```
def ryerson_letter_grade(n):
    if n < 50:
        return 'F'
    elif n > 89:
        return 'A+'
    elif n > 84:
        return 'A'
    elif n > 79:
        return 'A-'
    tens = n // 10
    ones = n % 10
    if ones < 3:
        adjust = "-"
    elif ones > 6:
        adjust = "+"
    else:
        adjust = ""
    return "DCB"[tens - 5] + adjust



def safe_squares_bishops(n, bishops):
    safec = 0
    for i in range(n):
        for a in range(n):
            safe=True
            for b in bishops:
                if abs(i - b[0]) == abs(a - b[1]):
                    safe=False
            if safe:
              safec+=1
    return safec

def first_preceded_by_smaller(items, k=1):
  leng=len(items)
  for i in range(leng):
      count = 0
      for a in range(i):
        if items[i]>items[a]:
          count += 1
      if k<= count:
        return items[i]


def count_consecutive_summers(n):
    count = 0
    for i in range(1, n+1):
        s = 0
        for a in range(i, n+1):
            s += a
            if s == n:
                count += 1
    return count

def collapse_intervals(items):
    final = ''
    i = 0
    leng=len(items)
    while i < leng:
        fir = i
        final += str(items[fir])
        while fir+1 < leng and items[fir]+1 == items[fir+1]:
```

```
            fir += 1
        if fir != i:
            i = fir+1
            final += '-'+str(items[fir])
        else:
            i += 1
        if i != leng:
            final += ','
    return final


def seven_zero(n):
  digit = 1
  final = 0
  while True:
    if n%2 == 0 or n%5 == 0:
      a = 1
      while digit >= a:
        value =int(a * '7' + (digit-a) * '0')
        if value%n == 0:
          final = value
        a += 1
    else:
      value = int(digit * '7')
      if value%n == 0:
        final = value
      else:
        value=0
    digit += 1
    if final > 0:
      return final


def remove_after_kth(items, k=1):
  final = []
  for i in items:
    if k > final.count(i):
      final.append(i)
  return final

def count_carries(a, b):
  count=0
  carry=0
  maxi=max(a,b)
  mini=min(a,b)
  while mini>0 or carry>0:
      x=mini%10
      y=maxi%10
      total=x+y+carry
      if total>9:
          count+=1
          carry=total//10
      else:
          carry=0
      mini//=10
      maxi//=10
  return count



def group_and_skip(n,out,ins):
   final=[]

   while(n>0):
       rem=n%out
```

```
        final.append(rem)
        group=n//out
        n=group*ins
    return final

def brangelina(first, second):
  i,vowels= 0,'aeiou'
  while second[i] not in vowels:
    i+=1
  second=second[i:]
  groups, in_group=[], False
  for j in range(len(first)):
    if first[j] in vowels:
      if not in_group:
        groups.append(j)
        in_group=True
    else:
      in_group=False
  if len(groups)==1:
    first= first[:groups[0]]
  else:
    first= first[:groups[-2]]
  return first+second

#helper funcction for collect_numbers
def find_inverse(perm):
    leng=len(perm)
    inv = [0]*leng
    for i in range(leng):
        inv[perm[i]] = i
    return inv
def collect_numbers(perm):
    inverse = find_inverse(perm)
    mx=-1
    lengt= len(inverse)
    c=1
    for i in range(lengt):
        if(inverse[i]>mx):
            mx=inverse[i]
        else:
            c+=+1
            mx = inverse[i];
    return c


def pyramid_blocks(n,m,h):
  final=0;
  for i in range(h):
    final+=(n*m)
    n+=1
    m+=1
  return final

def taxi_zum_zum(moves):
    x, y = 0, 0
    direction ='N'
    for i in moves:
      if i=='F':
            if direction=='S':y-=1
            elif direction=='N': y+=1
            elif direction=='W':x-=1
            elif direction=='E':x+=1


      elif i=='L':
```

```
            if direction=='E':direction='N'
            elif direction=='N': direction='W'
            elif direction=='S':direction='E'
            elif direction=='W':direction='S'



      elif i=='R':
            if direction=='E':direction='S'
            elif direction=='N': direction='E'
            elif direction=='S':direction='W'
            elif direction=='W':direction='N'



    return (x,y)
def words_with_letters(words, letters):
    final = []
    for i in words:
        count_a = 0
        count_b = 0

        while count_b<len(i) and count_a<len(letters):
            if i[count_b] == letters[count_a]:
                count_a += 1
            count_b += 1
        if count_a==len(letters):
            final.append(i)
    return final




def is_cyclops(n):
    n= str(n)
    if len(n)% 2 == 1:
        if n.count("0")>1:
            return False
        if n[((len(n)+1)//2)-1]=="0":
            return True
        else:
            return False
    else:
        return False


def is_ascending(items):
    if len(items) == len(set(items)):
        if sorted(items)== items:
            return True
        else:
            return False
    else:
        return False


def riffle(items, out=True):
  if items==[]:
    return items
  final = []
  half= len(items)//2
  if out==True:
    half1= items[:half]
    half2= items[half:]
  else:
```

```
    half1= items[half:]
    half2= items[:half]
  i=0
  while i < half:
    final.append(half1[i])
    final.append(half2[i])
    i=i+1
  return final

def only_odd_digits(n):
    if n<= 0:
        return False
    list= []
    while n >0:
        list.append(n%10)
        n//=10
    list=list[::-1]

    for n in list:
        if all(int(n) % 2 == 1 for n in list):
            return True
        else:
            return False


def domino_cycle(tiles):
   a = len(tiles)
   if a == 0:
       return True
   elif(tiles[0][0] == tiles[-1][1]):
       for i in range(a-1):
           if tiles[i][1] != tiles[i+1][0]:
               return False
       else:
           return True
   else:
       return False


def pancake_scramble(text):
  a= text
  b=len(text)
  for c in range(2,b+1):
    a= a[:c][::-1]+a[c:]
  return a




def count_dominators(items):
 l=len(items)
 if l == 0:
   return 0
 maxim = items[-1]
 count = 1
 for i in range(-2, -len(items) - 1, -1):
   if items[i] > maxim:
     count +=1
   if maxim < items[i]:
    maxim = items[i]

 return count
```

```
def m_of_3(a,b,c):
  return sorted([a,b,c])[1]
def tukeys_ninthers(items):
  while len(items)>1:
    medians= []
    for i in range(0,len(items),3):
      m=m_of_3(items[i], items[i+1], items[i+2])
      medians.append(m)
    items= medians
  return items[0]

def count_and_say(digits):
        if len(digits) == 0:
            return ""
        else:
            final = ""
            fir = digits[0]
            count = 1
            for i in digits[1:]:
                if i == fir:
                    count += 1
                else:
                    final += str(count)
                    final += fir
                    count = 1
                fir = i
            final += str(count)
            final += fir
            return final

def unscramble(words,word):
  final=[]
  leng= len(word)
  for i in words:
    if  leng ==len(i) and i[-1]==word[-1] and i[0]==word[0] :
      if sorted(list(word[1:-1]))==sorted(list(i[1:-1])):
        final.append(i)
  return final

def is_left_handed(pips):
  combo=[(1,2,3), (2,3,1), (3,1,2), (1,4,2), (2,1,4), (4,2,1), (1,3,5), (3,5,1), (5,1,3),
(1,5,4), (4,1,5), (5,4,1), (2,6,3), (3,2,6), (6,3,2), (2,4,6), (4,6,2), (6,2,4), (3,6,5),
(5,3,6), (6,5,3), (4,5,6), (5,6,4), (6,4,5)]
  if pips in combo:
    return True
  else:
    return False

def extract_increasing(digits):
    final = []
    current = 0
    previous = -1
    for i in range(len(digits)):
        d = int(digits[i])
        current = 10 * current + d
        if current > previous:
            final.append(current)
            previous = current
            current = 0
    return final


def josephus(n,k):
```

```
        sold=[(i+1) for i in range(n)]
        st=0
        lis=[]
        sz=n
        while len(sold)>1:
                st=(st+(k-1))%sz
                sz-=1
                lis.append(sold[st])
                del sold[st]
        lis.append(sold[0])
        return lis

def expand_intervals(intervals):
    intr = intervals.split(",")
    lst = []
    if len (intervals) == 0:
      return lst
    for num in intr:
        num = num.strip()
        if "-" in num:
            first = int(num.split("-")[0])
            last = int(num.split("-")[1])
            for i in range(first, last + 1):
                lst.append(i)
        else:
            lst.append(int(num))
    return lst

#helper funtion for three_summers
def two_summers(items, goal,i=0):
  j= len(items)-1
  while i< j:
    s = items[i]+items[j]
    if s ==goal:
      return True
    elif s <goal:
      i+=1
    else:
      j-=1
  return False

def three_summers(items,goal):
  for k in range(len(items)):
    if (two_summers(items,goal-items[k],k+1)):
      return True
  return False

def reverse_vowels(text):
  result, vowels = "", 'aeiouAEIOU'
  vowelList= [c for c in text if c in vowels]
  for c in text:
    if c not in vowels:
      result=result+c
    else:
      v= vowelList.pop()
      v= v.lower() if c.islower() else v.upper()
      result=result+v
  return result


def sum_of_two_squares(n):
    square = int(n ** 0.5)
    for i in range(square, 0, -1):
        tem = n - i * i
        if int(tem ** 0.5) ** 2 == tem and int(tem ** 0.5) > 0:
```

```
            return i, int(tem ** 0.5)


def can_balance(items):
  for i in range(len(items)):
      left = 0
      right = 0
      for j in range(i):
          left += items[j]*(i-j)
      for j in range(i+1, len(items)):
          right += items[j]*(j-i)
      if left == right:
          return i
  else:
      return -1

def colour_trio(colours):
  length=len(colours)
  while len(colours) > 1:
    final = ""

    for i in range(len(colours)-1):
      colour_1 = colours[i]
      colour_2= colours[i+1]

      if colour_1 == colour_2:
        final += colour_1

      else:
        final += "ybr".replace(colour_1,"").replace(colour_2,"")
    else:
      colours = final

  return colours



def give_change(amount, coins):
    final = []
    ind = 0
    while amount > 0:
        if amount >= coins[ind]:
            final.append(coins[ind])
            amount = amount - coins[ind]
        else:
            ind+=1

    return final

def count_growlers(animals):

    dogs = []
    cats = []
    dog_count = 0
    cat_count = 0
    for animal in animals:
        if(animal == 'dog' or animal == 'god'):
            dog_count += 1
            dogs.append(dog_count)
            cats.append(cat_count)
        else:
            cat_count += 1
            cats.append(cat_count)
            dogs.append(dog_count)
```

```
    final = 0
    a = 0
    while a < len(animals):
        if animals[a] == 'dog' or animals[a] == 'cat':
            if a > 0 and dogs[a-1] > cats[a-1]:
                final += 1

        elif animals[a] == 'god' or animals[a] == 'tac':
            if (dogs[len(dogs)-1] - dogs[a]) > (cats[len(cats)-1] - cats[a]):
                final += 1
        a += 1


    return final


def safe_squares_rooks(n,rooks):
    unsafe_r=set()
    unsafe_c=set()
    for rook in rooks:
        unsafe_r.add(rook[0])
        unsafe_c.add(rook[1])
    count_r=n-len(unsafe_r)
    count_c=n-len(unsafe_c)
    return count_r*count_c

def knight_jump(knight, start, end):
  new = []
  for i in range(len(knight)):
    new.append(knight[i])
  for i in range(len(knight)):
    val = abs(start[i] - end[i])
    if val in new:
      new.remove(val)
      continue
    else:
      return False
  return True
```