第一次小测题目已发布到：
2025 小测练习 - 题单 - 洛谷 | 计算机科学教育新生态
（提示：按住 ctrl 点击链接即可直接跳转到网站。）

题解：
1. 回文相似度计算
方法思路

理解回文对称特性：回文串的核心特性是正读和反读完全相同，即第 1 位字符与倒数第 1 位相同，第 2 位字符与倒数第 2 位相同……对于长度为奇数的字符串，中间的字符无需与其他字符比较，默认与自身对称。

计算对称字符对：遍历字符串的**前半**部分，对于每个位置 i（从 0 开始计数），比较该位置的字符与对称位置 n-1-i 的字符是否相同。统计相同字符对的数量 m 和不同字符对的数量 d。

计算相似度：相似度公式为 m / (m + d)，其中 m + d 是总字符对数（等于字符串长度的一半，向上取整）。

结果格式化：将计算得到的相似度保留两位小数输出。

串	a	b	c	e	f	e	d	b	a
下标	0	1	2	3	4	5	6	7	8
	⬆								⬆

对于一个有 9 个字符的序列，下标是 0 到 8，对于第 i 个字符，它的对称位置是 9-i-1。对于有 n 个字符的序列，下标是 0 到 n-1，对于第 i 个字符，它的对称位置是 n-i-1。

```python
s = input().strip()
n = len(s)
m = 0  # 相同字符对数
d = 0  # 不同字符对数

for i in range((n + 1) // 2):
    left = i
    right = n - 1 - i
    if s[left] == s[right]:
        m += 1
    else:
        d += 1

similarity = m / (m + d)
print(f"{similarity:.2f}")
```

2. 餐厅排队系统

方法思路 1：

根据题目的描述，实现一个排序算法：（此处以插入排序为例）

排序之后再计算查询顾客的位次。

补充：插入排序的思想：每次选取一个元素，根据比较找到它在已有的有序列表中的位置，将其插入进去。例如，现在有一个有序列表 1,2,4,5,6，要插入元素的元素是 3，则首先比较 3 与 6，再比较 3 与 5，再比较 3 与 4，再比较 3 与 2 由于 3＞2，则 3 应插入到 2 后面，4 前面。

补充 2：由于插入排序的时间复杂度高，耗时多，因此对于特别长的列表会超时。

```
# 读取输入并转换为列表
list1_input = input()
list1 = eval(list1_input)
list2_input = input()
list2 = eval(list2_input)
query_name = input().rstrip()

list_merge = list1 + list2

n = len(list_merge)
# 插入排序，逐个插入到前面已排序的部分
for i in range(1, n):
    # 当前要插入的元素（字典）
    current = list_merge[i]
    # 已排序部分的最后一个元素索引
    j = i - 1

    # 向前遍历已排序部分，找到 current 的插入位置
    # 比较依据：current 的 number vs 已排序元素的 number
    while j >= 0 and list_merge[j]["number"] >
current["number"]:
        # 元素后移（给 current 腾出位置）
        list_merge[j + 1] = list_merge[j]
        j -= 1

    # 插入 current 到正确位置
    list_merge[j + 1] = current

num = 0
for customer in list_merge:
    if customer['name'] == query_name:
        num += 1
        break
    num += 1
```

```
print(num)
```

方法思路 2：

注意到查询顾客只有 1 名，对于 1 名顾客，直接在顾客列表中找到有多少号码比他的小（或者找有多少号码比他的大）即可算出查询顾客的位次。这样只要遍历两遍列表，比排序算法更快。

先第一次遍历两个列表获得要查询顾客的号码，并记录顾客在哪个列表之中。

再第二次遍历两个列表，记录有多少号码小于当前顾客的号码。

对于在列表 1 中的号码，还要处理号码相等时，是否排在当前顾客的前面。

最后，把记录的数字再加一，就是当前顾客的位次。

```
# 读取输入并转换为列表
list1_input = input()
list1 = eval(list1_input)
list2_input = input()
list2 = eval(list2_input)
query_name = input().rstrip()

# 遍历两个列表查询顾客的号码
# 用 query_num 记录号码
# 用 list_label 记录在哪个列表中
for customer in list1:
    if customer['name'] == query_name:
        query_num = customer['number']
        list_label = 1

for customer in list2:
    if customer['name'] == query_name:
        query_num = customer['number']
        list_label = 2

# 再次遍历两个列表，计算有多少号码小于当前顾客的号码
# 设置一个计数变量
num = 0
for customer in list1:
    if customer['number'] < query_num :
        num += 1
# 在这个 elif 中处理同号码优先级的情况，当在第一个列表中出现了和
当前顾客相同的号码而且顾客是第二个列表中的，则顾客的位次要+1
    elif customer['number'] == query_num \
    and list_label == 2:
        num += 1

# 不论顾客是在哪个列表，第二个列表中都不需要处理相同号码的情况
```

```
for customer in list2:
    if customer['number'] < query_num :
        num += 1

# 现在的 num 记录了有多少号码小于当前顾客，对于当前顾客的位次，还需要+1
num += 1
# 输出结果
print(num)
```

方法思路 3：

可以使用时间复杂度更低的排序算法（例如使用 python 的 sort()函数会使用归并排序，算法效率比插入排序更快）。

输入处理：通过 eval 函数将输入的字符串格式转换为 Python 列表，同时读取要查询的顾客姓名。

标记列表来源：为了区分两个服务员的顾客（用于号码相同时的排序），给每个顾客字典添加一个来源标记（list1 标记为 0，list2 标记为 1）。

合并列表：将标记后的两个列表合并为一个总列表。

排序：按照“先按号码从小到大排序，号码相同时 list1 顾客优先”的规则排序。排序关键字使用元组 (number, source)，其中 source 为 0 的顾客会排在前面。

查询位次：遍历排序后的列表，找到目标顾客的索引，位次为索引 + 1（因为排队位次从 1 开始）。

```
# 读取输入并转换为列表
list1_input = input()
list1 = eval(list1_input)
list2_input = input()
list2 = eval(list2_input)
query_name = input().rstrip()

# 为每个列表的顾客添加来源标记（list1 为 0，list2 为 1），用于排序优先级
for customer in list1:
    customer['source'] = 0
for customer in list2:
    customer['source'] = 1

# 合并两个列表
combined = list1 + list2

# 排序：先按号码升序，号码相同则按来源升序（list1 优先）
combined.sort(key=lambda x: (x['number'], x['source']))

# 查找查询姓名对应的位次（从 1 开始计数）
```

```
rank = -1
for idx, customer in enumerate(combined):
    if customer['name'] == query_name:
        rank = idx + 1  # 位次从 1 开始
        break

# 输出结果
print(rank)
```

补充：思路 2 和思路 3，对于本题，思路 2 和思路 3 都可以处理长列表的情况，但对于更一般的情况：顾客可能源源不断地来，可能有很多顾客询问自己的位次，这时候思路 3 的效率更高，思路 2 的效率反而比思路 1 的还慢。思路 2 是对于本题的取巧做法。