Exercises for Python Programming

1023 合法的日期

题目描述

判断y年m月d日是否合法

关于输入

一行3个整数y（1000<=y<=9999），m，d，以空格隔开

关于输出

如果合法输出“YES”
否则输出“NO”

例子输入

```
2022 1 1
```

例子输出

```
YES
```

提示信息

m，d的输入没有范围

解题思路

要判断日期是否合法，需分**两步逐层验证**：

1. **验证月份有效性**：月份m必须在`1~12`之间（一年只有12个月，超出则直接不合法）；
2. **验证日期有效性**：不同月份的最大天数不同，分3类判断：
    - 大月（1、3、5、7、8、10、12月）：最大天数为31天；
    - 小月（4、6、9、11月）：最大天数为30天；
    - 二月（2月）：需额外判断**闰年**：
        - 闰年规则：`能被4整除但不能被100整除` **或** `能被400整除`；
        - 闰年2月有29天，平年2月只有28天。

只要任意一步不满足，日期就不合法。

题解代码

```
# 1. 读取输入：将输入的字符串拆分为3个整数（年、月、日）
y, m, d = map(int, input().split())

# 2. 初始化合法性标记（默认先假设合法）
is_valid = True

# 3. 第一步：检查月份是否在1~12之间
if not (1 <= m <= 12):
    is_valid = False  # 月份不合法，标记为False
else:
    # 4. 第二步：根据月份确定该月的最大天数
    if m in [1,3,5,7,8,10,12]:
        max_day = 31  # 大月最多31天
    elif m in [4,6,9,11]:
        max_day = 30  # 小月最多30天
    else:  # 剩下的是2月，需判断闰年
        # 闰年判断逻辑
        if (y % 4 == 0 and y % 100 != 0) or (y % 400 == 0):
            max_day = 29  # 闰年2月29天
        else:
            max_day = 28  # 平年2月28天

    # 检查日期是否在1~max_day之间
    if not (1 <= d <= max_day):
        is_valid = False  # 日期超出范围，标记为False

# 5. 输出结果：合法输出YES，否则NO
print("YES" if is_valid else "NO")
```

病人排队

题目描述

病人登记看病，编写程序将病人按以下原则排出看病的先后顺序：

1. 年龄≥60岁的老年人比其他人优先看病；
2. 老年人按年龄从大到小的顺序看病，年龄相同则按登记的先后顺序排序；
3. 年龄<60岁的病人，按登记的先后顺序排序。

关于输入

1. 第1行：输入一个小于100的正整数，表示病人的个数；
2. 后面按病人登记的先后顺序，每行输入一个病人的信息：长度小于10的字符串（病人ID，每个ID唯一） + 整数（病人年龄）。

关于输出

按排好的看病顺序，每行输出一个病人的ID。

例子输入

```
5
021075 40
004003 15
010158 67
021033 75
102012 30
```

例子输出

```
021033
010158
021075
004003
102012
```

解题思路

核心是**"分组+按规则排序"**：

1. **记录信息**：读取所有病人的ID、年龄，并额外记录**登记顺序**（输入的先后顺序，用索引表示）；
2. **分组**：将病人分为两类：
    - 老人组：年龄≥60岁；
    - 普通组：年龄<60岁；
3. **排序老人组**：排序规则是「年龄降序（大的优先）→ 登记顺序升序（先登记的优先）」；
4. **保持普通组顺序**：普通组直接沿用登记顺序（无需额外排序）；
5. **合并输出**：先输出排序后的老人组ID，再输出普通组ID。

题解代码

```python
# 1. 读取病人个数
n = int(input())

# 2. 记录每个病人的信息：(ID, 年龄, 登记顺序)
patients = []
for idx in range(n):  # idx就是登记顺序（0表示第一个登记）
    id_str, age = input().split()
    age = int(age)
    patients.append( (id_str, age, idx) )

# 3. 分组：拆分老人组和普通组
```

```
elderly = []  # 年龄≥60
ordinary = [] # 年龄<60
for p in patients:
    if p[1] >= 60:
        elderly.append(p)
    else:
        ordinary.append(p)

# 4. 排序老人组：按「年龄降序」→「登记顺序升序」
# 用lambda定义排序键：-p[1]是年龄降序，p[2]是登记顺序升序
elderly_sorted = sorted(elderly, key=lambda p: (-p[1], p[2]))

# 5. 合并结果：先老人组，再普通组
result = elderly_sorted + ordinary

# 6. 输出每个病人的ID
for p in result:
    print(p[0])
```

特别有趣的二进制

题目描述

最近北京大学药学院的小晨在计概课上学习到二进制之后对其产生了浓厚的兴趣，在每次操作仅能移动相邻的0和1的前提下，想知道把一个二进制数转换成另一个二进制数的最小操作数。

关于输入

输入共三行：

1. 第一行为一个整数n（0 < n ≤ 100000），代表二进制数的位数；
2. 第二行为第一个二进制数的每一位（以空格分隔）；
3. 第三行为第二个二进制数的每一位（以空格分隔）。

关于输出

输出将第一个二进制数转换为第二个二进制数的最少操作数，如果答案不存在，则输出-1。

例子输入

```
7
1 1 0 1 0 0 1
0 1 1 0 0 1 1
```

例子输出

```
4
```

解题思路

解决这个问题的核心是抓住**“移动规则”和“转换前提”**：

1. **判断转换可行性**：移动相邻的0/1不会改变二进制数中“1”的总数，因此若两个二进制数的“1”数量不同，直接输出-1；
2. **定位“1”的位置**：分别收集第一个数、第二个数中“1”出现的索引位置（比如从左到右的位置序号）；
3. **计算最小操作数**：由于每次只能移动相邻元素，**第i个“1”从原位置到目标位置的最少步数，就是两者位置差的绝对值**；将所有“1”的移动步数求和，就是总操作数（按顺序匹配“1”的位置时，总步数最小，符合排序不等式的“顺序和最小”规则）。

题解代码

```
# 1. 读取输入数据
n = int(input())
bin1 = input().split()  # 第一个二进制数的每一位（字符串列表）
bin2 = input().split()  # 第二个二进制数的每一位

# 2. 收集两个数中“1”的位置（索引从0开始计数）
pos1 = []  # 存储第一个数里“1”的位置
pos2 = []  # 存储第二个数里“1”的位置
for idx in range(n):
    if bin1[idx] == "1":
        pos1.append(idx)
    if bin2[idx] == "1":
        pos2.append(idx)

# 3. 先判断是否可转换：“1”的数量必须相同
if len(pos1) != len(pos2):
    print(-1)
else:
    # 4. 计算总操作数：对应“1”的位置差的绝对值之和
    total_steps = 0
    for a, b in zip(pos1, pos2):
        total_steps += abs(a - b)
    print(total_steps)
```

多边形游戏2

题目描述

多边形游戏是一种在具有n个顶点的多边形上进行的游戏：

- 每个顶点上有一个整数，每条边有一个运算符（+用`t`表示，*用`x`表示），所有边从1到n编号。
- 游戏首先移除一条边，接下来的操作是：选择一条边m和与之关联的点v1、v2，用一个新的点替换它们，新点的整数为v1的整数用边m的运算符运算v2的结果。

- 无剩余边时游戏结束，得分是最后剩下的顶点的整数。

请编写程序，计算给定多边形可能得到的最高分，并列出第一步移除哪些边可以得到这个最高分（边列表升序排列）。

关于输入

1. 第一行是正整数n（3≤n≤50），表示多边形的边数；
2. 第二行是多边形的描述：包含n条边和n个顶点，按边1→顶点1→边2→顶点2→...→边n→顶点n的顺序给出（边用`t`/`x`表示，顶点为整数）。

关于输出

1. 第一行输出可能得到的最高分；
2. 第二行输出边的列表（升序）：第一步移除这些边时，可能得到最高分。

例子输入

```
4
t -7 t 4 x 2 x 5
```

例子输出

```
33
1 2
```

解题思路

这是**环型区间DP问题**，核心是"拆环为链+维护区间最值"（因为乘法存在负数，最小值可能乘出最大值）：

1. **拆环为链**：
   多边形是环结构，移除某条边k后，可将其拆分为**线性链**（顶点顺序为k+1, k+2, ..., n, 1, 2, ..., k，对应的边为k+1, ..., n, 1, ..., k-1）。
2. **区间DP状态定义**：
   对拆分后的链，定义`dp[i][j][0]`为区间`[i,j]`的**最小值**，`dp[i][j][1]`为区间`[i,j]`的**最大值**（i、j是链中顶点的下标）。
3. **状态转移**：
   对于区间`[i,j]`，枚举分割点`m`（i≤m<j），中间运算符为`op[m]`（对应链中连接顶点m和m+1的边）：
   - 若`op[m]`是`+`：
     - 区间最小值 = 左区间最小值 + 右区间最小值
     - 区间最大值 = 左区间最大值 + 右区间最大值
   - 若`op[m]`是`x`：
     需要考虑4种组合（负负得正）：`左min×右min`、`左min×右max`、`左max×右min`、`左max×右max`，从这4个结果中取最小值和最大值作为当前区间的min和max。
4. **遍历所有拆分情况**：

对每个可能移除的边k，拆分链后执行区间DP，记录该拆分对应的最大值；最后找出全局最高分，并收集所有能得到该分数的边。

题解代码

```
def main():
    import sys
    input = sys.stdin.read().split()
    ptr = 0
    n = int(input[ptr])
    ptr += 1

    # 解析边和顶点：edges[0..n-1]是边1..n，nodes[0..n-1]是顶点1..n
    edges = []
    nodes = []
    for i in range(n):
        edges.append(input[ptr])
        ptr += 1
        nodes.append(int(input[ptr]))
        ptr += 1

    max_total = -float('inf')
    valid_edges = []  # 存储能得到最高分的边（编号1~n）

    # 遍历每个可能移除的边k（编号1~n，对应索引k-1）
    for k in range(1, n+1):
        # 1. 拆环为链：移除边k后，链的顶点顺序是k→k+1→...→n→1→2→...→k-1（索引对应nodes的调整）
        # 链的顶点列表：chain_nodes
        # 链的边列表（连接chain_nodes[i]和chain_nodes[i+1]）：chain_ops
        chain_nodes = []
        chain_ops = []
        # 先加顶点k（原nodes[k-1]），然后加边k+1到n，顶点k+1到n；再边1到k-1，顶点1到k-1
        # 顶点顺序：k, k+1, ..., n, 1, 2, ..., k-1
        for i in range(k-1, n):
            chain_nodes.append(nodes[i])
        for i in range(0, k-1):
            chain_nodes.append(nodes[i])
        # 边顺序：k+1, ..., n, 1, ..., k-1
        for i in range(k, n):
            chain_ops.append(edges[i])
        for i in range(0, k-1):
            chain_ops.append(edges[i])

        # 2. 区间DP初始化：长度为1的区间，min和max都是自身
        len_chain = n  # 链的顶点数等于n
        dp_min = [[0]*len_chain for _ in range(len_chain)]
        dp_max = [[0]*len_chain for _ in range(len_chain)]
        for i in range(len_chain):
            dp_min[i][i] = chain_nodes[i]
            dp_max[i][i] = chain_nodes[i]
```

```
        # 3. 区间DP：枚举区间长度l（从2到n）
        for l in range(2, len_chain+1):
            for i in range(len_chain - l + 1):
                j = i + l - 1
                # 初始化当前区间的min和max为极值
                current_min = float('inf')
                current_max = -float('inf')
                # 枚举分割点m
                for m in range(i, j):
                    op = chain_ops[m]  # 连接m和m+1的边
                    # 左区间[i,m]，右区间[m+1,j]
                    left_min = dp_min[i][m]
                    left_max = dp_max[i][m]
                    right_min = dp_min[m+1][j]
                    right_max = dp_max[m+1][j]

                    # 根据运算符计算所有可能的结果
                    res = []
                    if op == 't':  # 加法
                        res.append(left_min + right_min)
                        res.append(left_max + right_max)
                    else:  # 乘法，考虑4种组合
                        res.append(left_min * right_min)
                        res.append(left_min * right_max)
                        res.append(left_max * right_min)
                        res.append(left_max * right_max)

                    # 更新当前区间的min和max
                    current_min = min(current_min, min(res))
                    current_max = max(current_max, max(res))
                # 赋值给dp
                dp_min[i][j] = current_min
                dp_max[i][j] = current_max

        # 4. 该拆分对应的最大值是dp_max[0][n-1]
        current_score = dp_max[0][len_chain-1]
        # 更新全局最高分和有效边
        if current_score > max_total:
            max_total = current_score
            valid_edges = [k]
        elif current_score == max_total:
            valid_edges.append(k)

    # 输出结果
    print(max_total)
    print(' '.join(map(str, sorted(valid_edges))))

if __name__ == '__main__':
    main()
```

字符统计

题目描述

小明刚刚在计算机课上学习了字符串，现在他遇到了一个棘手的问题，希望你能帮他解决。他有很多的文本，其中只包含26个小写英文字母和空格。他希望能用编程的方式统计出26个英文字母出现的次数。

关于输入

1. 第一行一个正整数n（n≤10）；
2. 接下来有n行，每行一个字符串（仅包含26个小写英文字母和空格），每个字符串长度<200。

关于输出

输出共n行，每行输出从`a`到`z`共26个字母的出现次数，用一个空格隔开。

例子输入

```
2
peking university
the quick brown fox jumps over the lazy dog
```

例子输出

```
0 0 0 0 2 0 1 0 3 0 1 0 0 2 0 1 0 1 1 1 1 0 0 0 1 0
1 1 1 1 3 1 1 2 1 1 1 1 1 1 4 1 1 2 1 2 2 1 1 1 1 1
```

解题思路

这是一个**逐行字符统计**的基础问题，核心是“分类计数+忽略无关字符”，步骤清晰易理解：

1. **初始化计数容器**：对每一行字符串，创建一个长度为26的数组（下标0对应`a`，1对应`b`，...，25对应`z`），初始值均为0；
2. **遍历字符统计**：遍历当前行的每个字符：
   - 若字符是**空格**，直接跳过（不统计）；
   - 若字符是**小写字母**，计算其对应数组下标（`ord(c) - ord('a')`），并将对应位置的计数加1；
3. **输出结果**：每行统计完成后，将计数数组的元素用空格连接成字符串输出。

题解代码（Python，带讲解注释）

```python
def main():
    # 1. 读取需要统计的字符串行数n
    n = int(input())

    # 2. 逐行处理每个字符串
```

```
    for _ in range(n):
        # 读取当前行的字符串（保留空格，仅去除首尾多余空白）
        current_str = input().strip()
        # 初始化a-z的计数数组（26个位置，初始为0）
        char_counts = [0] * 26

        # 3. 遍历字符串的每个字符，统计字母出现次数
        for c in current_str:
            if c == ' ':  # 遇到空格，跳过不统计
                continue
            # 计算当前字母对应的数组下标（a→0, b→1, ..., z→25）
            idx = ord(c) - ord('a')
            char_counts[idx] += 1  # 对应位置计数+1

        # 4. 将计数数组转换为空格分隔的字符串并输出
        print(' '.join(map(str, char_counts)))

if __name__ == '__main__':
    main()
```

购买礼物

题目描述

小明准备给同学购买礼物，因预算有限需筛选部分礼物。每个礼物对应一个优先级（数值越大越优先购买），且每个礼物价格不同。需计算小明在预算内最多能为多少人购买礼物（优先选择优先级高的礼物）。

输入说明

1. 第一行：两个整数 K（礼物总数，$K < 100$）、M（总预算，$M < 1000$）；
2. 后续 K 行：每行包含两个整数 P（礼物价格，$P < 100$）、L（优先级，$L < 200$，所有优先级互不重复）。

输出说明

输出一行整数，表示最多能购买的礼物个数。

示例输入

```
5 10
2 4
6 3
1 5
7 2
1 1
```

示例输出

```
3
```

解题思路

核心逻辑是**“按优先级排序 + 贪心累加”**，步骤如下：

1. **按优先级排序**：将所有礼物按**优先级从大到小**排序（保证优先级高的礼物优先被选择）；
2. **贪心累加价格**：按排序后的顺序，依次累加礼物价格，直到累加总和超过预算 M；
3. **统计有效数量**：记录累加过程中未超过预算的礼物个数，即为最多能购买的数量。

题解代码

```python
# 读取礼物总数K和预算M
K, M = map(int, input().split())

# 存储每个礼物的(价格, 优先级)
gifts = []
for _ in range(K):
    P, L = map(int, input().split())
    gifts.append( (P, L) )

# 按优先级从大到小排序（关键：保证高优先级先被选）
gifts.sort(key=lambda x: -x[1])

total_cost = 0
count = 0
for P, L in gifts:
    # 累加当前礼物价格，若不超过预算则计数
    if total_cost + P <= M:
        total_cost += P
        count += 1
    else:
        # 超过预算则停止（后续礼物优先级更低，无需继续）
        break

# 输出最多能购买的个数
print(count)
```

代码说明

1. **排序逻辑**：通过 `sort(key=lambda x: -x[1])` 实现礼物按优先级降序排列；
2. **贪心累加**：遍历排序后的礼物，仅当当前礼物价格加入后不超过预算时，才累加价格并计数；
3. **终止条件**：一旦某礼物价格导致总费用超过预算，直接终止遍历（后续礼物优先级更低，无需考虑）。

该方法时间复杂度主要由排序决定（$O(K \log K)$）

趣味游戏大赛

题目描述

一年一度的趣味游戏大赛又开幕了。今年的主办方只为所有的参赛选手准备了一道题目，如下：
给定10个整数的序列，要求对其重新排序。排序要求：

1. 奇数在前，偶数在后；
2. 奇数按从大到小排序；
3. 偶数按从小到大排序。

当选手做出这道题目，即可得到丰厚的奖励。你准备好拿奖了吗？

关于输入

输入一行，包含10个整数，彼此以一个空格分开，每个整数的范围是大于等于0，小于等于100。

关于输出

按照要求排序后输出一行，包含排序后的10个整数，数与数之间以一个空格分开。

例子输入

```
4 7 3 13 11 12 0 47 34 98
```

例子输出

```
47 13 11 7 3 0 4 12 34 98
```

解题思路

题目核心是**“分组+分规则排序+合并”**，步骤清晰易执行：

1. **分组**：将10个整数拆分为“奇数组”和“偶数组”；
2. **分规则排序**：奇数组按**从大到小**排序，偶数组按**从小到大**排序；
3. **合并输出**：将排序后的奇数组放在前，偶数组放在后，拼接为最终序列。

题解代码（Python）

```python
# 读取输入的10个整数，转换为列表
nums = list(map(int, input().split()))

# 拆分奇数组与偶数组
odds = []  # 存储所有奇数
```

```
evens = []  # 存储所有偶数
for num in nums:
    if num % 2 != 0:
        odds.append(num)
    else:
        evens.append(num)

# 奇数组降序排序，偶数组升序排序
odds_sorted = sorted(odds, reverse=True)
evens_sorted = sorted(evens)

# 合并两组并输出结果
result = odds_sorted + evens_sorted
print(' '.join(map(str, result)))
```

代码说明

1. **输入处理**：通过 `input().split()` 拆分输入字符串，转换为整数列表；
2. **分组逻辑**：用 `num % 2 != 0` 判断奇数，将元素分别存入奇数组、偶数组；
3. **排序逻辑**：
    - 奇数组通过 `sorted(..., reverse=True)` 实现降序；
    - 偶数组通过 `sorted(...)` 默认实现升序；
4. **合并输出**：拼接排序后的两组，转换为空格分隔的字符串输出。

摘礼物

题目描述

圣诞节就要到了，圣诞树上挂了各式各样的礼物，调皮的小明想把自己能够得到的礼物全摘下来。已知礼物的个数n（n<=1000），和每个礼物的高度i，以及小明能够到的最大高度m。n、i、m均为正整数。你能算出小明最多能摘到多少个礼物吗?

关于输入

输入共3行，第一行为礼物的个数n（n<=1000）。第二行依次输入每个礼物的高度i。第三行输入小明最大能够到的高度m，i、m均为正整数。如果m大于或等于i，则表示能够到。

关于输出

输出1行，为小明能够到的礼物的个数。

例子输入

```
8
100 110 120 130 500 300 250 150
135
```

例子输出

```
4
```

解题思路

题目核心是**筛选并计数"高度不超过小明最大高度"的礼物**，步骤简洁明了：

1. 读取礼物总数 n；
2. 获取所有礼物的高度列表；
3. 读取小明能到达的最大高度 m；
4. 统计高度列表中**小于等于** m 的元素数量，即为结果。

题解代码（Python）

```python
# 读取礼物个数
n = int(input())
# 读取所有礼物的高度（转换为整数列表）
gift_heights = list(map(int, input().split()))
# 读取小明能到的最大高度
m = int(input())

# 统计符合条件的礼物数量
count = 0
for height in gift_heights:
    if height <= m:
        count += 1

# 输出结果
print(count)
```

代码说明

1. **输入处理**：
    - 通过 `input()` 逐行读取数据，利用 `split()` 拆分礼物高度的字符串，并转换为整数列表；
2. **计数逻辑**：遍历礼物高度列表，判断每个高度是否≤小明的最大高度 m，符合条件则计数加1；
3. **结果输出**：直接打印统计得到的有效礼物数量。