004.001-递归小叙

上节课讲完之后呢，很多同学在理解“递归思想”这个地方遇到了困难、卡住了。而且，有些同学之前学过一点编程，知道有“递归函数”这样一个术语，但理解的又不够透彻，于是过去的记忆又跳出来干扰现在的学习。所以，我感觉有必要在“递归思想”这里再点拔大家一下。

# 递归思想及其应用

首先，就像我在《学习指南》里说过的那样——初学者要关注的是当下的学习内容。触类旁通这种高级技能是你在至少很扎实、很透彻地学会一门编程语言之后才会具备的。不然的话，那些之前理解的不深入甚至不正确的内容就会不断地跳出来干扰你当下的学习、起到一个“触类旁歪”的反效果。如果你能仔细琢磨当下这第4讲讲的是什么，你就不会问“为什么SELECT语句里没有递归函数”这样的问题了。更不会问出“SELECT语句里没有循环为什么是递归”这种流星听了都能往回跑的问题。

其次，就当下我们讨论的“递归式语法定义”来说，请大家观察一下咱们使用UNION的那个例子，然后再来看这样一个问答：

| 小和尚：师父，什么是SELECT语句呀？ 老和尚：咳咳，SELECT语句变化多端，就像山下的女孩子，你可不要去招惹她们。 小和尚：师父，跟您说正事儿呢。 老和尚：哦，正事儿啊。**一个SELECT语句可以是一个SELECT语句UNION上另一个SELECT语句**。 小和尚：您这说了等于没说啊！那UNION**前面**和**后面**的两个SELECT语句又是什么呢？ 老和尚：可以是一个SELECT语句UNION上另一个SELECT语句...... 小和尚：呃……那这这样循环下去岂不是没有尽头了？ 老和尚：这不叫"循环"，俗语中把这种形式称为"自己**嵌套**自己"，术语中这个叫"**递归**"，所以话不能乱讲，不然让人笑话。 小和尚：没事儿，反正他们哪次也没笑话我————都是笑话师父、说师父没把我教好。 老和尚：而且，SELECT语句也有**不嵌套其它SELECT语句的简单形式**，比如"SELECT 100\*100"或者"SELECT \* FROM Person.Person",这时候，递归就**终止**了。 |
| --- |

所以，上面的对话中对SELECT语句的递归式语法定义做了进一步的解释。其中，有几个关键的词汇需要大家细细体会：自己嵌套自己、递归、终止。

如果把上面对话中蕴含的表达方式进一步抽象出来，我们就能得到“**递归思想**”的潦草表述（详细的表述请大家阅读数学文献）：

| 有概念A，B，C。 A要么是B，要么是A和C的联合体。  此时A的定义是**递归的**。  注意：因为B是A的子集，所以“B是A”在逻辑上成立。 |
| --- |

现在让我们把递归思想向外推广一下：

* 套娃是什么？
  + 非递归定义：N个单层套娃从小到大一个套一个。
  + 递归定义：套娃要么是一个单层套娃，要么是套娃里套着套娃。
* 扑克牌是什么？
  + 非递归定义：52张花色不同的牌。
  + 递归定义：要么是一张牌，要么是两组花色不同的扑克牌。牌的总数不超过52。
* 面团是什么？
  + 非递归定义：面粉加水。
  + 递归定义：要么是一粒面粉周围围绕着几个水分子，要么是两个面团揉在一起。

# 递归函数

最后，再让我们来看看“递归函数”是什么。注意：

* 这一小节写给有编程基础的学习者阅读。
* 没有编程基础或的学习者可略过。
* 认为自己有编程基础却读不懂的学习者，请适当修正自我评估🐶

简单来说，**递归函数就是递归思想在编程语言中的直接表达**。比如，经典的“[河内塔问题](https://zh.wikipedia.org/zh-hant/%E6%B1%89%E8%AF%BA%E5%A1%94)”（也叫“[汉诺塔问题](https://zh.wikipedia.org/wiki/%E6%B1%89%E8%AF%BA%E5%A1%94)”），就可以用递归思想来思考——当然也可以用非递归的思想来思考。但我估计这个问题对于大家来说仍然太有挑战，于是这里给大家换一个更简单的问题：如何计算1到100的和。其实两种计算思想：

* 非递归思想：从1开始，加2，加3……加100。
* 递归思想：1到100的和特价于1加上2到100的和，2到100的和特价于2加上3到100的和……100到100的和就是自己——100。

可能很多学习者今天才发现还有递归思想计算1到100之和的方法——这很正常。我们的基础数学中之所以不教这种思想，一个是因为大多数少年儿童的思考能力达不到要求；另一个是因为就算教了，用纸和笔来展开递归式运算也会非常不方便——在计算机上编程就没有这个问题。

好吧，让我们用Python来实现非递归和递归两种运算：

| # 非递归思想，使用循环语句，不断累加 **def** **sum**(start, end):  accumulator = 0  **for** n **in** range(start, end + 1):  accumulator += n  **return** accumulator   result = sum(1, 100) print("Non-Recursive:", result)  # 请Python"忘掉"上面那个sum函数 **del** sum  # 递归思想，使用递归函数的自然实现 **def** **sum**(start, end):  **if** start < end:  **return** start + sum(start+1, end)  **else**:  **return** end # 或return start   result = sum(1, 100) print("Recursive:", result) |
| --- |

注意：

* 第二个sum函数是递归函数
* 它之所以是递归函数，是因为它包含有“**return** start + sum(start+1, end)”这句代码——这句代码中包含了对sum函数自身的调用。这种“自己调用自己”的函数就是递归函数。因为sum直接调用了自己，所以它是一个**直接递归函数**。未来大家有可能会遇到“函数A的代码里调用了函数B，函数B的代码里又调用了函数A”这种**间接递归函数**。

总结一句，今天详辨的“递归函数”与“递归思想”不是一个层面上的东西，它们之间的关系是——**递归函数是递归思想在编程语言中用“函数”这种语法元素的表达**。也可以换个角度想，那就是递**归函数是编程语言对递归思想的内建支持**——有了这个支持，编程语言才能成为更好的数学工具。

如果你对递归思想和递归函数有更多兴趣、想做更深入的了解，请阅读《[算法之禅](https://item.jd.com/13017396.html)》这本书——京东有售，人在美国的学习者可以直接联系我，我手里还有几本。

# 挑战赛

如果你能以本篇文字为内容制作出一集与我的课程质量相当的视频，我将会把你的视频纳入这个课程系列，并向你赠送一本《算法之禅》（或$25 Amazon礼品卡）。

评比要求：

* 内容不能出现错误，逻辑必须自洽。
* 语言、图像表达流利自然。
* 优胜者将从参赛者提交的作品中由大家投票选出（前三名）。