Python算法之旅（第4期）

上期回顾：

上一期课后思考的题目是“丹丹买了10瓶啤酒，3个空瓶可以换1瓶啤酒，请问丹丹最多可以喝到多少瓶啤酒？”

函数：beer\_number(n, m)

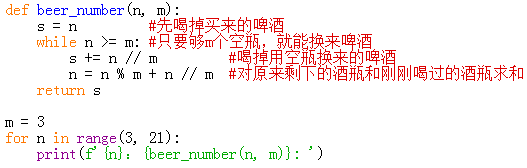
参数说明：n，整数，丹丹最初购买的啤酒数量。

m，整数，m个空瓶可以换1瓶啤酒。

返回值：丹丹最多能喝到的啤酒数量。

示例：当n=10，m=3时，丹丹可以喝到10+3+1=14瓶酒。

答案如下：



有些VB算法题取材于实际生活案例，虽然背景信息复杂了点，但是它源于生活，具有实际意义，也能较好的引发学生的解题兴趣，训练学生的建模能力和计算思维。

题目：赢取最多奖金

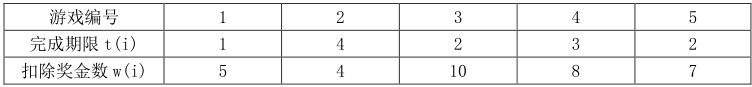
难度：2星 有趣：3星 有用：3星

分类：排序，贪心

描述：小炫报名参加“智力大冲浪”节目。比赛规则如下：

比赛开始时参赛者将预先得到 M 元奖金。

比赛时间分为 N 个时段(N≤100)，有 N 个小游戏，每个时段可完成 1 个游戏，第 i 个小游戏必须在规定期限 t(i)时段前完成(1≤t(i)≤N)，否则要从奖金 M 元中扣去奖金 w(i)，w(i)为自然数，不同的游戏扣去的奖金是不一样的。每个游戏必须从整时段开始。注意：比赛绝对不会让参赛者赔钱!

例如：当 N=5,M=100 时

在这种情况下，小炫依次完成编号为 5、3、4、2、1 的小游戏，获得了最高奖金 95 元。他的算法思想：让扣款高的游戏尽量准时完成。扣除的奖金越少，则最后赢取的奖金越多。

（1）按扣款数值从大到小排序，在样例中，排序后游戏编号依次为 3、4、5、1、2；

（2）对于游戏 i，在时间段 1 到 t(i)完成，效果都是一样的，所以尽量放置的时间段 t(i)完成，若该时段已经被占用，则依次考查时间段 t(i)-1, t(i)-2,...,1。在样例中，

1. 考虑游戏3，放置在时间段2完成（注：t(3)=2）；

2. 考虑游戏4，放置在时间段3完成（注：t(4)=3）；

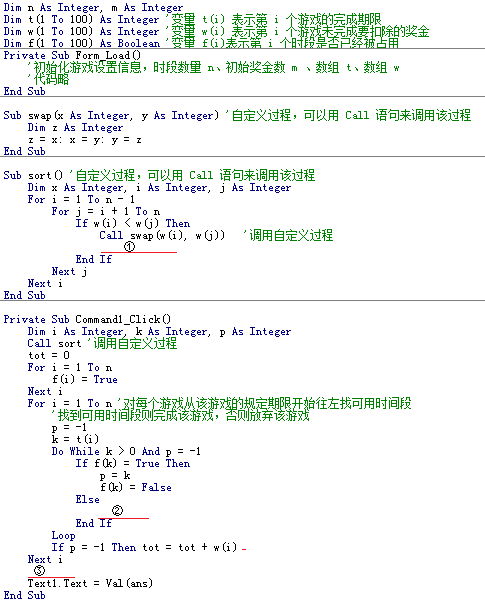
3. 考虑游戏5，时间段2已被游戏3占用，放置在时间段1完成（注：t(5)-1=1）；

4. 考虑游戏1，时间段1已被游戏5占用，不能按时完成，记录扣款；

5. 考虑游戏2，放置在时间段4完成（注：t(2)=4）

小炫按如上算法编写了一个VB 程序，计算“智力大冲浪”游戏中玩家最多可赢取的奖金，将结果输出到文本框 Text1 中。

VB 程序代码如下所示，请在划线处填入合适代码。



答案：（1）call swap(t(i),t(j)) (2分)

（2）k = k - 1 (2分)

（3）ans = m - tot (2分)

算法分析：因为不同的游戏不能准时完成时具有不同的扣款数值，而且是最优解问题，所以很容易就想到了贪心法。贪心的主要思想是要让扣款数值大的尽量准时完成。这样我们就先把这些任务按照扣款的数值进行排序，把大的排在前面，先进行放置。

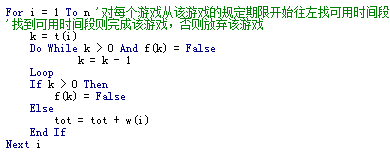
假如罚款最多的一个任务的完成期限是k，我们应该把它安排在哪个时段完成呢？应该放在第k个时段，因为放在1～k任意一个位置，效果都是一样的。

若第k个时段已经有任务占据，则向前查询是否有空的时间段，若完成期限之前的时间段全部被占据，则放弃该任务。这样能获得最优结果。

题目提供的示例，充分说明了本算法思想。

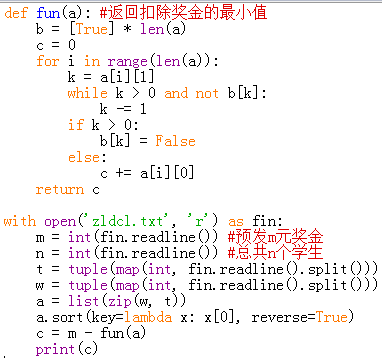
题目提供的代码，基本体现了上述算法思想，它设置了两个变量k和p，其中k用来遍历各个时段，p用来记录可用时段，一旦找到可用时段即退出循环。若循环结束后p=-1，说明该任务无法完成。

这样做固然可行，但不够简明。事实上变量p的设置是多余的，我们可以直接使用变量k来记录可用时段。修改后的核心代码如下：



贪心算法的思想是很简明的，代码一般也很简短，但是需要做好排序等准备工作。

Python语言拥有丰富的内置函数和强大的高阶函数，可以快速方便地解决排序问题，帮助我们减少思维干扰，直指问题的核心。对应Python代码如下：



课后思考：

虽然贪心算法不是对所有问题都能得到整体最优解，但只要能找到具备无后效性的贪心策略，就可以得到最优解，而且效率非常高。我这里再提供一道能用贪心算法解决的经典问题，供大家课后思考。

题目描述：学校的小礼堂每天都会有许多活动，有时候这些活动的计划时间会发生冲突，

只能选择举办其中的部分活动。小刘的工作就是安排学校小礼堂的活动，每个时间最多安排一个活动。

现在小刘有一些活动计划的时间表，他想尽可能的安排更多的活动，请问他该怎么办。

注意：如果上一个活动在t时间结束，下一个活动最早应该在t+1时间开始。

函数名：schedule(n, mt)

参数表：n，正整数，表示总共n个活动

mt，列表，mt[i][0]和mt[i][1]分别表示活动i的起始和结束时间，确保mt[i][0]<mt[i][1]

返回值：返回最多能够安排的活动数量。

示例1：n=2,mt=[(1,10),(10,11)]时，返回1

示例2：n=3,mt=[(1,10),(10,11),(11,20)]时，返回2

另外，如果你有更 Pythonic（优雅的、地道的、整洁的）代码，或者与本文不同的算法思路和代码实现，请你一定留言或联系我，让我们一起讨论，共同进步。