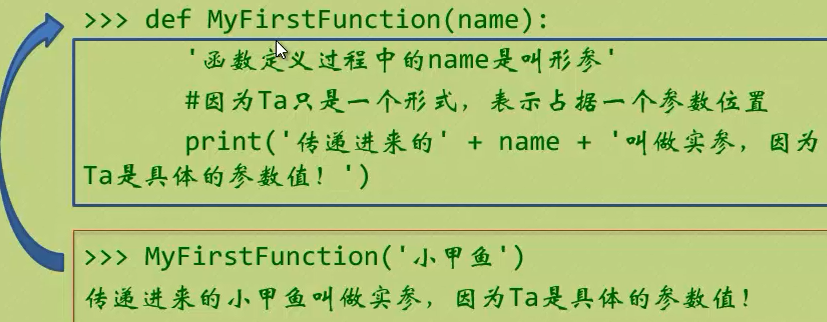
1. **递归函数求解阶乘：**

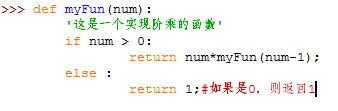


1. **函数文档：写函数文档很重要，主要是给别人看懂这个函数的功能,便于别人对这个函数进行调试。**

**在函数定义名字下面，加上一个字符串即为函数文档。**

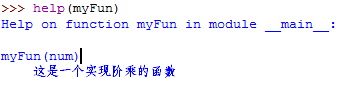
**区别：利用#号是Python做注释的标识符。**



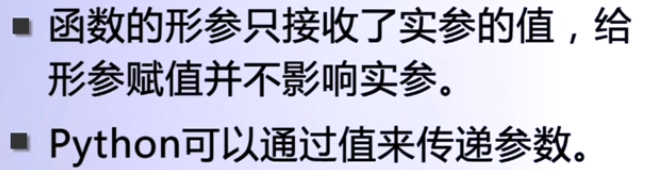


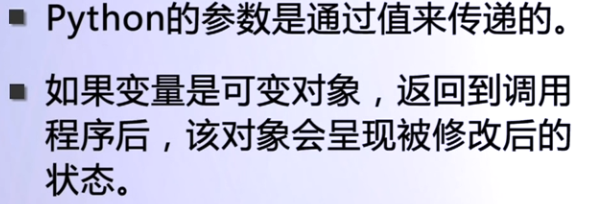
**查看某个函数的函数文档：**

1. **函数名.\_doc\_：不能调用，不知道为什么？**
2. **help (函数名)：**



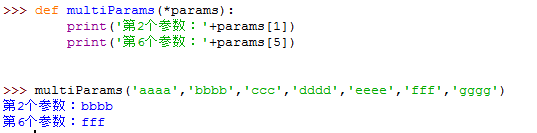
# 值传递参数





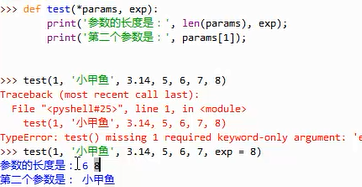
**函数中的实参与形参：与C语言中的定义相同。实参向形参传递值。**

# 收集参数又叫可变参数：参数个数不一定。相当于java中的…功能。

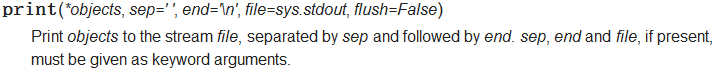


**如果单个变量参数位于后面，则必须使用变量赋值方式传入。**

**在java中，不允许单个变量参数放在…的后面。但是，在Python中，可以通过变量赋值方式解决。**

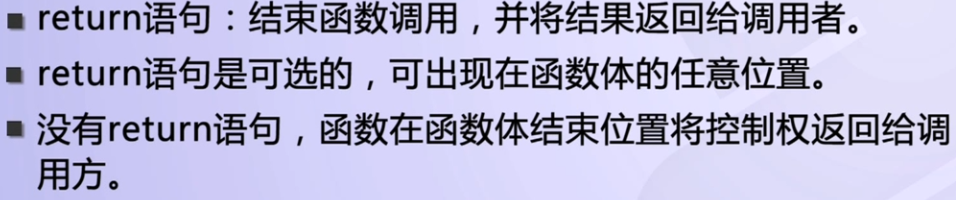


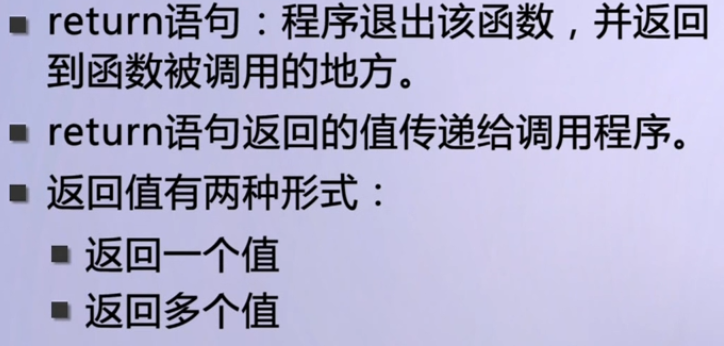
# print函数介绍：

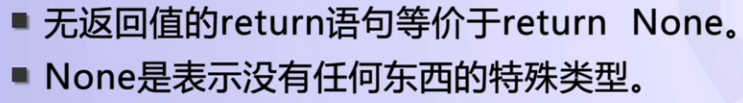


# 函数的return语句及返回值:

## return语句的作用







## 函数的默认返回值：None

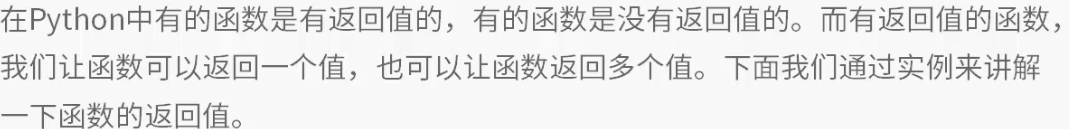
**函数与过程的区别：函数有返回值，而过程没有返回值。**

**Python中没有过程，也就是说，Python中的所有的函数都有具体的返回值，显式用return xxx指出，隐式返回None类型。**

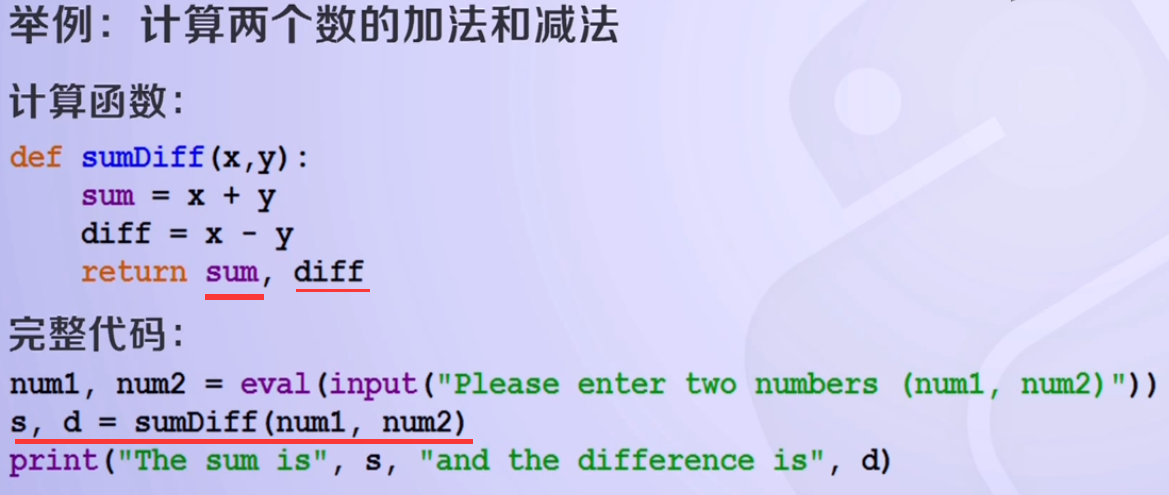


## Python函数返回值类型动态确定

**在其他高级语言中，对函数可以分成int函数、float函数等，是因为它们的返回值是int或float类型的，但是在Python中变量类型是动态确定的，所以Python中的函数不能这么区别。**

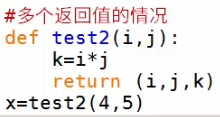


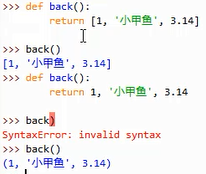
## 返回多个值



**返回多个数据：可以同时返回多个多种类型的数据，如果用中括号括起来，返回的就是一个列表list，如果用小括号或直接不括，默认返回的是一个元组。**

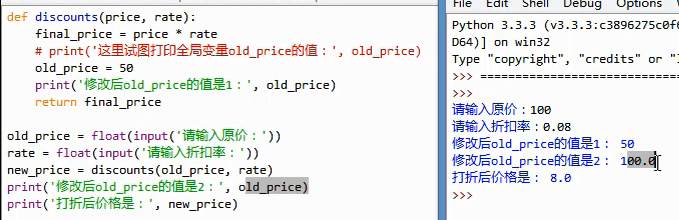
**利用一个变量接收，就是元组；利用多个变量接收，就是对应的元组的值。**



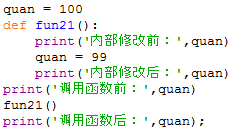


# 全局变量与局部变量：

**全局变量：如果在某个函数中仅仅只是访问全局变量的值，不对其修改，则函数中访问的就是全局变量，但是如果在函数中修改全局变量的值，则就会在函数内部创建一个和全局变量同名的局部变量，函数内部操作的仅仅是局部变量，对全局变量的值没有影响。**



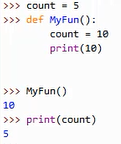
**注意这种情况就会出错：**



**以为函数内部对全局变量进行了修改，但是应该是先赋值，再调用才可以，函数内部不会先把变量quan当成全局的，再当成局部的，这是不可行的。**

**建议：最好不要在函数中去修改全局变量的值，因为这样会在函数内部创建同名字的局部变量，可能与预想的结果不一样，所有建议最好是只是访问全局变量，不要修改。**

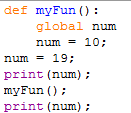
**Python会阻值在函数内部修改全局变量的值，也称为屏蔽。**



**声明全局变量：**

**当然在函数中真的要修改全局变量的值也是可以的，方法是：**

**在函数中对此变量进行声明global即可。**



**结果：**



# 闭包closure：

**比如：一个函数的作用域全部在另外一个函数的内部。**

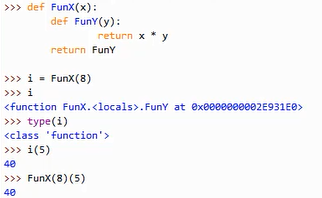
# 内嵌函数：

**在一个函数内部创建一个函数，内部的函数称为内嵌函数。内嵌函数的作用于仅仅只是在外函数内部。**



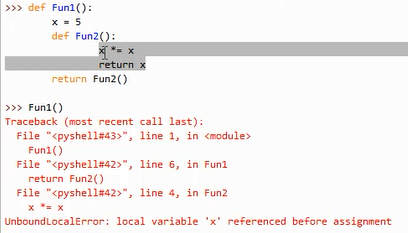


**内嵌函数的调用：**

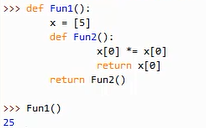


**常见的错误：非全局变量的外部变量：**

**外函数中的局部变量对于内嵌函数来说就是外部变量，只可以访问，不可以修改。**

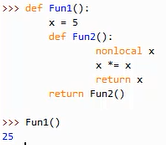


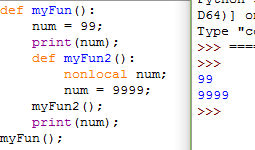
**解决方法1：采用了列表的方法，因为列表不是存放到栈里的。**



**x=[5]是定义的一个列表。**

**解决方法2：自从Python3开始，新功能。利用nonlocal 声明此变量不是局部变量。为什么不用global呢，因为此变量只是外部函数的局部变量，本身也不是全局变量。**

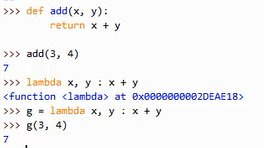




# 关键词：global、nonlocal、lambda。

# 匿名函数：利用lambda关键字进行声明匿名函数。



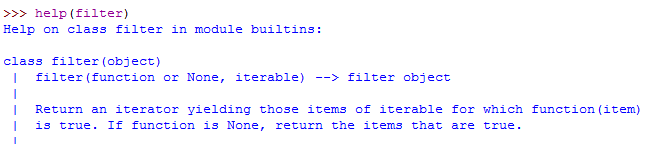


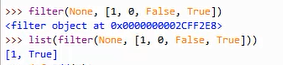
**lambda的好处：**

**Python常用于与Linux配合，写脚本语言。**

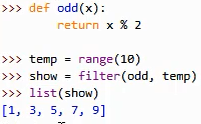


# 介绍filter函数：

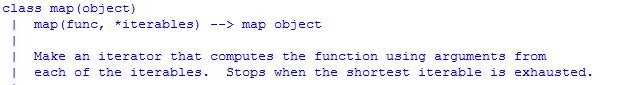




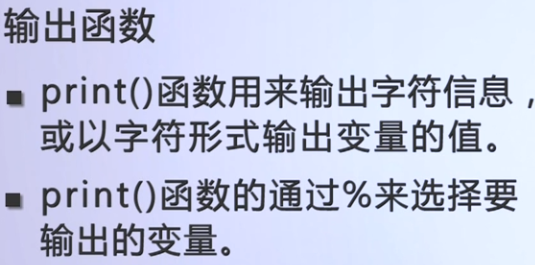
**不指定过滤条件时，用None声明，默认过滤出True的数据。**



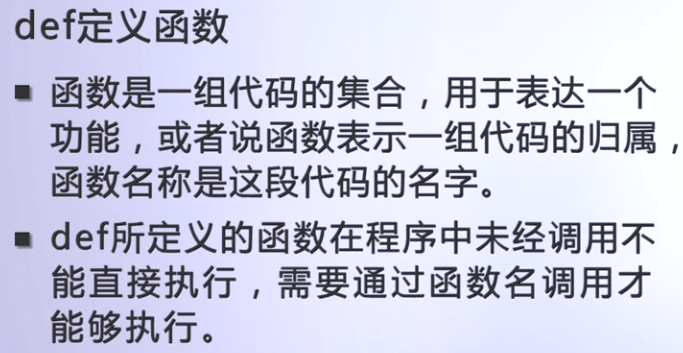
# 介绍map函数：



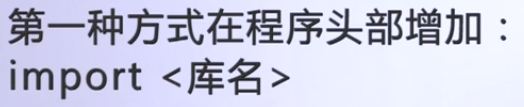




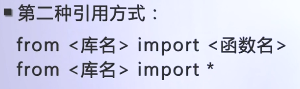
函数：



# Python对库函数的引用方式



库名.函数名



**直接使用函数**。

# Hani问题

1. **Hanio（汉诺）塔问题：**

**注意：递归是倒着来的，最后的需求是把所有的盘子移到C处，至于第一次是把一个盘子移到B处还是C处就靠递归的规律了。发现：num为奇数时，先是A🡺C,num为偶数时，先是A🡺B.**

**/\*\***

**\* 汉诺塔问题分析：**

**\* 64个盘子，从大到小，从下到上排列着，大的在下，小的在上，要从A处移到C处。**

**\* 解决思路： 移动64个到c处，如果能够移动63个到B处，然后把第64个移动到c处，再把63个移动到C处。**

**\* 存在三步：①移动n-1个从源移动到借助地B，**

**\* ② 移动第n个盘子从源到目的地；**

**\* ③ 再移动n-1个从借助地移动到目的地即可。**

**注意在移动n个、n-1个、n-2个……等等的过程中，源始终不变都是A，但是目的地会不断发生变化，因为移动n个时，目的地是C，但是移动n-1个时，目的地就是B，所以移动相邻个数n、n-1个时，目的地和借助地不断交替。**

**\***

**\* 按照递归思想，移动64个从A处到C处，那么只需要完成移动63个从一处移动到另一处，。。。。。。。。**

**\* 从而移动3个从A处移到C处，先移动2个从A处到B处，再把第三个移动到C处，再把2个从Ｂ处移到Ｃ处，**

**\* 移动２个从Ａ处移动到Ｃ处，首先移动一个从Ａ处移到Ｂ处，然后移动第二个从Ａ处到Ｃ处，最后是从Ｂ处移动到Ｃ处。**

**\* 最终究归结到把一个从一个地方移到另外一个地方，即可。**

count = 0  
  
def HaniMove(num,source,to,by):  
 global count  
 if num <= 1:  
 if num <= 0:  
 print('输入错误')  
 return  
 else:  
 count += 1  
 move(source,to)  
 else:  
 HaniMove(num-1,source,by,to)  
 move(source,to)  
 HaniMove(num-1,by,to,source)  
 return  
  
def move(source,to):  
 global count  
 count += 1  
 print(source+"==>"+to)  
  
def main():  
 num = eval(input('请输入盘子数目 ：'))  
 HaniMove(num,'A','C','B')  
 global count  
 print('移动的总数目：'+str(count))  
  
main()

