In [1]:
```
#two sum
def two_sum(nums,target):
    for i in range(len(nums)):
        for j in range(i+1,len(nums)):
            if nums[i]+nums[j]==target:
                return [i,j]
two_sum([1,2,3],4)
```

Out[1]:
```
[0, 2]
```

In [ ]:
```

for i in range(4,-1,-1):
    i+1
print i
```

In [8]:
```
#roman to integer
roman = {'I':1,'V':5,'X':10,'L':50,'C':100,'D':500,'M':1000}
def romanToInt(S):
    summ= 0
    for i in range(len(S)-1,-1,-1): #len(S)-1,...,0 大--小
        num = roman[S[i]]
        if 3*num < summ: #都是从大到小排列，一旦前比后小，就要━。
            #用3*：从4开始写法变了 e.g. print iv=4,i*3=3<5--v-i=4
            summ = summ-num
        else:
            summ = summ+num
    return summ
romanToInt("MCMXCIV")
```

Out[8]:
```
1994
```

In [53]:
```
strs = ["fliwer","flae","flight"]
strs.sort()
strs
```

Out[53]:
```
['flae', 'flight', 'fliwer']
```

In [52]:
```
#longest common prefix
def longest_common_pre(strs):
    size=len(strs) #length of strs. size=3,index=0,1,2
    if (size==0):
        return "" #return empty
    if (size==1): #one string
        return strs[0] #return the only one string
    strs.sort() #sort 从小到大--已经有common prefix了,后
    #两个类似，只要看1st和last就能保证全部过一遍common
    end=min(len(strs[0]),len(strs[size-1])) #end=minimum of
    #(1st string's length,last string's length)
    i=0 #initial i=0
    while (i<end and strs[0][i]==strs[size-1][i]): #while i=0<6
        #and 1st string的[i]个字母=last string的[i]个字母
        i=i+1 #update i+1
    pre=strs[0][0:i] # return： 1st string的[0]-[i-1]个字母
    return pre

longest_common_pre(["flower","flight","flow"])
```

Out[52]:
```
'fl'
```

In [113]:
```
#merge two sorted list
#method 1
def merge_two(list1,list2):
    res=sorted(list1+list2)
    return res
merge_two([1,2,4],[1,3,4])
```

Out[113]:
```
[1, 1, 2, 3, 4, 4]
```

In [14]:
```
def merge_two1(list1,list2):
    m=len(list1)
    n=len(list2)
    res=[] #output list
    i=0 #for index 0开始
    j=0
    while i<m and j<n:#index <length
        if list1[i]<list2[j]: #ith of list 1 < jth of list2
            res.append(list1[i])  #append ith of list1(append small
            i=i+1 #i 往下推进
        else:
            res.append(list2[j])
            j=j+1
    res=res+list1[i:]+list2[j:] #加上list1和list2里剩下的
    return res
merge_two1([1,2,4],[1,3,4])
```

Out[14]:
```
[1, 1, 2, 3, 4, 4]
```

In [112]:
```
#use stacking--each sorted list as a stack
def merge_two1(list1,list2):
    sortedlist=[] #creat empty list,用来return
    while list1 and list2: #只要list1和list2还有数, keep doing followi
        #不断加进smaller的数
        if list1[0]<list2[0]: #如果list1[0]smaller
            sortedlist.append(list1.pop(0)) #append list1里index=0的
        else:
            sortedlist.append(list2.pop(0)) #如果list2[0]smaller
            #or equal--append list2里index=0的数 (e.g. 2)
    sortedlist=sortedlist+list1 #append剩下的数
    sortedlist=sortedlist+list2
    return sortedlist
merge_two1([1,2,4],[1,3,4])
```

Out[112]:
```
[1, 1, 2, 3, 4, 4]
```

In [62]:
```
par={'(':')','[':']','{':'}'}
par.keys()
```

Out[62]:
```
dict_keys(['(', '[', '{'])
```

In [75]:
```
list=[')','}']
list.index(')')
```

Out[75]:
```
0
```

In [123]:
```
list=[1,2,4]
list.pop(0) #return去掉的那个数: 一个number
#list # after the pop, 去掉了index=0的数
```

Out[123]:
```
1
```

In [131]:
```
#remove duplicates from sorted array
def removeDuplicates(nums):
    size = len(nums) #size=length of nums list
    a = 1 #set a=1
    for i in range(1, size): #i =1,2,..,length-1
        # Found unique element
        if nums[i - 1] != nums[i]:   # if nums里前一个和后一个不相等
            #e.g. i=2,nums[1]!=nums[2]
                # Updating a in our main array
            nums[a] = nums[i]  # e.g. nums[a=1]=nums[2]=2
                # Incrementing a count by 1,有不等的就+1个, 相等的=dupl
            a = a + 1       #e.g.a=1+1=2..
    return a #a个distinct values
removeDuplicates([1,1,2,3,4])
```

Out[131]:
```
4
```

In [132]:
```
#plus one
def plusone(digits):
    num=0 #set num=0
    for i in range(len(digits)): # i in range(length of digits list
        num+=digits[i]*pow(10,(len(digits)-1-i)) #update num by:
        #ith index of digits * 10^length-1-i
        #e.g. num=1*10^3+2*10^2+3*10^1+4*10^0
    return [int(i) for i in str(num+1)] #for i in str(num+1)--
#return int(i)
     #i in str(1235):i='1','2','3','5'--int(i):change to int--
    #return as list
plusone([1,2,3,4])
```

Out[132]:
```
[1, 2, 3, 5]
```

In [137]:
```
#sqrt(x)
def sqrt_x(x):
    l=0 #min
    h=x #the input int=8 #max
    m=(l+h)>>1# >>1:divided by 2^1,m=4, center值
    while l<=h: #while 0<=8,4*4=16>x=8; 0<=3,1*1=1<8; 2<=3,2*2=4<=8
        #3<=3,3*3=9>8; 3<=2? false--out, 当min小于等于max
        if m*m==x: #m就是square root
            return m
        elif m*m>x: #m大了--max-1--max移到m左边
            h=m-1 #h=4-1=3; new_h=3-1=2
        else: #if m*m<x--m小了, min+1
            l=m+1 #new_l=1+1=2; new_l=2+1=3;
        m=(l+h)>>1 #new_m=(0+3(new_h))/2=1, only return integer
        #part; --update medium值
                    #new_m=(2+3)/2=2
                    #new_m=(3+3)/2=3
                    #new_m=(3+2)/2=2
    return m #m=2
sqrt_x(8)
```

Out[137]:
```
2
```

In [150]:
```
#climbing stairs
 #number of ways to reach step n=number of ways to reach n-1 +
    #number of ways to reach n-2
def climb_ways(n):
    if n<=1:
        return 1 #n=1,only 1 way
    # now consider n=2,3,...
    k=0 #number of ways to reach n, initialize =0
    prev_prev=1  #number of ways to reach n-2--initialize=1,
    #reach n=0 ways
    prev=1 #number of ways to reach n-1--initialize=1,reach n=1 way

    for i in range(2,n+1): #i=2,3--算到n,return k
        k=prev+prev_prev #k=1+1=2; k=3 ---- update k by summing
        #these two
        prev_prev=prev #prev_prev=prev=1; prev_prev=2 ----
        #update reach n-2 ways= reach n-1 ways b/c n increase 1
        prev=k #prev=2; prev=3---n-1 now become n
    return k #k=3
climb_ways(3)
```

Out[150]:
```
3
```

In [12]:
```
#merge two sorted array
def merge(nums1,m,nums2,n):
    pointer=len(nums1)-1 #track position to copy the larger
    #element into : for nums1
    m=m-1 #index for nums1
    n=n-1   #index for nums2   index=length-1
    while m>=0 and n>=0:
        if nums1[m]>=nums2[n]: # mth of nums1 >= nth of nums
            #2, 最后一个对比
            nums1[pointer]=nums1[m] #把mth放到nums1最后
            m=m-1 #update m-1, 下一个往前推进
        else: #if <
            nums1[pointer]=nums2[n] #把nums2的最后一个放到
            #nums1的最后一个
            n=n-1 #update n-1, 下一个往前推进
        pointer=pointer-1 #pointer-1, 往前推下一个
    if n>=0: #copy remaining in nums2  (如果nums2里还有)
        nums1[:n+1]=nums2[:n+1]  #把nums2从0th到nth的数
        #加到 nums1的0th-nth
    return nums1
merge([1,2,3,0,0,0],3,[2,5,6],3)
```

Out[12]:
```
[1, 2, 2, 3, 5, 6]
```

In [19]:
```
#binary tree inorder traversal, 从小到大
# left subtree(if subroot有小root:subroot's left subtree--
#subroot--right subtree)--大root--right subtree(if..)
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None

def inorderTraversal(root):
    answer = []

    inorderTraversalUtil(root, answer)
    return answer

def inorderTraversalUtil(root, answer):

    if root is None:
        return

    inorderTraversalUtil(root.left, answer)
    answer.append(root.val)
    inorderTraversalUtil(root.right, answer)
    return

root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
root.left.left = TreeNode(4)
root.left.right = TreeNode(5)

print(inorderTraversal(root))
```

```
[4, 2, 5, 1, 3]
```

In [24]:
```
#fizzbuzz
def fizzbuzz(n):
    res=[]
    for i in range(1,n+1):
        if i%3==0 and i%5==0:
            res.append("FizzBuzz")
        elif i%3==0:
            res.append("Fizz")
        elif i%5==0:
            res.append("Buzz")
        else:
            res.append(str(i))
    return res
fizzbuzz(5)
```

Out[24]:
```
['1', '2', 'Fizz', '4', 'Buzz']
```

In [26]:
```
15//10
```
Out[26]:
```
1
```

In [32]:
```
x=3200
str(x).rstrip('0') #remove trailing: remove end numbers
```
Out[32]:
```
'32'
```

In [74]:
```
#reverse an integer
def reverse(x):
    assert -2**31 <= x <= 2**31 - 1, 'Invalid integer range'
    #assertion,if no--output "invalid.."
    if x<0: #if negative--change to positive first
        x_1=x*-1
    else: #if positive--just positive
        x_1=x
    x_1=int(str(x_1).rstrip('0')) #remove trailing characters
    x_reversed=0 #set 0 first

    while(x_1 > 0):
        a = x_1 % 10 # mod--last digit
        x_reversed = x_reversed * 10 + a # +last digit--so
        #1*10+2=12--12*10+3=123 前往后
        x_1 = x_1 // 10 # 整除--去掉最后一位的剩下的数--位数不断往前进: 3位
    if x<0:
        return -x_reversed
    return x_reversed
reverse(3210)
```
Out[74]:
```
123
```

In [87]:
```
#find the 1st unique character in a string
def unique(s):
    s=s.lower() #lowercase all, since A and a 也算repeat
    counts={} #create dictionary store character and their counts
    for i in s: #add values to counts---是按顺序add的, 所以最后的
        #return就是1st unique
        if i not in counts: #1.character:1
            counts[i]=1
        else: #if repeat--character:1+1
            counts[i]+=1
    for j in range(len(s)): # in string s
        if counts[s[j]]==1: # select counts=1 的character--
            #它的index=j
            return j #符合条件, 一旦执行return, function结束, 不会继续for

unique('appsilon polland') #1st unique = 's' with index 3
```
Out[87]:
```
3
```

```
In [90]: #defanging an ip address
         def defang(address):
             address_split=address.split('.') #split by .--convert to list
             return '[.]'.join(address_split) #join by [.] between each str
         #character into string
         defang('127.0.0.1')

Out[90]: '127[.]0[.]0[.]1'
```

```
In [93]: s='123.4.5'
         s=s.split('.') #按'.'隔开成几个str character 放到一个list里
         s

Out[93]: ['123', '4', '5']
```

```
In [92]: '[.]'.join(s) #用[.]join起来变成string

Out[92]: '123[.]4[.]5'
```

```
In [101]: #check if a string is an anagram--重新排序
          def ifanagram(a,b):
              a=a.lower()
              b=b.lower() #lowercase--since A and a mean same thing
              if len(a)!=len(b): #they mush have same length
                  return False
              counts_a={} #create emtpy dictionary for two strings:
              #store character and their counts
              counts_b={}
              for i in a: #append elements to counts_a
                  if i not in counts_a:
                      counts_a[i]=1
                  else:
                      counts_a[i]+=1 #if repeat, +1
              for j in b:
                  if j not in counts_b:
                      counts_b[j]=1
                  else:
                      counts_b[j]+=1
              if counts_a==counts_b: #compare two dictionaries
                  return True
              else:
                  return False
          ifanagram('anagram','nagaram')

Out[101]: True
```

```
In [102]: dic1={'a':3,'m':1,'b':2}
          dic2={'b':2,'a':3,'m':1}
          dic1==dic2 #order does not matter

Out[102]: True
```

In [120]:
```
#check if a string is a palindrome
import string
def palindrome(s):
    s=s.lower()
    allowed=[*string.ascii_lowercase,*string.digits]
    s_fixed='' #for (new) fixed version
    for i in s: #remove non-alphanumeric 去掉非字母数字的,s可能含有这些
        if i in allowed:
            s_fixed=s_fixed+i #add element to s_fixed(from original
    s_reversed='' #for reverse version
    for i in s_fixed: #把s_fixed的reverse
        s_reversed=i+s_reversed #update s_reversed, 不断把i加到前面---
    if s_fixed==s_reversed: #same or not
        return True
    else:
        return False
palindrome('Bob')
```

Out[120]:
```
True
```

In [121]:
```
import string
s='abdeSEFD'
a=[*string.ascii_lowercase, *string.digits]
a #all lowercase letter a-z and digits 0-9
```

Out[121]:
```
['a',
 'b',
 'c',
 'd',
 'e',
 'f',
 'g',
 'h',
 'i',
 'j',
 'k',
 'l',
 'm',
 'n',
 'o',
 'p',
 'q',
 'r',
 's',
 't',
 'u',
 'v',
 'w',
 'x',
 'y',
 'z',
 '0',
 '1',
 '2',
 '3',
 '4',
 '5',
 '6',
 '7',
 '8',
 '9']
```

In [117]:
```
s='abded'
i='a'
s+i #i加后面
i+s #i加前面
```

Out[117]:
```
'aabded'
```

In [124]:
```
s='abcde'
s=s[::-1] #reverse s
s
```
Out[124]:
```
'edcba'
```

In [126]:
```
#calculate a factorial using recursion
def factorial(x):
    if x<0:#edge case
        return -1
    if x==0:
        return 1
    if x==1:
        return x
    else: #x=2,3,...
        return x*factorial(x-1) #不断call function----run, 直到return
factorial(3)
```
Out[126]:
```
6
```

In [ ]:
```
#class performance: given table: id,student,assignment1,assignment2
#output largest difference in total score----highest-lowest
import pandas as pd
import numpy as np
#create new column:add together for each student
box_scores['total score']=box_scores['assignment1']+box_scores['ass
box_scores['total score'].max()-box_scores['total score'].min() #ma
```

In [ ]:
```
#inspection scores for business
import pandas as pd
# Start writing code
new=sf_restaurant_health_violations
new=new[new.notnull()['inspection_score']==True] #notnull--变成true/
new.groupby(['business_name'])['inspection_score'].median().sort_va
# 算每个business name(左列) 的 inspection score (右列) 的median----sort
#根据左边, 的算右边的数
```

In [ ]:
```
#number of records by variety
iris.groupby(["variety"])["sepal_length"].count().sort_values().res
#算每个variety (在右边一列) 的sepal_length个数 (一共有多少个) ---sort asce
#两列, 根据左边的, 算右边的count个数
```

In [ ]:
```
#lowest priced orders
import pandas as pd
import numpy as np
merge=pd.merge(customers,orders,left_on="id",right_on="cust_id") #m
#个东西----以此为基准左右对齐
result=merge.groupby(["cust_id","first_name"])['total_order_cost'].
#按前两列的每个cust id, 算第三列 (cost) 的最小值min for each
```

In [ ]:
```
#income by title and gender
import pandas as pd
#>=1个bonus
#整理bonus表：左边id，算右边bonus的总数。to_frame():to dataframe.
sf_bonus_summary=sf_bonus.groupby(['worker_ref_id'])['bonus'].sum()
#merge两张表，employee的id和worker id是一个东西，按这个左右对齐
merged_df=pd.merge(sf_employee,sf_bonus_summary,left_on='id',right_
#在merge表里创一列新的，新的=merge里的salary列+bonus列，each value对应相加
merged_df['avg_total_comp']=merged_df['salary']+merged_df['bonus']
#groupby：第一列title，第二列sex，算第三列（avg）的mean（根据前两列each）
result=merged_df.groupby(['employee_title','sex'])['avg_total_comp'
```

In [ ]:
```
#product transaction count
import pandas as pd

#merge两个表，product_id是一个东西，左右按其变量对齐。自动去掉了无transactic
merge1=pd.merge(excel_sql_inventory_data,excel_sql_transaction_data
#groupby：第一列product id，第二列name，根据前两列的each，算第三列（transac
#自动按第一列从小到大排
result=merge1.groupby(['product_id','product_name'])['transaction_i
#重命名+去掉第一列
result=result.rename(columns={'transaction_id':'number of transacti
```

In [ ]:
```
#find the top 10 ranked song
import pandas as pd
#选取year=2010的 并且 rank1-10的
conditions=billboard_top_100_year_end[(billboard_top_100_year_end['
#选取其中三列，去掉里面重复的值（自动去掉相同行）----not same song twice
result=conditions[['year_rank','group_name','song_name']].drop_dupl
```

In [ ]:
```
#apt in ny and harlem
#3个conditions：apt+nyc+harlem，选取满足这三个的details
conditions=airbnb_search_details[(airbnb_search_details['property_t
```

In [ ]:
```
#duplicate emails
import pandas as pd

# 想要左列email，右列count email：最好要新建一列#，groupby email，然后用tr
employee['number']=employee.groupby('email')['email'].transform('co
#选取新建的number列和email列，去掉重复的----number对应email。（molly：4次，
result=employee[['email','number']].drop_duplicates()
#选取表里面：conditions：number列大于1（出现次数多于1次=duplicate）
result[(result['number']>1)]
```

In [ ]:
```
#review bins on reviews number:output price+its category for each a

import pandas as pd
import numpy as np

#选出number of reviews列，存成num_reviews variable
num_reviews=airbnb_search_details['number_of_reviews']
#conlist：dataframe：列：1st住处，2nd住处... 行：5个条件：0，1-5，6-15，1
#num_reviews从0th index开始----1st住处的review数量 判断五次----输出f,t,f
conlist=[num_reviews==0,num_reviews.between(1,5),num_reviews.betwee
#choicelist：建立一列，从上到下为no(对应第一个判断条件的分类word），few（对应
choicelist=['NO','FEW','SOME','MANY','A LOT']
#新建一列reviews qualification存结果。np.select:对1st住处#reviews，判断2
#第二个数代表 2nd住处。。。 自动选择判断条件为true的那个输出choicelist
airbnb_search_details['reviews qualification']=np.select(conlist,ch
#选取review quali和price两列----存成result 输出
result=airbnb_search_details[['reviews qualification','price']]
```

In [ ]:
```
#business name length: each business name+its #words (no special sy

import pandas as pd

#choose business_name列，去掉重复的name--存成business_name variable
business_name=sf_restaurant_health_violations['business_name'].drop
#用df.str.replace('[,]','')把special symbols替换成空格----words数量不算
business_name1=business_name.str.replace('[#,&,@,!,$,%,^,+,_,=,-,~,
#df.str.split()把name拆开变成list----["john","love"]
result=business_name1.str.split()
#df.str.len() ：算每个的words数----新建一列来存
sf_restaurant_health_violations['number of words']=result.str.len()
#选取words数列和name列----output
result1=sf_restaurant_health_violations[['business_name','number of
```

In [ ]:
```
#positions of letter 'a':find position(从1开始) of 'a' in 'Amitah'

import pandas as pd
#1-based indexing:从1开始，position of 2nd letter=2
#选出irst_name列叫Amitah的----存成worker1
worker1=worker[worker['first_name']=='Amitah']    #ouput为一行
#再单选出这列（output就一个），用str.find('letter')找index---+1 b/c use
worker1['first_name'].str.find('a')+1
#df.str.---对每格的词words进行操作
```

In [ ]:
```
#number of comments /user: each user+ #comments (for each)

import pandas as pd
#use to_datetime function
from datetime import timedelta

#30 days before 2020-02-10(end)
#2 conditions, 选create_at列
#interval:[begin(2020-02-10减掉30天),end(2020-02-10)]. pd.to_datetim
#timedelta (days=#) , delta=30天的差
result=fb_comments_count[(fb_comments_count['created_at']>=pd.to_da
#输出两列: groupby: 第一列 (userid, each user) , 算第二列 (comment数) 的va
#sum(): 里面数的和。 count(): 有几条/个, 一行是一条/个, 有几行for each in 
result.groupby('user_id')['number_of_comments'].sum().reset_index()
```

In [ ]:
```
#finding user purchases:output a list of userid(return user的)
import pandas as pd
import numpy as np
from datetime import datetime #操作日期加减

#更新created_at这一列: 格式改成03-03-2020
amazon_transactions["created_at"] = pd.to_datetime(amazon_transacti
#创一个新的df存: 从小到大sort user_id。再对每个user_id:从小到大sort日期
df = amazon_transactions.sort_values(by=['user_id', 'created_at'],
#新建一列prev_value存: shift得到previous orders的日期, e.g.去掉最后一次购
df['prev_value'] = df.groupby('user_id')['created_at'].shift() #往下
#新建一列days存: 日期差: df的created_at日期列 - df的prev_value日期列: 后一
df['days'] = (pd.to_datetime(df['created_at']) - pd.to_datetime(df[
#选df里, df的days列 (日期差) 在7天内的, 对应的userid----unique():get dist
result = df[df['days'] <= 7]['user_id'].unique()
```

In [ ]:
```
#customer revenue in march: output cust_id+revenue(cost) for each c

import pandas as pd
import numpy as np

#update order_date列: 用pd.to_datetime, 后续进行日期操作
orders['order_date']=pd.to_datetime(orders['order_date'])
#新建march_2019:只选取orders里, orders的order_date列month是3月并且year是
march_2019=orders[(orders['order_date'].dt.month==3)&(orders['order
#groupby: 第一列 (cust_id) , 算第二列 (cost) 的value for each第一列, sum(
#to_frame(''):改第二列的column名字, sort_values('',ascending=False):根
#reset_index()
result=march_2019.groupby("cust_id")['total_order_cost'].sum().to_f
```

In [ ]:
```
from sklearn import linear_model

c1 = df.groupby('companyname').get_group('SUNCOR ENERGY INC') #c1:d
c1['r-rf'] = c1['r'] - c1['rf']
y = c1['r-rf']
fac1 = c1['rmkt']-c1['rf']
fac2 = c1['rxle']-c1['rf']
x =  pd.DataFrame({'fac1': fac1, 'fac2': fac2})
regr = linear_model.LinearRegression()
regr.fit(x, y)
print("Coefficients: \n", regr.coef_)
print("alpha: \n", regr.intercept_)

x_new=pd.concat((curr,one_lagged,two_lagged),axis=1) #multivariate
```

In [1]:
```
#pascal triangle
def generate(numRows):
    result=[[1],[1,1]]#initialize
    for i in range(2, numRows):#从2开始，i=numrows=行，j：i行里的1st，2
        row=[1]#initialize
        for j in range(1,len(result[i-1])):#1 到 result里i-1th eleme
            row.append(result[i-1][j]+result[i-1][j-1]) #result里的
        row.append(1) #最后➕上1
        result.append(row) #把新的row加到结果里
    return result[:numRows]#从0到4th element
generate(5)
```

Out[1]:
```
[[1], [1, 1], [1, 2, 1], [1, 3, 3, 1], [1, 4, 6, 4, 1]]
```

In [ ]:

In [2]:
```
#best time to buy and sell stock
def maxprofit(prices):
    max_profit=0
    for i in range(len(prices)-1):#i只用到倒数第二个
        for j in range(i+1,len(prices)): #j从i后一个开始，到最后一个
            #不断的第一个和第二个/第三个/第四个。。。比较
            profit=prices[j]-prices[i] #后一个➖前一个
            if profit>max_profit: #大于0
                max_profit=profit #assign成输出profit
    return max_profit
maxprofit([7,1,5,3,6,4])
```

Out[2]:
```
5
```

In [11]:
```
#valid palindrome
def ispalindrome(s):
    if s=="":
        return True
    s=s.lower() #转成lowercase
    a='' #新建输出string
    for i in [*s]: #i=拆分的str
        if i.isalpha(): #如果是letter
            a=a+i #string直接➕上
        if i.isnumeric(): #如果是数字
            a=a+i #➕上
    return a==a[::-1]
ispalindrome("A man, a plan, a canal: Panama")
```

Out[11]:
```
True
```

In [4]:
```
s="sefasge102"
[*s] #拆分成单个string
```

Out[4]:
```
['s', 'e', 'f', 'a', 's', 'g', 'e', '1', '0', '2']
```

In [15]:
```
#single number
from functools import reduce
def singlenumber(nums):
    return reduce(lambda total, el: total ^ el, nums)
#reduce(function,sequence[,initial]) #从initial开始, run function: (2
#lambda arguments(variables): expression
#^: 先转成binary, 都是0 or 1--输出0, 不一样--输出1。再转成十进制
singlenumber([1,2,2])
```

Out[15]:
```
1
```

In [38]:
```
#majority element
def majorityelement(nums):
    majority_count = len(nums)//2
    for num in nums:
        count = sum(1 for elem in nums if elem == num)
        if count > majority_count:
            return num
majorityelement([2,2,3,2,2,1])
```

Out[38]:
```
2
```

In [10]:
```
#excel sheet column number
def titletonum(columnTitle):
    ans,pos=0,0
    for i in reversed(columnTitle): #reverse string
        digit=ord(i)-64 #ord(i) return unicode for letter A-Z: ord(
        #A:1=65-64...
        ans=ans+digit* 26**pos #update ans: +1*26^0 + 1*26^1
        pos=pos+1 #update pos (for expo index)
    return ans
titletonum("AB")
```

Out[10]:
```
28
```

In [6]:
```
ord("Z")
```

Out[6]:
```
90
```

In [18]:
```
#happy number
def ishappy(n):
    def get_next(n):
        total_sum=0
        while n>0: #boundary condition:positive number
            n,digit=divmod(n,10) #n/10--quotient,remainder
            total_sum=total_sum+digit**2 #udpate +each digit^2
        return total_sum #sum of digit square
    seen=set() #empty set
    while n!=1 and n not in seen: #boundary coundition: not 1(1 is
        seen.add(n) #add n to set (if n not in set)
        n=get_next(n) #udpate n: n=total sum. 不断的total sum
    return n==1  #if n=1, true

ishappy(19)
```

Out[18]:
```
True
```

In [13]:
```
a=set()
print (a)
```

```
set()
```

In [22]:
```
def reverselist(head):
    prev=None
    current=head

    while current:
        temp=current.next
        current.next=prev
        prev=current
        current=temp

    return prev

reverselist([1,2,3,4,5])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-22-b99d247b3e9e> in <module>
     11     return prev
     12 
---> 13 reverselist([1,2,3,4,5])

<ipython-input-22-b99d247b3e9e> in reverselist(head)
      4 
      5     while current:
----> 6         temp=current.next
      7         current.next=prev
      8         prev=current

AttributeError: 'list' object has no attribute 'next'
```

In [ ]:
```
#contains duplicate
def containdup(nums):
    hset=set() #empty set
    for i in nums: #i=each element
        if i in hset: #if i in set
            return True #true: duplicate
        else: #not in
            hset.add(i) #add i to set
```

In [24]:
```
#missing number
def missingnumber(nums):
    n=len(nums) #has n numbers=length
    for i in range(n+1): #i= 0,..,n---range[0,n]
        if i not in nums: # if i 不在nums list里
            return i #输出i
missingnumber([0,1,3])
```

Out[24]:
```
2
```

In [2]:
```
#move zeros
def movezero(nums):
    i=0
    for j in range(len(nums)): #fast=0,1,2,..
        if nums[j]!=0 and nums[i]==0: #e.g. j=1,nums[1]=1,nums[0]=0
            nums[i],nums[j]=nums[j],nums[i] #nums[0]=1,nums[1]=0
            #后面的数和前面的0交换，后面往前移
        if nums[i]!=0:#nums[0]=1
            i=i+1 #update i, i=1
    #第一个数=0，不管，到第二个
    return nums
movezero([0,1,3,12,0])
```

Out[2]:
```
[1, 3, 12, 0, 0]
```

In [8]:
```
#power of 3
def ispowerof3(n): #n integer

    def helper(i):
        if i<1:
            return False #not 3^x
        if i==1:
            return True
        if i%3!=0: #不是3的倍数，肯定不是power of 3
            return False
        return helper(i//3) # (不属于上面) 如果是3的倍数，一直除下去到i=1,

    return helper(n) #--call helper function

ispowerof3(28)
#e.g. n=28,return helper(28):i=28,return False--False
#e.g. n=27, return helper(27):i=27,return helper(27//3=9)--i=9,retu
#--retrun True--True
```

Out[8]:
```
False
```

In [3]:
```
5//3
```

Out[3]:
```
1
```

In [13]:
```
#reverse string
def reversestr(s): #s=["h","e","l","o"]
    l=0
    r=len(s)-1 #r=3--最后一个index
    while l<r:#0<3
        s[l],s[r]=s[r],s[l] #s[0]=s[3]=o,s[3]=s[0]=h--第一个和最后一个
        l=l+1 #update l:往后一个
        r=r-1 #update r: 往前一个
    return s
reversestr(["h","e","l","o"])
```

Out[13]:
```
['o', 'l', 'e', 'h']
```

In [ ]:
```
#intersection of 2 arrays
def intersect(nums1,nums2):
    list1=[] #建result list
    for i in nums1: #e.g.nums1=[1,2,2,1],i=1,2,2,1
        if i in nums2 and i not in list1: #i也在nums2 and 不在result
            list1=list1+[i]*min(nums1.count(i),nums2.count(i)) #upd
    return list1
```

In [14]:
```
[1]*3
```

Out[14]:
```
[1, 1, 1]
```

In [26]:
```
#first unique character in string
def firstunique(s):
    result={}
    for i in s:
        if i not in result:
            result[i]=1
        else:
            result[i]=result[i]+1
            #result已建好
    for j in range(len(s)):#j=0,1,..: j=index in s
        if result[s[j]]==1: #s[j]='',result['']=对应的value
            return j #break--output j index

firstunique("loveleetcode")
```

Out[26]:
```
2
```

In [31]:
```
#fizzbuzz
def fizzbuzz(n):
    result=[]
    for i in range(1,n+1):
        if i%3==0 and i%5==0:
            result.append("FizzBuzz")
        if i%3==0:
            result.append("Fizz")
        if i%5==0:
            result.append("Buzz")
        if i%3!=0 and i%5!=0:
            result.append(str(i))
    return result
fizzbuzz(5)
```

Out[31]:
```
['1', '2', 'Fizz', '4', 'Buzz']
```

In [ ]:
```
#linked list , tree
```

In [ ]:
```
```