First missing minimum positive

Method-1

def firstmisssingpositive(nums):

n=len(nums)

for I in range(n):

while 1 <= nums[i] <= n and nums[nums[i]-1] != nums[i]:

nums[nums[i]-1],nums[i]=nums[i],nums[nums[i]-1]

for I in range(n):

if nums[i] !=i+1 :

return i+1

return n+1

Method-2

from collections import counter

def firstmissingpositive(nums):

c=counter(nums)

for I in range(len(nums)):

if i+1 not in c:

return i+1

return len(nums)+1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

count pairs whose sum is less than target

def countpairs(nums,target):

nums.sort()

count=0

left=0

right=len(nums)-1

while (left < right):

if (nums[left]+nums[right] < target):

count+=right-left

left+=1

else: right-=1

return count

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

happy number

def ishappy(n):

already=[]

l=list(str(n))

if n==1: return true

sum1=0

while sum1 !=1:

for item in l:

sum1+=(int(item)\*\*2)

if sum1==1:

return true

else:

if sum1 in already:

return false

already.append(sum1)

l=list(str(sum1))

sum1=0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

finding duplicates

Method-1 : using set

def find duplicate(nums):

seen=set()

for num in nums:

if num in seen:

return num

seen.add(num)

Method-2 : using count array

def find duplicate(nums):

cnt=[0]\*(len(nums)+1)

for num in nums:

cnt[num]+=1

if cnt[num]>1:

return num

return len(nums)

Method-3 : marking in array approach

def find duplicate(nums):

for num in nums:

idx=abs(num)

if nums[idx]<0:

return idx

nums[idx]=-nums[idx]

return len(nums)

Method-4 : binary search

def find duplicate(nums):

for I in range(1,len(nums)):

if nums[i]==nums[i+1]:

return nums[i]

return len(nums)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Merge two 2D arrays by summing values

Def mergearrays2(nums1,nums2):

 d1,d2=defaultdict(int,{a:b for a,b in nums1}),defaultdict(int,{a:b for a,b in nums2})

        return [[k,d1[k]+d2[k]]for k in sorted(set(d1.keys()).union(set(d2.keys())))]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

find the array concatenation value

rdef findthearrayconcval(nums):

left=0

right=len(nums)-1

total=0

while left<=right:

if left<right:

total+=int(str(nums[left])+str(nums[right]))

else:

total+=nums[left]

left+=1

right-=1

return total

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Largest +ve int that exists with its -ve

Def findmaxk(nums):

L,r=0,len(nums)-1

Nums.sort()

While l<r:

If -nums[l]==nums[r]:

Return nums[r]

Elif -nums[l]>nums[r]:

L+=1

Else:

r-=1

return -1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Maximum matching of players with trainers

Def matchplayersandtrainers(players,trainers):

Players.sort()

Trainers.sort()

Res=0

I,j=0,0

While i<len(players) and j<len(trainers):

If players[i]<=trainers[j]:

Res+=1

I+=1

J+=1

Retuen res

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Relative ranks

From collections import defaultdict

Def findrelativeranks(score):

D=defaultdict(int)

Place=[“gold medal”,”silver medal”,”bronze medal”]

N=len(score)

A=[“ “]\*n

For I in range(n):d[score[i]]=i

Score.sort(reverse=true)

For I in range(n):

If i<3:

A[d[score[i]]]=place[i]

Else:

A[d[score[i]]]=str(i+1)

Return a

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Array partition

Def arraypairsum(nums):

Nums.sort()

Count=0

For I in range(0,len(nums),2):

Count+=nums[i]

Return count

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Def findlhs(nums):

Num\_counts={}

For num in nums:

Num\_counts[num]=num\_counts.get(num,0)+1

Max\_length=0

For num in num\_counts:

If num+1 in num\_counts:

Max\_length=max(max\_length,num\_counts[num]+num\_counts[num+1])

Return max\_length

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Fair candy swap

Def faircandyswap(alicesizes,bobsizes):

Totalalice=sum(alicesizes)

Totalbob=sum(bobsizes)

Targettotal=(totalalice+totalbob)//2

For alicecandy in alicesizes:

Bobcandy=alicecandy+(targettotal-totalalice)

If bobcandy in bobsizes:

Return[alicecandy,bobcandy]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Sort array by parity

Def sortarraybyparity(nums):  
 left,right=0,len(nums)-1

While left<right:

While left<right and nums[left]%2==0:

Left+=1

While left<right and nums[right]%2==1:

Right-=1

Nums[left],nums[right]=nums[right],nums[left]

Return nums

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Find all duplicates in an array

Def find\_duplicates(arr):

Duplicates=[]

N=len(arr)

For I in range(n):

Index=arr[i]%n

Arr[index]+=n

For I in range(n):  
 if arr[i]//n>=2:

Duplicates.apppend(i)

Return duplicates

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Find all missing numbers from a given sorted array

Def printmissingelements(arr,N):

Cnt=0

For I in range(arr[0],arr[N-1]+1):

If (arr[cnt]==i):

Cnt+=1

Else:

Print(I,end=” “)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_