算法设计

Python Reference

Algorithms & decomposition

算法 (algorithm) 是一份能解决问题的清晰步骤清单。分解 (decomposition) 的意思是把一个大问题拆成更小的、能逐个解决的部分。

```
# Algorithm: find the largest number in a list
def largest(nums):
    best = nums[0]
    for n in nums:
        if n > best:
            best = n
    return best

print(largest([3, 9, 2, 7]))   # 9
```

- 抽象 (abstraction) 的意思是忽略细节: 你直接用 `largest(...)`, 不必再去读它怎么实现。

Pseudocode & flowcharts

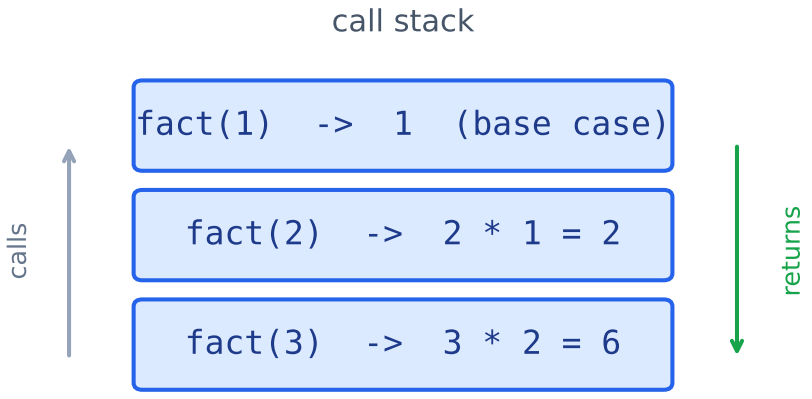
伪代码 (pseudocode) 是为算法写的、朴素而有结构的英文, 写在真正的代码之前。它不会被运行。

```
SET best TO first number
FOR Each number n
    IF n > best
      THEN
        SET best TO n
OUTPUT best
```

流程图 (flowchart) 把同样的计划画出来: 每个步骤一个方框, 每个判断 (decision) 一个菱形, 用箭头表示顺序。

Recursion & the call stack

递归 (recursion) 是指一个函数调用它自己。它需要一个基准情形 (base case)(一个简单的输入, 立刻返回), 以及一个递归情形 (recursive case)(它在更小的输入上调用自己)。



factorial(3) 的调用栈: 每个调用等待下层返回, 再按相反顺序返回

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

print(fact(5))      # 120
```

- 每个暂停的调用都停在调用栈 (call stack) 上, 直到它上面的调用返回。

常见错误

- 递归 (recursion) 必须有基准情形 (base case), 否则会无限自调用, 撑爆调用栈。
- 伪代码 (pseudocode) 用来规划 — 不必能运行, 但每一步都要清楚无歧义。
- 先把大问题拆成一个个有名字的小步骤, 再动手写代码。