# 什么叫递归

想要理解递归，首先要理解递归。如果能理解这句话，也就算理解了递归概念了。

递归算法是一种可以用简单逻辑代码解决复杂问题的程序设计技术，简单来说就是在程序中调用相同的程序。

当方法调用自身时，在堆栈上为新的局部变量和参数分配内存，并且使用这些新的变量从头开始执行方法的代码。当每次递归调用返回时，将旧的局部变量和参数从堆栈中移除，并将执行控制恢复到方法内部的调用点。所以，另一方面，递归方法也被称为”望远镜式”的方法，可以自由伸缩。从调用栈中能看出，每次调用一次自身函数，便压一次栈，函数返回再弹栈。介于每次调用和执行都是有一定的系统资源消耗的，而且可能资源消耗量是指数增长的，有些程序管理器都会对最大递归层数做出限制，以防止耗尽系统资源导致死机。真是因此，许多例程的递归版本的执行速度都要比等价的迭代版本慢上一些，因为增加了额外的函数调用负担。

递归算法多在每次操作相似的情况下使用，如果要对所有相似的元素都进行相同的操作，很有可能可以使用递归来解决。

递归算法的步骤基本都遵循几步骤原则:

1. 寻找到一个入口，传递参数，
2. 检查是否满足继续的条件，不满足则跳到操作5
3. 执行计算，将当前问题变成一个或多个更小的问题
4. 对每一个子问题，进行操作2
5. 合并结果表达式
6. 返回结果

实现递归算法的一个难点就是如何找到或者计算出这么一个递归函数。幸运的是，有很多常见问题已经有现成的解决方法了，作为人类知识文明在互联网上进行传播。

这是一个对快速排序的简单Python实现，

# -\*- coding: utf-8 -\*-  
**def quick\_sort**(*data*):  
 *"""Quick sort  
 递归解决列表排序   
  
 Args:  
 data (List[int]): 需要排序的列表  
  
 Returns:  
 List[int]: 有序列表  
 """* **if** len(*data*) <= 1: **return** *data* pivot = *data*[0] # 选取第一个元素为参照  
  
 left = quick\_sort([x **for** x **in** *data*[1:] **if** x <= pivot])  
 right = quick\_sort([x **for** x **in** *data*[1:] **if** x > pivot])  
 **return** right + [pivot] + left

# Test Part  
print(quick\_sort([1, 2, 3, 4, 5])) # [5, 4, 3, 2, 1]

可以看出，根据递归函数的定义，这里的边界条件是len(data) <= 1，pivot是作为对比的参照数。函数中，left和right都分别调用了函数自身，传递了一个列表作为参数，其中使用列表推导式来生成一个 筛选了data中元素的列表。最终返回一个由right(大数字) pivot(参照数) left(小数字)组成的列表。即便Python默认有1000的最大调用层数限制，但由于每一层调用都把原数据规模缩小了一半，可以达到的复杂度。

可以对函数进行稍微地修改，利用Python动态特性，使其不仅支持数字，还能对实现了比较方法的对象或者使用一个key参数进行对比。但是如果想用迭代法来实现快速排序算法，那将是一个相当困难的挑战。这也从另一方面展示了递归策略的一大优势。

很明显的一个递归算法的优点就是，减少了重复的代码量，使结构变得清晰，每一次都在缩小当前问题的复杂度。当然也带来了另一个问题，占用的资源比较大，每一次保存现场都会占用一部分内存空间，层层累加也是能达到不小的数量级的。递归算法也不好优化，尽量将不必要的操作移出函数体。

第一次接触递归算法的时候，可能会觉得无法理解，但随着程序结果越来越复杂，递归程序设计能够用逻辑一致的方式更好地维护和组织代码。另外在人工智能领域，递归算法也能够简单清晰地完成一般算法无法完成的复杂逻辑。