```
def palindrome_string(words):
    for word in words:
        if word==word[::-1]:
            return word
    return ""
words1=["abc","car","ada","racecar","cool"]
print(palindrome_string(words1))

def count_indices(nums1, nums2):
    answer1=sum(1 for num in nums1 if num in nums2)
    answer2=sum(1 for num in nums2 if num in nums1)
    return [answer1,answer2]
nums1_example1=[2,3,2]
nums2_example1=[1,2]
nums1_example2=[4,3,2,3,1]
nums2_example2=[2,2,5,2,3,6]
print(count_indices(nums1_example1, nums2_example1))
print(count_indices(nums1_example2, nums2_example2))

def sum_of_squares_of_distinct_counts(nums):
    n=len(nums)
    result=0
    for i in range(n):
        distinct=set()
        for j in range(i,n):
            distinct.add(nums[j])
            result+=len(distinct)**2
    return result
nums1=[1,2,1]
nums2=[1,1]
```

```
print(sum_of_squares_of_distinct_counts(nums1))
print(sum_of_squares_of_distinct_counts(nums2))

def count_pairs(nums,k):
    count=0
    n=len(nums)
    for i in range(n):
        for j in range(i+1,n):
            if nums[i]==nums[j] and (i*j)%k==0:
                count+=1
    return count
nums1=[3,1,2,2,2,1,3]
k1=2
nums2=[1,2,3,4]
k2=1
print(count_pairs(nums1,k1))
print(count_pairs(nums2,k2))

def find_max(nums):
    if not nums:
        return None
    return max(nums)
print(find_max([1,2,3,4,5]))
print(find_max([-10,2,3,-4,5,9,12]))

def sort_and_find_max(nums):
    if not nums:
        return None
    nums.sort()
    return nums[-1]
```

```
print(sort_and_find_max([]))
print(sort_and_find_max([5]))
print(sort_and_find_max([3,3,3,3,3]))

def unique_elements(nums):
    return list(set(nums))
print(unique_elements([3,7,3,5,2,5,9,2]))
print(unique_elements([-1,2,-1,3,2,-2]))

def bubble_sort(arr):
    n=len(arr)
    for i in range(n):
        for j in range(0,n-i-1):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
    return arr
arr=[64,34,25,12,22,11,90]
print(bubble_sort(arr))

def binary_search(arr,x):
    low,high=0,len(arr)
    while low<=high:
        mid=(low+high)//2
        if arr[mid]==x:
            return mid
        elif arr[mid]<x:
            low=mid+1
        else:
            high=mid-1
    return -1
```

```
arr1=[3,4,6,-9,10,8,9,30]
key1=10
print(binary_search(arr1,key1))

def merge_sort(arr):
    if len(arr)>1:
        mid=len(arr)//2
        left_half=arr[:mid]
        right_half=arr[mid:]
        merge_sort(left_half)
        merge_sort(right_half)
        i=j=k=0
        while i<len(left_half) and j<len(right_half):
            if left_half[i]<right_half[j]:
                arr[k]=left_half[i]
                i+=1
            else:
                arr[k]=right_half[j]
                j+=1
            k+=1
        while i<len(left_half):
            arr[k]=left_half[i]
            i+=1
            k+=1
        while j<len(right_half):
            arr[k]=right_half[j]
            j+=1
            k+=1
    return arr
arr=[12,11,13,5,6,7]
```

```
print(merge_sort(arr))


def find_paths(m,n,N,i,j):
    memo={}
    def dp(x,y,remaining_steps):
        if x<0 or x>=m or y<0 or y>=n:
            return 1
        if remaining_steps==0:
            return 0
        if (x,y,remaining_steps) in memo:
            return memo[(x,y,remaining_steps)]
        ways = (dp(x+1,y,remaining_steps - 1) +
                dp(x-1,y,remaining_steps-1)+
                dp(x,y+1,remaining_steps-1)+
                dp(x,y-1,remaining_steps-1))
        memo[(x,y,remaining_steps)]=ways
        return ways
    return dp(i,j,N)
print(find_paths(2,2,2,0,0))


def rob(nums):
    def rob_linear(houses):
        prev,curr=0,0
        for money in houses:
            prev,curr=curr,max(curr,prev+money)
        return curr
    if len(nums)==1:
        return nums[0]
    return max(rob_linear(nums[1:]),rob_linear(nums[:-1]))
print(rob([2,3,2]))
```

```
def climbStairs(n):
    if n==1:
        return 1
    dp=[0]*(n+1)
    dp[1],dp[2]=1,2
    for i in range(3,n+1):
        dp[i]=dp[i-1]+dp[i-2]
    return dp[n]
print(climbStairs(4))

def uniquePaths(m,n):
    dp=[[1]*n for _ in range(m)]
    for i in range(1,m):
        for j in range(1,n):
            dp[i][j]=dp[i-1][j]+dp[i][j-1]
    return dp[m-1][n-1]
print(uniquePaths(7,3))

def largeGroupPositions(s):
    result=[]
    i=0
    while i<len(s):
        start=i
        while i<len(s) and s[i]==s[start]:
            i+=1
        if i-start>=3:
            result.append([start,i-1])
    return result
print(largeGroupPositions("abbxxxxzzy"))
```

```
def gameOfLife(board):
    rows,cols=len(board),len(board[0])
    directions=[(-1,-1),(-1,0),(-1,1),(0,-1),(0,1),(1,-1),(1,0),(1,1)]
    def count_live_neighbors(r,c):
        live_neighbors=0
        for dr,dc in directions:
            nr,nc=r+dr,c+dc
            if 0<=nr<rows and 0<=nc<cols and abs(board[nr][nc])==1:
                live_neighbors+=1
        return live_neighbors
    for r in range(rows):
        for c in range(cols):
            live_neighbors=count_live_neighbors(r,c)
            if board[r][c]==1 and (live_neighbors<2 or live_neighbors>3):
                board[r][c]=-1
            if board[r][c]==0 and live_neighbors==3:
                board[r][c]=2
    for r in range(rows):
        for c in range(cols):
            if board[r][c]>0:
                board[r][c]=1
            else:
                board[r][c]=0
    return board
print(gameOfLife([[0,1,0],[0,0,1],[1,1,1],[0,0,0]]))
print(gameOfLife([[1,1],[1,0]]))

def champagneTower(poured,query_row,query_glass):
    tower=[[0]*k for k in range(1,102)]
    tower[0][0]=poured
    for r in range(query_row+1):
        for c in range(r+1):
            excess=(tower[r][c]-1.0)/2.0
            if excess>0:
                tower[r+1][c]+=excess
                tower[r+1][c+1]+=excess
    return min(1,tower[query_row][query_glass])
print(champagneTower(1,1,1))
print(champagneTower(2,1,1))
```