多行读入 不确定何时结束：

```
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
```

整体读入：

```
import sys
data = sys.stdin.read()   #读取整个输入流
print(data)
```

浮点数的输出：

```
小数：f"{value:.nf}"                                                    #其中 n 是保留小数位数
百分数  print(f"Percentage: {percentage:.2%}")                          # 输出：85.00%
科学计数法：  print(f"Scientific notation: {large_number:.2e}")      # 输出：1.23e+06
```

无空格输出

```
print(' '.join(map(str, x)))      #x 为列表或字符串
```

列表元素删除，例：my_list = [1, 2, 3, 4, 2, 5]

```
my_list.remove(2)          # 删除第一个 2
my_list.pop(2)             # 删除索引为 2 的元素  (即 3)
```

矩阵保护圈的写法

```
matrix = [[1] * (m + 2) for i in range(n + 2)]
for _ in range(1,n + 1):
    matrix[_][1:-1] = input()            #不带空格的输入
                      list(map(int, input().split()))    #带空格的输入
```

列表排序：

```
lst.sort(key=lambda x: x[0])
lst.sort(key=lambda x: (x[0], -x[1]))
```

多元运算

```
print("Yes" if flag else "No")
```

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)
```

字典的使用

```
# 创建字典
my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}

#通过键来搜索值
法一：print(my_dict['name'])  # 输出：Alice
法二：print(my_dict.get('name'))  # 输出：Alice
print(my_dict.get('address', 'Not Found'))  # 输出：Not Found

#通过值来搜索键
找到所有的键：
法一：keys = [key for key, value in my_dict.items() if value == search_value]
法二：keys = list(filter(lambda key: my_dict[key] == search_value, my_dict))
找到第一个符合条件的键：
key = next((key for key, value in my_dict.items() if value == search_value), None)

#添加或更新元素（键值对）：
my_dict['age'] = 26  # 更新
my_dict['country'] = 'USA'  # 添加

#向字典中某一个键下添加元素：
my_dict = {'key1': [1, 2, 3], 'key2': [4, 5]}
my_dict['key1'].append(4)
```

```
#删除键值对
法一：del my_dict['city']     # 删除 'city' 键值对
法二：age = my_dict.pop('age')
print(age)  # 输出：26

#遍历字典：
    # 遍历键
    for key in my_dict:

    # 遍历值
    for value in my_dict.values():
        print(value)

    # 遍历键值对
    for key, value in my_dict.items():
        print(f"{key}: {value}")


#字典推导式举例：
numbers = [1, 2, 3, 4, 5]
squared_dict = {n: n**2 for n in numbers}
print(squared_dict)

#字典排序：
sorted_dict = dict(sorted(my_dict.items(), key=lambda x: x[1], reverse=True))
```

ASCII 码表的使用：
chr() 数字转字符 ord() 字符转数字
注意：48 – 57 对应 0 – 9；65 – 90 对应 A – Z；97 – 122 对应 a – z。

排序中 Lambda 函数的使用（待补充）

```
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))
```

列表的深浅拷贝

```
浅拷贝：
import copy
shallow_copy = copy.copy(original)
深拷贝：
import copy
deep_copy = copy.deepcopy(original)
```

字典序比较大小

```
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))
```

函数打包（最大最小整数）

```
num = int(input())
lst = list(map(str,input().split()))
lst_copy = [item for item in lst]
max_len = len(max(lst,key=len))
for i in range(len(lst)):
    while len(lst[i]) <= max_len:
        lst[i] = lst[i] + lst[i]
zipped_lst = list(zip(lst,lst_copy))
zipped_lst_sorted1 = sorted(zipped_lst)
zipped_lst_sorted2 = sorted(zipped_lst, reverse=True)
sorted_lst1, sorted_lst_copy1 = zip(*zipped_lst_sorted1)
sorted_lst2, sorted_lst_copy2 = zip(*zipped_lst_sorted2)
sorted_lst_copy1 = list(sorted_lst_copy1)
sorted_lst_copy2 = list(sorted_lst_copy2)
min_num = "".join(sorted_lst_copy1)
max_num = "".join(sorted_lst_copy2)
print(max_num,min_num)
```

双指针：
用于二分查找

```
def binary_search(arr, target):
    left, right = 0, len(arr) - 1

    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid  # 返回目标元素的索引
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1  # 如果未找到目标元素，返回 -1
```

军备竞赛

```
p = int(input())
lst = list(map(int, input().split()))
num = 0
weapon = sorted(lst)
buy, sell = 0, len(lst)-1
while buy <= sell:
    if num < 0:
        num = 0
        break
    else:
        if p >= weapon[buy]:
            num += 1; p -= weapon[buy]; buy += 1
        else:
            if buy == sell:
                break
            else:
                num -= 1; p += weapon[sell]; sell -= 1
print(num)
```

最大长度回文子串

```
class Solution(object):
    def longestPalindrome(self, s):
        begin_with = 0; max_length = 1
        for i in range(len(s)):
            l = l; r = i + 1
            while l >= 0 and r < len(s) and s[l] == s[r]:
                l -= 1; r += 1
            if (r - l - 1) > max_length:
                max_length = r - l – 1; begin_with = l + 1
            l = l; r = i
            while l >= 0 and r < len(s) and s[l] == s[r]:
                l -= 1; r += 1
            if (r - l - 1) > max_length:
                max_length = r - l – 1; begin_with = l + 1
        return s[begin_with:begin_with + max_length]
```

螺旋矩阵

```
n = int(input())
matrix = [[0] * n for _ in range(n)]
directions = [[1,0],[0,1],[-1,0],[0,-1]]
d = 0
col,row = 0,0
for i in range(1,n ** 2 + 1):
    matrix[col][row] = i
    if row + directions[d][0] == n or col + directions[d][1] == n or matrix[col +
directions[d][1]][row + directions[d][0]] != 0:
        d = (d + 1) % 4
    row = row + directions[d][0]
    col = col + directions[d][1]
for i in range(n):
    print(" ".join(map(str, matrix[i])))
```

洋葱

```
n = int(input())
matrix = [list(map(int, input().split())) for i in range(n)]
ans = 0
directions = [[1,0],[0,1],[-1,0],[0,-1]]
d = 0
for i in range(n // 2):
    col,row = i,i
    cnt = 0
    step = 0
    while not(col == i and row == i and step > 0):
        cnt += matrix[col][row]
        if row + directions[d][0] == n - i or col + directions[d][1] == n - i or row +
directions[d][0] == i - 1 or col + directions[d][1] == i - 1:
            d = (d + 1) % 4
        row = row + directions[d][0]
        col = col + directions[d][1]
        step += 1
    ans = max(ans, cnt)
if n % 2 == 1:
    ans = max(ans, matrix[n // 2][n // 2])
print(ans)
```

最大连通域面积

```
s = 0
def dfs(a,b):
    directions = [[1,1],[1,0],[1,-1],[0,1],[0,-1],[-1,1],[-1,0],[-1,-1]]
    global s
    if a < 0 or b < 0 or a >= len(matrix) or b >= len(matrix[0]) or matrix[a][b] == ".":
        return
    matrix[a][b] = "."
    s += 1
    for i in range(len(directions)):
        dfs(a + directions[i][0],b + directions[i][1])
```

马走日（回溯）

```
ans = 0
def dfs(x,y,cnt):
    directions = [[1, 2], [2, 1], [1, -2], [2, -1], [-1, 2], [-2, 1], [-1, -2], [-2, -1]]
    global ans
    if cnt == n * m:
        ans += 1
        return
    for i in range(len(directions)):
        nx = x + directions[i][0]
        ny = y + directions[i][1]
        if 0 <= nx < n and 0 <= ny < m:
            if matrix[nx][ny] != 1:
                matrix[nx][ny] = 1
                dfs(nx, ny, cnt + 1)
                matrix[nx][ny] = 0
```

受到祝福的平方（一维 dfs）

```
from math import sqrt
def dfs(x):
    if x == len(A):
        return True
    n = 0
    for i in range(x,len(A)):
        n = n * 10 + A[i]
        if n != 0 and sqrt(n) % 1 == 0:
            if dfs(i + 1):
                return True
    return False
A = list(map(int,str(int(input()))))
if dfs(0):
    print("Yes")
else:
    print("No")
```

矩阵最大权值路径

```
maxValue = float("-inf")
def dfs(x, y, nowValue):
    directions = [[0, 1], [1, 0], [-1, 0], [0, -1]]
    global maxValue, maxValue_path
    if x == n and y == m:
        if nowValue > maxValue:
            maxValue = nowValue
            maxValue_path = nowValue_path[:]
        return
    for i in range(len(directions)):
        nx = x + directions[i][0]
        ny = y + directions[i][1]
        if matrix[nx][ny] != 9999:
            tmp = matrix[x][y]
            matrix[x][y] = 9999
            nextValue = nowValue + matrix[nx][ny]
            nowValue_path.append((nx, ny))
            dfs(nx, ny, nextValue)
            matrix[x][y] = tmp
            nowValue_path.pop()
n, m = map(int, input().split())
maxValue_path = []
nowValue_path = [(1, 1)]
matrix = [[9999] * (m + 2) for i in range(n + 2)]
for _ in range(1, n + 1):
    matrix[_][1:-1] = map(int, input().split())
dfs(1, 1, matrix[1][1])
for i in range(len(maxValue_path)):
    print(maxValue_path[i][0], maxValue_path[i][1])
```

水淹七军

```
import sys
sys.setrecursionlimit(1000000)
def dfs(matrix,matrix_,x,y):
    directions = [[0,1],[0,-1],[1,0],[-1,0]]
    for i in range(len(directions)):
        nx = x + directions[i][0]
        ny = y + directions[i][1]
        if 0 <= nx < m and 0 <= ny < n:
            if matrix[nx][ny] < matrix[x][y]:
                matrix[nx][ny] = matrix[x][y]
                matrix_[nx][ny] = True
                dfs(matrix,matrix_,nx,ny)
```

滑雪（dfs+dp）

```
def dfs(x, y):
    directions = [[1, 0], [0, 1], [-1, 0], [0, -1]]
    if dp[x][y] > 1:
        return dp[x][y]
    for i in range(len(directions)):
        nx = x + directions[i][0]
        ny = y + directions[i][1]
        if matrix[nx][ny] < matrix[x][y]:
            dp[x][y] = max(dp[x][y], dfs(nx, ny) + 1)
    return dp[x][y]

r,c = map(int,input().split())
matrix = [[10001] * (c + 2) for _ in range(r + 2)]
for i in range(1,r + 1):
    matrix[i][1:c + 1] = list(map(int, input().split()))
dp = [[1] * (c + 2) for _ in range(r + 2)]
ans = 0
for i in range(1,r + 1):
    for j in range(1,c + 1):
        ans = max(ans,dfs(i,j))
print(ans)
```

lake counting

```
def dfs(a,b):
    directions = [[1,1],[1,0],[1,-1],[0,1],[0,-1],[-1,1],[-1,0],[-1,-1]]
    if a < 0 or b < 0 or a >= len(matrix) or b >= len(matrix[0]) or matrix[a][b] == ".":
        return
    matrix[a][b] = "."
    for i in range(len(directions)):
        dfs(a + directions[i][0],b + directions[i][1])

ans = 0
n,m = map(int,input().split())
matrix = [["."] * (m + 2) for i in range(n + 2)]
for _ in range(1,n + 1):
    matrix[_][1:-1] = input()
for i in range(1,n + 1):
    for j in range(1,m + 1):
        if matrix[i][j] == "W":
            dfs(i,j)
            ans += 1
print(ans)
```

bfs

寻宝：

```
from collections import deque
def bfs(x,y):
    directions = [[0, 1], [1, 0], [-1, 0], [0, -1]]
    q = deque([(0,(x,y))])
    in_queue = {(x,y)}
    while q:
        step,(x,y) = q.popleft()
        if matrix[x][y] == 1:
            return step
        for i in range(len(directions)):
            nx = x + directions[i][0]
            ny = y + directions[i][1]
            if matrix[nx][ny] != 2 and (nx,ny) not in in_queue:
                in_queue.add((nx,ny))
                q.append((step + 1,(nx,ny)))
    return "NO"
m,n = map(int,input().split())
matrix = [[2] * (n + 2) for i in range(m + 2)]
for _ in range(1,m + 1):
    matrix[_][1:-1] = map(int,input().split())
print(bfs(1,1))
```

变换的迷宫

```
from collections import deque
def bfs(a,b,m,n):
    directions = [[1,0],[0,1],[-1,0],[0,-1]]
    q = deque([(0,a,b)])
    in_queue = {(0,a,b)}
    while q:
        time,x,y = q.popleft()
        if x == m and y == n:
            return time
        for i in range(len(directions)):
            nx = x + directions[i][0]
            ny = y + directions[i][1]
            t = (time + 1) % k
            if 0 <= nx < r and 0 <= ny < c and (t,nx,ny) not in in_queue:
                if t == 0 or matrix[nx][ny] != "#":
                    q.append((time + 1,nx,ny))
                    in_queue.add((t,nx,ny))
    return "Oop!"
```

小游戏

```
from collections import deque
def bfs(m,n,a,b):
    ans = []
    directions = [[0,1], [1,0], [-1,0], [0,-1]]
    q = deque([(-1,0,(m,n))])
    in_queue = {(-1,(m,n))}
    while q:
        d,s,(x,y) = q.popleft()
        if x == a and y == b:
            ans.append(s)
            break
        for i in range(len(directions)):
            nx = x + directions[i][0]
            ny = y + directions[i][1]
            if 0 <= nx < h + 2 and 0 <= ny < w + 2 and ((i, nx, ny) not in in_queue):
                nd = i
                if nd != d:
                    ns = s + 1
                else:
                    ns = s
                if nx == a and ny == b:
                    ans.append(ns)
                if matrix[nx][ny] != "X":
                    in_queue.add((nd, nx, ny))
                    q.append((nd, ns, (nx, ny)))
    if len(ans):
        return str(min(ans)) + " segments."
    else:
        return "impossible."
```

孤岛最短距离

```
import heapq
def dijkstra(a,b):
    directions = [[0,1],[1,0],[-1,0],[0,-1]]
    q = []
    visited = [[False] * len(matrix[0]) for _ in range(n)]
    heapq.heappush(q,(0,a,b))
    while q:
        step,x,y = heapq.heappop(q)
        if visited[x][y]:
            continue
        visited[x][y] = True
        if matrix[x][y] == 1 and step > 0:
            return step
        for i in range(len(directions)):
            nx = x + directions[i][0]
            ny = y + directions[i][1]
            if 0 <= nx < n and 0 <= ny < len(matrix[0]) and not visited[nx][ny]:
                heapq.heappush(q,(step + 1 - matrix[nx][ny],nx,ny))
n = int(input())
matrix = [list(map(int,input())) for _ in range(n)]
for i in range(n):
    for j in range(len(matrix[0])):
        if matrix[i][j] == 1:
            a,b = i,j
print(dijkstra(a,b))
```

走山路

```
import heapq
def dijkstra(a,b,u,v):
    directions = [[0, 1], [1, 0], [-1, 0], [0, -1]]
    q = []
    dic = {(a,b):0}
    heapq.heappush(q, (0, a, b))
    while q:
        step, x, y = heapq.heappop(q)
        if x == u and y == v:
            return step
        for i in range(len(directions)):
            nx = x + directions[i][0]
            ny = y + directions[i][1]
            if 0 <= nx < m and 0 <= ny < n and matrix[nx][ny] != "#":
                new_step = step + abs(int(matrix[nx][ny]) - int(matrix[x][y]))
                if (nx,ny) not in dic or new_step < dic[(nx,ny)]:
                    dic[(nx,ny)] = new_step
                    heapq.heappush(q, (new_step, nx, ny))
    return "NO"
```

dp

小偷背包

```
n,b=map(int, input().split())
price=[int(i) for i in input().split()]   #（注意这里下标从零开始）
weight=[int(i) for i in input().split()]
bag=[[0]*(b+1) for _ in range(n+1)]
for i in range(1,n+1):
    for j in range(1,b+1):
        if weight[i]<=j:
            bag[i][j]=max(price[i]+bag[i-1][j-weight[i]], bag[i-1][j])
        else:
            bag[i][j]=bag[i-1][j]
print(bag[-1][-1])
```

完全背包

```
n, a, b, c = map(int, input().split())
dp = [float('-inf')]*n
for i in range(1, n+1):
    for j in (a, b, c):
        if i >= j:
            dp[i] = max(dp[i-j] + 1, dp[i])
print(dp[n])
```

土豪购物（双 dp）

```
lst = list(map(int, input().split(",")))
dp1 = [0] * len(lst)
dp2 = [0] * len(lst)
dp1[0],dp2[0] = lst[0],lst[0]
for i in range(1,len(lst)):
    dp1[i] = max(dp1[i - 1] + lst[i],lst[i])
    dp2[i] = max(dp1[i - 1],dp2[i - 1] + lst[i],lst[i])
print(max(max(dp1),max(dp2))
```

最大整数

```
m = int(input())
n = int(input())
lst = list(map(str,input().split()))
lst.sort(key = lambda x: x * 10, reverse = True)
dp = [0] * (m + 1)
for num in lst:
    for i in range(m, len(num) - 1,-1):
        dp[i] = max(dp[i], dp[i - len(num)] * (10 ** len(num)) + int(num))
print(dp[m])
```

拦截导弹

```
num = int(input())
lst = list(map(int, input().split()))
dp_lst = [1] * num

for i in range(1,num):
    for j in range(i):
        if lst[j] >= lst[i]:
            dp_lst[i] = max(dp_lst[i],dp_lst[j]+1)

print(max(dp_lst))
```

最长上升子序列

```
def length_of_lis(nums):
    if not nums:
        return 0
    dp = [1] * len(nums)
    for i in range(1, len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j] + 1)
    return max(dp)
nums = [10, 9, 2, 5, 3, 7, 101, 18]
print(length_of_lis(nums)) #  输出: 4
```

核电站（N 个坑，不能有连续 M 个的方案总数）

```
n, m = map(int, input().split())
DP = [0] * 60
DP[0] = 1       #DP[i]是第 i 个位置的方案数。

for i in range(1, n + 1):
    if i < m:  #达不到连续放置 m 个的情况
        DP[i] = DP[i - 1] * 2     # 从第 1 个到第 m-1 个，方案都可以选择放/不放
    elif i == m:     #第 m 个要小心了
        DP[i] = DP[i - 1] * 2 - 1
    else: #i>m
        DP[i] = DP[i - 1] * 2 - DP[i - m - 1]
print(DP[n])
```

约瑟夫问题

```
while True:
    n,m= map(int,input().split())
    if n == 0 and m == 0:
        break
    else:
        lst = []
        for i in range(1,n+1):
            lst.append(i)
        index = 0
        while len(lst) > 1:
            index = (index + m - 1) % len(lst)
            lst.pop(index)
        print(lst[0])
```

全排列

```
def dfs(n, path, used, res):
    if len(path) == n:
        res.append(path[:])
        return
    for i in range(1, n+1):
        if not used[i]:
            used[i] = True
            path.append(i)
            dfs(n, path, used, res)
            path.pop()
            used[i] = False

def print_permutations(n):
    res = []
    dfs(n, [], [False]*(n+1), res)
    for perm in sorted(res):
        print(' '.join(map(str, perm)))

nums = []
while True:
    num = int(input())
    if num == 0:
        break
    nums.append(num)

for num in nums:
    print_permutations(num)
```

```
perms = itertools.permutations(arr, 2(c这里是长度))

for perm in perms:
    print(perm)itertools.permutations(iterable, r=None)
```

冒泡排序

```
for i in range(n):
    for j in range(n-1-i):
        if l[j] + l[j+1] > l[j+1] + l[j]:
            l[j],l[j+1] = l[j+1],l[j]
```

滑动窗口

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
```

字典序大小比较

```
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))
```

栈的内容

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
```

函数模块

math

math.ceil 上取整 math.floor 下取整

math.sqrt 根号 math.factorial 阶乘

遍历

import itertools

for item in itertools.product('AB', repeat=2): 、

print(item) # 输出: ('A', 'A'), ('A', 'B'), ('B', 'A'), ('B', 'B')

下一个全排列

```
def next_permutation(nums):
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i + 1]:
        i -= 1
    if i >= 0:
        j = len(nums) - 1
        while nums[j] <= nums[i]:
            j -= 1
        nums[i], nums[j] = nums[j], nums[i]
        nums[i + 1:] = reversed(nums[i + 1:])
        return nums
nums = [1, 2, 3]
print(next_permutation(nums)) # 输出: [1, 3, 2]
```

求解素数的方法

```
朴素法（时间复杂度是O(N^2)）
primesNumber = []
def is_prime(n):
    for i in range(2, n-1):
        if n % i == 0:
            return False
    return True
def primes(number):
    for i in range(2, number):
        if is_prime(i):
            primesNumber.append(i)
primes(10000)
print(primesNumber)
```

```
埃氏筛
def sieve_of_eratosthenes(n):
    # 创建一个布尔列表，初始化为 True，表示所有数字都假设为素数
    primes = [True] * (n + 1)
    primes[0] = primes[1] = False  # 0 和 1 不是素数

    # 从 2 开始，处理每个数字
    for i in range(2, int(n**0.5) + 1):
        if primes[i]:  # 如果 i 是素数
            # 将 i 的所有倍数标记为非素数
            for j in range(i * i, n + 1, i):
                primes[j] = False

    # 返回所有素数
    return [x for x in range(2, n + 1) if primes[x]]

欧拉筛
# 返回小于r的素数列表
def oula(r):
    # 全部初始化为0
    prime = [0 for i in range(r+1)]
    # 存放素数
    common = []
    for i in range(2, r+1):
        if prime[i] == 0:
            common.append(i)
        for j in common:
            if i*j > r:
                break
            prime[i*j] = 1
            #将重复筛选剔除
            if i % j == 0:
                break
    return common
prime = oula(20000)
print(prime)
```