一．语法：

1. 输入输出类

（1）读取多行输入：

|  |
| --- |
| 法一：应用循环  while True:  try:  line = input() # 读取一行输入  print(f"你输入的内容是：{line}")#或者主要代码部分 except EOFError:  break # 捕获到 EOFError 时退出循环  举例：快速堆猪（多行输入与辅助栈的应用结合例题） a = []  m = []  while True: try:  s = input().split() if s[0] == "pop":  if a:  a.pop() if m:  m.pop() elif s[0] == "min":  if m:  print(m[-1])  else:  h = int(s[1]) a.append(h)  if not m:  m.append(h) else:  k = m[-1]  m.append(min(k, h)) except EOFError:  break  法二：sys:  import sys# 使用 sys.stdin.read() 读取所有输入 data = sys.stdin.read() # 读取整个输入流  print(data) |

(2)list(map(int,input().split()))注意map只是一个生成器，要用list承接，这里应用于输入全是整数 如果不全是整数可以直接input.split() 返回的也是一个列表

注意如果输入中间没有间隔，不要加split，但这样也可以通过 matrix[i][j]搜索到。

输入







小数点



（4）格式化输出：

|  |
| --- |
| 基本模板：f"字符串 {表达式} 字符串"  print(f"My name is {name} and I am {age} years old.")  三元运算符：  print(f"Adult status: {'Yes' if age >= 18 else 'No'}") |

（5）其他注意：

|  |
| --- |
| 无空格输出： print(' '.join(map(str, minStep[i]))) #注意这里应该是字符串形式  保护圈： maze.append([-1] + [int(\_) for \_ in input().split()] + [-1]) |





（6）从列表中删除元素：

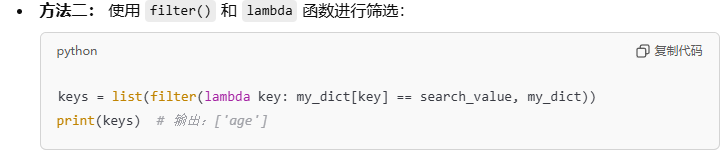
|  |
| --- |
| 法一：my\_list = [1, 2, 3, 4, 2, 5]  my\_list.remove(2) # 删除第一个 2  法二：my\_list = [1, 2, 3, 4, 5]  removed\_element = my\_list.pop(2) # 删除索引为 2 的元素 (即 3) |

2.字典与集合的使用（包括lambda函数的创建）

















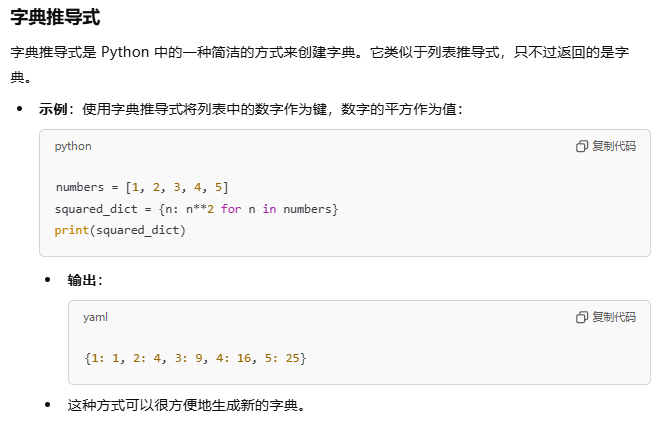


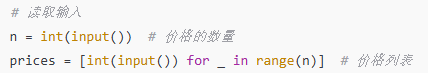
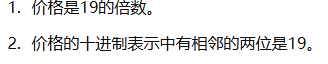


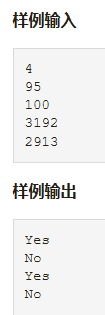
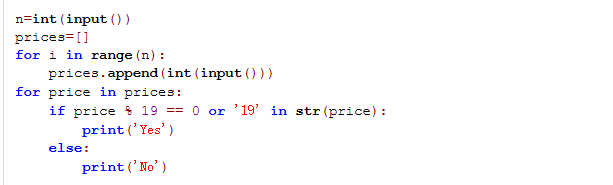












第一个数字是有几行的输入方法

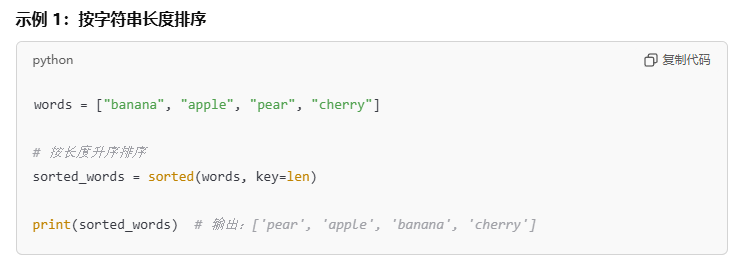
（2）集合的使用：

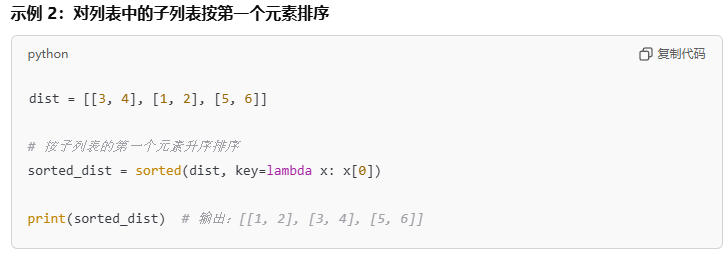
|  |
| --- |
| # 创建集合  my\_set = {1, 2, 3, 4, 5}  another\_set = set([3, 4, 5, 6, 7])  #注意：set() 用来创建集合时，它接受一个可迭代对象（如列表、元组、字符串等），因而这里set() 会自动 从列表中提取元素并创建集合，而不能直接set(3, 4, 5, 6, 7)，因为set()括号里只可以有一个参数，而{} 则不同。  # 添加元素  my\_set.add(6)  # 删除元素（不存在元素可抛出错误） my\_set.remove(2)  # 删除不存在的元素，不会抛出错误 my\_set.discard(10) |

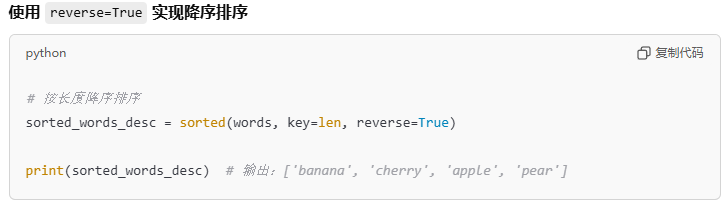
（3）lambda函数的使用（主要是在字典排序中）

|  |
| --- |
| 基本模板：lambda arguments: expression #参数：对参数进行的操作  在字典排序中：  sorted\_dict = sorted(my\_dict.items(), key=lambda x: x[1]) #按值升序排序，注意sorted得到的是一个列表!  #如果想要降序并转化为字典格式如下：  sorted\_dict = dict(sorted(my\_dict.items(), key=lambda x: x[1], reverse=True))  与map结合：  # 对列表中的每个元素进行平方操作  squared\_numbers = list(map(lambda x: x \*\* 2, numbers)) |

3.排序的几种方法与适用范围（语法方面）：



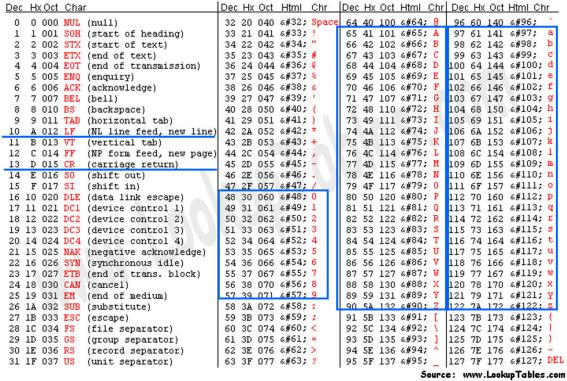






.sort() 是 **列表对象的方法**，只能对 **列表** 进行排序，原地修改列表，即 排序会直接修改原列表， 返回值为 None 。适用于不需要保留原列表的情况，或者希望节省内存的场景。

4.ASCII表及其用法



调用语法：

|  |
| --- |
| # 获取字符的 ASCII 码——ord() ascii\_value = ord('A')  print(ascii\_value) # 输出：65  # 获取 ASCII 码对应的字符——chr() char = chr(65)  print(char) # 输出：A |

5.enumerate:（返回枚举对象）









其他小技巧（剪枝，打标记，滑动窗口） 1.滑动窗口（本质可以从双指针出发）：

**初始化**：

. 维护一个窗口 [start + 1, i] ，表示当前的无重复子串。

. 使用一个字典 char\_index 来记录每个字符最近一次出现的位置。 **扩展窗口**：

 遍历字符串，逐个字符地扩展窗口的右边界 i 。 **收缩窗口**：

. 如果当前字符 c 在字典中且其上次出现的位置在当前窗口内，则需要收缩窗口的左边界 start ，使其不包 含重复字符。

|  |
| --- |
| 模板：  def lengthOfLongestSubstring(s):  start = -1 # 当前无重复子串的起始位置的前一个位置 max\_length = 0 # 最长无重复子串的长度  char\_index = {} # 字典，记录每个字符最近一次出现的位置 for i, char in enumerate(s):#遍历  if char in char\_index and char\_index[char] > start: # 如果字符在字典中且上次出  现的位置大于当前无重复子串的起始位置  start = char\_index[char] # 更新起始位置为该字符上次出现的位置  char\_index[char] = i # 更新字典中字符的位置  current\_length = i - start # 计算当前无重复子串的长度 max\_length = max(max\_length, current\_length)  return max\_length |

2.字典序的大小比较

|  |
| --- |
| n = int(input())  num=input.split()  for i in range(n-1):  for j in range(i+1,n):  if num[i]+num[j]<num[j]+num[i]:--这里是字符串的组合，比较的是字典序 num[i],num[j]=num[j],num[i]  ans = ''.join(num) num.reverse()  ans2 = ''.join(num)  print(ans+' ' +''.join(ans2)) |

3.stack(包括辅助栈)：后进先出

1. **入栈 (Push)**: 使用 append()方法将元素压入栈。

2. **出栈 (Pop)**: 使用 pop()方法将栈顶元素弹出。

3. **栈顶元素 (Peek)**: 使用索引 [-1] 访问栈顶元素。

4. **检查栈是否为空**: 使用 if not stack来判断栈是否为空。

|  |
| --- |
| stack.append(1) # 栈 : [1]  print("栈顶元素 :", stack[-1]) top = stack.pop()  print("出栈元素:", top)  print("栈是否为空:", len(stack) == 0) # 输出: False  stack.clear() # 栈变为空: []  print("栈是否为空:", len(stack) == 0) # 输出: True |

例题：括号匹配：

def is\_valid\_parentheses(s) stack = []

# 括号映射

parentheses\_map = {')': '(', '}': '{', ']': '['} for char in s:

if char in parentheses\_map.values(): stack.append(char)

elif char in parentheses\_map.keys():

if not stack or stack.pop() != parentheses\_map[char]: return False

return not stack

# 测试

print(is\_valid\_parentheses("()[]{}")) # 输出 : True

4.区间问题：

**按照右端点排序**：

（1）不相交区间数目最大

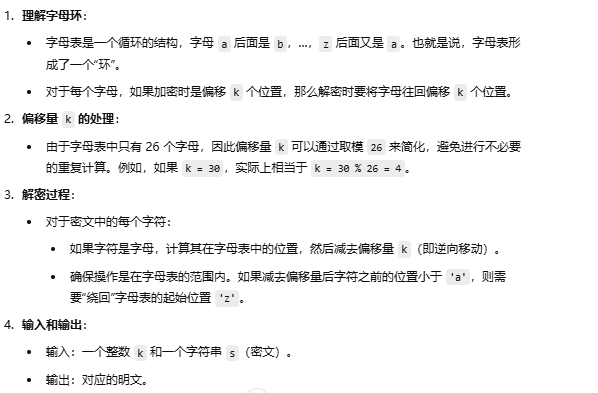
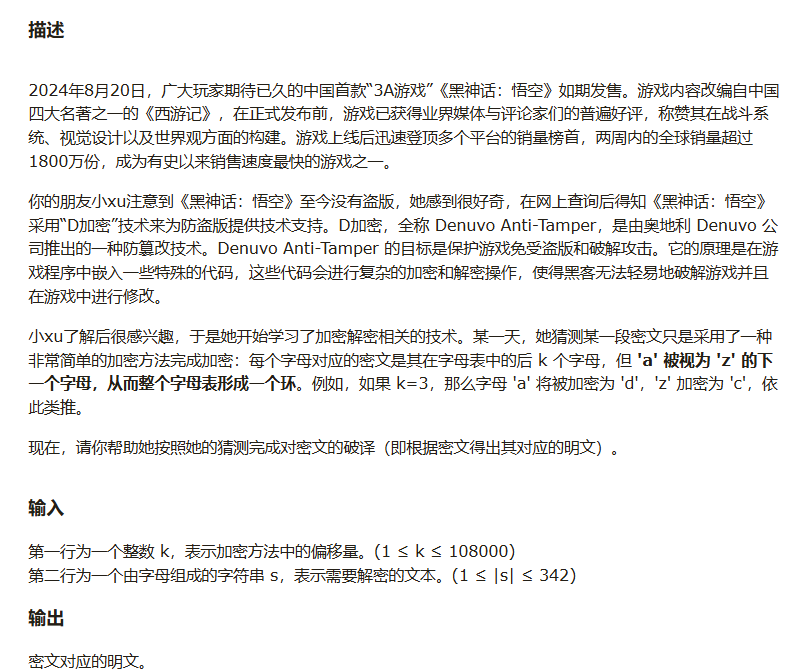
（2）区间选点问题（给出一堆区间，取**尽量少**的点，使得每个区间内**至少有一个点**）——尽量选择当前区间最右 边的点，同不相交区间数目最大

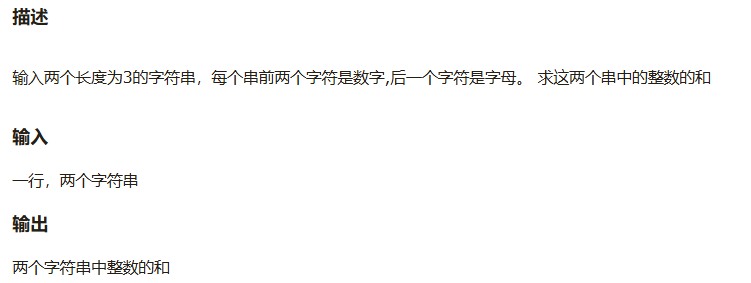
(3)区间覆盖问题：给出一堆区间和一个目标区间，问最少选择多少区间可以**覆盖**掉题中给出的这段目标区间。

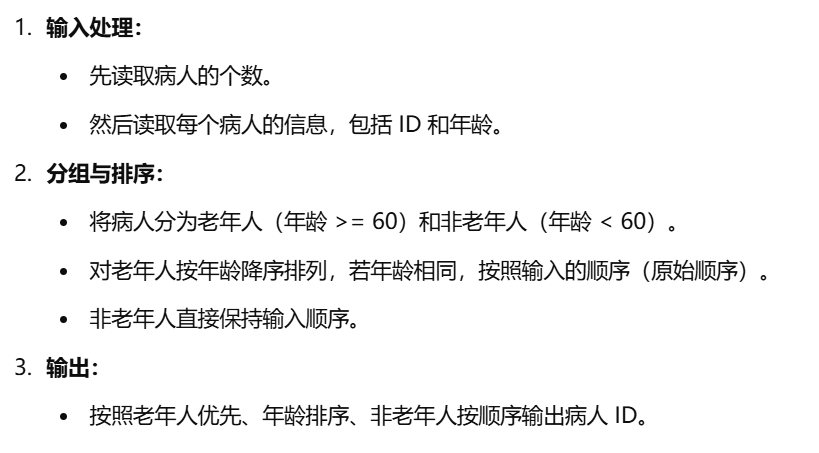
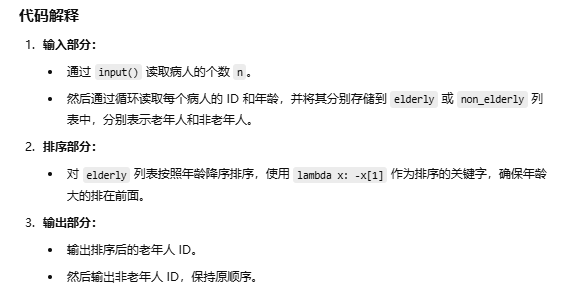
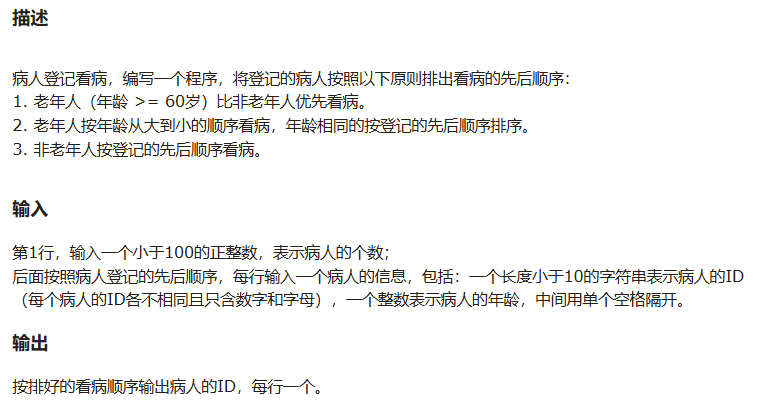
**按照左端点排序**：

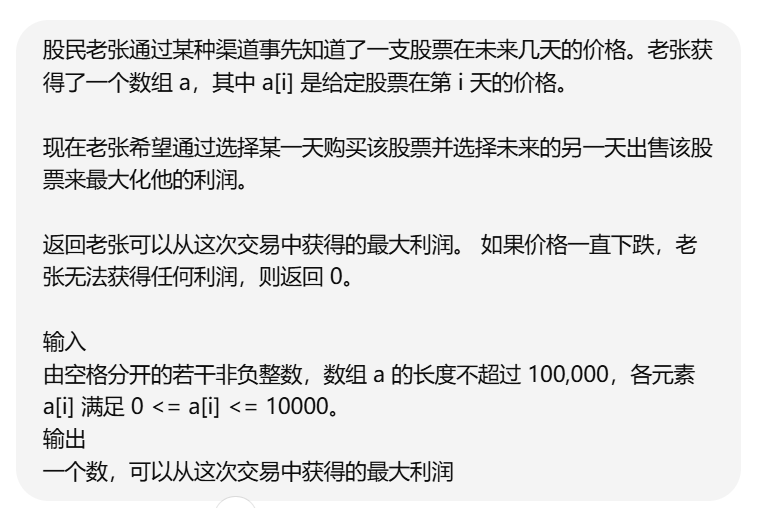
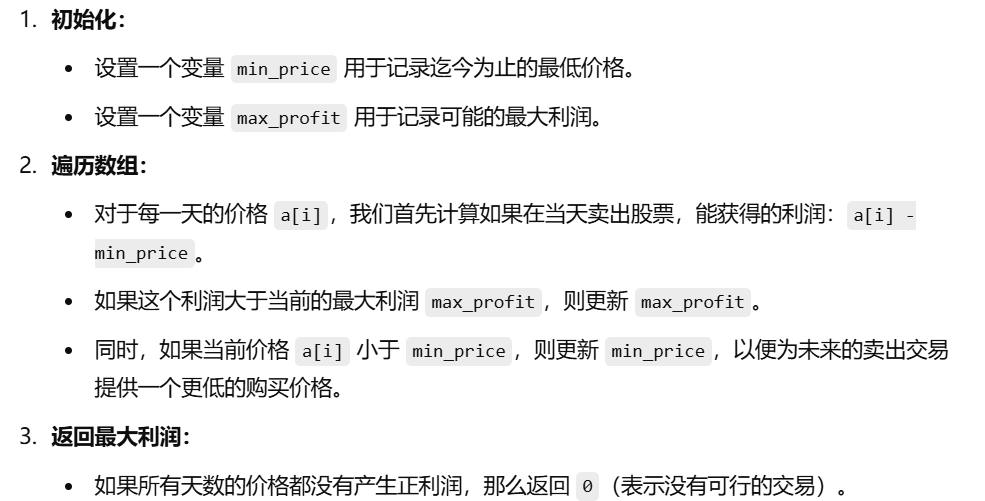
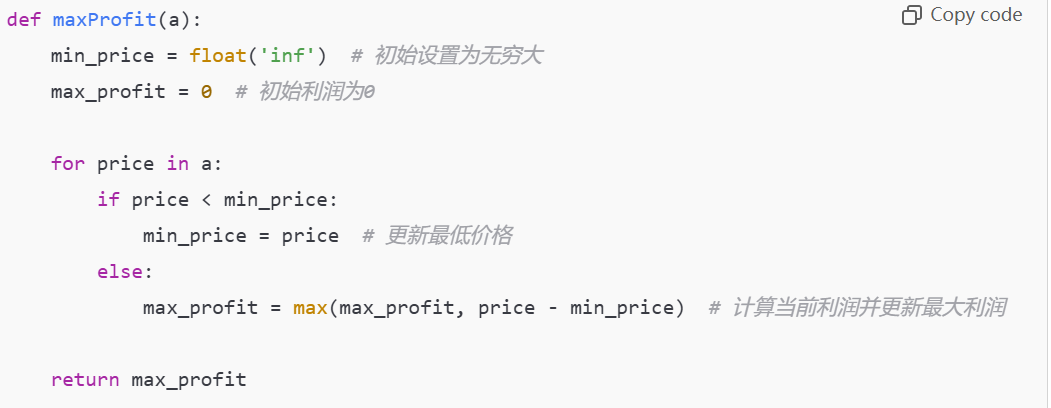
（1）区间合并（左端点由小到大排序，维护前面区间右端点ed）

（2）区间分组问题：从**前往后**依次枚举每个区间，判断当前区间能否被放到某个现有组里面。







****