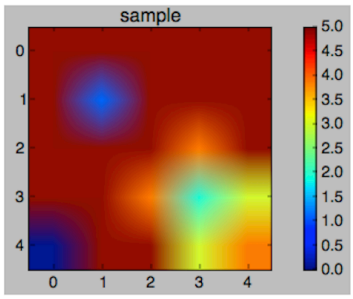
☆ Minima

Find the 3 most extreme local minima of a 2D surface.

Input Format	Output Format
The surface is given as a square matrix (n by n). A minima is a value that is smaller than all its neighbors. Arguments: n: int, the order of the array m: a 2D array (list of lists) of floats Constraints: n < 50	An array (list) of floats, sorted ascending, of the minima values. Return a max of 3 values. If none are found, return an empty array (list).

Sample Input		Sample Output
	n = 5 m= [[5., 5., 5., 5., 5.], [5., 1., 5., 5., 5.], [5., 5., 5., 4., 5.], [5., 5., 4., 2., 3.], [0., 5., 5., 3., 4.]]	[0., 1., 2.]

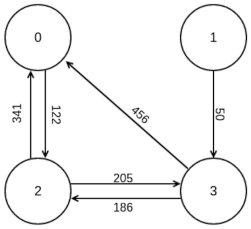
https://stackoverflow.com/questions/3986345/how-to-find-the-local-minima-of-a-smooth-multidimensional-array-in-numpy-efficie 第一个回答

☆ One-to-All Time on a Sparse Directional Network

Suppose you have a network of devices. Each device is directionally connected to a subset of other devices and each communication link has some fixed delay.

Suppose you want to send a One-to-All broadcast message. In this case, some origin node attempts to send a message to every other node. It sends the message to each of it's outbound neighbors, and each of those nodes forwards the message on to their outbound neighbors, and so on. Each edge traversal incurs a time-penalty equal to the edges weight.

For example, you may have some topology like this:



One can define such a network using an adjacency matrix where each row defines the outbound edges, and each column, the inbound, using Null/None to refer to the absence of an edge, and an Integer value indicating the edge penalty.

For example, the above network can be represented as:

Device ID	0	1	2	3
0	None	None	122	None
1	None	None	None	50
2	341	None	None	205

One can define such a network using an adjacency matrix where each row defines the outbound edges, and each column, the inbound, using Null/None to refer to the absence of an edge, and an Integer value indicating the edge penalty.

For example, the above network can be represented as:

Device ID	0	1	2	3
0	None	None	122	None
1	None	None	None	50
2	341	None	None	205
3	456	None	186	None

Write a function called "*broadcast_delivery_time*" that takes the ID of the origin device (list index of node), and an adjacency matrix as a list of lists (row-wise IE: input[0] == ROW 1), and outputs the minimum guaranteed time of delivery, or Null if the message *cannot* be delivered. That is, the time required for the message to have been received by every other device, or Null if the message *cannot* be delivered to every other device.

IE your function signature should look something like the following:

```
func broadcast_delivery_time(origin_id (int), adj_matrix (list of lists[int])) -> int or Null/None
```

```
class Solution(object):

    def networkDelayTime(self, times, N, K):

        from collections import defaultdict

        nodes = defaultdict(dict)

        Q = set(range(N))

        for u, v, w in times:

            nodes[u - 1][v - 1] = w

        dist = [float('inf')] * N

        dist[K - 1] = 0

        while len(Q):

            u = None

            for node in Q:

                if u == None or dist[node] < dist[u]:
```

```
                    u = node

            Q.remove(u)

            for v in nodes[u]:

                alt = dist[u] + nodes[u][v]

                if alt < dist[v]:

                    dist[v] = alt

        d = max(dist)

        return -1 if d == float('inf') else d
```

☆ Detect colinearity

Given n distinct lattice points (i.e., coordinates are integers) on the xy plane, determine if there are three points that lie on a straight line. There are at least 3 points in all test cases.

Input:
The first line is the number of points n. This number is guaranteed to be at least 3 in all test cases
The second line is the dimension of the points, which is fixed to be 2 in this problem.
The following n lines consists of two integers, separated by space.

Output:
If there are three points lying on the same line, return 1. Else return 0

Example 1:
Input:
3
2
-1 -1
0 0
1 1
Output:
1

Example 2:
Input:
3
2
-1 -1
0 0
1 2
Output:
0

```
def collinear(x1, y1, x2, y2, x3, y3):

    """ Calculation the area of
        triangle. We have skipped
        multiplication with 0.5 to
        avoid floating point computations """
    a = x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2)

    if (a == 0):
        print "1"
    else:
```

```
        print "0"

# Driver Code
x1, x2, x3, y1, y2, y3 = 1, 1, 1, 1, 4, 5
collinear(x1, y1, x2, y2, x3, y3)

# This code is contributed
# by Sachin Bisht
```

☆ 3-Card Poker

Imagine a game, similar to many poker games, in which each player is dealt 3 cards, and the player with the higher value hand wins the draw. We play this game with a 40-card deck: 4 cards of each value 0-9, and for simplicity, unsuited.

Hand rank is as follows (highest to lowest):
Three of a kind: All three cards have the same value.
One pair: Two cards of the same value.
High card: The highest value card in your hand.

Cards are valued (highest to lowest) 9, 8, ... 0.

If two players have the same ranked hand, then the rank made of the highest value wins, for example, a pair of 9's beats a pair of 2's, and a high card 9 beats a high card 2. If, however, both players have a pair of 8's, then the player with the higher value third card wins. For the high card case, if the second cards also tie, then the third card is considered, and if this card is also tied, then the hand results in a draw.

Consider the following five dealt hands:

P1	Rank	P2	Rank	Winner
742	HC	653	HC	P1
882	1P	742	HC	P1
752	HC	742	HC	P1
884	1P	882	1P	P1
888	3K	555	3K	P1

```
from collections import Counter

values = {
  '2' : 2,
  '3' : 3,
  '4' : 4,
  '5' : 5,
  '6' : 6,
  '7' : 7,
  '8' : 8,
  '9' : 9,
  '10' : 0,
  'A' : 1,
  }

class InvalidCard(~~Exception~~):  //change to “hand has more than 3 cards”
  pass

class Card(object):

  def __init__(self, card='AC'):
    if type(card) != str or len(card) != 2:
      raise InvalidCard
    self.value = values[card[0]]
    self.suite = suites[card[1]]
    self.card = card

  def __unicode__(self):
    return "suite: %s value:%s" % (self.suite, self.value)
```

```
class InvalidHand(~~Exception~~):  //change to “hand has more than 3 cards”
  pass

class Hand(object):

  def __init__(self, cards = []):
    if type(cards) != list or len(cards) != 5 or \
        not all(type(c) == Card for c in cards):
      raise InvalidHand
    self.cards = cards
    self.get_rank()

  def print_values(self):
    return "{} {} {} {}".format(self.card_values(),
                    self.cards_by_hand,
                    "%2d" % self.rank,
                    self.combination)

  def values(self):
    return sorted([c.value for c in self.cards])

  def values_desc(self):
    return sorted([c.value for c in self.cards], reverse = True)

  def card_values(self):
    return [c.card for c in self.cards]

  def suites(self):
    return [c.suite for c in self.cards]

  def get_rank(self):
    values = self.values()
    counter_values = Counter(self.values())
    counter_suites = Counter(self.suites())

    self.cards_by_hand = self.values_desc()
    if len(counter_suites.keys()) == 1 and set(values) == set(range(10, 15)):
      self.rank = 1
    elif len(counter_values.keys()) == 3:
      if 3 in counter_values.values():
        self.rank = 7
        self.combination = "Three Of A Kind"
        if not (self.cards_by_hand[0] == self.cards_by_hand[2]) :
          if not(self.cards_by_hand[1] == self.cards_by_hand[2]):
            self.cards_by_hand = self.cards_by_hand[2:] + self.cards_by_hand[:2]
          else:
            self.cards_by_hand = self.cards_by_hand[1:4] + self.cards_by_hand[:1] +
self.cards_by_hand[4:]
]
    elif len(counter_values.keys()) == 4:
      self.rank = 9
      self.combination = "Pair"
      if self.cards_by_hand[1] == self.cards_by_hand[2]:
        self.cards_by_hand = self.cards_by_hand[1:3] + self.cards_by_hand[:1] + self.cards_by_hand[3:]
      elif self.cards_by_hand[2] == self.cards_by_hand[3]:
        self.cards_by_hand = self.cards_by_hand[2:4] + self.cards_by_hand[:2] + self.cards_by_hand[4:]
      elif self.cards_by_hand[3] == self.cards_by_hand[4]:
        self.cards_by_hand = self.cards_by_hand[3:] + self.cards_by_hand[:3]
    else:
```

```
        self.rank = 10
        self.combination = "High Card"

class InvalidGame(Exception):
    pass

class Game(object):

    def __init__(self, hands = [], game_string = ""):
        if game_string:
            card_strings = game_string.split()
            cards = [Card(string) for string in card_strings]
            hands = [Hand(cards[:5]), Hand(cards[5:])]
        elif type(hands) != list or len(hands) != 2 \
                or not all(type(h) == Hand for h in hands):
            raise InvalidGame
        self.hands = hands

    def compare_hands(self, hands = []):
        if not hands:
            hands = self.hands
        winner = None
        if hands[0].rank != hands[1].rank:
            winner = 0 if hands[0].rank < hands[1].rank else 1
        else:
            winner = 0 if hands[0].cards_by_hand > hands[1].cards_by_hand else 1
        print "%s\n%s\nwinner: %s\n" %(hands[0].print_values(), hands[1].print_values(), str(winner))
        return winner


def construct_game(string):
    game = Game(game_string = string)
    return game.compare_hands()

def main():
    wins_player_A = 0
    with open("poker.txt", "r") as f:
        games = f.read().split("\n")
    for game in games:
        if game:
            wins_player_A += ((construct_game(game) + 1 ) % 2)
    print wins_player_A


if __name__ == "__main__":
    main()
```

https://codereview.stackexchange.com/questions/60738/optimizing-poker-hands-challenge-solution

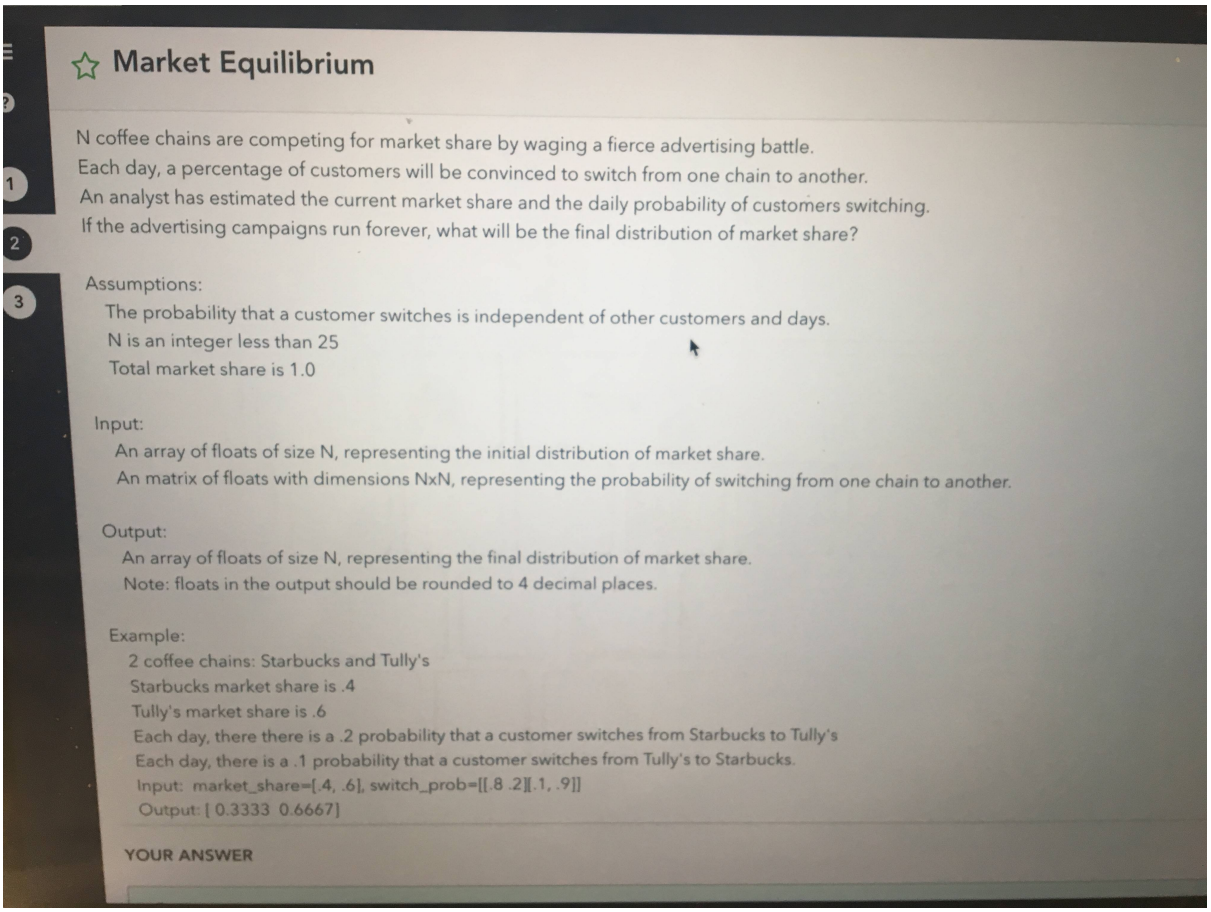
☆ Palindrome Dates

A palindrome date is the kind of date which reads the same backward or forward, in the MM/DD/YYYY format. There are 2 types of palindrome dates: seven-digit palindrome date and eight-digit palindrome date. For example, October 2, 2001 is the first eight-digit palindrome date (10022001) and September 2, 2090 is the last eight-digit palindrome date (09022090) in the 21st Century; Likewise, January 10, 2011 is the first seven-digit palindrome date (1102011) and September 30, 2039 (9302039) is the last seven-digit palindrome date in the 21st Century. Yeah, I know, it's fun!

Here comes the question. Given a year in YYYY format (an integer), can you come up with an algo that outputs the total number of palindrome dates (both seven-digit and eight-digit types) in its century? For example, if you are given the number 2016, your function should return 38 -- There are 38 palindrome dates in the 21st Century.

Please ignore all the years prior to 1000 and later than 9999 (not our concern!). You will only be given years that are in the four-digit format (1000-9999).

```
from calendar import monthrange


def get_palindrome_date(y):
    """
    This function return palindrome date in a century
    :param y:
    :return:
    """

    century_start = (y // 100) * 100 + 1
    century_end = century_start + 99
    count = 0
    for year in range(century_start, century_end):
        for month in range(1, 13):
            number_of_days = monthrange(year, month)[1]
            flag = False
            if len(str(month)) == 1:
                month_string = '0' + str(month)
                flag = True
            else:
                month_string = str(month)
            for day in range(1, number_of_days + 1):
                if len(str(day)) == 1:
                    date_string = month_string + '0' + str(day) + str(year)
                else:
                    date_string = month_string + str(day) + str(year)

                if list(reversed(date_string)) == list(date_string):
                    count += 1
                elif flag and list(reversed(date_string[0:len(date_string) - 1])) == list(
                        date_string[0:len(date_string) - 1]):
                    count += 1

                elif flag and list(reversed(date_string[1:len(date_string)])) == list(date_string[1:len(date_string)]):
                    count += 1

    return count
```



Market Equilibrium

N coffee chains are competing for market share by waging a fierce advertising battle.
Each day, a percentage of customers will be convinced to switch from one chain to another.
An analyst has estimated the current market share and the daily probability of customers switching.
If the advertising campaigns run forever, what will be the final distribution of market share?

Assumptions:
The probability that a customer switches is independent of other customers and days.
N is an integer less than 25
Total market share is 1.0

Input:
An array of floats of size N, representing the initial distribution of market share.
An matrix of floats with dimensions NxN, representing the probability of switching from one chain to another.

Output:
An array of floats of size N, representing the final distribution of market share.
Note: floats in the output should be rounded to 4 decimal places.

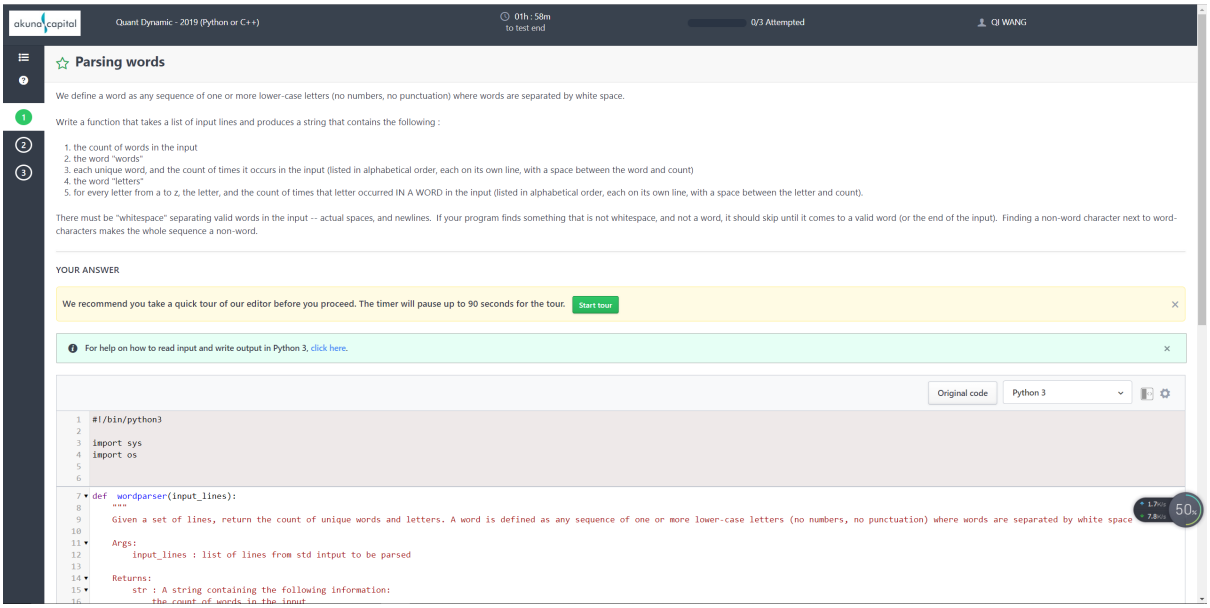
Example:
2 coffee chains: Starbucks and Tully's
Starbucks market share is .4
Tully's market share is .6
Each day, there there is a .2 probability that a customer switches from Starbucks to Tully's
Each day, there is a .1 probability that a customer switches from Tully's to Starbucks.
Input: market_share=[.4, .6], switch_prob=[[.8 .2][.1, .9]]
Output: [0.3333 0.6667]

YOUR ANSWER

TIME = 1000 # number of iterations

```
def main():
    n = int(input("Enter number of coffee chains: "))
    while not 25 > n > 0:      # error checking
        print("Should be below 25")
        n = int(input("Enter number of coffee chains: "))

    # Taking input from user about the chains and their shift
    print("Enter market share of each coffee chain(sum of share shoud be 1")
    mShare = [float(input()) for i in range(n)]
    shiftRate = []
    for i in range(n):
        print("Enter toShift chain for chain(" + str(i + 1) + "): ", end="")
        temp1 = int(input()) - 1
        print("Enter the shift value to that chain: ", end=" ")
        temp2 = float(input())
        shiftRate.append([temp1, temp2])

    # iterations so that market share changes as per specified manner
    # updation of the market shares should be done simultaneously
    for day in range(TIME):
        newShare = mShare[:]              # copy of market share
        shift = [0]*n                     # amount to be shifted array
```

```
    for chain in range(n):                # loop per chain
        temp1 = shiftRate[chain][1]       # storing shift value
        temp2 = newShare[chain] - temp1   # storing share value
        if temp2 < 0:                     # check if share is negative
            temp2 = 0                     # put share as 0
            temp1 = newShare[chain]       # store shift as its current market value
        newShare[chain] = round(temp2,1)
        shift[shiftRate[chain][0]] = temp1  # storing shift value
    mShare =[newShare[i]+shift[i] for i in range(n)]    # updating the new market share

print("\n")
print("Market share after " + str(TIME) + " days is:", end="")
print(mShare)


if __name__ == '__main__':
    main()
```




```
import
sys,
re,
string
```

```
        data = sys.stdin.readlines()


        tmp = []
        findNonWord = False
        wordList = []
```

```
for iLine in data:
	findNonWord = False
	for idx in range(len(iLine)):
		iChar = iLine[idx]


		if re.match("[a-z]", iChar) and not findNonWord:
			tmp.append(iChar)
		elif iChar == ' ':
			if not findNonWord and len(tmp)>0:
				wordList.append(''.join(tmp))
			tmp = []
			findNonWord = False
		else:
			findNonWord = True
			tmp = []


		if idx==len(iLine)-1:
			if not findNonWord and len(tmp)>0:


				wordList.append(''.join(tmp))
			tmp = []


# print wordList
# print ''


wordCount = len(wordList)
print wordCount
print 'words'
wordSet = sorted(set(wordList))


for iWord in wordSet:
	print iWord + " " + str(wordList.count(iWord))


print 'letters'
for iLetters in string.lowercase[:26]:
	count = 0
	for iWord in wordList:
```

```
            count += iWord.count(iLetters)
        print iLetters + " " + str(count)
        count = 0;
```

☆ Flipping signs

Given a string made of '+' and '-' signs of length L, the only allowed operation is to flip K consecutive signs at the same time.

Input:
String: length L, made of only '+' and '-'
Integer: K

Output:
Integer: the minimum number of times needed to flip all signs to '+'. if not possible, output -1.

Constraints:
L >= K
K >= 2

```
class Solution:

    def minNumFlip(self, nums, K):
        '''
        :param nums: 字符串
        :param K: 每次翻转的个数
        :return:
        '''
        temp = "+"
        count = 0
        nums = list(nums) # Python 中字符串是不可变类型，即无法直接修改字符串的某一位字符。需要把字符串转换成列表
        for i in range(len(nums)):
            if nums[i] != temp:
                for j in range(K):
                    nums[i + j] = temp # 重新赋值字符串的某一位字符
                count += 1

        return count

solution = Solution()
# test1
nums = "+++---+++---+++---+++"
```

☆ Packing Melons

Before you are two assembly lines, one with boxes and one with watermelons, both of varying size. Your desire is to put as many watermelons into boxes as possible. You can pick where you start taking watermelons from, but once you start, all melons going past you must be placed, or you must stop. We want you to calculate how many watermelons you can place.

As input, you are given two lists, one of box sizes and one of watermelon sizes. A watermelon will fit into a box with a number greater than or equal to the melon's. Only one melon can go into a box. You can hold onto the melon and skip a box to place it in a later one. Calculate how many watermelons can be placed.

```
Importsys
            import os


            ###


            def melon_count(boxes,melons):
                boxes.sort();
```

```
melons.sort();



result = 0;

i = 0;

j = 0;



while(i != boxes.__len__() and j != melons.__len__()):

     if boxes[i] >= melons[j]:



          result = result+1;

          i = i+1;

          j= j+1;

     else:



          i = i+1;







return result;
```

Output Format
A list of labels.

Sample Input

```
trades = [[99.0, 5.0, 20.0],  # green (good trade)
          [95.0, 15.0, 10.0],  # green (good trade)
          [5.0, 80.0, 40.0],  # red (bad trade)
          [3.0, 92.0, 20.0]]  # red (bad trade)

labels = ['green', 'green', 'red', 'red']

new_trades = [[90.0, 10.0, 15.0],
              [10.0, 98.0, 50.0]]
```

Sample Output
['green', 'red']

Techniques
We're just looking for a simple, moderately effective, solution.
Test cases can be passed with fewer than 30 lines of code.

☆ Integral Point Inside a Triangle

Given a triangle on a two-dimensional plane, output the integral point inside or on the boundaries of the triangle which has the minimum sum of distances from the three vertices.

Input: Coordinates (float number up to 3 decimal places) of three vertices are inputted as the x, y coordinate so that three vertices are p1 = (x1, y1), p2 = (x2, y2), p3 = (x3, y3).
Output: A list/array with integer coordinate [x, y] which the point has the minimum sum of distances from the three vertices.
If there are no valid points, output [None, None]
If there are multiple answers, output the one with the largest x-coordinate.
If there are still multiple answers, output the one with the largest y-coordinate.

Sample input:
x1 = 0.0, y1=0.0, x2 = 1.0, y2 = 0.0, x3 = 1.0, y3 = 1.0

Sample output:
[1, 0], point inside or on the boundary of the triangle, where the sum of distances from three vertices is minimum (equal to 2 in the example).

https://www.geeksforgeeks.org/check-whether-a-given-point-lies-inside-a-triangle-or-not/
https://www.geeksforgeeks.org/optimum-location-point-minimize-total-distance/

☆ Lake Escape

Albert is stranded on a frozen lake. He wants to know if he can make it back to shore. He is currently on a snowbank that gives him some traction, but once he steps on the ice, he will slide in the same direction until he hits another snowbank. There are also treacherous holes in the ice that he must avoid.

As a cruel twist of fate, Albert's young pup, Kuna, is also stranded, but on a different snowbank. Can Albert reach his pup AND make it to shore?

Albert can only move horizontally and vertically. He makes it to shore by leaving the lake grid.

Input Format
side_length -- the length of a side of the lake (it's a square)
lake_grid -- a 2D matrix representing the lake
  0 = ice, 1 = snowbank, -1 = hole
albert_row -- row of Albert's snowbank
albert_column -- column of Albert's snowbank
kuna_row -- row of Kuna's snowbank
kuna_column -- column of Kuna's snowbank

Output Format
integer -- 1 if he escapes with pup, 0 if he does not

Sample Input
```
7
0 0 0 0 0 0 0
0 0 -1 0 0 0 0
0 0 1 -1 0 -1 0
-1 0 0 0 0 0 0
0 0 1 0 0 1 0
-1 0 -1 0 -1 0 0
0 0 0 0 0 0 0
4
2
2
2
```

Sample Output
```
1
```

https://www.analyticsindiamag.com/openai-gym-frozen-lake-beginners-guide-reinforcement-learning/

☆ Classify the Trade

Classify new trades based on their similarity to old trades.

Every trade has 3 features: profit, risk, and latency.
You have a list of old trades. Every old trade has been labeled with a color.
You have a list of new trades. New trades are unlabeled.
Your task is to create an algorithm that uses the old trades to predict the labels of new trades.

Input Format
A list of feature vectors, representing the old trades.
A list of labels, corresponding to each of the old trades.
A list of feature vectors, representing the new trades.

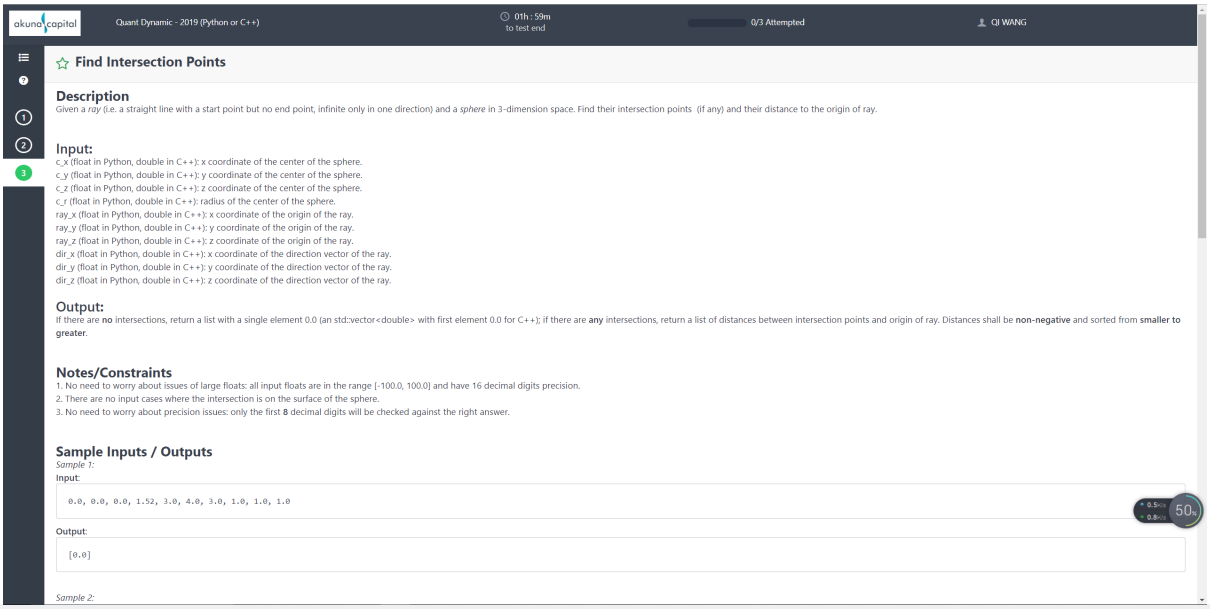
Constraints
Features are floats in the interval [0, 100].
Labels are strings.
Classes may be imbalanced (e.g. 'red' trades may greatly outnumber 'green' trades).
Classes may not be linearly separable in 3 dimensions.
The data set is small:
  The number of old trades is less than 100.
  The number of new trades is less than 10.

```
def
classify(trades,lables,new_trades):
                                        green_number = 0;
                                        red_number = 0;
                                        for i in range(lables.__len__()):
                                             if(lables[i] == "green"):
                                                  green_number = green_number+1;
                                             else:
                                                  red_number = red_number+1;


                                        count = 0;
                                        profit_green =0.0;
                                        risk_green =0.0;
                                        latency_green= 0.0;
                                        profit_red = 0.0
                                        risk_red = 0.0;
                                        latency_red= 0.0;
                                        for j in range(trades.__len__()):
                                             if(count < green_number):
                                                  count = count+1;
                                                  profit_green =
                                   profit_green+trades[j][0];
                                                  risk_green =
                                   risk_green+trades[j][1];
                                                  latency_green =
                                   latency_green+trades[j][2];
                                             else:
                                                  profit_red =profit_red +
                                   trades[j][0];
                                                  risk_red = risk_red+trades[j][1];
                                                  latency_red =
                                   latency_red+trades[j][2];


                                        profit_green = profit_green / green_number;
                                        risk_green = risk_green / green_number;
                                        latency_green = latency_green/ green_number;
                                        profit_red = profit_red/red_number;
                                        risk_red = risk_red/red_number;
                                        latency_red = latency_red/red_number;
                                        result = []
                                        for k in range(new_trades.__len__()):
                                             if(abs(new_trades[k][0] - profit_green)
                                   <(new_trades[k][0]-profit_red)):
                                                  result.append("green");
```

```
            elif(abs(new_trades[k][0] -
profit_green) >(new_trades[k][0]-profit_red)):
                result.append("red");
            else:
                if(abs(new_trades[k][0] -
risk_green) <(new_trades[k][0]-risk_red)):
                    result.append("green");
                elif(abs(new_trades[k][0] -
risk_green) >(new_trades[k][0]-risk_red)):
                    result.append("red");
                else:
                    if(abs(new_trades[k][0] -
latency_green) <(new_trades[k][0]-latency_red)):
                        result.append("green");
                    else:
                        result.append("red");
    return result;
```

Drone delivery
https://github.com/davidlu0517/Python-Code-Challenge/blob/master/Drone%20Delivery.pdf
code;

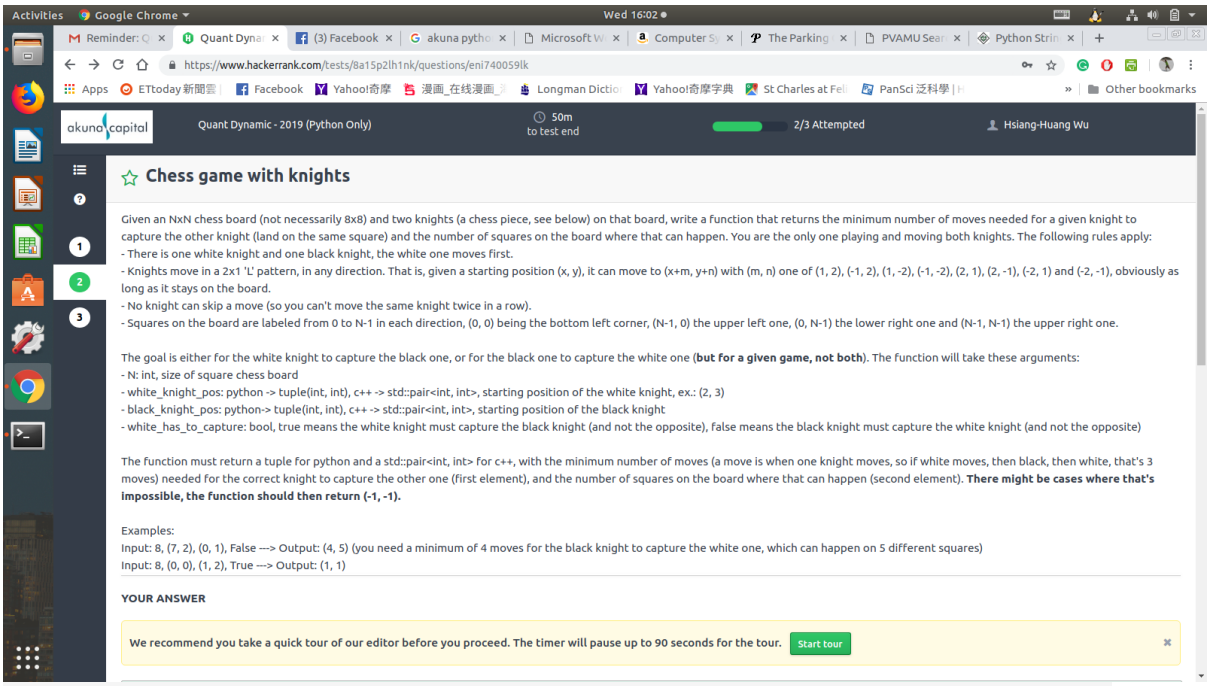
未解决的：




```
function dotProduct(v1, v2) {
```

```
  return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}

function squaredLength(v) {
  return dotProduct(v, v);
}

// Returns whether the ray intersects the sphere
// @param[in] center center point of the sphere (C)
// @param[in] radius radius of the sphere (R)
// @param[in] origin origin point of the ray (O)
// @param[in] direction direction vector of the ray (D)
// @param[out] intersection closest intersection point of the ray with the sphere, if any
function intersectRayWithSphere(center, radius,
                                origin, direction,
                                intersection) {
  // Solve |O + t D - C|^2 = R^2
  //      t^2 |D|^2 + 2 t < D, O - C > + |O - C|^2 - R^2 = 0
  var OC = intersection; // Use the output parameter as temporary workspace

  OC.x = origin.x - center.x;
  OC.y = origin.y - center.y;
  OC.z = origin.z - center.z;

  // Solve the quadratic equation a t^2 + 2 t b + c = 0
  var a = squaredLength(direction);
  var b = dotProduct(direction, OC);
  var c = squaredLength(OC) - radius * radius;
  var delta = b * b - a * c;

  if (delta < 0) // No solution
    return false;

  // One or two solutions, take the closest (positive) intersection
  var sqrtDelta = Math.sqrt(delta);

  // a >= 0
  var tMin = (-b - sqrtDelta) / a;
  var tMax = (-b + sqrtDelta) / a;

  if (tMax < 0) // All intersection points are behind the origin of the ray
    return false;

  // tMax >= 0
  var t = tMin >= 0 ? tMin : tMax;

  intersection.x = origin.x + t * direction.x;
  intersection.y = origin.y + t * direction.y;
  intersection.z = origin.z + t * direction.z;

  return true;
}
function dotProduct(v1, v2) {
  return v1.x * v2.x + v1.y * v2.y + v1.z * v2.z;
}

function squaredLength(v) {
  return dotProduct(v, v);
}
```

```
// Returns whether the ray intersects the sphere
// @param[in] center center point of the sphere (C)
// @param[in] radius radius of the sphere (R)
// @param[in] origin origin point of the ray (O)
// @param[in] direction direction vector of the ray (D)
// @param[out] intersection closest intersection point of the ray with the sphere, if any
function intersectRayWithSphere(center, radius,
                                origin, direction,
                                intersection) {
  // Solve |O + t D - C|^2 = R^2
  //      t^2 |D|^2 + 2 t < D, O - C > + |O - C|^2 - R^2 = 0
  var OC = intersection; // Use the output parameter as temporary workspace

  OC.x = origin.x - center.x;
  OC.y = origin.y - center.y;
  OC.z = origin.z - center.z;

  // Solve the quadratic equation a t^2 + 2 t b + c = 0
  var a = squaredLength(direction);
  var b = dotProduct(direction, OC);
  var c = squaredLength(OC) - radius * radius;
  var delta = b * b - a * c;

  if (delta < 0) // No solution
    return false;

  // One or two solutions, take the closest (positive) intersection
  var sqrtDelta = Math.sqrt(delta);

  // a >= 0
  var tMin = (-b - sqrtDelta) / a;
  var tMax = (-b + sqrtDelta) / a;

  if (tMax < 0) // All intersection points are behind the origin of the ray
    return false;

  // tMax >= 0
  var t = tMin >= 0 ? tMin : tMax;

  intersection.x = origin.x + t * direction.x;
  intersection.y = origin.y + t * direction.y;
  intersection.z = origin.z + t * direction.z;

  return true;
}
```



Ranked Election

Your task is to analyze a ranked election. Here's how the election works: each ballot consists of an ordered list of candidates based on a voter's preference. The ballots are processed in rounds. During each round, the first choice on each ballot is tallied up. If a candidate has the most votes AND surpasses a certain percentage of the votes, they win the election. If no candidate reaches the necessary percentage, then the candidate with the fewest votes is eliminated from all ballots. Once again the first choice on each ballot is tallied (remember, this is AFTER removing the last place finisher of the previous round). This process continues until a candidate reaches the necessary percentage. Some extreme cases to think about:

- If multiple candidates receive the same number of votes AND reach the necessary percentage in the SAME ROUND, the election is thrown out and nobody wins.
- If multiple candidates tie for last place, they are ALL removed from the ballots for the next round. This means you can be left with no candidates, in which case, the election is thrown out and nobody wins.
- Voters are not required to order all candidates on their ballot (i.e. even if there are 10 candidates, a ballot can consist of 3 ordered candidates)

To analyze the election, you must determine which candidate will win for a certain percentage given all the ballots. The expected output is a list of percentage intervals and the winning candidate for each interval. We are only concerned with whole number percentage points (i.e. 10.5% - 20% is not a valid interval).

Input Format:
num_candidates -> int
num_ballots -> int
ballots -> 2D int array with num_ballots rows and num_candidates columns *
* Candidates are labeled from 0 to num_candidates - 1. If a voter orders fewer than num_candidates, the blanks will be filled with -1.

Output Format:
2D int array of the form [[pct1, pct2, candidate], [pct3, pct4, candidate]...] *
* The intervals are inclusive on both sides. You must cover the entire percentage range from 0 to 100. If an interval exists where no candidate can win, put -1 in the candidate slot. You should minimize the number of intervals where possible.

Example:
num_candidates = 5
num_ballots = 5
ballots = [[1, 4, 3, -1, -1], [2, 1, -1, -1, -1], [1, 3, 2, 0, -1], [3, 2, 4, 0, 1], [2, 3, 4, 0, -1]]
Expected output: [[0, 40, -1], [41, 100, 2]]
For any whole number percent below 41, no candidate can win. For any whole number percent above 40, only candidate 2 can win.

Word Graph

There are two inputs to this question:

1) A list of words
2) A list of tuples that represent directed edges in a graph.

Imagine that there exists a graph consisting of 26 nodes wherein each node corresponds to one letter of the English language. You are given a list of words as well as a list of lists. Each of the sublists in the latter input represents a directed edge between two letters i.e. ('a', 'b') represents an edge going from a to b. Your task is to determine whether or not each word given in the input can be spelled out by tracing the edges in this graph, starting at any node. The output should be a list of 1's and 0's (1 = True, 0 = False), in order of the input words.

For example, let's say you get ['a', 'b', 'ab', 'ba'] and [('a', b')]. The output should be [1, 1, 1, 0].

YOUR ANSWER

We recommend you take a quick tour of our editor before you proceed. The timer will pause up to 90 seconds for the tour. Start tour

Draft saved 07:37 pm

Original code | Python 3

```
#!/bin/python3

import sys
import os


# Complete the function below.
```

☆ Sum of unique fibonacci numbers

Given positive integers x and n, determine if x can be expressed as a sum of n unique fibonacci numbers. Fibonacci numbers are from a series 1, 1, 2, 3, 5, 8, 13, 21, ... in which starting from the third number, each number is the sum of the previous two numbers.

Input Format	Output Format
Arguments: x: int, target sum n: the number of integers Constraints: x <= 10^9 n <= 5	A boolean value indicating whether this can be achieved.
Sample Input x = 6, n = 2 x = 5, n = 3 x = 10059560, n = 4	**Sample Output** True False True

https://www.geeksforgeeks.org/python-program-for-zeckendorfs-theorem-non-neighbouring-fibonacci-representation/