1. 宝石与石头

给定字符串J 代表石头中宝石的类型，和字符串 S代表你拥有的石头。 S 中每个字符代表了一种你拥有的石头的类型，你想知道你拥有的石头中有多少是宝石。

J 中的字母不重复，J 和 S中的所有字符都是字母。字母区分大小写，因此"a"和"A"是不同类型的石头。

示例 1:

输入: J = "aA", S = "aAAbbbb"

输出: 3

示例 2:

输入: J = "z", S = "ZZ"

输出: 0

注意:

S 和 J 最多含有50个字母。

J 中的字符不重复。

class Solution:

def numJewelsInStones(self, J, S):

"""

:type J: str

:type S: str

:rtype: int

"""

num=0

for i in S:

if i in J:

num+=1

return num

分析：语句（if i in X：）能快速判断一个对象里是否含有所找的目标

2. 判断路线成圈

初始位置 (0, 0) 处有一个机器人。给出它的一系列动作，判断这个机器人的移动路线是否形成一个圆圈，换言之就是判断它是否会移回到原来的位置。

移动顺序由一个字符串表示。每一个动作都是由一个字符来表示的。机器人有效的动作有 R（右），L（左），U（上）和 D（下）。输出应为 true 或 false，表示机器人移动路线是否成圈。

示例 1:

输入: "UD"

输出: true

示例 2:

输入: "LL"

输出: false

class Solution(object):

def judgeCircle(self, moves):

"""

:type moves: str

:rtype: bool

"""

return (moves.count("U") == moves.count("D")) and (moves.count("R") == moves.count("L"))

分析：

1. 判断是否回到原点，只需判断相反的操作的次数是否相同。
2. str.count（）方法能判断字符出现的次数

3.汉明距离

两个整数之间的汉明距离指的是这两个数字对应二进制位不同的位置的数目。

给出两个整数 x 和 y，计算它们之间的汉明距离。

注意：

0 ≤ x, y < 231.

示例:

输入: x = 1, y = 4

输出: 2

解释:

1 (0 0 0 1)

4 (0 1 0 0)

↑ ↑

上面的箭头指出了对应二进制位不同的位置。

class Solution:

def hammingDistance(self, x, y):

"""

:type x: int

:type y: int

:rtype: int

""

return bin(x^y).count('1')

分析：

1.bin（int）将十进制转化为二进制（0b111），类型为str。

2.^是异或运算符，相同为0，不同为1。

1. 数字的补数

给定一个正整数，输出它的补数。补数是对该数的二进制表示取反。

注意:

给定的整数保证在32位带符号整数的范围内。

你可以假定二进制数不包含前导零位。

示例 1:

输入: 5

输出: 2

解释: 5的二进制表示为101（没有前导零位），其补数为010。所以你需要输出2。

示例 2:

输入: 1

输出: 0

解释: 1的二进制表示为1（没有前导零位），其补数为0。所以你需要输出0。

class Solution:

def findComplement(self, num):

"""

:type num: int

:rtype: int

"""

n = len(bin(num))-2

return num^(2\*\*n-1)

分析：

1. bin（int）将十进制转化为二进制（0b111），类型为str。

2.按位直接01取反最好的方法是按位异或1运算。

3.所以要求出一共多少位，再与同样位数的111111异或你懂的

1. 自除数

自除数 是指可以被它包含的每一位数除尽的数。

例如，128 是一个自除数，因为 128 % 1 == 0，128 % 2 == 0，128 % 8 == 0。

还有，自除数不允许包含 0 。

给定上边界和下边界数字，输出一个列表，列表的元素是边界（含边界）内所有的自除数。

示例 1：

输入：

上边界left = 1, 下边界right = 22

输出： [1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12, 15, 22]

注意：

每个输入参数的边界满足 1 <= left <= right <= 10000。

class Solution:

def selfDividingNumbers(self, left, right):

"""

:type left: int

:type right: int

:rtype: List[int]

"""

li=[]

for index in range(left,right+1):

flage=True

num=int(index)

index=list(str(index))

for i in index:

if i == '0' or (num%int(i)!=0):

flage=False

if flage:

li.append(num)

return li

分析：这是我自己写的代码，我就是觉得这已经可以了。自除数要求先解析出数字的每一位数，要注意的有两点，一是被除数不能为0，所以像10，100，1000这些不符合要求，二是必须每一位都要能被整除%，所以只要有一位不能被整除，就不符合条件。因此需要设置一个flage，当都不存在上述两点注意的时候就是自除数。整型数据不能直接转化为列表，需要中间加个字符串转化一下。

1. 唯一摩尔斯密码词

国际摩尔斯密码定义一种标准编码方式，将每个字母对应于一个由一系列点和短线组成的字符串， 比如: "a" 对应 ".-", "b" 对应 "-...", "c" 对应 "-.-.", 等等。

为了方便，所有26个英文字母对应摩尔斯密码表如下：

[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]

给定一个单词列表，每个单词可以写成每个字母对应摩尔斯密码的组合。例如，"cab" 可以写成 "-.-.-....-"，(即 "-.-." + "-..." + ".-"字符串的结合)。我们将这样一个连接过程称作单词翻译。

返回我们可以获得所有词不同单词翻译的数量。

例如:

输入: words = ["gin", "zen", "gig", "msg"]

输出: 2

解释: 各单词翻译如下:

"gin" -> "--...-."

"zen" -> "--...-."

"gig" -> "--...--."

"msg" -> "--...--."

共有 2 种不同翻译, "--...-." 和 "--...--.".

注意:

单词列表words 的长度不会超过 100。

每个单词 words[i]的长度范围为 [1, 12]。

每个单词 words[i]只包含小写字母。

class Solution:

def uniqueMorseRepresentations(self, words):

"""

:type words: List[str]

:rtype: int

"""

d=dict(zip('abcdefghijklmnopqrstuvwxyz',[".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--","-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]))

newwords=[]

for word in words:

mos=''

for zi in word:

mos+=d[zi]

newwords.append(mos)

newwords= list(set(newwords))

return len(newwords)

分析：

1.思路就是通过拆解列表单词，再从单词拆解单个字母，根据字典找到字母对应的摩斯电码，利用字符串拼接出新的摩斯单词，摩斯单词再构成新的列表。用set方法队列表去重。最后输出去重后的长度。

2.构造字典时，当字典元素过多，可以使用zip（）方法进行打包构造。

3.set（）方法可以对列表进行去重，原理是利用集合的相关知识。Set过后的列表类型为set，所以还需要list（set（newwords））将列表类型转换回来。

4.字符串拼接直接+，列表元素添加使用append（）方法

1. 键盘行

给定一个单词列表，只返回可以使用在键盘同一行的字母打印出来的单词。键盘如下图所示。

示例1:

输入: ["Hello", "Alaska", "Dad", "Peace"]

输出: ["Alaska", "Dad"]

注意:

你可以重复使用键盘上同一字符。

你可以假设输入的字符串将只包含字母。

class Solution:

def findWords(self, words):

"""

:type words: List[str]

:rtype: List[str]

"""

lin1="qwertyuiop"

lin2="asdfghjkl"

newwords=[]

for word in words:

l1,l2,l3=0,0,0

word1=word.lower()

for zi in word1:

if zi in lin1:

l1+=1

elif zi in lin2:

l2+=1

else:

l3+=1

if l1==len(word) or l2==len(word) or l3==len(word):

newwords.append(word)

return newwords

分析：

1. 这题跟上一题很相似，也是先把单词提出来，再把单词的字母提出来，提出来之后就对照地找是不是在同一键盘行里，判断的方法就是如果单词的所有字母都在同一键盘行里，那么l123的某个长度肯定与单词的长度相等。
2. 判断字母的归属不一定要用列表，用字符串也可以。
3. 或运算用or ，将字符串大写转换成小写用lower（）方法，反之用higer（）方法
4. elif 后面直接加条件，不要加：号
5. Nim游戏

您和您的朋友，两个人一起玩 Nim游戏：桌子上有一堆石头，每次你们轮流拿掉 1 到 3 块石头。 拿掉最后一块石头的人就是胜利者。由您来开局。

你们两个都是聪明人，相信都有最佳的游戏策略。 请编写一个函数，来判断您是否可以在给定的石头数量的情况下赢得游戏。

比方说，如果堆中有4块石头，那么你永远不会赢得比赛：无论你拿走的是 1块，2块 还是 3块 石头，最后一块石头总是会被你的朋友拿走。

class Solution:

def canWinNim(self, n):

"""

:type n: int

:rtype: bool

"""

if n% 4 != 0:

return True

else:

return False

分析：题目给了很大的提示，只讨论必输的情况下，你拿一次，对手拿一次为一个回合，在每个回合中，你最多拿3个，对手最少能拿1，反之也是。两人拿取的总数和为4时，对手就能控制每回合的形式是：你先拿，他后拿。所以只要总数是4的倍数，必输。

1. 反转字符串中的单词 III

给定一个字符串，你需要反转字符串中每个单词的字符顺序，同时仍保留空格和单词的初始顺序。

示例 1:

输入: "Let's take LeetCode contest"

输出: "s'teL ekat edoCteeL tsetnoc"

注意：在字符串中，每个单词由单个空格分隔，并且字符串中不会有任何额外的空格。

class Solution:

def reverseWords(self, s):

"""

:type s: str

:rtype: str

"""

return (' '.join(x[::-1] for x in s.split(' ')))

分析：这题看起来简单，实则包含很多知识。

1. 字符串的操作，str.split（“”）能按照参数为边界切割字符串，并保存成一个新的列表。
2. str[：：-1]字符串的切片工具能直接将字符串倒序输出。
3. Str.join函数，能将一个列表的元素，按照str的内容间隔将列表元素拼接在一起。
4. ' '.join(x[::-1] for x in s.split(' '))，，方法（XX（操作） for XX in YY）的形式称为链表推导式，当要使用循环，循环内对XX进行操作时可以以参数的形式简写。功能等于正常的for循环操作。
5. 数组拆分 I

给定长度为 2n 的数组, 你的任务是将这些数分成 n 对, 例如 (a1, b1), (a2, b2), ..., (an, bn) ，使得从1 到 n 的 min(ai, bi) 总和最大。

示例 1:

输入: [1,4,3,2]

输出: 4

解释: n 等于 2, 最大总和为 4 = min(1, 2) + min(3, 4).

提示:

n 是正整数,范围在 [1, 10000].

数组中的元素范围在 [-10000, 10000].

class Solution:

def arrayPairSum(self, nums):

"""

:type nums: List[int]

:rtype: int

"""

nums=sorted(nums)

sum=0

for i in range(0,len(nums),2):

sum+=nums[i]

return sum

分析：这题看起来简单，实际上也确实非常简单。心知识点有：

1. 在python里，自带排序方法，可以写成nums=sorted（nums）也可以写nums.sort（），特么那我学python还学个屁快速排序合并排序。
2. 在for循环里，实现跳着循环，可以写用range（范围，跳步）这种方式写。比while循环好用吧我觉得，但是这样写得到的i指的就是列表的下标，所以引用时记清楚就行了。
3. 交替位二进制数

给定一个正整数，检查他是否为交替位二进制数：换句话说，就是他的二进制数相邻的两个位数永不相等。

示例 1:

输入: 5

输出: True

解释:5的二进制数是: 101

示例 2:

输入: 7

输出: False

解释:7的二进制数是: 111

示例 3:

输入: 11

输出: False

解释:11的二进制数是: 1011

示例 4:

输入: 10

输出: True

解释:10的二进制数是: 1010

class Solution:

def hasAlternatingBits(self, n):

"""

:type n: int

:rtype: bool

"""

l=bin(n)

i=2

while i<len(l)-1:

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

return False

i+=1

return True

分析：这题很简单，只需使用二进制转换，再从第三位开始遍历字符串的每一个元素，如果存在相等就不符合要求，直接返回False，只要if判断不执行，就说明符合要求，所以循环结束后可以直接返回True，注意到的是循环范围要写长度减1.

1. 翻转图像

给定一个二进制矩阵 A，我们想先水平翻转图像，然后反转图像并返回结果。

水平翻转图片就是将图片的每一行都进行翻转，即逆序。例如，水平翻转 [1, 1, 0] 的结果是 [0, 1, 1]。

反转图片的意思是图片中的 0 全部被 1 替换， 1 全部被 0 替换。例如，反转 [0, 1, 1] 的结果是 [1, 0, 0]。

示例 1:

输入: [[1,1,0],[1,0,1],[0,0,0]]

输出: [[1,0,0],[0,1,0],[1,1,1]]

解释: 首先翻转每一行: [[0,1,1],[1,0,1],[0,0,0]]；

然后反转图片: [[1,0,0],[0,1,0],[1,1,1]]

示例 2:

输入: [[1,1,0,0],[1,0,0,1],[0,1,1,1],[1,0,1,0]]

输出: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

解释: 首先翻转每一行: [[0,0,1,1],[1,0,0,1],[1,1,1,0],[0,1,0,1]]；

然后反转图片: [[1,1,0,0],[0,1,1,0],[0,0,0,1],[1,0,1,0]]

说明:

1 <= A.length = A[0].length <= 20

0 <= A[i][j] <= 1

class Solution:

def flipAndInvertImage(self, A):

"""

:type A: List[List[int]]

:rtype: List[List[int]]

"""

#方法一

a=[]

for x in A:

b=[1-y for y in x[::-1]]

a.append(b)

print(a)

#方法二

for row in A:

for i in range(int((len(row) + 1) / 2)):

row[i], row[~i] = row[~i] ^ 1, row[i] ^ 1

print(A)

分析：对于方法1

使用一个新的列表来保存翻转后的图像，思路就是先将数字取反，再将列表反转。其中用到的列表推导，最后会得到一个列表，可以省去一步列表创建。（其实用到的也是两个新的列表）

对于方法二2

不需要一个新列表，左右操作都在原始列表上进行修改，修改思路是将列表分成两部分，头尾互相取反然后交换位置。其中用到的是解包赋值，在整条语句完成之前，原值是不变的，整个语句结束之后才真正的交换数值和位置