递归

递归：自己搞自己---在当前函数内部调用自己
注意：避免无限递归
解决办法：设置递归的收敛条件

```
count=0
def fun():
    global count
    if count>=10:
        return None

    print('hehe')
    count+=1
    fun()
fun()
```

1. 死循环:函数内部，属于语句
2. 递归：对象，对象消耗的资源比较大
3. Python递归保护机制：设置Python解释器堆栈的最大深度以限制
   可以防止无限递归导致堆栈溢出和Python崩溃
   可以调节深度：

```
import sys
sys.setrecursionlimit(500)
def fun():
    print('hehe')
    fun()
fun()
```

4. 使用递归的时机
   1. 当解决难题的时候用递归
   2. 当解决大问题的时候，拆分成小问题，而小问题和大问题解决思想一致

5. 递归执行效率低

- 阶乘

```
# 3!=3*2*1
##def fun(n):
##    mult=1
##    for i in range(1,n+1):
##        mult*=i
##    return mult
##print(fun(5))


#6!=6*5!    5!=5*4! 4!=4*3!   3!=3*2!  2!=2*1

#递归：
def fun(n):
    if n==1:
        return 1
```

```
    else:
        return n*fun(n-1)
print(fun(6))
```

- 斐波那契数列

```
#斐波那契数列

##def r(n):
##    f1=1
##    f2=1
##    if n==1 or n==2:
##        return 1
##    else:
##        for  i in range(3,n+1):
##            f1,f2=f2,f1+f2
##        return f2
##print(r(12))


# 递归求解：
def fib(n):
    if n<3:
        return 1
    else:
        return fib(n-2)+fib(n-1)

print(fib(12))
```

- 汉诺塔

```
# 汉诺塔

def fun(a,b,c,n):
    if n==1:
        print('%s-->%s'%(a,b))
    else:
        fun(a,c,b,n-1)
        print('%s-->%s'%(a,b))
        fun(c,b,a,n-1)
        return None
fun('A','B','C',10)
```