**计概cheetsheet**

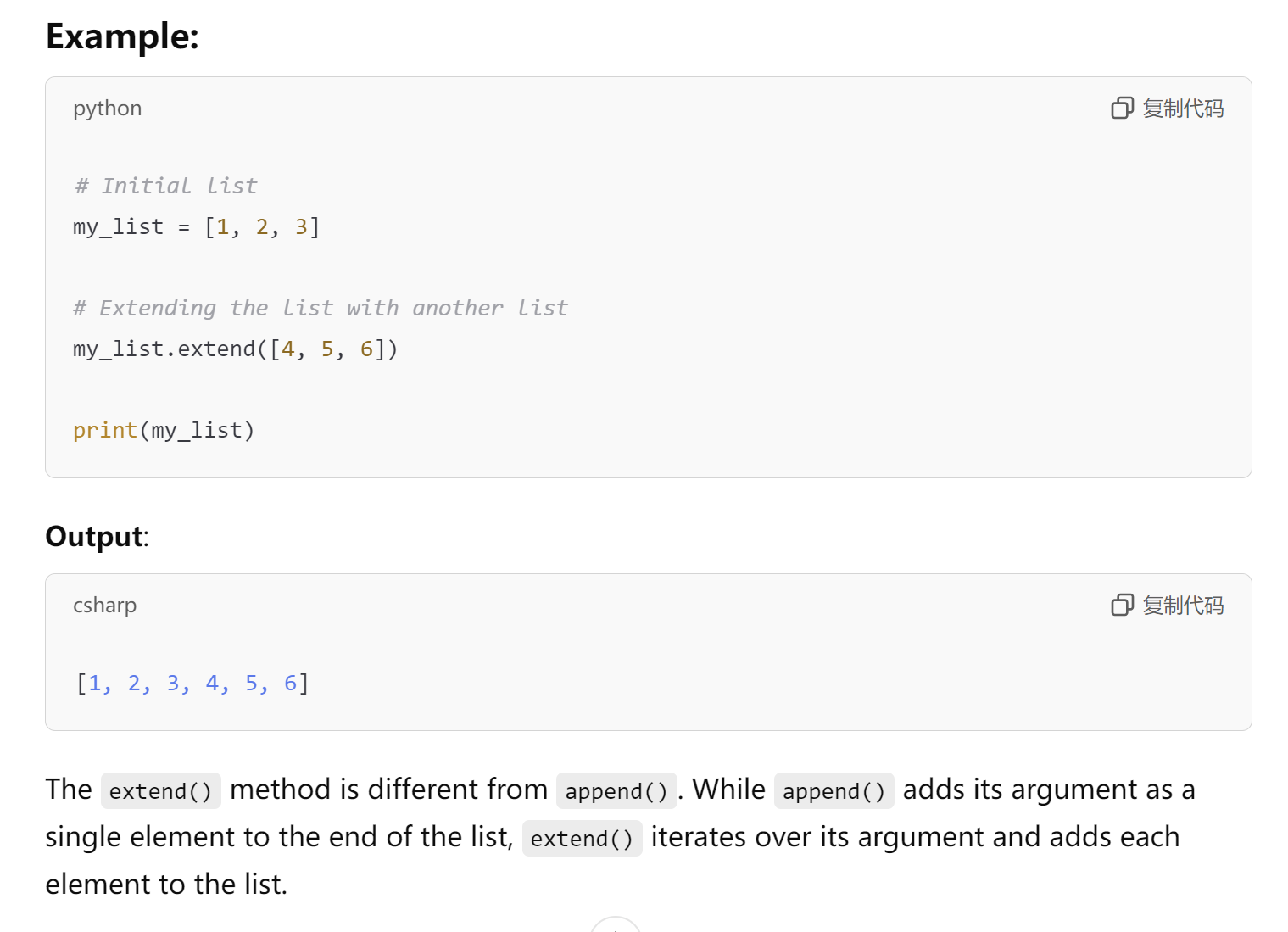
**勿忘：0.一定要记得考虑一些边界情况！！**比如只有1个路灯（那样的话距离序列就不存在），或者只有0个元素；以及一定要验证**第一个格子是否是陷阱**~（重点检查：首步、尾步、0个元素）

Vip：debug许久的时候，检查**是否拼写有误or字母写错**~

1. 如果TLE，可以**把list改成set**，这样判断一个数在不在一个东西内可以大幅减少时间复杂度！也可以**用dict记录数据**~（可以创建字典来**记录每个数和他对应的index**！！）

（p.s. 1.set只有add；2.set内只能有**int，float，str，tuple**，不能有set，list，dict这些不可哈希的对象！）

2.extend可以一次添加多个元素（但要把他们容纳到一个列表里）

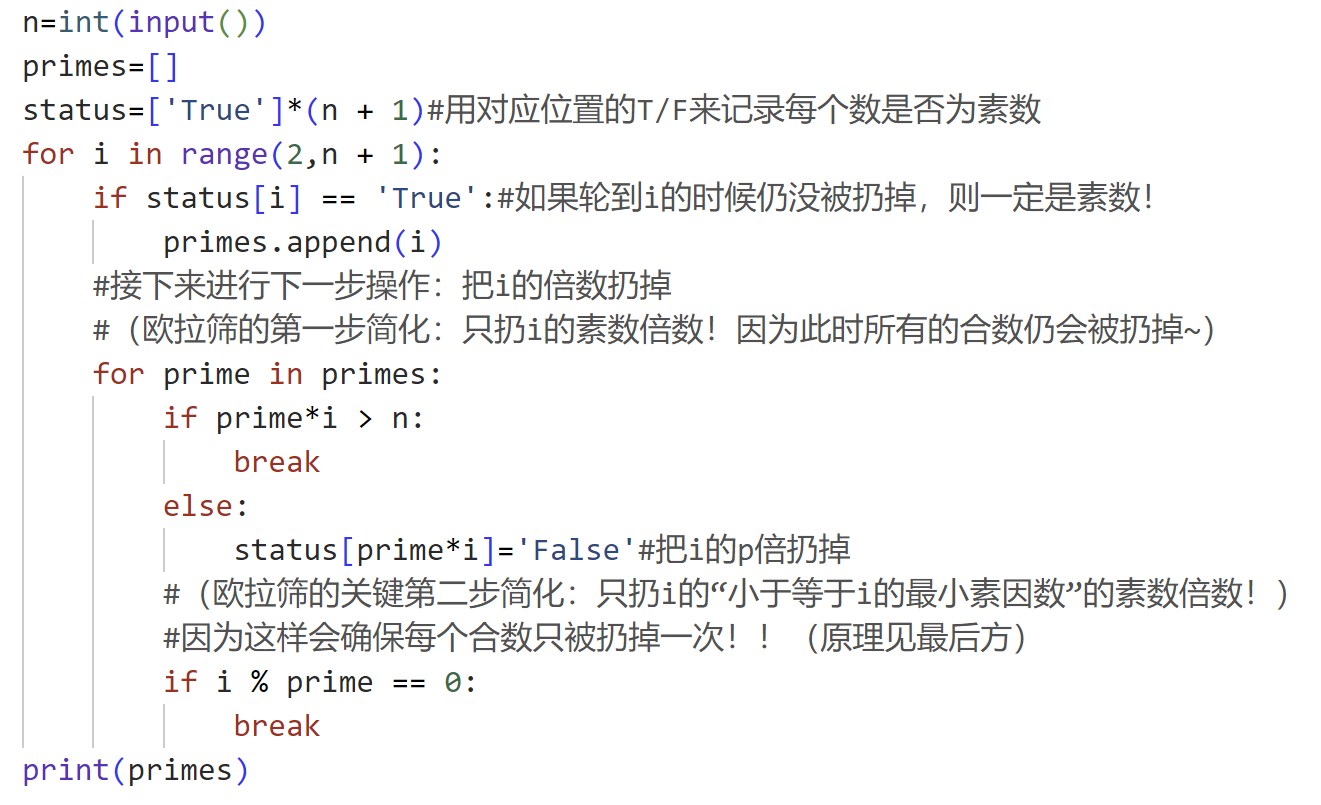
****

3.系统判断两个列表是否相同的时候会考虑元素顺序！（元组同理）

1. 用“**[[-1]\*m for \_ in range(n)]”**初始化矩阵！（不要复制空列表！）

5.ord将一个字符转换为他的Unicode 码点（整数），chr反之

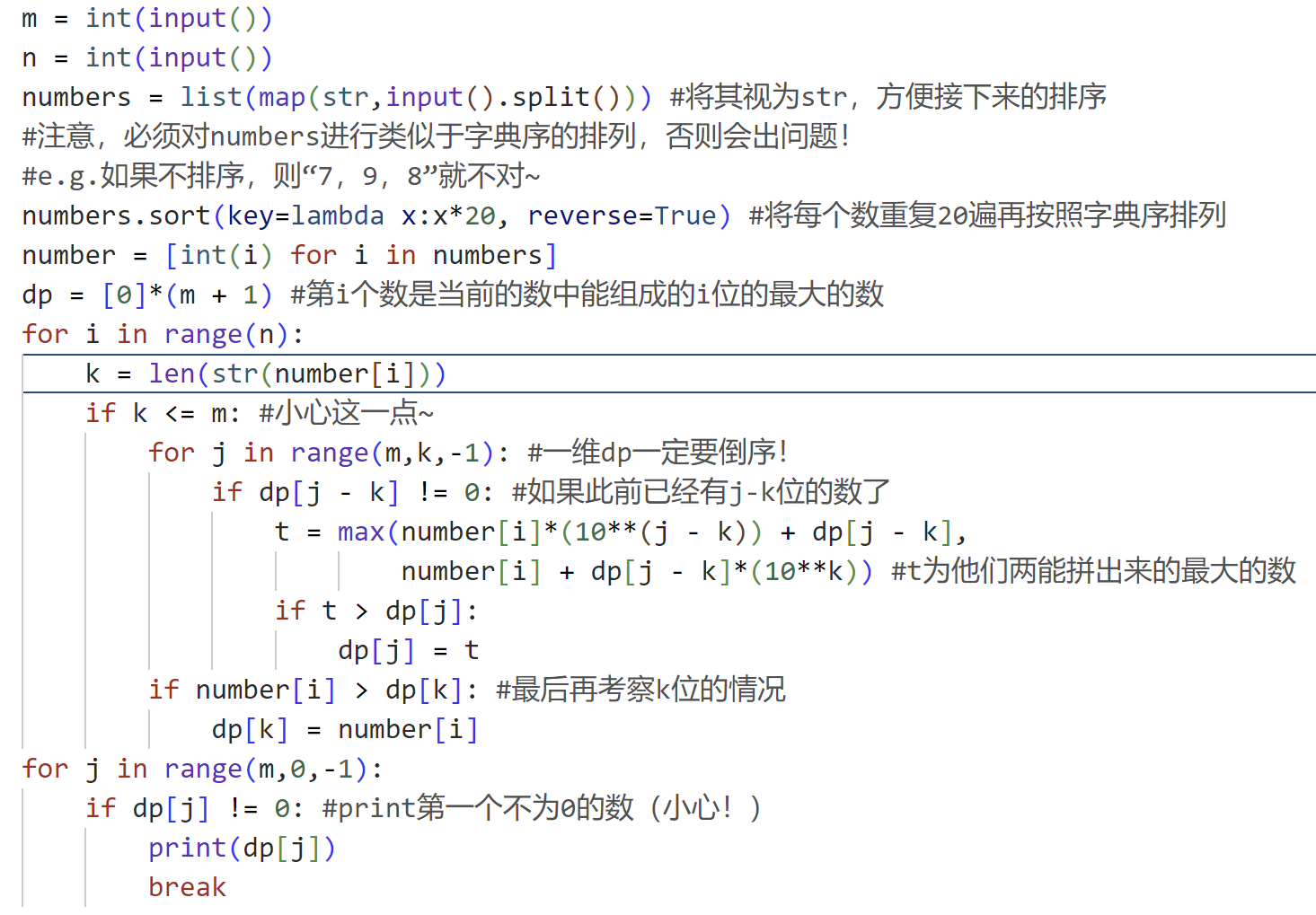
欧拉筛：



**（只扔掉i的“小于等于i的最小素因数”的《素数倍数》！！）**

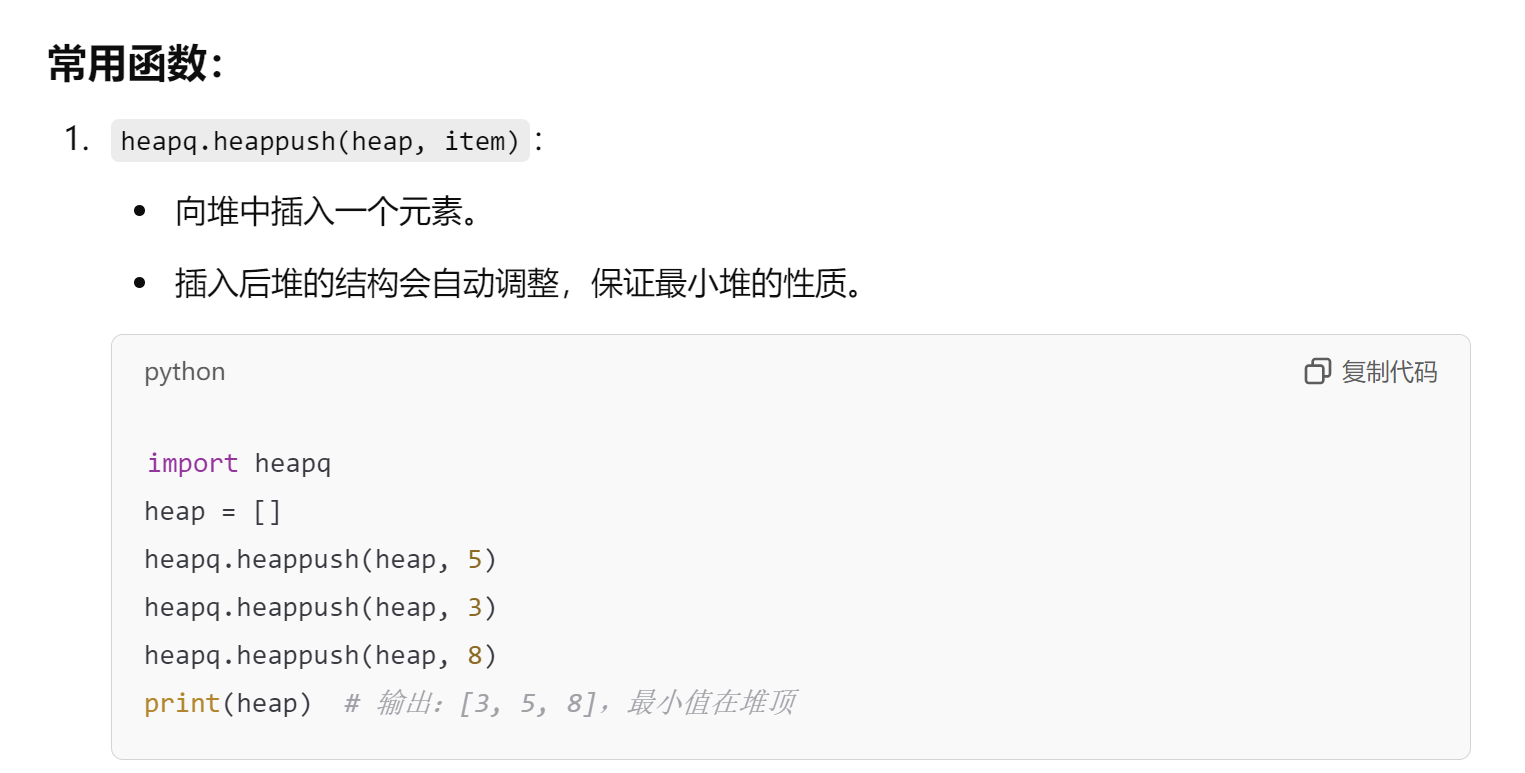
最大整数**：一维dp要倒序！！**

**x：x\*20**（对字符串而言，这是把他复制20次的意思）

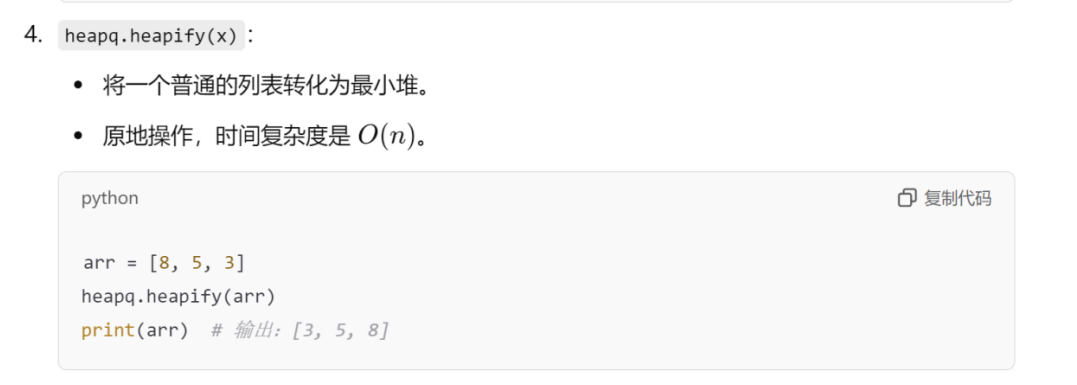


**import heapq**

**最小值永远在第一个元素！（函数：heappush/heappop/heapify）**

****

****

****

**from itertools import permutations**



（perms生成的是元组）



**格式：对象，长度（长度可省略，但对象必须以变量名的形式出现！）**



**（会自动按字典序生成）**

**注意：不能print（perms（x））！**

****

**因为perms只会返回一个迭代器！！**

**import bisect**

****

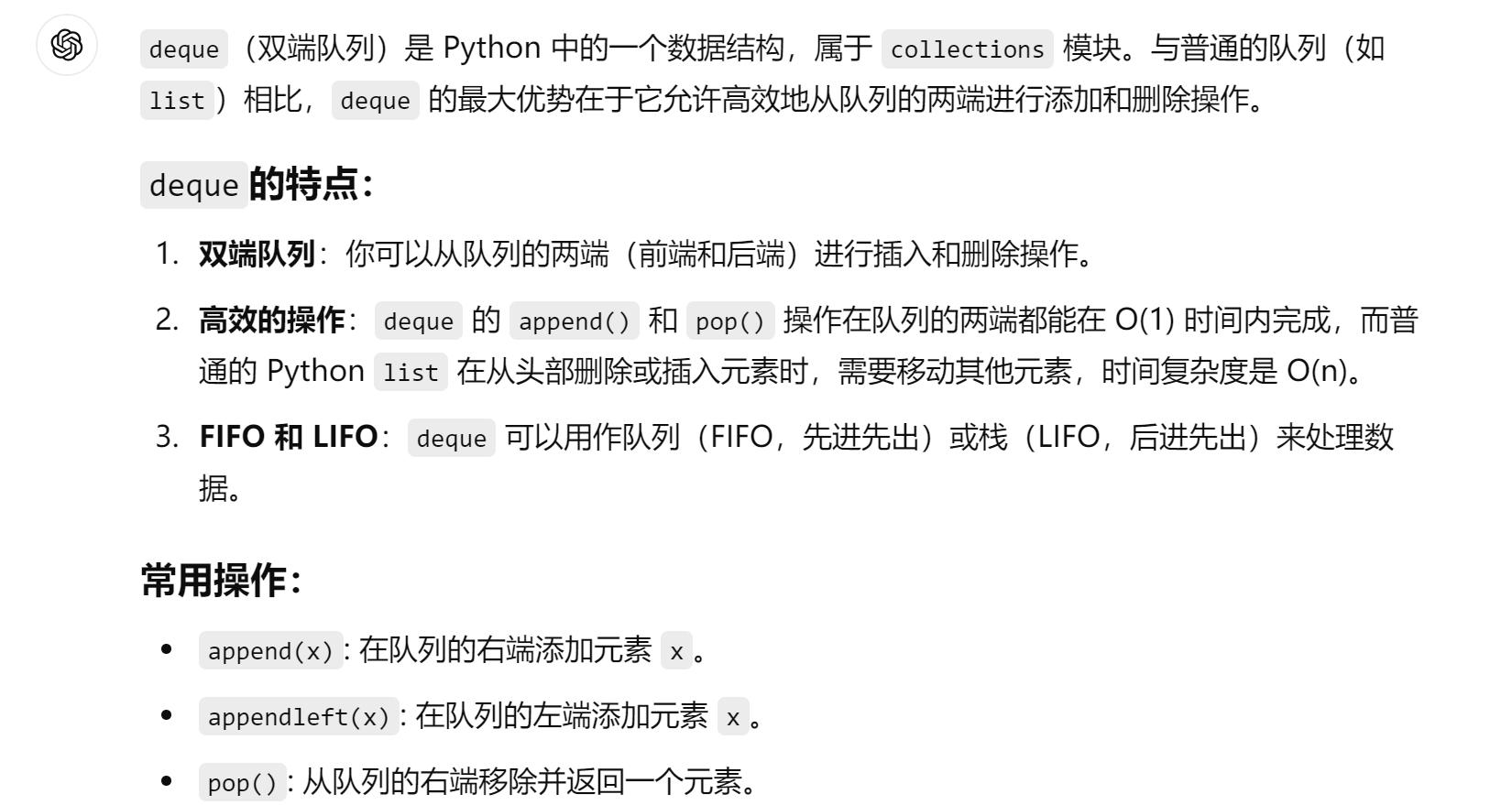
****

**格式：bisect.bisect\_left（列表，元素）**

**from collections import deque**

**函数：pop、popleft、append、appendleft！**

（还有extend、extendleft、rotate（往前平移））





**import sys**

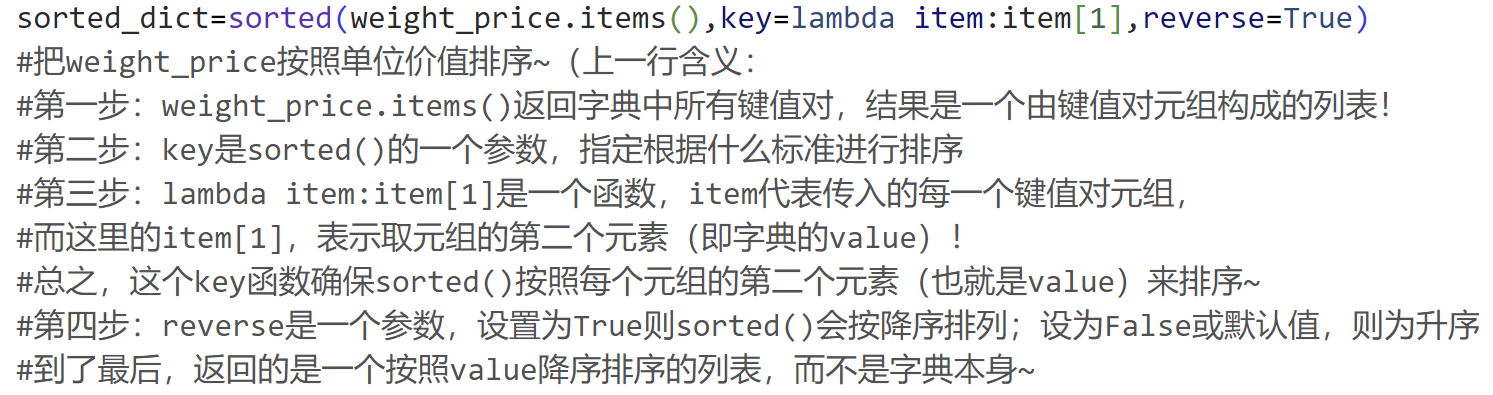
**sys.setrecursionlimit(10\*\*5)** #设置递归深度限制（默认1000）

**lines=list(sys.stdin.read().split())** #一次性读取所有数据，并按空格划分开，合并为一个列表（若要本地测试，需要ctrl+D）

**import copy**

**copy.deepcopy（X）**#浅拷贝（copy.copy(X)）只复制了引用，而深拷贝复制了整个对象及**其嵌套**的所有部分

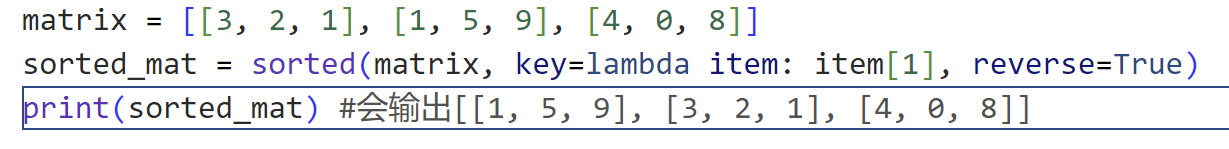
**琐碎**

****

**#对字典中的value进行排序**

（对第k个位置的元素排序，写item[k-1]即可）

这个方法也适用于对矩阵（列表套列表）里的某个固定位置的元素进行排序：

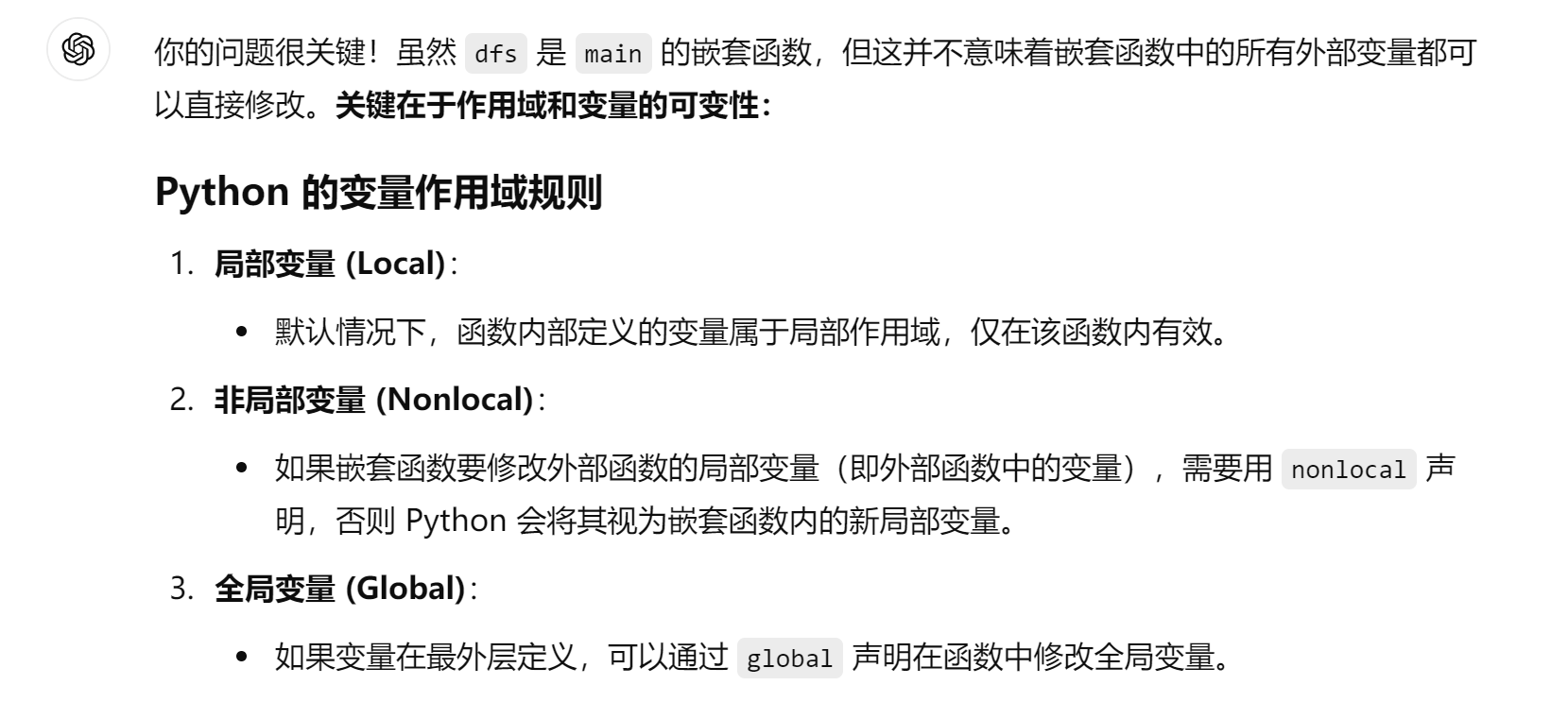


例如，这里就是对矩阵每行的第二个数倒序排列~

大函数里的小函数属于大函数的嵌套函数，可以直接访问大函数内的任何变量！**但是不能修改！！**

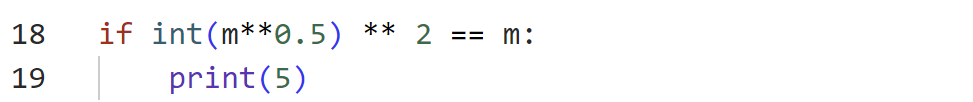
**Nonlocal：解禁不可修改但可引用的局部变量**

**Global：引用并修改全局变量**（有global便可对全局变量修改，否则只会视为函数里的局部变量，对外界没有任何影响）

****

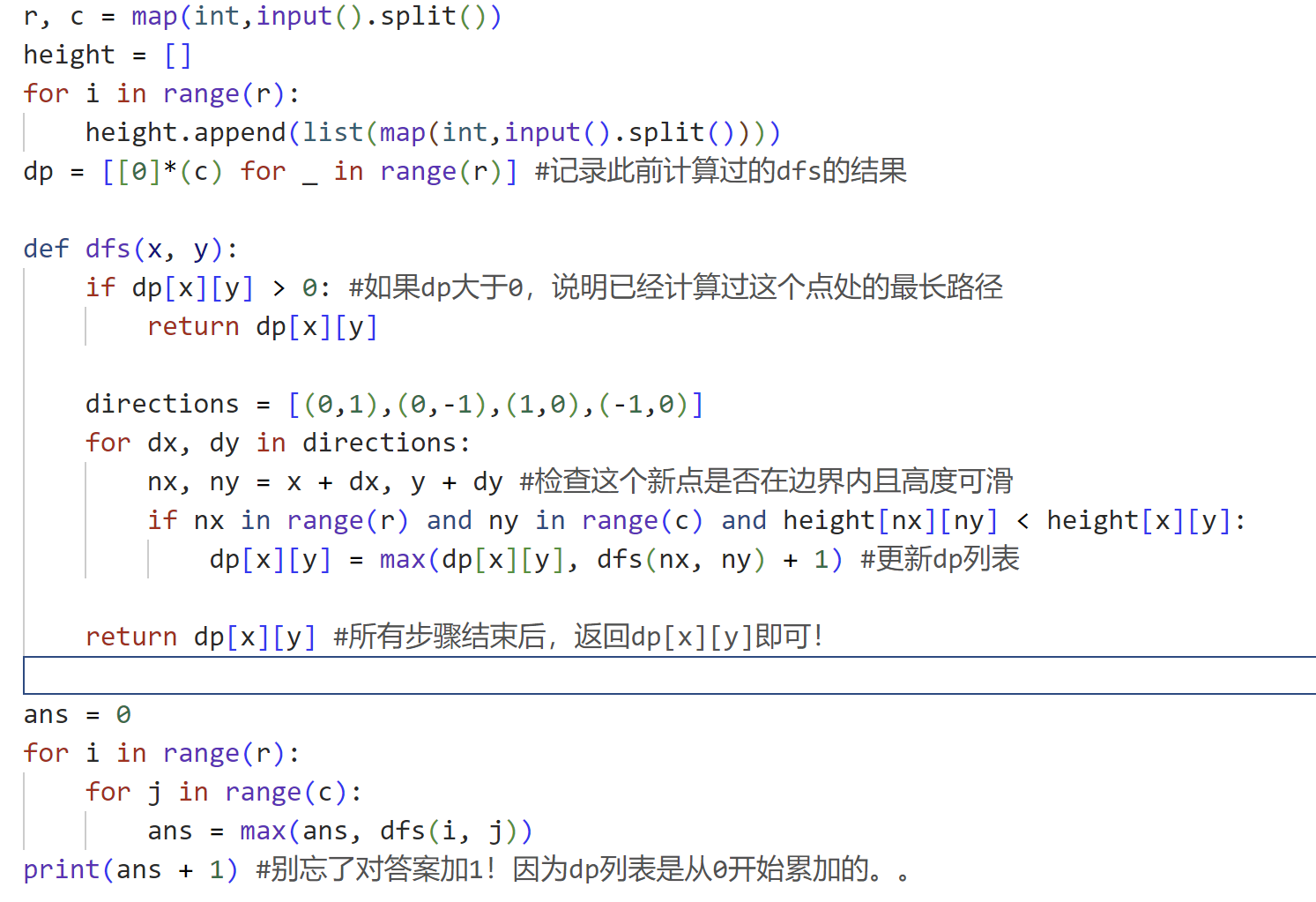


**永远正确的判断一个数是否平方数的方法：**

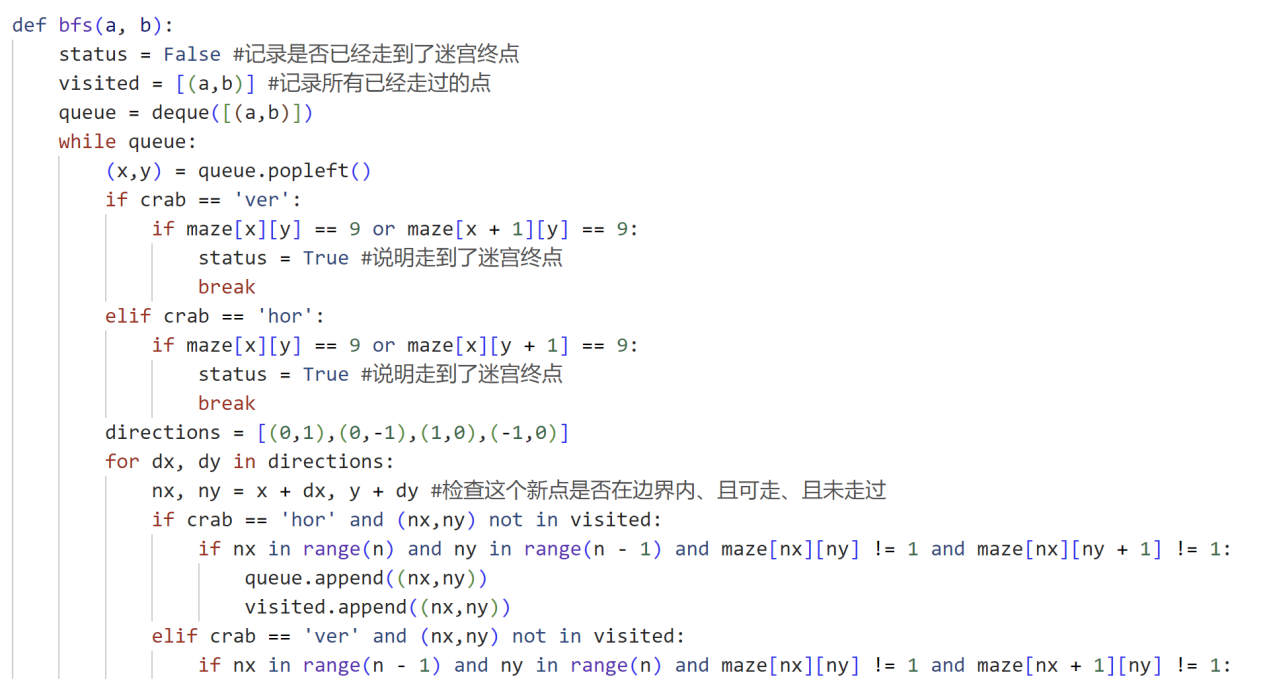


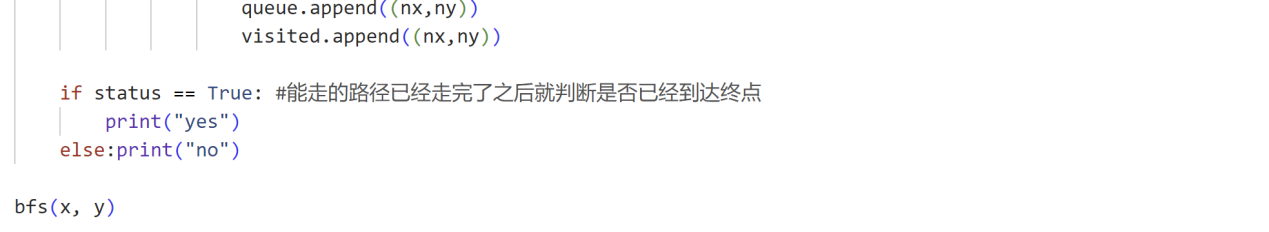
**例子**

**滑雪（dfs+dp）：**

****

**螃蟹（bfs）：**





**奶牛（二分查找）：**



熄灯问题：

把问题化归为**只需要对第一行的灯的情况枚举**即可

垃圾炸弹：

**只需要考虑有哪些垃圾在范围内**即可（这样只用枚举垃圾的数量！）（不需要考虑每个炸弹附近的每个点处是否有垃圾）

炸鸡排问题：

注意到每时每刻都得炸k块，所以**一部分比较小的鸡排每次必定占据不太少的锅**；n-t块鸡排每个时刻都至少占据k-t口锅，所以最小的n-t个和再除以k-t也是一个界！

走山路（dij方法）：

import heapq

def walk(a, b, c, d):

    if mountains[a][b] == '#' or mountains[c][d] == '#':

        print("NO")

    else:

        dist = [[float('inf')]\*n for \_ in range(m)] #记录从出发点到达每一点的最短距离

        dist[a][b] = 0

        queue = [(0,a,b)] #记录当下每个到达的点的所需要的距离和他们的坐标

        status = False #检查是否能够到达终点

        while queue:

            cur\_dis, x, y = heapq.heappop(queue) #将queue视为堆，并pop出距离最小的点

            if x == c and y == d:

                status = True #如果已经到达终点，则记录True

                continue

            if cur\_dis > dist[c][d]: #如果此时的距离已经大于已有的min，则可以扔掉

                continue

            directions = [(0,1),(0,-1),(1,0),(-1,0)]

            for p, q in directions:

                nx, ny = x + p, y + q

                if nx in range(m) and ny in range(n) and mountains[nx][ny] != '#':

                    now\_dis = cur\_dis + abs(int(mountains[x][y]) - int(mountains[nx][ny]))

#到达这个新点的距离

                    if now\_dis < dist[nx][ny]: #如果这是目前到达这个点的最好的方法，则更新它

                        dist[nx][ny] = now\_dis

                        heapq.heappush(queue,(now\_dis, nx, ny)) #将这个新点加入heapq

        if status == True: #最后，检查是否能够到达终点

            print(dist[c][d])

        else:

            print("NO")

m, n, p = map(int,input().split())

mountains = []

for i in range(m):

    mountains.append(list(map(str,input().split())))

for i in range(p):

    a, b, c, d = map(int,input().split())

    walk(a, b, c, d) **加油Jo！！必胜！！**