W为背包最大质量，N为物品个数分别输入物品质量，物品价值求背包中物品价值最大

W, N = list(map(int, input().split()))

wv = [list(map(int, input().split())) for \_ in range(N)]

f = [0] \* (W + 10)

for i in range(N):

for w in range(W, wv[i][0] - 1, -1):

f[w] = max(f[w], f[w - wv[i][0]] + wv[i][1])

print(max(f))

二分搜索

他计划移走一些岩石，使得从起点到终点的过程中，最短的跳跃距离最长。他可以移走除起点和终点外的至多M (0 ≤ M ≤ N) 个岩石。请帮助约翰确定移走这些岩石后，最长可能的最短跳跃距离是多少？输入：第一行包含三个整数L, N, M，相邻两个整数之间用单个空格隔开。接下来N行，每行一个整数，表示每个岩石与起点的距离。岩石按与起点距离从近到远给出，且不会有两个岩石出现在同一个位置。

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

rock = [0]

for i in range(n):

rock.append(int(input()))

rock.append(L)

def check(x):

num = 0

now = 0

for i in range(1, n+2):

if rock[i] - now < x:

num += 1

else:

now = rock[i]

if num > m:

return True

else:

return False

lo, hi = 0, L+1

ans = -1

while lo < hi:

mid = (lo + hi) // 2

if check(mid):

hi = mid

else: # 返回False，有可能是num==m

ans = mid # 如果num==m, mid就是答案

lo = mid + 1

#print(lo-1)

print(ans)

迷宫DFS

现有一个n\*m 大小的迷宫，其中1表示不可通过的墙壁，。表示平地。每次移动只能向上下左右移动一格（不允许移动到曾经经过的位置），且只能移动到平地上。求从迷宫左上角到右下角的所有可行路径的条数。

输入

第一行两个整数n、m （2≤n≤5,2≤m≤5），分别表示迷宫的行数和列数；

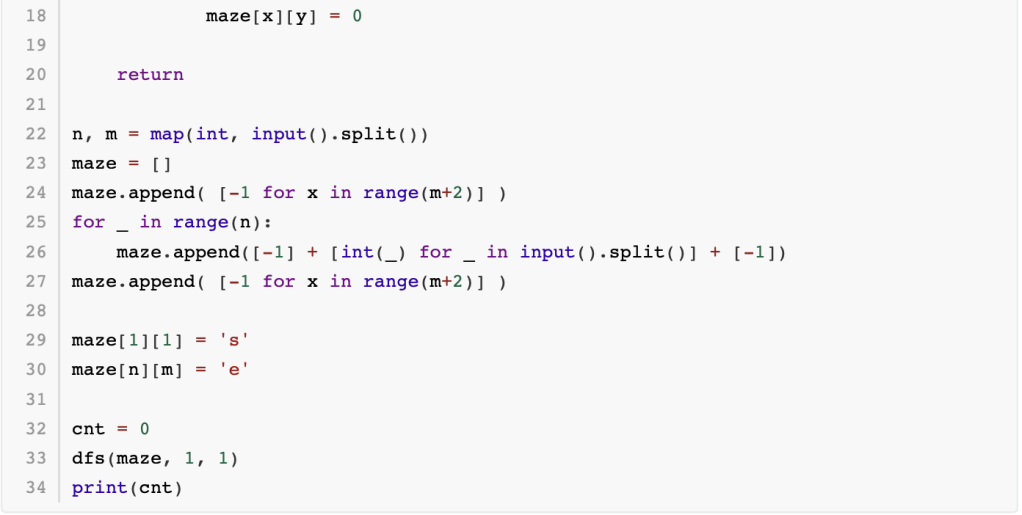
接下来n行，每行m 个整数（值为0或1），表示迷宫。

输出

一个整数，表示可行路径的条数。

global:声明为全局变量





如果要记录路径的话，恢复操作maze[x][y]=0还要加一条list.pop()，list为存储路径的列表

BFS

Billy获得了一张藏宝图，图上标记了普通点（0），藏宝点（1）和陷阱（2）。按照藏宝图，Billy只能上下左右移动，每次移动一格，且途中不能经过陷阱。现在Billy从藏宝图的左上角出发，请问他是否能到达藏宝点？如果能，所需最短步数为多少？

第一行为两个整数m,n，分别表示藏宝图的行数和列数。(m<=50,n<=50)  
此后m行，每行n个整数（0，1，2），表示藏宝图的内容。

q = []

step = [[0, 1], [1, 0], [-1, 0], [0, -1]]

vis = [[0] \* 52 for \_ in range(52)]

g = []

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

for i in range(m):

g.append([int(x) for x in input().split()])

def check(x, y):

if (x < 0 or y < 0 or x >= m or y >= n):

return False

if (vis[x][y] or g[x][y] == 2):

return False

return True

q.append((0, 0))

head = 0

tail = 1

level = 0

while (head < tail):

# i = head

# j = tail

for k in range(head, tail):

x, y = q[head]

head += 1

if (g[x][y] == 1):

print(level)

exit(0)

for z in range(4):

newx = x + step[z][0]

newy = y + step[z][1]

if (check(newx, newy)):

vis[newx][newy] = 1

q.append((newx, newy))

tail += 1

level += 1

print('NO')

某同学在一处山地里，地面起伏很大，他想从一个地方走到另一个地方，并且希望能尽量走平路。现有一个m\*n的地形图，图上是数字代表该位置的高度，"#"代表该位置不可以经过。  
该同学每一次只能向上下左右移动，每次移动消耗的体力为移动前后该同学所处高度的差的绝对值。现在给出该同学出发的地点和目的地，需要你求出他最少要消耗多少体力。第一行是m,n,p，m是行数，n是列数，p是测试数据组数接下来m行是地形图再接下来n行每行前两个数是出发点坐标（前面是行，后面是列），后面两个数是目的地坐标（前面是行，后面是列）（出发点、目的地可以是任何地方，出发点和目的地如果有一个或两个在"#"处，则将被认为是无法达到目的地）

输出n行，每一行为对应的所需最小体力，若无法达到，则输出"NO"

def bfs(x, y):

directions = [(0, -1), (0, 1), (1, 0), (-1, 0)]# 定义方向的偏移量

queue = [(x, y)]# 初始化队列，将起点加入队列

distances = {(x, y): 0} # 初始化距离字典，将起点的距离设为0，其他节点设为无穷大

while queue:

current\_x, current\_y = queue.pop(0)# 弹出队列中的节点

for dx, dy in directions:# 遍历四个方向上的相邻节点

new\_x, new\_y = current\_x + dx, current\_y + dy

if 0 <= new\_x < m and 0 <= new\_y < n:# 判断新节点是否在合法范围内

if d[new\_x][new\_y] != '#':# 判断新节点是否为墙

# 计算新节点的距离new\_distance=distances[(current\_x,current\_y)]+abs(int(d[new\_x][n

ew\_y]) - int(d[current\_x][current\_y]))

# 如果新节点的距离小于已记录的距离，则更新距离字典和队列

if (new\_x, new\_y) not in distances or new\_distance < distances[(new\_x, new\_y)]:

distances[(new\_x, new\_y)] = new\_distance

queue.append((new\_x, new\_y))

return distances

m, n, p = map(int, input().split())# 读取输入

d = []

for \_ in range(m):

row = input().split()

d.append(row)

for \_ in range(p):

x1, y1, x2, y2 = map(int, input().split())# 读取起点和终点坐标

if d[x1][y1] == '#' or d[x2][y2] == '#':# 判断起点和终点是否为墙

print('NO')

continue

distances = bfs(x1, y1)# 使用BFS计算最短距离

# 输出结果，如果终点在距离字典中，则输出对应的最短距离，否则输出'NO'

if (x2, y2) in distances:

print(distances[(x2, y2)])

else:

print('NO')

节省空间，时间(要加在函数前面)，参数不包含可变类型，如列表

from functools import lru\_cache

@lru\_cache(maxsize=None)

@lru\_cache(maxsize=2048)也可以

防爆栈

import sys

sys.setrecursionlimit(1<<30)

import copy

matrix\_=copy.deepcopy(matrix)

可以在改动matrix时matrix\_不变

快排

def quicksort(arr):

if len(arr)<=1:

return arr

else:

pivot=arr[0]

left=[x for x in arr[1:] if x<pivot]

right=[x for x in arr[1:] if x>=pivot]

return quicksort(left)+[pivot]+quicksort(right)

应用

arr=[1,7,4,1,10,9,-2]

sorted\_arr=quicksort(arr)

此时sort\_arr=[-2,1,1,4,7,9,10]

埃氏筛算法生成范围内的所有素数

import math

def sieveOfEratosthenes(n):

primes = [True] \* (n+1)

primes[0] = primes[1] = False

p = 2

while p \* p <= n:

if primes[p]:

for i in range(p \* p, n+1, p):

primes[i] = False

p += 1

return primes

最长上升子序列

dp=[1]\*n

for i in range(n):

for j in range(i):

if numbers[j]<numbers[i]:

dp[i]=max(dp[i],dp[j]+1)

print(max(dp))

try-execpt见正则举例

try:

while True:

pro=input()

if not pro:

break

except EOFError:

pass

continue 解释当前循环，进入下次循环

len(x) x可以是字符串、列表、字典

type(x) returns the type of x (string, float, list, dict …)

str(x) converts x to string

list(x) converts x to a list

int(x) converts x to a integer number

int(str,2) 2是进制

float(x) converts x to a float number

eval(‘+’) 字符转换为运算符

map(function, L) Applies function to values in L

使⽤反斜杠 \ 在⾏尾进⾏换⾏

print(string,list,sep=’str’) 以️str 为分隔输出string,list 默认为,

print()分行print(“qweerty”

“qrwfbsdf”)

print("{:.2f}".format(x))) 输出两位有效数字x

print(f"{num:.2f}")可以输出保留两位小数的num

bisect 模块是 Python 中用于处理已排序序列的模块之一。它提供了用于插入元素和查找元素插入位置的函数。

bisect.bisect\_left(arr, x, lo=0, hi=len(arr))：返回在已排序数组 arr 中将元素 x 插入的位置，如果元素已经存在，则返回其左侧的位置。

bisect.bisect\_right(arr, x, lo=0, hi=len(arr))：返回在已排序数组 arr 中将元素 x 插入的位置，如果元素已经存在，则返回其右侧的位置。

bisect.insort\_left(arr, x, lo=0, hi=len(arr))：向已排序数组 arr 中插入元素 x，保持数组的有序性。

bisect.insort\_right(arr, x, lo=0, hi=len(arr))：向已排序数组 arr 中插入元素 x，保持数组的有序性。

高效读取大量输入

import sys

input = sys.stdin.readline

然后照常输入代码

iteetools模块

itertools.permutations(iterable, r)是itertools模块中的一个函数，用于生成指定长度的元素排列。其中，iterable是输入的可迭代对象，r是用于生成排列的长度。

itertools.combinations(iterable, r)是itertools模块中的一个函数，用于生成指定长度的元素组合。其中，iterable是输入的可迭代对象，r是用于生成组合的长度。

import itertools

perms = list(itertools.permutations([1, 2, 3], 2))

print(perms)#得到[(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)]

combs = list(itertools.combinations([1, 2, 3], 2))

print(combs)#得到[(1, 2), (1, 3), (2, 3)]

数

abs() 绝对值

bin(num) 将十进制转为二进制

oct(num) 将十进制转为八进制

hex(num) 将十进制转为十六进制

round(n1,n) 是 Python 中用于对浮点数进行四舍五入的函数。其中，n1是要进行四舍五入的数字，n是保留的小数位数。

带小数的运算结果都是小数

/除法：结果是小数

\_在数中起分隔作用 1\_000=1000

math.pi

math.ceil() 向上取整

math.floor() 向下取整

math.pow(x,y) x的y次方,y不一定是整数，返回float

math.inf 无穷大

math.log(x, base) 以base为底的对数，base默认值为e

math.sin()

math.cos()

math.tan()

math.degrees() 弧度转化成角度

math.radians() 角度转换为弧度

math.comb(n, k) 函数返回从 n 个项目中选取 k 个项目的组合数

math.perm(n, k)函数返回从 n 个项目中选取 k 个项目的排列数

math.e

math.sqrt() 开平方

math.gcd(x,y) 求x与y的最大公约数

float判断不能用‘==’要用绝对值小于某个极小量是可以的

math.isclose(a, b, \*, rel\_tol=1e-09, abs\_tol=0.0)接受两个参数a和b，分别表示要比较的两个浮点数。此外，它还接受两个可选的命名参数rel\_tol和abs\_tol，分别表示相对误差和绝对误差的阈值，相等返回True

random.random() 生成一个 0 到 1 之间的随机浮点数。

random.randint(a, b) 生成一个 a 到 b 之间的随机整数（包括 a 和 b）。

random.choice(sequence) 从序列 sequence 中随机选择一个元素并返回。序列可以是列表、元组或字符串。

random.shuffle(sequence) 将序列 sequence 中的元素随机打乱。注意，shuffle 函数会直接修改原始序列，而不会返回新的打乱后的序列副本。

random.sample(population, k) 从总体 population 中随机选择 k 个样本，并以列表的形式返回这些样本，而不会改变原始总体数据。

random.uniform(a, b) 生成一个 a 到 b 之间的随机浮点数，包括 a 但不包括 b。

～ 按位取反，即把每个二进制的数变一次，由于最高位是决定正负号，故正整数~n=-n-1

堆

**import** heapq

将列表转化为堆

pq **=** [1,2,3]

heapq**.**heapify(pq) *#转为堆*

**print**(pq[0]) *#1*

**print**(pq[**-**1]) *#3*

*# 创建一个空堆*

heap **=** []

*# 使用heapq模块的heappush函数将元素插入堆中*

heapq**.**heappush(heap, 2)

heapq**.**heappush(heap, 1)

heapq**.**heappush(heap, 3)

*# 输出堆中的元素print(heap) # 输出: [1, 2, 3]*

j=heapq.heappop(heap)*#弹出最小元素，并赋值给j*

j=heap[0]*#将最小元素赋值给j*

def find\_largest\_k\_elements(nums, k):

min\_heap = [] *# 使用小顶堆来存储数据集中最大的K个元素*

for num in nums:

if len(min\_heap) < k:

heapq.heappush(min\_heap, num)

else:

if num > min\_heap[0]: *# 如果当前元素大于堆顶元素，则将堆顶元素弹出，再插入当前元素*

j=heapq.heappop(min\_heap)

heapq.heappush(min\_heap, num)

*#与以下等价：*

*#j=heapq.heapreplace(min\_heap,num)*

return sorted(min\_heap, reverse=True) *# 最后取出堆中的元素并排序，即为最大的K个元素*

merged=heapq.merge(heap1,heap2)*#将heap1,heap2合并为merged*

集合

s={1,2,3,4}

s=set()

s={}错误：创建的是字典

s=set(list/range/元组/字符串)

s=set(‘lkjhjk’) #s={‘l’,’k’,’j’,’h’}

’j’ in s #True判断

s.add(‘er’) #s={‘l’,’k’,’j’,’h’,’er’}加一个元素

s.update()#添加至少一个元素,括号中的东西与set()中的一样，效果一样

s.remove(‘l’)#删除’l’，不存在时报错

s.discard(‘l’)#输出’l’，不存在时不报错

s.pop()#删除任意元素

s.clear()#清空集合

s1.issubset(s2)#判断s1是否是s2的子集

s2.issuperset(s1)#判断s2是否是s1的超集

s1.isdisjoint(s2)#判断s1、s2是否有交集

s=s1.intersection(s2)#s是s1与s2交集,对s1、s2无影响

s=s1&s2 #s是s1与s2交集,对s1、s2无影响

s=s1.union(s2) #s是s1与s2并集,对s1、s2无影响

s=s1 | s2 #s是s1与s2并集,对s1、s2无影响

s=s1.difference(s2)#s是s1有的s2没有的元素组成的集合，又称差集

s=s1-s2 #s是s1有的s2没有的元素组成的集合，又称差集

s=s1^s2 #两个集合中所有不重叠的元素的集合,即对称差集

s=s1.symmetric\_difference(s2)#两个集合中所有不重叠的元素的集合,即对称差集

矩阵

输出

for i in range():

output=' '.join(map(str,matrix[i][]))

print(output)

创建矩阵matrix = [[0] \* (m + 2) for \_ in range(n + 2)]

乘法

for (x\_row, x\_col), x\_val in x\_dict.items():

for (y\_row, y\_col), y\_val in y\_dict.items():

if x\_col == y\_row:

prod = x\_val \* y\_val

key = (x\_row, y\_col)

if key in result\_dict:

result\_dict[key] += prod

else:

result\_dict[key] = prod

加保护圈

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

maze = []

maze.append [-1 for x in range (m+2)] )

for - in range (n):

maze.append([-1] + [int(\_) for \_ in input() •split ()] + [-1])

maze.append( (-1 for x in range (m+2)] )

字符串

split()是Python中的一个字符串方法，它可以将一个字符串按照指定的分隔符分割成多个子字符串，并返回一个列表。不包含sep内容。它的使用方法如下：

string.split(sep, maxsplit)

其中，string是要分割的字符串，sep是分隔符（默认为空格），maxsplit是可选参数，用于指定最大分割次数。

strip()是Python中的一个字符串方法，它用于去除字符串两端的指定字符（默认为空格字符）。strip()方法的使用方式如下：

string.strip(characters)

string.rstrip()除尾端空白

string.lstrip()除句首空白

其中，string是要进行操作的字符串，characters是可选参数，用于指定要去除的字符。如果不指定characters，则默认去除字符串两端的空格字符。需要注意的是，**strip()方法不会修改原始的字符串，而是返回一个新的字符串。**如果你希望修改原始的字符串，你可以将返回值赋值给原始的字符串变量，例如：s = s.strip()。

string.replace（x,y）会把所有x换成y非原地

string.zfill(x) 在左侧填充0至字符串长度为x,非原地

string.rejust(x,’m’) 在左侧填充’m’至字符串长度为x,非原地

string[i:j]包括i不包括j

string.upper() 所有字母大写，非原地

string.lower() 所有字母小写，非原地

string.capitalize() 首字母大写，非原地

string.count(x) counts how many times x appears

string.find(x) position of the x first occurrence ,若没有，输出-1

str.startswith(prefix)是Python中字符串对象的一个方法，用于检查字符串是否以指定的前缀开头。如果字符串以指定的前缀开头，该方法返回True，否则返回False。

后缀str.endswith(suffix)

str.isalpha()是Python中字符串对象的一个方法，用于检查字符串是否只包含字母字符。如果字符串中的所有字符都是字母，则该方法返回True，否则返回False

数字：str.isdigit() 字母和数字：str.isalnum()

‘.’.join(L) returns a string with L values joined by . L[0].L[1].L[2]

string.format(x) returns a string that includes formatted x

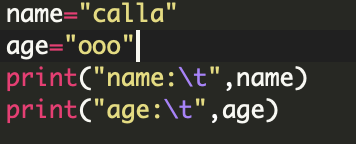
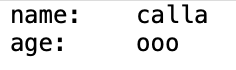
合并字符串string=f"{string1}{string2}"

使⽤ ord() 函数来返回字符的 ASCII 值 chr()返回ASCII对应的字符

string.title()单词首字母大写

\n换行

\t列表模式对齐

 得到 

字典序：在ASCII表中的排序。

字符串的比较：从头到尾一一比较eg”2”>”10”,because”2”>”1”

列表

del list[i]

list.append(x) 列表后加单个元素

list.extend(L) 列表后加单个列表

list.insert(i,x) 列表i位置加x，i以后的都后退一个索引

list.remove(x) removes the first list item whose value is x

list.pop(i) removes the item at position i and returns its value 既有输出又有原地改动

list.clear() removes all items from the list

list.index(x) 返回指定元素 x 在列表中第一次出现的索引

list.count(x) 用于计算列表中某个元素 x 出现的次数

list.sort() sorts list items (reverse=True降序)

sorted(list) 排序，**不会修改原始的序列，而是返回一个新的序列**

len(list) 列表长度

list.reverse() 反排

list.copy() returns a copy of the list 效果与list\_=list[:]]相同，list\_不随list变化而变化

if list: 当列表不为空时

list.remove(x)会除去第一个x

range(i,j) 包括i不包括j,得到的不是列表,甚至直接用print()会得到range(i,j)

list[i:j]包括i不包括j

candies.sort(key=lambda x: x[0]/x[1], reverse=True) 是⼀个排序操作，⽤于对 candies 列表中的糖果进⾏排序。在这⾥，我们根据每个糖果的单位重量价值（即每克糖果 的价值）进⾏排序，单位重量价值⾼的排在前⾯。这⾥使⽤了 sort 函数的 key 参数，它接受⼀个函数作为参数，⽤于指定⽤于排序的⽐较指 标。 lambda x: x[0]/x[1] 是⼀个匿名函数，它接受⼀个参数 x ，表示糖果的元组（价值，重量），并返回糖果的单位重量价值 x[0]/x[1] 所以， key=lambda x: x[0]/x[1] 意味着我们使⽤糖果的单位重量价值作为排序的依据

key=lambda x: (x[0],x[1])先按照第一个元素排序，若第一个元素相等再按照第二个元素排序

for index,value in enumerate([‘a’,’b’,’c’])遍历列表中的元素，并返回元素的索引和值

from collections import Counter

data=[‘a’,’s’,’d’,’a’,’s’,’a’]

counter\_result=Counter(data)

print(counter\_result)#输出{‘a’:3,’s’:2,’d’:1}

字典

按照字典的键来排序sorted\_dict =dict(sorted(my\_dict.items()))

按照字典的值来排序sorted\_dict = dict(sorted(my\_dict.items(), key=lambda item: item[1]))

dict[k] retrieves the item with key k

del dict[k] removes the item with key k

dict.keys() returns a list of keys

dict.values() returns a list of values

dict.items() returns a list of pairs (key,value)

dict.get(k,’y’) returns the value associtated to the key k ,没有x对应的值时，返回’y’

dict.pop() removes the item associated to the key and returns its value

dict.update(D) adds keys-values (D) to dictionary 相当于两个字典合并

dict.clear() removes all keys-values from the dictionary

dict.copy() returns a copy of the dictionary

dic[key]=value 添加一对

set(dic.values()) 将字典中的所有值提取出来，并去除重复值，最终生成一个包含所有不重复值的集合

for key,value in dict.items(): 遍历字典所有键与值，其余以此类推

from collections import defaultdict

my\_dict = defaultdict(int) # 默认值为0的字典

my\_dict = defaultdict(lambda: 100)#默认值为100

my\_dict.default\_factory = lambda: 200# 修改默认工厂函数为返回值为200

my\_dict['a'] = 1

print(my\_dict['a']) # 打印输出：1

print(my\_dict['b']) # 打印输出：0，因为 'b' 不存在，会返回默认值 0

deque

from collections import deque

queue = deque()

queue.append(1) # 将元素 1 添加到队列的右侧

item = queue.popleft() # 从队列的左侧取出元素

item = queue.pop() # 从栈的右侧取出元素

d = deque([1, 2, 3, 4, 5])

d.rotate(2) # 将队列向右循环移动2个位置

print(d) # 输出: deque([4, 5, 1, 2, 3])

限制大小的队列： 可以使用 maxlen 参数创建一个具有固定大小的双向队列，当超过指定大小时，旧的元素会被自动移除。

d = deque(maxlen=3)

d.append(1)

d.append(2)

d.append(3)

print(d) # 输出: deque([1, 2, 3])

d.append(4)

print(d) # 输出: deque([2, 3, 4])

索引访问、拼接与list一样

正则表达式

以下是一些常用的正则表达式元字符及其含义：

.: 匹配任意单个字符（除换行符外）

\.：匹配.

\w: 匹配任意字母、数字或下划线字符

\d: 匹配任意数字字符

\s: 匹配任意空白字符（包括空格、制表符、换行符等）

[]: 匹配方括号内的任意一个字符

^: 匹配字符串的开始位置，或取反

$: 匹配字符串的结束位置

\*: 匹配前一个元字符的0次或多次重复

+: 匹配前一个元字符的1次或多次重复

?: 匹配前一个元字符的0次或1次重复

{n}: 匹配前一个元字符的n次重复

{n, m}: 匹配前一个元字符的n至m次重复

|: 匹配多个表达式中的一个

在正则表达式中，这些字符通常与其他字符一起使用来构建匹配模式。例如，使用\d+可以匹配一个或多个数字字符，使用\w+可以匹配一个或多个字母、数字或下划线字符。

此外，还有一些特殊字符需要进行转义，如\、[、]、(、)、{、}等，以便其能够正常匹配对应的字符本身。

举例：

描述 POJ 注册的时候需要用户输入邮箱，验证邮箱的规则包括： 1)有且仅有一个'@'符号 2)'@'和'.'不能出现在字符串的首和尾 3)'@'之后至少要有一个'.'，并且'@'不能和'.'直接相连 满足以上3条的字符串为合法邮箱，否则不合法， 编写程序验证输入是否合法 输入 输入包含若干行，每一行为一个代验证的邮箱地址，长度小于100 输出 每一行输入对应一行输出 如果验证合法，输出 YES 如果验证非法：输出 NO

import re

def validate\_email(email):# 使用正则表达式验证邮箱格式

pattern = r'^[^@\s]+@[^@\s]+\.[^@\s]+$'

if re.match(pattern, email):

return "YES"

else:

return "NO"

# 逐行读取输入并验证邮箱地址

while True:

try:#这段代码是一个无限循环的结构，表示会一直循环执行下去。

在循环内部，使用 try-except 结构来捕获可能发生的异常。

try 块中的代码是尝试执行的部分，这里是用来获取用户输入的电子邮件地址。input() 函数用于获取用户的输入，并将输入的值赋给变量 email。

如果用户输入时没有发生异常，那么代码会继续执行循环，再次等待用户的输入。

而如果用户在输入时发生了异常，比如输入了一个无效的值，那么代码会跳转到 except 块中执行异常处理的逻辑。

email = input()

if not email:

break

print(validate\_email(email))

except EOFError:#EOFError 是一个内置的异常类型，它会在 input() 函数遇到文件结束（例如用户按下了 Ctrl+D）时引发。

break

## 一、校验数字的表达式 1 数字：^[0-9]\*$ 2 n位的数字：^d{n}$ 3 至少n位的数字：^d{n,}$ 4 m-n位的数字：^d{m,n}$ 5 零和非零开头的数字：^(0|[1-9][0-9]\*)$ 6 非零开头的最多带两位小数的数字：^([1-9][0-9]\*)+(.[0-9]{1,2})?$ 7 带1-2位小数的正数或负数：^(-)?d+(.d{1,2})?$ 8 正数、负数、和小数：^(-|+)?d+(.d+)?$ 9 有两位小数的正实数：^[0-9]+(.[0-9]{2})?$ 10 有1~3位小数的正实数：^[0-9]+(.[0-9]{1,3})?$ 11 非零的正整数：^[1-9]d\*$或^([1-9][0-9]\*){1,3}$或^+?[1-9][0-9]\*$ 12 非零的负整数：^-[1-9][]0-9"\*$或^-[1-9]d\*$ 13 非负整数：^d+$或^[1-9]d\*|0$ 14 非正整数：^-[1-9]d\*|0$或^((-d+)|(0+))$ 15 非负浮点数：^d+(.d+)?$或^[1-9]d\*.d\*|0.d\*[1-9]d\*|0?.0+|0$ 16 非正浮点数：^((-d+(.d+)?)|(0+(.0+)?))$或^(-([1-9]d\*.d\*|0.d\*[1-9]d\*))|0?.0+|0$ 17 正浮点数：^[1-9]d\*.d\*|0.d\*[1-9]d\*$或 ^(([0-9]+.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*.[0-9]+)|([0-9]\*[1-9][0-9]\*))$ 18 负浮点数：^-([1-9]d\*.d\*|0.d\*[1-9]d\*)$或 ^(-(([0-9]+.[0-9]\*[1-9][0-9]\*)|([0-9]\*[1-9][0-9]\*.[0-9]+)|([0-9]\*[1-9][0-9]\*)))$ 19 浮点数：^(-?d+)(.d+)?$或^-?([1-9]d\*.d\*|0.d\*[1-9]d\*|0?.0+|0)$ 二、校验字符的表达式 1 汉字：^[一-龥]{0,}$ 2 英文和数字：^[A-Za-z0-9]+$或^[A-Za-z0-9]{4,40}$ 3 长度为3-20的所有字符：^.{3,20}$ 4 由26个英文字母组成的字符串：^[A-Za-z]+$ 5 由26个大写英文字母组成的字符串：^[A-Z]+$ 6 由26个小写英文字母组成的字符串：^[a-z]+$ 7 由数字和26个英文字母组成的字符串：^[A-Za-z0-9]+$ 8 由数字、26个英文字母或者下划线组成的字符串：^w+$或^w{3,20}$ 9 中文、英文、数字包括下划线：^[一-龥A-Za-z0-9\_]+$ 10 中文、英文、数字但不包括下划线等符号：^[一-龥A-Za-z0-9]+$或^[一-龥A-Za-z0-9]{2,20}$ 11 可以输入含有^%&',;=?$"等字符：[^%&',;=?$"]+ 12 禁止输入含有~的字符：[^~"]+ 三、特殊需求表达式 1 Email地址：^w+([-+.]w+)\*@w+([-.]w+)\*.w+([-.]w+)\*$ 2 域名：[a-zA-Z0-9][-a-zA-Z0-9]{0,62}(/.[a-zA-Z0-9][-a-zA-Z0-9]{0,62})+/.? 3 InternetURL：[a-zA-z]+://[^s]\*或^http://([w-]+.)+[w-]+(/[w-./?%&=]\*)?$ 4 手机号码： ^(13[0-9]|14[5|7]|15[0|1|2|3|5|6|7|8|9]|18[0|1|2|3|5|6|7|8|9])d{8}$ 5 电话号码("XXX-XXXXXXX"、"XXXX-XXXXXXXX"、"XXX-XXXXXXX"、"XXX-XXXXXXXX"、"XXXXXXX"和"XXXXXXXX)：^((d{3,4}-)|d{3.4}-)?d{7,8}$ 6 国内电话号码(0511-4405222、021-87888822)：d{3}-d{8}|d{4}-d{7} 7 身份证号(15位、18位数字)：^d{15}|d{18}$ 8 短身份证号码(数字、字母x结尾)：^([0-9]){7,18}(x|X)?$或^d{8,18}|[0-9x]{8,18}|[0-9X]{8,18}?$ 9 帐号是否合法(字母开头，允许5-16字节，允许字母数字下划线)： ^[a-zA-Z][a-zA-Z0-9\_]{4,15}$ 10 密码(以字母开头，长度在6~18之间，只能包含字母、数字和下划线)： ^[a-zA-Z]w{5,17}$ 11 强密码(必须包含大小写字母和数字的组合，不能使用特殊字符，长度在8-10之间)：^(?=.\*d)(?=.\*[a-z])(?=.\*[A-Z]).{8,10}$ 12 日期格式：^d{4}-d{1,2}-d{1,2} 13 一年的12个月(01～09和1～12)：^(0?[1-9]|1[0-2])$ 14 一个月的31天(01～09和1～31)：^((0?[1-9])|((1|2)[0-9])|30|31)$ 15 xml文件：^([a-zA-Z]+-?)+[a-zA-Z0-9]+.[x|X][m|M][l|L]$ 16 中文字符的正则表达式：[一-龥] 17 双字节字符：[^-ÿ] (包括汉字在内，可以用来计算字符串的长度(一个双字节字符长度计2，ASCII字符计1)) 18 空白行的正则表达式：s\* (可以用来删除空白行) 19 HTML标记的正则表达式： <(S\*?)[^>]\*>.\*?</>|<.\*? /> (网上流传的版本太糟糕，上面这个也仅仅能部分，对于复杂的嵌套标记依旧无能为力) 20 首尾空白字符的正则表达式：^s\*|s\*$或(^s\*)|(s\*$) (可以用来删除行首行尾的空白字符(包括空格、制表符、换页符等等)，非常有用的表达式) 21 腾讯QQ号：[1-9][0-9]{4,} (腾讯QQ号从10000开始) 22 中国邮政编码：[1-9]d{5}(?!d) (中国邮政编码为6位数字) 23 IP地址：d+.d+.d+.d+ (提取IP地址时有用) 24 IP地址：((?:(?:25[0-5]|2[0-4]d|[01]?d?d).){3}(?:25[0-5]|2[0-4]d|[01]?d?d)) 25 钱的输入格式： 1）.有四种钱的表示形式我们可以接受:"10000.00" 和 "10,000.00", 和没有 "分" 的 "10000" 和 "10,000"：^[1-9][0-9]\*$ 2）.这表示任意一个不以0开头的数字,但是,这也意味着一个字符"0"不通过,所以我们采用下面的形式：^(0|[1-9][0-9]\*)$ 3）.一个0或者一个不以0开头的数字.我们还可以允许开头有一个负号：^(0|-?[1-9][0-9]\*)$ 4）.这表示一个0或者一个可能为负的开头不为0的数字.让用户以0开头好了.把负号的也去掉,因为钱总不能是负的吧.下面我们要加的是说明可能的小数部分：^[0-9]+(.[0-9]+)?$ 5）.必须说明的是,小数点后面至少应该有1位数,所以"10."是不通过的,但是 "10" 和 "10.2" 是通过的：^[0-9]+(.[0-9]{2})?$ 6）.这样我们规定小数点后面必须有两位,如果你认为太苛刻了,可以这样： ^[0-9]+(.[0-9]{1,2})?$ 7）.这样就允许用户只写一位小数.下面我们该考虑数字中的逗号了,我们可以这样：^[0-9]{1,3}(,[0-9]{3})\*(.[0-9]{1,2})?$ 8）.1到3个数字,后面跟着任意个 逗号+3个数字,逗号成为可选,而不是必须： ^([0-9]+|[0-9]{1,3}(,[0-9]{3})\*)(.[0-9]{1,2})?$ 备注：这就是最终结果了,别忘了"+"可以用"\*"替代如果你觉得空字符串也可以接受的话(奇怪,为什么?)最后,别忘了在用函数时去掉去掉那个反斜杠,一般的错误都在这里