

递归 — 节点的定义  
 \ 重复性 (自相似性)

递归 Recursion

递归 — 循环  
 本质 →

通过函数体来进行的循环

通过循环体调用自己来进行所谓的循环

## 递归 Recursion

1. 从前有个山
2. 山里有个庙
3. 庙里有个和尚讲故事
4. 返回1

计算 n!

$n! = 1 * 2 * 3 * \dots * n$

```
def Factorial(n):
    if n <= 1:
        return 1
    return n * Factorial(n - 1)
```

```
factorial(6)
6 * factorial(5)
6 * (5 * factorial(4))
6 * (5 * (4 * factorial(3)))
6 * (5 * (4 * (3 * factorial(2))))
6 * (5 * (4 * (3 * (2 * factorial(1)))))
6 * (5 * (4 * (3 * (2 * 1))))
6 * (5 * (4 * (3 * 2)))
6 * (5 * (4 * 6))
6 * (5 * 24)
6 * 120
720
```

## Java 代码模版

```
public void recur(int level, int param) {
    // terminator
    if (level > MAX_LEVEL) {
        // process result
        return;
    }
    // process current logic
    process(level, param);
    // drill down
    recur(level: level + 1, newParam);
    // restore current status
}
```

Recursion Terminator 递归终结条件

写递归的函数开始的时，一定要先把函数  
 i.e. 递归终止条件写上 要不然：死循环

Process logic in current level  
 处理当前层逻辑

Drill Down 下探到下一层

parameter 参数标记当前是哪一层，level + 1  
 同时把相应的参数  
 P1, P2, P3 ... 放下去

Restore the current level  
 status if needed 清理当前层

## 思维要点

1. 不要人肉进行递归（最大误区）

2. 找到最近最简方法，将其拆解成可重复解决的问题（重复子问题）  
i.e. 找最近重复子问题

3. 数学归纳法思维  
最开始 = 最简单的条件 成立 e.g.  $n=1, n=2$   
&  $n$  成立的时候  $\rightarrow n+1$  也成立

程序指令又包括

if...else  
for loop  
while loop  
&  
递归调用

$\therefore$  所有代码只有这三部分

比较复杂的程序（逻辑较多）

却可以用五行/十行语句解决

why  $\rightarrow \because$  复杂  $\therefore$  有很多可重复性  
逻辑